UNIVERSIDAD NACIONAL DE INGENIERÍA
RECINTO UNIVERSITARIO "SIMÓN BOLÍVAR"
FACULTAD DE ELECTROTECNIA Y COMPUTACIÓN

TRABAJO MONOGRÁFICO

Estándares de la Arquitectura Orientada a Modelos (MDA)

Diseño de una Metodología para Análisis y Diseño de Software

PARA OPTAR AL TÍTULO DE
INGENIERO EN COMPUTACIÓN

ELABORADO POR:
Br. Gabriel Rafael Lacayo Saballos
Br. Roberto Enrique Silva Navas

TUTOR:
Lic. Magda Luna Molina

MANAGUA, NICARAGUA
ABRIL 2011

Dedicatoria

A nuestra tutora la Lic. Magda Luna que siempre mostró su actitud positiva y de colaboración al trabajar junto a nosotros y poder guiarnos por el camino a la finalización de este trabajo Monográfico.

A todas las personas que nos brindaron su apoyo en este largo y difícil camino.

A todos ellos, nuestro eterno agradecimiento!!...

Agradecimientos

A mi Madre Amalia Teresa Saballos Gómez que siempre creyó en mí. Y que con su apoyo incondicional, que sólo una madre puede dar, he podido alcanzar esta meta tan importante en mi vida académica.

Gabriel Lacayo Saballos.

Gracias a Dios, a mi padre Enrique Silva por su grandísimo apoyo en el camino hacia esta meta, por siempre ser excelente padre, mentor y amigo. A mi madre Cristina Navas por estar siempre conmigo y ayudarme a perseverar hacia este logro, A Cris por su incondicional compañía y soporte en este camino y a toda mi familia por todo el apoyo brindado en esta etapa de mi vida tan importante. ¡Muchas gracias!

Roberto Silva Navas

Resumen

Model Driven Architecture (MDA) surgió hace casi una década cómo un nuevo enfoque estructural para el desarrollo de software, tiempo suficiente para poder hacer un análisis sobre el alcance de la misma y sus estándares, así también como la experimentación con algunas herramientas de la industria. Con este objetivo se ha hecho este estudio sobre el enfoque MDA y sus estándares, que han permitido extraer los conceptos más importantes para integrarlos a una metodología de selección de estándares y aplicarla al ciclo de vida del software.

En el estudio se plasma el papel fundamental de los modelos en el desarrollo de software para potenciar la reutilización de los diferentes elementos del software y facilitar la labor de los diversos roles que participan del proceso. La Arquitectura Dirigida por Modelos (MDA) propone un proceso de desarrollo basado en la realización y transformación de modelos. Los principios en los que se fundamenta MDA son la abstracción, la automatización y la estandarización. El proceso central de MDA es la transformación de modelos que parten del espacio del problema (CIM) hasta modelos específicos de la plataforma (PSM), pasando por modelos que describen una solución independientemente de la computación (PIM).

En el desarrollo de software dirigido por modelos, las transformaciones de modelos son consideradas como activos importantes que deben ser manejadas con principios sólidos de ingeniería de software: estas transformaciones deben ser analizadas, diseñadas, implementadas, probadas, mantenidas y sujetas a la administración de configuración. Debido a esto existe la necesidad de identificar los modelos y sus características, técnicas y estándares que permitan un desarrollo basado en el enfoque MDA de calidad.

En este trabajo se hace un especial hincapié en la arquitectura de metamodelado de cuatro capas definidas por el grupo de trabajo OMG. Se estudian los estándares más importantes definidos por OMG en el contexto de MDA detallando sus objetivos, su estructura y su definición, situando cada uno de ellos en la arquitectura MDA.

Otro punto de vista desarrollado ha sido la realización de pruebas con determinadas herramientas, sobre la base de un conjunto de características específicas de la aproximación MDA. La finalidad de esto es conocer y comprobar el funcionamiento de la arquitectura, los estándares y la teoría correspondiente. Las herramientas seleccionadas han sido: *AndroMDA y OpenMDX* (ambas open source).

El aporte en este trabajo es el diseño de una metodología para la selección de estándares dentro del ciclo de vida del software usando el enfoque MDA, así como la identificación de técnicas y herramientas que colaboren a mejorar la calidad del desarrollo de software. Este aporte constituirá una guía para el desarrollador de software permitiéndole comprender los conceptos del enfoque MDA, sus principales diferencias respecto al desarrollo tradicional y brindará un conjunto de actividades y tareas sistematizadas en las cuales se pueden aplicar los estándares del enfoque MDA.

Definiciones y Acrónimos

Metodología: conjunto de procedimientos basados en principios lógicos, utilizados para alcanzar una gama de objetivos que rigen en una investigación científica o en una exposición doctrinal. [1]

Metodología de desarrollo de software: Una metodología de desarrollo de software se refiere a un framework que es usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información. [2]

Arquitectura: La arquitectura de un sistema es una especificación de las partes y los conectores del sistema y las reglas para la interacción de las partes usando los conectores. [3]

Componente: Una de las partes que componen un sistema. Un componente puede ser hardware o software y puede subdividirse en otros componentes. Nota: Los términos "módulo", "componente" y "unidad" a menudo se utilizan indistintamente o definidos como sub-elementos entre sí de diferentes maneras dependiendo del contexto. [8]

Estudio de Viabilidad del sistema: Para este trabajo se consideran dentro del estudio de viabilidad del sistema los siguientes aspectos: Descripción de la solución, Catálogo de requisitos, normas y usuarios.

Validación: El proceso de evaluación de un sistema o componente durante o al final del proceso de desarrollo para determinar si cumple los requisitos especificados. [8]

Verificación: El proceso de evaluación de un sistema o componente para determinar si los productos de una determinada fase de desarrollo reúnen las condiciones fijadas en el comienzo de esa fase. [8]

Implementación: El proceso de traducción de un diseño en componentes de hardware, componentes de software, o ambos. [8]

Fase de Implementación: El período de tiempo en el ciclo de vida del software en el que se crea un producto de software a partir de la documentación de diseño y es depurado. [8] Cabe destacar que para el enfoque MDA la codificación en esta etapa es generada por herramientas automatizadas a partir de los modelos PSM construidos.

Fase de Instalación y Pruebas: El período de tiempo en el ciclo de vida del software en el que se integra un producto de software en su entorno operativo y es

probado en este entorno para asegurarse de que se desempeña según lo requerido. [8]

Fase de Explotación (operación y mantenimiento): El proceso de modificación de un sistema o componente de software después de la entrega, para corregir fallas, mejorar el rendimiento u otros atributos, o adaptar a un entorno cambiado. [8]

Sistema de Información: Componentes interrelacionados para reunir, procesar, almacenar y distribuir información para apoyar la toma de decisiones, la coordinación, el control, el análisis y la visualización de una organización. [7]

Sistema: Para efectos de este trabajo se asume “Sistema” como abreviación de “Sistema de Información”, heredando la misma definición.

Software: Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de información. [8]

OMG: El **Object Management Group** u **OMG** (de sus siglas en inglés *Grupo de Gestión de Objetos*) es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA, MDA. Es una organización sin ánimo de lucro que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones para las mismas. El grupo está formado por compañías y organizaciones de software como: Hewlett-Packard (HP), IBM, Sun Microsystems y Apple Computer. [4]

Modelo: Un modelo representa algo concreto o abstracto de interés, con un propósito específico en mente. El modelo tiene que ver con la cosa por un isomorfismo explícita o implícita. Modelos en el contexto del MDA FoundationModel son instancias de metamodelos MOF, por lo que consisten en los elementos del modelo y los vínculos entre ellos. [3]

Modelo de dominio: Abstracción de lo que es una empresa y cómo distribuye un producto o servicio; también muestra como crea riqueza la empresa. [7]

Meta datos: Los datos que representan los modelos. Por ejemplo, un modelo de UML, un modelo de objetos de CORBA, y un esquema de base de datos relacional expresado utilizando CWM. [3]

Meta modelo: Un tipo especial de modelo que especifica la sintaxis abstracta de un lenguaje de modelado. Un lenguaje abstracto de algún tipo de metadatos. [3]

MDD: Una aproximación al desarrollo de software en el que el foco y componentes principales del desarrollo son los modelos (en oposición a los programas y código) y las transformaciones de modelos. [5]

MDA: Model-Driven Architecture o MDA (La arquitectura orientada a modelos). Un enfoque en cuanto a las especificaciones del sistema que separa la especificación de la funcionalidad de la especificación de la aplicación de esa funcionalidad en una plataforma tecnológica específica. Propuesta de aplicación de MDD por parte de OMG. [3]

CIM: Computation Independent Model (Modelo independiente de computación) [3]

PIM: Platform Independent Model (Modelo independiente de plataforma). [3]

PSM: Platform Specific Model (Modelo específico de plataforma). [3]

XML: eXtensible Markup Language. Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). [3]

Estándares MDA

UML: Un lenguaje estándar de OMG para especificar la estructura y comportamiento de los sistemas. La norma define una sintaxis abstracta y una sintaxis concreta gráfica.[3]

Meta-ObjectFacility (MOF): Una norma de OMG, estrechamente vinculada a UML, que permite la gestión de metadatos y la definición de lenguaje de modelado. [3]

QVT: MOF Query / Views / Transformation. Estándar para la transformación de modelos. [4]

Object Constraint Language (OCL): Lenguaje formal de restricción, especificado como parte de UML, permitiendo la imposición de condiciones previas y las condiciones post, y la especificación de los invariantes y demás condiciones de una invocación. [3]

XML Metadata Interchange (XMI): La especificación de OMG, expresada como DTD XML y esquemas, más normas de producción de DTD XML y esquemas para el intercambio de meta modelos y modelos como documentos XML. [4]

Java Metadata Interface (JMI): La especificación JMI permite la implementación de una infraestructura dinámica, independiente de la plataforma, para gestionar la creación, almacenamiento, acceso,

descubrimiento, y el intercambio de metadatos. JMI se basa en la especificación de Meta ObjectFacility (MOF) del Object Management Group (OMG). [6]

Perfiles UML: Un conjunto estandarizado de limitaciones y extensiones que adaptan UML a una plataforma en particular, o entorno de aplicación. [4]

Document Type Definitions (DTD): Es un conjunto de declaraciones de marcas que definen un tipo de documento para lenguajes de marcado SGML de la familia (SGML, XML, HTML). [9]

Common Warehouse Metamodel (CWM): Una especificación de OMG para la integración de datos de un repositorio. [3]

METRICA Versión 3: Se autodefine como una "metodología para la sistematización de las actividades que dan soporte al ciclo de vida del software". Es un trabajo de distribución y uso libre, realizado por el Ministerio de Administraciones Públicas Español [28].

Ingeniería de Software (IS): El estudio de enfoques sistemáticos, disciplinados y cuantificables para el desarrollo, operación y mantenimiento de software. [8]

Ciclo de vida del software: El período de tiempo que comienza cuando un producto de software es concebido y termina cuando el software ya no está disponible para su uso. El ciclo de vida del software normalmente incluye una fase de concepto, viabilidad y requerimientos, análisis, la fase de diseño, fase de ejecución, fase de prueba, instalación y fase de salida, funcionamiento y fase de mantenimiento, y, a veces, la fase de jubilación. Nota: Estas fases pueden superponerse o realizar iterativamente. [8].

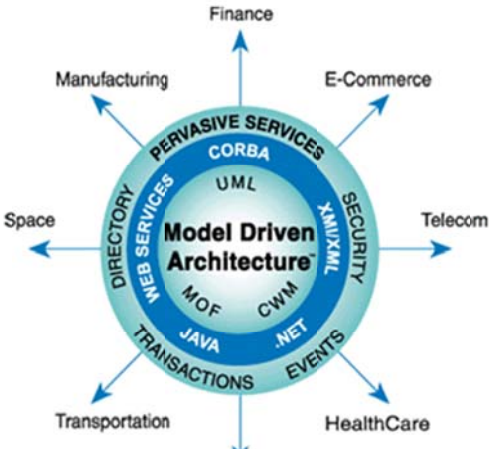
Índice

1. Aspectos Metodológicos....3
1.1. Introducción. 3
1.2. Objetivos 4
1.2.1. Objetivo General 4
1.2.2. Objetivos Específicos 4
1.3. Justificación 5
1.4 Diseño Metodológico 6
1.4.1 Tipo de Investigación 6
1.4.2 Hipótesis 6
2. Marco Teórico....7
2.1 Estándares del Enfoque MDA 7
2.2. El ciclo de vida de desarrollo de sistemas MDA. 14
2.2.1. Contexto 14
2.2.2. Identificación de las necesidades y oportunidades. 14
2.2.3. Análisis, Diseño y Código 14
2.2.4. Puesta en producción. 17
2.3. Visión alternativa de MDA 17
2.4 Análisis de las herramientas MDA 19
2.4.1 Criterios para evaluar las herramientas MDA. 20
2.5 AndroMDA 22
2.5.1 Aplicación de AndroMDA 22
2.5.2 Evaluación de AndroMDA 23
2.6 OpenMDX 25
2.6.1 Aplicación deOpenMDX 26
2.6.2. Evaluación de OpenMDX 26
2.7. Resultados de la Evaluación de las Herramientas MDA. 28
3. Diseño de una Metodología para la selección de estándares MDA. 29
3.1. Necesidad de una Metodología 29
3.2. Definición de la Metodología. 30
3.3. Fundamentación para la Selección de Estándares usados en la Metodología. 32

3.3.1 Estándares OMG y la arquitectura de metamodelado de cuatro capas. 33
3.4 Etapas de Análisis y Diseño para la Selección de estándares MDA 37
3.4.1 Análisis de Sistemas de Información 37
3.4.2 Diseño del sistema de información 55
4. Análisis y Presentación de los Resultados. 73
5. Conclusiones y Recomendaciones 74
7. Referencias 76
Anexos I
Anexo A: Guía de Instalación AndroMDA. I
Anexo B: Guía de Instalación OpenMDX. XII

1. Aspectos Metodológicos

1.1. Introducción.



El desarrollo de software se enfrenta a continuos cambios en las tecnologías de implementación, lo que implica realizar esfuerzos importantes en el análisis y diseño de las aplicaciones para poder integrar estas diferentes tecnologías. Un ejemplo de ello es en el mantenimiento del software, donde se necesita adaptar la aplicación a cambios por requerimientos de los usuarios a partir de las plataformas tecnológicas que van surgiendo.

El mercado de herramientas es actualmente activo, pero deben pasar todavía algunos años para explotar en mejor nivel las posibilidades del desarrollo de software basado en modelos.

El uso de modelos en las áreas del desarrollo de software ha venido teniendo una gran aprobación. Hoy en día es muy común ver en la Ingeniería de software (IS) el uso de modelos como herramientas principales para desarrollar software; debido a esto se observa una tendencia sobre el enfoque basado en modelo. A esta estrategia se le conoce como *Model Driven Development*.

Model Driven Development (MDD) es desarrollo de software basado en la creación y generación de modelos a partir de las transformaciones entre ellos. Por tanto, proporciona una estrategia genérica a seguir para dicho desarrollo, pero no define: técnicas a utilizar, fases del proceso, así como ningún tipo de guía metodológica [10].

Existen diferentes enfoques que aplican el MDD tales como: MDA (Model Driven Architecture), Computo de Modelo Integrado (MCI), entre otros. Para fines del enfoque de este trabajo Monográfico utilizaremos el enfoque MDA, sus estándares y herramientas de trabajo.

Este trabajo monográfico abre las puertas a estudios posteriores sobre el enfoque MDA, para lograr mejores prácticas de los diferentes estándares. Las actividades y tareas propuestas por la Metodología de Selección de Estándares se encuentran más en la línea de un desarrollo orientado a objetos, separando datos y procesos. Para ello se ha analizado alguna de las propuestas de las metodologías orientadas a objetos y se han tenido en cuenta la mayoría de las técnicas que contempla UML 2.0 (Unified Modeling Language) [11] y las herramientas que lo contemplan.

1.2. Objetivos

1.2.1. Objetivo General

Diseñar una metodología para la selección de estándares basados en el enfoque MDA, que permita mejorar los procesos de desarrollo de software.

1.2.2. Objetivos Específicos

1. Evaluar los estándares UML, MOF, QVT, OCL, XMI aplicados en las etapas de análisis y diseño en el proceso de desarrollo de software con MDA.
2. Declarar los fundamentos aplicados en la selección de los estándares.
3. Realizar pruebas con herramientas AndroMDA y OpenMDX que implementan los estándares MDA en las etapas de Análisis y Diseño.
4. Analizar estrategias para diseño de una metodología.
5. Definir los elementos de la metodología a crear.

1.3. Justificación

El enfoque de desarrollo de software MDA (Model Driven Architecture) considera un conjunto de herramientas para especificar un sistema independiente de la plataforma de desarrollo o implementación.

Actualmente existe una variedad de estándares definidos, pero muchos desarrolladores de software no conocen a cabalidad el abanico de herramientas que ofrecen estos estándares de trabajo MDA, ya que los modelos pueden representar elementos estructurales o dinámicos en sí mismos y por otro lado se puede estar hablando de diferentes diagramas que representan estos mismos aspectos y que, en su conjunto, representan el modelo completo del sistema. Al no tener un dominio de los estándares y cómo aplicarlos, los desarrolladores se enfrascan en una sola herramienta haciendo el trabajo basado en modelo, “más tedioso y complicado” o simplemente escogen una metodología de desarrollo tradicional.

El estudio propuesto apunta a brindar a los desarrolladores e ingenieros de software una guía para la selección de estándares cuando apliquen el enfoque MDA, con el propósito de que por medio de su práctica puedan mejorar la calidad de los procesos de desarrollo de Software aplicando a su vez las herramientas que el mercado MDA ofrece para dicho fin.

1.4 Diseño Metodológico

1.4.1 Tipo de Investigación

Esta investigación se define respectivamente como:

- Exploratoria ya que se propone estudiar y documentar un tema del cual en Nicaragua hay poco conocimiento, como es el caso del enfoque MDA. Esto servirá en el desarrollo de nuevos métodos o guías relacionados con la arquitectura orientada a modelos y,
- Descriptiva debido a que se analizará el enfoque MDA, sus estándares y las metodologías de desarrollo de software.

1.4.2 Hipótesis

El diseño de una metodología permite la selección de estándares basados en el enfoque MDA para las etapas de análisis y diseño del ciclo de vida del desarrollo de software, lo que facilitará el trabajo del desarrollador y contribuye en la calidad del producto final.

2. Marco Teórico

El enfoque MDA parte de la conocida y largamente establecida idea de separar la especificación de la operación de un sistema, de los detalles de la forma que el sistema utiliza las capacidades de su plataforma. [3]

MDA proporciona un método que permite a las herramientas que lo soportan:

- La especificación de un sistema independiente de la plataforma que le da soporte,
- La especificación de las plataformas,
- La selección de una plataforma en particular para el sistema y,
- La transformación de la especificación del sistema en una sola para una plataforma determinada.

Por tanto, los tres principales objetivos de MDA son la portabilidad, la interoperabilidad y la reutilización mediante una separación arquitectónica.

2.1 Estándares del Enfoque MDA

Los diferentes estándares desarrollados por OMG están soportados por una infraestructura en común, ésta infraestructura, está definida como núcleo del lenguaje UML.

A partir de este núcleo se definen constructores básicos y conceptos comunes que se reutilizan para puntualizar varios metalenguajes (estándares) como MOF o CWM, también permite una personalización de UML mediante perfiles. Este paquete Core es un metamodelo completo diseñado para una alta reusabilidad, por lo tanto, es el corazón que sustenta toda la arquitectura de la aproximación MDA. [12]

El conocimiento y el estudio de los diferentes estándares MDA es la base para el diseño de la metodología realizada. De igual forma la experimentación con las herramientas de trabajo MDA permite una conceptualización más amplia de los diferentes estándares. Por tanto es importante conocer la utilidad que cada estándar aporta a la arquitectura MDA.

A continuación se hace una referencia a los estándares más importantes.

UML

El lenguaje unificado de modelado (UML) es un lenguaje gráfico para la representación de sistemas y nos permite modelar la arquitectura, los objetos, las interacciones entre objetos, datos y aspectos del ciclo de vida de una aplicación, así como otros aspectos más relacionados con el diseño de componentes incluyendo su

construcción y despliegue. Los elementos del modelado de UML, es decir, clases, interfaces, casos de uso, diagramas de actividad, etc. además pueden ser intercambiados entre herramientas del ciclo de vida utilizando XMI.

Actualmente nos encontramos con algunos perfiles UML para diferentes tecnologías como pueden ser CORBA, EJB, EDOC, etc. y otros muchos que están en desarrollo. Estos perfiles son muy importantes en la comunidad UML, dentro de los modelos de diseño y también proporcionan un enlace con la comunidad de desarrolladores y de integradores. [13]

Los grandes objetivos que se persiguen con UML son los siguientes:

- Proveer a los usuarios de un lenguaje de modelado visual, fácil de usar para el desarrollo de modelos.
- Proporcionar mecanismos de especialización y extensión de conceptos elementales del Core.
- Soportar especificaciones independientes de lenguajes de programación y procesos de desarrollo específicos.
- Alentar a la industria para aportar nuevas herramientas al mercado.
- Soportar conceptos de desarrollo de alto nivel como componentes, colaboraciones, entornos de trabajo y patrones.
- Integrar las buenas prácticas en los procesos de desarrollo.

UML es actualmente la mejor alternativa para modelar PIMs y PSMs (usando para este último caso, alguno de los variados PROFILES que especializan un modelo UML para representar cierta tecnología). Surge entonces una fuerte necesidad de aprendizaje y dominio profundo de este lenguaje de modelado, a fin de insertarse y ser útil en la visión MDA. [14]

MOF

El OMG plantea una arquitectura de cuatro capas para la definición de sus estándares, en donde cada capa se define como instancia de la anterior. Esta arquitectura de modelos denominada MOF (Meta ObjectFacility), extiende UML para que este sea aplicado en el modelado de diferentes sistemas de información. El estándar MOF define diversos meta modelos, esencialmente abstrayendo la forma y la estructura que describen los metamodelos. MOF es un ejemplo de un meta-meta modelo o un modelo del metamodelo orientado a objetos por naturaleza. Define los elementos esenciales, sintaxis y estructuras de metamodelos que son utilizados para construir modelos orientados a objetos de sistemas. Además, proporciona un modelo común para los metamodelos de QVT, OCL y UML. [15]

El framework MOF tiene bastantes ventajas sobre los sistemas de modelado simples y permite: [16]

- Soportar cualquier tipo de modelo y paradigma de modelado imaginable.
- Permite relacionar diferentes tipos de metadatos.
- Permite añadir de forma incremental metamodelos y nuevos tipos de metadatos.
- Permite intercambiar modelos y metamodelos entre elementos que usen el mismo meta-metamodelo.

El hecho de que tanto MOF como UML tengan en común la Infrastructure Library, incluye los siguientes beneficios: [16]

- Simplifica las reglas para el modelado de metadatos
- Las tecnologías de mapeos de MOF (XMI, JMI, etc.), se pueden aplicar a los modelos UML, incluidos los perfiles UML.
- Permite un amplio abanico de herramientas para el metamodelado, ya que cualquier herramienta UML se podrá utilizar para modelar metadatos fácilmente.

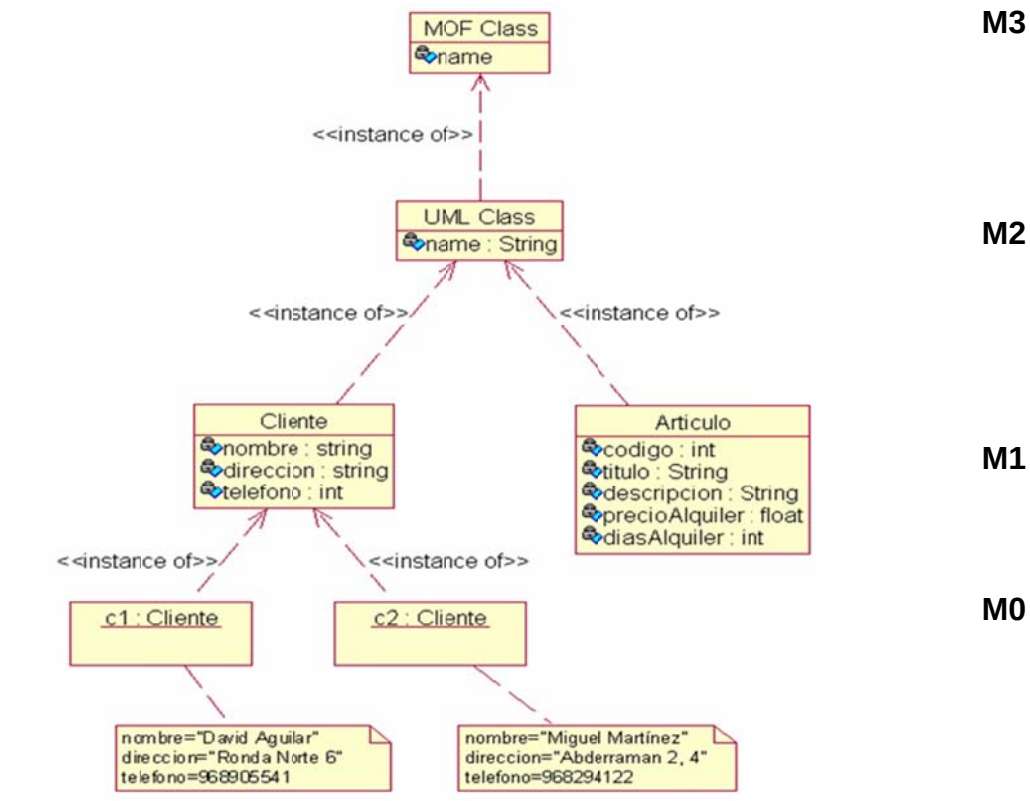


Figura 2: Ejemplo del Uso de los Estándares en MDA.

