DESARROLLO BASADO EN MDA DE UN SISTEMA DE GESTIÓN DOUMENTAL BAJO LA NORMA ISO 9001:2000 PARA UNA EMPRESA DE PROYECTOS DE INGENIERIA

CAROLINA MASSO SOLARTE

RODRIGO PEÑA RICARDO

UNIVERSIDAD DEL CAUCA

FACULTAD DE INGENIERIA ELECTRONICA Y TELECOMUNICACIONES

Departamento de Sistemas

Grupo de Investigación GTI Grupo I+D en

Tecnologias de la Informacion

POPAYAN

2004

DESARROLLO BASADO EN MDA DE UN SISTEMA DE GESTIÓN DOUMENTAL BAJO LA NORMA ISO 9001:2000 PARA UNA EMPRESA DE PROYECTOS DE INGENIERIA

Monografia presentada como requisito parcial para optar por el titulo de Ingenieros en Electrónica y Telecomunicaciones

CAROLINA MASSO SOLARTE
RODRIGO PEÑA RICARDO

Director
Ing. Esp. Julio Ariel Hurtado

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERIA ELECTRONICA Y TELECOMUNICACIONES
Departamento de Sistemas
Grupo de Investigación GTI Grupo I+D en
Tecnologias de la Informacion
POPAYAN
2004

TABLA DE CONTENIDO

INTRODUCCION

Las empresas desarrolladoras de software necesitan crear soluciones con muy buena calidad, a bajos costos y en el menor tiempo posible. Estas empresas utilizan procesos de desarrollo que manejan los flujos de trabajo y permiten organizar las tareas de los integrantes de la empresa. Sin embargo, estos procesos no permiten obtener una reutilización adecuada de los modelos de los sistemas, disminuyendo la eficiencia en la producción de sus aplicaciones.

En la búsqueda de soluciones a estos inconvenientes se encontró a MDA, Model Driven Architecture, un framework realizado por el Object Management Group [OMG], que se encuentra en una etapa no muy madura de desarrollo, pero que a medida que se investiga y se realizan estándares para su utilización, empieza a tomar importancia en el ámbito del desarrollo software ya que empresas que han adoptado este enfoque han mejorado en la productividad y flexibilidad de sus aplicaciones y lo más importante han reducido el tiempo y los costos en la realización de sus proyectos. Como no existen muchas experiencias en el momento con este enfoque, se decidió trabajar con el proceso de desarrollo convencional UP y hacer que MDA se integre con este proceso en todos sus flujos de desarrollo, para obtener toda la potencialidad que este nuevo enfoque brinda al desarrollo software. Para tener una experiencia real con este nuevo enfoque, se realizó el Sistema de Gestión Documental para la empresa colombiana GAMMA INGENIEROS S.A. Debido a que es una aplicación para un caso real, se necesitaba cumplir con las exigencias que una empresas demanda, es por esta razón que [J2EE] Java Enterprise Edition se tomó como plataforma de desarrollo, ya que permite flexibilidad, escalabilidad y transaccionalidad características esenciales para cualquier aplicación empresarial.

El desarrollo software es muchas veces comparado con el desarrollo hardware en términos de madurez. Mientras que el desarrollo hardware es mucho más apreciable, el desarrollo software de aplicaciones ha evolucionado de una manera no tan evidente pero muy significativa. Esto se puede apreciar en el desarrollo de los sistemas actuales que poseen una alta complejidad. Además, el desarrollo software todavía tiene muchos problemas por vencer, por ejemplo, cada vez que surge una nueva tecnología, se deben adaptar los sistemas existentes a ésta, lo cual requiere de un arduo trabajo. Otro ejemplo son los continuos cambios de requerimientos que también afectan el desarrollo de los sistemas, haciendo que los desarrolladores vuelvan a realizar los

modelos desde un principio o cambien la consistencia de estos, llevando a una sustancial pérdida de tiempo. Otro aspecto al que se enfrenta el software en la actualidad es la falta de robustez arquitectónica, ya que se invierte mucho más tiempo en otros procesos como implementación, que en el análisis de los requerimientos del sistema.

PROBLEMAS EN EL DESARROLLO SOFTWARE

A continuación se analizan los problemas más comunes del software según [MDAEX].

La falta de productividad

Los procesos de desarrollo orientados hacia la programación abarcan generalmente la captura de requerimientos, análisis, diseño, codificación, pruebas e implantación, dando como resultado una serie de documentos y diagramas que a la hora de codificar, pierden rápidamente su valor, y peor aun, cuando el sistema va evolucionando, estos diagramas se van alejando cada vez más de la realidad o la solución del problema y muchas veces el tiempo es apremiante si se desean realizar de nuevo cambios en los documentos. En la figura 1.1 se observa los procedimientos del proceso de desarrollo tradicional de un sistema con su respectiva documentación.

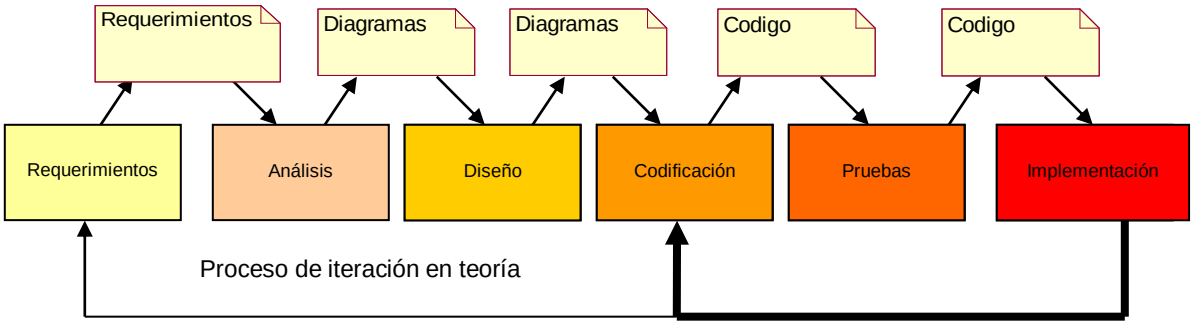


Figura 1.1 Ciclo de Vida del desarrollo software

Se han desarrollado muchas soluciones para tratar de resolver todos estos inconvenientes, como por ejemplo, trabajar con Programación Extrema[1] la cual ahorra ciertos pasos del proceso y da mayor énfasis a la codificación y a las pruebas. Inicialmente esta solución es efectiva pero

[1] La Programación Extrema define una manera de reunir a clientes y programadores en un equipo firmemente integrado con condiciones de trabajo que promueven la comunicación y solución de un problema. Kent Beck creó esta metodología de desarrollo en 1996.

presenta problemas en el momento de realizar mantenimiento. Esto nos lleva a la siguiente pregunta: ¿Es preciso realizar todos los pasos de un proceso de desarrollo o se pueden automatizar aquellos pasos innecesarios los cuales nos conllevan a la perdida de tiempo y peor aun a la falta de productividad, o existe una solución que permita resolver estos dos aspectos?

La Falta de portabilidad

Con el avance tecnológico que hoy en día el mundo ofrece, plataformas como J2EE, .NET, CORBA entre otras, se han vuelto mas y más necesarias para aquellas compañías que desean estar a la par con las demandas de los clientes los proveedores y porque no, sus propias necesidades. La solución entonces radica en desarrollar sistemas que puedan migrar a nuevas versiones, adaptarse fácilmente a otras tecnologías y que puedan ínter-operar con los nuevos sistemas.

La Falta de Interoperabilidad

En el momento de desarrollar software, es necesario tener en mente que los sistemas necesitan comunicarse unos con otros. Esto es una meta difícil de conseguir debido al alto número de tecnologías en el mercado que prometen estandarizar las comunicaciones entre sistemas pero que en muchas situaciones no lo cumplen, e incluso presentan versiones incompatibles dentro de ellas. Una posible solución a este problema es el desarrollo de software basado en componentes como lo ofrece J2EE (junto con tecnologías como XML o IIOP) que permite la interacción entre estos y una posible expansión compatible con versiones futuras.

Falta de Mantenimiento y la Documentación

A los desarrolladores no les gusta documentar y si lo hacen es por seguir ciertos requerimientos impuestos, pero no lo hacen por su propia voluntad. Lo que si es seguro es que saben a ciencia cierta que DOCUMENTAR es una de las principales tareas del desarrollador, ya que los sistemas que ellos realizan deben ser mantenidos o cambiados con el tiempo. Existen soluciones a bajo nivel, herramientas que pasan el código a documentación, pero la documentación de alto nivel solo se puede realizar a mano hasta el momento.

Realizar sistemas en este ambiente donde la tecnología sufre cambios continuamente no es tarea fácil ya que los requerimientos de los sistemas deben ser innovados continuamente. Además los desarrolladores de software no desean estar atados a los sistemas operativos, a los programas de lenguaje, a un set de instrucciones arquitecturales o a cualquier otra plataforma existente. Por esta razón se quiere explotar y reutilizar elementos que agilicen el proceso de desarrollo. MDA, Arquitectura Manejada por Modelos, permite tener una visión más amplia y ayuda a solucionar la mayoría de los problemas anteriores, ya que este nuevo enfoque permite a los desarrolladores de

software poder enfocarse mucho más en los aspectos de negocio de su empresa y en la satisfacción de las necesidades del cliente, sin preocuparse por el tipo de tecnología en la cual sus aplicaciones están montadas. Con los tres modelos fundamentales CIM(Modelo Independiente de la computación), PIM(Modelo Independiente de la Plataforma) y PSM(Modelo Independiente de la Plataforma) de MDA, se logra este objetivo al separar la lógica de la aplicación de su implementación.

Con este trabajo de grado, se ha querido dejar la documentación necesaria para futuras investigaciones que necesiten de los conceptos que se presentan aquí y deseen perfeccionar este proceso ya sea mejorando esta propuesta o creando herramientas que sistematicen parte del proceso que aquí se realizó.

CAPITULO1. LA ARQUITECTURA SOFTWARE Y MDA-Arquitectura Dirigida por Modelos.

Diariamente los ingenieros de software se enfrentan al desarrollo de aplicaciones software cada vez más complejas, que demandan en alto grado, productividad y calidad. Por esta razón se necesita inculcar conceptos arquitecturales en toda la organización así como involucrar a todas las personas en el desarrollo del sistema, para trabajar en conjunto hacia un objetivo común, y más importante, hacer de esto un permanente proceso de optimización. En otras palabras establecer la arquitectura como una cultura para el desarrollo de soluciones.

¿Por qué es tan importante la arquitectura? A partir de los requerimientos que el cliente entrega del sistema, cada persona en la organización crea diferentes conceptos y metas que podrían ser contradictorias y contraproducentes para el proceso de desarrollo. Es aquí donde la arquitectura se vuelve indispensable.

1.1 LA ARQUITECTURA SOFTWARE

No podemos negar que la arquitectura siempre será influenciada por elementos de la organización como son los desarrolladores, el ambiente tecnológico, la experiencia del arquitecto y en fin por todas y cada una de las personas que se involucran con el sistema incluyendo al cliente y sus requerimientos. Sin embargo la arquitectura también afecta de manera inherente a todos estos elementos y es una pieza fundamental del proceso de desarrollo. Si se observa la figura 1.2 se puede ver que la arquitectura es el resultado de un conjunto de decisiones de negocio y decisiones técnicas que cambian dependiendo de los elementos y del ambiente en que ésta se encuentre [ARQPRA].

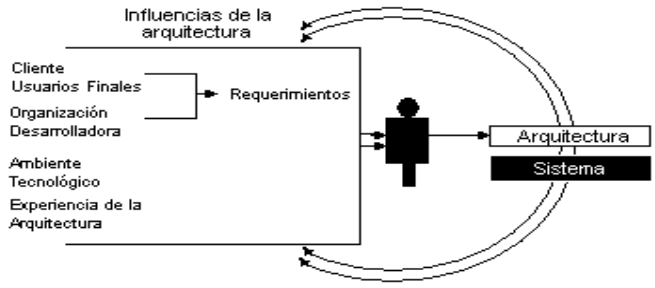


Figura 1.2 de [ARQPRA] Influencias de la arquitectura sobre la organización

1.1.1 Definición de Arquitectura Software

Existen varios puntos de vista de acuerdo a [ARQPRA] a la hora de definir la arquitectura. Estos son algunos de ellos

- La arquitectura es un diseño de alto nivel: Es cierto pero no es una definición suficiente pues el diseño y la arquitectura van de la mano, pero no son intercambiables a la hora de definirlos. Existen tareas asociadas con el diseño que no son arquitecturales como por ejemplo la estructura de datos de la aplicación.
- La arquitectura es toda la estructura del sistema: Esta definición implica que el sistema sólo tendrá una sola estructura lo cual no es cierto ya que diferentes estructuras conforman el sistema.
- La arquitectura es la estructura de componentes de un programa o un sistema, sus Inter-relaciones, sus principios y guías que gobiernan su diseño y su evolución todo el tiempo: Cada sistema tiene una arquitectura independientemente del proceso con la cual esta arquitectura fue diseñada, así que no se hace necesario la información de las guías y los principios.
- La arquitectura son componentes y conectores entre estos componentes: Conectores implican un mecanismo de transferencia y control de datos en tiempo de ejecución, es decir esta definición se centra en las estructuras arquitecturales en tiempo de ejecución. Sería mejor tratar esta definición con el concepto de relaciones entre elementos ya que se podría hablar de relaciones dinámicas (tiempo de ejecución) y estáticas (no-ejecución).

Siguiendo algunas de definiciones se realiza el siguiente esquema:

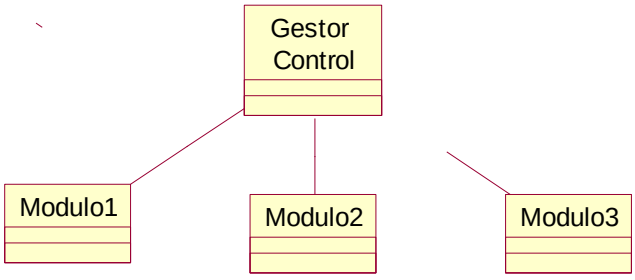


Figura 1.3 Típica representación de la arquitectura, con poca información.

Podríamos deducir lo siguiente:

- El sistema tiene cuatro elementos

- Hay tres elementos Modelo 1, Modelo 2, Modelo 3, que deben tener algo más en común con el elemento control proceso, además están ubicados en un mismo nivel
- Cada elemento tiene alguna clase de relación con el resto.

Asumiendo la mayoría de las definiciones tradicionales anteriormente citadas, en donde la arquitectura es un conjunto de componentes y conexiones entre estos, el esquema de la figura 1.3 satisface estas definiciones. Pero la pregunta es: ¿Se puede considerar a esto como una arquitectura? La respuesta es no, ya que este diagrama no muestra la naturaleza de los elementos, las responsabilidades de los elementos, lo que realizan, sus funciones, ¿Corren al mismo tiempo o en procesos distintos?, ¿Cuál es el significado de las conexiones, control, datos?, no se muestran los mecanismos de comunicación, las capas o niveles de estructura en que se encuentran ¿Cuál es el nivel que controla?, y en fin una serie de características que necesitan ser vistas por todos los miembros de la organización para tener claro que es lo que el sistema va a realizar.

La definición más acertada que se encontró de la arquitectura se encuentra en [ARQPRA] :

> "*La arquitectura software de una aplicación o un sistema de computo es la estructura de estructuras del sistema la cual compromete elementos software (1), propiedades visibles externamente(2) de estos elementos y las relaciones(3) entre ello.*"

(1)Elementos software, los cuales contienen atributos públicos y privados. La arquitectura enfatiza en la relaciones (3), lo cual implica que omite cierta información de estos elementos. La arquitectura se preocupa por los atributos públicos que pueden ser accedidos por medio de interfaces. Los otros atributos (privados) no son considerados arquitecturales.

(2)Propiedades visibles externamente, son las que le permiten a un elemento hacer uso de otro. Estas son percibidas por otros elementos tales como servicios, características de desempeño, manejo de fallos, uso de recursos compartidos, entre otras.

Además de lo anterior esta definición implica dos cosas. La primera, cada sistema o aplicación debe tener una arquitectura y la segunda que el comportamiento de cada elemento hace parte de la arquitectura, y además ese comportamiento puede ser observado por otro elemento dentro de esta arquitectura.

1.1.2 Patrones de Arquitectura o Estilos Arquitecturales

Son una descripción de elementos y sus diferentes tipos de relaciones con una serie de restricciones y de pautas para su utilización [ARQPRA]. Elementos como por ejemplo clientes, servidores, filtros, capas, bases de datos y relaciones o conexiones como por ejemplo procedimientos de llamada, eventos de broadcast, protocolos entre bases de datos o pipes. Un patrón también puede ser pensado como un conjunto de restricciones que se aplican sobre una arquitectura (sobre sus tipos de elementos y sus patrones de iteración). Estas restricciones definen una familia de arquitecturas.

Un ejemplo de estilo arquitectural, es la estructura Cliente-Servidor. El cliente y el servidor son los elementos y su coordinación queda descrita en términos del protocolo de comunicación entre estos. Los patrones arquitecturales no son la arquitectura. La arquitectura de un sistema puede estar enmarcada bajo los lineamientos y restricciones de estos patrones arquitecturales para lograr la obtención de ciertos atributos de calidad como desempeño, seguridad, portabilidad o los que el desarrollador crea conveniente adoptar, así, el patrón arquitectural se escoge según los requisitos que se deseen cumplir. Se recomienda escoger un solo patrón arquitectural que solucionan no todos, pero la mayoría de los requerimientos del sistema y más adelante, se escogen uno o más patrones de diseño, que solucione problemas más específicos. Más adelante se retoma el tema de los patrones de diseño.

1.2 LA ARQUITECTURA EN EL PROCESO DE DESARROLLO

Para que una organización pueda llevar a cabo el desarrollo software de sus aplicaciones, se necesita de un buen proceso que gestione las actividades de desarrollo software. Estas actividades están involucradas en la realización de una arquitectura software para la creación del diseño y luego para la creación de la aplicación. Las actividades que expone [ARQPRA] ayudan a organizar la toma de decisiones que los diferentes grupos de trabajo deben hacer. Estas actividades son:

- ✓ **Creación del Caso del Negocio del Sistema**: La creación del modelo de negocio de un sistema va más allá de la consecución de las necesidades de mercado del sistema. Este es un paso muy importante para la creación y delimitación de cualquier requerimiento futuro. Cual debe ser el costo del producto? Cual es el mercado objetivo? Que tanto tiempo se requiere para salir al mercado? Existen limitaciones en las cuales se puede

trabajar? Estas son las principales preguntas que se deben formular los arquitectos del sistema en esta primera actividad para llegar a adquirir las características del negocio.

- ✓ **Entendimiento de los requerimientos**: Los requerimientos se pueden obtener con técnicas como casos de uso o con modelos de maquinas de estado finito o lenguajes formales de especificación. Una de las técnicas más usadas es la recolección de características similares de sistemas ya implementados, aplicando patrones arquitecturales ya establecidos para ahorrar tiempo en el desarrollo lo que genera una alta productividad a bajos costos al proyecto. No importa la técnica de recolección de requerimientos, lo que importa es que los requerimientos del sistema determinarán la forma de su arquitectura.

- ✓ **Creación o selección de la Arquitectura**: No existe una metodología universal para desarrollar sistemas o escoger la arquitectura de estos, lo único que se debe tener en cuenta es que el método cumpla con los requerimientos del sistema de lo contrario el método no servirá. Uno de los métodos que propone [ARQPRA] es el de recolectar atributos de calidad que el sistema debe cumplir, para decidir qué arquitectura o que estilos arquitecturales ayudan a cumplir de manera eficiente las necesidades del sistema.

- ✓ **Comunicación de la Arquitectura:** En esta actividad la arquitectura es comunicada a todos los miembros de la empresa de una manera clara y sin ambigüedades, para que esta pueda ser utilizada eficientemente. De la misma forma la documentación de la arquitectura debe ser entendible y lo suficientemente abstracta para ser rápidamente comprendida por los nuevos trabajadores de la empresa, y lo suficientemente detallada para que pueda servir para posteriores análisis de sistemas.

- ✓ **Análisis y Evaluación e la Arquitectura**: En cualquier proceso de desarrollo habrán múltiples opciones arquitecturales a ser consideradas, y es aquí donde el arquitecto juega un rol importante en el proceso de desarrollo. Es importante que la arquitectura que se seleccione sea la mejor opción, para que los desarrolladores y los grupos de trabajo cumplan con sus necesidades y al mismo tiempo el sistema cumpla con las necesidades de calidad.

- ✓ **Implementación del sistema basado en la Arquitectura:** Esta actividad es para que los desarrolladores mantengan firme las restricciones de la estructura y protocolos de la

arquitectura. Para esto debe haber un ambiente de comunicación y asistencia permanente con los desarrolladores para mantenerla.

- ✓ **Aseguramiento de la Implementación conforme a la Arquitectura:** Cuando ya se ha adoptado una arquitectura y ya ha sido creada, se debe hacer una fase de mantenimiento para sostener la trazabilidad con las fases de desarrollo anteriores.

1.3 ATRIBUTOS DE CALIDAD

La consecución de los atributos de calidad del sistema se hace desde la etapa de especificación en donde se determinan las cualidades del negocio. Estas cualidades se encuentran sobre las funcionalidades que el sistema debe tener, pero algunas veces se les da muy poca importancia. Muchos de los sistemas son rediseñados una y otra vez, no porque las funcionalidades del sistema estén fallando si no porque estos sistemas carecen de portabilidad, escalabilidad o se ven afectados por virus o hackers en la red. Por esta razón es muy importante tener en cuenta estos aspectos no sólo en la parte de análisis y diseño donde se elige la arquitectura (aspectos arquitecturales) sino también en la parte de implementación e implantación (aspectos no arquitecturales) [ARQPRA], por ejemplo:

- **Usabilidad**
  a. **Aspectos NO Arquitecturales:** Hacer una interfaz lo suficientemente clara para el usuario (botones, capas, etc). Esto se puede obtener en la actividad de diseño de la interfaz de usuario y no afecta sino los módulos de presentación.
  b. **Aspectos Arquitecturales:** Posibilidad de cancelar operaciones, rehacerlas, o reutilizar flujos de datos
- **Modificabilidad**
  a. **Aspectos NO Arquitecturales:** Técnicas de codificación dentro de los módulos.
  b. **Aspectos Arquitecturales:** Cómo son desacoplados los módulos del sistema.
- **Desempeño**
  a. **Aspectos NO Arquitecturales:** Escoger los algoritmos para la implementación, y su codificación
  b. **Aspectos Arquitecturales:** Número de comunicaciones entre componentes, sobre qué componentes han sido localizados ciertos servicios.

La arquitectura es un aspecto muy importante para la consecución de estos atributos de calidad, por esto es necesario decidir cuales de estos aspectos se necesitan para el sistema en el momento de escoger la estructura de este. Algunos de los atributos de calidad más importantes son:

Disponibilidad: La disponibilidad es un atributo de calidad de un sistema y está asociada a las fallas que pueda tener éste y a las consecuencias de las mismas. La falla es observable por el usuario y ocurre cuando el sistema deja de entregar el servicio según su especificación. La disponibilidad de un sistema es la probabilidad de que éste funcione cuando se necesite. La disponibilidad así:

$$\alpha \equiv \frac{\text{Tiempo antes de la falla}}{\text{Tiempo antes de falla + Tiempo antes de reparación}}$$

Se espera que el sistema funciones con un 99.9% de probabilidad o que no trabaje con un 0.1% de probabilidad.

Modificabilidad: Está relacionada con que tanto puede un sistema ser modificado y que tan alto es su costo. Aquí se deben tener en cuenta dos aspectos: Lo que se va a cambiar, por ejemplo, la plataforma, el sistema operativo, los protocolos, las cualidades, etc y quién y cuando se cambia, lo hace el desarrollador (cambiando el código fuente), el usuario (una interfaz), el cambio se hace en ejecución, en compilación? Una vez el cambio se realice, este podría ser diseñado, probado he implementado de nuevo, lo que implica costo y tiempo.

Existen otros atributos de calidad importantes como son extensibilidad, escalabilidad o portabilidad pero en este caso se podrían incluir en el atributo de modificabilidad, por ejemplo modificar la capacidad (Escalabilidad), modificar la plataforma (Portabilidad), o adicionar módulos (Extensibilidad).

Desempeño: Tiene que ver con las palabras tiempo y recursos. Eventos, alarmas mensajes, ocurren y el sistema debe responder a estos y asignar recursos. El desempeño mide el tiempo que gasta el sistema en responder a estos eventos y qué recursos pueden ser asignados para ello.

Seguridad: Es la medida de la habilidad del sistema para resistir al uso desautorizado de éste, mientras se estaá prestando el servicio o se está legitimando al usuario.

Factibilidad de Pruebas: Que tan fácil puede ser demostrar las fallas del sistema por medio de pruebas. Casi el 40% del costo de desarrollo de un sistema bien implementado se va en pruebas.

Usabilidad: Esto tiene que ver con la facilidad que tiene el usuario para completar exitosamente una tarea y el tipo de soporte que el sistema provee para esto.

La decisión de tomar cualquiera de estos atributos no debe hacerse de forma aislada, puesto que la elección de uno de ellos puede causar efectos tanto positivos como negativos sobre los otros,

por ejemplo la portabilidad y el desempeño. Para que un sistema sea portable se deben aislar las dependencias del sistema lo que produce una sobrecarga en los procesos y procedimientos de las interfaces afectando de esta manera el desempeño del sistema.

En resumen es muy importante tener en cuenta estos atributos de calidad ya que de estos depende el buen funcionamiento y la aceptación de los usuarios del sistema. Como vemos no es sólo en el momento del diseño que se consideran estos aspectos sino en el momento en el que el desarrollador empieza a plantear la arquitectura del sistema.

1.3.1 Escenarios de los Atributos de Calidad

Estos escenarios son una herramienta para el diseñador (Ver figura 1.4) en el momento de establecer los atributos de calidad del sistema [ARQPRA]. Estos consisten de seis partes que son:
Fuente de Estimulo: Este es cualquier actuador, por ejemplo, un humano, un computador, otro sistema, que genera el estimulo.
Estímulo: Es la condición que necesita ser considerada en el momento que llegue este estimulo al sistema.
Artefacto: Lo que es estimulado. Puede ser el sistema o parte de este.
Ambiente: Son las condiciones en que se crea el estimulo. El estimulo se da por ejemplo, cuando el sistema esta corriendo, o esta sobrecargado etc.
Respuesta: Es lo que se produce después que llega el estimulo.
Medida de Respuesta: Cuando ocurre la respuesta esta debe ser medida en cualquier aspecto, para la realización de pruebas.

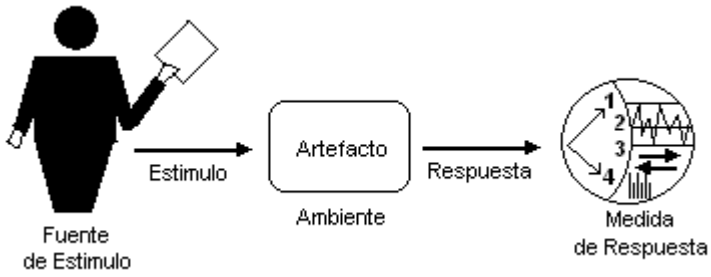


Figura 1.4 Escenarios de Atributos de Calidad

Existen escenarios Generales que son independientes de los sistemas y que permiten definir los atributos a cualquier sistema, y escenarios concretos, que son específicos de un sistema. Tomando los escenarios generales se pueden definir los concretos. En este trabajo de grado

utilizaremos los escenarios generales que se encuentran en [ARQPR] y se dará un ejemplo más adelante en el desarrollo del sistema piloto.

1.4 INTRODUCCIÓN A LA ARQUITECTURA MANEJADA POR MODELOS MDA

Cuando se desea desarrollar un sistema software es importante llevar a cabo un análisis y un modelado del sistema antes de desarrollar código ya que estos modelos ayudan a dirigir, mantener e integrar el desarrollo. MDA es un marco de trabajo para el desarrollo software, implementado y definido por el OMG , que esta basado en una serie de modelos que guían el desarrollo del sistema. MDA surge a partir de la necesidad de establecer la idea de SEPARAR las especificaciones de operación de un sistema, de la plataforma en que ese sistema podría ser implementado.

1.4.1 Conceptos Básicos de MDA

MDA es un nuevo enfoque que aporta grandes beneficios al desarrollo actual de aplicaciones, sin embargo introduce nuevos conceptos que es necesario conocer para entenderlo. A continuación se mostrarán las definiciones básicas de MDA[2] en términos de un sistema existente.

1.4.1.1 Sistema: Este sistema puede incluir: un programa, un sistema de computadores, una combinación o parte de diferentes sistemas controlados separadamente, gente, una empresa, un conjunto de empresas entre otros. MDA se enfoca en el software existente dentro de este sistema.

1.4.1.2 Modelo: El modelo de un sistema es una descripción o especificación de este y su entorno para cierto propósito. Un modelo es con frecuencia presentado como la combinación de dibujos y texto. El texto puede estar en lenguaje de modelado o en un lenguaje natural.

1.4.1.3 Dirigido por Modelos: MDA es un enfoque para el desarrollo de un sistema, el cual incrementa el potencial de trabajo de los modelos. Se llama dirigido por modelos porque usa modelos para dirigir el curso de la conceptualización, diseño, construcción, desarrollo, operación, mantenimiento y modificación de un desarrollo software.

1.4.1.4 Arquitectura: La arquitectura de un sistema es una especificación de las partes y de los conectores de un sistema así como las reglas para la interacción de estas partes utilizando los conectores ya definidos.

1.4.1.5 Punto de Vista: Un punto de vista sobre un sistema es una técnica de abstracción utilizando un selecto conjunto de conceptos arquitecturales y reglas de estructuración con el fin de

[2] Este Trabajo se basa en MDAGuide Versión 1.0.1 el cual contiene la especificación de MDA www.omg.org

enfocar un punto de interés particular dentro de un sistema. Abstracción, definida como el proceso de supresión de ciertos detalles seleccionados para establecer un modelo simplificado.

MDA especifica tres Puntos de Vista en un sistema: punto de vista independiente de la computación, punto de vista independiente de la plataforma, punto de vista específico a una plataforma.

1.4.1.6 Vista: La vista de un sistema es la abstracción de éste desde una perspectiva específica.

1.4.1.7 Plataforma: Una plataforma es un conjunto de subsistemas y tecnologías que provee un conjunto de servicios coherentes a través de interfaces y patrones específicos con los cuales cualquier aplicación soportada por esa plataforma puede usar, sin importar los detalles de cómo es implementada, la funcionalidad provista por la plataforma.

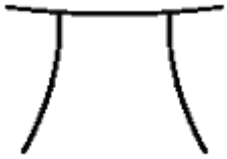


Figura 1.5 Representación de una Plataforma en la Especificación MDA

Hay diferentes tipos de plataformas. Las plataformas genéricas que soportan ciertos estilos arquitecturales, peticiones de servicio, flujos de datos, etc. Las plataformas especificas de la tecnología como por ejemplo [CORBA], [CORBAComponent], [J2EE], y las plataformas especificas del vendedor como son para CORBA:Iona Orbix, Borland VisiBroker , para J2EE: BEAWenLogic Server, IBM WebSphere Software Plataform, otras como MICROSOFT .NET[5]

1.4.1.8 Aplicación: Sistema software que está siendo desarrollado. Un sistema es descrito en términos de una o varias aplicaciones, soportadas por una o más plataformas.

1.4.1.9 Independencia de Plataforma: Este término hace referencia a una cualidad de un modelo. Si el modelo posee esta característica, quiere decir que es independiente de las características de un tipo de plataforma en particular.

1.4.1.10 Puntos de vista de MDA

I.Punto de Vista Independiente de la Computación

Se enfoca en el ambiente y los requerimientos para el sistema. Los detalles de la estructura o el procesamiento no se toman en este punto de vista.

II. Punto de Vista Independiente de la Plataforma

Se enfoca en la operación del sistema ocultando los detalles de la plataforma en donde se encuentra. Esta vista muestra la parte de la especificación que no cambia de una plataforma a otra.

III. Punto de vista Especifico de la Plataforma

Combina el punto de vista independiente de la plataforma con la forma de uso especifica de una plataforma por parte de un sistema.

1.4.1.11 Modelo Independiente de la Computación (CIM): Es la vista de un sistema desde el punto de vista independiente de la computación

1.4.1.12 Modelo Independiente de la Plataforma (PIM): Es la vista de un sistema desde el punto de vista independiente de la Plataforma

1.4.1.13 Modelo Específico de la Plataforma (PSM): Es la vista de un sistema desde el punto de vista específico de la plataforma

1.4.1.14 Modelo de Transformación: Es el proceso de convertir un modelo en otro modelo del mismo sistema. En la figura 1.6 el modelo PIM junto a una serie de información son combinados para producir el PSM.

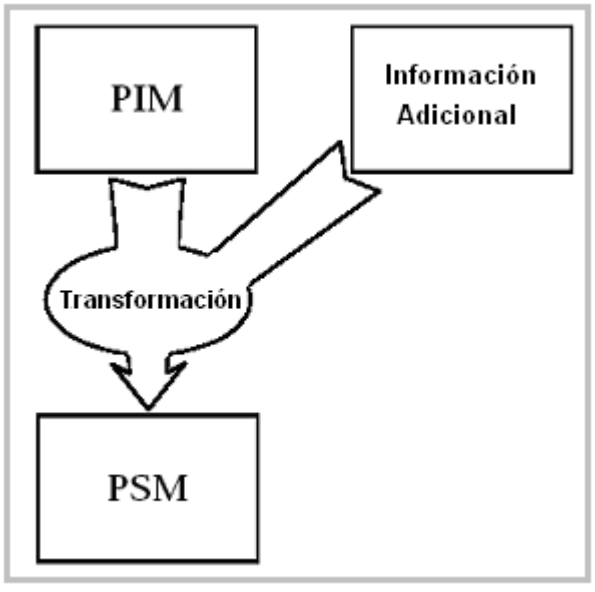


Figura 1.6 Modelo de transformación

1.4.1.15 Mapeo: Un mapeo MDA contiene las especificaciones para transformar un CIM a un PIM, un PIM a un PSM y un PSM a código. Existen diferentes técnicas de mapeo, la más utilizada consiste en identificar elementos en el PIM, por medio de marcas, los cuales deben ser transformados de una manera particular en el PSM.

1.4.1.16 Marcas: Representa un concepto en el PSM y es aplicado a un elemento en el PIM para indicar como se debe transformar en el PSM. Las marcas son dependientes de la plataforma y por esto no pertenecen al PIM. Se genera entonces un PIM marcado, que es el que realmente se prepara para ser convertido en PSM. Estas marcas se pueden generar de diferentes fuentes:

- Tipos desde un modelo, como clases o asociaciones
- Roles provenientes de patrones
- Estereotipos de perfiles UML
- Elementos de metamodelos
- Especificaciones de requerimientos de calidad y servicio

En el caso de los requerimientos de calidad o servicio la marca representa sólo un requisito en el modelo objetivo. Más adelante se retoma este concepto con más detalle.

1.4.1.17 Plantillas: Son modelos parametrizados, usados en el mapeo, que especifican clases particulares de transformaciones. Estas son como los patrones de diseño, pero con especificaciones más estrictas para guiar la transformación.

1.4.1.18 Registro de Transformación: Son los resultados de la transformación de un PIM usando una técnica específica. Este incluye un mapa del elemento desde el PIM al PSM.

MDA tiene como propósito principal brindar un acercamiento abierto e independiente de cualquier proveedor, para esto MDA se basa en estándares establecidos por la OMG como lo son UML, MOF, XMI, y CWM. A continuación se presenta una pequeña descripción de cada uno de estos estándares y las ventajas que brindan a MDA:

UML: Unified Model Lenguage

Un lenguaje grafico de especificación para visualizar, especificar, construir y documentar las características de objetos de sistemas distribuidos. UML 2.0 incorpora semántica de acción que adiciona a UML la sintaxis y la semántica necesaria para ejecutar acciones y procedimientos incluyendo semántica en tiempo de ejecución. UML es un lenguaje notacional para especificar y visualizar aplicaciones generalmente basadas en orientación a objetos. UML esta basado en métodos previos de notación como [Booch], [OMT] y [OOSE].

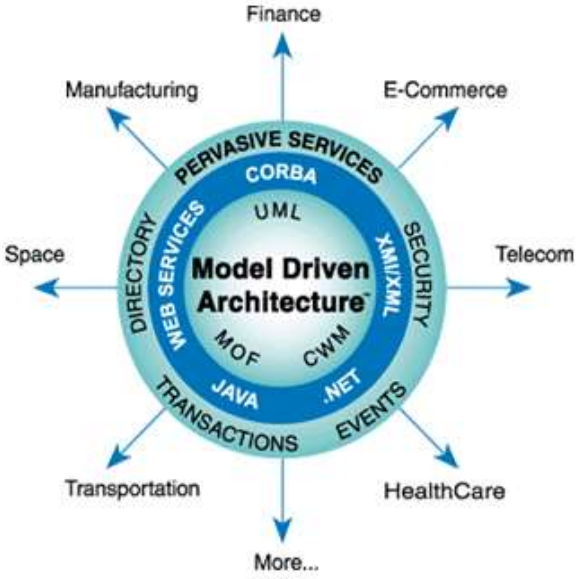


Figura 1.6.1 El entorno de MDA

MOF: ***Meta-Object Facility***

MOF es un estándar muy relacionado con UML que permite la gestión de la Metadata y la definición del lenguaje de modelado. Existen dos formas de ver a MOF, la primera, es desde el punto de vista del modelado en donde MOF es utilizado para definir un modelo de información para un dominio de interés en particular. La segunda forma es desde el punto de vista de los datos, aquí el producto del MOF es utilizado para aplicar el paradigma de computación distribuida basado en OMA para gestionar la información correspondiente a un modelo.

CWM: ***Common Warehouse Metamodel***

Interfaces estándar que pueden ser usadas para permitir el fácil intercambio de warehouse y metadata del negocio entre herramientas de warehouse, plataformas de warehouse y warehouse metadata que se encuentran distribuidos en ambientes distintos.

XMI / XML: ***XML Interchage Metadata***

XMI significa *Intercambio de Metadata a través de XML*, una aplicación XML que facilita la estandarización del intercambio de los modelos y de la metadata a través de Internet entre grupos de trabajo en equipos de desarrollo que utilizan herramientas y aplicaciones de diferentes distribuidores.

XMI esta basado en tres estándares de la industria, XML, UML y MOF. La arquitectura permite que las herramientas compartan metadata usando interfaces XML o CORBA especificadas en los estándares UML o MOF.

MDA busca separar la lógica de la aplicación de la plataforma tecnológica, de esta forma, las aplicaciones que se desarrollen en MDA pueden correr en plataformas propietarias o abiertas tales como .NET, J2EE, WebServices, CORBA entre otros, a continuación se explican algunas de las plataformas con las cuales interactúa MDA:

.NET:

.NET es un ambiente de desarrollo en el cual el usuario puede crear y desplegar sus aplicaciones así como componentes de nueva generación. Todas las herramientas existentes en Microsoft como Visual Studio e incluso Office serán gradualmente integradas en .NET y cada uno de ellos brindara servicios que permitirán una mayor integración entre los productos

WEB SERVICES:

El termino Web Service describe la estandarización de aplicaciones basadas en Web que utilizan XML, SOAP, WSDL y estándares abiertos UDDI sobre un protocolo de Internet. XML es usado para etiquetar la información, SOAP es usado par enviar los datos, WSDL para describir los servicios disponibles y UDDI es usado para listar los servicios disponibles. Usado fundamentalmente como medio para la comunicación entre negocios y entre estos y clientes, Web Services permite a las organizaciones comunicarse sin necesidad de profundizar en las tecnologías que cada uno utilice.

Los Web Services permiten a las diferentes aplicaciones de diferentes fuentes, comunicarse entre si sin gastar mucho tiempo en codificaciones especiales para la tecnología y todo porque las comunicaciones se llevan a cabo con XML. Los Web Services no están ligados a un sistema operativo o lenguaje de programación. Por ejemplo, Java puede comunicarse con Perl o aplicaciones Windows con aplicaciones Linux.

CORBA:

CORBA es una arquitectura que permite que partes de los sistemas llamados objetos, se comuniquen entre si a pesar del lenguaje de programación en que fueron escritos o sobre que sistema operativo están instalados. CORBA fue desarrollado por el consorcio empresarial OMG.

Existen muchas implementaciones de CORBA, las más utilizadas son SOM y DSOM de IBM. CORBA también a sido recibido por Netscape como una parte de su producto Netscape ONE (RMI de Red Abierta). Tres modelos que compiten son DCOM de Microsoft y DCOM y RMI de SUN Microsystems

1.4.2 El ciclo de vida de MDA

El ciclo de vida que propone MDA se muestra en la figura 1.7 En esta figura notamos que las fases del proceso en general son las mismas del proceso que comúnmente se utiliza, la única diferencia radica en la cantidad de modelos que reproducen, ya que como se ve en la figura solo existen 3, (CIM, PIM, PSM) sin embargo, la documentación sigue teniendo un rol importante y se sigue generando pero en menor proporción.

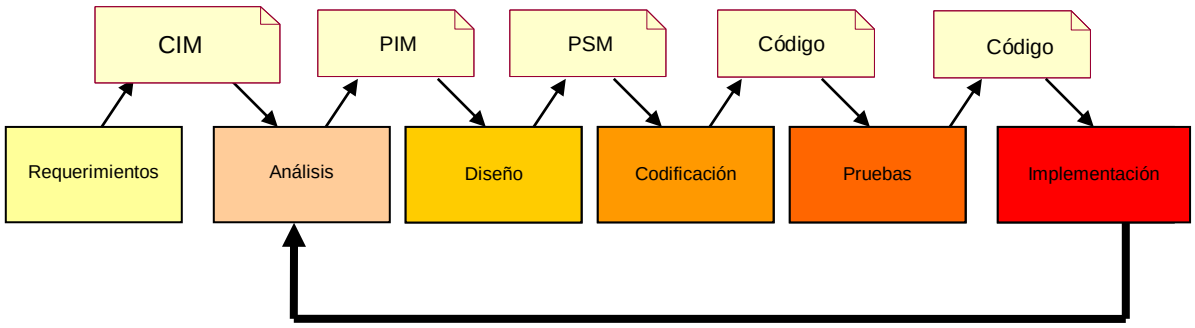


Figura1.7 Ciclo de Vida del MDA

Los modelos que se generan en MDA son los siguientes:

1.4.2.1 CIM (Modelo independiente de la computación): En este se modelan los requerimientos del sistema describiendo la situación en la cual el sistema será desarrollado. Utilizando la similitud con el proceso de desarrollo UP, el CIM se puede constituir con el modelo de negocio, el modelo conceptual y la información de los atributos de calidad y de funcionalidad del sistema, que nos ayudan a ocultar detalles de operación y nos permiten ver con claridad lo que se espera de él. Este modelo es muy importante ya que nos ayuda a entender el problema y el contexto en el que se va a desarrollar y donde funcionará el sistema, además nos brinda un lenguaje en común para utilizarlo en diferentes ocasiones. En la especificación MDA el CIM debe estar relacionado con los modelos PIM (Modelo independiente de la plataforma) y con el PSM (Modelo dependiente de la plataforma) es decir, que debe existir trazabilidad. Cuando un modelo cambie los demás deben cambiar.

1.4.2.2 PIM (Modelo independiente de la plataforma): Este modelo describe al sistema, pero no muestra detalles acerca de la plataforma en el cual estará implementado. Este puede llevarse a diferentes tipos de plataforma o similares, si es el caso.

El punto de vista independiente de la plataforma es muy amplio, ya que pueden existir modelos de plataformas específicas que respecto a otros modelos, son totalmente independientes. Esto tiene que ver con qué tanto, las clases de estos modelos, se ven afectadas cuando corren sobre otros modelos. Por ejemplo, las maquinas virtuales están definidas como un conjunto de partes o servicios, los cuales, están definidos y creados para trabajar independientemente de cualquier plataforma específica. Sin embargo, una maquina virtual es una plataforma y un "modelo" para la maquina virtual, es especifico para esta, pero, este "modelo" es independiente de la plataforma sobre la cual esta maquina virtual se ha implementado. Para dar otro ejemplo, los PIM´s de los sistemas de tecnologías en internet son en cierta forma, dependientes de un estilo arquitectural Cliente-Servidor, así que son dependientes de una arquitectura específica, que sin tener que ver con tecnología, dan características específicas al sistema.

1.4.2.3 PSM (Modelo dependiente de la plataforma): Este modelo combina las especificaciones del PIM con detalles que especifican cómo debe el sistema usar un tipo específico de plataforma.
El arquitecto del sistema escoge una o varias de las plataformas heterogéneas que permiten la implementación del sistema con las características y requerimientos deseados. El PIM entonces se convierte en uno o varios PSM (Para cada plataforma específica es generado un PSM.) teniendo en cuenta modelos de plataformas con estructuras específicas como por ejemplo J2EE en donde se encuentran componentes que tienen términos como "Interfaz Home" "Bean Entidad", o un modelo relacional donde se encuentran términos como "tablas", "columnas" etc.
Algo muy importante que se debe recalcar es que un PSM *debe* incluir todos los detalles necesarios para la transformación a código. Si no se puede incluir en el modelo los detalles completos del sistema, se debe anexar información adicional por medio de documentación en el caso de la transformación manual o por medio de componentes software en lenguaje de transformación para poder llevarlos a código. El PSM tiene toda la información necesaria para construir un sistema y ponerlo en operación. Si este modelo no contiene todos los elementos del sistema, podría adoptar un papel distinto y convertirse en un PIM intermedio que más adelante podría ser refinado y llevado a un PSM para ser directamente implementado.

El punto de vista dependiente o independiente de la plataforma tiene mucho que ver con el concepto de plataforma, y lo que se tome como plataforma depende de la clase de sistema a ser implementado. Por ejemplo, desde el punto de vista de un desarrollador de middleware o de frameworks, las clases de plataformas serán los sistema operativos u otras plataformas, así, cada plataforma objetivo es un sistema operativo en particular. Para los usuarios de MDA la plataforma

será el framework o el middleware y el modelo independiente de la plataforma del desarrollador de middleware será un modelo específico de la plataforma para el desarrollador de aplicaciones.

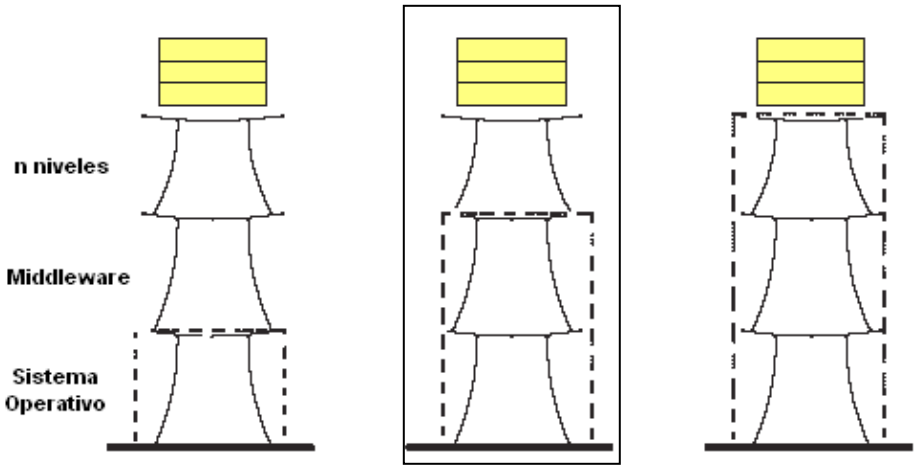


Figura 1.8 Qué se cuenta como plataforma[MDAGUI].

El concepto de "dependencia" o "independencia" de la plataforma esta sujeto al sistema que se desee realizar o al punto de vista que adopte el desarrollador. En la figura 1.8 se puede observar que cualquiera de las partes del modelo encerradas en líneas punteadas puede ser considerado como plataforma. Para el OMG y el marco de Trabajo de MDA, lo que se adopta como plataforma es el middleware, es decir aquellas tecnologías como CORBA, J2EE o .Net y así se tomará de ahora en adelante en este trabajo de grado (Ver recuadro figura 1.8).

1.4.2.4 Código: En la especificación MDA este es el paso final donde cada elemento de los modelos PSM se transforma a código, por medio de herramientas que lo hacen automáticamente o manualmente, teniendo en cuenta patrones de diseño, de implementacón y especificaciones del lenguaje de la plataforma que se escoja.

MDA define todos los modelos anteriores y lo más importante, cómo estos modelos se relacionan entre sí. Existe entonces una clase de reglas de transformación para poder hacer el mapeado o las transformaciones de un modelo a otro. Estas reglas pueden estar escritas en lenguaje natural, como algoritmo, o como un lenguaje para mapeo de modelos, lo que importa es que este lenguaje sea portable para poder usarlo en diferentes herramientas de diseño, en el caso de transformaciones automáticas.

1.5 LA TRANSFORMACIÓN DE LOS MODELOS

Como se definió anteriormente, la transformación es el proceso de convertir un modelo a otro modelo en el mismo sistema. Existen varios modelos de transformación, como son el modelo de transformación por meta-modelos, el modelo de transformación aplicando patrones y el modelo de transformación por marcas [MDAGV1]. En este trabajo de grado se utilizarán los dos últimos.

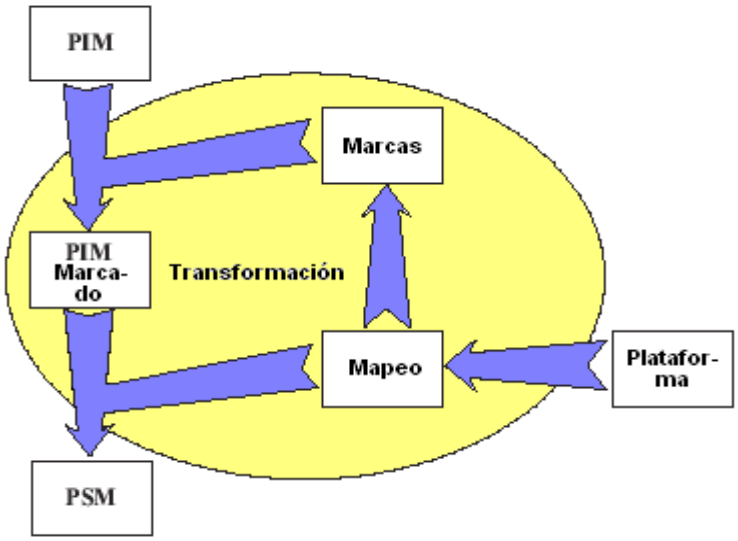


Figura 1.9 Modelo de Transformación con Marcas

En el modelo de transformación por marcas (Ver figura 1.9), se escoge la plataforma, se realiza el mapeo que incluye un conjunto de marcas las cuales son usadas para identificar ciertas restricciones en los elementos del PIM y guiar las transformaciones hacia el PSM. El modelo marcado se transforma, según el mapeo, en el PSM. Se debe producir un registro de transformación junto con este modelo.

El modelo de transformación que aplica patrones, consiste en, como su nombre lo dice, aplicar patrones ya sean arquitecturales, de diseño o de implementación según el modelo o el nivel de abstracción en el cual se esté trabajando. Este modelo de transformación, junto al mapeo por marcas es de gran utilidad en el proceso de generación del PSM (Ver figura 1.10).

Como se observa se tiene el modelo PIM marcado con las restricciones obtenidas de los patrones de diseño. Nombres de patrones se refieren a los nombres y conceptos de los patrones de diseño que son específicos de cada plataforma. Los elementos marcados en el PIM se transforman según el patrón que se utilice, para producir el PSM.

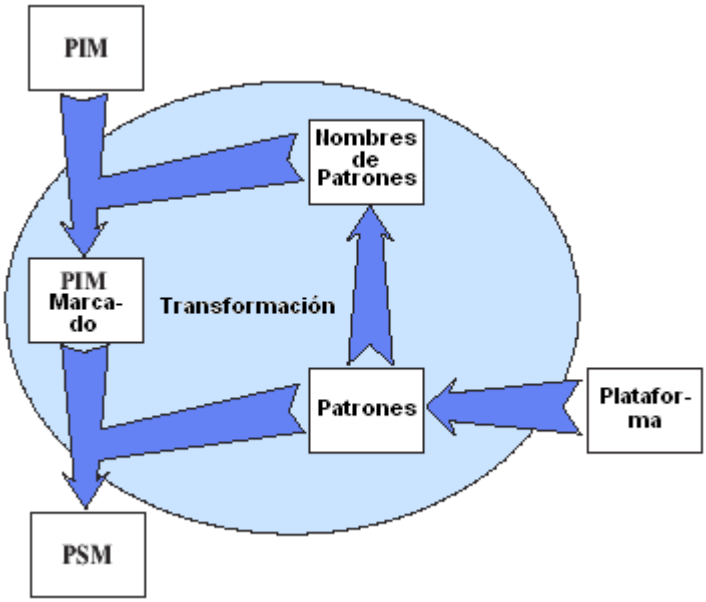


Figura 1.10 Modelo de transformación usando Patrones y Marcas

Además de los patrones y las marcas también se puede incluir información adicional, para guiar las transformaciones. Ejemplos de información adicional es la información de atributos de calidad, o las especificaciones de las cualidades de servicio del sistema.

En la figura 1.11 se puede observar el concepto de adición de información adicional.

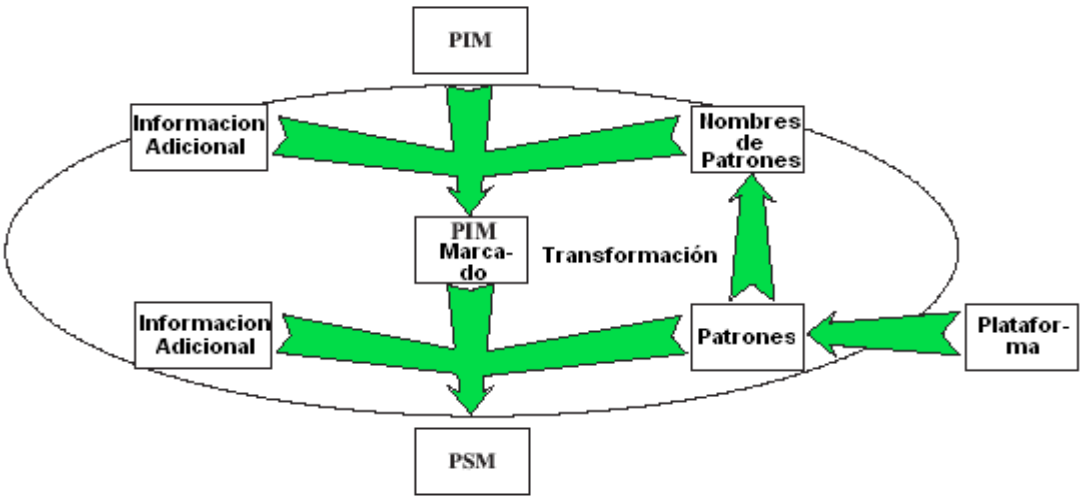


Figura 1.11 Uso de Información adicional en la Transformaciones

Se puede observar, que se puede incluir información adicional antes de marcar el PIM y después que este haya sido marcado.
El objetivo de este trabajo de grado es combinar la transformación por marcas con la transformación por patrones de diseño, combinados con patrones arquitecturales tradicionales y atributos de calidad que el sistema demanda, para así formar los diferentes modelos de MDA.

Los anteriores ejemplos se realizan con los modelos PIM y PSM ya que son los que más se han trabajado en este nuevo enfoque. El modelo CIM, es nuevo y no se ha tomado en cuenta para el desarrollo de transformaciones MDA, pero en este trabajo de grado es considerado ya que éste es el que aporta las cualidades de negocio al sistema y es a partir de éste que se genera el PIM.

1.5.1 Enfoque MDA

En el enfoque MDA según [MDAEX] existe una gran diferencia entre la transformación y definición de transformación. La primera es el proceso de generar un modelo de otro modelo y la segunda es lo que las herramientas necesitan para saber cómo se deben transformar los modelos. Estas definiciones de transformación contienen información acerca del lenguaje del modelo objetivo. Se tiene por ejemplo Definiciones de transformación de UML a C++ o de UML a SQL dependiendo de los PSM y la plataforma en que se vaya a trabajar. Ver figura 1.12
Actualmente muchos de los modelos que genera el proceso de desarrollo son transformados de manera manual o automática, ayudados con herramientas de desarrollo software como Rational Rose, que crean el código a partir de estos. El problema está en que el código creado no se acerca a la realidad y los desarrolladores deben terminarlo a mano.

Figura1.12 La definición de transformación entre lenguajes

Lo que propone MDA es tomar los modelos y transformarlos automáticamente por medio de herramientas como se ve en la figura 1.13 Aunque no se descarta la posibilidad de mapear

manualmente los modelos, la ventaja de automatizar estos procesos esta en el rendimiento que esto le puede entregar al desarrollo software.

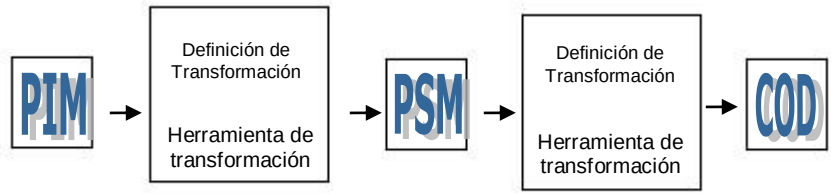


Figura 1.13 Definiciones de transformación dentro de las herramientas de transformación

Muchos de los desarrolladores de aplicaciones software que utilizan el procedimiento normal gastan considerablemente mucho tiempo en construir código o componentes desde un modelo de alto nivel. Con las herramientas MDA se reduce este problema.

1.5.2 Herramientas de Transformación

Las herramientas existentes hoy en día todavía no alcanzan el nivel para convertir el modelo PIM a PSM o a código ya que de todas maneras el desarrollador manualmente debe corregir los modelos PSM y/o código. Hasta el momento las herramientas que existen en el mercado transforman el PIM a un modelo PSM que tiene funcionalidades básicas, una especie de esqueleto de PSM. Estas herramientas son de mucha ayuda y permiten mantener la persistencia a la hora de hacer modificaciones en uno de los modelos. El CIM no tiene una especificación o unas reglas claras ya que es uno de los últimos modelos en MDA y las herramientas no lo toman en cuenta hasta el momento, pero es necesario su realización y posterior transformación según los patrones arquitecturales y la diferente información adicional para poder llegar al PIM.

A continuación se lista el tipo de herramientas software que brindan un buen soporte al momento de aplicar MDA:

- Herramientas de Transformación PIM a PSM: Existentes en el mercado con pocas funcionalidades.
- Herramientas de Transformación PSM a Código: Esta filosofía de transformación la siguen herramientas tradicionales las cuales permiten desde un diagrama de clases generar automáticamente código.
- Herramientas de Transformación PIM a Código: Este tipo de herramienta es la ideal ya que reduce considerablemente el desarrollo software. Hasta el momento las herramientas que

tienen esta característica manejan UML y los desarrolladores deben añadir información adicional manualmente.

Se sabe que en el mercado existen ya este tipo de herramientas para facilitar el proceso de desarrollo software. En la siguiente lista encontraremos las herramientas más comunes en el mercado para la transformación de los modelos MDA:

1.5.2.1 ArcStyler: Es una plataforma de Interactive Objects Software que automatiza los procesos de desarrollo software. Esta hecha en Java y soporta plataformas como J2EE y .Net así como cualquier plataforma. [ArcStyler] trabaja con módulos llamados Cartuchos-MDA los cuales permiten capturar la experiencia y las habilidades de los desarrolladores y arquitectos para obtener un alto nivel de calidad y re-uso de elementos entre los grupos de trabajo. ArcStyler trabaja con UML haciendo más comprensible los procesos. Esta herramienta se basa en modelos visuales y construye aplicaciones a partir del modelo PIM.

1.5.2.2 OptimalJ: Herramienta desarrollada por [Compuware] acelera el desarrollo de las aplicaciones J2EE generando estas a partir de modelos visuales con la filosofía que brinda MDA. Por estas características OptimalJ decrece la necesidad de codificar y diseñar a bajo nivel, en consecuencia se logra gran productividad y consistencia. OptimalJ soporta transformación modelo a modelo y modelo a código.

1.5.2.3 Xcoder/J – Xcoder/EJB: Estas son herramientas de Liantis IT Solutions[3] las cuales son software libre, que permiten convertir los modelos hechos en UML a código java o EJB, para esto sólo se utilizan diagramas de clase, otros modelos serán descartados a la hora de la transformación. Se utiliza Rational Rose para modelar las clases (No esta sujeto a esta herramienta). Tiene las características de las herramientas anteriores, con la ventaja de ser software libre.

1.5.2.4 AndroMDA: Esta herramienta es un framework de libre distribución para la generación de código que sigue el paradigma de MDA. Este toma modelos UML y genera clases o componentes J2EE. Esta herramienta no posee interfaz de usuario, ya que se basa en la ejecución de comandos para transformar los modelos generados en por otras aplicaciones como Racional Rose.

	Software Libre	Transforma de PIM a PSM	Transforma de PIM a Codigo	Presta soporte para el proceso de desarrollo	Trabaja con diferentes tecnologías (.NET, J2EE, Java)
ArcStyler		x	x	x	.NET / J2EE

[3] http://www.liantis.com/

OptimalJ		x	J2EE
Xcoder/EJB	x	x	J2EE
Xcoder/J	x	x	Java
AndroMDA	x	x	J2EE / Java /Web Pages (Struts)

Los compiladores de estas herramientas no están muy completos ya que estas herramientas aún son muy incipientes, pero lo que se debe tener en cuenta es el potencial de cambio radical en el desarrollo software, además de la evidente ventaja de trabajar en el más alto nivel de abstracción. En el capitulo cuatro de este trabajo de grado se hablará en más detalle de las características que una herramienta de transformación MDA debe tener.

1.6 BENEFICIOS DE MDA

Muchos son los beneficios que MDA le brinda al desarrollo software [MDAEX]. Tomando los problemas que tiene el desarrollo software, anteriormente mencionados tenemos que MDA brinda soluciones para mejorar:

1.6.1 La Productividad

Con MDA el desarrollador se enfoca más en el desarrollo del PIM ya que desde este modelo se pueden generar los demás. Esto es más reconfortante para el desarrollador ya que el trabajo se independiza de la plataforma especifica en la cual el sistema será implementado.

Dos de las principales razones por las cuales la productividad aumenta son las siguientes:

Primero, menos trabajo para los desarrolladores porque no se tiene que especificar aspectos de la plataforma. Se realiza menos código ya que mucho del código se genera desde el PIM.

Segundo, los desarrolladores se enfocan más en el PIM que en el código gastando más tiempo al análisis del negocio, lo cual conlleva a satisfacer más las necesidades del cliente o usuario final.

Pero la verdadera ganancia está en utilizar buenas herramientas de desarrollo que hagan las respectivas transformaciones.

1.6.2 La Portabilidad

El PIM puede ser transformado en diferentes PSM´s , entonces todo lo que se especifique en el PIM será portable; esta característica depende de las herramientas de transformación que se utilicen; para las nuevas plataformas habrá indudablemente herramientas, para las viejas plataformas quizá se tenga que realizar las transformaciones manualmente.

1.6.3 Interoperabilidad

MDA no solo se encarga de la generación automática de los diferentes PSM´s sino que también se encarga de los puentes o conexiones entre estos para que haya comunicación entre los sistemas.

En la figura 1.14 se puede observar los puentes de comunicación entre los modelos. Más adelante cuando se expliquen las reglas de transformación de MDA veremos la importancia que tienen estos puentes de comunicación.

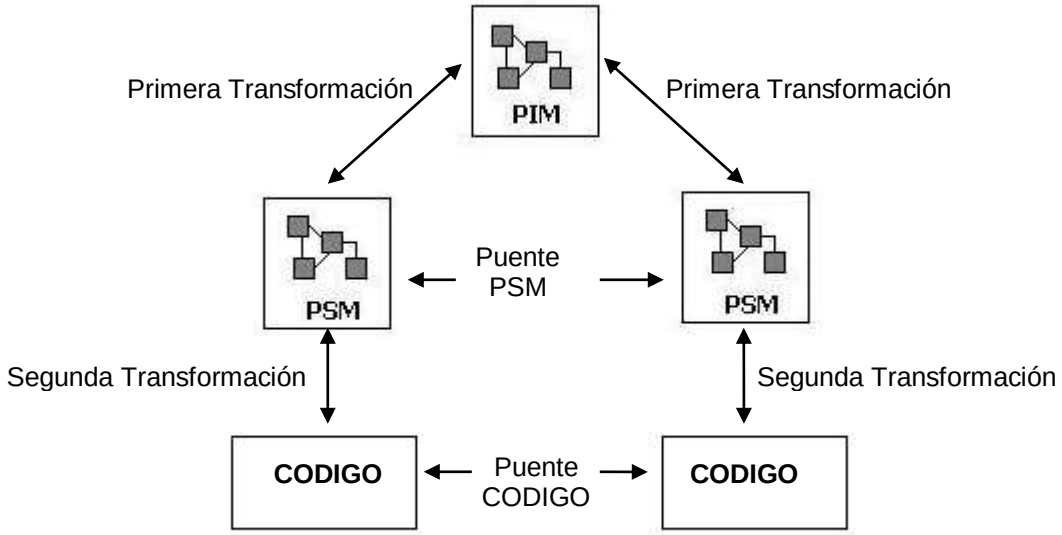


Figura1.14 Puentes de Interoperabilidad en MDA

1.6.4 Mantenimiento y Documentación

El PIM es una abstracción en alto nivel del código así que los cambios que se produzcan en este modelo se ven reflejados en el código. Algunas de las herramientas mencionadas anteriormente cambian el código a partir del PSM, pero lo ideal es que se cambie a partir del PIM y se mantenga consistencia en los cambios. Y viceversa, si el PSM cambia las herramientas deben estar en capacidad de hacer los correspondientes cambios en PIM. Por lo anterior el PIM debe ofrecer la información necesaria de alto nivel que otros modelos brindan y es aquí donde se ahorra tiempo.
La documentación siempre ha jugado un papel muy importante, pero a veces se vuelve engorrosa y difícil de manejar. En MDA de todas maneras se deben documentar los modelos, aunque no con la misma intensidad que en los procesos normales, pero si es importante dejar por escrito las decisiones los cambios en los modelos, por lo menos en los registros de configuración.

1.6.5 Reutilización a nivel arquitectónico y generación automática de código

En el ciclo de vida de un proyecto la reutilización es importante, pero cobra más importancia cuando trabajamos las re-utilización a nivel de la arquitectura, ya que esto provee grandes ventajas para los sistemas que tienen similares requerimientos. Así que no sólo el código puede ser re-

utilizado, también los requerimientos, artefactos y decisiones arquitecturales pueden liderar en principio la toma de decisiones oportunas para un proyecto.

La generación automática de código está entonces muy ligada a la reutilización a nivel arquitectónico ya que con los modelos que se creen en un principio, basados en los requerimientos y modelos de negocio, se generan modelos más específicos que llevan a la creación automática del código como tal.

La generación automática de código es un factor muy importante en MDA ya que hace los procesos de desarrollo mucho más ágiles. Además, el hecho de tener normas para realizar transformaciones entre patrones, implica tener una trazabilidad muy bien definida entre los modelos, aparte de esto, si se añade el hecho que la generación automática puede ser realizada por herramientas software, se tiene una alta mejora en los procesos así como una disminución de tiempo de desarrollo haciendo mucho más eficiente el proceso de creación de aplicaciones software.

1.6.6 Inconvenientes de MDA

Ya que MDA es un enfoque muy reciente, existen algunas ventajas e inconvenientes. A continuación se listan algunos de ellos.

- La especificación de MDA plantea tres tipos de transformaciones, por Metamodelos, por Marcas o por Patrones [MDAGV1], cada uno de estos 3 tiene sus características propias, sin embargo en el caso que se desee realizar una aplicación de muy alta complejidad, en donde tenga que manejarse muchos aspectos arquitecturales y conceptos específicos que no estén normalizados (En una empresa desarrolladora de software), la mejor opción es la transformación por metamodelos, sin embargo este tipo de transformación es muy compleja y necesita mucho estudio y preparación. Las otras dos opciones por marcas y por patrones presentan la ventaja de ser mucho más simples que la transformación por meta-modelos, pero para aplicaciones muy grandes, ésta es la mejor opción.
- MDA tiene como fundamento los modelos CIM y PIM para la creación de todos los demás modelos, sin embargo por la falta de madurez de esta, no se ha especificado en qué momento y de qué forma se deben capturar los requisitos no funcionales de la aplicación a pesar de ser un factor crítico en el desarrollo, haciendo que el comienzo del desarrollo pueda ser muy desordenado y la creación de los modelos sea poco precisa. Por esto es necesario un adecuado proceso de desarrollo.

- MDA es un framework incipiente en el mercado por esto carece de los lineamientos adecuados que debe tener un proceso de desarrollo confiable como lo tiene UP, es por esto que para este proyecto tuvo que tomarse el proceso de UP que brinda una muy buena organización. La idea es enriquecer a MDA con la coordinación y organización de los flujos de trabajo para llegar a la realización de los modelos. De todas maneras muchas empresas han puesto su confianza en MDA y están trabajando para que este enfoque surja y sea la solución que todo desarrollador estaba esperando.

1.7 EJEMPLO DEL MARCO DE TRABAJO DE MDA

Para dar una visión global de lo que es el marco de trabajo de MDA y algunas de las normas básicas que [MDAEX] propone para este enfoque, se tiene el siguiente ejemplo, el cual consiste de un modelo en alto nivel de lo que sería un PIM y a continuación el modelo correspondiente al PSM para java, escrito en UML.

1.7.1 De atributos públicos a privados

Para empezar se debe observar la figura1.15:

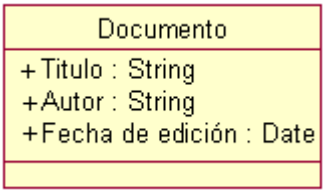


Figura 1.15 Modelo Independiente de la Plataforma PIM

La clase *Documento* consta de tres atributos públicos *Titulo, Autor,* y *Fecha de edición*. El objeto de este ejemplo es convertir este pequeño PIM en su correspondiente PSM. Para esto primero nos dedicaremos a la transformación de los atributos públicos a sus correspondientes elementos en el PSM. Es común encontrar los atributos como públicos en un modelo PIM ya que esto da flexibilidad para los continuos cambios que se presentan. Pasa lo contrario con el PSM, pues siguiendo normas de diseño se aplica el concepto de encapsulamiento y se accede a los atributos por medio de las operaciones. Como se puede ver en la figura 1.16. el objeto Documento tiene absoluto control sobre el uso y las modificaciones de sus atributos.

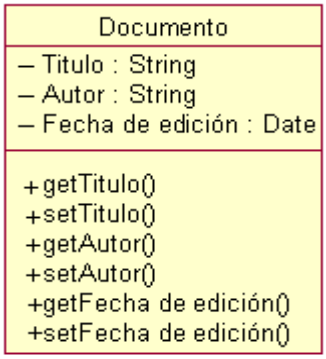


Figura 1.16 Modelo Dependiente de la Plataforma PSM JAVA

Para transformar un modelo independiente de la plataforma PIM a uno dependiente de la plataforma específica PSM se siguen ciertas reglas de transformación. A continuación se muestra en la Tabla 1.1 algunas transformaciones (Aplicadas al ejemplo), que en capítulos posteriores se utilizarán y se apliarán para la transformación de los modelos del Sistema de Gestión Documental.

PIM	**Regla de Transformación PIM a PSM**	**PSM**
Clase Documento	Para cada clase class<Nombre> en el PIM debe haber una class<Nombre> en el PSM	ClaseDocumento
Atributo +Titulo:String	Por cada atributo publico llamado Atributo:Tipo, existe: • Un Atributo privado con el mismo nombre • Una operación pública getAtributo retornando el tipo de atributo • Una operación pública setAtributo con el tipo de atributo como parámetro.	Atributo –Titulo:String Operación +getTitulo():String Operación +setTitulo(Titulo:String)

Tabla 1.1 Ejemplo de las reglas para pasar de PIM a PSM

Así mismo las reglas de transformación se invierten para pasar de PSM a PIM ya que debe existir persistencia entre los modelos. Ver Tabla 1.2.

PSM	**Regla de Transformación**	**PIM**
Clase Documento	Para cada clase class<Nombre> en el PSM debe haber una class<Nombre> en el PIM	ClaseDocumento
Atributo –Titulo:String Operación +getTitulo():String Operación +setTitulo(Titulo:String)	Por cada combinación de los siguientes atributos y operaciones en la misma clase <Nombre> existe un atributo publico llamado Atributo:Tipo • Un Atributo privado con el mismo nombre del atributo en cuestión. • Una operación pública getAtributo retornando el tipo de atributo • Una operación pública setAtributo con el tipo de atributo como parámetro.	Atributo +Titulo:String

Tabla 1.2 Ejemplo de las reglas para la transformación inversa de los modelos

1.7.2 Asociaciones: Para poder apreciar otra de las reglas de transformación MDA, se crea otra clase llamada Proceso la cual esta ligada a la clase Documento. Ver figura 1.17

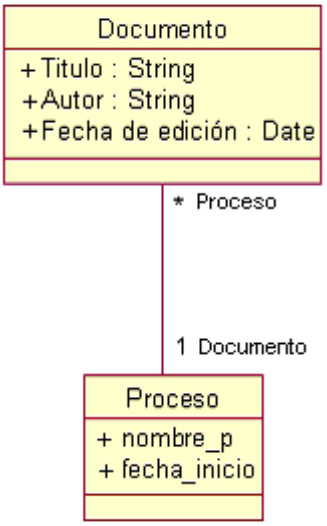


Figura 1.17 Modelo PIM extendido

Las reglas de transformación para las asociaciones se pueden observar en la tabla 1.3

En la Clase Documento	Regla de Transformación	En la clase Proceso
-Proceso	Por cada asociación <NombreAsociación> hay un atributo privado de nombre<Nombre Asociación> en la clase opuesta	-Documento
-Proceso: Proceso	El tipo de este nuevo atributo será: Del tipo de la Clase si su asociación es cero o uno Del tipo Set si la multiplicidad de la asociación es más de uno	-Documento: Set
getProceso():Proceso setProceso(p:Proceso)	Para cada atributo nuevo creado le corresponde las operaciones set y get correspondiente, siguiendo las mismas reglas de los atributos	getDocumento:Set setDocumento(d:Set)

Tabla 1.3 Ejemplo Reglas para asociaciones de clases

Para observar el resultado de las transformaciones en nuestro ejemplo se debe observar la figura 1.18

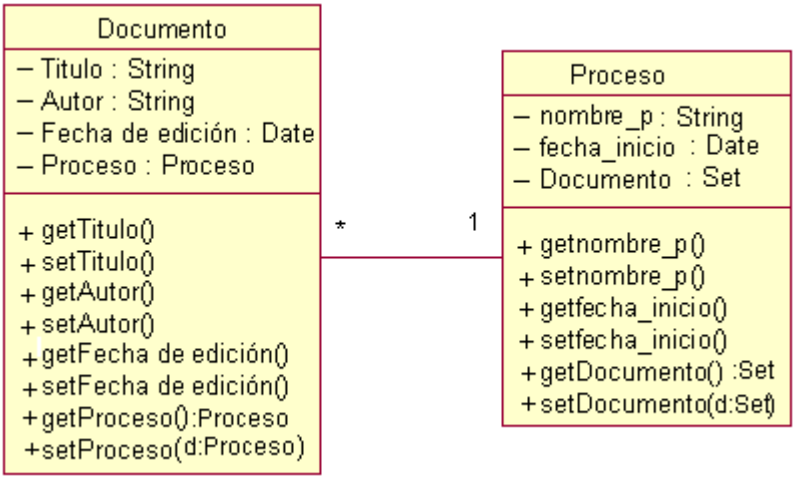


Figura 1.18 PSM del modelo PIM extendido

Como hemos observado en el ejemplo anterior existen ciertas reglas que pueden ser aplicadas para la transformación de los modelos, pero estas reglas pueden cambiar de acuerdo a las necesidades de los desarrolladores. Si se utilizan herramientas configurables, estas permiten

colocar restricciones o condiciones especificas diferentes a las propuestas por [MDAEX]. La mayoría de estas herramientas trabajan con el lenguaje [UML] y [OCL] para la definición de estas restricciones.