Además de estos beneficios, MOF incluye una serie de paquetes que facilitan la consecución de las capacidades de reutilización de MOF desde otros modelos o metamodelos. Estos paquetes son los descritos a continuación:

- **Reflection:** que extiende un modelo con la habilidad de ser auto descriptivo.
- **Identifiers:** que provee una extensión para objetos del metamodelo identificados excepcionalmente, sin contar con el dato del modelo que puede ser sujeto de cambio.
- **Extension:** un simple significado para extensiones de elementos del modelo con el par nombre – valor

A continuación se definen las diferentes capas especificadas en la arquitectura del OMG que sustenta MOF. [3]

- **Capa M3 (Meta metamodelo).** Corresponde a MOF, es una especificación que define un lenguaje abstracto para especificar, construir y manejar elementos comunes a cualquier metamodelo.
- **Capa M2 (Meta modelos).** Especifica las entidades de un lenguaje de modelado. Los lenguajes que se han definido como instancias de MOF son: UML, QVT, OCL y MOF en sí mismo. Adicionalmente se definen metamodelos para otros propósitos como objetos de negocio, workflows y modelos de componentes.
- **Capa M1 (Modelos).** Se refiere a los modelos de usuarios que suelen desarrollarse en el momento de construir un sistema de información.
- **Capa M0 (Instancias).** Describe instancias de las entidades propuestas en un modelo de un sistema de información. Es en este nivel en donde pueden usarse los diagramas de objetos como instancias de las clases para verificar que se cumplen las restricciones definidas en el nivel de los modelos (M1).

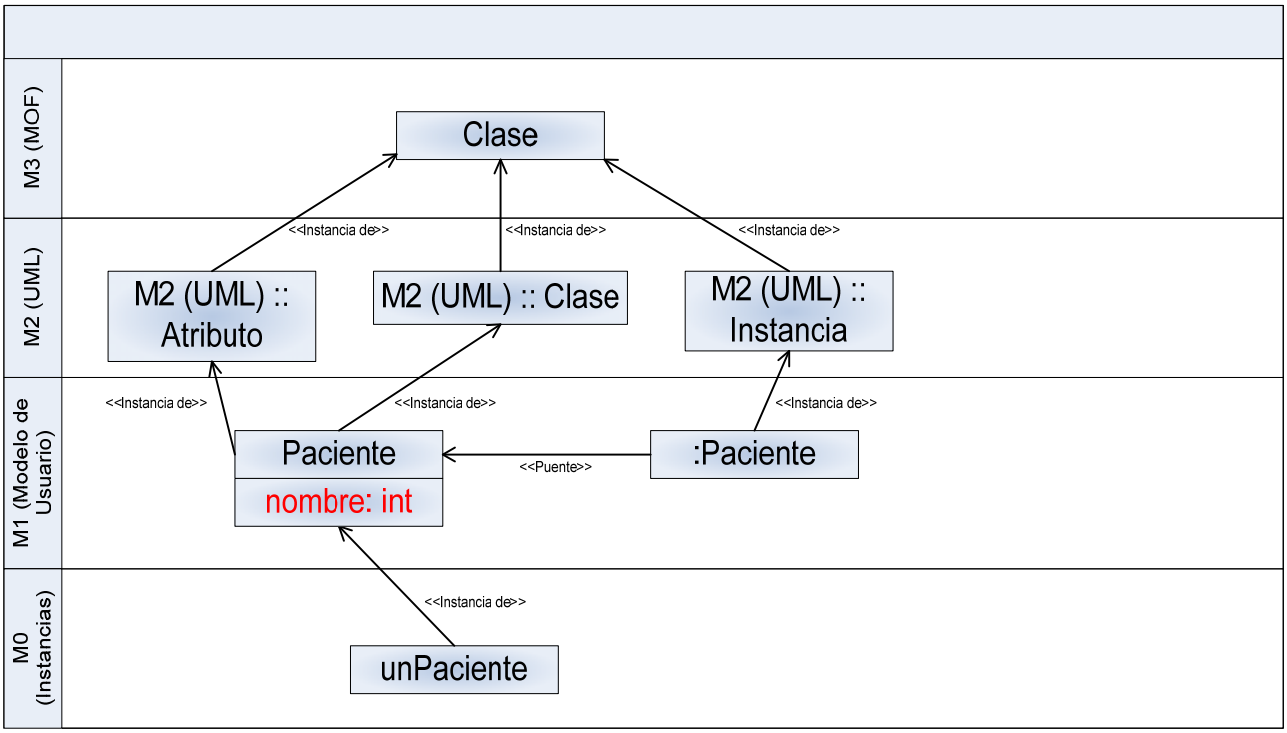


Figura 3: Descripción de los Metamodelos de MDA.

QVT

OMG ha propuesto un lenguaje estándar para la especificación de transformaciones llamada Query View Transformations (QVT) [3], que depende a su vez de otros dos estándares: MOF 2.0 y OCL 2.0. De esta manera, se pretende promover el uso estándar de las transformaciones entre modelos, lo que llevará asociado notables mejoras en la productividad, la interoperabilidad y la calidad, permitiendo que el desarrollo de software se lleve a cabo en un nivel de abstracción más elevado y utilizando conceptos más cercanos al dominio del problema.

QVT y Moment

La especificación QVT se define a través de dos dimensiones ortogonales: la dimensión del lenguaje y la dimensión de la interoperabilidad, donde cada una de las cuales tiene una serie de niveles.

La dimensión del lenguaje define los diferentes lenguajes de transformación presentes en la especificación QVT. Concretamente son tres: [17]

- Relations,
- Core y
- Operational

La principal diferencia entre ellos es su naturaleza declarativa o imperativa.

En la dimensión de la interoperabilidad se encuentran aquellas características que permiten a una herramienta, que cumple el estándar QVT, relacionarse con otras herramientas. La intersección de niveles de las dos dimensiones especifica un punto de cumplimiento QVT (QVT-compliance).

Gran parte de los lenguajes propuestos para transformación de modelos se basan en el lenguaje QVT. La definición de QVT consta de 3 paquetes: 2 declarativos (Relations y Core) y uno imperativo. En su mayoría, estas propuestas de lenguajes son híbridas: declarativas e imperativas. Para definir una relación de transformación entre modelos, bastaría especificarla declarativamente y posteriormente elegir en que lenguaje imperativo se podrá ejecutar y recién entonces escribir sentencias ejecutables que se correspondan a las declaraciones. [18]

Gráficamente, una instanciación de esta transformación podría ser, por ejemplo, la que se muestra en la Figura 4 entre dos modelos concretos Uml1 (de UML) y Rel1 (de Rdbms), donde n = 'Persona' y un nombre de atributo podría ser 'nombre'. O sea, para la clase Persona del modelo Uml1, existe una Tabla en el modelo relacional Rel1 con el mismo nombre. Lo mismo sucede con los atributos: para cada atributo de la clase Persona existe una columna en la tabla Persona con el mismo nombre. O sea, para el atributo `nombre´, existe la columna `nombre´.

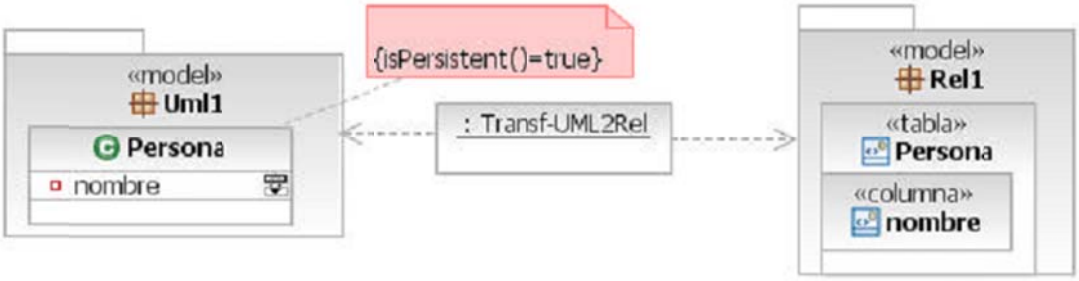


Figura 4: Una Instancia Gráfica de la Transformación.

OCL

OCL (Object Constraint Language) es un lenguaje formal usado para describir expresiones sobre modelos UML. Estas expresiones suelen utilizarse con el fin de describir restricciones sobre dichos modelos, y por tanto realizan consultas sobre los objetos descritos en él. Un elemento importante a tener en cuenta es, que cuando se evalúa una expresión OCL, su resultado no puede influir sobre el modelo (por ejemplo, no puede alterar el estado de la correspondiente ejecución del sistema). [19]

Las expresiones OCL se pueden usar para especificar operaciones y expresiones, que cuando sean ejecutadas alteren el estado del sistema. Los desarrolladores pueden utilizar OCL para especificar restricciones específicas de una aplicación en

sus modelos. También pueden usar OCL para especificar consultas sobre un modelo UML que esté completamente programado en un lenguaje de programación independiente.

OCL es un lenguaje tipado ya que cada expresión OCL tiene asociado un tipo. Para ser un lenguaje bien formado cada expresión debe cumplir con las reglas del lenguaje, de manera que por ejemplo, no permita comparar un entero con una cadena de caracteres. Cada Clasificador definido en un modelo UML representa a distintos tipos del lenguaje OCL. No obstante, OCL tiene su propio conjunto de tipos predefinidos. OCL se puede utilizar para diferentes propósitos: [20]

- Como un lenguaje de consulta.
- Para especificar invariantes en clases y tipos en los modelos de clases.
- Para especificar tipos invariantes para estereotipos.
- Para describir precondiciones y pos condiciones en métodos y operaciones.
- Para describir objetivos para mensajes y acciones.
- Para especificar restricciones a las operaciones.
- Para especificar reglas de derivación para atributos de cualquier expresión de modelos UML.

Las principales tipos de restricciones que se pueden construir utilizando OCL son: a) Invariante, condición para una clase, tipo o interfaz que siempre debe ser satisfecha por todas sus instancias. b) Precondición, condición que debe ser cierta antes de ejecutar una operación de una clase. c) Pos condición: condición que debe ser cierta después de ejecutar una operación de una clase. Es importante resaltar que usando OCL se pueden escribir restricciones en el nivel de los meta-modelos; de esta forma, por ejemplo, se definen las reglas de consistencia de un lenguaje como UML. [19]

XMI

XML Metada Interchange (XMI) es un estándar de la OMG que mapea MOF a XML. XMI define como deben ser usadas las etiquetas XML que son usadas para representar modelos MOF serializados en XML.

Los meta modelos MOF son convertidos en DTD (Document Type Definitions) y los modelos son convertidos en documentos XML que son consistentes con su DTD correspondiente. XMI resuelve muchos de los problemas encontrados cuando intentamos utilizar un lenguaje basado en etiquetas para representar objetos y sus asociaciones, y además el hecho de que XMI esté basado en XML significa que tanto los metadatos (etiquetas) y las instancias que describen (elementos) pueden ser

agrupados en el mismo documento, permitiendo a las aplicaciones entender rápidamente las instancias por medio de los metadatos. [15]

Otra propuesta de estándares MDA es el proyecto Eclipse Modeling Project que implementa tecnologías alternativas a muchas de las especificaciones MDA. Los estándares Eclipse son equivalentes a los de OMG en una versión simplificada y en plataforma Open Source, aunque sin una colaboración real entre las dos empresas.

2.2. El ciclo de vida de desarrollo de sistemas MDA.

2.2.1. Contexto

El ciclo de desarrollo MDA no es muy distinto al desarrollo de software convencional, por tanto se hará un repaso de cada una de las etapas del desarrollo de software MDA resaltando las etapas en donde MDA juega un papel ponderarte.

Ahora bien, desde un punto de vista general puede considerarse que el ciclo de vida de un software tiene tres etapas claramente diferenciadas, los cuales se detallan a continuación.

2.2.2. Identificación de las necesidades y oportunidades.

En esta primera etapa el analista de sistemas elabora un documento donde se pone en manifiesto, las necesidades, los requerimientos y las funcionalidades que ofrecerá a los usuarios el sistema a implementar. En esta etapa de se trata de puntualizar los recursos requeridos para definir una solución al problema.

En esta parte, los usuarios finales, desarrolladores y administradores son los que están involucrados en esta primera fase del desarrollo de software MDA, entre las actividades que se pueden identificar en esta fase están: entrevistar a los encargados de coordinar a los usuarios, sintetizar el conocimiento obtenido, estimar el alcance del proyecto y documentar los resultados.

Como se podrá notar esta fase del ciclo de vida MDA es similar a su contraparte, el ciclo tradicional de desarrollo de software.

2.2.3. Análisis, Diseño y Código

Ahora que sabemos con qué hacer y con qué hacerlo, tenemos que determinar el ¿cómo lo vamos hacer?, por eso MDA representa un nuevo paradigma de desarrollo de software en la que los modelos guían todo el procesos de análisis y diseño. A este nuevo paradigma se ha denominado Ingeniería de Modelos o Desarrollo Basado en Modelos.

La idea clave que nace de MDA es que el desarrollo esta guiado por Modelos del software, tendrán beneficios importantes en aspectos fundamentales como son la productividad, la portabilidad, la interoperabilidad y el mantenimiento.

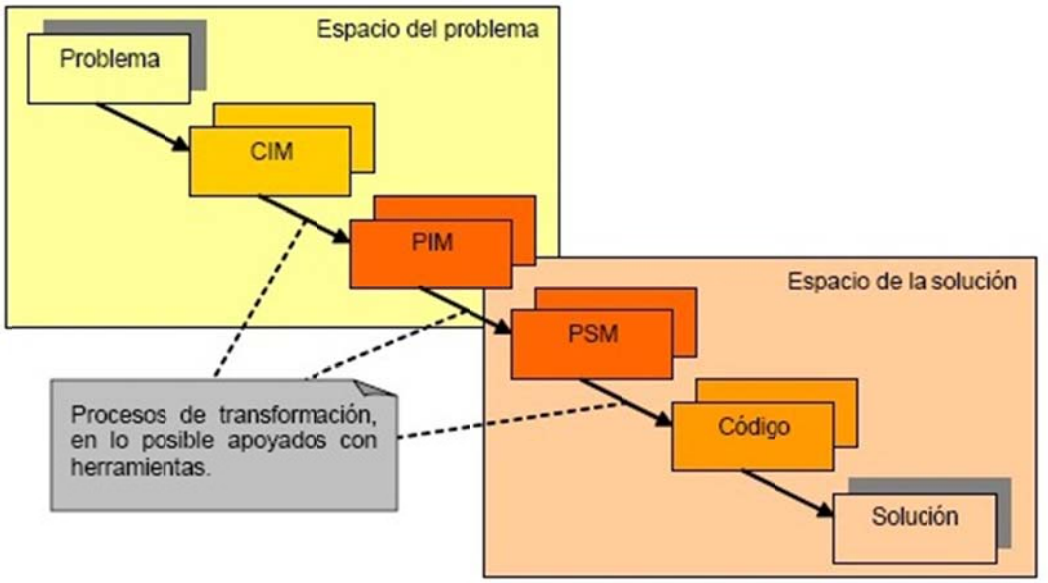


Figura 5: Ciclo de Vida del Desarrollo de Software MDA.

La principal diferencia entre el enfoque tradicional y el enfoque propuesto por MDA radica en la formalización y consistencia en que se realiza el proceso de transformación del modelo de una fase a otra, esta transformación de modelos se hace en la fase de análisis y diseño del sistema, eje fundamental donde gira el proceso metodológico de la selección de estándares. Es decir, en el enfoque tradicional, la consistencia del proceso de transformación depende de la habilidad del desarrollador, mientras que en MDA, el proceso de transformación se realiza de forma asistida utilizando mecanismos que garantizan la consistencia con los modelos anteriores.

Para conseguir estos beneficios, MDA plantea el siguiente proceso de desarrollo:

1. **PIM** (Modelos de Negocios): de los requisitos se obtiene un Modelo Independiente de Plataforma, estos PIM proporcionan especificaciones formales de la estructura y del funcionamiento del sistema y abstrae los detalles técnicos. Una PIM-View, describe los componentes de cómputos y sus iteraciones de manera independiente de la plataforma de trabajo. Por tanto, los PIM representan los modelos que describen una solución de software que no contiene detalles de la plataforma concreta en que la solución va a ser implementada, de ahí su nombre de modelos independientes de la plataforma. Estos modelos surgen como resultado del análisis y diseño.

2. **PSM** (Transformación de Modelos): Luego del proceso de elaboración de los PIMs el modelo es transformado en uno o más modelos específicos de la plataforma. PSM está indicado para especificar el sistema con términos que

estén disponibles en una tecnología específica de implementación. En el modelo tradicional las transformaciones de modelo a modelo o de modelo a código se hacen a mano. En MDA todas las transformaciones se hacen con herramientas: el paso de PSM a código y la transformación de PIM a PSM. En otras palabras los PSM son los modelos derivados de la categoría anterior, que contienen los detalles de la plataforma o tecnología con que se implementará la solución.

3. **Código**: El último paso en MDA es la transformación de PSM a código, para ello existen ya muchas herramientas lo que hace que la transformación sea prácticamente directa. Los programadores únicamente tendrán que programar la funcionalidad que no puede reflejarse en los modelos y, si es necesario tendrán que refinar el código generado.

La idea básica de MDA, más allá de los modelos PIMs y PSMs, es utilizar los dos conceptos clave: Modelo y Transformación. El primer paso es construir un modelo conceptual (PIM) expresado en UML, el cual será transformado hacia varios PSM. Estas transformaciones de un PIM en varios PSM pueden realizarse las veces que se requiera. Si se hacen cambios al modelo conceptual se pueden regenerar los PSM automáticamente.

Con este planteamiento, el desarrollo se focaliza en la definición de los modelos PIM, ya que tanto los PSM, como el código se van a generar mediante transformaciones. Es cierto que la transformación precisa del esfuerzo que requiere una tarea tan especializada, pero la ventaja es que haciéndola una vez, se puede aplicar en muchos sistemas. Los desarrolladores definen mejor sus modelos porque se aíslan de los detalles técnicos, lo que permite centrarse exclusivamente en los detalles del negocio concreto, obteniendo resultados en menos tiempo. Por otro lado las trasformaciones, generarán tanto el PSM, como el código de una forma rápida, sin perder ningún detalle técnico, ya que todos ellos quedan definidos en las transformaciones.

Al contrario que el planteamiento tradicional, MDA obtiene teóricamente una alta productividad, porque la documentación generada en las primeras fases, es algo más que documentación. Cualquier persona será capaz de leer y comprender los modelos que a su vez sirven como punto de partida para la generación automática del código.

Por lo tanto, MDA incorpora la idea de transformaciones entre los modelos, por lo que las herramientas de transformación apoyan los procesos de automatización de estas tareas, de hecho estas herramientas son uno de los elementos básicos de MDA.

2.2.4. Puesta en producción.

En esta parte es donde se presenta al cliente o al usuario final el producto terminado, sabiendo que funciona correctamente y responde a los requerimientos solicitados en su momento.

Durante esta fase se trabaja con los usuarios para desarrollar documentación efectiva para el software, como manuales de procedimientos, ayuda en línea y sitios Web que incluyan respuestas a preguntas frecuentes (FAQ, Frequently Asked Questions) en archivos "Léame" que se integrarán en el nuevo software. La documentación indica a los usuarios cómo utilizar el software y lo que deben hacer en caso de que surjan problemas derivados de este uso.

De igual forma se realizarán las pruebas correspondientes al producto terminado. El objeto de las pruebas, expresado en forma sencilla, es encontrar el mayor número posible de errores con una cantidad razonable de esfuerzo, aplicado sobre un plazo de tiempo realista.

Los modelos PSM permiten al diseñador crear la arquitectura de diseño integrando componentes reusables. La arquitectura de Modelos MDA extiende el modelo de diseño aun dominio de ejecución. Un lenguaje de programación Orientado a Objetos se puede usar para traducir los modelos PSM de tal manera que puedan ejecutarse en una máquina.

El objetivo de esta etapa es similar a la etapa de desarrollo de software convencional, con la ventaja de que los modelos PSM permiten generar el código de manera automática.

2.3. Visión alternativa de MDA

Una visión práctica y alternativa de MDA se encuentra en un manifiesto [21] elaborado por los especialistas de IBM trabajando en MDA, en donde se resume MDA como la combinación de tres ideas Complementarias. Estas tres ideas se plantean a continuación.

- **Representación directa:** Se desplaza el foco del desarrollo del software del dominio de las tecnologías hacia las ideas y conceptos del dominio del problema. Reduciendo la distancia semántica entre el dominio del problema y su representación, permitiendo un mayor acoplamiento de las soluciones de los problemas, logrando diseños más acertados e incrementando la productividad.
- **Automatización:** Usar herramientas para mecanizar aquellas facetas del desarrollo del software que no dependen del ingenio humano. MDA incrementa la velocidad del desarrollo y reduce errores usando herramientas automatizadas para transformar modelos específicos de plataforma en código

de implementación. Es lo mismo que hacen los compiladores para los lenguajes de programación tradicionales.

- **Estándares abiertos:** Los estándares han sido uno de los mejores impulsores del progreso en la historia de la tecnología. Los estándares de la industria no solo ayudan a eliminar la diversidad gratuita, sino que también alientan a los vendedores a producir herramientas tanto para propósitos generales como para propósitos especializados, incrementando así las posibilidades del usuario. El desarrollo de código abierto asegura que los estándares se implementan consistentemente y anima la adopción de estándares por parte de los vendedores.

2.4 Análisis de las herramientas MDA

Como se ha descrito anteriormente, los objetivos que persigue MDA son de mejorar la productividad, la portabilidad, la interoperabilidad y la reutilización de los sistemas de información.

A grandes rasgos, el proceso de desarrollo de software basado en el enfoque MDA se puede dividir en tres pasos:

- Construcción de los modelos independientes de la Plataforma (PIM).
- Transformación del Modelo anterior a uno o varios Modelos Específicos de Plataforma (PSM).
- Generación del Código a partir de los PSM.

El paso de PIM a PSM y de PSM a código no se realiza “a mano”, sino que se usan herramientas de transformación para automatizar estas tareas.

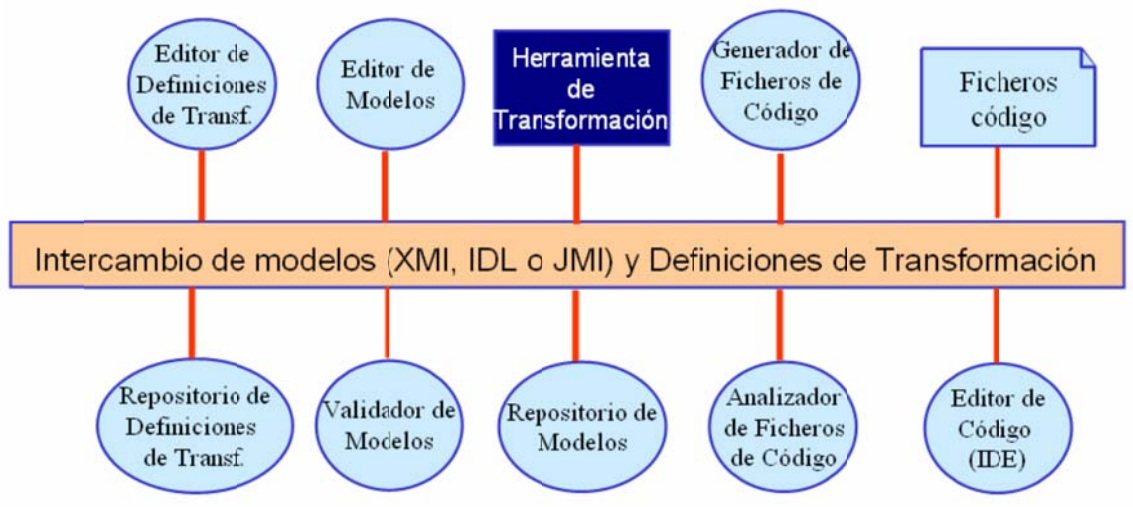


Figura 6: Uso de las Herramientas MDA.

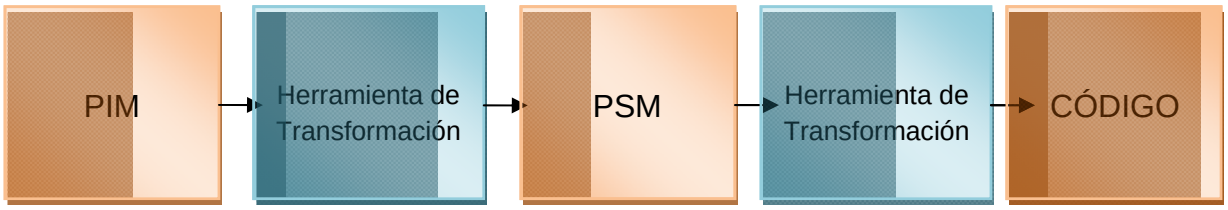


Figura 7: Transformaciones de las Herramientas MDA.