1.8 EL PROCESO DE DESARROLLO PARA MDA

El grupo OMG no ha especificado un proceso de desarrollo software para trabajar con MDA, pero si recomienda una buena selección ya que de este proceso depende el rendimiento y la satisfacción de los resultados, además el proceso que se escoja debe permitir trabajar con todas las pautas y especificaciones dadas por MDA.
Los siguientes, son los procesos de desarrollo más destacados en el ámbito del desarrollo software

1.8.1 Programación Extrema

La programación extrema es uno de los llamados procesos ágiles, el cual su filosofía principal se basa en escribir código y aumentarlo poco a poco al sistema, cada incremento del código esta supervisado por test de prueba específicos para cada uno de estos incrementos. Cuando se añaden nuevas funcionalidades al código, se compila el nuevo test y los anteriores para asegurar que la funcionalidad no este dañada. A parte de esto, [XP] también tiene otras características:

- **Desarrollo iterativo e incremental**: pequeñas mejoras, unas tras otras.
- **Pruebas unitarias continuas**, frecuentemente repetidas y automáticas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- **Programación por parejas**: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. El código es revisado y discutido mientras se escribe.
- **Frecuente interacción del equipo de programación con el cliente** o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- **Corrección de todos los errores** antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- **Refactoreo del código**, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en el refactoreo no se ha introducido ningún fallo.
- **Propiedad del código compartida**: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores sean detectados.
- **Simplicidad** en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario.

1.8.2 Proceso Unificado (UP)

Este en un proceso más complejo en comparación con los anteriores. Se utiliza para proyectos extensos en los cuales se desea tener control sobre todo el sistema; es un proceso guiado por casos de uso, muy completo en donde se tienen en cuenta aspectos como: riesgos, mitigación de riesgos , flujos, arquitectura del sistema, entre otros. UML es de vital importancia para este proceso y es aquí donde RUP puede ser utilizado para trabajar con MDA ,debido a los inconvenientes que se mencionaron anteriormente, ya que con los modelos producidos pueden fácilmente convertirse en PSM y automáticamente a código, claro está que se necesita configurar el RUP para que funcione con todos los requerimientos propuestos de MDA.

Un de los objetivos de esta tesis es tener una primera experiencia con el nuevo enfoque que brinda MDA y que junto a un proceso de desarrollo piloto, en este caso el proceso de desarrollo UP, se logre obtener los modelos propuestos por MDA para llevar a cabo la construcción del sistema de Gestión Documental basado en J2EE

CAPITULO 2. EL PROCESO DE DESARROLLO Y MDA: ENFOQUE MDA-UP

Como se ha visto anteriormente, MDA es una nueva forma de desarrollar aplicaciones basándose en un CIM (Modelo independiente de la computación), un PIM (modelo completamente independiente de la plataforma) hecho en UML, más uno o más modelos PSM dependientes de la plataforma en la cual el desarrollador decide implementar la aplicación.
El desarrollo de MDA se enfoca primero en la funcionalidad y el comportamiento de una aplicación o un sistema distribuido sin tener en cuenta la tecnología o tecnologías en las cuales será implementado. En este enfoque, se separa completamente la implementación del modelo del negocio, de esta manera, no es necesario repetir el modelado ni el comportamiento de las funcionalidades del sistema cada vez que se quiera implantarlo en una tecnología diferente. Otros modelados arquitecturales por lo general están atados a plataformas específicas.

Lo que se pretende con este trabajo de tesis es tratar de incorporar, los conceptos y el enfoque que brinda MDA, con el tradicional proceso de desarrollo UP que hasta el momento es manejado por la facultad de Ingeniería Electrónica y Telecomunicaciones, y que hoy en día, es el proceso de desarrollo más completo y confiable que los desarrolladores de sistemas utilizan. Así, a partir de todos los artefactos que se utilizan en UP podremos llegar a conformar los modelos que MDA propone y así empezar a experimentar con la Arquitectura Dirigida por Modelos.

2.1 CONCEPTOS BÁSICOS DEL PROCESO UNIFICADO DE DESARROLLO SOFTWARE

El Instituto de Ingeniería Software SEI, define el proceso de desarrollo del software como un conjunto de actividades, métodos y prácticas usadas por un grupo de personas para desarrollar y mantener el software y sus productos asociados [SEI].

En general el proceso de desarrollo se puede ver como un conjunto de actividades y procedimientos que permiten realizar un sistema software a partir de los requerimientos que entrega el cliente, requerimientos que se transforman en una serie de modelos hasta obtener el producto final (figura 2.1). El proceso de desarrollo unificado utiliza UML para realizar los modelos.

UP es un proceso guiado por casos de uso y centrado en la arquitectura y lo más importante es un proceso iterativo e incremental.

Figura 2.1 Concepto del proceso de desarrollo software

A continuación se describen las tres características más importantes del proceso de desarrollo:

2.1.1 El Proceso de Desarrollo es guiado por casos de uso

Los casos de uso son las diferentes funcionalidades que el sistema ofrece a sus usuarios, en ellos se encuentra toda la información de los requerimientos del sistema y son estos los que guían al desarrollador a lo largo del proyecto. El proceso de desarrollo es guiado por casos de uso ya que desde el inicio de las actividades los casos de uso brindan información a los desarrolladores para construir las clases, los diagramas de secuencia, las interfaces que el usuario requiere y lo más importante ayuda a repartir actividades a los diferentes grupos de trabajo. Esta es la razón por la cual se dice que el proceso de desarrollo unificado es dirigido por los Casos de Uso.

2.1.2 El Proceso de desarrollo esta centrado en la arquitectura

La arquitectura es importante ya que nos brinda una visión general del sistema y ayuda a los desarrolladores a trabajar sobre una misma estructura para alcanzar el objetivo final. Haciendo una analogía con la construcción de un edificio la arquitectura comprende los planos y las guías para realizar la construcción. La arquitectura debe ser muy flexible ya que debe ser capaz de adaptarse a los cambios que se produzcan y permitir de esta manera la re-usabilidad.
Las claves para realizar una buena Arquitectura según [ARQPRA] son:

- El arquitecto debe ser una sola persona o un grupo muy pequeño de personas.
- El arquitecto debe tener una lista de los requerimientos técnicos, con características priorizadas las cuales el sistema espera satisfacer.
- Se debe documentar muy bien la arquitectura preferiblemente con notaciones claras.

- La arquitectura debe circular hacia los grupos de trabajo los cuales siempre deben involucrarse en la revisión.
- Los módulos deben ser de tal manera que representen los grupos de trabajo en la organización.
- La arquitectura no debe depender de una versión específica de una herramienta comercial.
- Los módulos que producen datos deben estar separados de los módulos que consumen datos. Esto es para facilitar la adición o la supresión de módulos en el sistema.

2.1.3 El proceso Unificado es iterativo e incremental

Las iteraciones hacen referencia al flujo de trabajo, empiezan con los casos de uso, pasa por las diferentes etapas análisis, diseño, implementación y termina en los casos de uso en forma de código ejecutable. El incremento se refiere al crecimiento del producto en estas iteraciones. La intención, consiste en ir trabajando poco a poco en el sistema ya que si se deben realizar cambios, no se produzca una situación grave que pueda producir la pérdida de tiempo y trabajo.
Apoyado por UML el UP es uno de los procesos de desarrollo más acogidos por los desarrolladores ya que gracias a las características mencionadas anteriormente este proceso ha dado muy buenos resultados en el desarrollo de productos software.

2.2 EL CICLO DE VIDA DEL PROCESO UNIFICADO

El proceso unificado propuesto por Grady Booch y Jacobson está compuesto por ciclos lo cual hace más fácil la gestión de la evolución del sistema, en el tiempo. Cada ciclo a su vez se divide en fases, y estas a su vez en un número de iteraciones que contienen flujos de trabajo logrando así al término de cada ciclo una nueva versión del producto.

Las fases del proceso de desarrollo son:
Fase de Inicio: En esta fase se establece la viabilidad del proyecto, haciendo un análisis del negocio, seguido de la lista de los posibles riesgos críticos, un esbozo de la arquitectura sobretodo lo que cause riesgo, una delimitación del ámbito del sistema y la posible viabilidad del proyecto.
Fase de Elaboración: En esta fase se realizan los casos de uso más críticos y se construye la arquitectura, esto ayuda a determinar la factibilidad del sistema.
Fase de Construcción: Aquí se llega a un prototipo del producto el cual puede ser entregado a los usuarios finales. En esta fase se finaliza el análisis, se mantiene la arquitectura y se revisan los riesgos críticos acumulados desde las dos fases anteriores.

Fase de Transición: Aquí el producto se prueba por usuarios experimentados y se informa de defectos y deficiencias, se entrega una versión Beta a los usuarios finales acompañada de manuales y documentos de soporte. Se revisan los errores y se verifica la compatibilidad con el entorno del usuario final.

Los flujos de trabajo se ubican dentro de cada iteración en cada fase del UP. Estos flujos son un conjunto de actividades las cuales contienen ciertos artefactos de UML para poder estructurar el sistema. Los flujos más importantes son:

Planificación: En este flujo se asignan los tiempos de cada fase, los objetivos a cumplir por fase, y en general el plan del proyecto donde se definen el número de fases e interacciones.

Captura de Requerimientos o especificación: En este flujo se recoge la información de las diferentes funcionalidades que el sistema debe hacer según los requisitos del cliente. En este flujo se estudia el contexto del sistema y se recogen los requerimientos funcionales y no funcionales.

Análisis: En este flujo se produce el modelo conceptual del sistema, se realizan las clases, paquetes y casos de uso de análisis. Por último se realiza la vista arquitectural, que es la integración lógica de todos los artefactos mencionados anteriormente.

Diseño: En este flujo se modela el sistema, se realizan las clases de diseño, las interfaces y dependencias de los subsistemas, se realizan los casos de uso de diseño, se obtiene el modelo de despliegue donde se ven todas las especificaciones físicas y de red y se genera una vista arquitectónica del modelo de diseño.

Implementación: En esta etapa ya se implementa el sistema en términos de componentes como son: el código fuente, scripts, archivos binarios, código ejecutable entre otros.

Pruebas: Aquí se verifica el resultado de la implementación probando cada uno de los componentes realizados. Aquí se generan casos de uso de pruebas los cuales nos brindan la información necesaria de las pruebas que se le deben hacer al sistema.

Además de los anteriores flujos de trabajo existen componentes de soporte para la gestión del proyecto, el control y la organización de los diferentes grupos de trabajo, la supervisión de los proyectos, la escogencia de las herramientas adecuadas, y las actividades que se requieren para implantar el sistema en el cliente. Estos son: el componente de gestión, el de entorno y el de implantación respectivamente.

Cada iteración pasa por estos flujos de trabajo pero en mayor o menor proporción dependiendo de la fase en que se encuentran. Para entender mejor todos estos flujos se puede observar la figura 2.2:

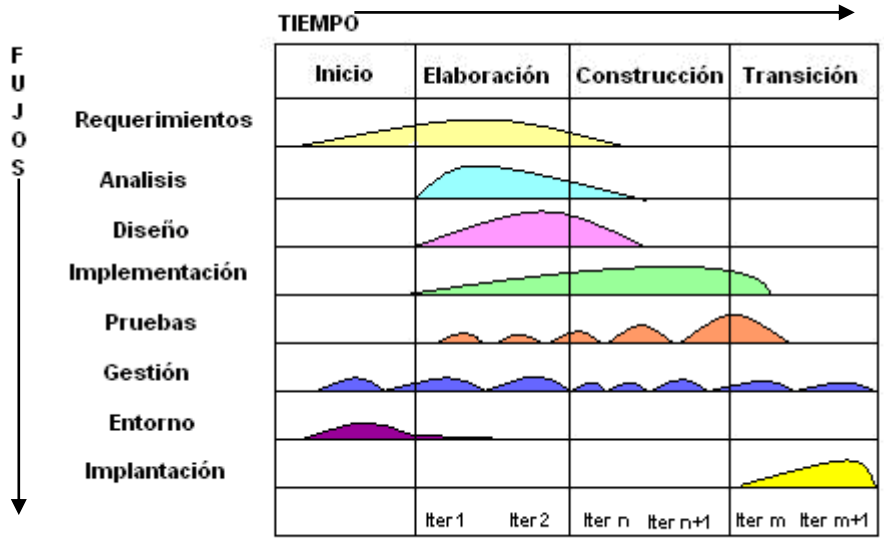


Figura 2.2 Organización del Proceso Unificado

Como ya se ha dicho UML es una parte esencial en este proceso y es con la ayuda de este lenguaje que se fusionan dos grandes paradigmas de desarrollo el UP y el MDA. Los artefactos producidos en el proceso como son casos de uso, diagrama de clases, diagrama conceptual entre otros, pueden asociarse a la generación de el PIM o PSM y estos pueden ser convertidos automáticamente a código mediante el uso de una muy buena herramienta MDA o si se quiere manualmente con técnicas tradicionales de codificación.

La ventaja que brinda el proceso UP a MDA, es la organización de los procesos para la obtención de los principales modelos como es el CIM y el PIM, ya que en MDA no se especifica cómo deben obtenerse estos modelos. Para la generación de los otros modelos PSM y código el MDA propone la realización de reglas de transformación según la especificación propuesta por el OMG [MDAGV1] y además reglas de transformación de acuerdo a patrones de diseño y arquitecturales que siempre se han manejado con UP. Más adelante en el desarrollo del sistema se puede ver lo anterior con más detalle.

2.3 UNIFICACION DEL MDA Y UP: ENFOQUE MDA-UP

Teniendo en cuenta todos los conceptos vistos hasta el momento y algunas reglas de transformación de MDA se realizan a continuación una serie de diagramas en los cuales se describen los principales flujos del proceso de desarrollo con los artefactos que se generan en UP y con los cuales podemos llegar a los modelos de MDA.

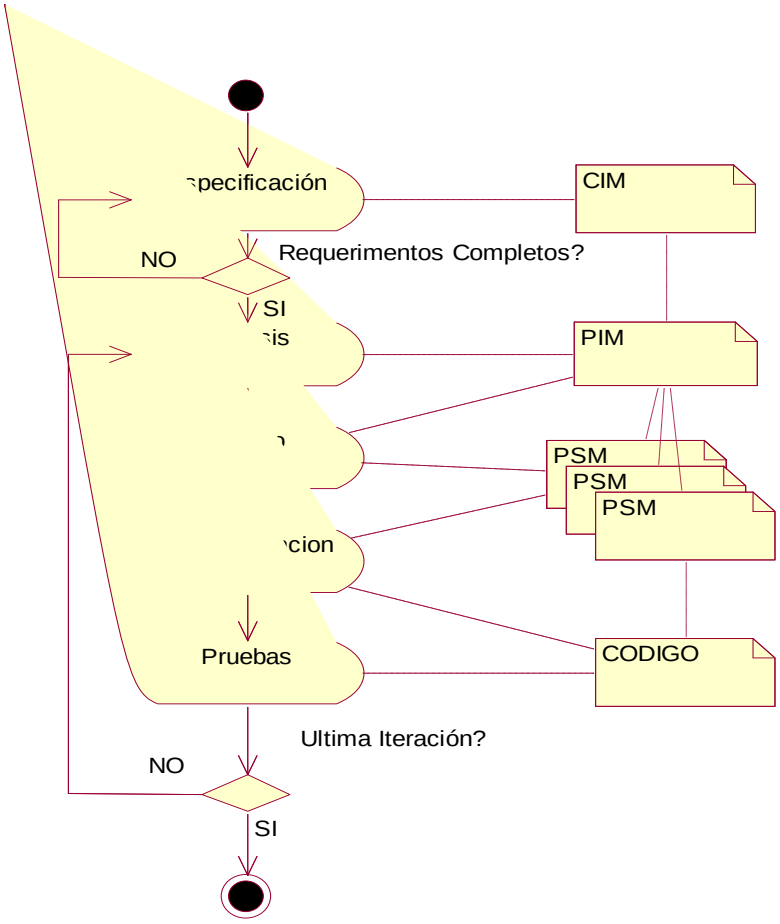


Figura 2.3 Ciclo de vida de MDA

El flujo de trabajo de MDA se muestra en la figura 2.3. Se puede observar el proceso de desarrollo con los flujos y las iteraciones que maneja UP. De cada uno de los flujos se generan los diferentes modelos. A partir de la información del modelo independiente de la computación CIM se crea un modelo independiente de la plataforma un PIM. Después, a partir de este PIM se crean los

diversos PSM y luego el código de la aplicación. Con estos modelos se puede manejar la información del sistema a diferentes niveles de abstracción de una manera más práctica.

Las transformaciones de los diferentes modelos se realizan de dos formas: automática por medio de las herramientas de conversión, las cuales contienen algoritmos y reglas de transformación estándar dependiendo de la arquitectura a la cual se va a llevar al sistema y, manual realizando el mapeo y las transformaciones manualmente, aplicando las reglas de transformación de MDA entre los modelos y utilizando patrones de diseño conocidos para llegar del PIM al PSM y luego al código.

En este trabajo de grado se pretende combinar las reglas básicas de transformación de MDA con reglas de transformación que surgen a partir de los diferentes patrones de diseño, requerimientos de calidad del sistema, estilos arquitecturales, y reglas de transformación que surgen de los conceptos de MDA y la experiencia del desarrollo de este Sistema de Gestión Documental propuesto.

2.3.1 Diagramas del enfoque MDA-UP

Para empezar a entender un poco el proceso de generación de modelos, se dará un resumen de los diferentes flujos de trabajo básicos del UP y la generación de diagramas o artefactos para la construcción de los modelos MDA.

2.3.1.1 Especificación: En la figura 2.5 se muestra el primer flujo de trabajo en UP, Especificación, el cual nos genera información para el primer modelo del enfoque MDA, el CIM.
Como ya se había dicho el CIM se enfoca en el entorno del sistema y sus requerimientos, así, los detalles de la estructura y del procesamiento del sistema están ocultos en este modelo.
Como el CIM es relativamente nuevo los autores no lo toman en cuenta en el momento de explicar el enfoque MDA, pero es de vital importancia ya que las características del entorno del sistema quedan plasmadas en este modelo. No existen como tal reglas de transformación del modelo CIM al PIM pero si se crean, se debe mantener la consistencia entre modelos ya que, según las especificaciones MDA, cualquier cambio que afecte al modelo origen debe afectar el modelo objetivo.
Según la experiencia de trabajo, el CIM (Modelo Independiente de la Computación) se puede obtener mediante el modelo de negocio del UP que es fruto del flujo de captura de requerimientos y esta compuesto por diferentes diagramas como son los diagramas de casos de uso del negocio y el modelo de objetos del negocio.

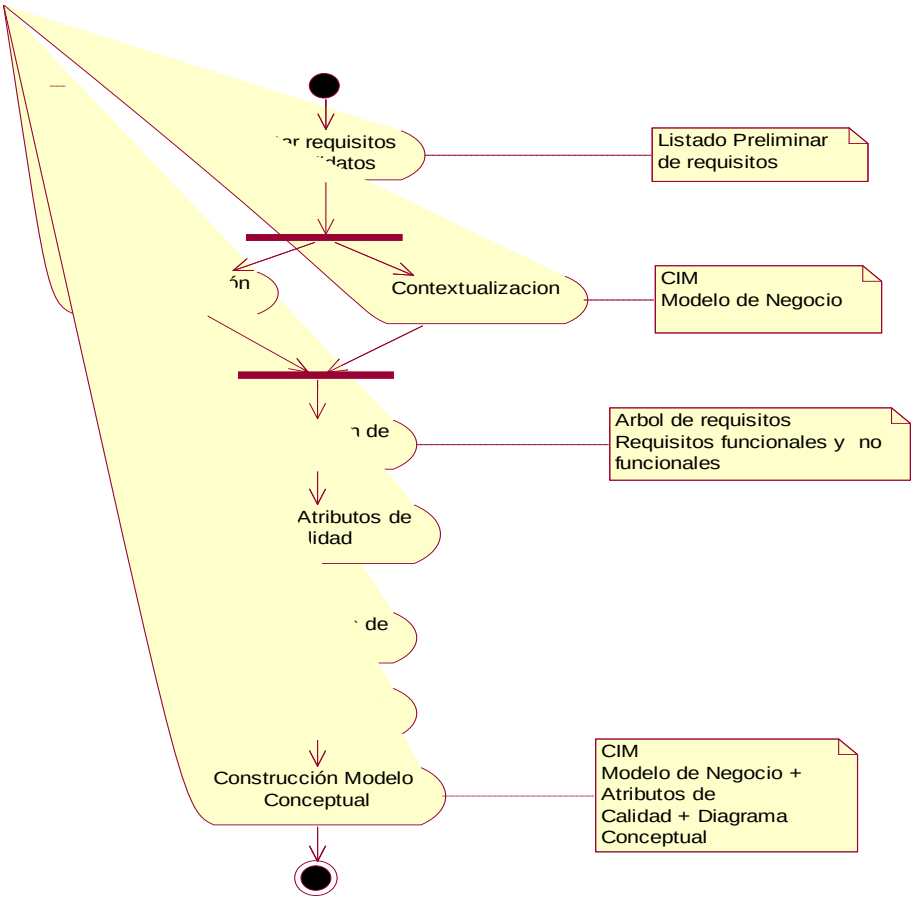


Figura 2.5 Flujo de trabajo: Especificación

Pero, además de la información del modelo de negocio, el CIM requiere información adicional para ser transformado en PIM, como son los atributos de calidad del sistema y el patrón o estilo arquitectural. Los atributos de calidad son de mucha importancia ya que actúan sobre las funcionalidades del sistema y el estilo arquitectural o la arquitectura que se escoja, ayuda en la obtención de estos atributos de calidad que el cliente necesita. Como el CIM requiere de una estructura especifica, este se apoya en el modelo conceptual, un modelo concreto al cual se le pueden aplicar las reglas de transformación MDA según los patrones arquitecturales y los atributos de calidad correspondientes, para que pueda ser transformado y pueda llegar a convertirse en PIM. Este diagrama nos muestra las cosas y los eventos que ocurren en el ámbito del sistema y se crea a partir de las reuniones que se tienen con el cliente. Resumiendo, según la experiencia obtenida, el CIM es un modelo que se compone por tres cosas: modelo de negocio, el diagrama

conceptual y las características de calidad del sistema. En este momento el CIM está completo y se puede proseguir a la evaluación de los casos de uso del sistema.

2.3.1.2 Análisis: El segundo flujo de trabajo del RUP es análisis (Ver figura 2.6). Es aquí donde surge el PIM, generado con toda la información que contienen el CIM, los casos de uso y su descripción, los paquetes de análisis y las clases de análisis.

Los casos de uso, brindan la información de las funcionalidades que va a tener el sistema. Estos son de vital importancia, ya que a partir de su descripción y priorización, podremos luego llegar a las clases de análisis. Cuando se recopiló información sobre MDA casi todos los ejemplos que había en estas fuentes de información daban por hecho que el PIM se había generado a partir de

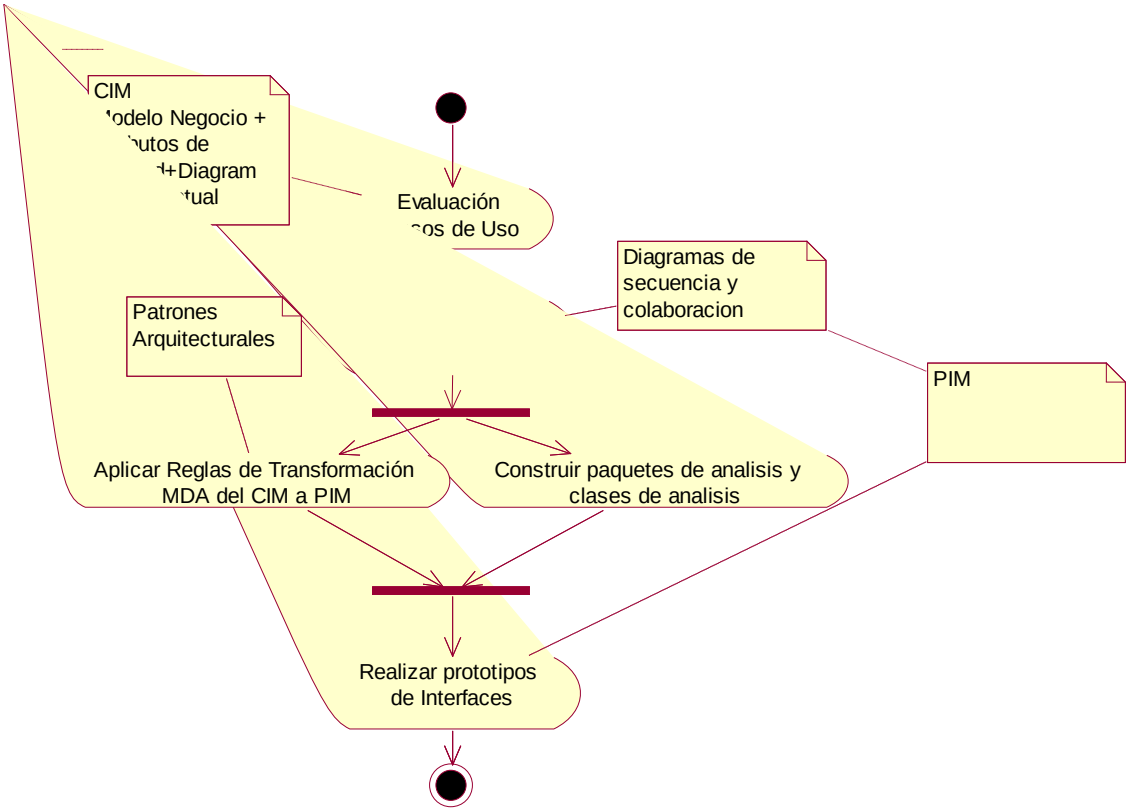


Figura 2.6 Flujo de trabajo: Análisis

los conocimientos que el lector había adquirido anteriormente con UML, y no mostraban detalle de su elaboración. Sin embargo es importante anotar, que estos PIM, parecían solamente diagramas conceptuales del sistema hechos en UML, los cuales no tenían en cuenta los aspectos dinámicos del sistema.

Para poder cumplir con estos aspectos, los desarrolladores de MDA usan, para las reglas de transformación de las herramientas, la combinación los lenguajes [UML] y [OCL] Object Constraint Lenguage lo que permite especificar la parte dinámica del sistema mediante pre y post-condiciones sobre las operaciones del sistema. Esto si se pretende hacer la transformación usando herramientas que automaticen el proceso. Como en nuestro caso las transformaciones se hacen manualmente, los diagramas de secuencia y colaboración o los de actividad ayudan a resolver este problema, permitiendo obtener las características dinámicas del sistema.
Más adelante se mostrará la aplicación de todos estos diagramas y conceptos en el ejemplo concreto para el Sistema de Gestión Documental.

2.3.1.3 Diseño: En la figura 2.7 vemos el flujo de trabajo de diseño:

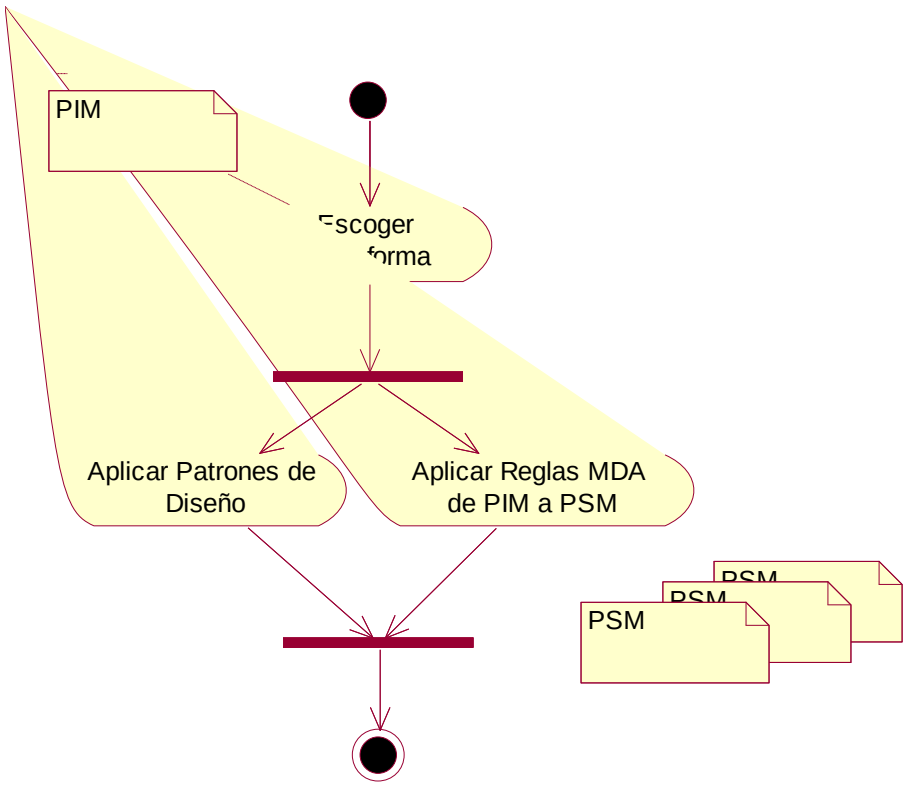


Figura 2.7 Flujo de trabajo: Diseño

En este flujo de trabajo se utiliza al PIM, combinado con la información de la plataforma específica a implementar y la información de los patrones de diseño. Las marcas que se realizan en el PIM son específicas del patrón de diseño que se utilizará. A partir de esto se crean los distintos PSM, de acuerdo a las reglas de transformación de MDA.

Según MDA, el PSM o PSM´s que se generan deben tener la información completa para ser convertidos a código. En este punto al PSM le hace falta información acerca de los diferentes componentes del código, por ejemplo, y la manera de cómo debe ser implantado físicamente. Las herramientas de transformación cuentan con módulos que contienen esta información según la tecnología que se desee utilizar. Como la transformación manual necesita de esta información es necesario pasar al siguiente flujo de trabajo en donde se encuentran los diagramas de componentes e implantación.

2.3.1.4 Implementación: El siguiente flujo de trabajo figura 2.8 es la Implementación. Con los PSM´s, las reglas de transformación especificas de la plataforma, los patrones de implementación (IDIOMS) que son código reutilizable, la información suministrada por los diagramas de componentes y diagramas de implantación podremos llegar al código final, de la manera tradicional con la que se trabaja en RUP, ya sea por reingeniería con los diagramas en Racional o simplemente codificando.

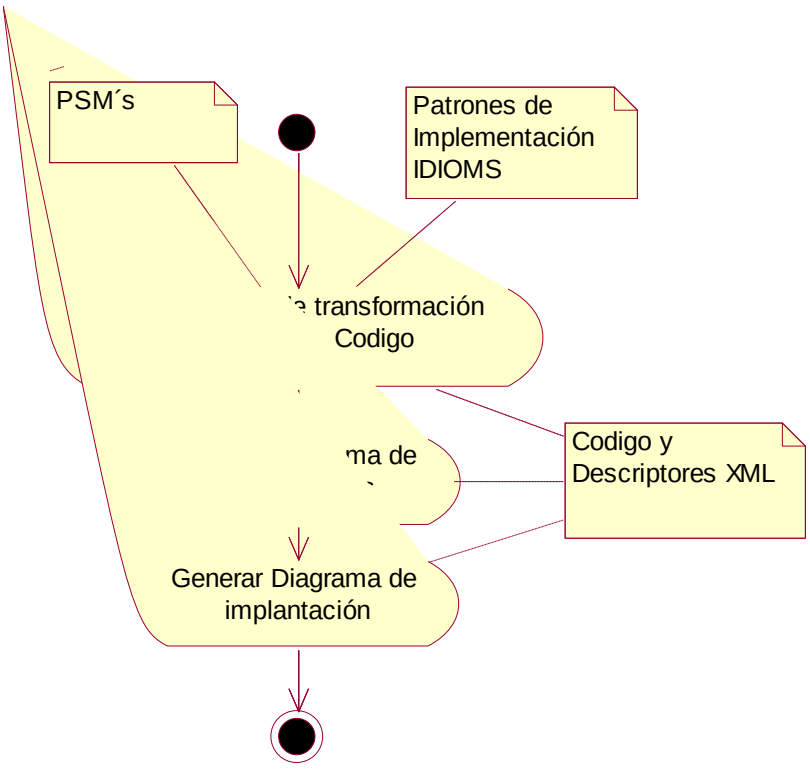


Figura 2.8 Flujo de trabajo: Implementación

Como se puede observar el uso de herramientas de transformación automática en esta parte del proceso es de gran importancia para agilizar la creación de código. Y no sólo es esta parte del

proceso, lo ideal, y lo que el grupo OMG busca con el enfoque MDA, es tener herramientas de transformación tan avanzadas que me permitan introducir el PIM para ser transformado automáticamente a código. Esto permite que los proyectos de desarrollo software obtengan las aplicaciones con menos esfuerzo y ganen en productividad, tiempo y costos.

2.3.1.5 Pruebas: En este flujo se realiza un plan de pruebas para asegurar que el código funcione bien. Los fallos que se encuentren realimentan a las iteraciones para producir los distintos cambios al código. Estos cambios demandan trazabilidad entre los diferentes modelos. Si un elemento de uno de los modelos cambia, el elemento mapeado en el siguiente modelo también debe cambiar y así en todos los modelos, por esto se requiere la realización de un registro de transformación. El resumen de todos los diagramas anteriores se tiene la figura 2.9

El enfoque MDA se nutre de los flujos del proceso de desarrollo UP para poder obtener los modelos a partir de las transformaciones en forma manual. Cuando se utilizan herramientas el proceso se hace más ágil y algunos flujos del proceso tienden a desaparecer, pero lo más importante es que estas herramientas permitan obtener la información necesaria en cada etapa del proceso.

.

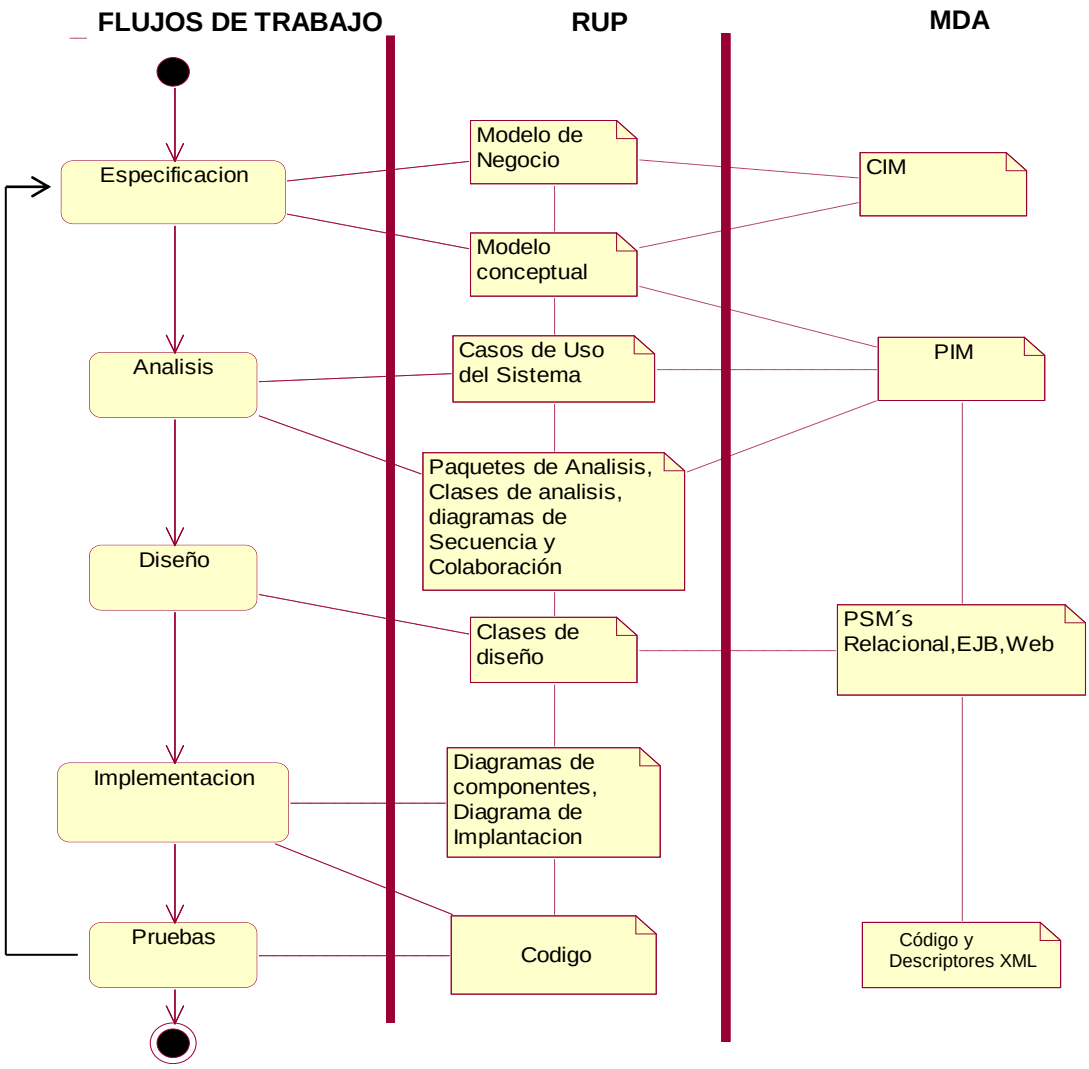


Figura 2.9 Creación de los modelos MDA a partir de los artefactos UP

CAPITULO 3. EXPERIENCIA CON EL DESARROLLO DEL SISTEMA DE GESTION DOCUMENTAL BASADO EN EL ENFOQUE MDA-UP

La fusión de MDA con el proceso UP potencializa el desarrollo software de las empresas desarrolladoras, sin embargo, son muy pocos los trabajos que se han realizado con estos dos elementos, y si existen, el acceso a la documentación se encuentra muy restringido. Es por esto que en este capitulo se presenta un primer acercamiento al enfoque MDA-UP aplicado al Sistema de Gestión Documental bajo la norma ISO 9001:2000 para la empresa GAMMA INGENIEROS S.A., para que se pueda visualizar tal potencialidad y sirva de base para futuros desarrollos.