2.4.1 Criterios para evaluar las herramientas MDA.

En esta parte se presentan los criterios que se tomaron en cuenta al probar las herramientas MDA para ello se han identificado unos criterios completos y bien definidos que debe de cumplir una herramienta con soporte completo MDA.

A continuación se muestra una tabla con las propiedades evaluadas en el estudio.

Id	Propiedades funcionales.	Descripción
1	Soporte de PIMs	La herramienta permite que se especifique un sistema mediante un modelo independiente de cualquier plataforma (PIM).
2	Soporte para PSMs	La herramienta permite construir modelos del sistema que capturan los aspectos esenciales de una tecnología de implementación determinada (PSMs)
3	Transformaciones de los Modelos	Permite verificar el grado de automatización de los procesos de transformación de los modelos.
4	Integración de Modelos	Permite integrar diferentes modelos para producir una única aplicación, principalmente mediante la generación de los “puentes” apropiados para comunicar las distintas partes entre sí.
5	Interacción con el usuario.	Permite verificar el grado de participación de los usuarios en el proceso de transformación de los modelos.
6	Tipo de transformaciones	Evaluar, si permite realizar transformaciones verticales y/u horizontales. Se entiende por transformaciones verticales, las que se realizan entre los diferentes niveles de abstracción (PIM y PSM) y, por horizontales, las que se realizan entre los diferentes modelos de un mismo nivel de abstracción.
Id	**Propiedades Técnicas**	**Descripción**
7	Lenguaje de almacenamiento y gestión de Modelos	Para realizar las transformaciones de los modelos entre los distintos niveles de abstracción es necesario un lenguaje que permita el almacenamiento y la gestión de estos modelos de forma que éstos se puedan intercambiar entre los distintos niveles PIMs y PSMs.

8	Uso de patrones	La herramienta aplica o permite aplicar patrones de diseño en la Construcción de PIMs, PSMs y Código y pueden definirse otros nuevos o modificar los existentes.
9	Plataformas y tecnologías soportadas	Verificar que plataformas de desarrollo soporta la herramienta.
10	Ámbito de aplicación	De debe determinar el tipo de desarrollo en el que se centran, en particular se interesa determinar si las herramientas están orientadas hacia el desarrollo de SI Web o hacía desarrollos orientados a servicios, o ambos.
11	Herramientas de soporte	Además de las herramientas de transformación, la aplicación incluye otras herramientas para dar soporte completo a MDA: editor de Código, editor de Modelos, herramientas para la prueba y despliegue.
Id	**Propiedades de Calidad**	**Descripción**
12	Ciclo de Vida	Incluye la mayor parte de un ciclo de vida de un desarrollo con MDA, esto es, el análisis, el diseño, la implementación, el ensamblado, el despliegue y el mantenimiento.
13	Estandarización	La herramienta utiliza los estándares propuestos para la metodología planteada en el estudio.
14	Calidad del Código Generado	La aplicación genera código de Calidad, bien documentado, legible y que puede ser adaptado o extendido fácilmente por el desarrollador.

Para evaluar con detalle cada herramienta, puntuaremos del 0 al 4 cada propiedad de la tabla anterior, según el siguiente criterio.

Puntuación	Descripción
0	No soporta la propiedad.
1	Poco soporte de la propiedad.
2	Algún soporte de la propiedad.
3	Fuerte soporte de la propiedad.
4	Excelente soporte de la propiedad.

2.5 AndroMDA

AndroMDA [22][23] es una herramienta de generación de código que toma modelos UML en formato XMI como entrada y genera código como salida, en cualquier lenguaje de programación. Nace en el año 2002 como una iniciativa de Matthias Bohlen [24].En el año 2003 adquiere el nombre de AndroMDA debido a que toma las bases del paradigma MDA.

AndroMDA está compuesta por *cartridges*. Los *cartridges* son un tipo especial de plug-ins, donde se definen los meta-modelos y reglas para la transformación de elementos del modelo de acuerdo al meta-modelo. En muchos casos, un *cartridge* puede contener solamente los meta-modelos, ya que las reglas de transformación pueden ser manejadas por muchos *cartridges* y pueden ser contenidas en un *cartridge* común.

Se ha seleccionado AndroMDA, porque permite la generación de código en diferentes lenguajes de programación.

2.5.1 Aplicación de AndroMDA

El funcionamiento AndroMDA depende del lenguaje con el que se vaya implementar el Sistema de Información.

Se ha seleccionado como lenguaje de implementación C# (utilizando el IDE Visual Studio 2005), por ello se ha instalado el *cartridge Android/VS.* Además ha sido necesaria la instalación de herramientas adicionales como MagicDraw 9.16 para realizar los diferentes diagramas y *Apache'sMaven* que permite el acceso a las librerías necesarias de AndroMDA de forma transparente para el usuario.

Para definir el meta-modelo se ha utilizado MagicDraw, y posteriormente se ha exportado a .Net. AndroMDA toma como meta-modelo el meta-modelo de UML. Como el meta-modelo utiliza extensiones UML usando estereotipos, que si bien pueden ser representados con la herramienta MagicDraw, cuando se exporta a .Net se eliminan los estereotipos. Esto ocurre porque en el *cartridge* de AndroMDA para .Net no están definidos los estereotipos ni la forma de realizar la transformación de los mismos a .Net. Como se ha dicho anteriormente los meta-modelos y las reglas de transformación se encuentran definidas en los *cartridges*, por lo cual para poder desarrollar modelos de clases acordes al meta-modelo se debería definir el *cartridge* correspondiente con el meta-modelo y el lenguaje de desarrollo que se desee utilizar.

2.5.2 Evaluación de AndroMDA

Id	Propiedad	Puntuación	Comentarios
1	Soporte de los PIMs	4	Permite crear Modelos Independientes de la Plataforma (PIM) a través del Modelo de Clases.
2	Soporte para PSMs	3	No tiene soporte para construir PSM's explícitos, se usa un PIM, los *cartridge* y los *perfles* UML para generar directamente el código.
3	Transformaciones de los Modelos	4	Permite la transformación de modelo a modelo ayudando así a levantar el nivel de abstracción, y adicionalmente permite escribir las propias transformaciones del usuario.
4	Integración de Modelos.	4	Los Modelos se integran perfectamente entre sí de forma transparente y automática.
5	Interacción con el usuario.	4	La interacción con el usuario es alta ya que debe de participar activamente en todas las etapas del desarrollo.
6	Tipos de Transformaciones	2	La aplicación permite tanto la creación de nuevos *cartridge* (permitiendo usar hasta la propia herramienta para generarlos ya que son archivos .JAR) así como también la extensión de los existentes. Esto nos permite tener transformaciones Horizontales modificando el *cartridge.*
7	Lenguajes de almacenamiento y gestión de modelos.	3	Los modelos generados son almacenados en XMI, permitiendo realizar el intercambio de modelos entre los diferentes niveles
8	Uso de Patrones	4	Se hace uso de múltiples patrones, tanto en la transformación de modelos como en la generación del código. Se pueden definir nuevos patrones.
9	Plataformas y Tecnologías soportadas.	3	Utiliza herramientas adicionales tales como UML (MagicDraw, Poseidon, ArgoUML entre otras),

			Apache'sMaven, que simplifica su uso ya que permite el uso de los *cartridges* de AndroMDA para los diferentes lenguajes. Además se pueden utilizar herramientas CASE que permitan generar los diagramas en formato XMI. Utiliza Nhibernate para asegurar la persistencia de los objetos. También posee *cartridges* que permiten generar código en plataformas como Java, .Net o PHP.
10	Ámbito de Aplicación	3	Está pensada para el desarrollo de software orientado a servicios y SI Web.
11	Herramientas de Soporte.	4	A parte del editor de modelos AndroMDA incorpora un completo entorno IDE ya sea en plataformas Java, .Net o PHP.
12	Ciclo de Vida	3	Soporta casi todas las fases del ciclo de vida, desde análisis, diseño, codificación, mantenimiento, pruebas. No posee soporte para la fase del manejo de requisitos.
13	Estandarización	4	Cumple con todos los estándares propuestos en la metodología
14	Calidad del Código Generado	4	Bien documentado y legible.

2.6 OpenMDX

OpenMDX es una herramienta para el desarrollo de sistemas de información. Fue concebido como una herramienta CASE para realizar el análisis y el diseño de sistemas orientados a MDA.

OpenMDX es una implementación de los estándares de la arquitectura dirigida a modelos (MDA) de OMG. En contraste con la mayoría de las implementaciones de MDA OpenMDX es un framework y no implementa el enfoque de mapear desde el PIM hacia el PSM. OpenMDX ofrece como características principales un lenguaje orientado a objetos puro, modelos independientes de la plataforma (PIM) y plataforma juntas en forma ortogonal.

OpenMDX es líder en lo que respecta a *FrameWorks* que aplican MDA. Este *FrameWork* nos permite crear aplicaciones Orientada a servicios y aplicaciones independientes de la plataforma.

La interfaz de aplicaciones está definida a través de los Modelos Independientes de la Plataforma (PIM) y la lógica de negocios se implementa como POJOs (plainold Java objects). Las aplicaciones resultantes son ligeras (openmdx-base.jar solo 2.9M) y se pueden desplegar en cualquier plataforma J2SE o J2EE. [25]

OpenMDX[26] está desarrollado bajo los estándares establecidos de la OMG tales como:

- Unified Modeling Language (UML)
- Meta Object Facility (MOF)
- XML Metadata Interchange (XMI)
- Common Warehouse Metamodel (CWM)

Los estándares MDA separan la capa de negocios de las aplicaciones lógicas de la plataforma de aplicaciones. Los PIMs de una aplicación pueden ser construidos usando UML en conjunto con los otros estándares propuestos por la OMG. Estos modelos pueden ser construidos a través de MDA prácticamente en cualquier plataforma, abierta o propietaria, incluyendo, Web Service, .NET, CORBA, J2EE, y otras. Estos permiten crear modelos independientes de la plataforma de codificación disponible.

El Modelo de Domino basado en una aplicación OpenMDX se especifica mediante un PIM. Estos modelos no contienen ninguna información específica de la plataforma.

Se ha seleccionado OpenMDX para este estudio, ya que es una herramienta que ha nacido como propuesta de la comunidad de desarrolladores Open Source. Está

desarrollado utilizando el lenguaje JAVA, permitiendo así que funcione en cualquier plataforma que tenga instalada JAVA2.

2.6.1 Aplicación de OpenMDX

Para este caso se ha utilizado la versión openMDX SDK 2.8.0, como lenguaje de implementación se ha utilizado Eclipse 3.5, para ello se ha instalado las herramientas necesarias adicionales tales como Tomcat + OpenEJB y Apache Ant 1.8 que nos permite el acceso a las librerías de OpenMDX[27].Este proceso de instalación nos permite usar las herramientas UML 2.0 en Eclipse. Esto hace posible ver los diagramas UML que nos suministra el SDK de OpenMDX.

Ya que OpenMDX es una herramienta OpenSource se puede agregar la funcionalidad correspondiente para poder definir los estereotipos con la herramienta e implementar las transformaciones correspondientes.

2.6.2. Evaluación de OpenMDX

Id	Propiedad	Puntuación	Comentarios
1	Soporte de los PIMs	4	Permite definir PIMs completos mediante los diagramas de Clases UML.
2	Soporte para PSMs	0	OpenMDX no utiliza un PSM explícito, en lugar de un PSM utiliza un PIM estereotipado junto con las marcas para generar el código directamente
3	Transformaciones de los Modelos	4	Los Modelos se integran bastante bien de forma transparente y automatizada.
4	Integración de Modelos.	2	La herramienta permite que los elementos de un modelo interactúen con los elementos de otro modelo. Sin embargo se necesita codificar algunas partes de esos modelos para que la integración fuera completa.
5	Interacción con el usuario.	4	La interacción con el usuario es alta ya que debe de participar activamente en todas las etapas del desarrollo.
6	Tipos de Transformaciones	2	OpenMDX genera mapeos de PIM hacia PIM los cuales reduce el número de generaciones de código drásticamente comparados con

			herramientas que manejan un enfoque PIM hacia PSM.
7	Lenguajes de almacenamiento y gestión de modelos.	0	Los modelos generados son almacenados en JMI (Estándar basado en MOF para el intercambio de datos en plataformas J2SE).
8	Uso de Patrones	4	OpenMDx ofrece un amplio rango de patrones y *plugins* esto hace que pueda implementar los patrones que desee.
9	Plataformas y Tecnologías soportadas.	4	OpenMDX trabaja en múltiples plataforma de desarrollo tales como: Web Service, .NET, CORBA, Java.
10	Ámbito de Aplicación	4	OpenMDX está pensada para el desarrollo de sistemas de información. Posteriormente, con la evolución de los SI Web se adaptó al desarrollo de software orientado a servicios.
11	Herramientas de Soporte.	4	Para su funcionamiento requiere el uso de Apache Ant que proporciona los *cartridges* de OpenMDX. También posee *cartridges* que permiten generar código en plataformas como Java o .Net
12	Ciclo de Vida	4	La herramienta engloba casi todas las fases del ciclo de vida: análisis, diseño, codificación, mantenimiento, pruebas.
13	Estandarización	3	Cumple con casi todos los estándares, la excepción es el uso de JMI en vez del estándar XMI
14	Calidad del Código Generado	4	Bien documentado y legible.

2.7. Resultados de la Evaluación de las Herramientas MDA.

AndroMDA

Teniendo en cuenta la evaluación realizada sobre AndroMDA, se aprecia claramente que la herramienta cumple mayoritariamente los criterios propuestos para utilizarse en un desarrollo basado en el enfoque MDA.

Además de las características mencionadas en el estudio evaluativo, hay otro factor importante a destacar. Este factor es que AndroMDA tiene definidos los *cartridges* para la mayoría de los lenguajes de desarrollo, por lo que la implementación en distintas plataformas es muy fácil de realizar. Los *cartridges* se basan en el meta-modelo de UML. Esto y el uso de los estándares propuesto para la Metodología hacen de AndroMDA una buena opción en el desarrollo de Sistema de Información MDA.

OpenMDX

Se basa en el meta-modelo de UML. Por tanto, al igual que ocurre con AndroMDA, si los modelos que se desean generar son acordes a UML, éstos son fáciles de implementar. Pero si se desea realizar modelos basados en perfiles UML, se debe codificar la funcionalidad necesaria que permita recoger los estereotipos.

En OpenMDX el PIM pasa directamente a código sin que exista un PSM que pueda cambiar el desarrollador, sin embargo tiene un punto a favor, al igual que AndroMDA es una herramienta que permite generar código para diferentes plataformas mediante la arquitectura (*cartridges*). Aunque la herramienta utiliza el estándar JMI este proporciona un modelo común de programación que está en la forma de la interfaz generada automáticamente desde el metamodelo. Las interfaces proporcionan un formato común de intercambio basado en XML, se trata del XMI.

Lo arriba descrito hace de OpenMDX una buena opción de Desarrollo de Software basado en el enfoque MDA.

3. Diseño de una Metodología para la selección de estándares MDA.

3.1. Necesidad de una Metodología

En la década de 1970 los programas fueron creciendo en complejidad, el concepto de Ingeniería de Software se fue consolidando como una ciencia para el desarrollo de software, por lo que la antigua técnica de codificar y corregir terminó quedando obsoleta. Está técnica se basaba en requerimientos ambiguos y sin especificaciones puntuales. Al no seguir normas para el proyecto el cliente o usuario final sólo impartían especificaciones muy generales del producto final. Se programaba, se corregía y se volvía a programar sobre la misma marcha del proyecto. El ciclo de vida de este tipo de proyecto finalizaba cuando se satisfacían las especificaciones, no sólo las primeras por las cuales nació la necesidad del programa, sino de todas aquellas que fueron surgiendo sobre la marcha.

Los proyectos de software se habían venido desarrollando bajo el paradigma de prueba y error, esta actividad no estaba administrada, supervisada o gestionada de ningún modo, porque se iba corrigiendo a medida que surgían los errores, tanto los lógicos provenientes de la codificación, como los vacíos por deficiencia en el levantamiento de requerimientos solicitados por el usuario final o el cliente.

Este paradigma de prueba y error evolucionó a paradigmas más estructurados y posteriormente a la Ingeniería de Software. De la madurez de esta última surge el enfoque MDA como alternativa al desarrollo de software tradicional. Esta nueva alternativa presenta muchas ventajas tales como: un alto grado de abstracción y la posibilidad de aislarse de detalles técnicos, uso de lenguaje de modelado gráfico que simplifica la generación, entendimiento y modificación de los modelos a desarrollar; a la vez los modelos poseen la característica fundamental que pueden ser procesados por una computadora , por lo que las transformaciones se pueden realizar automáticamente, esto significa un aumento considerable de la productividad y la calidad del código generado. Otras ventajas que encontramos son la portabilidad, la interoperabilidad, y la reutilización de los modelos.

Actualmente el enfoque MDA tiene definida su arquitectura y muchos estándares necesarios para su implementación, así como también existen herramientas que soportan algunos de estos estándares, pero no existe un proceso lógico para hacer uso de este enfoque y obtener las ventajas que nos puede brindar. Por lo tanto surge la necesidad de construir una metodología para la selección de estos estándares y a la vez que contenga la aplicación de estos estándares en el ciclo de vida del software.

Para el diseño de la metodología de selección de estándares en sus etapas de análisis y diseño, se han estudiado los siguientes documentos:

- Métrica 3: se autodefine como una "metodología para la sistematización de las actividades que dan soporte al ciclo de vida del software". Es un trabajo de distribución y uso libre, realizado por el Ministerio de Administraciones Públicas Español (MAP).[28]
- La Guía MDA Versión 1.0.1 “MDA Version 1.0.1”.[3]
- Las Guías de Especificación de la OMG para los estándares MOF, OCL, QVT, XMI “OMG Specification”
- UML Infraestructure and Superstructure of OMG Versión 2.0
- El Modelo de ciclo de vida de desarrollo propuesto por la norma ISO/IEC 12207 “InformationTechnology Software LifeCycleProcesses”.

3.2. Definición de la Metodología.

En la definición de la metodología de selección de estándares se parte del hecho que el desarrollador conoce del enfoque MDA, sus conceptos y ventajas, y a su vez está familiarizado con los estándares y arquitectura que son parte de MDA.

La metodología está desarrollada para las etapas de análisis y diseño únicamente, tomando en cuenta un desarrollo orientado a objetos, dada la naturaleza de MDA. Por tanto la fase de análisis tomará como insumo el resultado y entregables de la fase previa de viabilidad del sistema. Así mismo al finalizar la etapa de diseño, se hará constar gráficamente que los entregables generados servirán de insumo para las siguientes fases consideradas en el ciclo de vida del software.

La metodología se descompone en las etapas de análisis y diseño, cada una de las etapas se divide en actividades, y éstas a su vez en tareas. Para cada una de las tareas se realiza una descripción, se define una meta y se proponen técnicas, herramientas (software) y estándares del enfoque MDA a aplicar, en esta última parte se hará una breve explicación del uso del estándar en determinada tarea.

El orden asignado de las actividades no debe interpretarse como una secuencia en su realización, ya que éstas pueden realizarse en orden diferente a su numeración o bien en paralelo, como se muestra en los gráficos de cada proceso.

A continuación se muestra el diagrama de la Estructura Lógica de la Metodología (Figura#8).

Figura 8: Estructura Lógica de la Metodología

3.3. Fundamentación para la Selección de Estándares usados en la Metodología.

Para llevar a cabo la Metodología, se seleccionaron los estándares más significativos como son UML, MOF, QVT, OCL, XMI. Esto de acuerdo a la base del estudio y análisis del paradigma MDD [12], que alienta a hacer de los modelos valores conductores primarios en todos los aspectos del desarrollo de software, y específicamente en las fases de análisis y diseño; por tales motivos se ha optado por un lenguaje gráfico de modelado como UML (Unified Modeling Language) y un lenguaje formal para especificar restricciones, fuertemente aceptado como OCL (Object Constraint Language); características y funcionalidades de cada estándar propuesto para su adecuada utilización en las fases de análisis y diseño en el proceso de desarrollo de software, aplicando y combinando notaciones gráficas de UML con fundamentos formales de MOF y que puedan además ser intercambiados entre herramientas del ciclo de vida utilizando XMI.

Un factor de fuerte influencia en la selección de estos estándares es el grado de formalismo del lenguaje de modelado utilizado. Combinando las ventajas de las notaciones gráficas instituidas en UML y en los formalismos matemáticos exactos, resultando muy fácil para el usuario interactuar con un modelo gráfico, confiando en la base formal que el esquema de modelos subyacente le proporciona. [30]

De este modo se introduce la precisión al proceso de modelado en un ámbito de desarrollo de software, conservando la usabilidad y la aceptación por parte de los desarrolladores. Disponer de estas ventajas conduce a una mejor compresión de conceptos, permitiendo un uso eficaz de las tecnologías involucradas.

El uso de UML y MOF permite la representación del sistema mediante diagramas, esto es una gran ventaja al momento de generar los modelos en MDA ya que nos permite por ejemplo definir funciones de cada actor del sistema en función del papel que desempeña dentro del sistema. Esta característica, junto con XMI, nos proporciona un gran punto de apoyo para la generación de PIM a código.

MOF nos permite hacernos una idea del diseño del Sistema, básicamente provee un marco de trabajo de gestión de sistemas dirigidos por modelos y metadatos. Además a partir de MOF surgen los conceptos de PIM y el mapeo de este a plataformas específicas.

De igual forma es mediante los diagramas que se puede explicar mejor al cliente, en especial al finalizar el proyecto donde muchas veces no se tiene la información necesaria para visualizar un flujo de información en el documento del sistema.

Por otro lado, una buena documentación es importante para la extensión de la vida útil de un producto, resultado de un proyecto de software, y el uso combinado de los

estándares UML, OCL y QVT permite generar la documentación necesaria (parte de una de la grandes ventajas de usar MDA) para que el cliente puede hacer uso de ella en un futuro con el fin de actualizar el sistema o simplemente darle mantenimiento.

3.3.1 Estándares OMG y la arquitectura de meta modelado de cuatro capas.

Tras ver los estándares que representan todos los conceptos de MDA, cabe preguntarse dónde encaja cada uno de ellos dentro de la arquitectura de cuatro capas que soporta la aproximación MDA.

Los estándares seleccionados, se van a ubicar básicamente en las capas de meta-metamodelo (M3) y metamodelo (M2), aunque alguno de ellos puede estar situado incluso a nivel del modelo (M1), ya que pueden ser utilizados para complementar los modelos definidos. Tal es el caso del OCL que puede ser utilizado para realizar restricciones a nivel M3, M2 y/o M1. Por tanto, OCL es un estándar difícil de situar en un nivel concreto de la arquitectura, ya que al igual que puede utilizarse a nivel M1, está presente en los niveles M2 y M3. OCL no es el único, por lo que en los casos que esta situación se dé, su representación en la arquitectura estará presente en varios niveles a la vez.

Como se ha comentado más arriba, el núcleo de toda la arquitectura dirigida por modelos es la Infrastructure Library, en concreto el paquete Core. Éste forma el núcleo de los estándares MOF, UML y CWM. No obstante MOF es el meta-metamodelo que permite definir lenguajes de modelado o metamodelos, por lo que, a pesar de compartir el paquete Core, MOF se encuentra a nivel M3 y UML y CWM, al ser instancias de MOF, se encuentran en el nivel M2 de la arquitectura MDA (Figura 9).

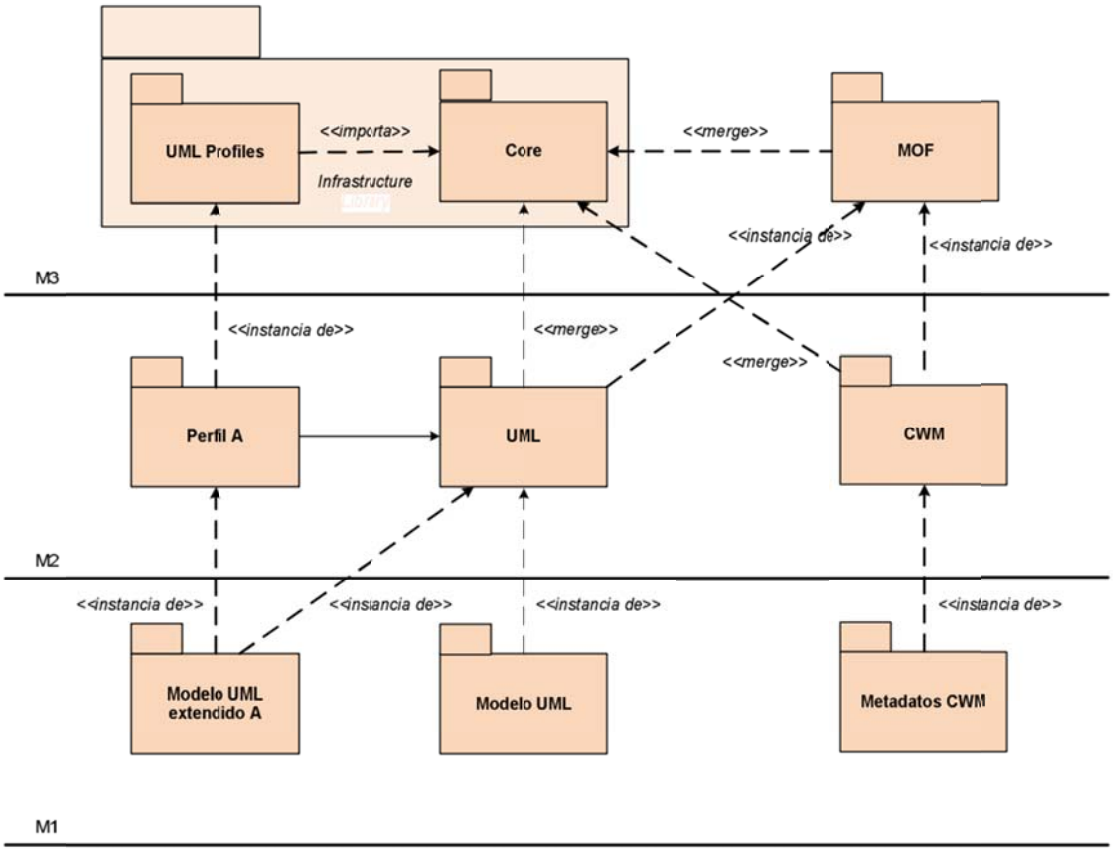


Figura 9: Infraestructura Común de los Estándares MDA.