Siguiendo paso a paso los flujos de trabajo vistos anteriormente se llegará a la obtención de los modelos CIM, PIM, PSM´s dando una serie de reglas de transformación en texto plano que ayudarán a los desarrolladores de software a llegar a cada uno de estos modelos MDA. Es importante destacar la obtención del CIM y el PIM, ya que no hay mucha información de cómo se pueden generar estos dos modelos. Se brinda también una serie de consejos, según la experiencia, a través del desarrollo del Sistema de Gestión Documental, para pasar de los PSM´s a código. En el cuarto capítulo se dará una serie de recomendaciones para aquellas empresas que pretendan trabajar con el enfoque y desarrollar herramientas de transformaciones MDA.

3.1 PRESENTACIÓN

Después de haber obtenido la certificación ISO 9001:2000 en sus procesos, con el Sistema de Gestión de Calidad actual, la empresa GAMMA INGENIEROS S.A., decide innovar y actualizar tecnológicamente los procesos de consulta a sus documentos de calidad (Procedimientos, formularios de registros, manual de calidad e información adicional) ya que esto permite agilizar los procesos de producción de sus servicios y así conseguir un buen lugar a nivel de competitividad en el mercado. Por esto se propuso al Departamento de Sistemas de la Facultad de Ingeniería Electrónica y Telecomunicaciones la realización de este proyecto. A continuación se hará una presentación corta del proyecto para darle paso al proceso de desarrollo según el enfoque UP-MDA visto anteriormente.

La norma ISO no exige una manera particular de cómo las empresas deben hacer sus Sistema de Gestión de Calidad, ni mucho menos pretende que éste sistema tenga una tecnología en particular.

El Sistema de Gestión de Calidad que posee la empresa GAMMA INGENIEROS S.A sigue los requerimientos de la norma ISO 9001:2000 y todos los procedimientos que ésta sigue están plasmados en documentos. Como estos documentos deben llegar a cada miembro de la organización, la empresa cuenta con una Intranet a la que el personal accede a los documentos del Sistema de Gestión de Calidad para realizar las actividades de los procesos, y a los Registros para evidenciar que las actividades diarias sí se realicen.

Los procesos de calidad de la empresa son los siguientes:

- **Dirección**: Es el proceso que se encarga de la parte de políticas y objetivos de calidad, además de la asignación de recursos humanos y presupuesto para cada proyecto en la empresa.
- **Gestión Calidad:** Maneja todas las actividades que tienen que ver con Calidad y el mantenimiento del Sistema de Gestión de Calidad.
- **Planificación:** Proceso que reúne las actividades necesarias para llevar a cabo un proyecto en la empresa.
- **Diseño:** Actividades del departamento de Proyectos con todo lo relacionado a diseño.
- **Construcción:** Actividades del departamento de Proyectos con todo lo relacionado a construcción.
- **Recursos Económicos:** Las actividades del departamento de contabilidad
- **Recursos Humanos y Recursos Materiales:** Los procesos que reúnen los recursos humanos y materiales que se necesitan para llevar a cabo un proyecto

Cada proceso tiene en la Intranet una determinada carpeta, en donde se guardan todos los registros y la información adicional que cada uno de estos procesos genere. En la carpeta de Gestión de Calidad se almacenan los “Documentos de Calidad” (procedimientos, manual de calidad, y formularios de registros). Sólo el administrador puede acceder a estos documentos y hacer cambio o modificaciones si es necesario.

Los documentos están en formato xsl y word el manejo de seguridad se realiza de acuerdo a políticas de permisos en las carpetas. Para ver más información acerca de la estructura de los documentos del sistema y la forma cómo estos se almacenan se puede remitir al anexo A.

El Sistema de Gestión de Calidad se soporta básicamente en la documentación y es aquí donde nuestro trabajo de tesis toma importancia ya que la solución que se pretende dar a la empresa es un Sistema de Gestión Documental que pueda ofrecer las mismas características que brinda sus sistema de gestión de calidad pero con muchas más facilidades en cuanto a distribución de los

documentos en Internet, interfaces mucho más amables al usuario y tecnología de punta para este tipo de aplicaciones empresariales como lo es [J2EE].

Nombre del proyecto: Sistema de Gestión Documental bajo la norma ISO 9001:2000
Cliente del proyecto: GAMMA INGENIEROS S.A.
Metas del Proyecto: El objetivo de este proyecto es desarrollar un sistema que permita apoyar al Ingeniero de Calidad de la empresa en el mantenimiento de los documentos del sistema de gestión de calidad y dar una herramienta a los empleados para que puedan acceder a estos documentos desde la Web desde cualquier sucursal de la compañía.

3.1.1 Características del Sistema

El Sistema de Gestión Documental sobre Web desarrollado con tecnología J2EE, almacenará todos los documentos del Sistema de Gestión de Calidad de la empresa, como son los procedimientos, el manual de calidad, los de información general y las plantillas de todos los registros, almacenados estos, por proyectos y por procesos. Para ver la estructura de los documentos remítase al Anexo A.
Los documentos pueden ser accedidos a través de la red por los diferentes miembros de la empresa por medio de restricciones de seguridad. Los registros hechos por los trabajadores quedan en la Intranet pero no forman parte del Sistema de Gestión Documental, están almacenados en los computadores de cada uno o en carpetas especificas de cada proceso en la red. Se debe permitir el control del cambio en los documentos y la creación de los registros.
A continuación se hace un listado de las características del sistema:

1. Realizar un sistema de gestión documental que ayude al Ingeniero de Calidad a llevar un control sobre los documentos de Calidad y de acceso de estos a los empleados de la empresa.
2. Administrar los documentos, es decir se pueden insertar, eliminar, modificar y ver.
3. Organizar los documentos por proyectos y por procesos.
4. Administrar a los usuarios del sistema.
5. Ver e insertar los datos correspondientes al control de los registros o evidencias de los empleados de la empresa.
6. Ver e insertar los datos correspondientes al cambio los documentos.
7. Sistema suficientemente flexible y escalable, de arquitectura cliente servidor, cuya parte cliente debe ser independiente del sistema operativo.
8. El sistema se puede acceder de manera remota.

3.2 DESARROLLO DEL CIM

Para llegar a construir el CIM (Modelo independiente de la computación) del sistema es necesario tener en claro la contextualización del sistema. El CIM es el conjunto de un modelo de negocio, junto al modelo conceptual y las definiciones de los requerimientos de calidad del sistema.

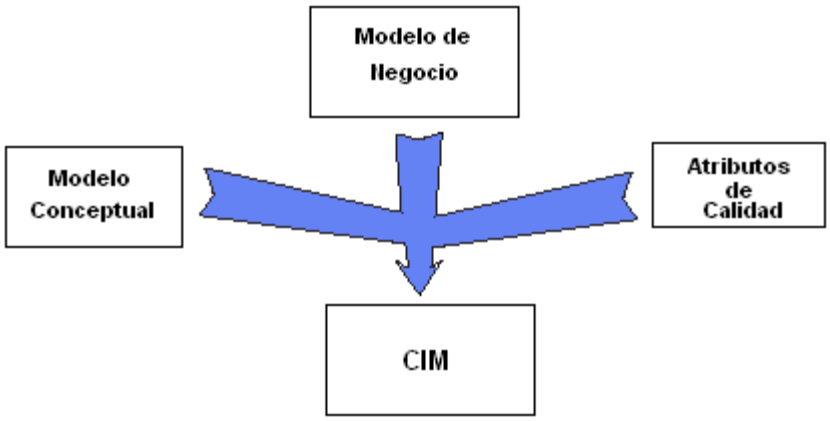


Figura 3.1 Representación gráfica de las partes del CIM

3.2.1 Modelo de Negocio

El modelo de negocio describe todo el entorno de la empresa donde el sistema va a interactuar. Este modelo consiste de casos de uso del negocio que representan los procesos de negocio de la empresa, actores de negocio que representan los clientes u actores externos a la empresa.
El modelo de la organización se muestra en la figura 3.2

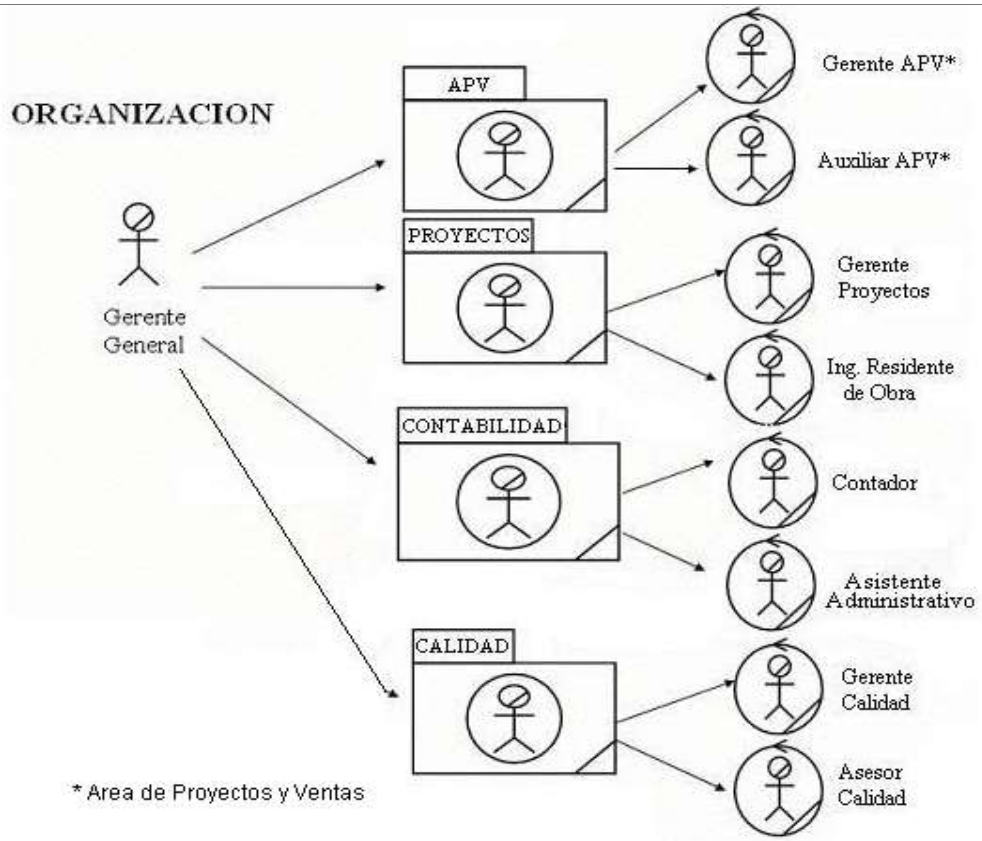


Figura 3.2 Modelo de la Organización GAMMA INGENIEROS S.A.

Existen cuatro departamentos en la empresa: APV, Proyectos, Contabilidad y Calidad. Cada departamento tiene un director y uno o más ayudantes. El gerente general dirige las operaciones que se realizan en cada uno de estos. El departamento de calidad esta compuesto por el Ingeniero de Calidad, persona que construye el sistema de gestión de calidad y lo mantiene, y un asesor de calidad el cual ayuda al gerente de calidad en las actividades de este departamento. Los procedimientos generales que la empresa GAMMA ING S.A. realiza para llevar a cabo sus objetivos y brindar los servicios que presta a sus clientes son los siguientes:

Procedimiento de licitación
Procedimiento de adjudicación de contratos
Procedimiento de realización de proyectos
Procedimiento de cobro de contratos
Procedimiento de Documentación.

Gracias a estos procedimientos se logra obtener una serie de casos de uso de negocio que representan el objeto de la empresa. Ver figura 3.3.

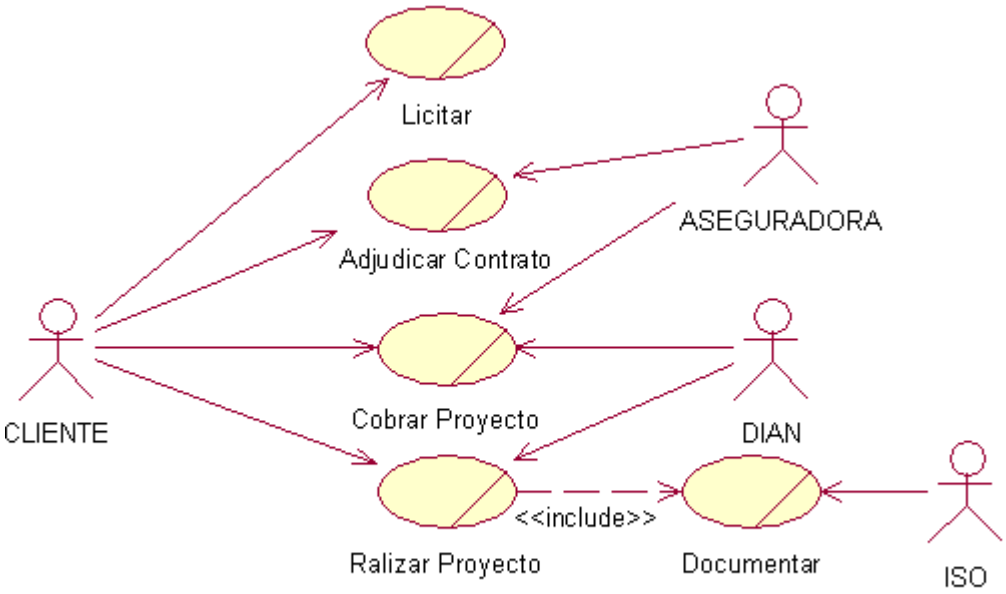


Figura 3.3 Casos de Uso del Negocio

- **Cliente:** Aquel que genera la necesidad de la realización de los procesos de toda la empresa para la consecución de servicios que satisfacen sus necesidades, por esto el cliente va a iniciar todos los casos de uso del negocio.

- **Aseguradora y DIAN:** Son actores externos que rigen las actividades de cada uno de los procesos en la empresa. Se crean actividades y departamentos para satisfacer las necesidades de estos.

- **ISO:** Es un cliente externo que mantiene auditando todos los procesos referentes a calidad. En el sistema.

Caso de Uso de Negocio Licitar: Este caso de uso empieza cuando el cliente publica la licitación y al interior de la organización se inician los procesos para realizar una propuesta, esto es, conocer las condiciones de la licitación, elaborar un presupuesto, hacer planeación de recursos y finalmente entregar estos resultados en un documento especifico.

Caso de Uso de Negocio Adjudicar Contrato: Este caso de uso se inicia cuando el cliente asigna a la organización el proyecto en cuestión. En el interior de la empresa se empiezan a realizar actividades relacionadas con la adquisición de pólizas, la revisión de los detalles del contrato y la formalización por medio de firmas de las partes el inicio del proyecto.

Caso de Uso de Negocio Realizar Proyecto: Este caso de uso hace referencia al desarrollo del proyecto, son las actividades que involucran la participación de los ingenieros, la elaboración de actas, consecución de materiales y las entregas parciales del proyecto al cliente, dentro de este se generan documentos de entrada y salida de inventarios en los almacenes.

Caso de Uso de Negocio Cobrar Proyecto: Este es el caso de uso relacionado con la parte económica final del proyecto, es dentro de este proceso donde se generan los documentos fiscales que solicita el gobierno, las facturas y otros documentos necesarios para el ingreso a la contabilidad de la información para ambas partes.

Caso de Uso de Negocio Documentar: Este caso de uso es esencial para la empresa ya que desde el momento del inicio del proyecto se empieza a documentar todos y cada una de las actividades y procesos. Según ISO 9001 las actividades de la empresa deben ir agrupadas en *procesos*, por esto, la empresa ha definido los siguientes procesos:

- **Dirección**: Es el proceso que se encarga de la parte de políticas y objetivos de calidad, además de la asignación de recursos humanos y presupuesto para cada proyecto en la empresa.
- **Gestión Calidad:** Maneja todas las actividades que tienen que ver con calidad y el mantenimiento del Sistema de Gestión de Calidad.
- **Planificación:** Proceso que reúne las actividades necesarias para llevar a cabo un proyecto en la empresa.
- **Diseño:** Actividades del departamento de proyectos con todo lo relacionado a diseño.
- **Construcción:** Actividades del departamento de proyectos con todo lo relacionado a construcción.
- **Recursos Económicos:** Las actividades del departamento de contabilidad
- **Recursos Humanos y Recursos Materiales:** Los procesos que reúnen los recursos humanos y materiales que se necesitan para llevar a cabo un proyecto.

Como este sistema va a dar soporte al Sistema de Gestión de Calidad en la parte de Gestión Documental, es necesario desglosar un poco más el caso de uso del negocio *Documentar*, para así empezar la contextualización del sistema objetivo. El siguiente diagrama de casos de uso de negocio involucra a los actores internos de la empresa con sus respectivas actividades para el caso de uso de negocio *Documentar*.

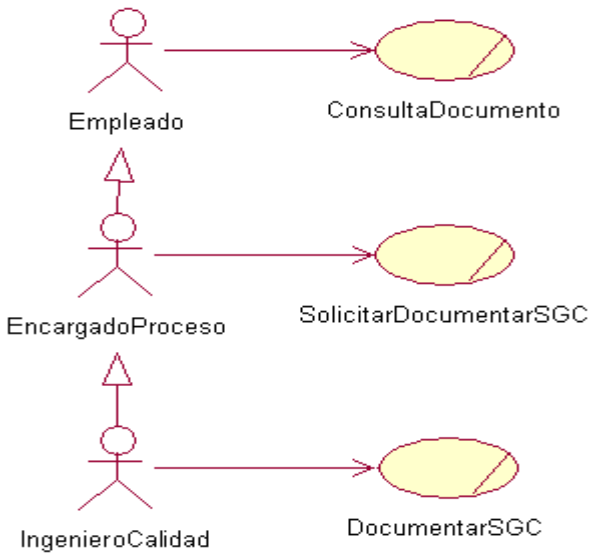


Figura 3.4 Extensión Caso de uso de Negocio Documentar

3.2.1.1 Actores del negocio

- **Empleado:** Representa a todos los empleados de la empresa, los cuales deben utilizar el sistema de gestión documental para consultar los documentos que ayuden a controlar sus actividades y procedimientos cuando se esta realizando un proyecto en la empresa.

EncargadoProceso: Es la persona que dirige el proceso de la empresa. Este puede proponer la creación de los siguientes documentos: procedimientos, formularios para registros o procedimientos. Hace solicitudes solamente, ya que ningún documento debe estar en el Sistema de Gestión de Calidad (SGC) sin la autorización del Ingeniero de Calidad.

IngenieroCalidad: Este es la persona que documenta el Sistema de Gestión de Calidad y da los permisos para la administración de documentos (creación, modificación y eliminación).

3.2.1.2 Procesos del Caso de Uso de Negocio Documentar: Los siguientes son los procesos que se llevan a cabo dentro del caso de uso de negocio *Documentar.*

Consultar Documento: Este proceso de negocio empieza cuando un empleado de la compañía necesita consultar un documento (Ver figura 3.5) en el sistema de gestión de calidad para poder registrar sus actividades del proyecto o ver los procedimientos de un proceso o simplemente consultar información de la empresa. Se realiza una copia del documento de calidad (de la plantilla), se llena, se guarda en una carpeta específica que cada proceso tiene en la red y se registra en el ControlRegistros del Sistema de Gestión.

SolicitarDocumentarSGC: Cada vez que haya la necesidad de crear, eliminar o modificar un documento de calidad por parte de un proceso, el encargado de este debe solicitar la autorización al ingeniero de calidad (Ver figura 3.6). Este revisa que cumpla con la norma ISO 2001 y autoriza el cambio. El encargado registra el cambio del documento en CambioDocumentos, el cual es un documento que hace parte del procedimiento obligatorio de la empresa.

DocumentarSGC: Este proceso de negocio lo inicia el Ingeniero de Calidad (Ver figura 3.7), cada vez que la empresa requiera de otro procedimiento, formato o cualquiera de los documentos que se utilicen en los procesos, además de la realización de los cambios a estos documentos.
El ingeniero analiza la necesidad de la realización del documento tomando nota de la información que suministre el proceso. Por lo general esto se hace en un comité o junta. Cuando ya se tiene claro esto, se consulta el procedimiento de la compañía para la realización y control de documentos. Este es un documento que se encuentra en el Sistema de Gestión de Calidad. Después se introduce el documento nuevo.

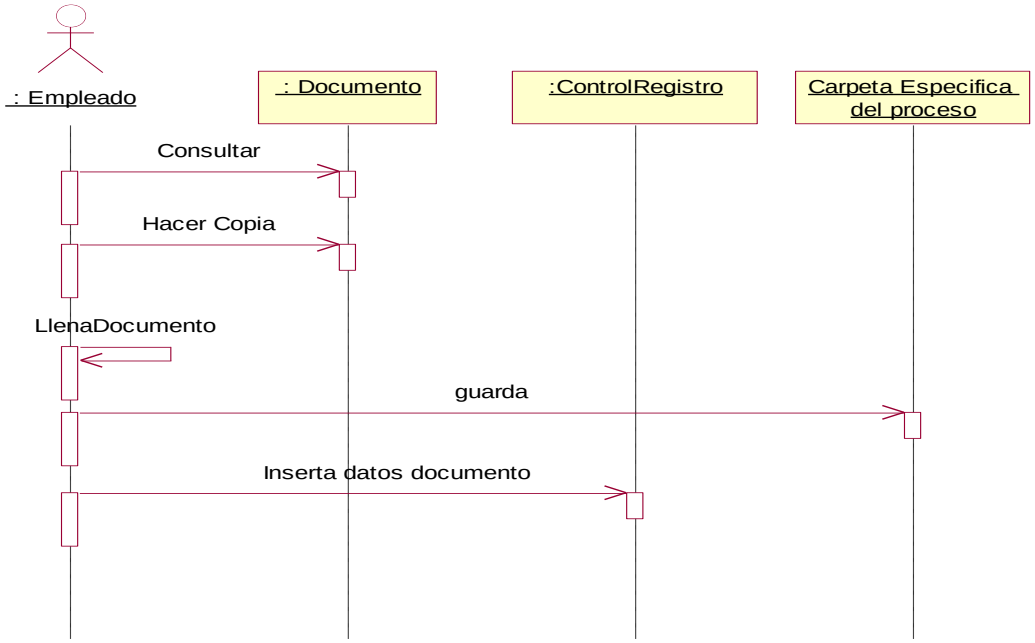


Figura 3.5 Realización de Proceso de Negocio ConsultarDocumento

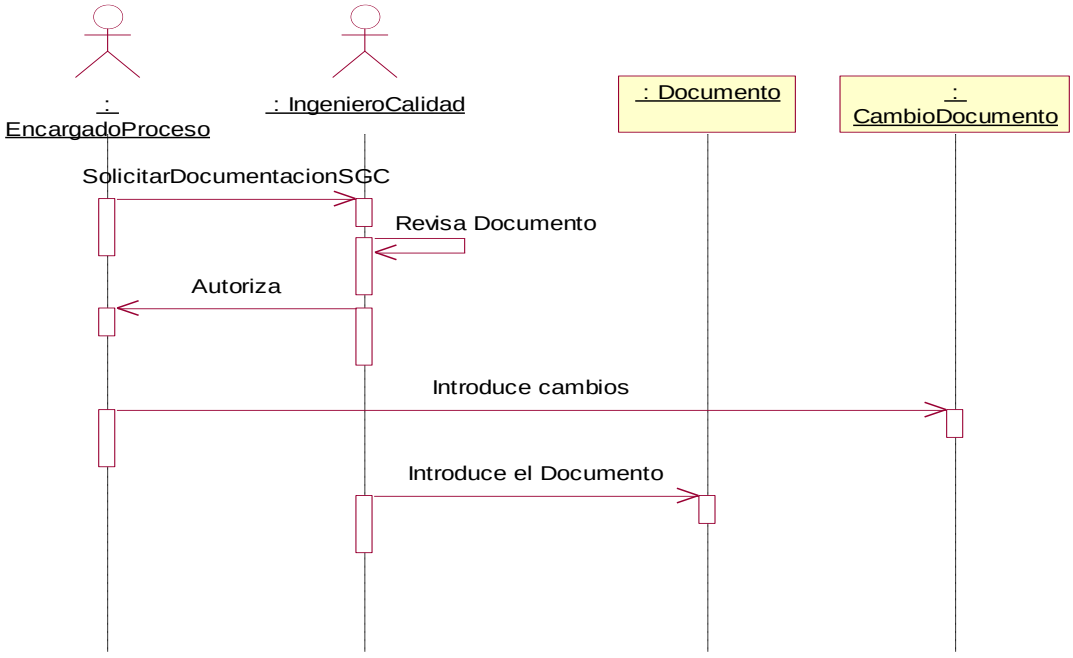


Figura 3.6 Este Proceso de Negocio SolicitarDocumentarSGC

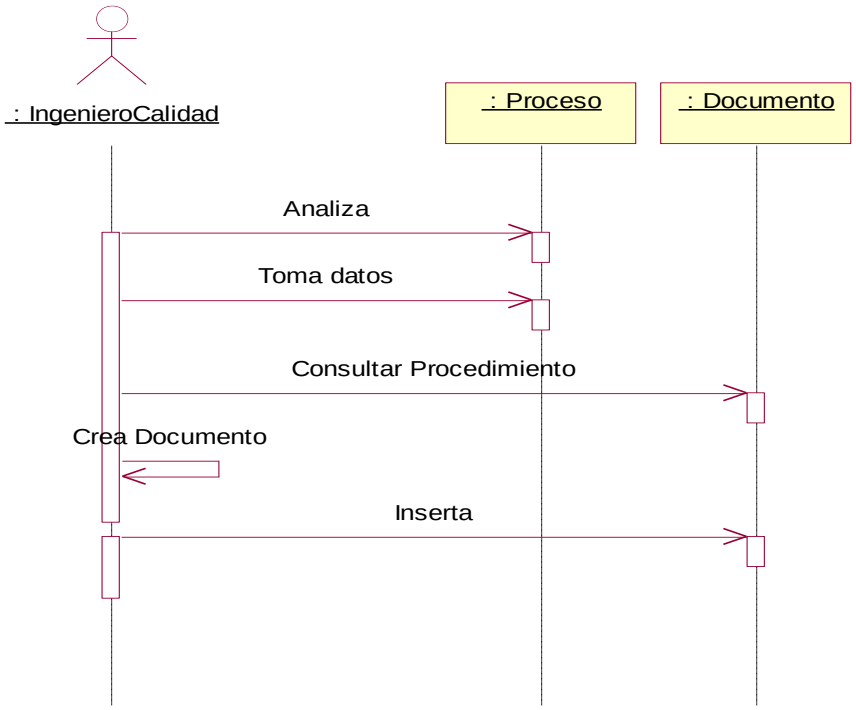


Figura 3.7 Realización Operación de Negocio DocumentarSGC

3.2.2 Modelo Conceptual

En este modelo se capturan los conceptos más importantes del contexto del sistema y los requerimientos que el cliente solicita. En la figura 3.8 se muestran los conceptos obtenidos del problema y sus respectivos atributos.

3.2.2.1 Descripción Modelo Conceptual: El Modelo conceptual se observa en la figura 3.9. Muchos usuarios pertenecen a muchos proyectos. Cada usuario hace un registro de su actividad en el sistema por medio de una Bitácora. Un proyecto tiene muchos procesos y estos a su vez muchos documentos. Los Procesos tienen varios documentos asociados mediante una ruta. Un documento tiene varios cambios y varios registros.

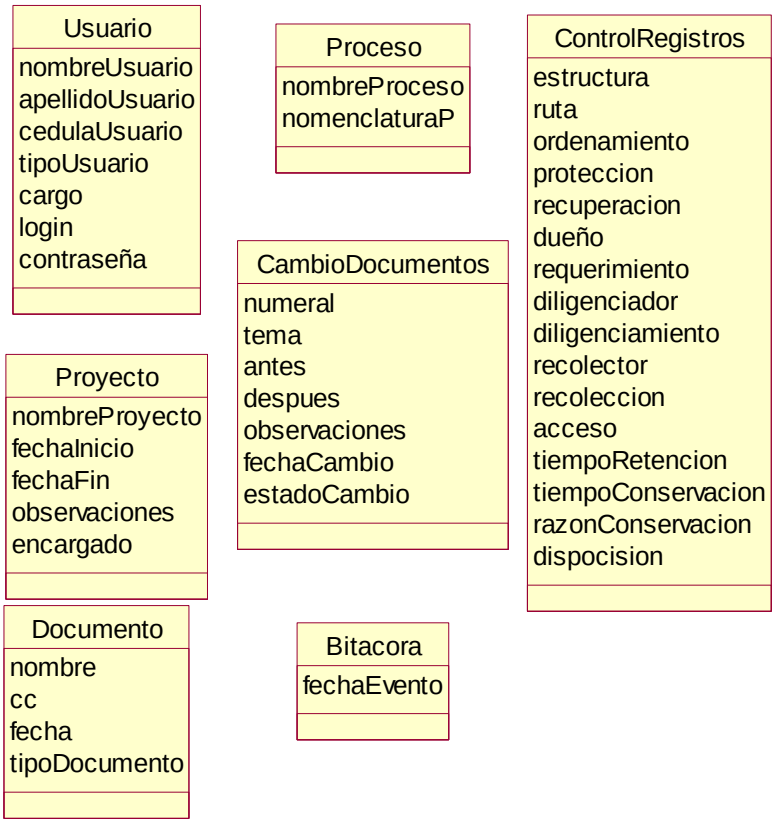


Figura 3.8 Conceptos Sistema de Gestión Documental GAMMA ING. S.A.

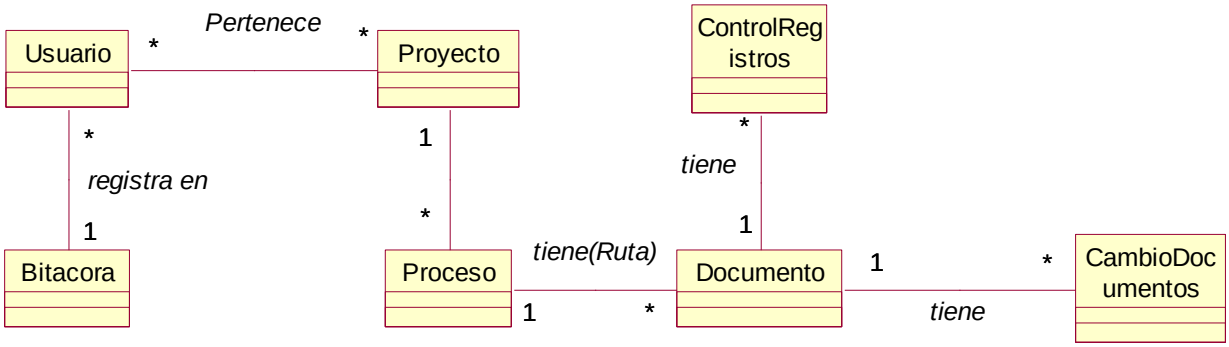


Figura 3.9 Modelo Conceptual

3.2.3 Lista de Funciones del Sistema-Árbol de Funciones

A continuación, se realiza la lista de las funcionalidades que el sistema debe tener. Estas funcionalidades deben estar completas para el desarrollo del PIM así que, se hizo indispensable hacer varias reuniones con el cliente.

REFERENCIA	FUNCION	CATEGORIA
R.1	**GESTIONAR PROYECTOS**	
R.1.1	Crear Proyecto	Evidente
R1.2	Modificar Proyecto	Evidente
R.1.3	Eliminar Proyecto	Evidente
R.1.4	Ver Información Proyecto	Evidente
R.1.5	Crear Proceso	Oculto
R.1.8	Crear Carpeta	Evidente
R.1.9	Modificar Carpeta	Evidente
R.1.10	Eliminar Carpeta	Evidente
R.1.11	Buscar Proyecto	Evidente
R.2	**GESTIONAR DOCUMENTOS**	
R.2.1	Insertar documento	Evidente
R.2.2	Buscar Documento	Evidente
R.2.3	Ver Documento	Evidente
R.2.4	Eliminar Documento	Evidente
R.2.5	Crear Documento	Evidente
R.2.6	Ver Estado Documento	Evidente
R.2.8	Registrar Estado Documento	Oculto
R.2.9	Ver Registros	Evidente
R.2.10	Crear Registros	Evidente
R.2.11	Ver Cambios Documentos	Evidente
	GESTIONAR USUARIOS	
R.3.1	Registrar Usuario	Evidente
R.3.2	Modificar Usuario	Evidente
R.3.3	Eliminar Usuario	Evidente
R.3.4	Buscar Usuario	Evidente
R.3.5	Ver Usuario	Evidente
R.3.6	Validar Usuario	Evidente
R.3.7	Registrar Actividades Usuario	Oculto
R.3.8	Listar Actividades Usuario	Evidente

Tabla 3.1 Lista de Funcionalidades del Sistema

3.2.4 Requisitos No-Funcionales

Estos son los requisitos que no se modelan pero que tienen mucha importancia para el sistema en el momento de tomar decisiones tanto arquitecturales como de diseño. En la tabla 3.2 se puede observar lista de características no funcionales del sistema. Estos requisitos se obtuvieron de las reuniones con el cliente y de la asesoría que se brinda a la empresa por medio de la experiencia obtenida por los desarroladores.

Características	Descripción	Funciones Afectadas	Obligatoria/opcional
Sistema Operativo	Windows 98/me/NT/XP	Todas	Opcional
Lenguaje de Programación	Java, jsp, HTML	Todas	Obligatoria
Sistema de gestión de bases de datos	FireBird	Todas	Opcional
Interfaz de Usuario	Interfaz Web 256 Colores	Todas menos las ocultas	Obligatoria

Tabla 3.2 Lista de Características no funcionales

Otras características no funcionales son los atributos de calidad, estos son muy importantes ya que hacen que el sistema responda correctamente a las necesidades tanto del cliente como del desarrollador.

Los atributos de calidad generalmente están distribuidos en todos los componentes arquitectónicos, pero pueden adquirirse a medida que se tomen las decisiones de la tecnología que se quiere implementar. Estos atributos son la información adicional para nuestros próximos modelos, lo que va a permitir tomar decisiones como la selección de patrones arquitecturales y patrones de diseño. A continuación se lista el conjunto de atributos de calidad que se deben tener en cuenta a lo largo del desarrollo del proyecto y además se muestra un ejemplo de escenario de calidad el cual le ayuda al desarrollador a obtener los atributos de calidad como se observa en el primer capítulo.

Atributo de Calidad	Descripción
Distribución	Poder colocar la lógica del negocio separada de los datos.
Flexibilidad	Sistema base para la creación de otras aplicaciones.
Escalabilidad	El sistema debe permitir variaciones capacidad sin la intervención de nadie. Debe tener la opción de agregar mas funcionalidades al sistema.
Disponibilidad	El sistema debe estar disponible 24 horas 7 días a la semana si es posible.
Seguridad	El sistema debe autentificar a los usuarios y proteger sus datos contra accesos desautorizados.
Usabilidad	Diferentes usuarios deben acceder a diferente contenido en diferentes formas. Interfaz intuitiva.
Desempeño	El sistema debe responder al usuario ante algún evento.

Portabilidad	El sistema puede funcionar sobre Windows o Linux.
Modificabilidad	Poder cambiar las respuestas del sistema sin tener que cambiar la interfaz Visual. Poder hacer cambios futuros a la aplicación sin efectos drásticos.
Interoperabilidad	Para próximas conexiones con tecnologías como CORBA.

Tabla 3.5 Atributos de Calidad

3.2.4.1 Ejemplo de Escenario de Calidad: Uno de los atributos más importantes es la disponibilidad del sistema, ya que este debe estar funcionando 24/7, pero sabemos que esto es imposible, ya que existen factores tanto internos como externos al sistema que lo afectan, llevándolo a fallas. En el capitulo uno se describieron los escenarios de calidad y se especificó la descripción de cada atributo de calidad. A continuación se muestra un ejemplo del escenario de Calidad de disponibilidad propuesto por [ARQPRA].

Escenario de Calidad de **DISPONIBILIDAD**

ESTIMULO: Externo al Sistema, dependiendo del estimulo el sistema responde de una u otra forma.

FUENTE: Tiempo. Un componente responde al fallo pero la respuesta se adelanta o se retraza.

ARTEFACTO: Procesador del sistema, canales de comunicación. Este es el recurso que debe tener la más alta disponibilidad si se produce la falla.

AMBIENTE: Normal operación. El estado del sistema puede afectar la respuesta del sistema cuando ocurre la falla.

RESPUESTA: El sistema debe notificar de la inesperada falla por medio de un mensaje y continuar operando normalmente.