El paquete UML Profiles pertenece a la Infrastructure Library, por lo que se encuentra a nivel de meta-metamodelo. Partiendo de ellas se pueden instanciar perfiles concretos (M2), mediante los que crear modelos UML extendidos (M1).

El estándar XMI (Figura 10), aparece a diferentes niveles de la arquitectura de cuatro capas de MDA, aunque la definición de las reglas XMI están definidas a nivel M3, donde se definen las transformaciones necesarias para generar los esquemas XML (ficheros .xsd) partiendo de un metamodelo basado en MOF. Aplicando los esquemas generados sobre instancias concretas, se generarán los ficheros XML que permitan exportarlos a otras herramientas. En definitiva, aunque la definición de las reglas se encuentre a nivel M3, el estándar XMI está presente además, en los niveles M2 y M1 de la arquitectura.

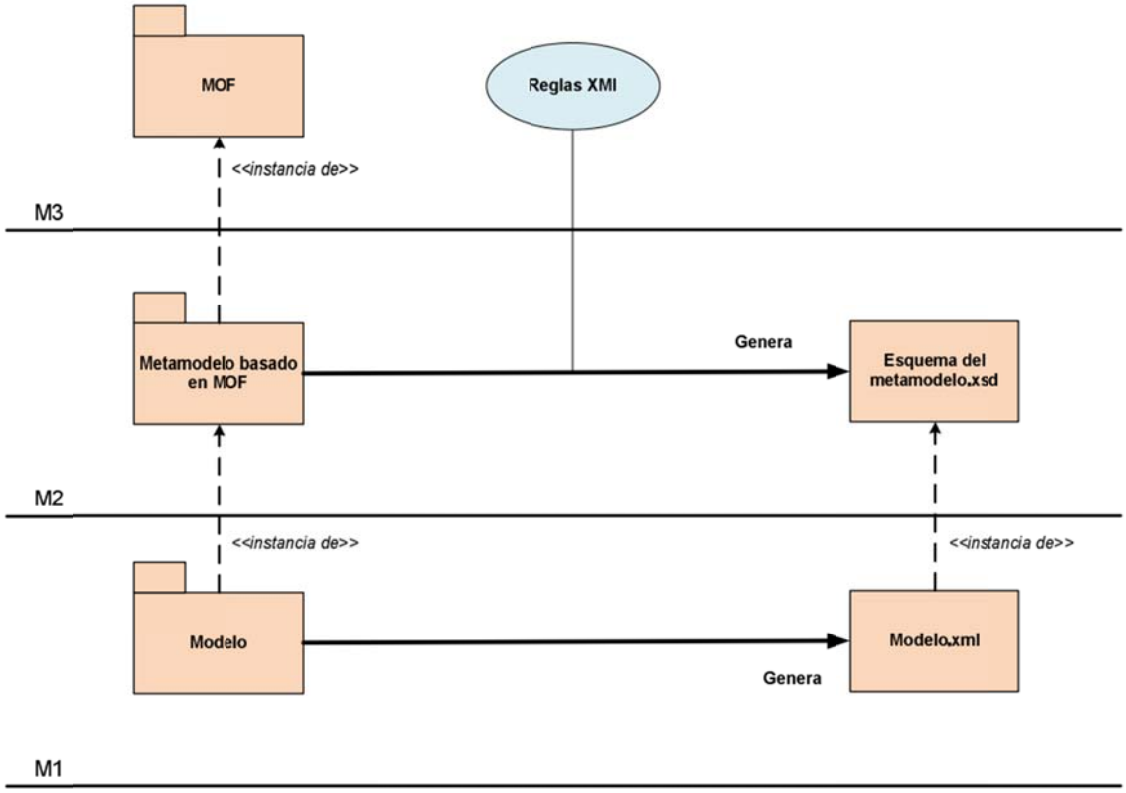


Figura 10: Intercambio entre Modelos MDA.

El estándar OCL llega incluso a abarcar tres niveles, ya que mediante este lenguaje, se pueden crear consultas y restricciones sobre meta-metamodelos, metamodelos y modelos. Un ejemplo es el uso de OCL para crear restricciones a la hora de definir nuevos lenguajes de modelado, mediante perfiles UML. Igualmente el propio QVT se puede situar en dos de los niveles (M3 yM2). La definición de la sintaxis está a nivel M3, pero como se ha comentado más arriba, las transformaciones se definen a nivel de metamodelos mediante las reglas QVT definidas según QVT.

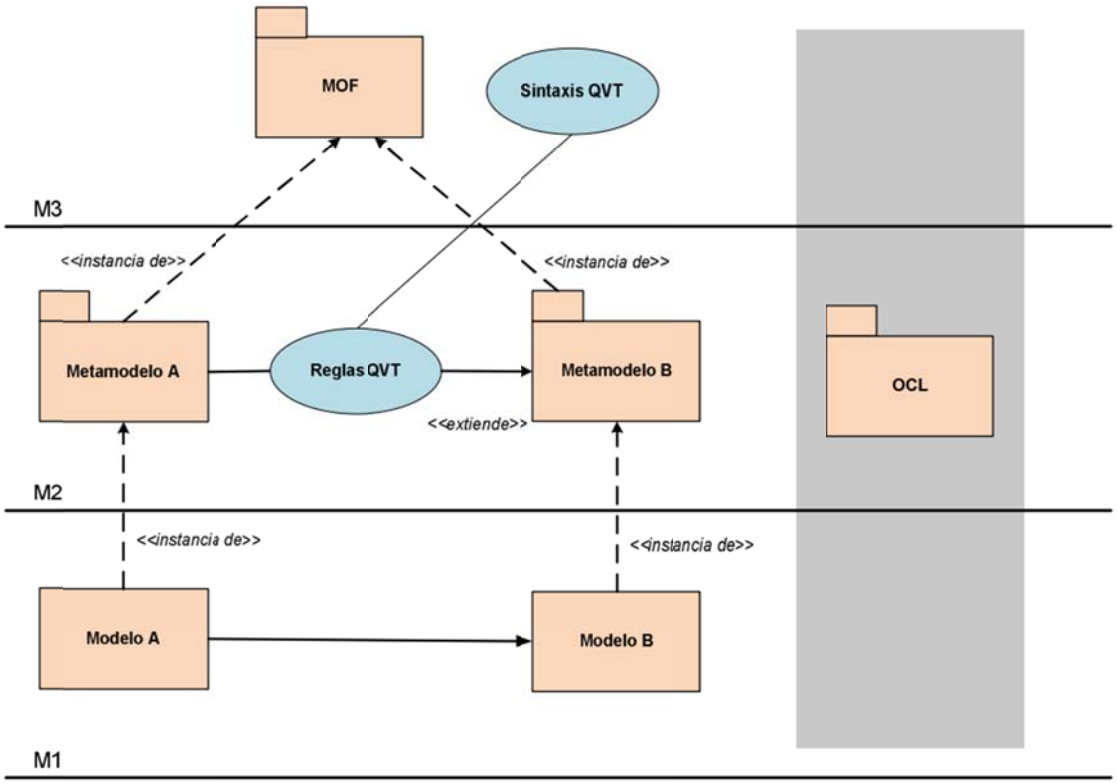


Figura 11: Transformación de Modelos MDA.

En la Figura 11 se observa que las reglas de transformación entre el metamodelo A y el B, se definen a nivel M2, aunque se apliquen sobre los modelos a nivel M1. Un motor de transformación, al ejecutar las reglas QVT definidas, traducirá el Modelo A generando el Modelo B.

Por tanto podemos concluir que los estándares MDA seleccionados para este estudio poseen la información más importante de este enfoque de desarrollo, brindando de esta manera los conocimientos necesarios para llevar a cabo un proyecto de desarrollo de software con MDA, así como también ayuda a la selección de herramientas necesarias para el proceso.

3.4 Etapas de Análisis y Diseño para la Selección de estándares MDA

3.4.1 Análisis de Sistemas de Información

La metodología de selección de estándares en la etapa de Análisis del Sistema, se desarrolla tomando como insumo los productos generados en la fase del estudio de viabilidad del sistema. Las actividades en esta fase tienen como fin traducir y establecer todas las necesidades de los usuarios en el Modelo Independiente de Plataforma (PIM), para que de esta manera sirva como base para el proceso de Diseño del Sistema. En esta etapa se consideran 7 actividades que se describen a continuación de forma independiente. Para una visión integrada referirse a la Figura # 12.

Actividad 1 DEFINICIÓN DEL SOFTWARE

Tarea 1.1: Determinación del alcance del software.
Tarea 1.2: Identificación de las clases (atributos y métodos).
Tarea 1.3: Especificación de estándares y normas organizativos.
Tarea 1.4: Especificación de estándares y normas TIC.

Actividad 2 ANÁLISIS DEL DOMINIO

Tarea 2.1: Validación de los requerimientos del usuario.
Tarea 2.2: Definición de los requerimientos del sistema en el modelo PIM.
Tarea 2.3: Especificación de escenarios de casos de uso.
Tarea 2.4: Clasificar los elementos del dominio.

Actividad 3 DEFINICIÓN DE COMPONENTES DEL DOMINIO

Tarea 3.1: Definir los componentes del dominio.
Tarea 3.2: Análisis los componentes del dominio.

Actividad 4 ANÁLISIS DE LOS MODELOS Y CLASES

Tarea 4.1: Identificación de clases asociadas a un caso de uso.
Tarea 4.2: Descripción de la interacción de objetos.
Tarea 4.3: Identificación de responsabilidades y atributos.
Tarea 4.4: Identificación de asociaciones y agregaciones.
Tarea 4.5: Identificación de generalizaciones.

Actividad 5 DEFINICIÓN DE INTERFACES DE USUARIO.

Tarea 5.1: Análisis y modelado de las tareas de la interfaz.
Tarea 5.2: Especificación de los escenarios de usuarios.
Tarea 5.3: Especificación del comportamiento dinámico de la interfaz.
Tarea 5.4: Especificación de formatos de impresión.

Actividad 6 ESPECIFICACIÓN DEL PLAN DE PRUEBAS DEL SISTEMA

Tarea 6.1: Definición del alcance de las pruebas.

Tarea 6.2: Definición de requisitos del entorno de pruebas.

Tarea 6.3: Definición de pruebas-resultados y aceptación del sistema.

Actividad 7 APROBACIÓN DEL ANÁLISIS DEL SISTEMA DE INFORMACIÓN

Tarea 7.1: Presentación y aprobación del análisis del sistema de información.

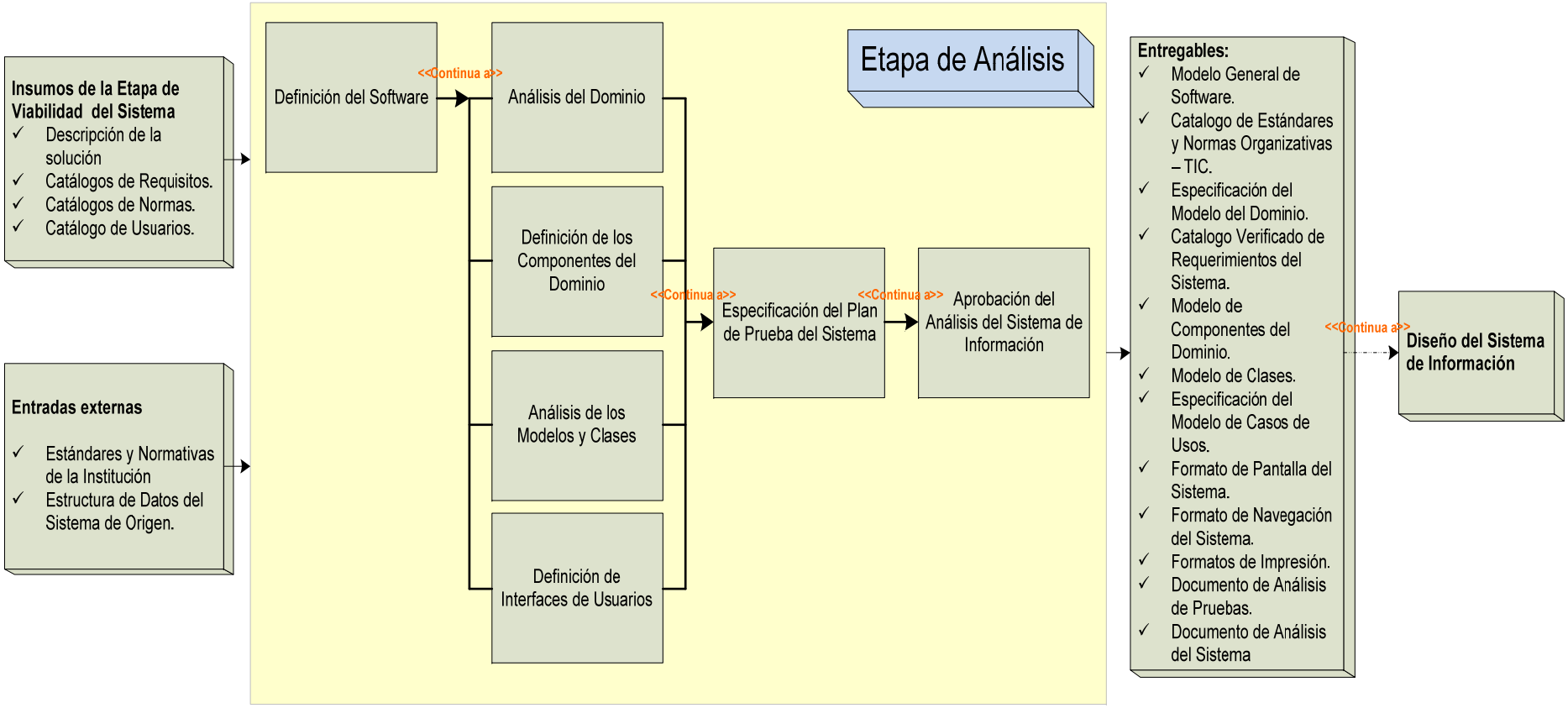


Figura 12: Análisis del Sistema de Información.

Actividad 1 DEFINICIÓN DEL SOFTWARE

Propósito de la actividad:

Esta actividad consiste en hacer una descripción del software, estableciendo su alcance y sus limitaciones. Muchas de las tareas de esta actividad están basadas en el estudio de viabilidad, por tanto se comienza a partir de esa idea.

Entregable:

- Modelado General del Software.
- Catálogo de Estándares y Normas Organizativas - TIC.

Tarea 1.1: Determinación del Alcance del Software

- Descripción:

  Para esta tarea se delimita el sistema de información partiendo del modelo de negocios del estudio de viabilidad, con el propósito de validar los requerimientos del usuario, empleando las diferentes técnicas UML.

- Meta:

  Identificación de los procesos que pertenecen al sistema de información así como de las entidades externas al sistema que aportan o reciben información.

Técnicas	Herramientas	Estándares
• **Diagrama de Flujos de Datos** • **Diagrama Entidad/Relación.**	• **EnterpriseArchitect, MagicDraw UML**	• **UML**

Tarea 1.2: Identificación de las Clases (atributos y métodos)

- Descripción:

  Esta tarea consiste en describir las clases del sistema, identificando las responsabilidades, sus atributos y las relaciones existentes entre ellas. Para ello se debe de tomar en cuenta el Modelo de Negocios de tal manera que cumpla con los criterios del mismo para evitar posibles inconsistencias en el diseño.

- Meta:

  Modelo de Clases del Sistema de Información.

Técnicas	Herramientas	Estándares
• **Diagrama de Clases.**	• **Enterprise Architect, MagicDraw UML.**	• **UML** • **MOF**

Tarea 1.3: Especificación de Estándares y Normas Organizativos

- Descripción:

  La puesta en marcha de esta tarea permite tener en cuenta los aspectos desde la perspectiva de las normas y especificaciones a considerar en el análisis del sistema de información, desde el punto de vista de la organización.

- Meta:

  Catálogo de Estándares y Normas Organizativas.

Técnicas	Herramientas	Estándares
• **Catalogación** • **Control de la Documentación del Negocio.**	• **Normas y estándares relacionados con la Organización.**	**No Aplican.**

Tarea 1.4: Especificación de Estándares y Normas TIC

- Descripción:

  Esta tarea es similar a la tarea anterior, la diferencia está en que se consideraran las normas y especificaciones que estén estrechamente relacionados con los aspectos TIC del análisis del software.

- Meta:

  Catálogo de Estándares y Normas TIC.

Técnicas	Herramientas	Estándares
• **Catalogación** • **Control de la Documentación TIC.**	• **Normas y estándares relacionados con los aspectos Técnicos.**	**No Aplican.**

Actividad 2 ANÁLISIS DEL DOMINIO

Propósito de la actividad:

En esta actividad se pretende analizar, validar y establecer los requisitos del software, a partir de la información brindada por el cliente y los catálogos de Estándares y Normas Organizativos y TIC.

Entregable:

- Especificación del Modelo de Dominio.
- Catalogo Verificado de Requerimientos del Sistema.

Tarea 2.1: Validación de los Requerimientos del Usuario

- Descripción:

  En esta tarea se validan los requerimientos detallados obtenidos del trabajo con los usuarios. De esta manera se obtienen los requisitos que debe de seguir el software, así como de las prioridades de dichos requisitos. Para lograr esta tarea es necesario el uso de la técnica de catalogación con el propósito de identificar los actores y sus actividades.

- Meta:

  Catálogo Detallado de Requerimientos del Usuario.

Técnicas	Herramientas	Estándares
• **Catalogación.** • **Control de la Documentación de Requerimientos.**	**Normas y estándares de la institución.**	**No Aplican**

Tarea 2.2: Definición de los Requerimientos del Sistema en el Modelo PIM

- Descripción:

  La definición y modelado de los requerimientos del sistema permite establecer los criterios del sistema, posibles restricciones del entorno tanto Hardware como software, que afecten al sistema de información. Dentro de los principales requisitos que se deben validar están los de funcionamiento, rendimiento, seguridad, implantación y disponibilidad del sistema. Para el modelado de los requisitos se hará uso de la técnica de casos de usos, que surge en gran manera de los estándares y normas, tanto organizativos como técnicos (Tarea 1.3 y Tarea 1.4), de igual manera se toma en cuenta la Determinación del Alcance del Sistema (Tarea 1.1), de forma que todos los requisitos especificados estén dentro del ámbito del análisis del sistema.

- Meta:

  Catalogo Actualizado de Requisitos del Sistema.

  Modelo de Casos de Usos tomando en cuenta los Requerimientos del Sistema.

Técnicas	Herramientas	Estándares
• **Especificación de Casos de Usos** • **Catalogación** • **Control de la Documentación de Requisitos**	• **Enterprise Architect, MagicDraw UML.**	• **UML** • **MOF**

Tarea 2.3: Especificación de Escenarios de Casos de Uso

- Descripción:

  El objetivo esencial de esta tarea es la especificación de escenarios de casos de uso, que se identificaron en tareas anteriores. Para la realización de esta tarea es necesario hacer una descripción de los escenarios (como un actor interactúa con el sistema), condiciones de fallo del sistema y como este debe de responder a esas situaciones. Como técnica a utilizar en la descripción de escenarios complejos, está la utilización de Diagrama de Transición de Estados. Cabe mencionar que para la obtención de esta información es necesaria la participación activa de los usuarios.

- Meta:

  Especificación de Escenarios de Casos de Usos.

Técnicas	Herramientas	Estándares
• **Diagrama de Transición de Estados.**	• **Enterprise Architect, MagicDraw UML.**	• **UML** • **MOF** • **OCL**

Tarea 2.4: Clasificar los Elementos del Dominio

- Descripción:

  El análisis de los requerimientos y del Modelo de Casos de Uso, permite identificar y clasificar las funcionalidades o comportamientos comunes, esto permite reestructurar los Casos de Usos a través de las generalizaciones evitando inconsistencias, ambigüedades, duplicidad o simplemente falta de información.

- Meta:

  Especificación del Catálogo de del Sistema.
  Especificación actualizada de Casos de Usos.

Técnicas	Herramientas	Estándares
• **Modelo de casos de Uso.** • **Diagramas de Transición de Estados**	• **Enterprise Architect, MagicDraw UML.**	• **UML** • **MOF** • **OCL**

Actividad 3 DEFINICIÓN DE COMPONENTES DEL DOMINIO

Propósito de la actividad:

El Objetivo esencial de esta actividad es facilitar el análisis del sistema de información definiendo los diferentes Componentes del Dominio.

Entregable:

- Modelado de Componentes del Dominio.

Tarea 3.1: Definir los Componentes del Dominio

- Descripción:

  La descomposición del sistema en componentes debe, en su gran mayoría, estar ligada a los procesos del negocio con el propósito de identificar los requisitos comunes del dominio de aplicación específica, normalmente para su reutilización dentro del mismo dominio de aplicación.

  Para llevar a cabo esta tarea el analista debe de identificar aquellas áreas del negocio que son de interés en el dominio.

- Meta:

  Descripción de los Componentes del Sistema y sus Interfaces.

Técnicas	Herramientas	Estándares
• **Diagrama de Procesos de Negocios** • **Diagrama de Paquetes.**	• **Enterprise Architect.** • **MagicDraw.**	• **UML** • **MOF**

Tarea 3.2: Análisis de los Componentes del Dominio

- Descripción:

  El objeto de esta tarea es el análisis y la coordinación de los componentes del dominio.

  Al igual que la tarea anterior el producto de esta tarea es la descripción de los componentes del sistema solo que en esta parte se detallan los procesos con el propósito de asegurar la ausencia de duplicidad de los componentes, permitiendo tener una visión global y unificada de todos los componentes.

- Meta:

  Análisis de la Descripción de los Componentes del Sistema y sus Interfaces.

Técnicas	Herramientas	Estándares
• **Diagrama de Procesos de Negocios** • **Diagrama de Paquetes.**	• **Enterprise Architect.** • **MagicDraw.**	• **UML** • **MOF** • **OCL**

Actividad 4 ANÁLISIS DE LOS MODELOS Y CLASES

Propósito de la Actividad:

El objetivo de esta actividad es identificar las clases cuyos objetos son necesarios para realizar los Casos de Uso y describir las iteraciones de dichos objetos. Esta actividad se lleva a cabo para cada uno de los Componentes del Dominio. Es muy importante señalar que las tareas de esta actividad no se desarrollan en forma secuencial, sino en paralelo de tal forma que cada una de ellas se retroalimenten.

Entregable:

- Modelo de Clases.
- Especificación del Modelo de Casos de Uso.

Tarea 4.1: Identificación de Clases Asociadas a un Caso de Uso

- Descripción:

  A partir de las Especificaciones del Caso de Uso se extrae una lista de objetos candidatos a ser clases. Algunos de los objetos representan mejor la información del sistema si se identifican como atributos en vez de clases. Una técnica propuesta para poder diferenciarlos es el estudio del Diagrama de Iteración y Diagramas de Transición de Estados y además el tener en cuenta una serie de reglas, como puede ser suprimir palabras no convenientes como sinónimos o significados ambiguos.

  Una vez identificadas las clases se generan y se incorporan al Modelo de Clases donde se identifican sus atributos, responsabilidades y relaciones.
  Las clases que se identifican en esta tarea pueden ser:

  - ✓ Clases de Entidad (representan la información manipulada en el caso de uso).
  - ✓ Clases de Interfaz de Usuario (se utilizan para describir la interacción entre el sistema y sus actores. Suelen representar abstracciones de ventanas, interfaces de comunicación, formularios, etc.).
  - ✓ Clases de Control (son responsables de la coordinación, secuencia de transacciones y control de los objetos relacionados con un caso de uso).

  Es posible que inicialmente no se disponga de la información necesaria para identificar todas las clases, por lo que se hace una pequeña aproximación que se irá refinando a lo largo de toda esta actividad y en el proceso de diseño del sistema.

- Meta:

  Modelos de Clases de Análisis.

  Comportamiento de Clases de Análisis.