MEDIDA DE RESPUESTA:

$$\alpha = \frac{\text{Tiempo antes de falla}}{\text{Tiempo antes de falla + Tiempo antes de reparación}} = 99.997\%$$

Esta medida puede ser un porcentaje, o puede ser un intervalo de tiempo de la falla o del tiempo de reparación, en este caso es un porcentaje.

Después de haber definido el contexto del sistema, junto al modelo conceptual y los atributos de calidad, que dieron las principales pautas para la elección del patrón arquitectural, el CIM tiene la información necesaria para ser transformado al PIM.

3.3 DESARROLLO DEL PIM

Para llegar a construir el PIM (Modelo independiente de la plataforma) debemos hacer un análisis muy completo del sistema que se va a desarrollar. El PIM debe ser más que un modelo conceptual como algunos autores lo toman, un modelo que muestre información tanto estática como dinámica de las funcionalidades de la aplicación. El UP en este caso aporta en la consecución de este objetivo con los artefactos tradicionales, como los casos de uso, las clases y paquetes de análisis y los diagramas de colaboración y secuencia los cuales nos muestran la parte dinámica del sistema. En la siguiente figura se puede observar la generación del PIM a través de los artefactos UP.

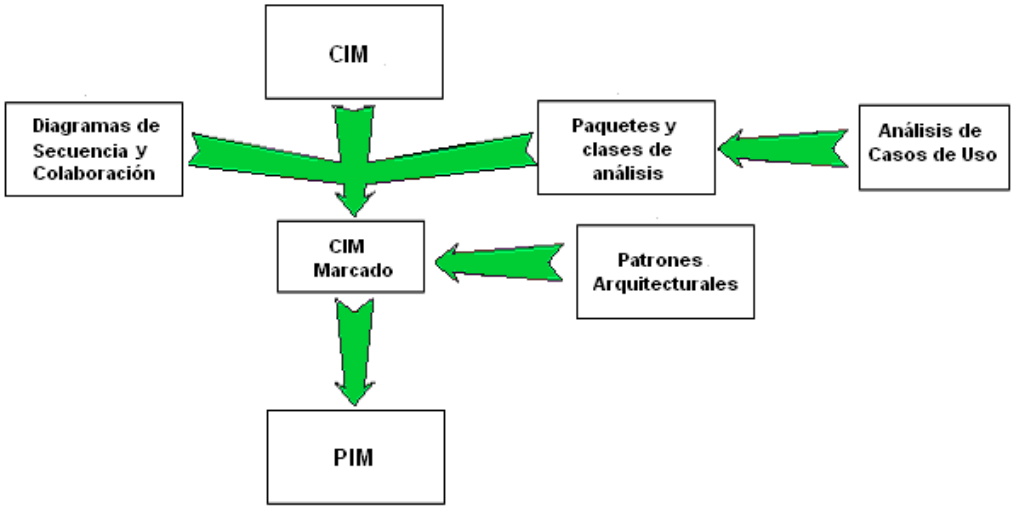


Figura 3.10 Modelo de transformación del CIM al PIM

3.3.1 El patrón Arquitectural

Los patrones arquitecturales definen la estructura organizacional que va a llevar el sistema. Provee un grupo definido de subsistemas, especifica sus responsabilidades, e incluye algunas reglas o lineamientos para organizar las relaciones entre estos además de resolver no todos pero si algunos de los mas importantes requisitos de calidad[4].

[4] http://c2.com/cgi/wiki-MultiTierArchitecture

Para el desarrollo del sistema de Gestión Documental se tomó el patrón de capas, el cual se basa en la división del sistema no solo en la capa de base de datos y de cliente, si no en múltiples capas las cuales hacen que los elementos en una capa envíen y reciban información de las capas adyacentes que usualmente son componentes J2EE, objetos CORBA, entre otros. Estos envían la información entre los distintos niveles para ser posteriormente procesada y dependiendo del caso devuelven la información correspondiente al cliente.

Algunas de las ventajas que ofrece este patrón son:

- Reusabilidad de las capas, debido al uso de componentes. Al presentar una alta independencia, pueden algunas capas ser usadas en otros sistemas.
- El mantenimiento es mucho más fácil ya que los niveles de lógica del negocio pueden ser modificados sin necesidad de afectar a los clientes.
- El número de capas no esta definido, así que si el sistema lo necesita puede crecer a cuantos niveles sea necesario, además de presentar facilidad en el crecimiento de las funcionalidades.

Sin embargo el modelo en capas presenta algunos inconvenientes, si es usado de forma incorrecta:

- Para sistemas pequeños de poco crecimiento, puede significar trabajo innecesario.
- Definir la granularidad de los paquetes de información que se intercambian entre capas es un factor crítico que muchas veces perjudica el desempeño del sistema.

Teniendo el CIM y la información del patrón de arquitectura, se da paso a la segunda etapa de captura de requerimientos por medio de los diagramas de casos de uso donde se especifican las funcionalidades del sistema. Lo que se pretende con el PIM es llegar a un diagrama de clases hecho en UML con características definidas que me permitirán la conversión a los diferentes PSM.

3.3.2 Modelado y Análisis de Casos de Uso

Los casos de uso del sistema se toman de la lista de funcionalidades del primer flujo de trabajo (Especificación):

Caso de Uso Gestión Proyectos: Como los documentos se tienen que almacenar por proyectos, los cuales tienen procesos, el administrador tiene la opción crear, eliminar, modificar y buscar los proyectos.

Caso de Uso Gestión Documentos: Este caso de uso permite tener control sobre los documentos del Sistema de Gestión de Calidad (Documentos y registros), permitiendo al administrador crear, eliminar, modificar y buscar los documentos.

Caso de Uso Gestión Usuarios: Los perfiles de los usuarios y los permisos de admisión al sistema deben ser gestionados por el administrador del sistema.

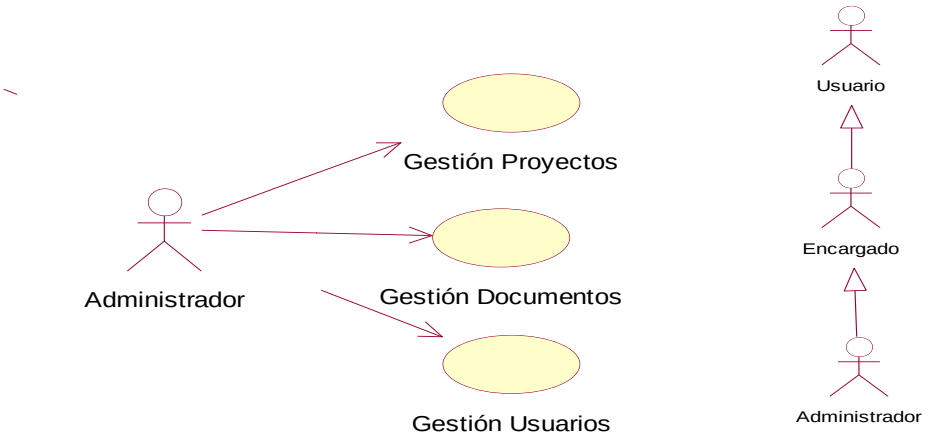


Figura 3.11 Casos de Uso del Sistema y Jerarquía de Roles

En la jerarquía, el Administrador del sistema hereda las operaciones del Encargado que a su vez hereda las del Usuario, así el Administrador puede realizar todo lo que el usuario y el encargado hacen. Para ver más detalle de las funcionalidades del sistema se tiene el diagrama de Casos de Uso extendido:

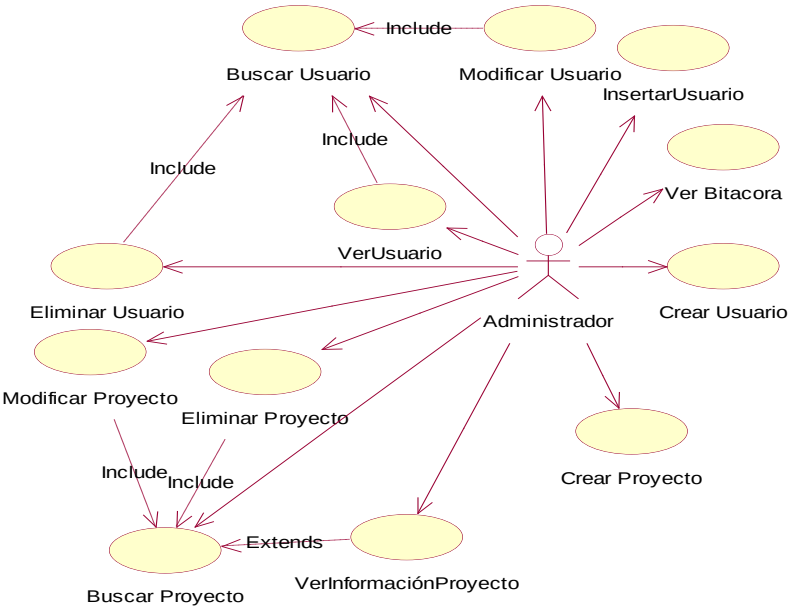


Figura 3.12 Diagrama Casos de Uso Extendidos 1/2

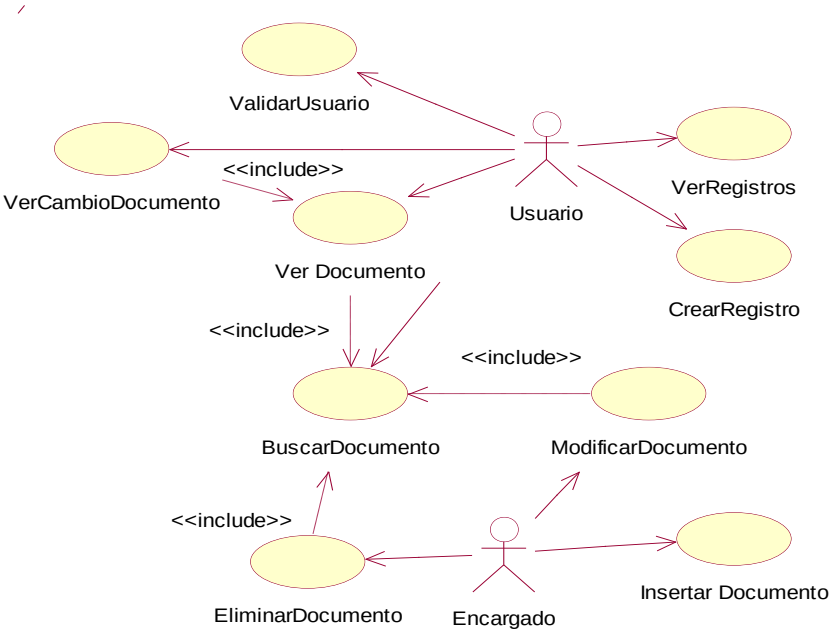


Figura 3.12 Diagrama Casos de Uso Extendidos 2/2

Teniendo el diagrama de los casos de uso del sistema, se procede a realizar la descripción detallada de cada uno de estos lo cual proporciona la información necesaria para la parte de análisis, en donde se genera el PIM. En este trabajo de grado se escoge uno de los casos de uso como ejemplo para seguir todo el proceso de transformación, el caso de uso Eliminar Proyecto.
Se escogen los casos de uso reales porque brindan más información que los de análisis, necesaria para la realización de los modelos.
Para nombrar las interfaces de administrador, se coloca el prefijo "IA"

3.3.2.1 Caso de Uso Eliminar Proyecto

ID Caso de Uso	**CU8**
Nombre	ELIMINAR PROYECTO
Autor	Analista
Fecha de creación	25 de Febrero del 2004
Actor	Administrador
Descripción	Este caso de uso se inicia después que el administrador ha ejecutado una búsqueda de proyecto, se selecciona un proyecto y se despliega todas las subcarpetas que estén relacionadas con este.
Precondiciones	Sistema Activo, Sesión de Administrador iniciada, Caso de Uso Buscar Proyecto ejecutado
Postcondiciones	Interfaz de confirmación.
Prioridad	Media
Flujo normal de eventos	

Administrador	**Sistema**
1. Selecciona uno de los proyectos que arrojo la búsqueda en la interfaz IAResulatdoBusquedaProyecto y hace clic en eliminar	
	2. Devuelve la interfaz IAEliminarProyecto con la información del proyecto así como explorador para elegir las carpetas a eliminar La elección de las carpetas se hace por medio de un radio botón. Contiene los botones Aceptar y "cancelar".
3. Selecciona los procesos que va a eliminar o el proyecto si así lo desea y presiona aceptar o cancelar.	
	4. Retorna un mensaje de precaución con un botón aceptar y otro cancelar

	5. Acepta la eliminación	
		6.Retorna IAExitoEliminarProyecto y se le da la opción al usuario de "volver al menú principal" o de "realizar una nueva búsqueda"
Flujos Alternativos	En el paso 3 - El administrador cancela la operación al hacer clic en cancelar y vuelve a IAMenuPrincipal En 4 podría cancelar la operación de eliminación. Se muestra de nuevo la interfaz IAMenuPrinciapal	
Excepciones	En el paso 1 – El administrador no selecciono ningún usuario así que se emite una alarma de notificación. En el paso 1 – Falla de conexión con el servidor de aplicación o de BD, se notifica al usuario en la IAEliminarProyectoErrorPage En el paso 3 – Falla de conexión con el servidor de aplicación o de BD, se notifica al usuario en la IAExitoEliminarProyectoErrorPage	
Incluye		
Puntos de extensión		
GUIS Relacionadas	IAMenuPrincipal IAResultadoBusqueda IAEliminarProyecto IAExitoEliminarProyecto IAEliminarProyectoErrorPage	

Esta descripción se hace para cada caso de uso del sistema. De lo anterior se deben crear el diagrama de paquetes de análisis, las clases de análisis, los diagramas de secuencia y el de colaboración.

3.3.3 Diagrama de Paquetes de análisis y Clases de Análisis

Como ya se ha definido con anterioridad un patrón arquitectural Multi-capas, se realiza el diagrama de paquetes de acuerdo a las tres capas que tenemos para el sistema: capa de presentación, capa de lógica y capa de Acceso.

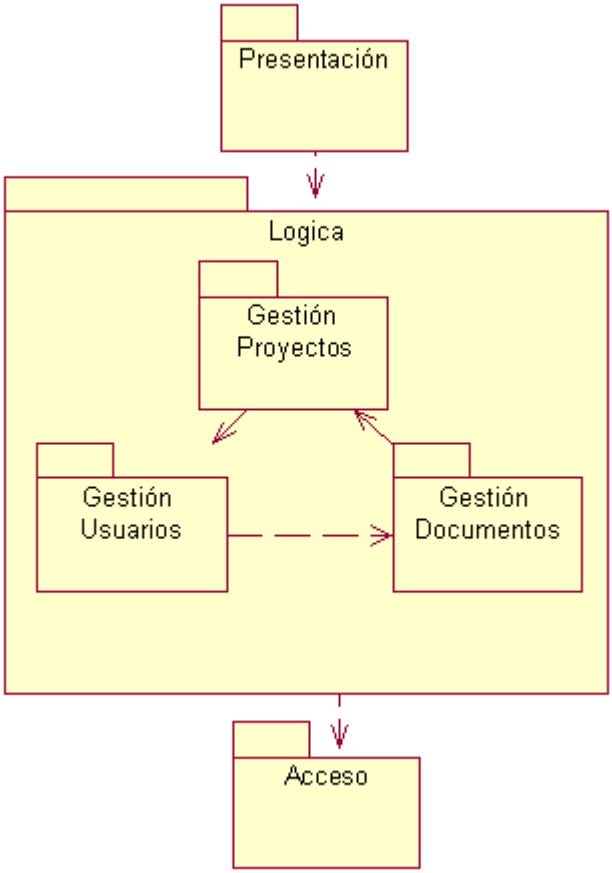


Figura 3.13 Diagrama de paquetes de análisis

El diagrama de la figura 3.13 está compuesto por un paquete de presentación, en el cual se encuentran las interfaces que se van a presentar al usuario. En el paquete de lógica se incluyen los paquetes de gestión los cuales a su vez están compuestos de todas las clases que gestionan o controlan los documentos, los proyectos y los usuarios. El paquete de acceso agrupa todas las clases que tiene que ver con el acceso a la información.

Siguiendo el análisis de los casos de uso llegamos a la construcción del diagrama de Clases.

- Si la clase se extrae del paquete de <NombrePaquete> llevará la etiqueta <<NombrePaquete>>.
- Por cada paquete en la capa de lógica se crea una clase de análisis con el nombre del paquete.

- Por cada caso de uso se crea una o varias clases de tipo <<Presentación>> de acuerdo a la necesidad que se cree en el caso de uso extendido.
- Las relaciones de las clases de análisis se hacen de acuerdo a las descripciones de los casos de uso extendidos (reales). (Ver figuras 3.14 y 3.15)

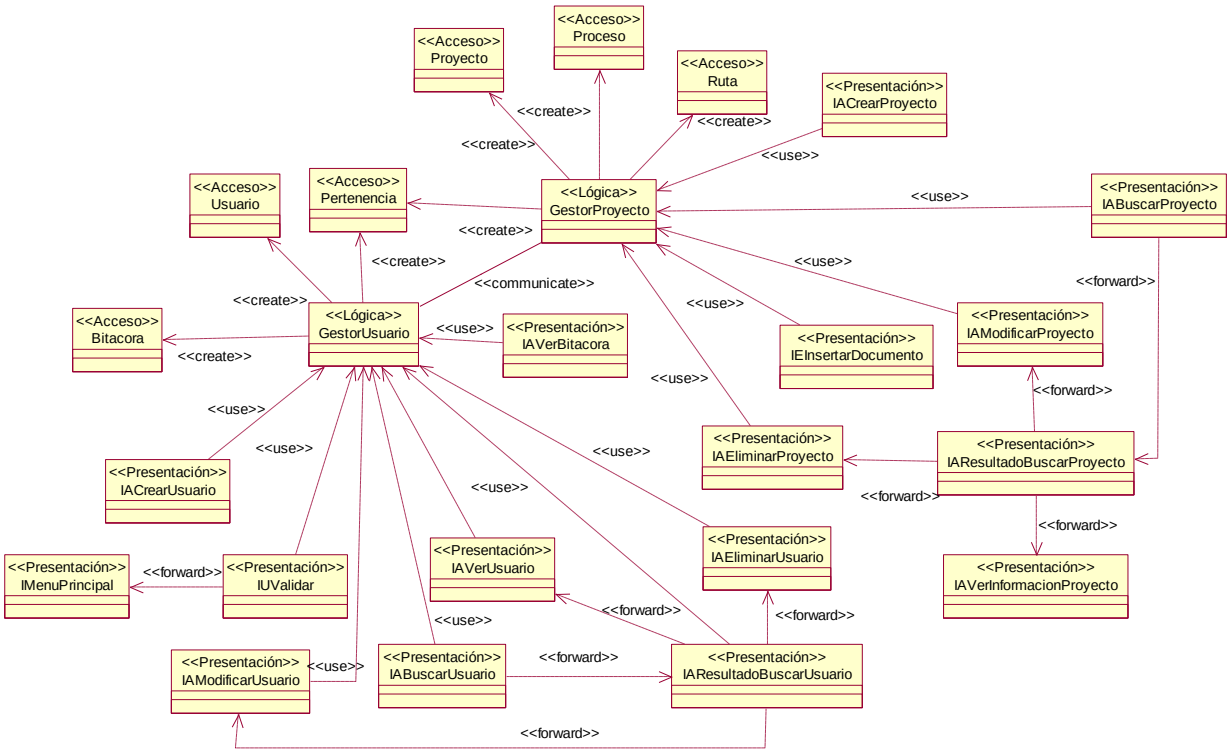


Figura 3.14 Diagramas de clases de análisis 1/2

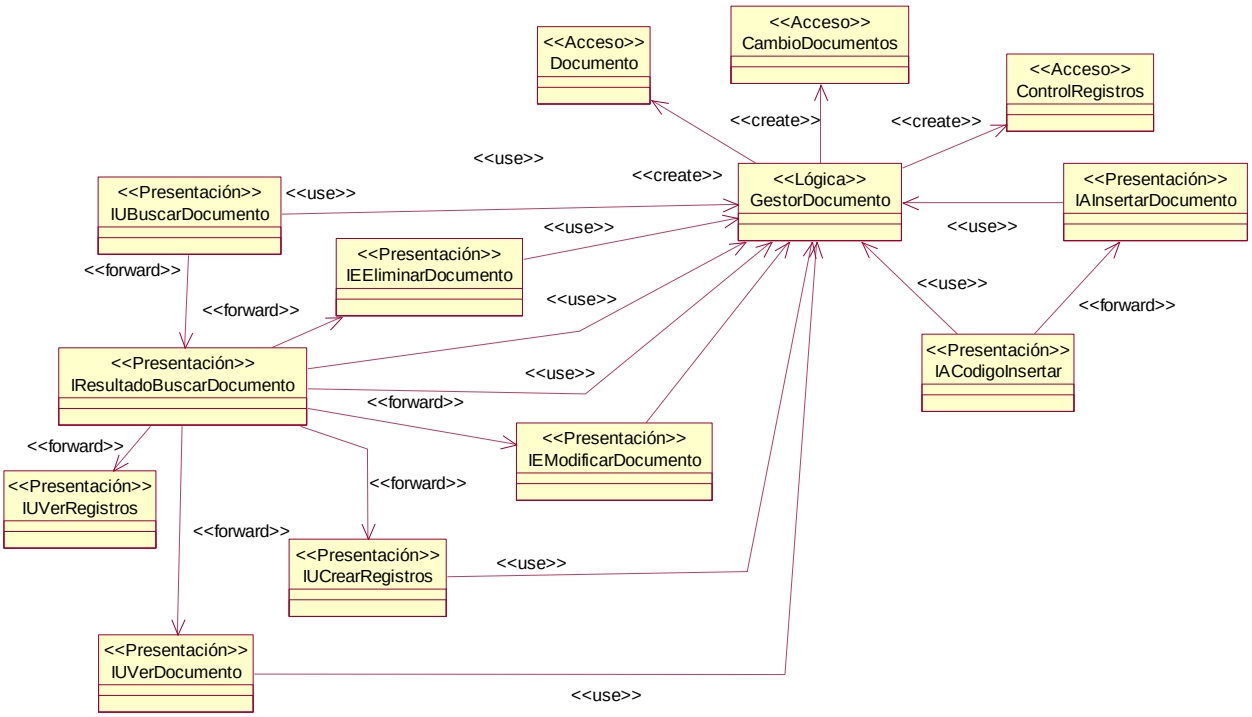


Figura 3.15 Diagrama de clases de análisis 2/2

Para diferenciar las clases de análisis de tipo presentación, se coloca prefijo “I” en el nombre de la clase, seguido por “A” si es una interfaz de Administrador, “E” si es interfaz de Encargado o “U”si la interfaz es de Usuario.

3.3.4 Diagramas de secuencia

Los diagramas de secuencia ayudan a tener en cuenta los aspectos dinámicos del sistema. Y es aquí donde se insertan los conceptos del patrón de arquitectura que seleccionamos, estos conceptos serán las “Marcas” que colocaremos en el PIM para la transformación. El concepto de entidad, control e interfaz se representan de esta manera:

Figura 3.16 Conceptos del Patrón Arquitectural Modelo-Vista-Control

Como se puede observar en la figura 3.17 y 3.18 se realizan los correspondientes diagramas de secuencia y colaboración, teniendo en cuenta la descripción de casos de uso y las representaciones del estilo arquitectural.

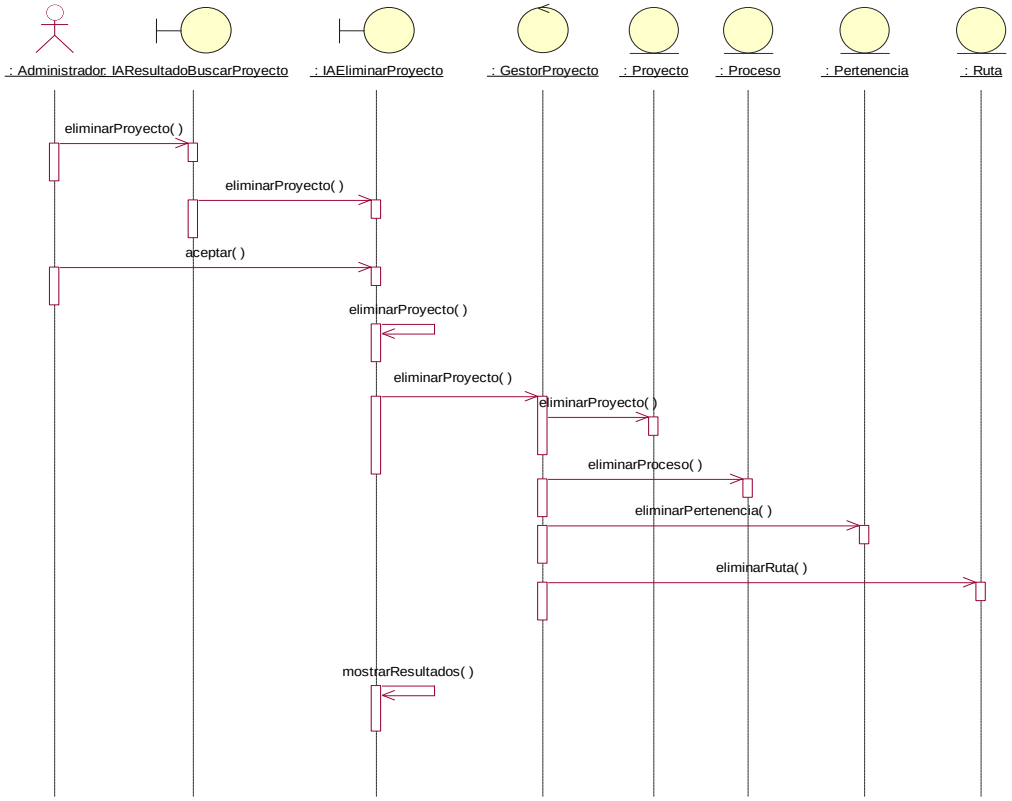


Figura 3.17 Diagrama de secuencia Caso de Uso Eliminar Proyecto

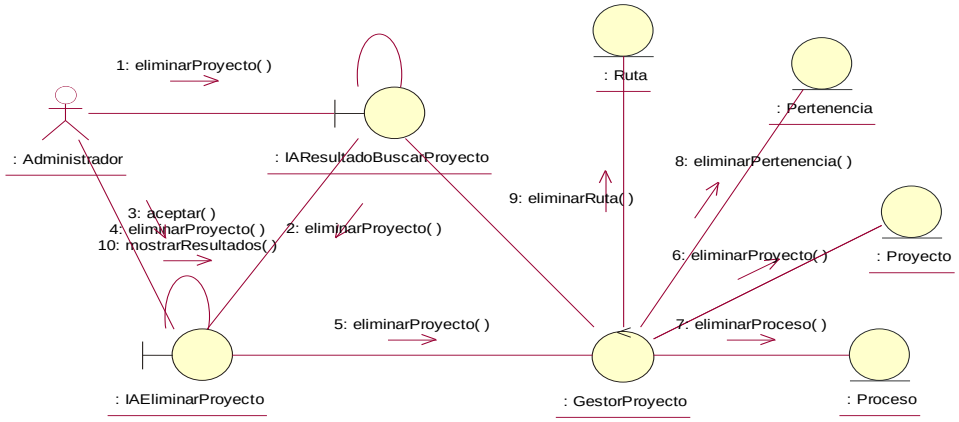


Figura 3.18 Diagrama de Colaboración Caso de Uso Eliminar Proyecto

3.3.5 Reglas de Transformación CIM a PIM

Después de definir los diagramas de paquetes y clases de análisis, los diagramas de secuencia y colaboración, el estilo arquitectural y el CIM, se deben aplicar las reglas de transformación para pasar al modelo PIM. No existen como tal reglas que se hayan especificado para los modelos MDA pero como se vio en el capitulo uno MDA se basa en especificaciones de [UML 2.0], XMI, MOF, OCL y modelos UML (Profile UML), de donde autores como los de [MDAEX] han realizado un primer acercamiento a lo que podrían ser estas normas. Basándonos en estas y en las que por experiencia a través del trabajo con el Sistema de Gestión Documental se han adquirido, se listan las reglas de transformación, que en modo de sugerencia, se dan para poder llevar a la conversión de los modelos.

a. Por cada concepto <NombreConcepto> del diagrama conceptual del CIM, habrá en el PIM una clase <NombreConcepto> de tipo entidad.
b. Por cada clase de análisis de tipo <<Lógica>> y <NombreClase> en el diagrama de clases de análisis, habrá en el PIM una clase <NombreClase> con el estereotipo <<control>>.
c. Por cada clase de tipo <<Presentación>> y de <NombreClaseInterfaz> en el diagrama de clases de análisis, habrá en el PIM una clase <NombreInterfaz> con el estereotipo <<interfaz>>.
d. Por cada asociación de conceptos en el diagrama conceptual del CIM, existe una asociación de las mismas clases en el PIM, conservando la misma multiplicidad.
e. Por cada asociación de multiplicidad muchos a muchos en el diagrama conceptual del CIM, habrá en el PIM una clase de tipo entidad <Nombre Asociación> relacionada con las dos clases entidad correspondientes, con una relación de muchos a uno.
f. Por cada atributo <NombreAtributo> de un concepto en el CIM habrá en el PIM un atributo <NombreAtributo> público en la clase correspondiente.
g. Por cada flujo de tipo <use> de una clase tipo interfaz hacia una clase tipo control en el diagrama de clases de análisis, habrá una relación de “use” en el PIM de la clase interfaz a la clase control correspondiente.
h. Por cada flujo de tipo <create> de una clase tipo control hacia una clase tipo entidad en el diagrama de clases de análisis, habrá en el PIM una relación “create” de la clase control a la clase entidad.
i. Por cada flujo de tipo <forward> de una clase tipo interfaz hacia una clase tipo interfaz en el diagrama de clase de análisis, habrá en el PIM una relación de “forward” de la clase interfaz inicial a la clase interfaz final.

j. En cada clase entidad de nombre <NombreEntidad> del PIM se crean por lo menos cuatro funciones de gestión básicas que son: crear <NombreEntidad>, buscar <NombreEntidad>, elimina r<NombreEntidad> y modificar <NombreEntidad>.

k. En cada clase Gestor de nombre <NombreGestor> del PIM habrá por lo menos cuatro funciones de gestión, relacionadas con las entidades con las cuales esta asociado directamente así: crear <NombreEntidad1>, buscar <NombreEntidad1>, eliminar <NombreEntidad1> y modificar <NombreEntidad1>, crear <NombreEntidad2>, buscar <NombreEntidad2>, etc.

l. Cada operación de los flujos del diagrama de secuencia, debe estar en el PIM, dentro de la clase que le corresponda, es decir la clase a la cual llega el flujo.

Después de haber aplicado las reglas de transformación que surgieron a través de la experiencia y análisis de los modelos, podemos ver que el PIM es más completo que un diagrama conceptual, como lo describen los libros. En la figura 3.19 se muestra el PIM del caso de uso eliminar Proyecto y en la figura 3.20 se muestra el PIM total, resultado de aplicar las transformaciones manualmente.

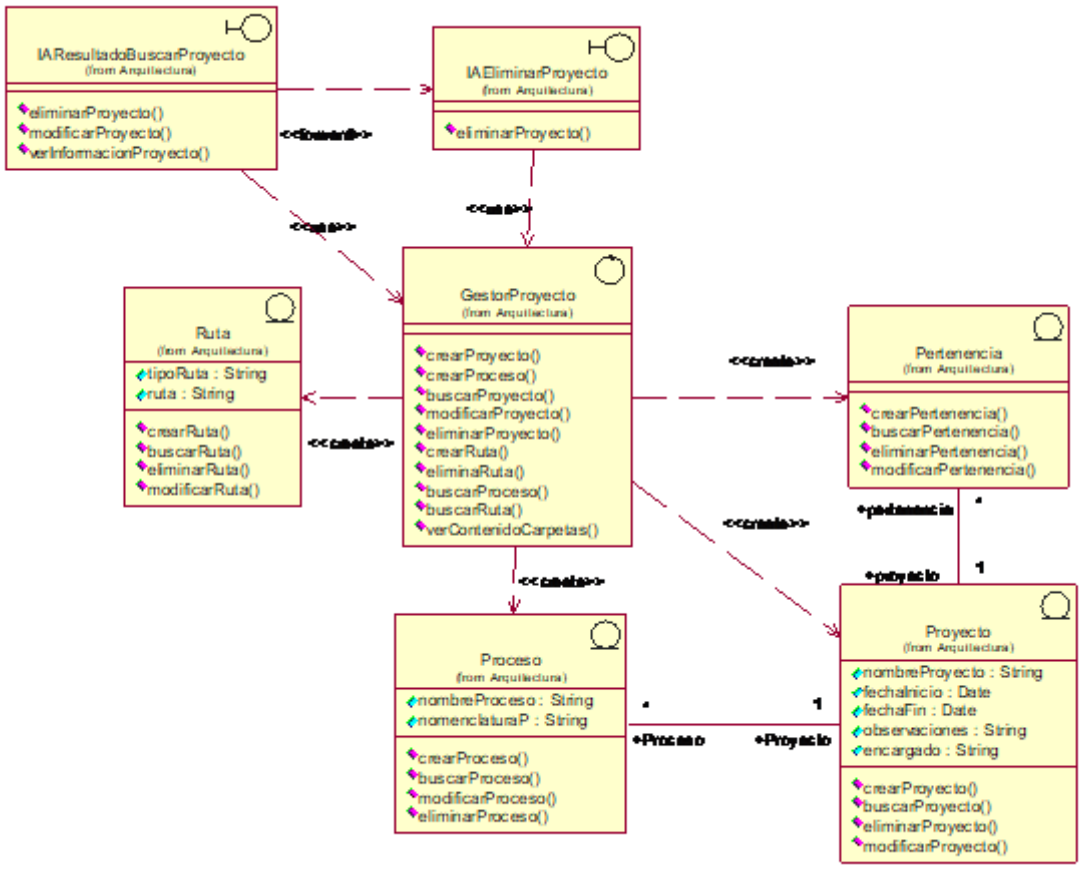


Figura 3.19 Modelo parcial PIM caso de Uso Eliminar Proyecto

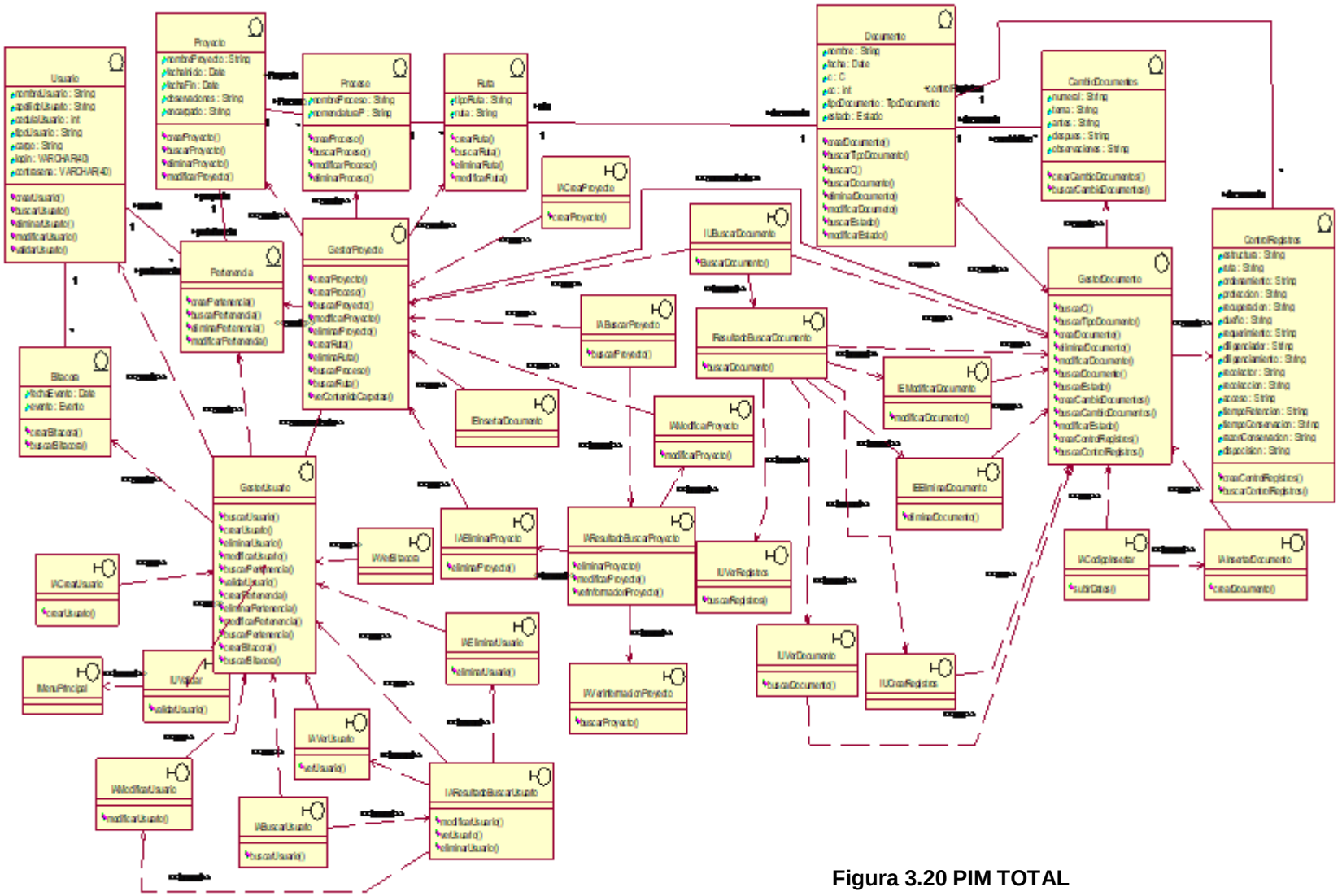


Figura 3.20 PIM TOTAL

3.4 DESARROLLO DEL PSM

Como vimos anteriormente, el PIM nos muestra un modelo del Sistema de Gestión Documental completamente independiente de la plataforma en la cual se va a trabajar. El PSM, es el siguiente modelo, en el cual se va a introducir información específica de la plataforma.
Como se vio en el segundo capitulo el PSM surge en el flujo de trabajo de diseño y para empezar a desarrollar este modelo es necesario tener en cuenta dos conceptos importantes los atributos de calidad y los patrones de diseño. Los atributos nos brindan las pautas para escoger a los patrones de diseño y la plataforma en la que se va a trabajar.
La siguiente figura nos muestra los artefactos UP con los cuales se puede obtener el PSM.