Técnicas	Herramientas	Estándares
• **Diagrama de Interacción.** • **Diagrama de Transición de Estados.**	• **Enterprise Architect.** • **MagicDraw.**	• **UML** • **MOF** • **OCL**

Tarea 4.2: Descripción de la Interacción de Objetos

- Descripción:

  El objetivo de la tarea es describir las cooperaciones de los objetos utilizados en los casos de usos. Para representar esta información, se usan Diagramas de Interacción que contienen instancias de los objetos, y la secuencia de mensajes intercambiados entre ellos. Se pueden establecer criterios para determinar qué tipo de objetos y mensajes se va a incluir en este diagrama, como por ejemplo: si se incluyen objetos y llamadas a bases de datos, objetos de interfaz de usuario, de control, etc. Estos diagramas pueden ser tanto de secuencia como de colaboración, y su uso depende de si se quieren centrar en la secuencia cronológica o en cómo es la comunicación entre los objetos.

  Esta tarea es una continuación de la clase anterior puesto que se trabaja sobre el mismo Modelo de Clases de Análisis.

- Meta:

  Descripción del Modelo de Clases de Análisis.

Técnicas	Herramientas	Estándares
• **Diagramas de Interacción.** • **Diagramas de Secuencia.** • **Diagramas de Colaboración.**	• **Enterprise Architect.** • **MagicDraw.**	• **UML** • **MOF** • **OCL**

Tarea 4.3: Identificación de Responsabilidades y Atributos

- Descripción:

  Las Responsabilidades de una clase define la funcionalidad de la misma en el sistema, esta se obtiene a partir del estudio de las funciones que desempeñan los objetos dentro de los diferentes Casos de Usos. Por otro lado los atributos especifican las propiedades de las clases y se identifican por estar involucrados con las responsabilidades, los tipos de estos atributos deben de ser conceptuales y conocidos en el dominio. Como

recomendación se propone es uso de Diagrama de Estado para aquellas clases que cuyo comportamiento depende del estado en que se encuentre.

- Meta:

  Comportamiento de Clases de Análisis.

Técnicas	Herramientas	Estándares
• **Diagramas de Clases.** • **Diagrama de Estados.**	• **Enterprise Architect.** • **MagicDraw.**	• **UML** • **MOF** • **OCL**

Tarea 4.4: Identificación de Asociaciones y Agregaciones

- Descripción:

  En esta tarea se identifican los mensajes establecidos por los objetos en el Diagrama de Iteración con el fin de determinar las Asociaciones entre las clases.

  Las Relaciones surgen como respuesta a las demandas de los distintos Casos de Usos por ello es necesario la definición de las agregaciones y herencia entre los objetos.

  Una Asociación se caracteriza por:

  - ✓ La función que desempeña.
  - ✓ Su direccionalidad, que representa el sentido en el que se debe interpretar.
  - ✓ Su cardinalidad, que representa el número de instancias implicadas en la asociación.

  Dichas características pueden obtenerse a partir de la especificación de los Casos de Uso.

- Meta:

  Identificación de las Asociaciones y Agregaciones del Comportamiento de clases de Análisis.

Técnicas	Herramientas	Estándares
• **Diagramas de Clases.** • **Análisis de la Realización del Modelo de Casos de Usos.**	• **Enterprise Architect.** • **MagicDraw.**	• **UML** • **MOF**

Tarea 4.5: Identificación de Generalizaciones

- Descripción:

  El objetivo de esta Tarea es presentar una esquematización de las diferentes clases que permita la implementación de la herencia, así como de una agrupación semántica de las clases conforme a las normas y estándares Técnicos y Organizativos.

- Meta:

  Especificación del Modelo de Clases de Análisis.

Técnicas	Herramientas	Estándares
• **Diagramas de Clases.**	• **Enterprise Architect.** • **MagicDraw.**	• **UML** • **MOF**

Actividad 5 DEFINICIÓN DE INTERFACES DE USUARIO

Propósito de la actividad:

Para esta actividad el analista identifica las interfaces entre el sistema y el usuario, los formatos de navegación y los formatos de impresión. Para esta actividad es necesaria una iteración con el usuario con el propósito de satisfacer sus necesidades y demandas.

Se identifican los distintos grupos de usuarios de acuerdo a las funciones que realizan, conocimientos y habilidades que poseen, así como de las características del entorno en que trabajan, de esta manera se asegura identificar las mejor las necesidades y particularidades de los usuarios.

Finalmente se definen el formato y contenido de cada una de las interfaces así como de los modelos de navegación dinámica del sistema.

Entregable:

- Formatos de Pantallas del Sistema.
- Formatos de Navegación del Sistema.
- Formatos de Impresión.

Tarea 5.1: Análisis y Modelado de las tareas de la Interfaz

- Descripción:

  El objetivo de esta tarea es especificar las normas, directrices y elementos generales a tener en cuenta en la definición de la interfaz de usuario, tanto

para la interfaz interactiva (gráfica o carácter), como para los informes y formularios impresos.

Entre algunas normas y principios que se pueden analizar están:

- ✓ Normas para la presentación de la ayuda.
- ✓ Normas de impresión.
- ✓ Normas de Navegación.
- ✓ Principios de descomposición general y ubicación de los elementos en la pantalla.

- Meta:

  Documento de principios o directrices general de la Interfaz de Usuario.

Técnicas	Herramientas	Estándares
• **Control de documentación de normas a definir.**	**No aplican**	**No Aplican**

Tarea 5.2: Especificación de los Escenarios de Usuarios

- Descripción:

  El objetivo de esta tarea es identificar los formatos individuales de la Interfaz de usuario estática. En este caso los formatos individuales van completando las especificaciones de los Casos de Usos.

  En la definición de cada interfaz de pantalla se deben definir aquellos aspectos considerados de interés para su posterior diseño y construcción:

  - ✓ Posibilidad de cambio de tamaño, ubicación, modalidad.
  - ✓ Conjunto y formato de datos asociados, identificando qué datos se usan y cuáles se generan como consecuencia de su ejecución.
  - ✓ Controles y elementos de diseño asociados, indicando cuáles aparecen inicialmente activos e inactivos al visualizar la interfaz de pantalla.

- Meta:

  Identificación de Formatos Interfaz de Usuario Individuales.

  Documento de Control y Elementos Asociados a los Formatos de Interfaz de Usuario Individuales.

Técnicas	Herramientas	Estándares
• **Control de documentación de formatos de interfaz.** • **Casos Usos.**	• **No Aplican**	• **UML** • **MOF**

Tarea 5.3: Especificación del Comportamiento Dinámico de la Interfaz

- Descripción:

  El objetivo de esta Tarea es definir los flujos entre la interfaz de pantallas, así como de las pantallas mismas. Este tipo de comportamiento se define a través un modelo de navegación de interfaz de pantalla.

  Para cada formato individual desarrollado en la Tarea 5.2 se establece la entrada lógica de los datos y la regla de validación. Para esta tarea se recomienda el uso de la Técnica de Diagrama de Estados.

- Meta:

  Modelo de Navegación de Interfaz de Pantalla.

Técnicas	Herramientas	Estándares
• **Diagrama de de Transición de Estados.** • **Diagrama de Interacción.**	• **Enterprise Architect.** • **MagicDraw.** • **Microsoft Visio.**	• **UML** • **MOF**

Tarea 5.4: Especificación de Formatos de Impresión

- Descripción:

  El objetivo de esta tarea es especificar los formatos de impresión de las salidas (o entradas) impresas al sistema, se definen los formatos individuales de informes de formularios, así como sus características principales, entre las que se especifican la periodicidad, confidencialidad, procedimientos de entrega o difusión, y salvaguarda de la copia.

- Meta:

  Prototipo de Interfaz de Impresión.
  Formatos de Impresión.

Técnicas	Herramientas	Estándares
• **Prototipos.**	• **Enterprise Architect.** • **MagicDraw.** • **Microsoft Visio.**	• **UML**

6 ESPECIFICACIÓN DEL PLAN DE PRUEBAS DEL SISTEMA

Propósito de la actividad:

En esta actividad se inicia la definición del plan de pruebas, el cual sirve como guía para la realización del diseño de las pruebas, y permite verificar que el sistema de información cumple las necesidades establecidas por el usuario, con las debidas garantías de calidad.

Entregable:

- Documento de Análisis de las Pruebas.

Tarea 6.1: Definición del Alcance de las Pruebas

- Descripción:

  En esta parte definimos los niveles de prueba del sistema. En esta tarea se especifican y justifican de los niveles de pruebas a realizar, así como el marco general de planificación de cada nivel de prueba, según el siguiente esquema:

  - ✓ Definición de los perfiles implicados en los distintos niveles de prueba.
  - ✓ Criterios de verificación y aceptación de cada nivel de prueba.
  - ✓ Definición, generación y mantenimiento de verificaciones y casos de prueba.
  - ✓ Análisis de los resultados de cada nivel de prueba.
  - ✓ Productos a entregar como resultado de la ejecución de las pruebas.

- Meta:

  Documento de Especificación del Plan de Pruebas.

Técnicas	Herramientas	Estándares
• **Sesiones de Trabajo.** • **Control de Documentación de Pruebas**	**No Aplican**	**No Aplican.**

Tarea 6.2: Definición de Requisitos del Entorno de Pruebas

- Descripción:

  El objetivo de esta tarea es la definición o recopilación de los requisitos relativos al entorno de pruebas, completando el plan de pruebas.

  Con la puesta en marcha de estas pruebas se aconseja disponer de un entorno de pruebas separado del entorno de desarrollo y del entorno de operación, entre algunos entornos de pruebas, se pueden mencionar los siguientes:

  - ✓ Requisitos básicos de hardware y software base: sistemas operativos, gestores de bases de datos, etc.
  - ✓ Requisitos de configuración de entorno: librerías, bases de datos, ficheros, procesos, comunicaciones, necesidades de almacenamiento, configuración de accesos, etc.
  - ✓ Herramientas auxiliares. Por ejemplo, de extracción de juegos de ensayo, análisis de rendimiento y calidad, etc.

- Meta:

  Documento de Requisitos del Entorno de Pruebas.

Técnicas	Herramientas	Estándares
• **Catálogos de requisitos.** • **Control de Documentación del Plan de pruebas anterior.**	**No Aplican**	**No Aplican.**

Tarea 6.3: Definición de Pruebas-Resultados y Aceptación del Sistema

- Descripción:

  En esta tarea se realiza la especificación de las pruebas de aceptación del sistema, labor fundamental para que el usuario valide el sistema, como último paso, previo al diseño del Sistema.

  Los criterios de aceptación deben ser definidos de forma clara, prestando especial atención a aspectos como:

  - ✓ Procesos críticos del sistema.
  - ✓ Rendimiento del sistema.
  - ✓ Seguridad.
  - ✓ Disponibilidad.

- Meta:

  Planificación Plan de Pruebas.

Técnicas	Herramientas	Estándares
• **Sección de Trabajo.** • **Control de Documentación del Plan de Pruebas.**	**No Aplican**	**No Aplican.**

7 APROBACIÓN DEL ANÁLISIS DEL SISTEMA DE INFORMACIÓN

Propósito de la actividad:

En esta actividad se realizara una revisión de la documentación y modelos de análisis para presentar el informe final del Análisis del Sistema de Información.

Entregables:

- Informe y Presentación del Documento de Análisis del Sistema de Información.

Tarea 7.1: Presentación y Aprobación del Análisis del Sistema de Información

- Descripción:

  En esta tarea se realiza la presentación del análisis del sistema al comité directivo para su aprobación.

- Meta:

  Presentar y Entregar el Análisis Completo del Sistema.

Técnicas	Herramientas	Estándares
• **Presentación Multimedia e Informes**	• **Power Point** • **Adobe Presenter.**	**No Aplican**

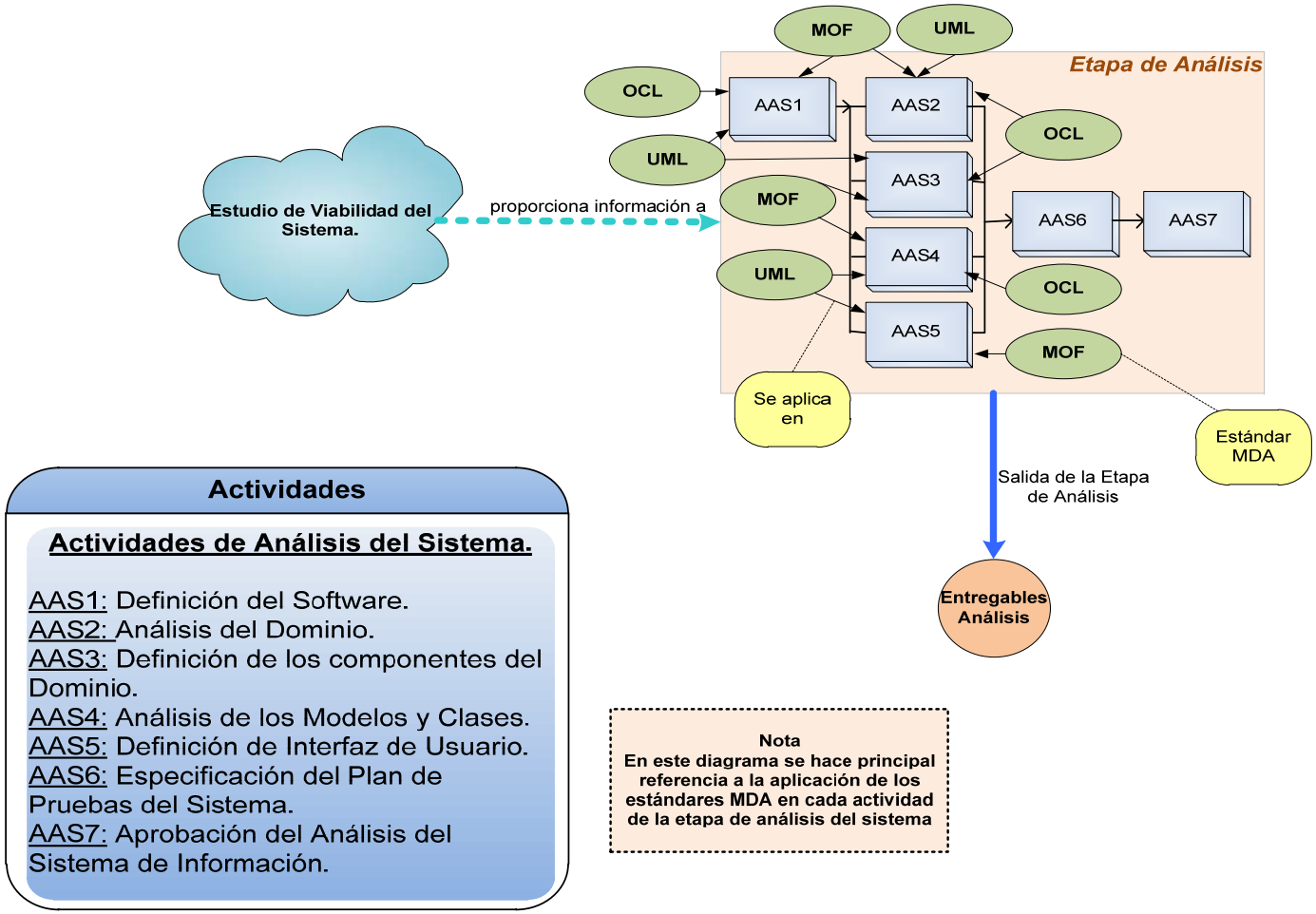


Figura 13: Diagrama General de la Metodología – Fase de Análisis

3.4.2 Diseño del sistema de información

La metodología de selección de estándares en la etapa de Diseño del Sistema, se desarrolla tomando como insumo los productos generados en la fase de Análisis del Sistema. Las actividades en esta fase tienen como fin la definición y transformación del modelo PIM a los modelos PSM requeridos a generar, de esta manera al finalizar esta fase estará completo el diseño del sistema, incluyendo la arquitectura del o los PSM, plan de pruebas y requisitos de implantación. En esta etapa se consideran 9 actividades que se describen a continuación de forma independiente. Para una visión integrada referirse a la Figura # 14.

Actividad 1 DEFINICIÓN DE LA ARQUITECTURA DEL SISTEMA

Tarea 1.1: Definición arquitectónica del diseño.
Tarea 1.2: Identificación y especificación de requisitos de diseño.
Tarea 1.3: Especificación de excepciones.
Tarea 1.4: Identificación de componentes del diseño.
Tarea 1.5: Especificación de requisitos de operación y seguridad.

Actividad 2 DISEÑO DE LA ARQUITECTURA DE COMPONENTES

Tarea 2.1: Diseño de componentes de soporte.

Actividad 3 DISEÑO DE CASOS DE USO

Tarea 3.1: Identificación de clases asociadas a un caso de uso.
Tarea 3.2: Diseño de la realización de los Casos de Uso.
Tarea 3.3: Diseño de la interfaz de usuario.
Tarea 3.4: Revisión de componentes de diseño e interfaces de usuario.

Actividad 4 DISEÑO DE OBJETOS (CAPA DE CLASES)

Tarea 4.1: Identificación de clases adicionales.
Tarea 4.2: Diseño de asociaciones y agregaciones.
Tarea 4.3: Identificación de atributos de las clases.
Tarea 4.4: Identificación de operaciones de las clases.
Tarea 4.5: Diseño de la jerarquía.

Actividad 5 DISEÑO DE LA GESTIÓN DE DATOS.

Tarea 5.1: Diseño del modelo de datos.
Tarea 5.2: Especificación de la distribución de datos.

Actividad 6 GENERACIÓN DE ESPECIFICACIONES DE CONSTRUCCIÓN

Tarea 6.1: Especificación del entorno de construcción.
Tarea 6.2: Definición de componentes y subsistemas de construcción.

Actividad 7 DISEÑO DEPRUEBAS DEL SISTEMA

Tarea 7.1: Definición de las pruebas.

Tarea 7.2: Revisión de la planificación de pruebas.

Actividad 8 ESTABLECIMIENTO DE REQUISITOS DE IMPLANTACIÓN

Tarea 8.1: Especificación de requisitos de documentación de usuario.

Tarea 8.2: Especificación de requisitos de implantación.

Actividad 9 APROBACIÓN DEL DISEÑO DEL SISTEMA DE INFORMACIÓN

Tarea 9.1: Presentación y aprobación del diseño del sistema de información.

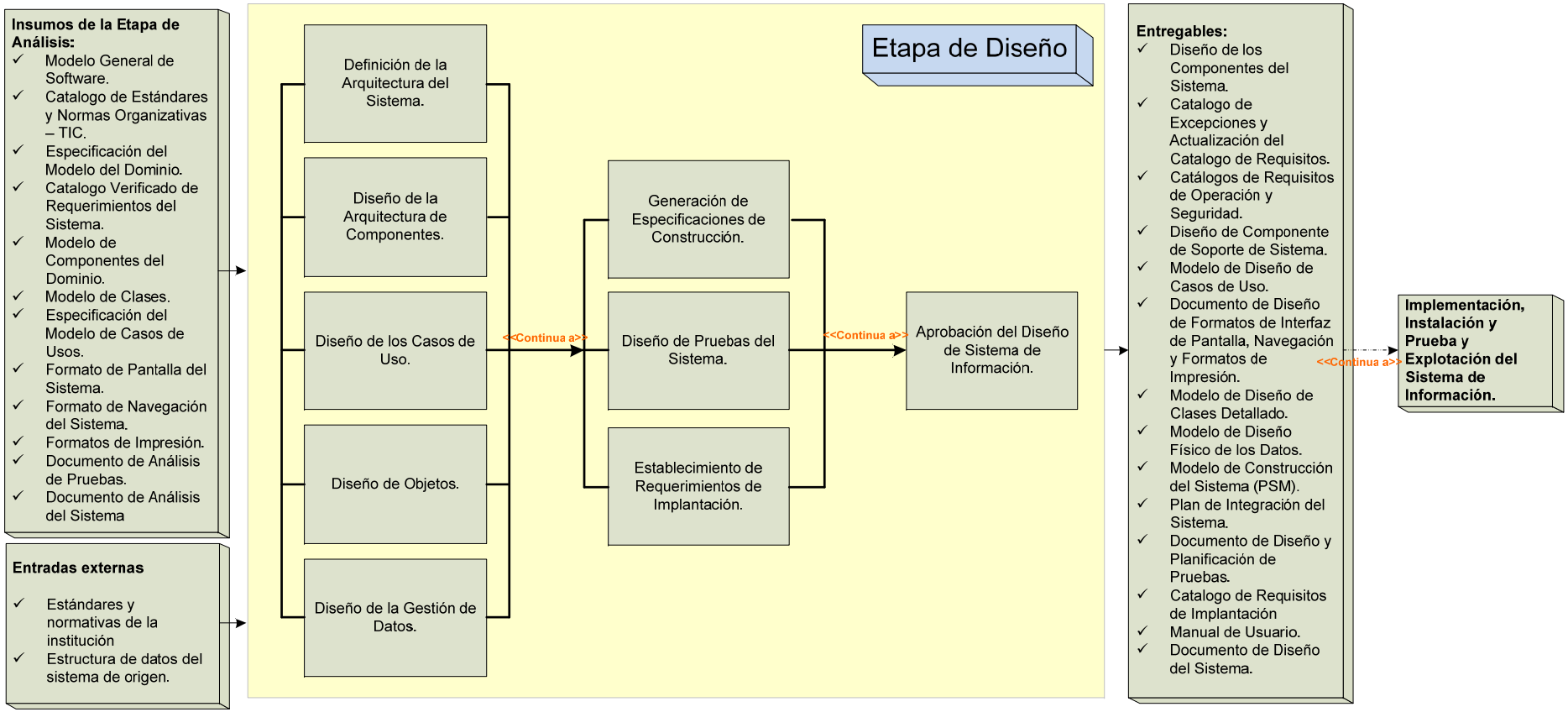


Figura 14: Diseño del Sistema de Información.

Actividad 1 DEFINICIÓN DE LA ARQUITECTURA DEL SISTEMA

Propósito de la actividad:

En esta actividad se define la arquitectura general del sistema de información a partir de su descomposición lógica en partes, las cuales cada una tendrá su especificación de diseño propia, con el fin de independizar, en la medida de lo posible, las funcionalidades que va a cubrir el sistema de información de la infraestructura que le dará soporte.

Entregables:

- Diseño de los Componentes del Sistema
- Catálogo de Excepciones y Actualización del Catálogo de Requisitos
- Catálogo de Requisitos de Operación y Seguridad

Tarea 1.1: Definición Arquitectónica del Diseño

- Descripción:

  Se definirán las principales particiones físicas del sistema de información con sus propias características, diseño y funciones, así como también los elementos de infraestructura más significativos.

- Meta:

  Diseño de la Partición Física del Sistema de Información.

Técnicas	Herramientas	Estándares
• **Diagramas de Despliegue y Representación.**	• **MagicDraw UML** • **Microsoft Visio**	• **UML** • **MOF**

Tarea 1.2: Identificación y Especificación de Requisitos de Diseño

- Descripción:

  En esta actividad se especificaran los requisitos que están relacionados con la adopción de una arquitectura o infraestructura concreta que puedan condicionar el diseño de la arquitectura del sistema de información. Entre algunos requisitos pueden estar: lenguajes, ubicación de los módulos y datos en las distintas partes del sistema.

- Meta:

  Actualización del Catálogo de Requisitos elaborado en la fase de Análisis del Sistema de Información.

Técnicas	Herramientas	Estándares
• **Sesiones de trabajo y Clasificación de los Requisitos.** • **Catalogación.**	**No Aplican**	**No Aplican**

• **Control de la Documentación de Requisitos.**		

Tarea 1.3: Especificación de Excepciones

- Descripción:

  Esta actividad tiene como objetivo la definición de todas las excepciones o comportamientos no habituales en el sistema, principalmente las que tienen que ver con el funcionamiento general de este. Entre las que encontramos: problemas de comunicación entre los componentes del sistema, disponibilidad de los gestores de base de datos, formatos específicos, datos de entrada no válidos, etc.
- Meta:

  Catálogo de Excepciones.

Técnicas	Herramientas	Estándares
• **Sesiones de Trabajo y Clasificación de las Excepciones.** • **Control de la Documentación de Excepciones.** • **Catalogación**	**No aplican**	**No aplican**

Tarea 1.4: Identificación de Componentes del Diseño

- Descripción:

  Esta actividad tiene como objetivo la división lógica del sistema en componentes de diseño para la reducción de la complejidad así como también para facilitar el mantenimiento. Primeramente se tomaran en cuenta los componentes de análisis especificados en el proceso de Análisis del Sistema de Información.
- Meta:

  Descripción y Modelado de los Componentes de Diseño.

Técnicas	Herramientas	Estándares
• **Catalogación los Componentes Específicos y de Soporte.** • **Diagramas de Despliegue.** • **Diagramas de Paquetes.** • **Diagrama de Interacción.**	• **MagicDraw UML**	• **MOF** • **UML**

Tarea 1.5: Especificación de Requisitos de Operación y Seguridad

- Descripción:

  Esta actividad tiene como objetivo la definición de los procedimientos necesarios para no comprometer el funcionamiento del sistema en todos sus niveles de operación. Se tomarán en cuenta el catálogo de requisitos ya definido, la arquitectura diseñada y el entorno tecnológico que se pretende utilizar. Los procedimientos que se definirán estarán divididos en los grupos de: control de acceso y seguridad, y procedimientos de operación para los distintos elementos del sistema (módulos, clases, sistema de ficheros, etc.)

- Meta:

  Documento de Procedimientos de Seguridad y Control de Acceso.
  Procedimientos de Operación y Administración del Sistema.

Técnicas	Herramientas	Estándares
• **Especificación los Requisitos en los Grupos de Seguridad y Operación.** • **Restricciones de comportamiento, usuarios y roles al modelo PIM desarrollado.** • **Control de la documentación de requisitos de operación y seguridad.**	• **MagicDraw UML**	• **MOF** • **UML** • **OCL**

Actividad 2 DISEÑO DE LA ARQUITECTURA DE SOPORTE

Propósito de la actividad:

En esta actividad se especifica la arquitectura de soporte que comprende el diseño de los componentes de soporte identificados en la actividad de Definición de la Arquitectura del Sistema. Así mismo se definen los mecanismos de reutilización y diseño genérico para cada componente.

Entregables:

- Diseño de los Componentes de Soporte del Sistema. (Actualización del Modelo de Diseño)

Tarea 2.1: Diseño de Componentes de Soporte

- Descripción:

  Esta actividad tiene como objetivo especificar y diseñar los módulos que forman parte de los componentes de soporte identificados en la tarea de

Identificación de Componentes de Diseño. El diseño sigue las mismas reglas y pautas establecidas para los Componentes Específicos del Sistema.

- Meta:

  Diseño de los Componentes de Soporte del Software.

Técnicas	Herramientas	Estándares
• **Diagrama de estructura** • **Diagrama de interacción** • **Diagrama de clases**	• **MagicDraw UML**	• **MOF** • **UML** • **OCL**

Actividad 3 DISEÑO DE CASOS DE USO

Propósito de la actividad

En esta actividad se especifica el comportamiento del sistema de información a partir de la definición de sus casos de uso. Algunos aspectos a tomar en cuenta en esta actividad son: clases participantes, interfaz de usuario, escenarios detallados, excepciones y requisitos de implementación que surjan.

Entregables:

- Modelo de Diseño de Casos de Uso (incluyendo diagramas de interacción y colaboración o secuencia).
- Documento de Diseño de Formatos de Interfaz de Pantalla, Navegación y Formatos de Impresión.

Tarea 3.1: Identificación de Clases Asociadas a un Caso de Uso

- Descripción:

  Esta actividad tiene como objetivo identificar las clases que intervienen en un caso de uso, así mismo agregar clases que puedan surgir al modelo de clases.

- Meta:

  Diseño detallado de Casos de Usos y Clases Asociadas.

Técnicas	Herramientas	Estándares
• **Diagrama de Casos de Uso** • **Diagrama de Interacción**	• **MagicDraw UML**	• **MOF** • **UML**

Tarea 3.2: Diseño de la Realización de los Casos de Uso

- Descripción:

  Esta actividad tiene como objetivo identificar cómo interactúan los objetos identificados en la tarea anterior y la definición de su comportamiento y diseño específico.

- Meta:

  Diagramas de Comportamiento e Interacción para los casos de usos.

Técnicas	Herramientas	Estándares
• **Diagrama de interacción.** • **Diagrama Colaboración o Secuencia.**	• **MagicDraw UML**	• **MOF** • **UML**

Tarea 3.3: Diseño de la Interfaz de Usuario

- Descripción:

  Esta actividad tiene como objetivo especificar el diseño detallado del comportamiento de la interfaz de usuario a partir de la especificación ya hecha en la fase de análisis. Los elementos necesarios a definir en esta tarea son: interfaz de usuario, navegación, elementos que forman cada interfaz, características, gestión de los eventos, etc.

- Meta:

  Diseño de los Formatos Individuales de Pantalla, Controles y elementos de Diseño.
  Modelo de Navegación de Interfaz.
  Formatos de Impresión.

Técnicas	Herramientas	Estándares
• **Diagrama de interacción de objetos.** • **Diagrama de Transición de Estados.** • **Prototipado.** • **Control de Documentación de Interfaces de Usuario.**	• **MagicDraw UML**	• **MOF** • **UML**

Tarea 3.4: Revisión de Componentes de Diseño e Interfaces de Usuario

- Descripción:

  Esta actividad tiene como objetivo identificar cada caso de uso con otro y los mensajes que intercambian para verificar y detallar las interfaces de cada componente, de esta manera se completa la Identificación de componentes de diseño.

- Meta:

  Diseño de Interfaces de Componentes en los casos de Usos.

Técnicas	Herramientas	Estándares
• **Diagrama de interacción** • **Diagrama de Casos de Usos.**	• **MagicDraw UML**	• **MOF** • **UML** • **OCL**

Actividad 4 DISEÑO DE OBJETOS (CAPA DE CLASES).

Propósito de la actividad:

En esta actividad se específica de manera detallada el modelo de clases proveniente del análisis y se convierte en un modelo de clases de diseño que incluye: atributos, operaciones, métodos y las relaciones precisas entre ellas. También se toman en cuenta la estructura física de los datos que manejara cada clase y la jerarquía entre estas.

Entregables:

- Modelo de Diseño de Clases. (Incluyendo Diagrama de Transición de Estados)

Tarea 4.1: Identificación de Clases Adicionales

- Descripción:

  Esta actividad tiene como objetivo identificar el conjunto de clases que completen el modelo de clases, algunas de estas pueden ser clases de control, de entidades, de interfaz, abstractas que definen características comunes a otras.

- Meta:

  Diseño de las Clases adicionales para el Modelo de Clases.

Técnicas	Herramientas	Estándares
• **Diagrama de clases.**	• **MagicDraw UML**	• **MOF** • **UML** • **QVT**

Tarea 4.2: Diseño de Asociaciones y Agregaciones

- Descripción:

  Esta actividad tiene como objetivo completar las asociaciones que fueron identificadas en el Análisis, tomando en cuenta también la secuencia de

mensajes entre los objetos definida en los diagramas de interacción anteriores.

- Meta:

  Identificación y modelación de las Asociaciones en el Modelo de Clases.

Técnicas	Herramientas	Estándares
• **Diagrama de clases.**	• **MagicDraw UML**	• **MOF** • **UML**

Tarea 4.3: Identificación de Atributos de las Clases

- Descripción:

  Esta actividad tiene como objetivo identificar y describir los atributos de cada clase a partir del modelo de análisis (PIM) obtenido. Para cada atributo se define su tipo, formatos específicos, restricciones y atributos adicionales que se generen a partir del mismo.

- Meta:

  Descripción del Modelo de Clases con todos los atributos correspondientes.

Técnicas	Herramientas	Estándares
• **Diagrama de clases.**	• **MagicDraw UML**	• **MOF** • **UML** • **OCL**

Tarea 4.4: Identificación de Operaciones de las Clases

- Descripción:

  Esta actividad tiene como objetivo definir las operaciones de cada clase a partir del modelo de clases generado en el PIM, del diseño de los casos de uso y requisitos de diseño. Para cada operación se tiene que especificar su nombre, parámetros y visibilidad, también se puede hacer uso del polimorfismo y sobrecarga de operaciones para simplificar el modelo de clases.

- Meta:

  Descripción del Modelo de Clases con las Operaciones de cada clase.

Técnicas	Herramientas	Estándares
• **Diagrama de clases.** • **Diagrama de Transición de Estado.**	• **MagicDraw UML**	• **MOF** • **UML**

Tarea 4.5: Diseño de la Jerarquía

- Descripción:

  Esta actividad tiene como objetivo revisar la jerarquía de clases que ha surgido del modelo de clases a partir de las tareas anteriores. Si es necesario se definirán clases abstractas o superclases que agruparan atributos y operaciones para otras clases que heredaran esto.

- Meta:

  Descripción del Modelo de Clases con las jerarquías entre las clases.

Técnicas	Herramientas	Estándares
• **Diagrama de clases.**	• **MagicDraw UML**	• **MOF** • **UML**

Actividad 5 DISEÑO DE LA GESTIÓN DE DATOS.

Propósito de la actividad:

En esta actividad se define la estructura física de los datos que utilizara el sistema a partir del modelo de clases especificado anteriormente, se deben tener presentes las características específicas del sistema gestor de datos a utilizar, los requisitos del sistema de información y las particularidades del entorno tecnológico.

Entregables:

- Modelo de Diseño Físico de Datos.

Tarea 5.1: Diseño del Modelo de Datos.

- Descripción:

  Esta actividad tiene como objetivo realizar el diseño del modelo físico de los datos dependiendo del modelo de clases definido. También se revisará el acceso a los datos tratando de optimizar el rendimiento de la obtención de la información.

- Meta:

  Diseño del Modelo Físico de Datos.

Técnicas	Herramientas	Estándares
• **Modelo Físico de datos.** • **Transformación del Modelo de Clases a Modelo Físico de Datos.** • **Cálculos de Accesos Físicos y**	• **MagicDraw UML** • **Framework MDA: AndroMDA, OpenMDX**	• **MOF** • **UML** • **QVT**

Caminos de Accesos.		

Tarea 5.2: Especificación de la Distribución de Datos

- Descripción:

  Esta actividad tiene como objetivo determinar el modelo de distribución de datos teniendo en cuenta los requisitos de diseño. En esta tarea se establecerán la ubicación de gestores de base de datos, elementos de la estructura física de datos, todo de acuerdo al diseño de la arquitectura del sistema realizado.

- Meta:

  Actualización del Modelo Físico de Datos, identificando esquemas físicos de datos y asignaciones a los diferentes nodos.

Técnicas	Herramientas	Estándares
• **Definición de los esquemas físicos de datos.** • **Técnicas matriciales.**	• **MagicDraw UML** • **Framework MDA: AndroMDA, OpenMDX.**	• **MOF** • **UML**

Actividad 6 GENERACIÓN DE ESPECIFICACIONES DE CONSTRUCCIÓN

Propósito de la actividad:

En esta actividad se generan las especificaciones para la construcción del sistema de información, a partir del diseño detallado. Se tomarán en cuenta la división arquitectónica y de componentes, dependencias entre los módulos, especificaciones físicas de datos.

Entregables:

- Modelo de Construcción del Sistema (PSM), esto incluye los Diagramas de Componentes y Diagramas Estructura y Despliegue.
- Plan de Integración del Sistema.

Tarea 6.1: Especificación del Entorno de Construcción

- Descripción:

  Esta actividad tiene como objetivo definir de forma detallada el entorno que se utilizara para la construcción del sistema de información, en este caso se definirán los elementos que se necesiten de la arquitectura MDA.

- Meta:

  Especificar el Entorno de Construcción.

Técnicas	Herramientas	Estándares
• **Transformación del modelo PIM detallado al modelo PSM según el entorno definido.** • **Generación del Modelo XMI del Modelo UML.**	• **MagicDraw UML** • **Framework MDA: AndroMDA, OpenMDX.**	• **MOF** • **UML** • **XMI** • **QVT**

Tarea 6.2: Definición de Componentes y Subsistemas de Construcción

- Descripción:

  Esta actividad tiene como objetivo la especificación de cada subsistema de definición de componentes, así como también la integración de todos estos elementos. Otro aspecto que se deberá definir es la distribución de los componentes que integran el sistema.

- Meta:

  Especificar los Componentes y Subsistemas de Diseño del Sistema de Información.
  Diseño del Plan de Integración.

Técnicas	Herramientas	Estándares
• **Diagrama de Estructuras.** • **Diagrama de Componentes. (incluyendo restricciones y parámetros necesarios).** • **Diagrama de Despliegue.** • **Técnicas Matriciales.**	• **MagicDraw UML** • **Framework MDA: AndroMDA,OpenMDX.**	• **MOF** • **UML** • **XMI** • **OCL**

Actividad 7 DISEÑO DE LAS PRUEBAS DEL SISTEMA

Propósito de la actividad:

En esta actividad se realizara la especificación del plan de pruebas del sistema de información a partir de lo definido en la fase de análisis. Los siguientes tipos de prueba se definirán:

- Pruebas unitarias
- Pruebas de integración
- Pruebas del sistema
- Pruebas de implantación
- Pruebas de aceptación

Los documentos que se tomaran en cuenta son: catálogo de requisitos, catálogo de excepciones, diseño detallado del sistema.

Entregables:

- Documento de Diseño y Planificación de Pruebas.

Tarea 7.1: Definición de las Pruebas

- Descripción:

  Esta actividad tiene como objetivo la definición detallada del entorno necesario para ejecutar el plan de pruebas para el sistema.

- Meta:

  Especificación del Entorno de Pruebas.

Técnicas	Herramientas	Estándares
• **Definición de Procedimientos y Tipos de Pruebas.** • **Diseño de Pruebas de Caja Blanca y caja Negra** • **Definición del Entorno Tecnológico para Ejecución de Pruebas.**	• **Juegos de ensayo.** • **Análisis de resultados.** • **Procedimientos de Emergencia y Recuperación del Sistema.**	**No Aplican**

Tarea 7.2: Revisión de la Planificación de Pruebas

- Descripción:

  Esta actividad tiene como objetivo completar la especificación de la planificación de las pruebas, se determinaran la forma de evaluación de los resultados y el tiempo estimado para cada tipo de prueba, dependiendo de la estrategia de integración.

- Meta:

  Planificación del Plan de Pruebas.

Técnicas	Herramientas	Estándares
• **Formas de evaluación y planificación de tiempos de pruebas.** • **Medidas de amplitud de las pruebas, perfiles de fallos y profundidad de las pruebas** • **Control de documentación del plan de pruebas.**	**No Aplican**	**No aplican**

Actividad 8 ESTABLECIMIENTO DE REQUISITOS DE IMPLANTACIÓN

Propósito de la actividad:

En esta actividad se completara el catálogo de requisitos con los relacionados a la documentación que se requiere para operar el sistema y los requisitos para la propia implantación del sistema en el entorno de operación.

Entregables:

- Catálogo de Requisitos de Implantación.
- Manual de Usuario (actualización del catálogo de requisitos general).

Tarea 8.1: Especificación de Requisitos de Documentación de Usuario

- Descripción:

  Esta actividad tiene como objetivo recoger toda la información necesaria para especificar la documentación a entregar al usuario, incluyendo manuales de usuario.
- Meta:

  Requisitos de Documentación de Usuario.

Técnicas	Herramientas	Estándares
• **Catalogación.** • **Control de la Documentación de Usuario.**	**No Aplican.**	**No Aplican**

Tarea 8.2: Especificación de Requisitos de Implantación

- Descripción:

  Esta actividad tiene como objetivo especificar los requisitos de implantación con el fin de tener preparado todo lo necesario para la implantación del sistema de información. Se deberán tomar en cuenta los requisitos de infraestructura e instalación, equipamiento hardware, software y comunicaciones.
- Meta:

  Requisitos para Implantación del Sistema.

Técnicas	Herramientas	Estándares
• **Catalogación**	**No Aplican**	**No Aplican**

Actividad 9 APROBACIÓN DEL DISEÑO DEL SISTEMA DE INFORMACIÓN

Propósito de la actividad:

En esta actividad se realizara una revisión de la documentación y modelos diseñados para presentar el informe final del diseño completo del sistema de información.

Entregables:

- Informe y Presentación del diseño del sistema de información. (incluyendo fuentes de modelos diseñados y documentos trabajados).

Tarea 9.1: Presentación y Aprobación del Diseño del Sistema de Información

- Descripción:

  Esta actividad tiene como objetivo realizar la presentación del diseño del sistema de información para su aprobación.
- Meta:

  Presentar y Entregar el Diseño del Sistema.

Técnicas	Herramientas	Estándares
• **Presentación multimedia e Informes**	• **Power Point.** • **Adobe presenter.**	**No Aplican**

Actividades

Actividades de Diseño del Sistema.

ADS1: Definición de la Arquitectura del Sistema.
ADS2: Diseño de la Arquitectura de Componentes.
ADS3: Diseño de los Casos de Usos.
ADS4: Diseño de Objetos.
ADS5: Diseño de las Gestión de Datos.
ADS6: Generación de Especificaciones de Construcción.
ADS7: Diseño de las Pruebas del Sistema.
ADS8: Establecimiento de Requerimientos de Implantación.
ADS9: Aprobación del Diseño del Sistema.

Nota:
En este diagrama se hace principal referencia a la aplicación de los estándares MDA en cada actividad en la etapas de diseño del sistema.

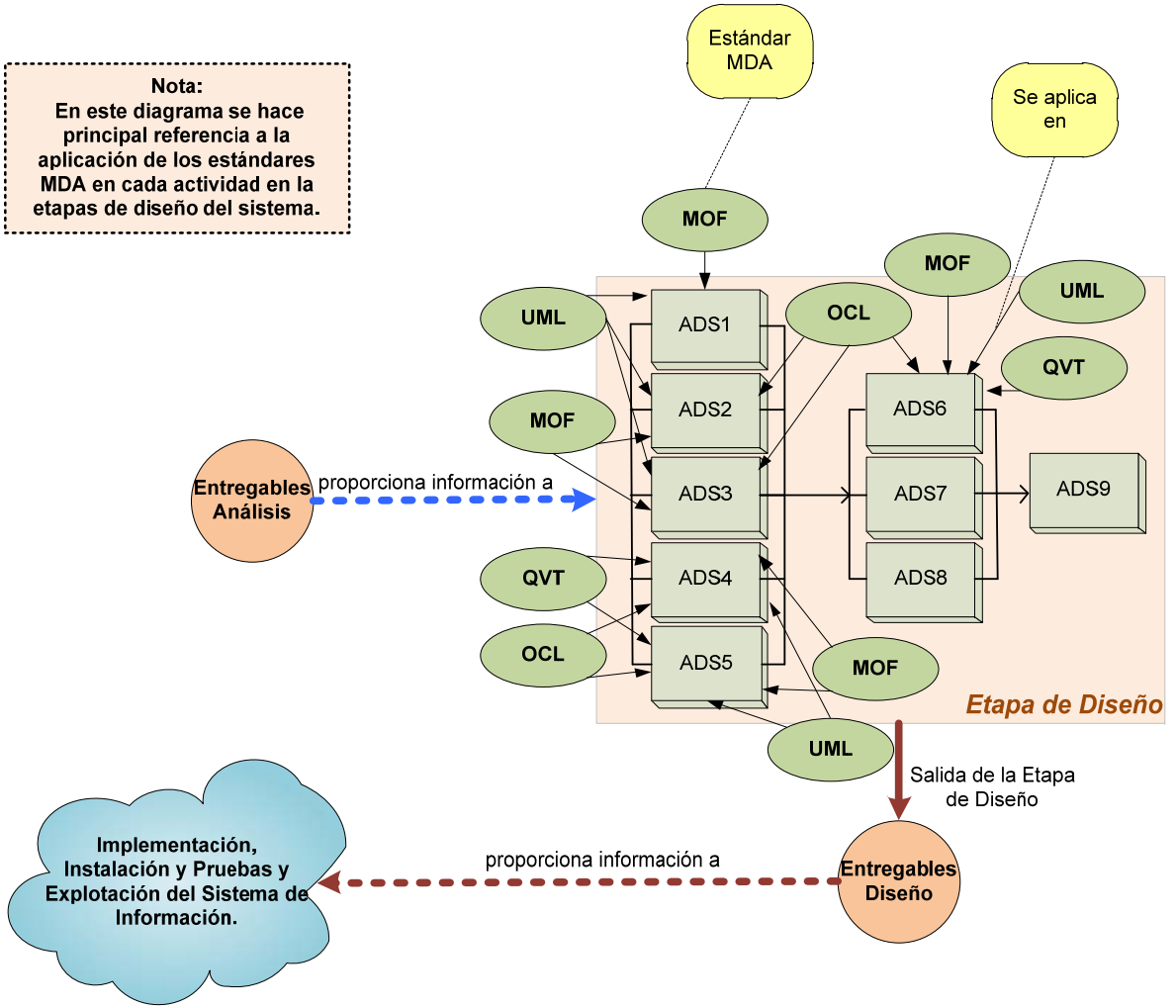


Figura 15: Diagrama General de la Metodología – Fase de Diseño

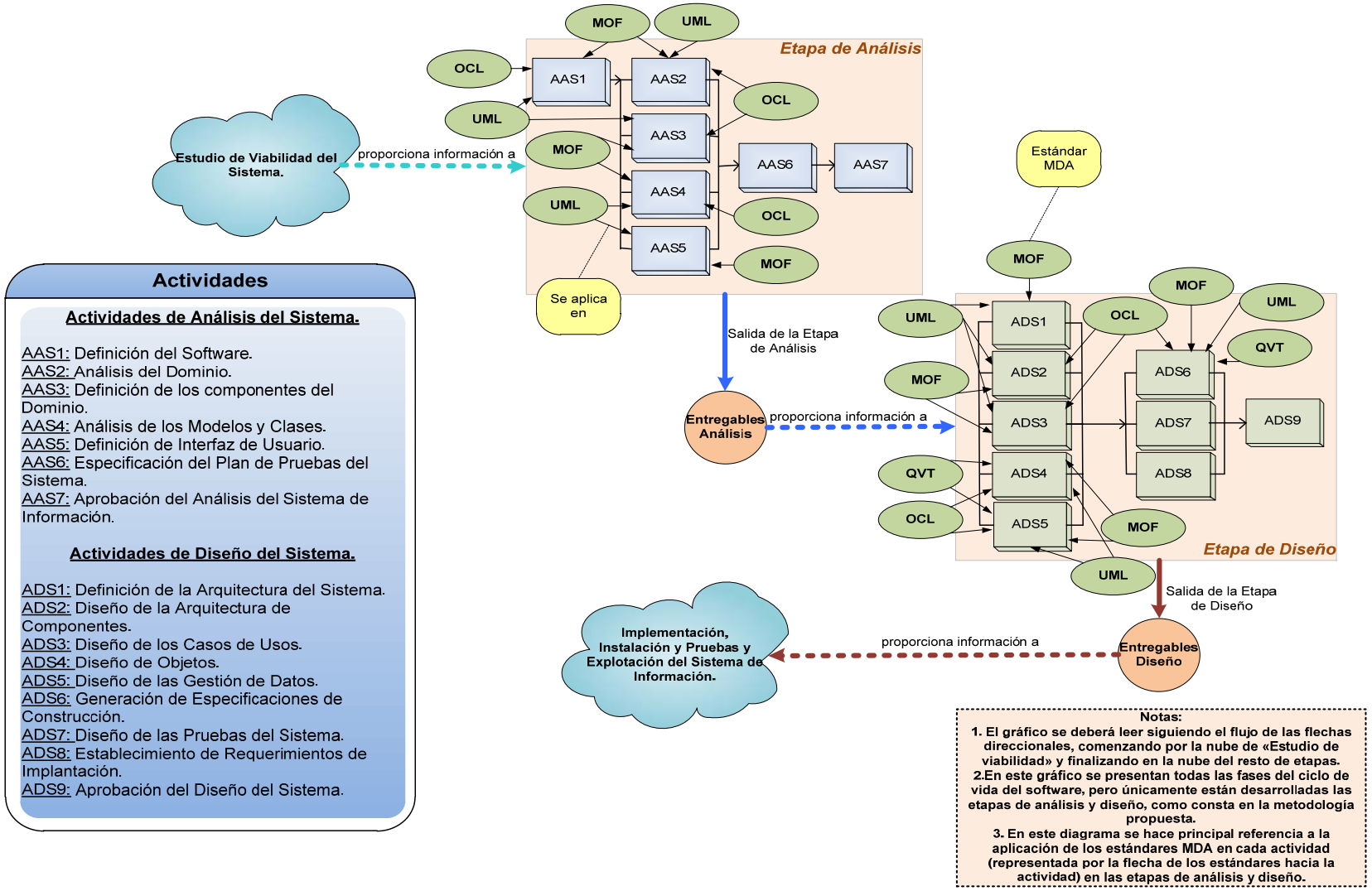


Figura 16: Diagrama General de la Metodología para la Selección de Estándares

4. Análisis y Presentación de los Resultados.

En el presente estudio se realizó un análisis amplio sobre el ciclo de vida de un proyecto de desarrollo de software y de las distintas metodologías para desarrollo orientado a objetos en las cuales se reflejan los pasos necesarios que se pueden aplicar cuando se hace uso de los estándares MDA, tales como: MOF, UML, QVT, OCL, XMI, los cuales fueron evaluados y seleccionados para su aplicación en la metodología propuesta en esta monografía. La selección fue fundamentada de acuerdo al estudio de los conceptos y principios definidos para MDA y se delimitó el resultado al uso de dichos estándares en el Análisis y Diseño.

Se realizaron diversas pruebas utilizando las herramientas MDA de software libre que llevan por nombre AndroMDA y OpenMDX. Mediante estas pruebas se pudo comprobar la aplicación de los estándares MDA: MOF, UML y XMI. Estas pruebas se realizaron con el propósito de conocer el funcionamiento de la arquitectura MDA, sus estándares y la aplicación de la teoría correspondiente.

Como resultado principal de esta monografía se desarrolló una propuesta metodológica orientada a objetos bajo el enfoque MDA, para la selección de los estándares mencionados, que se podrían aplicar durante las etapas de análisis y diseño de un proyecto de desarrollo de software. Para el desarrollo de la propuesta metodológica se utilizaron como referencias básicas algunas metodologías muy bien documentadas y aceptadas internacionalmente, tales como: METRICA V3 y Metodología orientada a objetos por Roger S. Pressman.