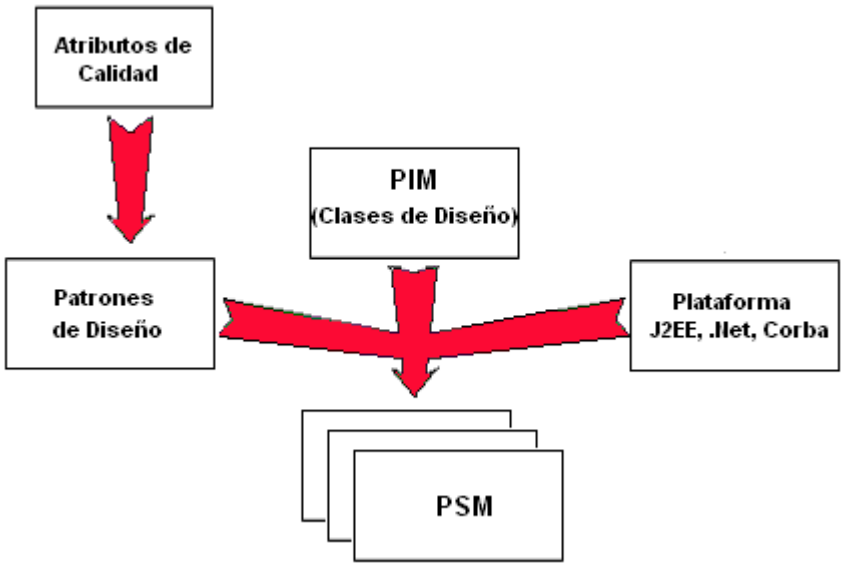


Figura 3.21 Transformación para el PSM a partir del PIM

Las tradicionales clases de diseño que son el equivalente del PIM se le aplican las transformaciones de MDA. Acompañado de la información de los atributos de calidad y los patrones de diseño el PIM se convierte en varios PSM según la tecnología que se utilice.

3.4.1 Plataforma J2EE

Antes de entrar a hablar de la plataforma [J2EE], es necesario comprender el concepto de componentes y aplicaciones distribuidas, estas definiciones se explican a continuación:

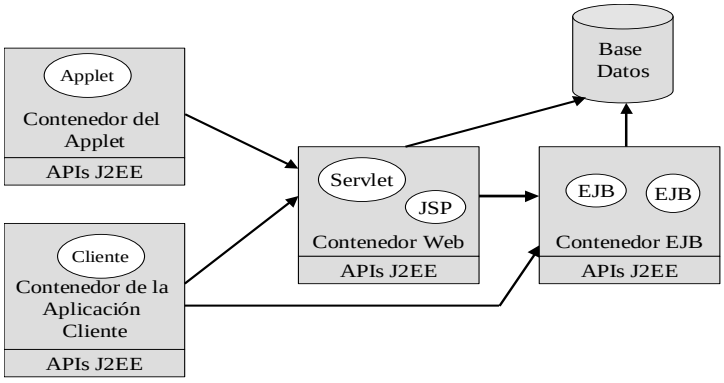
3.4.1.1 Plataformas para Aplicaciones Distribuidas: El desarrollo con componentes es una forma de trabajo que permite implementar software en un dispositivo físico o lógico que implementa interfaces con las cuales se comunica con otros componentes o aplicaciones. Esto permite un aislamiento casi total entre las diferentes partes de un sistema y simplifica la concepción de una aplicación.

Una gran ventaja de trabajar con componentes, es que estos pueden encontrarse distribuidos en la red y aun así poder comunicarse a través de protocolos como JNDI haciendo que puedan trabajar no en una si no en varias aplicaciones al mismo tiempo.

En el mercado existen varias tecnologías para trabajo en aplicaciones distribuidas como lo son .NET, CORBA y J2EE Estas poseen grandes diferencias en cuanto a desarrollo pero se mantiene la misma filosofía, dándole la oportunidad al cliente de tener sistemas altamente flexibles y escalables gracias al trabajo con componentes.

3.4.1.2 Plataforma J2EE: Las aplicaciones empresariales demandan servicios como procesamiento de transacciones, acceso a bases de datos, mensajería, entre otros, ya que la arquitectura de estas es mucho mas compleja y la información del cliente debe ser bien manejada.

J2EE brinda acceso a estos servicios a través de un elemento denominado contenedor. El contenedor es uno de los elementos software de la arquitectura de la plataforma J2EE, que en tiempo de ejecución permite que los clientes tengan acceso a todas estas funciones empresariales y alberga a una serie de componentes llamados Enterprise Java Beans (EJB).Estos componentes deben cumplir con ciertas características especiales que especifica la plataforma. Tanto el contenedor como los componentes EJB residen en el servidor de componentes. En la siguiente figura se puede ver como es la arquitectura típica de una aplicación J2EE.



3.22 Arquitectura de J2EE

Como se puede ver en la grafica, se tienen 4 contenedores, el contenedor Web, el contenedor EJB, el contenedor del Applet y el de la aplicación cliente, cada uno provee de acceso a los usuarios para que puedan disponer de los servicios. Estos usuarios son de dos tipos, usuarios Web que están en navegadores web y usuarios EJB, que a su vez pueden ser, componentes que se encuentran en el contenedor Web, o, componentes EJB que se encuentran corriendo en el mismo o en otro contenedor EJB. De esta forma cada uno de los contenedores puede estar en equipos distintos y funcionar perfectamente. Además de esto, J2EE brinda una gran ventaja que es "librar" al desarrollador de la configuración de operaciones que no tienen que ver con la lógica del sistema, como lo es la sincronización de los EJB ya que el contenedor lo hace automáticamente disminuyendo el trabajo, y agilizando los procesos de desarrollo y de implantación.

3.4.1.3 Características de J2EE: La base de J2EE, como se menciono anteriormente, son los componentes denominados EJB´s, es por esto que para traer más organización a la plataforma se dividen estos en tres tipos:

Componentes de Sesión: Estos componentes denominados EJB de sesión se encargan de hacer funciones que no tienen persistencia en la aplicación, por ejemplo imprimir en pantalla un texto, o sumar dos números. Los componentes de sesión poseen la lógica para hacerlo. En las aplicaciones empresariales son los que encapsulan la lógica del negocio y son accedidos por otros componentes de sesión o por los clientes web.

Componentes de Persistencia: Estos componentes denominados EJB de entidad son los encargados de manejar la base de datos en las aplicaciones software, ya que tienen la función de insertar, modificar y consultar información de la base de datos. Este tipo de componentes es manejado principalmente por otros componentes, ya que no se recomienda que sea usado por clientes directamente.

Componentes de mensajes: Estos componentes denominados EJB de mensajes son diferentes a los demás componentes ya que no tienen interfaces (una de las principales características de los componentes distribuidos) y sirven para trabajar asincrónicamente, es decir que se activan cuando reciben una orden, empezando así un proceso, o haciendo llamadas a otros componentes (ya sea componentes de persistencia o de sesión).

Además de esto J2EE brinda una gran ventaja que es el uso de patrones de diseño. Estos patrones dan solución a un problema específico y al ser utilizados, hacen que la aplicación software mejore su desempeño, o su escalabilidad o su organización entre otros aspectos.

Para el sistema de Gestión de Calidad se utilizará la plataforma [J2EE]. Esta plataforma me permite cumplir con ciertos atributos de calidad que el cliente requiere.

a. Portabilidad: Es una plataforma hecha en Java, lo cual permite instalar la aplicación en cualquier sistema operativo.
b. Flexibilidad: Permite la inserción de componentes sin afectar el resto del código.
c. Interoperabilidad: En J2EE existen puentes de conexión con otras tecnologías como CORBA, XML, SOAP, entre otras.

3.4.2 Patrón de Diseño

Un patrón de diseño es una descripción de la forma de solucionar un problema recurrente de diseño, haciendo énfasis en el contexto, las fuerzas alrededor del problema y en el impacto y las consecuencias de la solución[5]

En este trabajo de grado se utilizarán tres patrones de diseño: El patrón de fachada, el controlador frontal y el patrón de objeto valor. Estos patrones se tomaron ya que ayudan en a mejorar la organización de la aplicación así como el desempeño de esta. Por otro lado, estos patrones fueron tomados especialmente para poder cumplir con los requisitos de calidad planteados para el sistema.

3.4.2.1 Controlador Frontal: (Front Controller): La capa de presentación, basa su funcionamiento en peticiones que se realizan a las capas adyacentes, estas peticiones deben controlarse y procesarse para cada usuario que las ejecuta. El control de estas peticiones puede hacerse de forma centralizada o distribuida. La mejor manera de llevar cabo este control es de una manera centralizada debido a que si no se hace de esta forma cada uno de los objetos de presentación (en este caso en particular las paginas JSP) que ejecuta peticiones, deberá encargarse de procesarlas, incurriendo así, en duplicación de código ya que muchos elementos Web deben hacer funciones similares. Otro inconveniente que se tiene es el de mezclar código de contenido, de control, de navegación y de presentación haciendo mas difícil el entendimiento de estos elementos.

[5] Definición de **JavaTM BluePrints**

El elemento que se utilizó como controlador frontal en este caso es un servlet, encargándose así del manejo de errores, validación e interacción con otros elementos de control que pueden hacer parte de la capa de presentación por un elemento que no interactúa con la presentación de la información. Sin embargo el uso de un controlador frontal no se restringe a utilizar solo un elemento, pueden existir varios elementos y cada uno de ellos gestionar un grupo de componentes Web (en este caso paginas JSP). La forma como trabaja el patrón de control frontal se muestra en la figura siguiente:

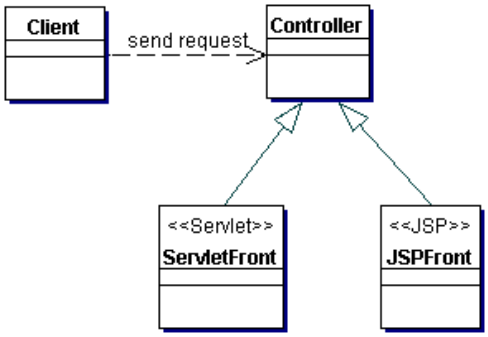


Figura 3.23 Diagrama de clases del Controlador Frontal

Como se ve en la figura, el controlador frontal no debe ser necesariamente un servlet, pero si debe encargarse de un grupo de elementos que necesiten hacer peticiones a otras capas.

3.4.2.2 Fachada: (Session Facade): Reducir un sistema complejo en subsistemas es una buena forma de enfrentar un problema, en el desarrollo software, mas específicamente en J2EE esto es lo que se hace al dividir el sistema en capas y asignar una funcionalidad a cada una de estas. Sin embargo cuando se tienen muchos elementos comunicándose entre si, los niveles pueden llegar a confundirse debido a que hay una gran dependencia entre ellos y se puede volver al problema que una vez se quiso solucionar, para esto es necesario utilizar el patrón de fachada, que permite que exista un bajo acople entre niveles haciendo que una posible expansión del sistema sea mucho mas fácil de lograr que si se tienen múltiples conexiones. El patrón de fachada se encarga de recibir todas las peticiones y reenviarlas al MiddleWare de una forma mas organizada como se ve en la figura 3.24.

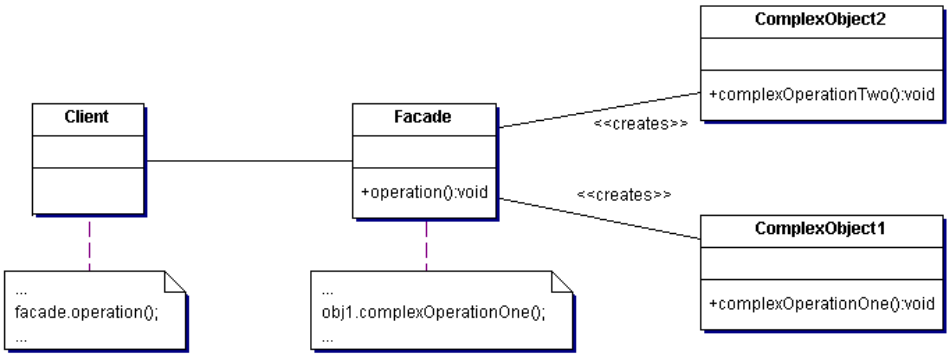


Figura 3.24 Patrón Fachada

Sin embargo, se debe tener en cuenta que al implementar el patrón de fachada el camino que deben recorrer las peticiones se incrementa, esto hace que el tiempo de respuesta de una petición aumente y por tanto disminuya el desempeño de la aplicación, es por esto que se debe ser conciente de las repercusiones que trae este patrón y determinar si el sistema que se esta desarrollando en realidad lo necesita o no.

3.4.2.3 Objeto Valor: (Value Object): En muchos casos se tiene que un cliente solicita información de una entidad y ejecuta métodos “get” (Específicamente cuando se trabaja con J2EE) para obtener la información. El problema que surge es cuando un cliente solicita muchos resultados de una tabla y ejecuta muchos get para traer los datos, esto hace que los encabezados que se generan sean demasiados y puedan congestionar la red y en un sistema que tenga muchos usuarios haciendo muchas consultas el desempeño de la aplicación se vera seriamente afectado. Para solucionar este problema se implementa el patrón de Objeto Valor, (también llamado objeto de transferencia) este patrón permite que la información que se obtiene de una consulta a una entidad sea empaquetada y enviada al cliente y se aloje en la RAM de su equipo permitiendo que los get que se ejecuten no afecten la red y no influyan en otras peticiones que pueden hacer otros usuarios.

Una vez mas, el usuario debe ser conciente de los requerimientos de calidad del sistema ya que al implementar este patrón para aplicaciones pequeñas que no lo necesitan, incurrirán en la creación de código innecesario que no afectara la aplicación, pero sí el tiempo del proceso de desarrollo.

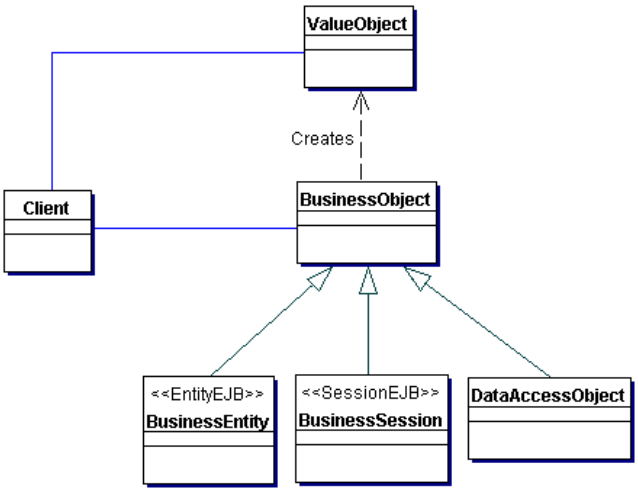


Figura 3.25 Patrón Objeto Valor

Además de los patrones que se utilizaron, existen otros que se proponen para mejorar el desempeño de las aplicaciones o para mejorar la organización del código en las capas de presentación, negocio e integración, a continuación se muestra una lista de los patrones de diseño que existen, pero no fueron utilizados, con una breve descripción[6]:

1. **Filtro de Decoración:** (Decorating Filter) Define nuevas responsabilidades a las peticiones o respuestas que pasan el controlador frontal.
2. **Ayudante de Presentación:** (View Helper) Ayuda a encapsular la lógica relacionada con la obtención de la información, validación y formato de esta, así el componente de presentación se limita a adaptar la forma en que la información será presentada.
3. **Vista Compuesta:** (Composite View) Toma múltiple vistas trozos de presentación ya sean estáticos o dinámicos y los fusiona para crear un solo template.
4. **Delegado de Negocio:** (Business Delegate) Reduce el acoplamiento entre capas, y provee un punto de acceso a los servicios por parte de otras capas.
5. **Servicio al Trabajador y Despachador de Vista:** (Service to Worker - Dispatcher View) Estos son la unión de los patrones Controlador Frontal, Vista Compuesta y Ayudante de Presentación.
6. **Localizador de Servicios:** (Service Locator) Se encarga del proceso de búsqueda de componentes en la red.
7. **Ensamblador de Objeto Valor:** (Value Object Assembler) Se encarga de construir los objeto valor.

[6] http://java.sun.com/developer/technicalArticles/J2EE/patterns/J2EEPatternsRelationships.html

8. **Controlador de Lista Valor:** (Value list Handler) Busca eliminar la abundancia de encabezados en la red al buscar grandes cantidades de beans entidad devolviendo solo partes de esos resultados
9. **Agregación de entidad:** (Aggregate Entity) Ofrece ayuda para generar beans de entidad de "grano grueso"
10. **Activador de Servicios:** (Service Activator) Permite procesamiento asíncrono de EJB.
11. **Objeto de Acceso a Datos:** (Data Access Object) Permite una mayor independencia con el nivel de integración al hacer que todas las peticiones no vayan directamente a la base permitiendo que esta pueda ser de cualquier tipo y esto sea transparente para el cliente.

3.4.3 Reglas de transformación PIM-PSM

Debido a que se trabaja con una arquitectura de múltiples capas, es necesario generar un PSM por cada una de estas teniendo así, un PSM-Relacional para la capa de acceso, un PSM-EJB para la capa de negocio y PSM-Web para la capa de presentación.
A continuación se muestran las reglas que se obtuvieron a través del proceso para transformar el PIM a los diferentes PSM. Debido a que este es una primer encuentro con las reglas de transformación que ofrece MDA, es posible que algunas de las reglas que se sugieren no puedan tomarse como lineamientos obligatorios para cualquier sistema, pueden necesitar refinamiento y en algunos casos ser replanteadas para poder ser aplicadas de nuevo, sin embargo, ofrecen un muy buen punto de inicio para el desarrollo de otras aplicaciones con características lógicas similares a este sistema. A continuación se mostraran las reglas que se obtuvieron para los tres PSM.

3.4.3.1 Reglas de transformación PIM a PSM Relacional: Aquí se genera un PSM relacional el cual será convertido más adelante a lenguaje SQL, para implementar luego, la Base de Datos.

a. Cada clase marcada con el estereotipo de entidad será una tabla en el PSM Relacional.
b. Cada Atributo publico en el PIM será privado en el PSM.
c. Las siguientes son las reglas para mapear los tipos de datos del PIM al PSM.
    - Un String en el PIM será un VARCAR(40) en el PSM (40 es u valor aleatorio puede ser diferente según el caso que se necesite)
    - Un integer en el PIM será un INTEGER en el PSM
    - Un date en el PIM será un DATE en el PSM
    - Un <TipoDato> en el PIM el cual tiene atributos más no operaciones, será una tabla de nombre <TipoDato> en el PSM
    - Si el tipo de dato es una clase o la representación de esta, en la tabla del PSM habrá una llave foránea que la represente.

d. Cada clase de tipo <<Entidad>> en el PIM es mapeada a una tabla en el modelo PSM y cada atributo de la clase en el PIM es una columna para la respectiva tabla en el PSM.

e. Cada tabla<NombreTabla> en el PSM Relacional tiene una llave primaria <IdNombreTabla>.

f. Las asociaciones entre clases tipo entidad en el PIM necesitan ser transformadas a una relación de llaves foráneas en el modelo relacional, inclusive creando tablas. Si tenemos una asociación de dos clases tipo entidad de A a B, las posibilidades de multiplicidad son:

- Multiplicidad es cero o uno
- Multiplicidad de A es uno
- Multiplicidad de A es más que uno

Lo mismo sucede con B, luego tenemos diferentes combinaciones de multiplicidad en ambos lados. El siguiente pseudocódigo nos permite aplicar de manera más clara esta regla:

Si la multiplicidad de ambos lados (A y B) es más que uno

Entonces se crea una tabla representando la asociación dentro de la cual se crea una llave foránea que representa a A y una llave foránea que representa a B

Sino si la multiplicidad en uno de los lados es cero o uno

Entonces se crea en la tabla una llave foránea que representa esa terminación, referenciando a la otra tabla.

Sino si /* La multiplicidad de la asociación en uno a uno */ Se crea una llave foránea en una de las tablas, referenciando a la otra

Fin Si

Fin Si

Fin Si

Por asunto de simplicidad, no se tiene en cuenta la navegabilidad de las asociaciones. Se da por hecho que todas las asociaciones son navegables en ambos sentidos. Las llaves foráneas de las tablas asociadas se escriben IdNombreTablaAsociación.

g. Las asociaciones entre clases y la multiplicidad de las asociaciones en el PIM se conservan igual en el PSM.

h. Los atributos que se mapean a columnas en PSM pueden tener o no valor NULL, excepto los que se crearon por medio de las asociaciones.

i. Por cada asociación <NombreAsociación> hay un atributo privado de nombre <Nombre Asociación> en la clase opuesta.

En la figura 3.26 se muestra el PSM Relacional. Se puede observar que hay una tabla por cada clase entidad del PIM y otras tablas extra debido a los tipos de atributos. Por ejemplo la tabla Tipo Documento fue construida debido a que el "tipo dato" Tipo Documento posee atributos como nomenclatura y tipo documento.

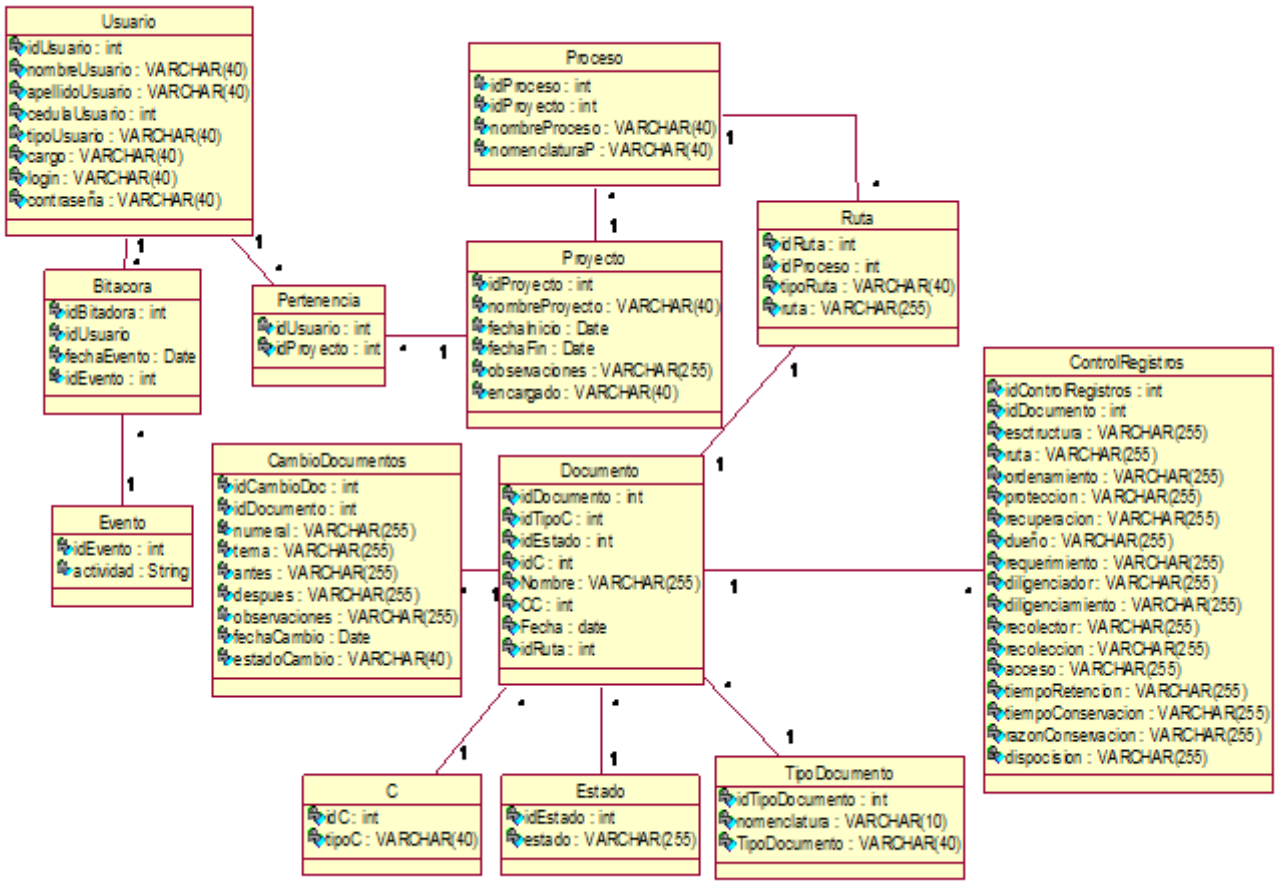


Figura 3.26 PSM Relacional

3.4.3.2 Reglas de Transformación PIM a PSM EJB: Las reglas de transformación para el PIM EJB son muy influenciadas por los patrones de diseño que se eligieron, en este caso se crean los elementos del patrón fachada y el patrón valor objeto. Las transformaciones que se sugieren en este punto son las siguientes:

a. Existe un único Gestor Fachada de tipo <<EJB Session>>

b. Por cada <<EJB Entity>> de <NombreEJB> en el PIM se crea una clase de tipo <<ValueObject>> de nombre <NombreEJBData> el cual contiene todos los atributos de la clase <<EJBEntity>>.

c. Por cada operación <NombreOperacion> en los gestores del PIM habrá una operación <NombreOperación> en el Gestor Fachada

d. Por cada clase Entidad <NombreClase> en el PIM habrá en el PSM un EJB de tipo <<EJB Entity>> <NombreClase>

e. Por cada clase Control <Nombre Clase> en el PIM habrá en el PSM un EJB de tipo <<EJB Session>><NombreClase>

f. Por cada Atributo en el PIM, <NombreAtributo> existen dos operaciones: get<NombreAtributo> y set<NombreAtributo>

En la figura 3.27 se muestra el resultado de aplicar las reglas de transformación al PIM a PSM EJB El gestor fachada es el que se comunica con los gestores o las clases control. Cuando un cliente solicite una función, entrega los datos al gestor fachada. Con esto se disminuye la cantidad de invocaciones remotas directamente a los gestores o a los beans entidad. Además tener la información encapsulada en los EJB de sesión permite re-usar todos los datos de las beans entidad en futuras aplicaciones.

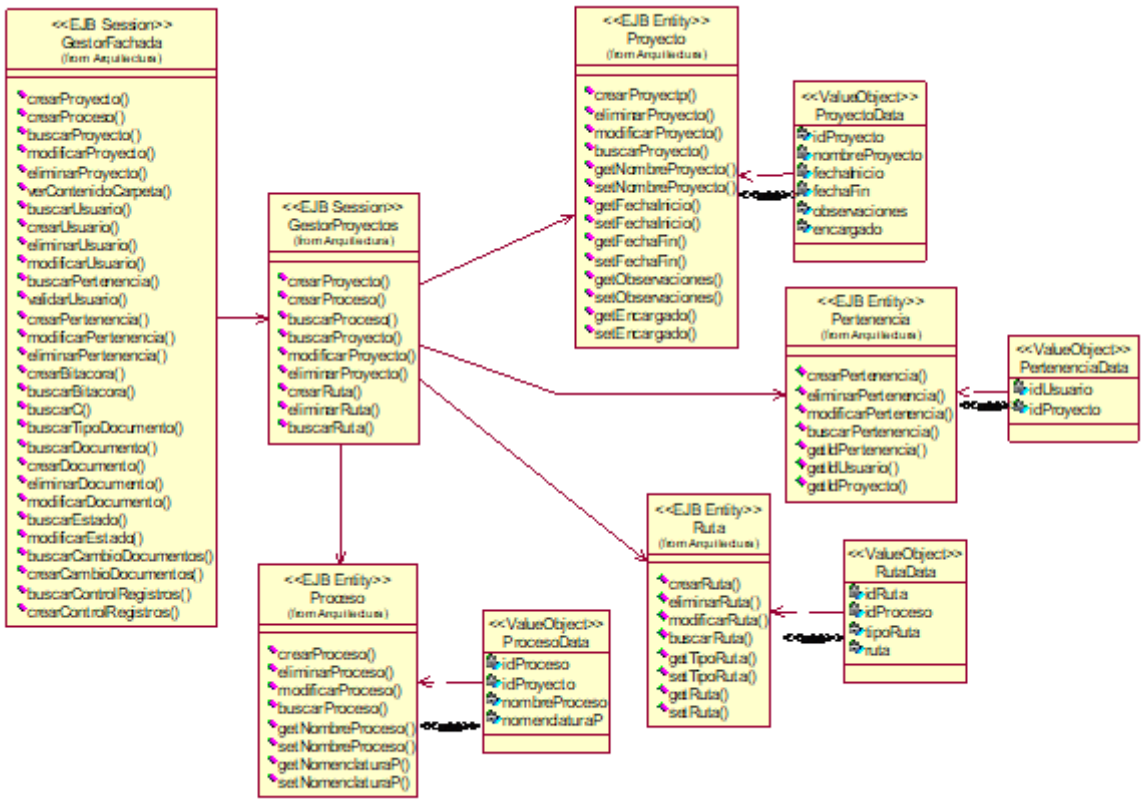


Figura 3.27 Modelo Parcial PSM EJB Caso de Uso Eliminar Proyecto

3.4.3.3 Reglas de transformación PIM a PSM WEB: Este último PSM contiene la información de los componentes de la capa del cliente. Las reglas de transformación que se obtuvieron en este proyecto son las siguientes:

a. Por cada Clase <<NombreClase>> de tipo Interfaz en el PIM, habrá en el PSM Web una clase <<NombreClase>> con el estereotipo <<JSP>>.
b. Por cada Gestor <<NombreGestor>> en el PIM habrá un UNICO Servlet <<SNombreGestor>> en el PSM-Web
c. Por cada tipo dato <NombreTipoDato> de una clase en el PIM que se relacione con un gestor <Nombre Gestor>, habrá en el gestor<NombreGestor> del PSM las operaciones básicas para este tipo de dato.
c. En cada Servlet del PSM, se encontrarán las funciones básicas (eliminar, modificar, buscar, crear) que la clase Gestor del PIM contenga.
d. Las asociaciones entre la clase JSP y Servlet son:

- Forward: Entre dos JSP o de un Servlet a un JSP
- Submit: Del JSP al Servlet

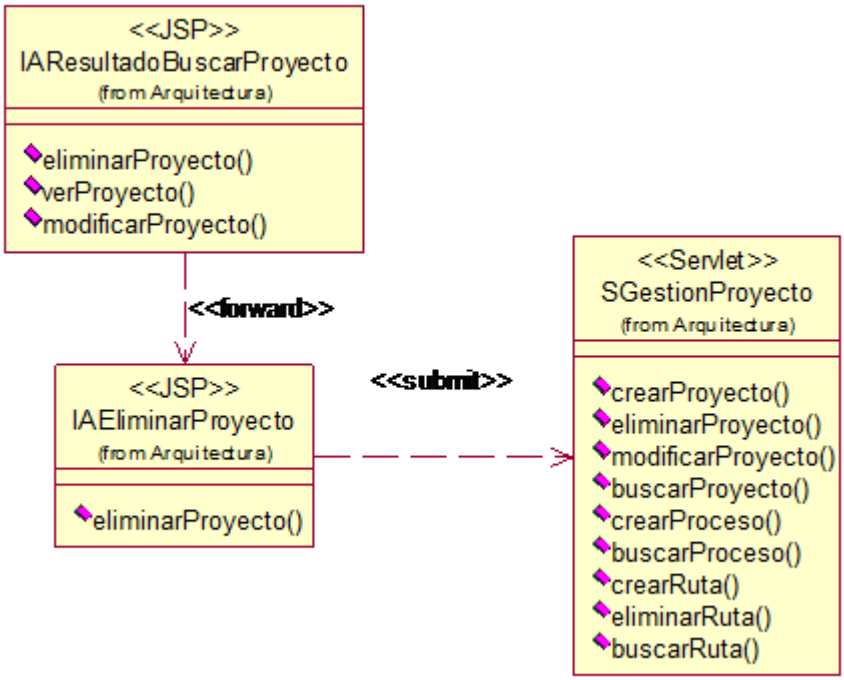


Figura 3.28 PSM Web

El proceso de MDA no termina con la creación de los modelos PSM, debido a que estos deben interactuar entre si, es necesario definir elementos con los cuales estos modelos se van a comunicar, para posteriormente, en la transformación a código, se conozca que relación hay entre que elemento y mas específicamente que búsqueda tienen que hacer para comunicarse con su contraparte. Para esto se definen los puentes de comunicación que muestran los elementos que interactúan conectando distintos modelos, existen en nuestro caso 2 puentes, el puente Web – EJB y el puente EJB – Relacional, en donde además de mostrar los elementos se define una direccionalidad de uso de los modelos.

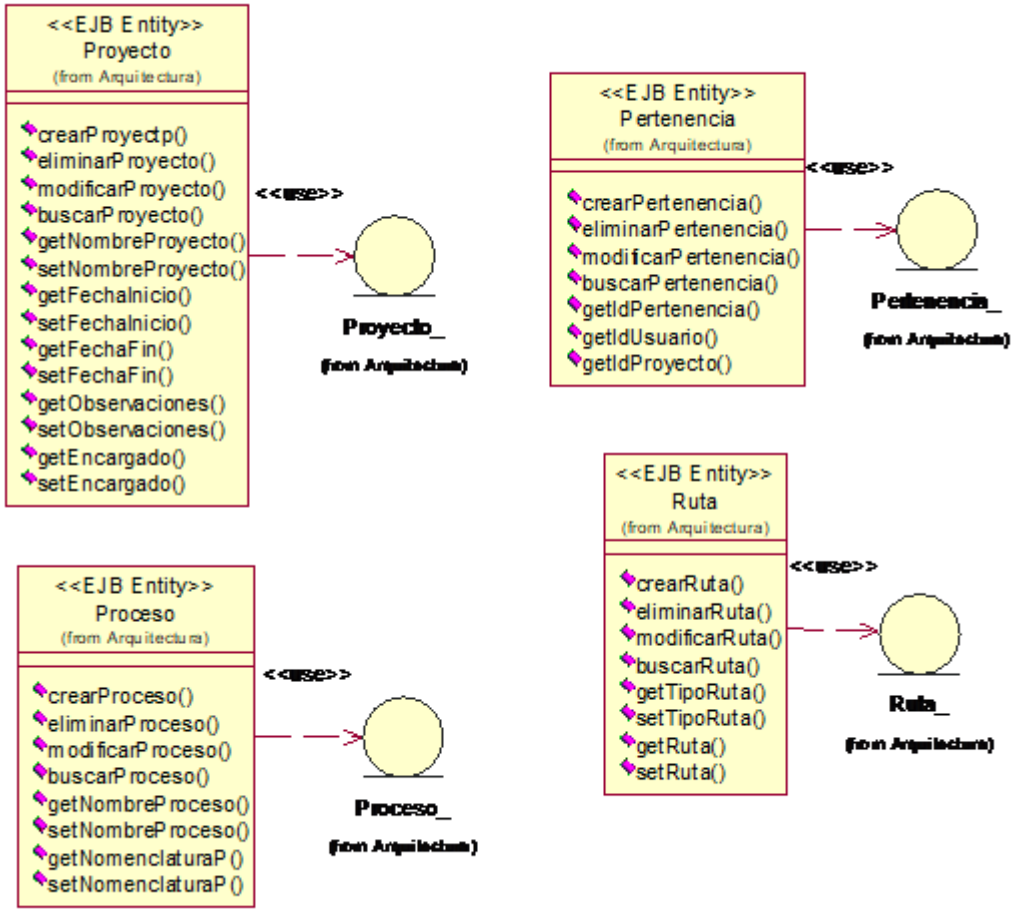


Figura 3.29 Ejemplo Puente de Comunicación Modelo PSM-EJB y PSM-Relacional

Este caso es similar para cada uno de los EJB Entity que se crean, ya que cada uno se relaciona con una tabla de la base de datos.

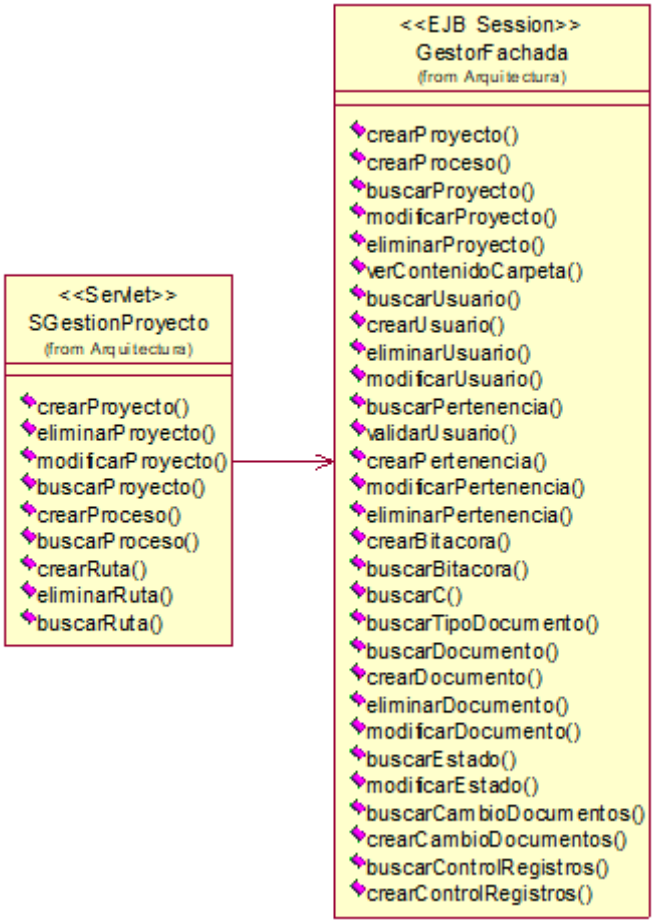


Figura 3.30 Ejemplo Puente de Comunicación Modelo PSM-Web y PSM-Ejb

En nuestro sistema se tienen tres controladores frontales para cada uno de los paquetes de análisis, es por esto que se tendrán tres puentes distintos en todo el sistema entre los controladores frontales y el Gestor Fachada.