La metodología fue desarrollada solamente para dos de las etapas principales del ciclo de vida de un proyecto de desarrollo de software, la etapa de análisis y la etapa de diseño. La etapa de análisis consta de 7 actividades y 23 tareas; la etapa de diseño consta de 9 actividades y 24 tareas. Las actividades y tareas descritas para cada una de las etapas representan lo que requiere el desarrollador para un desarrollo de software utilizando el enfoque MDA. Para cada una de las actividades se redactó una descripción y se definieron los respectivos entregables. En el caso de las tareas se redactó una descripción, se definió una meta y se identificaron estándares que se pueden aplicar a algunas de ellas, aunque también se determinó que algunas tareas no pueden ser soportadas por los estándares seleccionados, lo que quedó reflejado en la metodología.

5. Conclusiones y Recomendaciones

Conclusiones

En el presente trabajo se consideran cumplidos los objetivos generales y específicos de la monografía, así como también se considera comprobada la hipótesis formulada en el capítulo “Diseño Metodológico”.

Durante la realización del trabajo monográfico, se efectuó una evaluación amplia del enfoque MDA, la fundamentación para la selección de los estándares a ser utilizados y la aplicación de herramientas MDA, lo cual se puede ver reflejado tanto en el capítulo del “Marco Teórico” como en el capítulo de “Fundamentación para la selección de estándares usados en la metodología.”

El estudio realizado brinda una descripción general del enfoque MDA, los estándares seleccionados, su aplicación y ubicación dentro de la arquitectura usada por el enfoque MDA.

La característica fundamental de los modelos PIM y PSM es que pueden ser procesados por una computadora mediante un conjunto de transformaciones con reglas bien definidas, para lo cual son de gran utilidad las herramientas que dan soporte al enfoque MDA. Por lo tanto, las diversas trasformaciones que se deben realizar durante las etapas de desarrollo de software, se pueden hacer de forma automática aplicando el enfoque MDA. De esta manera se resuelven de forma óptima los problemas de productividad y calidad, durante las etapas de análisis y diseño del desarrollo de software.

Mediante el presente trabajo se analizaron los principales conceptos sobre el enfoque MDA y los estándares seleccionados. Sobre esta base se desarrolló una propuesta de metodología que comprende las etapas de Análisis y Diseño en un proyecto de desarrollo de software, las cuales son de gran relevancia cuando se utiliza el enfoque MDA.

La metodología desarrollada propone un conjunto de actividades y tareas que permiten la selección y aplicación de los estándares basados en el enfoque MDA, con el propósito de mejorar el proceso de desarrollo de software en las etapas de análisis y diseño, generando la documentación que las soporta.

Como parte de las conclusiones del trabajo monográfico se ha considerado relevante mencionar que tanto la investigación teórica realizada como la propuesta

metodológica desarrollada constituyen una guía útil para los desarrolladores de software interesados en la implementación del enfoque MDA.

Recomendaciones

A partir de los resultados del presente trabajo, se recomienda promover investigaciones para la complementación de las etapas desarrolladas en la metodología propuesta.

Se recomienda profundizar en un análisis detallado de los conceptos, estándares y herramientas que engloban el desarrollo de software basado en el enfoque MDA, para lograr resultados efectivos sobre el uso de la metodología.

7. Referencias

[1] EYSSAUTIER de la MORA, Maurice. *Metodología de la Investigación: desarrollo de la inteligencia*. Quinta edición. México: Cengage Learning Editores, 2006. pp. 97. ISBN 970-686-384-2

[2] *Selecting a development approach [PDF].* USA: Department of Health and Human Services, 2005. Disponible en web: <http://www.cms.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf>.

[3] OMG. MDA Guide Version 1.0.1. Ed. Miller Joaquin, MukerjiJishnu, Junio 2003. Disponible en web: http://doc.omg.org/omg/03-06-01

[4]OMG.Object Management Group. [enlínea] <http://www.omg.org.ni> [consulta: 15 Marzo 2011]

[5] COOK, Steve; KENT, Stuart. *"The Domain-Specific IDE". The European Jounal for the Informatics Proffesional* [en linea]. Vol. IX. Abril 2008 [ref. 21 de marzo 2011], pp. 17. Disponible en internet: <http://www.lcc.uma.es/~av/Publicaciones/08/upgrade-vol-IX-2.pdf >.

[6] ORACLE. Java Metadata Interface (JMI). [enlinea] <http://java.sun.com/products/jmi/> [consulta: 01 Abril 2011]

[7] LAUDON, Kenneth; LAUDON, Jane. *Sistemas de Información Gerencial: Administración de la Empresa,* digital.Décima edición. México: Pearson Educación, 2008. 645 p.

[8] *IEEE Standard Glossary of Software Engineering Terminology* [PDF]*.*Std 610.12-1990(R2002). Nueva York: IEEE Standards Board, 1990. ISBN 1-55937-067-X

[9] W3C. Extensible MarkupLanguage (XML) 1.0.Quinta Edición. Noviembre 2008. Disponible en web: <http://www.w3.org/TR/REC-xml/#sec-prolog-dtd>. [consulta: 01 abril 2011]

[10] BRAN Selic, *"Model-Driven Development: Its Essence and Opportunities*", Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2006. Disponible en web http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1630449. [Consulta: 05 abril 2010]

[11] PRESSMAN, Roger S., *SOFTWARE ENGINEERING. A Practitioner's Approach.*5th Edition.Editorial The McGraw-Hill Companies.2001

[12] FERNÁNDEZ SAÉZ Pedro Antonio*. Un Análisis Crítico de la Aproximación Model-DrivenArchitecture,* Universidad Complutense de Madrid (Madrid, España) *año académico 2008/2009.* Disponible en web: http://eprints.ucm.es/9880/1/Un_An%C3%A1lisis_Cr%C3%ADtico_Sobre_la_Aproximaci%C3%B3n_Model-Driven_Architecture.pdf [Consulta: 05 Enero 2011]

[13] OMG Available Specification. Unified Modeling Language: Infrastructure Version 2.0.[et al.], Mayo 2005. Disponible en web: http://doc.omg.org/omg/05-07-05.

[14] OMG Available Specification. Unified Modeling Language: Superstructure Version 2.0.[et al.], Mayo 2004. Disponible en web: http://www.omg.org/spec/XMI/20071001.xml

[15] OMG Available Specification. MOF 2.0 /XMI Mapping, Version 2.1.1. [et al.], Diciembre 2001. Disponible en web: http://www.omg.org/spec/XMI/20071001.xml

[16] OMG Available Specification. Meta Object Facility (MOF) Core Specification, Version 2.0. [et al.], Junio 2001. Disponible en web: http://doc.omg.org/omg/06-01-01

[17] BOLLATI Verónica A., SÁNCHEZ E. Víctor, VELA Belén, Esperanza Marcos, "Análisis de QVT OperationalMapping: un caso de estudio"[ref. 23 de noviembre 2010].

[18] OMG Available Specification. Meta Object Facility (MOF) Query/View/Transformation Specification, Version 2.0. [et al.], Julio 2007. Disponible en web: http://doc.omg.org/omg/07-07-07

[19] OMG Available Specification. Object Constraint Language, Version 2.0. [et al.], Julio 2007. Disponible en web: http://doc.omg.org/omg/06-05-01

[20] RODRÍGUEZ Alfonso, *"Transformaciones QVT para la obtención de Clases de Análisis a partir de un Modelo de Proceso de Negocio Seguro"* Universidad de BioBio Chillán, Chile. Departamento de auditoría e informática.

[21] BOOCH, G., BROWN A., and RUMBAUGH J.. *An MDA Manifesto*, IBM Rational Software. *2004*

[22] KONTIO, M. Architectural Manifiesto: MDA in Action. Octubre 2005.Disponible en web: http://www-128.ibm.com/developerworks/library/wiarch 19/index.html.

[23] KOZIKOWSKI, J. *"A bird's eye view of AndroMDA".*Disponible en web: http://galaxy.andromda.org/docs/contrib/birds-eye-view.html. [Consulta: 03 de Dicembre 2010].

[24] An Interview with Matthias Bohlen. http://www.codegeneration.net. [Consulta: 15 de Diciembre 2010].

[25] OpenMDX Sitio Oficial. Disponible en web: http://www.openmdx.org/#HOME [Consulta: 04 de Enero 2011]

[26] OpenMDX Sitio Oficial. Introduction to openMDX.Disponible en web: http://sourceforge.net/apps/trac/openmdx/wiki/Introduction [Consulta: 14 de Febrero 2011]

[27] OpenMDX Documentation. Disponible en web: http://sourceforge.net/apps/trac/openmdx/wiki [Consulta: 13 de Noviembre 2010]

[28] Ministerio de Administraciones Públicas Español (MAP). http://www.csi.map.es/[Consulta: 24 de Septiembre de 2011].

[30] Workshop on Model Driven Development (WMDD 2004) - June 15, 2004 - at ECOOP 2004, Oslo, Norway (June 14–18, 2004) - *http://heim.ifi.uio.no/~janoa/wmdd2004*

Anexos

Anexo A: Guía de Instalación AndroMDA.

Environment Setup

Here we will download and install the components required for development with AndroMDA. This tutorial assumes that you have:

- **Microsoft Visual Studio 2005** installed. Note that the Express edition will **not** work as it does not have add-in support.
- **Microsoft SQL Server** installed. Any version will work (2000, 2005, MSDE, etc.)

As you follow the software install instructions select the default options in the installers unless otherwise instructed.

Install the Java Runtime

AndroMDA is written in Java and needs the Java runtime to execute. **Don't get nervous**, you won't have to worry about it after it is installed or write any Java code.

1. Download either the **JRE 6.0** from http://java.com/de/download/index.jsp or the **J2SE Development Kit 6.0** fromhttp://www.oracle.com/technetwork/java/javase/downloads/index.html
2. Run the installer

Install Maven

Maven is a build tool similar to ant and nant. It is used to run AndroMDA.

1. Download the Maven binary zip fromhttp://maven.apache.org/download.html. At the time of this writing this is **apache-maven-2.2.1-bin.zip**.
2. Create a folder for maven (example: C:\Program Files\Apache Software Foundation\Maven 2.2.1
3. Unpack the maven zip into this folder

Configure the Environment

1. Open the windows **Control Panel** then the **System** control panel. Click the **Advanced** tab and click the **Environment Variables** button. Ensure the following environment variables are set:

Variable	Value	Example
M2_HOME	Maven installation directory	C:\Program Files\Apache Software Foundation\Maven 2.2.1

JAVA_HOME	Java installation directory	C:\Program Files\Java\jdk1.6.0_xx
PATH	The system search path. You need to add the M2_HOME\bin and JAVA_HOME\bin to the path.	%JAVA_HOME%\bin;%M2_HOME%\bin

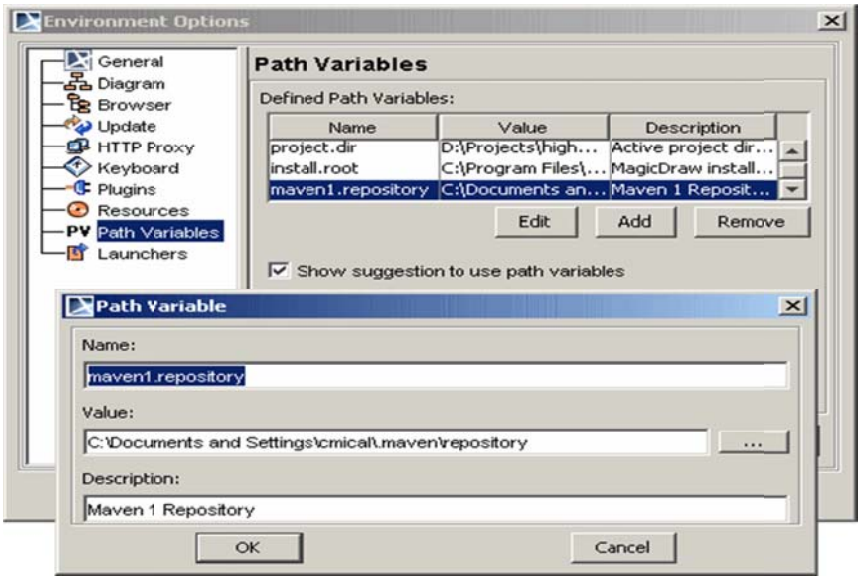
2. It is recommended you restart after making these changes

Install MagicDraw 9.5/16.5 Community Edition

AndroMDA reads UML 1.4 models that have been saved in XMI 1.2 format and UML 2.0 models saved in uml2.2 format. MagicDraw 9.5 is a UML 1.4 modeling tool that saves XMI model files. MagicDraw 11.x and newer is a UML 2 modeling tool that saves uml model files.

1. Download the MagicDraw 16.8 (or newer) Community Edition installer from http://www.magicdraw.com/.
2. Run the installer. Make sure to select the install application specific JRE option.
3. Run Magic Draw
4. Click the Options menu and select Environment. Select the Path Variables option in the tree on the left.
5. Click the Add button and enter the values below. Be sure to replace **your user name** with your system username.

Field Name	**Value**
Name	maven2.repository
Value	C:\Documents and Settings**your user name**\.m2\repository
Description	Maven 2 Repository



6. *(optional)* In the main MagicDraw window select **Options** | **Look and Feel** | **Windows** to enable a more windows-like interface.

Install Android/VS

Android/VS is a Visual Studio 2005 add-in that makes working with AndroMDA and Visual Studio much easier.

1. Download the latest version of the Android/VS installer from http://sourceforge.net/projects/andromdaplugins/files/Android_VS/
2. Run the installer

Create Time Tracker Application

Now that you have your environment setup you can start using AndroMDA in your projects. This section will guide you through creating the base TimeTracker solution. These are the same steps you would follow when you are starting your own application. Please note the TimeTracker completed sample is available here:Northwind.TimeTracker.zip.

Couple of tips before we start:

- Please follow the instructions in this tutorial very carefully and precisely. **Do not** take any shortcuts, as doing so will only waste your time. There is a reason for every step in the tutorial. You will not find any fluff here!

Create the Solution

1. Run Visual Studio 2005
2. Select **File** | **New Project**
3. Create a new blank solution, found under **Other Project Types | Visual Studio Solutions**. Name the solution **Northwind.TimeTracker**.
4. Click **Ok**

Run the Solution Wizard

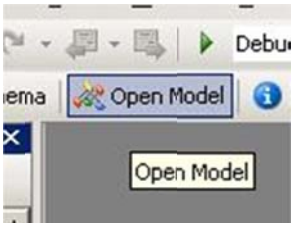
1. Click the **Run MDA Solution Wizard** 

and click **Next** to exit the welcome page.
2. Accept the default project name of **Northwind.TimeTracker** click **Next**.
3. Accept the default common project name of **Northwind.TimeTracker.Common** and click **Next**.
4. Accept the default core project name of **Northwind.TimeTracker.Core** and click **Next**.
5. Accept the default checked **Create schema export project** option and the default project name of **Northwind.TimeTracker.SchemaExport** and click **Next**.
6. Check the **Configure web project** box and accept the default project name of **Northwind.TimeTracker.Web**. Check the **Add membership support** box and click **Next**.

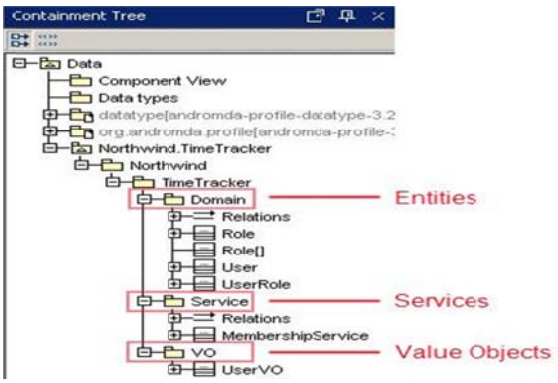
7. Accept the default web common project name of **Northwind.TimeTracker.Web.Common** and click **Next**.
8. Verify that your project settings look like the screenshot below, and click **Next** to run the wizard.
9. Your solution is now configured for use with AndroMDA. Here is a list of what was created for you:
    - **mda** directory: AndroMDA configuration files and an empty model. Because we checked membership support basic user and roles entities were added automatically.
    - **Northwind.TimeTracker.Web**: Web site with nhibernate support
    - **Northwind.TimeTracker.Common**: Common project. This is where AndroMDA generates value objecst and enumerations
    - **Northwind.TimeTracker.Core**: Core project. This is where AndroMDA generates entities, nhibernate mappings, services, and data access objects.
    - **Northwind.TimeTracker.SchemaExport**: A console application that will generate database schema from your model. It can optionally export the schema to a running database.
    - **Northwind.TimeTracker.Web.Common**: Project for classes and controls the web site will use. Because we checked membership support an ASP.NET 2.0 Membership Provider that uses the user and role entities was created for us, and the web site is pre-configured to use it.

Open the UML Model

1. Click the **Open Model** button to open your project's UML model.



2. MagicDraw will open and it will automatically load the model that was generated by the add-in. An initial package structure was created by the add-in using the name of the solution you selected. Additionally a **User** entity and some supporting classes were created and added to the model because the membership support option was selected.



Leave MagicDraw open for now, as we will be adding to the model.

Creating the TimeTracker Database

We need to create a database for TimeTracker to store the timecards. While you can use any type of database NHibernate supports in your project, this tutorial will focus on SQL Server 2000, 2005 and MSDE. Here are the steps to creating a database for TimeTracker:

Creating the Database with SQL Server 2000/2005

1. Run Enterprise Manager (2000) or SQL Server management Console (2005)
2. Expand your local server (2000) or connect to your local server (2005)
3. Right click the Databases folder and select **Create new database**
4. Enter the database name **NorthwindTimeTracker** and click **Ok**.

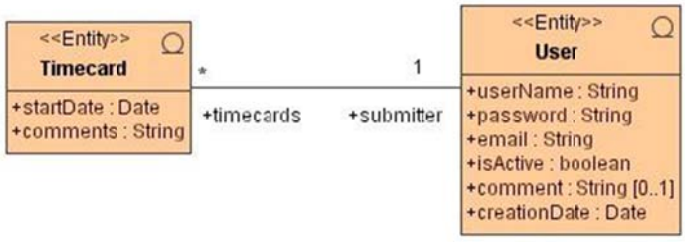
Creating the Database with MSDE

1. Open a console window by clicking **Start** |**Run**, typing **cmd**, and clicking **Ok**.
2. Type **osql -U sa -P password -S localhost\myinstance** with your MSDE login credentials and **myinstance** changed to your instance name.
3. Type the following to create a database called **NorthwindTimeTracker**:

```
1>use master
2:> go
1> CREATE DATABASE NorthwindTimeTracker
2> go
The CREATE DATABASE process is allocating 0.63 MB on disk
'NorthwindTimeTracker'.
The CREATE DATABASE process is allocating 0.49 MB on disk
'NorthwindTimeTracker_log'.
```

Entities

In this section you will create your first entity using AndroMDA. We will create an entity called **Timecard** that will represent a timecard in the application. We will associate it with the **User** object, which will be the person that submitted the timecard. Our finished entity diagram will look like this:

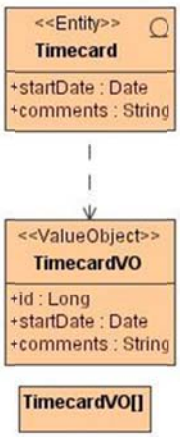


Now let's add the **Timecard** entity to the model:

1. Create a new diagram by right clicking on the **Domain** package and clicking **New Diagram |Class Diagram**. Set the name of the diagram to **Entity Diagram** and click **Ok**.
2. Click the **Class** tool  in the toolbar and make a new class on the diagram surface. Double click on the class, name it **Timecard**, and add the **Entity** stereotype to it.
3. Right click on the **TimecardVO** object and select **Insert new attribute**. Add three public attributes to the class by:
   - One called **startDate** of type**Date**.
   - One called **comments** of type**String**.
4. We now want to relate the User entity to the Timecard entity. Drag the **User** class onto the diagram. If it looks very cluttered you can right click on it and select **Symbol Properties**. Uncheck the **Show Tagged Values** option under **General** and also uncheck **Show Attributes Properties** in the **Attributes** section. Finally click **Ok**, right click on the **User** class again, and select **Autosize**.
5. Because each user can submit many timecards, we need to create an association that defines this relationship. Click the **Association** tool  and draw an association between the **Timecard** class and the **User** class.
6. Set the association multiplicity to ***** on the timecard end of the association by right clicking on the association end near the **Timecard** entity and selecting.
7. Set the name of the timecards association endpoint by right clicking on the association end near the **Timecard** entity and select **Edit name**. Name the end **+timecards**.
8. Set the association multiplicity to **1** on the user end of the association by right clicking on the association end near the **User** entity and selecting**1**.
9. Set the name of the submitter association endpoint by right clicking on the association end near the **User** entity and select **Edit name**. Name the end **+submitter**.

Value Objects

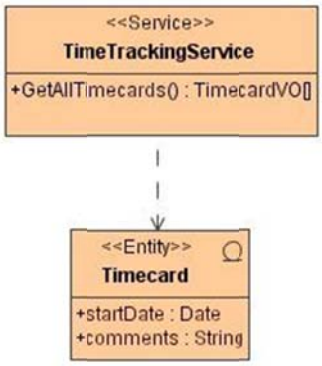
We will now add a value object called **TimecardVO** to the model. This will be the value object for the **Timecard** entity and we will keep it simple by mapping it one-to-one with the **Timecard** entity. The model for the **TimecardVO** looks like this:



1. Create a new diagram by right clicking on the **VO** package and clicking **New Diagram** |**Class Diagram**. Set the name of the diagram to **Value Object Diagram** and click **Ok**.
2. Click the **Class** tool  in the toolbar and make a new class on the surface. Double click on the class, name it **TimecardVO**, and add the **Value Object** stereotype to it.
3. Right click on the **TimecardVO** object and select **Insert new attribute**. Add three public attributes to the class by:
   - One called **id** of type **Long**.
   - One called **start Date** of type **Date**.
   - One called **comments** of type **String**.
4. Drag the **Timecard** class onto the diagram. Click the **Dependency** tool  and draw a dependency from the **Timecard** class to the **TimecardVO** class. Adding this dependency will tell AndroMDA that the **Timecard** entity will be converted to **TimecardVO** objects, and it will generate helper code in the generated **TimecardDao** data access class to support translation between them.
5. Click the **Class** tool  in the toolbar and make a new class on the surface. Double click on the class and name it**TimecardVO[].** We do not need to add a stereotype to it. This will create a type that we can use to return collections of **TimecardVO** objects from service methods.

Services

We now have the entities and value objects modeled and we need to add services that will operate on these objects. Here we will create the **TimeTrackingService**. For this tutorial we will create one method on the service called **GetAllTimecards** method which will return all the timecards in the system. The model for the **TimeTrackingService** looks like this:



1. Create a new diagram by right clicking on the **Service** package and clicking **New Diagram** |**Class Diagram**. Set the name of the diagram to **Service Diagram** and click **Ok**.
2. Click the **Class** tool  in the toolbar and make a new class on the surface. Double click on the class, name it**TimeTrackingService**, and add the **Service** stereotype to it.
3. Right click on the **TimeTrackingService** class and select **Insert New Operation**. Add a method called **GetAllTimecards()** that returns type**TimecardVO[]**.

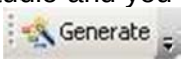
Note: Sometimes if you type **TimecardVO[]** on the diagram surface, MagicDraw will set the type to **TimecardVO** with a return type modifier of**[]**. This is incorrect, as you want the return type to be **TimecardVO[]** with no return type modifier set. This can be changed by double clicking on the method name and using the **Operation Specification** dialog.

4. Drag the **Timecard** class onto the diagram. Click the **Dependency** tool  and draw a dependency from the **TimeTrackingService** class to the **Timecard** class. Adding this dependency tells AndroMDA to make it easy to access the timecard data access class from methods inside in the **TimeTrackingService** class.
5. Save your model by clicking the **Save**  tool.

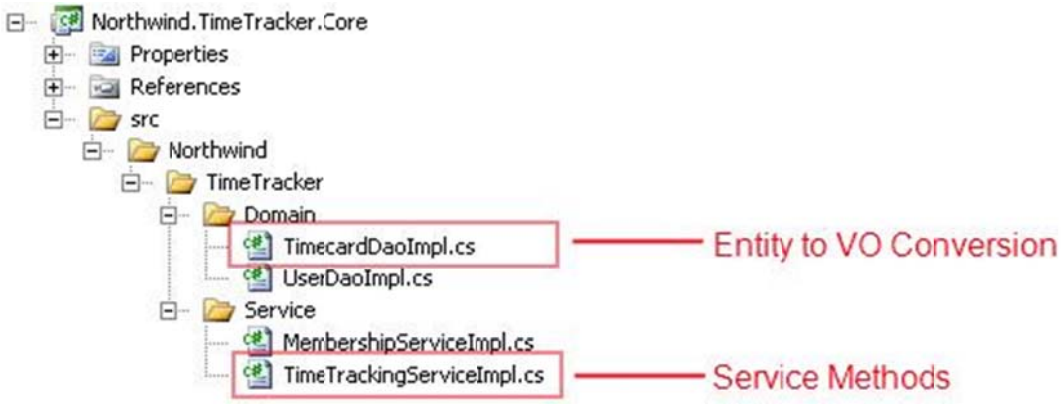
Display Timecards

Now that we have our model created and saved, we can write some code to see some results. First we will generate code with AndroMDA, and then we will implement the **GetAllTimecards** service method. After that we will export the schema to our TimeTracker database and finally we will create a screen that lists all the timecards in the database. Please note the TimeTracker completed sample is available here: Northwind.TimeTracker.zip.