3.5 TRANSFORMACIONES DEL PSM A CODIGO DEL SISTEMA

Para completar el desarrollo del Sistema de Gestión Documental se debe proseguir con la generación de código. Los artefactos UP pierden un poco de importancia ya que se utilizan herramientas de desarrollo software que ayudan a automatizar la obtención del código. De todas maneras es necesaria toda la información de los modelos anteriores condensada ésta en los PSM´s, los diagramas de componentes y de implantación. Además de estos artefactos, se

necesitan patrones de implementación como son los IDIOMS por ejemplo, los cuales ayudan a agilizar el diseño de la presentación al usuario.

En la siguiente figura se muestran los artefactos mencionados anteriormente y a información necesaria para llegar al código.

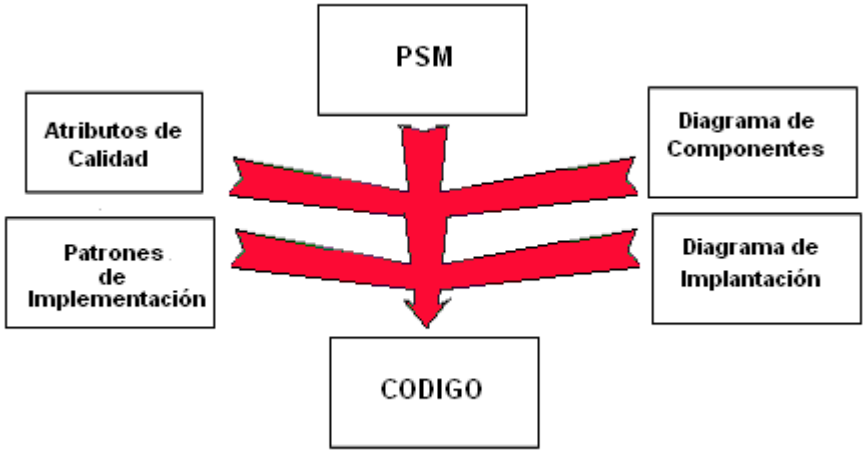


Figura 3.31 Generación de Código a partir de los artefactos UP

Las reglas de transformación se dividen en tres grupos según las capas que se manejan en la arquitectura.

3.5.1 Del PSM Relacional a Código

Se generan dos scripts desde el modelo Relacional. Uno de ellos es para la creación de las tablas y el otro para borrar las tablas. Se trabajará con el script de Creación ya que el segundo es similar.

Las reglas de transformación que se sugieren para este caso son:

a. Por cada tabla <<NombreTabla>> en el PSM-Relacional se genera una sentencia CREATE TABLE <<NombreTabla>>

b. Por cada columna de las tablas en el PSM-Relacional se genera el nombre de la columna, seguido por el tipo de dato y si es nula o no nulo.

c. Se crea la llave Primaria colocando en el texto PRIMARY KEY(<<columnaLlavePrimaria>>)

d. Si existe una llave foránea en la tabla se crea como cualquier columna de la tabla

El ejemplo de código para la tabla Proyecto:

```
CREATE TABLE Proyecto
idProyecto          Integer         NOT NULL
nombreproyecto      VARCHAR(40)     NOT NULL
fechaInicio         Date            NOT NULL
fechaFin            Date            NULL
observaciones       VARCHAR(255)    NULL
encargado           VARCHAR(255)    NOT NULL
PRIMARY KEY (idProyecto)
);
```

3.5.2 Reglas de transformación del PSM EJB a Código

Primero se debe observar la figura 3.32 las transformaciones que hasta el momento se han realizado.

En este trabajo de Grado se realizará la transformación del PSM EJB al PSM –EJB(Clases) primero. Un ejemplo de la generación de código a partir de estas clases se dará más adelante

Según la Especificación EJB[7] los Entity Beans deben estar provistos generalmente de lo siguiente:

- Una clase Entity Bean
- Una clase de Llave Primaria
- Una clase Interfaz Remota del Entity Bean y una clase Interfaz Remota Home
- Una clase Interfaz Local del Entity Bean y una clase Interfaz Local Home (Si el cliente tiene una vista local de su sistema)
- Si el Bean maneja su propia persistencia debe existir la implementación de los métodos en una clase.

[7] Sun Microsystems, Enterprise JavaBeans Specification Versión 2.1,2002

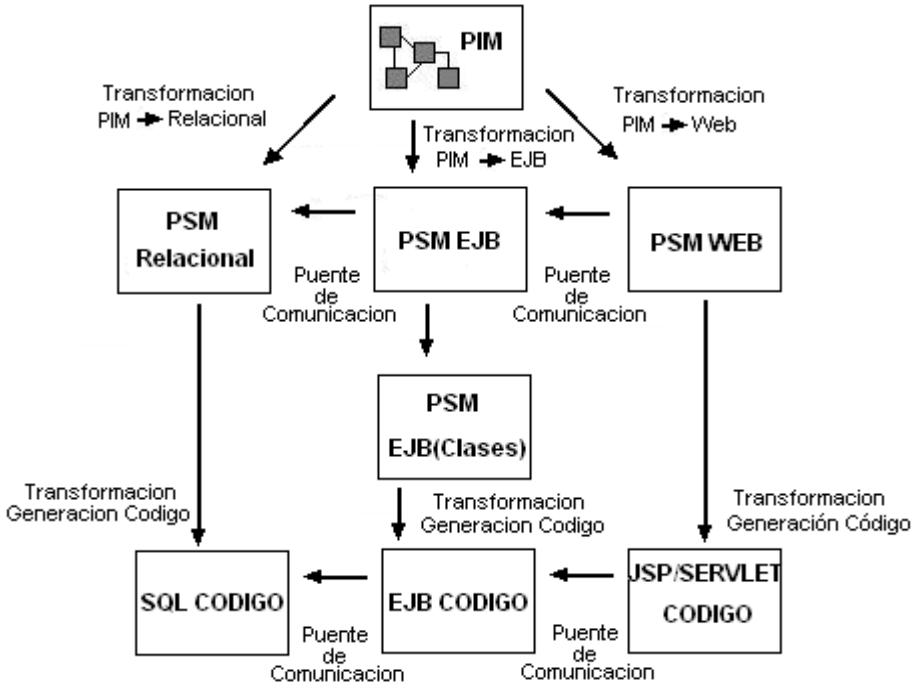


Figura 3.32 Los Procesos de transformación en MDA

Siguiendo con las las reglas de transformación, se tienen las siguientes para los Bean Entidad:

a. Para cada <<EJB Entity>> con nombre <<NombreEJB>> en el PSM, se genera una clase Interfaz EJB Home <<NombreEJBHome>> , una Interfaz EJB Remota <<NombreEJB>>, una clase en donde esta la implementación del EJB <<NombreEJBBean>> y en este caso como se maneja persistencia por el Bean, una clase para implementar los métodos llamado <<NombreEJBBeanBMP>>.

b. Para cada <<EJB Entity>> con nombre <<NombreEJB>> en el PSM debe existir una clase <<NombreEJBPK>> para cumplir con la especificación de llave primaria y poder acceder a este atributo del EJB de tipo entidad. Esta clase lleva las funciones hashCode(), equals() y toString(). Estas según la especificación EJB.

c. Por cada método de la clase <<EJBEntity>> hay un método en la interfaz Remota y un método en la clase de implementación del EJB <<NombreEJBBean>>.

d. Un método “get< Atributo >” y un “set< Atributo >” por cada atributo de un Bean Entidad en cada interfaz Remota y en cada clase de implementación del EJB <<NombreEJBBean>>.

e. Para la llave primaria de cada tabla no se generan las funciones get y set, para esto es la clase PrimaryKey.

e. Todos los métodos estándar (Según la especificación EJB store(),remove(),load()) para los beans deben estar implementados en la clase de implementación del Bean <<NombreEJBBean>> y si se maneja persistencia por el Bean en la clase de persistencia <<NombreEJBBeanBMP>>..

f. En la Interfaz Remota Home se deben generar tres métodos

1. Un método Creación (create)
2. Un método Búsqueda basado en la implementación de la llave EJB
3. Un método de Búsqueda *findall* para recuperar todos los valores del bean entidad.

g. Cada operación que se quiera adicionar a la Interfaz Home debe tener el nombre precedido del prefijo find o create.

h. Cada operación que se quiera adicionar a la clase de implementación, debe tener el nombre precedido del ejb.

Las reglas de transformación para los Bean Sesión son las siguientes:

a. Para cada <<EJB Session>> con nombre <<NombreEJB>> en el PSM, se genera una clase Interfaz EJB Home <<NombreEJBHome>> , una Interfaz EJB Remota <<NombreEJB>>, una clase en donde esta la implementación del EJB <<NombreEJBBean>>

b. Por cada método en el <<EJB Session>> hay un método en la interfaz Remota y un método en la clase de implementación del EJB <<NombreEJBBean>>.

c. Todos los métodos estándar (Según la especificación EJB. store(),remove(),load()) para los beans deben estar implementados en la clase de implementación del bean.

Cuando se está trabajando en un sistema distribuido, es bueno trabajar con la interfaz remota, pero existe una interfaz que es la interfaz local. Se recomienda trabajar con esta cuando el cliente del bean y el bean esten en la misma máquina. Un cliente puede ser un Servlet, un Applet, otro bean o una aplicación. Si es así, por cada Bean de tipo sesión o de tipo entidad se crea una clase <<NombreEJBLocalInterface>> en donde se implementan las funciones que se van a utilizar.

Como se vio en la descripción de J2EE existe otro tipo de Bean: Bean de Mensaje, estos Bean´s no se utilizaron en este trabajo de grado ya que no se vio la necesidad de trabajar con ellos en un principio. Pero teniendo en cuenta los trabajos futuros que pueden hacerse con este sistema, los Bean Mensaje son de gran ayuda para la intercomunicación de los beans en los sistemas de automatización Workflows o Sistema de automatización de flujos de trabajo. Esto es importante

tenerlo en cuenta para trabajos futuros en donde se necesite intercambio inmediato de información entre Bean´s

Para graficar las anteriores reglas de transformación se presenta el ejemplo del bean Entidad Proyecto:

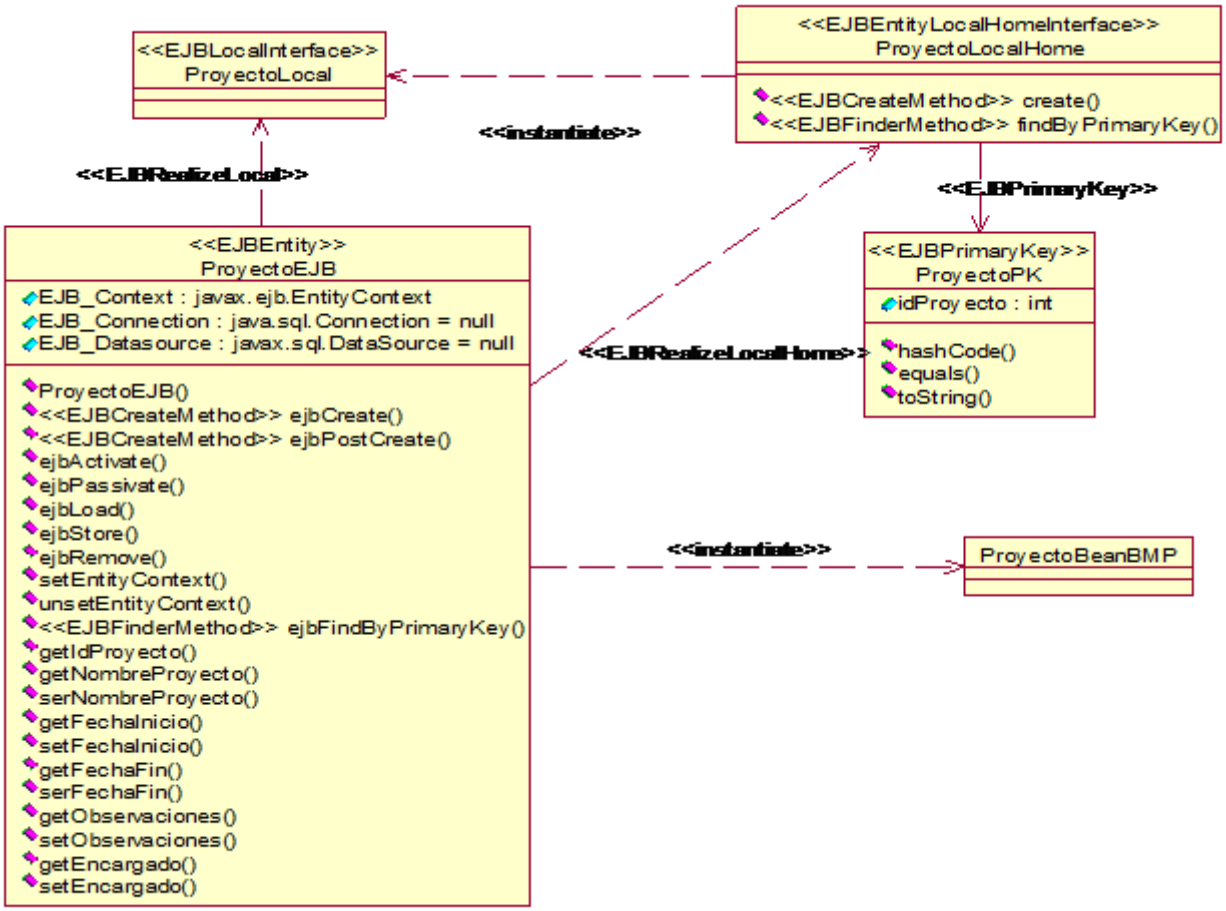


Figura 3.33 Clases de Diseño EJBEntity Proyecto

[JBuilder], la herramienta que ofrece [Borland™] ofrece un módulo para trabajar con J2EE. Solo basta con introducir los datos de los nombres y las conexiones de la base de datos para que se generen los esqueletos de las clases. En este trabajo de grado se utilizó esta herramienta para generar todos los EJB de sesión y entidad. Además de esto también se producen los descriptores XML y los diagramas UML. A continuación se muestra el bean Entidad Proyecto generado por esta herramienta:

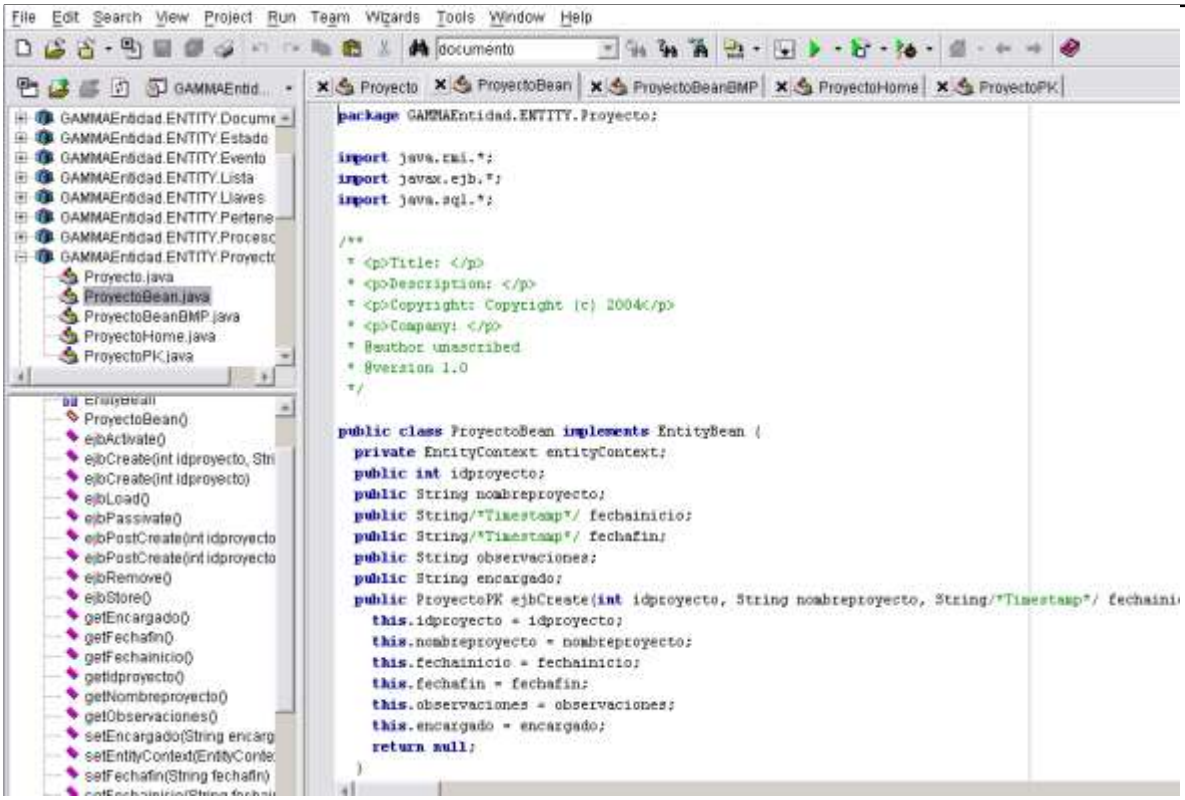


Figura 3.34 Herramienta JBuilder-Produccion de EJBEntityProyecto

3.5.3 Del PSM Web a Código

No es tan sencillo llevar el modelo Web a Código, aún con herramientas de transformación es difícil llegar a un código que tenga todas las funcionalidades del sistema. Para generar las interfaces del cliente se utilizará JSP[8] y patrones de Implementación (Idioms) y para el Servlet se usa el patrón de diseño Controlador Frontal propuesto por Sun Mycrosystems TM

Las reglas de transformación que surgieron en la parte Web de la aplicación son las siguientes:

a. Por cada Interfaz se crea una página jsp.

b. Según el puente de comunicación entre el PSMWeb y el PSMEjb serán especificados los atributos en las páginas. Así por ejemplo la relación existente entre la interfaz IACrearProyecto y el EJB Entidad Proyecto, brindan los atributos del proyecto necesarios para colocar en la interfaz del cliente.

c. Por cada interfaz <<nombreInterfaz>> se crea una página <<nombreInterfazErrorPage>> para mostrar las excepciones al usuario.

[8] SunMicroSystems , Java ServerPages Especification Versión 1.2 2001

d. Por cada servlet se crea una clase de tipo HttpServlet, con la implementación de las funciones que contenga en el PSM Web.

e. Cada servlet, además de las funciones anteriores, lleva las siguientes operaciones:

- init()
- destroy()
- doGet()
- doPost()

f. Por cada operación en el JSP del PSMWeb se generará un botón en la interfaz JSP.

En nuestro Proyecto de grado se utilizaron, para el desarrollo de las páginas los siguientes IDIOMS que Sun Mycrosystems recomienda[9]:

- Browse Idiom para búsquedas de documento.
- Add-and-Remove Idiom para la selección de proyectos y procesos en las páginas de creación.

Como podemos ver el modelo PSM Relacional es más estructurado y puede ser fácilmente modificado a código. El código de los EJB se entrega en forma de clases para ser implementadas por el desarrollador sin menor esfuerzo y el último modelo es muy sencillo y sirve para guiar el desarrollo de la aplicación Web.

A continuación se presenta un ejemplo de Servlet SGestorProyectos:

```
public class SGestorProyecto extends HttpServlet {
  static final private String CONTENT_TYPE = "text/html";

FachadaHome fachadah = null; //Este elemento llama a el EJB Fachada para comunicarse con la parte lógica del sistema

public void init() throws ServletException {

 }

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

   response.setContentType(CONTENT_TYPE);
   PrintWriter out = response.getWriter();
   out.println("<html>");
   out.println("<head><title>SGestorUsuario</title></head>");
   out.println("<body>");
   out.println("<p>The servlet has received a GET. This is the reply.</p>");
   out.println("</body></html>");
    }
```

[9] http://www.java.sun.com

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
response.setContentType(CONTENT_TYPE);
PrintWriter out = response.getWriter();
String pagina = request.getParameter("pagina");

If(pagina == crearProyecto){
  this.crearProyecto{...  }
}

        If(pagina == eliminarProyecto){
  this.eliminarProyecto{…   }
}

        If(pagina == modificarProyecto){
  this.modificarProyecto{…   }
}
If(pagina == buscarProyecto){
this.buscarProyecto{…   }
}

  }
    public void destroy() {
  }
```

3.6 DIAGRAMAS DE IMPLANTACION Y COMPONENTES

Estos diagramas nos dan información para finalizar con la codificación, dándonos información acerca de protocolos a utilizar y nos da una idea de la distribución real de la aplicación.

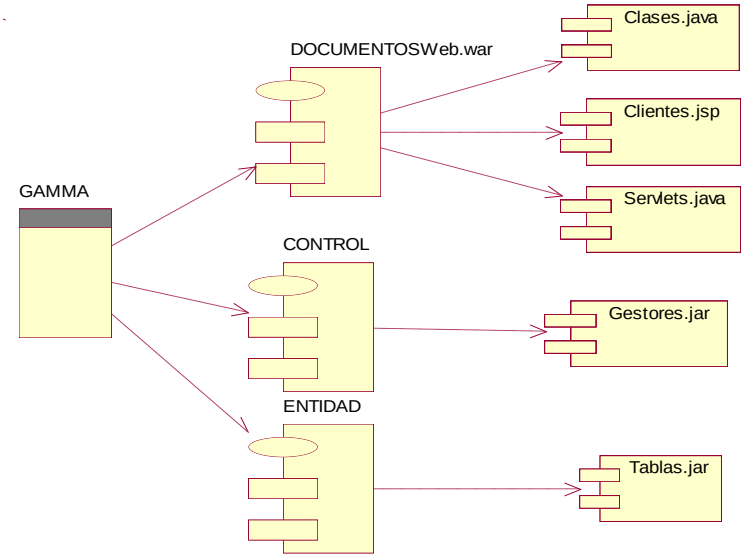


Figura 3.35 Diagrama de Componentes

Los componentes a la derecha son la representación de las clases, y los beans del sistema. El DocumentosWeb.war contiene la aplicación Web, todo lo de la capa de presentación como lo son las páginas, servlets y algunas clases extras. El componente de control contiene a los gestores (jar de los Beans de Sesión), y el de Entidad a los beans de Entidad.

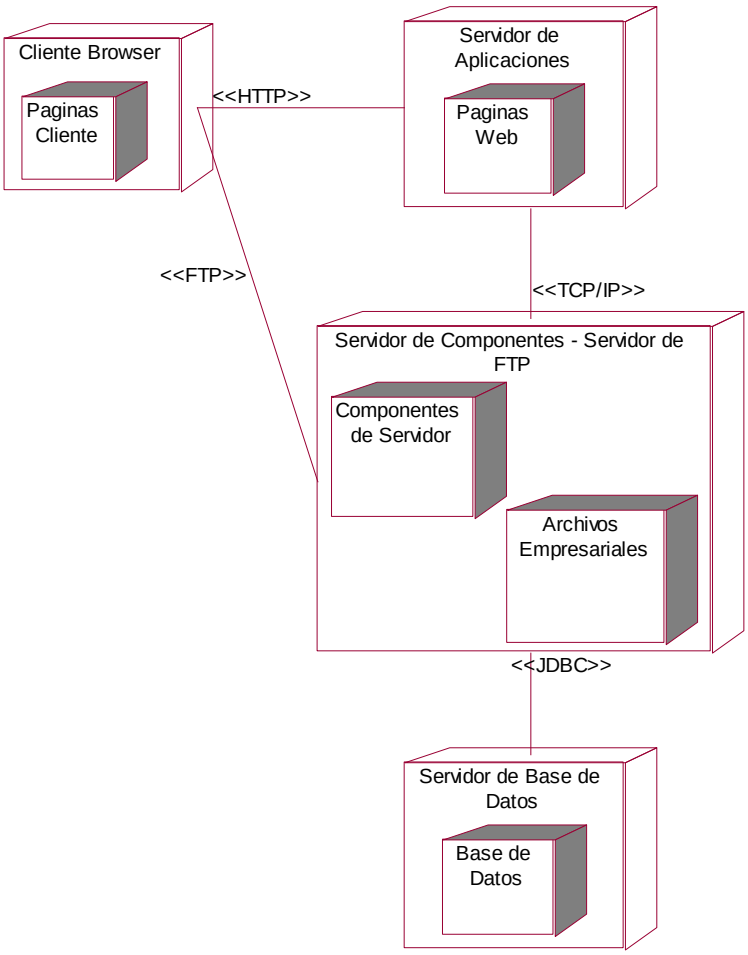


Figura 3.36 Diagrama de implantación

3.7 EJEMPLO DEL LOG DE TRANSFORMACIÓN PARA LA CLASE DE TIPO ENTIDAD PROYECTO

El log de transformación es un registro de las transformaciones hechas a un modelo para obtener otro a partir de reglas y patrones. Se realiza para observar de manera más sencilla la transformación de un objeto de un modelo a otro. El log puede servir de referencia para la creación de las Definiciones de Transformación que necesitan las herramientas. Además este log me permite ver la transaccionalidad de las transformaciones de un modelo a otro.
A continuación se realiza el log del concepto Proyecto

3.7.1 Transformación CIM a PIM

Evaluando clases…
Proyecto (concepto) ->Proyecto (Clase)

Evaluando atributos…
nombreProyecto(Atributo)->nombreProyecto(Atributo):String
fechaInicio(Atributo)->fechaInicio(Atributo):Date
fechaFin(Atributo)->fechaFin(Atributo):Date
observaciones(Atributo)->observaciones(Atributo):String
encargado(Atributo)->encargado(Atributo):String
Creando clases básicas en el PIM…
Proyecto.crearProyecto()
Proyecto.eliminarProyecto()
Proyecto.modificarProyecto()
Proyecto.buscarProyecto()

La herramienta que se este utilizando debe dar la opción para introducir al PIM información como el tipo de dato de cada atributo y clases diferentes a las básicas.

3.7.2 Transformación PIM a PSM

Tomando la clase proyecto tenemos el siguiente Log:
Verificando tipo de clase proyecto…
Clase Proyecto = Entity
Convirtiendo atributos públicos a privados…

+nombreProyecto: String-> -nombreProyecto:VARCHAR(40)
+fechaInicio: Date-> -fechaInicio:Date
+fechaFin: Date-> -fechaFin:Date
+observaciones:String-> -observaciones:VARCHAR(255)
+encargado:String-> -encargadoVARCHAR(255)

Generando las operaciones básicas…

Proyecto.getNombreProyecto
Proyecto.getFechaInicio
Proyecto.getFechaFin
Proyecto.getObservaciones
Proyecto.getEncargado
Proyecto.setNombreProyecto
Proyecto.setFechaInicio
Proyecto.setFechaFin
Proyecto.setObservaciones
Proyecto.setEncargado

Etiquetando clase Proyecto

<<EJBEntity>> Proyecto

3.7.3 Transformación PSM a CODIGO

Verificando clase de EJB

Proyecto= EJBEntity

Aplicando Reglas de transformación PSM – EJB (Clases) para Bean Entidad…

<<EJBEntity>>Proyecto ->

Generando Interfaz Home…

<<EJBEntityHomeInterface>> ProyectoHome

Generando funciones ProyectoHome…

EJBMétodoCreación…

createProyecto(int idProyecto, String NombreProyecto, Date fechaInicio, Date fechaFin,

String Observaciones, String encargado):Proyecto

EJBMétodoBúsqueda...

findByPrimaryKey(ProyectoPK primaryKey):Proyecto

findAll():Collection

Generando Interfaz Remota…

<<EJBRemoteInterface>> Proyecto

Verificando llaves Foraneas…

Verificando llave Primaria.

Generando funciones Proyecto…

EJBMétodoRemoto…

getNombreProyecto()
getFechainicio()
getFechafin()
getObservaciones
getEncargado()
setNombreProyecto(String nombreProyecto)
setFechainicio(Date fechaInicio)
setFechafin(Date fechaFin)
setObservaciones(String observaciones)
setEncargado(String encargado)

Generando Persistencia por el Bean…

<<EJBPersistencia>> ProyectoBeamBMP

Generando funciones ProyectoBeamBMP…

ejbCreateProyecto(int idProyecto, String NombreProyecto, Date fechaInicio, Date fechaFin, String Observaciones, String encargado):Proyecto
ejbCreateProyecto(int idProyecto):Proyecto
ejbRemove()
ejbLoad()
ejbStore()
setEntityContext(EntityContext entityContext)
unsetEntityContext()
ejbFindByPrimaryKey(ProyectoPK proyectoKey): ProyectoPK
ejbFindAll():Collection

Generando EJB…

<<EJBImplementacion>>ProyectoBean

Generando atributos…

idProyecto:int
nombreProyecto:VARCHAR(40)
fechaInicio Date
fechaFin:Date
observaciones:VARCHAR(255)
encargado:VARCHAR(40)

Generando Funciones…

ejbCreateProyecto(int idProyecto, String NombreProyecto, Date fechaInicio, Date fechaFin, String Observaciones, String encargado):Proyecto
ejbCreateProyecto(int idProyecto):Proyecto

ejbREmove()
ejbPostCreateProyecto(int idProyecto, String NombreProyecto, Date fechaInicio, Date fechaFin, String Observaciones, String encargado):Proyecto
ejbPostCreate(idProyecto)
setEntityContext()
ejbActivate()
ejbPassivate()
ejbLoad()
ejbStore()
getIdProyecto()
getNombreProyecto()
getFechainicio()
getFechafin()
getObservaciones()
getEncargado()
setNombreProyecto(String nombreProyecto)
setFechainicio(Date fechaInicio)
setFechafin(Date fechaFin)
setObservaciones(String observaciones)
setEncargado(String encargado)

3.7.4 Transformación PSM Relacional a Código

Leyendo atributos <<EJBEntity>>Proyecto…
Creando Tabla
CREATE TABLE Proyecto
Recibiendo datos NULL…
idProyecto:Integer->input=NOT NULL
nombreproyecto:VARCHAR(40)->input=NOTNULL
fechaInicio:Date->input=NOT NULL
fechaFin:Date->input=NULL
observaciones:VARCHAR(255) ->input=NULL
encargado:VARCHAR(255)->input=NOT NULL
introducir llave primaria
obtener imput
creando llave primaria…
PRIMARY KEY (idProyecto));

En este log se observa que se necesitan entradas de parte del desarrollador de la aplicación, así que las herramientas que se desarrollen deben permitir personificar las definiciones de transformación automáticas.

CAPITULO 4. RECOMENDACIONES PARA EL TRABAJO CON MDA

Después de haber realizado todo un proceso de desarrollo para un sistema, en este caso el sistema de gestión documental basado en la norma ISO 9001:2000, podemos observar que la gran diferencia que tiene el desarrollo convencional al desarrollo basado en el enfoque MDA-UP, es que este último se concentra mucho más en los aspectos del negocio, haciendo que aumente la reutilización del software en un alto nivel de abstracción lo que aumenta la productividad, calidad y el ahorro de tiempo y costos en una empresa desarrolladora de software.

En este capitulo se mostrará la influencia que MDA tiene sobre el proceso de desarrollo convencional y sus flujos de trabajo, las ventajas que una empresa desarrolladora de software puede adquirir al adoptar este enfoque en los procesos de desarrollo tradicionales que se maneja, así como futuras ampliaciones que se deben seguir para hacer de MDA el futuro en el desarrollo de aplicaciones software.

4.1 EL LENGUAJE DE LAS DEFINICIONES DE TRANSFORMACIÓN

Como se vio en el capitulo 1 las definiciones de transformación son aquellas que las herramientas necesitan para saber cómo se deben transformar los modelos [MDAEX]. Las reglas que se sugirieron para la conversión de modelos son entendibles para un lector pero no para las herramientas de transformación. Hoy en día está bajo construcción un estándar llamado QVT (Query Views and Tranformation) el cual definirá como serán las reglas de transformación entre modelos cuyos lenguajes estén definidos en MOF. Por el momento muchos autores manejan el lenguaje OCL el cual sirve para definir expresiones sobre los modelos, como cuerpos de operaciones, derivar reglas de atributos de clases o realizar restricciones con post y precondiciones sobre operaciones. Las expresiones que se realicen con OCL pueden ser pasadas a un lenguaje de programación como java por ejemplo para que puedan ser ejecutadas.

En MDA el OCL se utiliza generalmente en UML para realizar restricciones a los modelos pero también se ha utilizado para definir reglas de transformación en MDA. Por ejemplo una sentencia OCL podría especificar el elemento en el modelo fuente mientras que la otra sentencia podría especificar el elemento en el modelo específico.

Un ejemplo de cómo podrían ser estas definiciones de transformaciones se muestra a continuación:

```
Transformation CalssToTable(UML,SQL){ //Es una transformación de un modelo UML a codigoSQL
Params
        Setterprefix: String = ´set´;
        Setterprefix: String = ´get´;
Target
        c:SQL::Colum; //Es
        f:SQL::ForeingKey;
Target Condition
        f.value = c and c.tupe = f.refersTo.value.type;
maping //reglas de mapeo
        c.name <~> t.name;
clases->forAll(c | tables ->exist (t | c<~> t);//Para cada clase en el PIM una tabla en el código.
and
tables->forAll(t | clases ->exist (c | c<~> t);//Por cada tabla en el código, una clase en el PIM
}
```

Esta es la definición de transformación para pasar todas las clases del PIM en tablas a código SQL.

Queda mucho trabajo por hacer con MDA. Muchas personas aseguran que esta es la mejor solución a los problemas de diseño otras ven a MDA como una idea incipiente que tiende a estancarse. Pero lo que se puede decir con esta experiencia, es que MDA como concepto es muy bueno ya que permite crear un nivel más de abstracción que ayudará a eliminar muchos de los procedimientos que tradicionalmente se siguen. Así empezaron a evolucionar los sistemas software, ahora la programación orientada a objetos le gana a la vieja programación estructurada y el panorama de desarrollo software que se ve en un futuro con MDA es mucho mejor que el de ahora.

4.2 EL PROCESO DE DESARROLLO DE SISTEMAS EMPRESARIALES

Uniendo la definición que hace el SEI (Instituto de Ingeniería Software): el proceso de desarrollo software es un conjunto de actividades, métodos y prácticas utilizadas por un grupo humano para desarrollar y mantener el software y sus productos asociados, con MDA y su nuevo enfoque de obtención de modelos a partir de un modelo enfocado al negocio, se obtiene como resultado una serie de implicaciones que se aplican tanto a los artefactos que se producen en el UP como a las actividades que comúnmente se siguen mediante las diferentes iteraciones y fases del proceso.

Como se observó en el anterior capitulo, algunos elementos tienden a desaparecer y otros a volverse un poco más robustos teniendo en cuenta que el mayor esfuerzo se encuentra en los primeros flujos en donde se crean los modelos más abstractos como lo son el modelo independiente de la plataforma, y que el menor esfuerzo, en el caso de contar con una buena herramienta de transformación, estaría en el flujo de implementación.
El enfoque MDA-UP que se trabajó en esta tesis influye también en los grupos de trabajo de la organización. Se requiere de personal especializado en los estándares que maneja MDA, así como los diferentes patrones arquitecturales y de diseño. Además si se desea crear herramientas de transformación se necesita de especialistas en metamodelos y lenguajes de programación para estos.

A continuación se tiene una lista de recomendaciones y características de los flujos que se obtienen al aplicar e enfoque MDA-UP a los diferentes flujos de trabajo:

4.2.1 Especificación

Este flujo de trabajo es importante ya que es aquí en donde se obtienen los requerimientos del cliente. Es necesario dejar muy claro estos requerimientos de tal manera que se construya un modelo conceptual CIM adecuado, ya que este modelo es básico para implementar el PIM. Pero además de la obtención de los requerimientos del cliente, en este flujo se obtienen las cualidades o atributos de calidad a partir de las características de negocio y de los requerimientos no funcionales que entrega el cliente. Los atributos de calidad son vitales para tomar decisiones respecto a los patrones y la arquitectura con los que se va a trabajar. Estas cualidades pueden ser obtenidas a partir de los escenarios de calidad vistos en el primer capitulo.
Se puede observar que la influencia de MDA en este flujo, hace que este sea mucho más robusto que el tradicional flujo en UP.

4.2.2 Análisis

Es en este flujo de trabajo donde se crea el PIM con la ayuda de las clases de análisis y los diagramas de secuencia, artefactos del proceso UP. Es necesario realizar en cada flujo el Log de Transformación para tener en cuenta la bidireccionalidad de los modelos y así sea muy fácil añadir cambios cuando se pase de una fase del proceso a otra.
Este flujo utiliza los mismos diagramas que en el desarrollo UP tradicional. El PIM se forma básicamente de los diagramas de clases de análisis, el cual es completado por toda la información

referente a los patrones de arquitectura escogidos por el arquitecto y el grupo de análisis, dependiendo del las características de calidad del flujo anterior junto a una serie de información que el arquitecto ha adquirido con la experiencia.

Durante el análisis del PIM debe existir un grupo personas que deben ser concientes de las funcionalidades del sistema que deben ser implementadas y estos deben guiarse por la lógica del negocio obtenida en el anterior flujo.

En este flujo también se toman en cuenta los atributos de calidad para la elección de los patrones de diseño del siguiente flujo.

4.2.3 Diseño

El diagrama principal en este flujo es el PSM. Básicamente a las tradicionales clases de diseño que son el equivalente del PIM se le aplican las transformaciones de MDA. Acompañado de la información de los atributos de calidad y los patrones de diseño el PIM se convierte en varios PSM según la tecnología que se utilice. Si el PSM de la plataforma que se escoge esta predefinido, la conversión puede hacer que el proceso de desarrollo se agilice, pero si esta no es estándar, como .Net o J2EE, el proceso de desarrollo puede convertirse en este flujo de trabajo algo dispendioso que no aportaría eficiencia a la producción software.

Para la generación del PSM, se necesita un grupo de personas que tengan conocimiento de diferentes plataformas y diferentes arquitecturas y sobre todo que tengan conocimiento de las correspondientes transformaciones MDA y aspectos de calidad de los sistemas, específicamente, se necesita un grupo de personas que sean responsable de la bidireccionalidad de los modelos, se recomienda que sean los mismo que realizan las transformaciones en la empresa.

La elección de este grupo de personas es importante, ya que de las decisiones que se tomen en este flujo de trabajo depende la obtención de un buen producto software.

Como se ve, este flujo es uno de los más afectados por el enfoque ya que se eliminan la mayoría de los artefactos tradicionales del desarrollo UP.

4.2.3 Implementación

Este flujo de trabajo pierde importancia si se utilizan herramientas que generan el código a partir o de un PSM o a partir del PIM. Incluso, si se utiliza la transformación manualmente para la plataforma [J2EE] por ejemplo, se utilizan IDE´s como [Jbuilder] o [NetBeans], el cual genera la estructura del código de cada EJB y sólo se tiene que realizar la implementación de las funciones particulares de la aplicación manualmente. En este flujo se requiere de información adicional de patrones de implementación que ayudan a generar el código manualmente. La transformación del PSM a código realizada manualmente no aumenta productividad al proceso. En este flujo es

importante contar con herramientas de transformación automática de modelos MDA para obtener valor agregado en cuanto a la disminución del tiempo de desarrollo.

El grupo de personas que se necesita debe ser experto en programación de código en la que la plataforma específica esta hecha ya que las herramientas de hoy en día generan solo esqueletos de código.

Con el tiempo y la evolución de estas herramientas, MDA logrará ocultar las plataformas y es en este momento donde este grupo de personas tiende a desaparecer.

4.3 LAS HERRAMIENTAS

4.3.1 Características claves de las transformaciones

En el primer capítulo se definió el concepto de transformación como la generación de un modelo objetivo a partir de un modelo fuente. Significa que la transformación es un proceso el cual está descrito por una serie de definiciones de transformaciones las cuales a su vez se constituyen por reglas de transformación MDA y es ejecutado por herramientas de transformación MDA. [MDAEXPLAINED]. Las características más deseables que el proceso de transformación debe tener son las siguientes:

- Configurabilidad: Esta consiste en que se puedan cambiar las características de las reglas de transformación. Por ejemplo en la transformación del PSM Relacional, cuando se convierte un String a un VARCHAR, se le puede dar cualquier tamaño que se desee dependiendo de las necesidades del atributo. Otro ejemplo en el mismo caso seria que un String pudiera transformarse en un TEXT si es necesario.
- Trazabilidad: Cualquier elemento en el modelo objetivo, puede ser llevado al modelo fuente del cual ha sido generado.
- Bidireccionalidad: cualquier cambio de información en el modelo fuente debe afectar al modelo objetivo y viceversa.

4.3.2 Características de las Herramientas de Transformación

Además de las características para el proceso de transformación, existen características que se deben tener en cuenta en el momento de escoger o desarrollar las herramientas de transformación que soportan este proceso. Las más relevantes son:

- Control Manual: para introducir algunas condiciones adicionales que el desarrollador considere importante para el desarrollo del sistema. Uno de los métodos para llevar a cabo esto es adicionando a las definiciones de transformación, condiciones especiales que

pueden introducirse por medio de la interfaz de la herramienta manualmente. Otro método es adicionar parámetros que se encarguen de modificar estas características.

- Selección de plataformas: La Herramienta debe proporcionar al administrador la opción de escoger la plataforma específica para la transformación del modelo. Estas herramientas podrían contener módulos o cartuchos para las diferentes plataformas. Dependiendo de la tecnología que se vaya a utilizar se coloca o se quitan los cartuchos.
- Usabilidad en el contexto del proceso de desarrollo: Las herramientas deben seguir ciertos pasos que permitan seguir con las fases y los flujos del proceso de desarrollo, ya que esto ayudará a agilizar la información que se necesita del sistema. Entre más rápido y completa se obtenga la información del sistema a desarrollar, más rápido se hace el desarrollo software.
- Flexibilidad: Para poder cambiar dentro de la plataforma que se escogió diferentes estrategias de implementación, arquitecturas y patrones.
- Herramienta PlugIn: La herramienta deben soportar la inclusión de diferentes paquetes con lenguajes de transformación diferentes, es decir dar soporte a lenguajes como [XMI], [UML 2.0] y [OCL].
- Soporte para dominios estándar y BluePrints
- Integración: con otras herramientas que automatizan el desarrollo software, por ejemplo herramientas que gestionan requerimientos, realizan ingeniería inversa, de prueba, de configuración de desempeño entre otras.

4.3.3 Desarrollo de Herramientas de Transformación

Una de las ventajas que el MDA ofrece, es la conversión automática de los modelos. Es en esta transformación donde esta uno de los valores agregados que nos brinda este nuevo enfoque, ya que con la automatización se reduce mucho el tiempo de la realización de las aplicaciones. En la siguiente lista se encuentran algunas de las recomendaciones que las empresas desarrolladoras de software y los futuros trabajos de la Facultad de Ingeniería Electrónica y de Telecomunicaciones pueden tomar si piensan en desarrollar herramientas basadas en MDA:

1. Tener en cuenta las reglas de transformaciones vistas anteriormente.
2. Generar herramientas de transformación basadas en lenguajes de transformación como [OCL] el cual permite generar reglas de transformación entendibles tanto para una maquina como para un humano. Otro lenguaje muy utilizado es el [XML] se podrían realizar plantillas de reglas de transformación y reutilizarlas según las necesidades de los modelos a transformar.
3. Existen tres tipos de transformación de modelos, los vistos en el Capitulo uno: El modelo de transformación por marcas y el modelo de transformación por patrones e información

adicional. El tercer tipo de transformación de modelos es el meta-modelo de transformación. Este tipo de transformación produce los modelos a partir de especificaciones de transformación de los meta-modelos. [MDAGV1]. La figura 4.1 muestra este tipo de transformación. Para entender un poco mejor este concepto se puede tomar un profile UML en lugar de el meta-modelo independiente de la plataforma, de este modelo surge el PIM con todas las especificaciones del profile UML, a este PIM se le aplican las especificaciones de transformación para el metamodelo especifico de la plataforma que para este ejemplo podría ser un profile para Java J2EE. Así el PSM se realiza con los elementos que el meta-lenguaje específico de la plataforma ofrece. Trabajar con meta modelos o si es posible meta-meta-modelos, significa que se deben tener en cuenta otra clase de lenguajes para su definición. Estos lenguajes se llaman meta-lenguajes y están definidos según otro metalenguaje. La dificultad de las herramientas aumenta a medida que se definan más capas de metalenguajes para realizar los modelos. Pero queda la inquietud en este trabajo de grado para la investigación de herramientas que automaticen estos procesos a este alto nivel de abstracción.

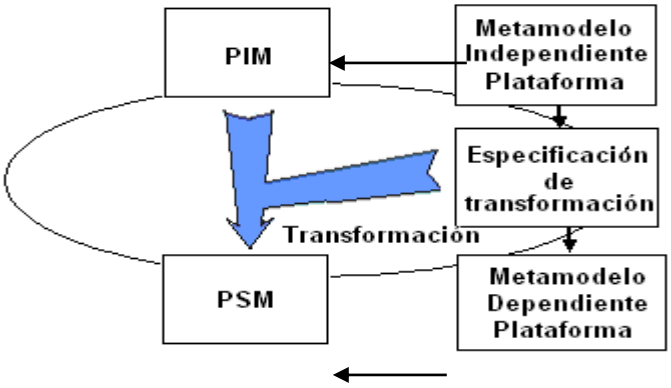


Figura 4.1 Meta-modelo de transformación

4. Las herramientas de transformación que se desarrollen deben permitir la obtención de información en cada etapa o flujo del proceso de desarrollo y además, facilitar al desarrollador la obtención de esta. En este ítem se puede observar la arquitectura que un [IDE] debe tener. Esta arquitectura esta basada en el trabajo de [ARQCON].

   - **Módulo de Modelo de Negocio y Requerimientos**: El flujo de trabajo de especificación requiere de herramientas que fácilmente tomen y manipulen estructuras y flujos de negocio. Las actividades de modelado en esta parte son altamente iterativas. Por esta razón la herramienta debe ayudar al diseñador capturar y estructurar de manera rápida cantidad de información de negocio sin impedir o retrazar las dinámicas sesiones de el grupo de trabajo a cargo.

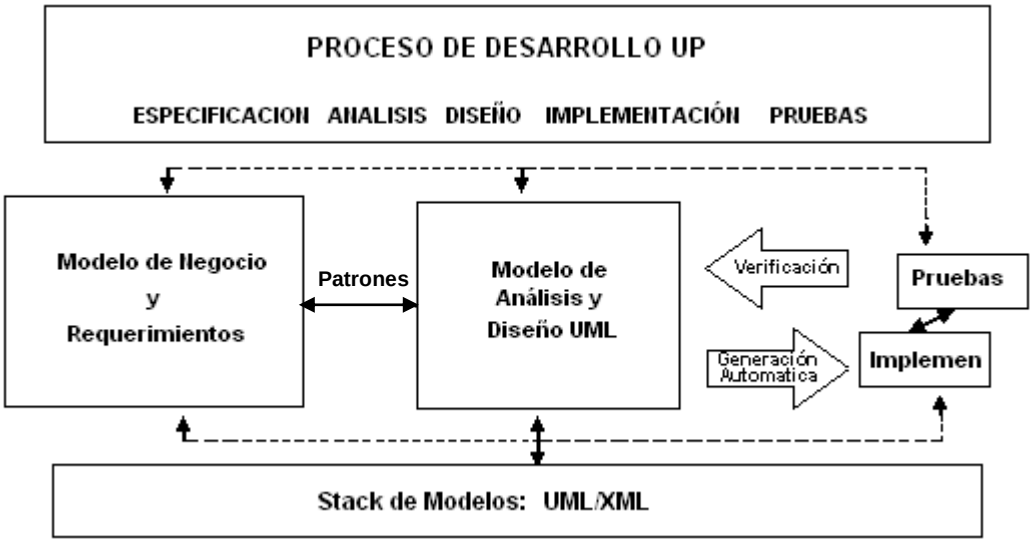


Figura 4.2 Arquitectura de una Herramienta de transformación de modelos o IDE para MDA

- **Módulo de Stack de Modelos:** Se necesita un hilo conductor desde los modelos de negocio a los modelos de diseño, en este caso desde que se empieza a generar el PIM hasta que se llega al PSM, a lo largo de todos los flujos de trabajo. Para soportar esta trazabilidad, toda la información de los modelos debe estar almacenada y definida en un mismo formato (podría ser UML) o estar almacenada en un almacén común de modelos. Este almacén debe ser abierto si se desea interconectar otras herramientas (XMI/XML, open Java API)
- **Módulo de Modelo de Diseño UML:** La creación de los modelos UML tanto en el flujo de análisis como en el de diseño, deberían ser procesados de una manera automática o asistida usando patrones de arquitectura o los estilos arquitecturales. Pero más importante que la automatización es ayudar al desarrollador a refinar los modelos con todos los conceptos de los patrones. Este proceso de soporte debe llegar hasta que el modelo este completo para que se pueda pasar a la generación automática de los aspectos del sistema software que pueden ser representados por UML. La herramienta también debe permitir la verificación automática del modelo generado, de acuerdo con los requerimientos del estilo arquitectural y los requerimientos del entorno de desarrollo objetivo.
- **Módulos de Pruebas e Implementación:** La herramienta debe permitir la configuración de la plataforma específica en la cual se desea trabajar, y además crear nuevas plataformas específicas de acuerdo a sus necesidades de negocio o modificar las existentes según sus necesidades.

- **Generación automática y verificación:** En la parte de generación automática de modelos se debe contar con módulos que permitan cambiar la tecnología con la cual se debe trabajar. Estos módulos pueden ser cartuchos como los que utiliza la herramienta [ArcStyler]. Estos cartuchos deben tener la posibilidad de ser configurados por medio de de la IDE y guardados como parte de la configuración del proyecto en el que se este trabajando.

La línea futura de trabajo para estas herramientas no esta muy lejos, existen herramientas en el mercado como [ECLIPSE] que trabaja con[MOF], lenguaje para desarrollo de metamodelos, la cual contiene módulos para hacer herramientas software de este tipo.

4.4 VENTAJAS DE MDA PARA LAS EMPRESAS DESARROLLADORAS DE SOFTWARE

Los procedimientos tradicionales de las empresas desarrolladoras de software, obligan a los desarrolladores a gastar muy poco tiempo y energía en el diseño de la aplicación, ya que son empresas que deben actuar rápidamente frente a los cambios de tecnologías y a la gran demanda software que existe en el mercado. Debido a esto, los sistemas quedan poco documentados y la reutilización de la lógica del negocio se pierde.

Como MDA separa la lógica del negocio de la implementación, las empresas pueden adaptarse con mayor facilidad a los diferentes cambios, sin tener que invertir mucho en personal desarrollador en la migración a la nueva tecnología, ya que toda la lógica del negocio se encapsula en UML y puede ser aplicada a diversos ambientes de desarrollo tecnológico como J2EE o .Net, dando como resultado una alta reutilización de código y una ventaja competitiva y estratégica ante el mercado cambiante.

Las principales ventajas que brinda MDA al desarrollo de las aplicaciones en una empresa desarrolladora son:

- Portabilidad: Incrementando el re-uso y reduciendo la complejidad del desarrollo de las aplicaciones.
- Interoperabilidad entre plataformas: MDA usa métodos para garantizar que estándares basados en múltiples tecnologías sean implementados con las mismas funcionalidades de negocio.
- Independencia de la plataforma: Se reduce tiempo, costo y complejidad en la transformación a diferentes tecnologías incluyendo las que todavía no están en el mercado.
- Productividad: Permite a los miembros de la empresa utilizar lenguajes entendibles para todos los roles, permitiendo así, más comunicación entre los grupos de trabajo.

- Reducción de costos: El costo de un proyecto desarrollado con el enfoque MDA se reduce de un 40% a un 50 %. Acompañado de una estrategia organizacional orientada hacia la reutilización puede alcanzar una reducción mucho mayor.

4.4.1 MDA y las líneas de productos

Una de las potencialidades más importantes de las empresas desarrolladoras, es el poder ofrecer a sus clientes una serie de productos con características similares. A estas se les conoce como líneas de productos.

Una línea de producto es un conjunto de sistemas intensivos de software con características comunes y gestionables que satisfacen necesidades específicas de un sector del mercado y son desarrollados de forma prescrita a partir de una serie de activos núcleo reutilizables. Estas líneas de producto comparten una arquitectura que es usada para estructurar los activos reutilizables con los que se desarrollan los diferentes productos.

Lo más importante en el contexto de las líneas de producto es la reutilización de elementos, pero además, estas necesitan un proceso de desarrollo que brinde bidireccionalidad entre la ingeniería de dominio y la aplicación, de este modo, con solo cambiar las características del dominio en donde estas líneas de producto se utilicen, se generaría de una manera más eficiente el aspecto técnico que se esté solicitando. Es aquí donde MDA cobra mucha importancia, puesto que como se ha visto en este trabajo de grado, este marco de trabajo proporciona las características que se necesitan para la reutilización del dominio de la aplicación y la generación de sistemas en diferentes plataformas.

4.4.2 DSCB, MDA y las líneas de producto

El concepto de desarrollo de sistemas de software basado en componentes, o simplemente "desarrollo de software basado en componentes" [DSBC] es un paradigma de desarrollo software que utiliza una serie de técnicas para la elaboración de sistemas abiertos y distribuidos mediante el ensamblaje de partes software reutilizables llamadas componentes. Un componente es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un
conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio [Szyperski, Pfister,1997].

En la figura 4.3 se muestra la arquitectura basada en DSCB:

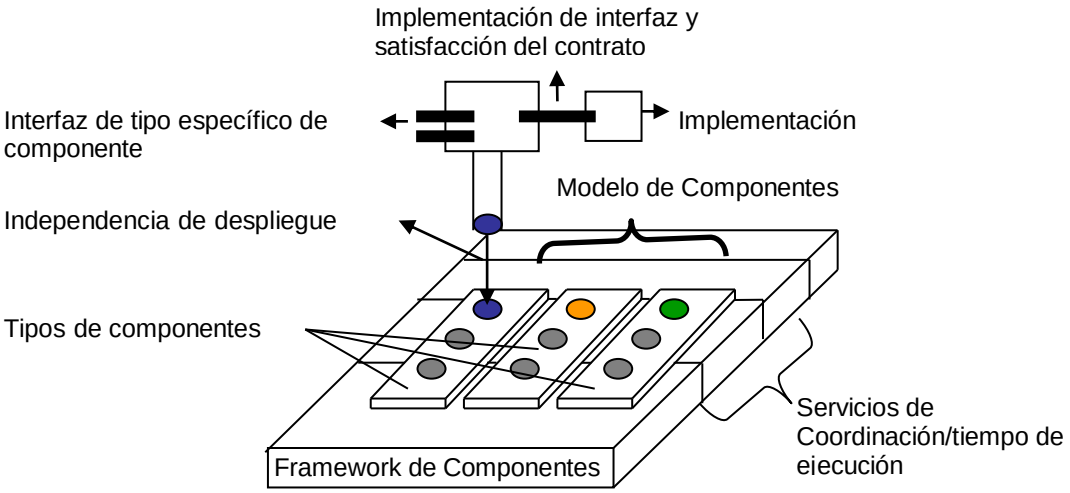


Figura 4.3 DSBC Tomado de [CMU/SEI-2000-TR-008]

Los componentes residen en un framework el cual cuenta con una serie de servicios que son utilizados por estos componentes software. Las funciones de los componentes se acceden únicamente por medio de las interfaces que cada componente posee.

Con base en lo anterior, una empresa puede realizar su propia plataforma específica de características y elementos reutilizables para su línea de productos, la cual podría ser montada sobre una plataforma middleware. Esta plataforma se convertiría desde la perspectiva MDA, como la plataforma específica a la cual se va a llevar el o los modelos PIM de la empresa. La plataforma que se propone se ve en la figura 4.4.

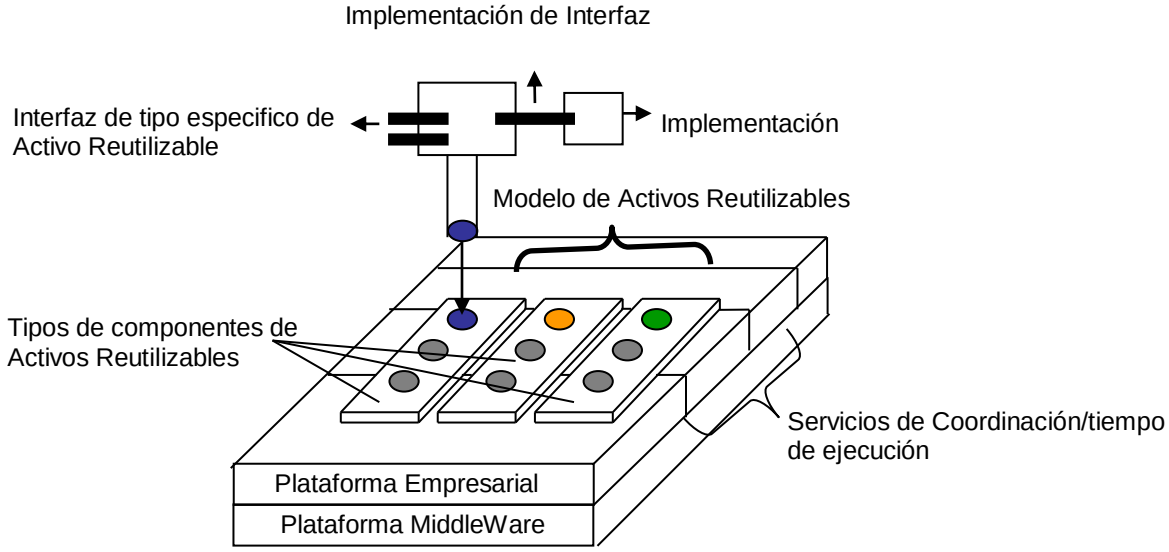


Figura 4.4 Plataforma Empresarial

La nueva plataforma que en este trabajo de grado toma el nombre de Plataforma Empresarial y está soportada por plataformas específicas como J2EE, .Net. Corba, entre otras.

El modelo independiente de la computación consiste para este caso, en un conjunto de características de negocio y unos atributos de calidad exclusivos para cada línea de productos que se desee implementar. Realizar los modelos y las transformaciones para la plataforma empresarial no es un asunto trivial, ya que se debe trabajar con MOF para realizar los meta-modelos los cuales tienen información acerca de cómo estos modelos deben ser implementados, y que relaciones existen entre los objetos creados en los nuevos modelos. Una vez obtenidos los meta-modelos, los modelos PIM y PSM pueden ser generados de acuerdo a necesidades de negocio específicas.

Esta nueva plataforma ofrece gran potencialidad en la producción de aplicaciones basadas en características de negocio específicas que pueden ser desarrolladas bajo el UP basado en MDA, el cual tendrá en cuenta una serie de patrones y cualidades del servicio más particulares para la empresa.

5. CONCLUSIONES

1. MDA es un enfoque que aun se encuentra en una etapa muy inmadura con respecto a la creación de los primeros modelos que son la base de MDA (CIM y PIM), es por esto es necesario que sea complementado con procesos en los cuales se tenga más experiencia como lo es UP. En este trabajo de grado se unieron los conceptos de MDA y UP para solucionar la falta de especificaciones de la captura de requerimientos para la creación del CIM y su posterior transformación a PIM en MDA. Esta transformación fue llevada a cabo utilizando algunas de las reglas propuestas por diferentes autores de MDA, además de estas fue necesario proponer nuevas reglas de transformación que surgieron de la experiencia del desarrollo del Sistema de Gestión Documental. Es necesario tener en cuenta que las especificaciones de los lenguajes de las reglas de transformación aun se encuentran en etapa de desarrollo, y por esto las reglas de transformación que surgieron se encuentran en texto plano.

2. MDA se encuentra, según el OMG, en un nivel de madurez de grado 4/5 en el cual los modelos deben ser acompañados de un lenguaje que defina la parte dinámica del sistema, este lenguaje es OCL. Para el desarrollo de esta aplicación no se tiene en cuenta este lenguaje ya que MDA aun no define reglas de transformación para restricciones definidas en OCL y ningún autor hace posibles sugerencias que se puedan usar en un desarrollo. Sin embargo para el crecimiento de MDA es necesario que futuros proyectos incorporen OCL, MOF y QVT estándares que aun no han sido terminados pero que permitirán crear el código de manera más automática desde los modelos MDA.

3. MDA es un enfoque que ofrece grandes ventajas sobre otros procesos de desarrollo que se puedan estar usando en la actualidad. La mayor ventaja de MDA es la característica que tiene de ser independiente de la plataforma (J2EE, CORBA, Plataformas propietarias), permitiendo así el reuso de modelos que necesitan ser aplicados a plataformas heterogéneas e incluso plataformas definidas por los clientes o en sistemas con características lógicas y de negocio similares, por otro lado, el tiempo que se invierte en un proceso desarrollado con MDA es mucho mas bajo que con un proceso tradicional como lo es UP, es por esto que MDA tarde o temprano será la opción definitiva para empresas desarrolladoras de software que desean implementar sus proyectos en poco tiempo y con

muy buena calidad, o para empresas que necesiten desarrollar soluciones internas de una manera ágil.

4. MDA necesita de cierto grado de conocimiento para ser utilizado. Este enfoque trabaja con transformaciones par obtener los diferentes modelos, sin embargo el tipo de transformación con base en meta-Modelos es muy complejo y al tomar este como opción puede incurrirse en un esfuerzo innecesario en caso de tener una empresa con un equipo de desarrollo pequeño. MDA es una buena opción si se trabaja con transformaciones a base de marcas y patrones (como se hizo en este proyecto) y en caso de empresas con equipos de trabajo especializados y producción de software en gran cantidad, la transformación a partir de meta-modelos es la mejor opción a pesar de su complejidad.

5. Un elemento crucial en el desarrollo de aplicaciones con MDA, es el uso de herramientas que generen el código automáticamente a partir de los modelos que se obtienen en el proceso, simplificando de gran forma el uso de meta-modelos o las marcas según sea el caso. Sin embargo, es necesario que se entienda que estas no son la única parte ni la más importante de este enfoque aunque disminuyan el trabajo, es necesario un buen proceso de desarrollo para obtener los requerimientos de una forma correcta y no exista ambigüedad en ningún parámetro por irrelevante que pueda parecer. Además no se puede llegar a pensar que una herramienta eliminará completamente la necesidad de personal que esté en capacidad de manejar diferentes tecnologías de implementación (JAVA, .NET, C++), por el contrario, se necesitaran personas que además de manejar todas estas tecnologías, también tengan conocimientos acerca de las transformaciones que crearon el código que estan utilizando y en caso de alguna modificación debido a que se necesite algún otro patrón de desarrollo por implementación de algún componente dinámico, el desarrollador pueda sustentar el cambio y ayudar a la actualización de los modelos superiores.

6. MDA es un enfoque bastante reciente, es por esto que a pesar de tener muchas empresas apoyándolo, algunos de los aspectos mas importantes no han sido abordados, tal es el caso de la captura de requerimientos. Debido a que MDA se basa en los requerimientos que se obtuvieron del negocio, es de esperar que el proceso para la obtención de estos, este muy bien documentado, sin embargo, en toda la literatura consultada, no se especifica como hacerlo, es por esto que se tomó UP como proceso de desarrollo para así aplicar el concepto de MDA a UP y aprovechar todas las ventajas de ambos, teniendo así un primer acercamiento al trabajo con MDA, y los pasos que se deben seguir a partir de los lineamientos que recomienda UP.

7. Para empresas desarrolladores de software, esto es un punto muy critico, ya que a pesar de que se tiene un una recomendación para trabajar con MDA (dada a lo largo de este proyecto), este enfoque todavía es experimental, así que se debe evaluar muy cuidadosamente si MDA es la solución adecuada en este momento para la generación de aplicaciones, y no caer en el error, al creer que una herramienta por poderosa que sea, pueda dar solución a este problema.

8. J2EE es una plataforma de desarrollo muy potente que ofrece una alta escalabilidad y permite que el desarrollo se haga de una forma ordenada. Los patrones de diseño que se han desarrollado con base en esta tecnología, mejoran tanto el desempeño como el orden de las aplicaciones. Sin embargo, no se debe caer en el error de aplicar todos los patrones a una misma aplicación ya que en algunos casos se daría una sobredimensión innecesaria al proyecto y se incurriría en el consumo de tiempo y esfuerzo que no se vería reflejado como mejora en el prototipo final. Es por esto que se debe tener muy en claro cuales de estos patrones que se van a utilizar desde el principio. Finalmente, es necesario tener el concepto claro de cada uno de los patrones y en qué se incurre al ejecutarlo, tanto para saber que repercusiones trae como para poder llevar a cabo una transformación consistente que permita al desarrollador tener una correspondencia precisa entre los modelos existentes y el código generado.
9. La certificación en calidad ISO 9001-2000 es una gran oportunidad para las empresas para mejorar sus procesos, y en este caso en particular para organizar y agilizar el acceso a una parte crucial de su documentación. Pero además de esto, si una empresa desea tener control total sobre los procesos de su empresa, licitación de proyectos, facturación, compra de equipos, e incluso la generación de su propia documentación, resultaría interesante utilizar el concepto de “Workflow” para el control de flujos en la empresa. Integrando ese concepto y una plataforma tan potente como lo es J2EE, el abanico de funcionalidades que se puede obtener es muy grande. Para trabajar con este concepto de “Workflow” la plataforma XFLOW [**http://xflow.sourceforge.net**] basa su funcionalidad en el uso de los beans de mensaje de la plataforma J2EE, permitiendo la colaboración entre flujos de trabajo, ver el estado de los flujos de trabajo en cualquier instante, trabajar con flujos paralelos, entre otras funcionalidades. Además utiliza el JBOSS como contenedor de EJB, disminuyendo así los costos en los que pueda incurrir una empresa.

BIBLIOGRAFIA

[ARQPRA] Len Bas, Paul Clements, Rick Kazman. *Software Architecture in Practice*. Boston, United States. Pearson Education. 2003.

[MDAEX] Anneke Kleppe, Jos Warmer, Wim Bast. MDA Explained, *The Model Driven Architecture: Practice and Promise. Boston*, United States: Addison –Wesley. 2003.

[ARQCON] Richard Hubert, *Convergent Architecture*. New Cork, United Status: Wiley Computer Publishing. 2002.

[J2EE] Sun Microsystems. *Java 2 Enterprise Edition Pataform Specification*. 1.3 2001 *http://java.sun.com/j2ee/index.jsp*

[EJB] Sun Microsystems. *Enterprise JavaBeans Specification*. Versión 1.1

[MDAGV1] Jon Siegel, Richard Soley, Andrew Watson, y otros. MDA Guide Version 1.0.1. 2003

[JBRU] Jacobson, G. Booch J., Rumbaugh. *El lenguaje Unificado de Modelado. Manual de Referencia*. Adison Wesley. 1999

DSBC] Luis F Iribarne Martinez, *Capitulo 1, Tesis de Doctorado Un modelo de Mediación para el desarrollo Software Basado en Componentes* Universidad de Malaga 2003

[UML] OMG. “UML Summary. V1.1”. UML *Specification Document* ad/97-08-05

[Szyperski, Pfister,1997] Szyperski C. (1997), *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley Longman

[JBuilder] *http://www.borland.com/jbuilder*

[XP] *http://es.wikipedia.org*

[ISOGUI] *Sistema de Gestión de calidad ISO 9000-2000 Guía explicativa de los requisitos* Edición Instituto Colombiano de normas técnicas y certificación ICONTEC.

[ARCSTYLER] http://*www.ArcStyler.com*

[COMPUWARE] *http://www.OtimalJ.com*

[OMG] http://*www.org.com*

[Desarrollo de Software basado en Componentes en la Plataforma J2EE Ing. Julio Ariel Hurtado, Ing. Lina María Castillo Paredes]

[Propuesta de en Proceso para el Análisis de una Línea de Productos dentro del Contexto del Proceso ARCALP Ing. Julio Ariel Hurtado, Ing. Juan Carlos Vidal]

[Definición y representación de reglas de transformación de un modelo independiente en otro específico de la plataforma J. A. Ortega , J. M. Márquez, A. Reina y J. A. Álvarez. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. Avda. Reina]

[Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach *Productivity Analysis June 2003 MDA PRODUCTIVITY CASE STUDY COPYRIGHT © 2003 THE MIDDLEWARE COMPANY, casestudy@middleware-company.com]*

[Developing in OMG's New Model Driven Architecture OMG Masaharu Obayashi san of CBOP/Kanrikogaku Ltd. and OMG CEO and Chairman Richard Soley ,Toronto TC Meeting]

[MDA: An Idea Whose Time Has Come Paul Harmon, Senior Consultant and Market Analyst CUTTER CONSORTIUM]

[Capitulo1, Software Architecture and Product Lines © Pearson Education 1999 (Draft version)]

ACRONIMOS

APV: Área de Proyectos y ventas.

BMP: Bean Managed Persistence.

CIM: Modelo Independiente de la Computación

CMP: Container Managed Persistence.

CORBA: Common Object Request Broker Architecture.

CWN: Common Warehouse Metamodel.

DCOM: Distributed Component Object Model.

DIAN: Dirección de Impuestos y Aduanas Nacionales.

EJB: Enterprise Java Bean.

IDE: Integrated Development Enviroment.

IIOP: Internet Inter-ORB Protocol.

ISO: Internacional Standard Organization.

J2EE: Java 2 platform Enterprise Edition.

MDA: Model Driven Architecture

MOF: Meta-Object Facility.

OMG: Object Managment Group.

PIM: Modelo Independiente de la Plataforma.

PSM: Modelo Dependiente de la Plataforma.

RMI: Remote Method Invocation.

SOAP: Simple Object Access Protocol.

SQL: Structured Query Language.

UDDI: Universal Description, Discovery and Integration.

UML: Unified Modeling Language.

UP: Unified Process.

WSDL: Web Services Description Language.

XMI: XML Interchage Metadata.

XML: Extensible Markup Language.

GLOSARIO

.NET: Plataforma que involucra aplicaciones, herramientas y servicios para el desarrollo de aplicaciones.

BMP: Forma de manejo de los Bean entidad en J2EE a través del bean.

APV: Área de Proyectos y ventas de la Empresa Gamma Ingenieros S.A.

Broadcast: Envió simultaneo de mensajes a múltiples receptores

CMP: Forma de manejo de los Bean entidad en J2EE a través del contenedor de componentes.

Contabilidad: Sistema para llevar las cuentas de una empresa o entidad. Departamento de la Empresa Gamma Ingenieros encargado de llevar el estado de cuentas de la empresa.

CORBA: Arquitectura que permite, partes de un programa llamadas objetos comunicarse entre si sin importar el lenguaje en que se encuentren o el sistema operativo donde estén.

CWM: Interfaces estándar que permiten el intercambio de warehouse y metadata.

DCOM: Conjunto de protocolos que permite la comunicación de componentes COM (Component Object Model) a través de la red.

DIAN: Dirección de Impuestos y Aduanas Nacionales. Entidad destinada a prestar un servicio de facilitación y control a los agentes económicos, para el cumplimiento de las normas que integran el Sistema Tributario, Aduanero y Cambiario, obedeciendo los principios constitucionales de la función administrativa, con el fin de recaudar la cantidad correcta de tributos, agilizar las operaciones de comercio exterior, propiciar condiciones de competencia leal, proveer información confiable y oportuna, y contribuir al bienestar social y económico de los colombianos.

EJB: API de Java desarrollado por Sun Microsystems que define una arquitectura de componentes para sistemas multi-capas Cliente/Servidor. Con este nombre se denota también los componentes con que trabaja la plataforma J2EE

Gerente: Persona que dirige y administra una sociedad mercantil

Hacker: Termino que define a la persona que disfruta aprender de lenguajes de programación, sistemas computacionales y puede ser considerado un experto en el tema. En el ámbito profesional este término puede ser usado de forma peyorativa para referirse a una persona que atenta contra la integridad de los sistemas computacionales.

IDE: Integrated Development Enviroment. Herramienta de desarrollo Software como por ejemplo OptimalJ, JBuilder, Eclypse.

IIOP: Protocolo desarrollado por la OMG que implementa soluciones CORBA sobre la WWW permitiendo a los navegadores y servidores el intercambio de objetos.

Inventario: Relación detallada de bienes o pertenencias

ISO: Organización Internacional de Estandarización. Entidad desarrolladora de estándares tanto para elementos como para procesos.

J2EE: La plataforma J2EE consiste en un grupo de servicios, APIs y protocolos que proveen funcionalidad para el desarrollo de aplicaciones Web multicapas.

Licitacion: Oferta que se hace en una subasta o en un concurso público, sobre todo si se trata de un contrato o servicio

Maquina Virtual: Ambiente de operación autocontenido que se comporta como si estuviera separado del computador

Metadata: Información que describe como, cuando y por quien un grupo de información fue adquirido y como fue almacenado

Metamodelo: Un metamodelo es una definición precisa de las reglas y estructura necesaria para crear modelos semánticos

Middleware: Software que reside entre un programa o una aplicación y la red. Este maneja la interacción entre varias aplicaciones que se encuentran implementadas sobre diferentes plataformas.

MOF Grupo de Interfaces estándar que pueden ser usadas para definir y manipular meta-modelos con sus correspondientes modelos. Los estándares basados en MOF son usados para la integración de herramientas, aplicaciones y datos.

OMG: Consorcio sin ánimo de lucro que produce y mantiene especificaciones para garantizar interoperabilidad entre aplicaciones empresariales.

Patrones de Diseño: Formas de solucionar un problema recurrente en una plataforma estándar.

Poliza: Documento justificativo del contrato de seguros, operaciones de bolsa, etc.: presentó la póliza del seguro para cobrar la indemnización

Proyecto: Actividades y elementos relacionados con la realización de un contrato para el cliente de Gamma Ingenieros S.A.

RMI: Conjunto de protocolos desarrollados por JavaSoft que permite a los objetos Java comunicarse remotamente entre si.

SOAP: Protocolo de mensajeria basado en XML usado para codificar información en una petición o respuesta de un Web Service antes de ser enviado por la red.

SQL: Lenguaje estándar para el manejo de información en una base de datos.

UDDI: Un directorio distribuido basado en Web que permite a los negocios listarse en la red y ver la información de otros.

UML: Lenguaje notacional para especificar y visualizar software complejo.

UP: Proceso de desarrollo que organiza la creación de software en fases.

WareHouse: Colección de información destinada a ayudar en la toma de decisiones empresariales.

WorkFlow: Serie de tareas dentro de una organización que producen un resultado final

WSDL: Un lenguaje basado en XML usado para describir las funciones de un Web Service como colecciones de puntos de comunicación capaz de intercambiar mensajes. WSDL es el lenguaje que usa UDDI.

XMI: Aplicación XML que facilita la estandarización del intercambio de los modelos y de la metadata a través de Internet.

XML: Especificación diseñada específicamente para documentos Web. Permite a los diseñadores crear sus propias etiquetas permitiendo la definición, transmisión, validación e interpretación de información entre aplicaciones y organizaciones.