Generate Code

First we will generate code from our model. Switch to Visual Studio and you can either select the **Build** | **Rebuild All** option or you can click the **Generate** 

button. This will run AndroMDA inside a Visual Studio tool window and generate the entity classes, hibernate mappings, data access objects, and service interfaces and base classes.



When this operation is complete you will see a number of files generated. Files that you don't need to edit and are always generated are placed in the **target** directory. Files that require you to fill in some code will be placed in the **src** directory if they don't exist. If they do exist they will not be overwritten so your changes will be preserved.

Writing the Entity to VO Conversion Code

The first thing we need to do is write some code that will convert between **Timecard** entities and **TimecardVO** value objects.

1. Open the **TimecardDaoImpl.cs** file.
2. Note the generated file already has created the method stubs ready for us to implement. Fill these methods in. Your final code should look like this:

```
usingNorthwind.TimeTracker.VO;

publicclassTimecardDaoImpl:TimecardDaoBase
{
publicoverrideTimecardVOToTimecardVO(Timecard entity)
{
// Entity to VO conversion
TimecardVOvalueObject=newTimecardVO();

valueObject.StartDate=entity.StartDate;
valueObject.Comments=entity.Comments;
valueObject.Id=entity.Id;

returnvalueObject;
}

publicoverrideTimecardTimecardVOToEntity(TimecardVOtimecardVO)
{
// VO to entity conversion
Timecard entity =Timecard.Factory.newInstance();

entity.StartDate=timecardVO.StartDate;
entity.Comments=timecardVO.Comments;

return entity;
}
}
```

Implement the Service Method

Now we can implement the method in our**TimeTrackingService**.

1. Open the **TimeTrackingServiceImpl.cs** file.
2. Note the generated file already has created the method stubs ready for us to implement. In addition, the service base class already has a method called **GetAllTimecards()**. When called, it will start a NHibernate transaction and then call our method:**HandleGetAllTimecards()**. Here is an example implementation of the **GetAllTimecards** method:

```
usingSystem.Collections;
usingNorthwind.TimeTracker.VO;
```

```
publicclassTimeTrackingServiceImpl:TimeTrackingServiceBase
{
protectedoverrideTimecardVO[]HandleGetAllTimecards()
{
IListtimecards=this.TimecardDao.LoadAll();
IListtimecardVOs=this.TimecardDao.ToTimecardVOList(timecards);
TimecardVO[]voarray=newTimecardVO[timecardVOs.Count];
timecardVOs.CopyTo(voarray,0);
returnvoarray;
}
}
```

Generate Database Schema

Now we will generate the database schema to store the timecards.

1. Click the **Export Schema** 

button. This will run the schema export console project in Visual Studio. The AndroMDA Visual Studio add-in defaults are set to pass the export option so it will write the schema to your database.
2. After the schema is created, we need to create some test data. Run the following SQL in any tool (query analyzer, management studio, Toad, etc.) You can also run the **database/testdata.sql** file that is in the completed downloadable solution.

```
        DELETE FROM TIMECARD
        DELETE FROM AppUser
        SET IDENTITY_INSERT AppUser ON
        INSERT INTO
AppUser(ID,UserName,Password,Email,IsActive,Comment,CreationDate)
        VALUES (1,'bob','n/a','bob@bob.net',1,'',getdate())
        SET IDENTITY_INSERT AppUser OFF
        SET IDENTITY_INSERT TIMECARD ON
        INSERT INTO TIMECARD (ID, START_DATE, COMMENTS,
SUBMITTER_FK)
        VALUES (1,getdate(),'This is the first timecard',1)
        INSERT INTO TIMECARD (ID, START_DATE, COMMENTS,
SUBMITTER_FK)
        VALUES (2,getdate(),'This is another timecard',1)
        SET IDENTITY_INSERT TIMECARD OFF
        GO
```

Create a Test Page

Now that we have our service code, database schema, and test data we can create a web page that lists all the timecards.

1. Right click on the web project **Northwind.TimeTracker.Web** and select **Add new item**
2. Select **Web Form** and accept the default name of **Default.aspx** by clicking **Add**.
3. Add a grid to the page by adding the following code to**Default.aspx**:

```
<asp:GridViewID="GridView1"runat="server"AutoGenerateColumns="false">
<Columns>
<asp:BoundFieldDataField="ID"HeaderText="ID"/>
<asp:BoundFieldDataField="StartDate"HeaderText="StartDate"/>
<asp:BoundFieldDataField="Comments"HeaderText="Comments"/>
</Columns>
</asp:GridView>
```

4. Add the code to get the timecards and bind to the grid by adding the following code to **Default.aspx.cs**:

```
usingNorthwind.TimeTracker.Service;
usingNorthwind.TimeTracker.VO;

public partial class _Default :System.Web.UI.Page
{
protectedvoidPage_Load(objectsender,EventArgs e)
{
if(!IsPostBack)
{
ITimeTrackingService service =newTimeTrackingServiceImpl();
TimecardVO[] timecards =service.GetAllTimecards();
                GridView1.DataSource= timecards;
                GridView1.DataBind();
}
}
}
```

5. Right click on the web project and select **View in browser**

You should see a list of 2 timecards. Congratulations, you have created your first application using AndroMDA!

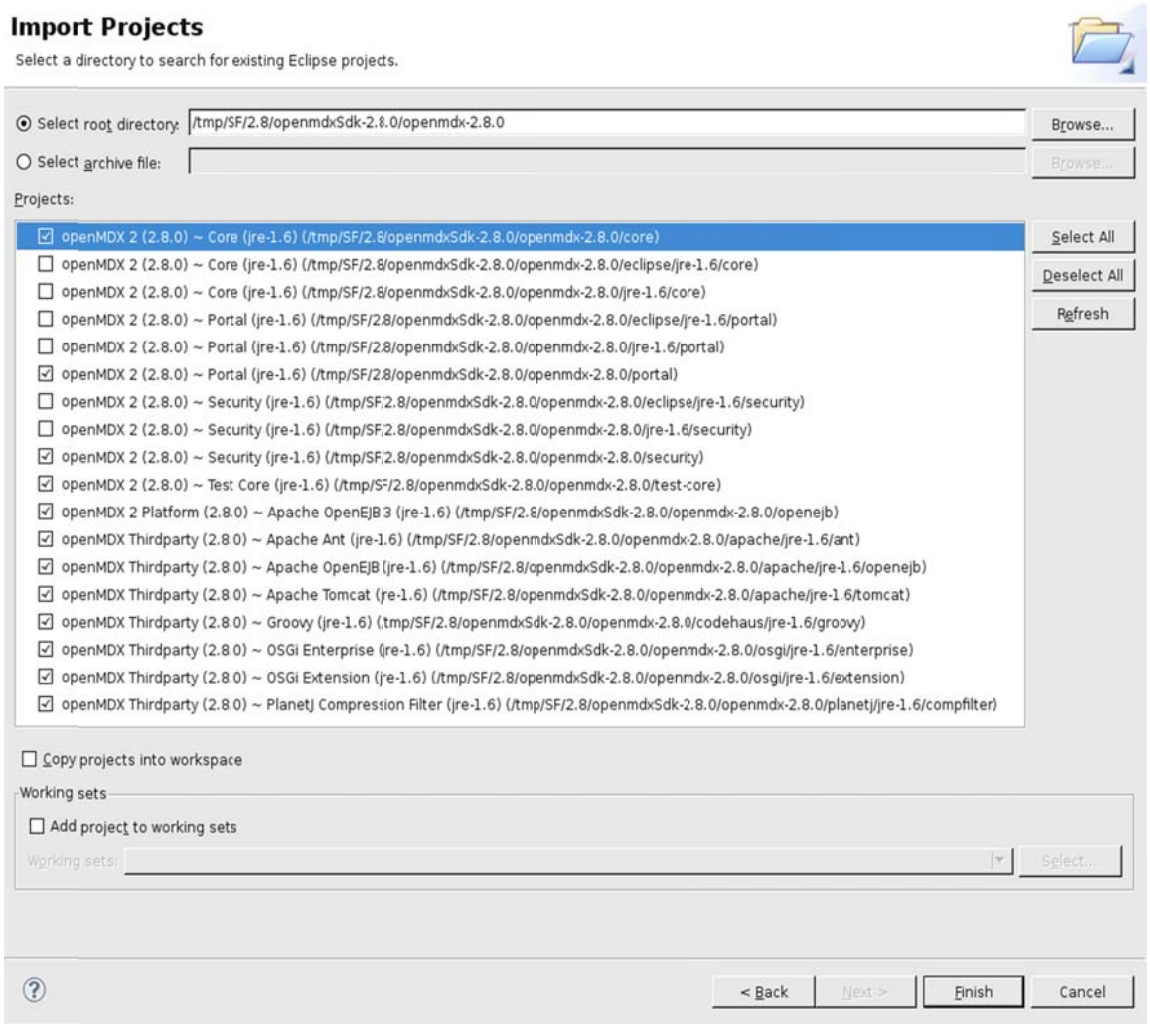
Anexo B: Guía de Instalación OpenMDX.

OpenMDX SDK 2.8 for Eclipse Step-by-Step Guide

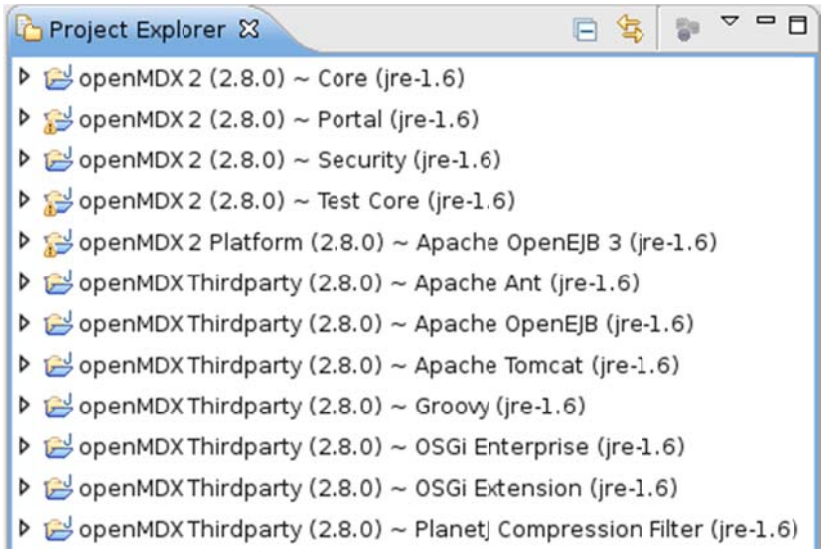
This guide explains how to setup *openMDXSDK2.8* for *Eclipse*.

IMPORTANT: This guide assumes that the *openMDXSDK2.8* projects are successfully setup as described in theopenMDXSDK2.8forAntStep-by-Step and openMDXSDK2.8onTomcat+OpenEJBStep-by-Step guides.

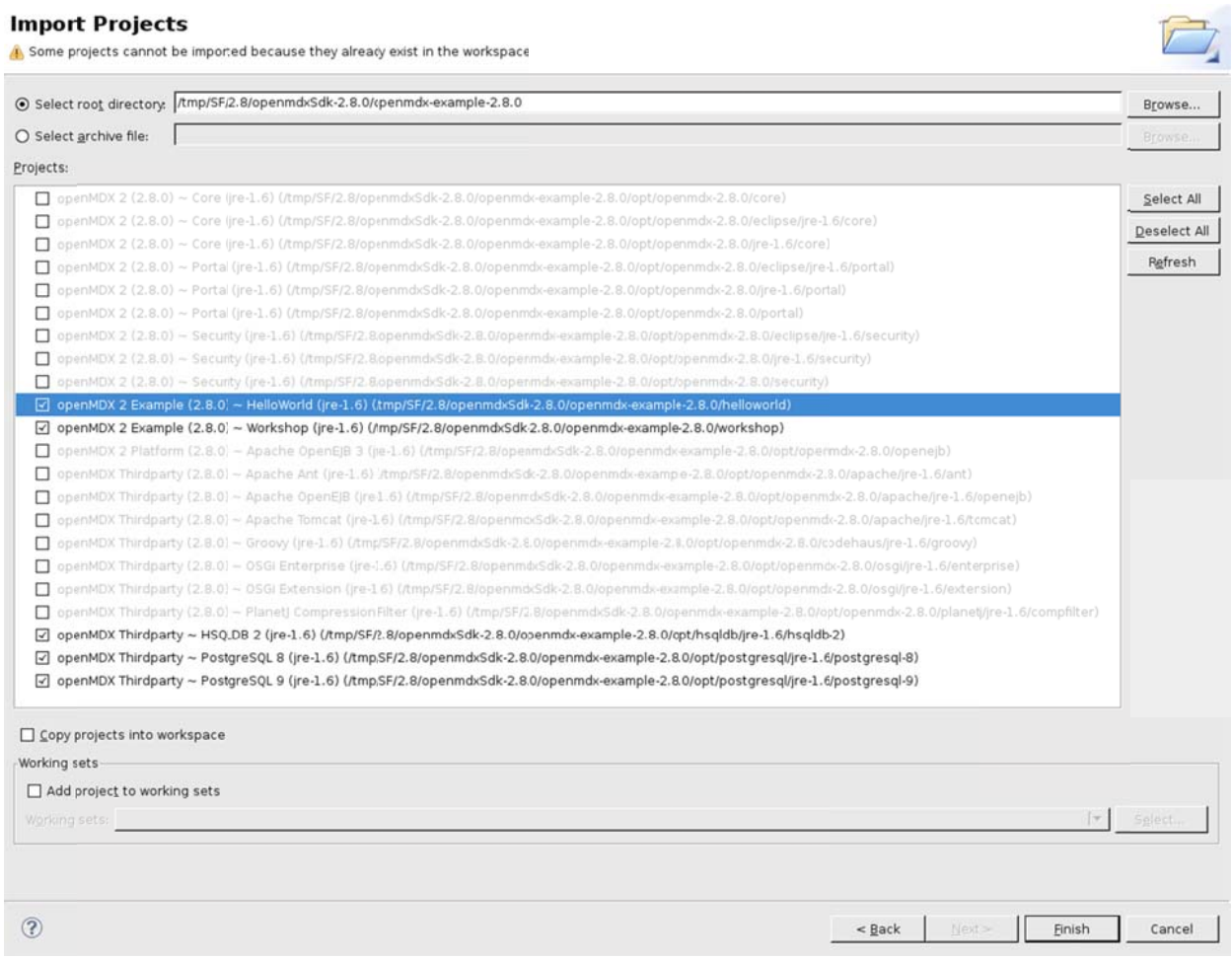
1. Make sure that you have installed *Eclipse3.4* or higher (download it from here).
2. After installation launch *Eclipse*. Create a new workspace by selecting *File>SwitchWorkspace>Other*. Enter the directory nameof the new workspace. In this guide we will use the directory*/tmp/SF/2.8/eclipse**openmdx-2.8.0.Eclipse* will be launched with a new empty workspace.
3. Close the Welcome page. Next let us install the Eclipse UML2Tools. This allows to view the UML diagrams delivered with the SDK. In Eclipse navigate to *Help>Install New Software*, lookup up and select *UML2ToolsSDK.*
4. Now we have to configure the JDK. Open the preferences dialog with *Window>Preferences*. Navigate to then try *Java>Installed JREs*. Because the *openMDX* examples require a *JDK1.6* we first have to add a *JDK1.6* compliant *JRE*. Click on *Add* and then enter *JRE6* as *JRE name* and select the home directory of an installed *JDK1.6*. *Eclipse* automatically completes the other fields of the dialog.
5. Select the newly added *JRE6* as default JDK and remove any other JDKs from the list.
6. Next open the entry *Java>Compiler* and set the compiler compliance level to 1.6.
7. Next we will import the *openMDXSDK* projects. Select*File>Import*.
8. Select *Existing Projects into Workspace*.
9. Navigate to the *openMDXSDK* installation directory and then to the project older *openmdx-2.8.0*.Eclipse recursively scans all directories. Deselect the projects *openMDX/Core*, *openMDX/Security* and *openMDX/Portal* because they occur multiple time.
10. Navigate to the *openMDX SDK* installation directory and then to the project folder*openmdx-2.8.0*. Eclipse recursively scans all directories. Deselect the projects *openMDX/Core*, *openMDX/Security* and *openMDX/Portal* because they occur multiple times as indicated below:



11. Your package explorer view now lists the projects shown below:



12. Next we import the projects from *openMDX/Example*. Open the import dialog with *File>Import>Existing Projects into Workspace*. Navigate to the directory.*openmdxSdk-2.8.0**openmdx-example-2.8.0*.Eclipse recursively scans all directories. Select the projects as indicated below:

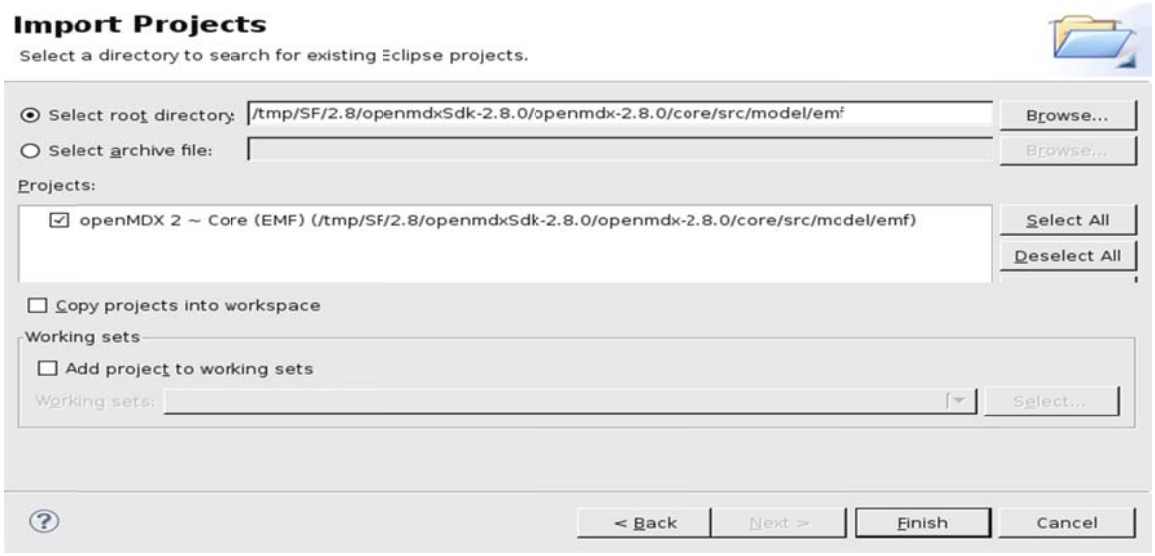


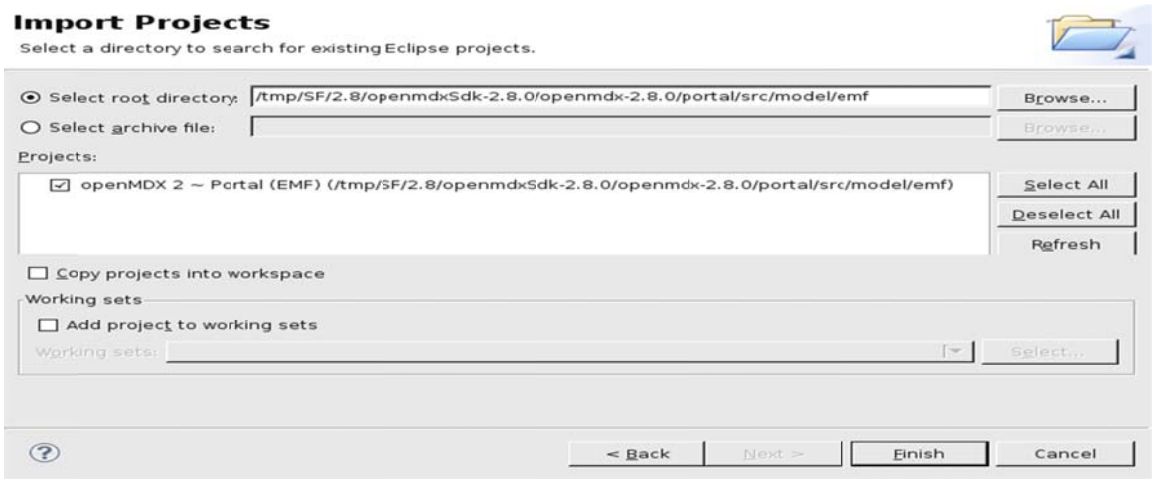
13. *Eclipse* must be able to compile all projects without errors.

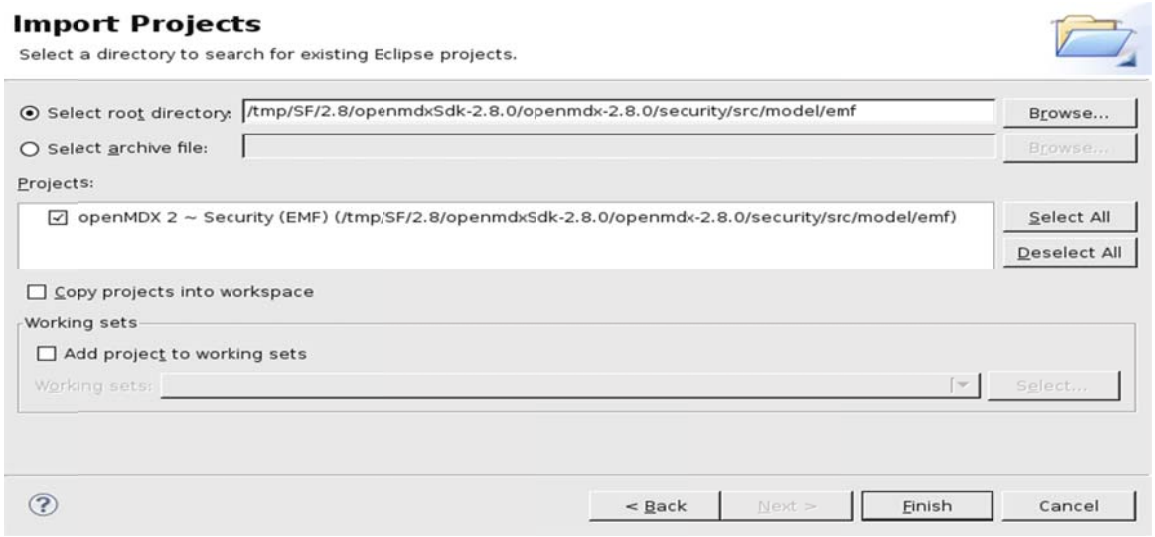
Finally, we add the EMF (EclipseModelingFramework) project files to the workspace. The projects contain the following UML class diagrams:

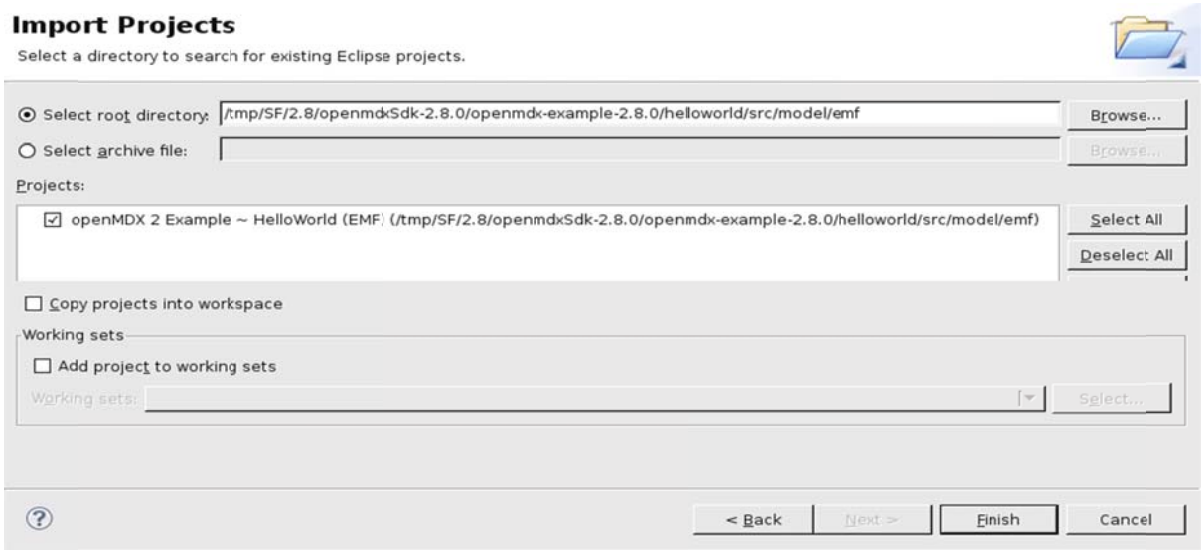
- *openMDX2~Core(EMF)*
- *openMDX2~Security(EMF)*
- *openMDX2~Portal(EMF)*
- *openMDX2Example~HelloWorld(EMF)*
- *openMDX2Example~Workshop(EMF)*

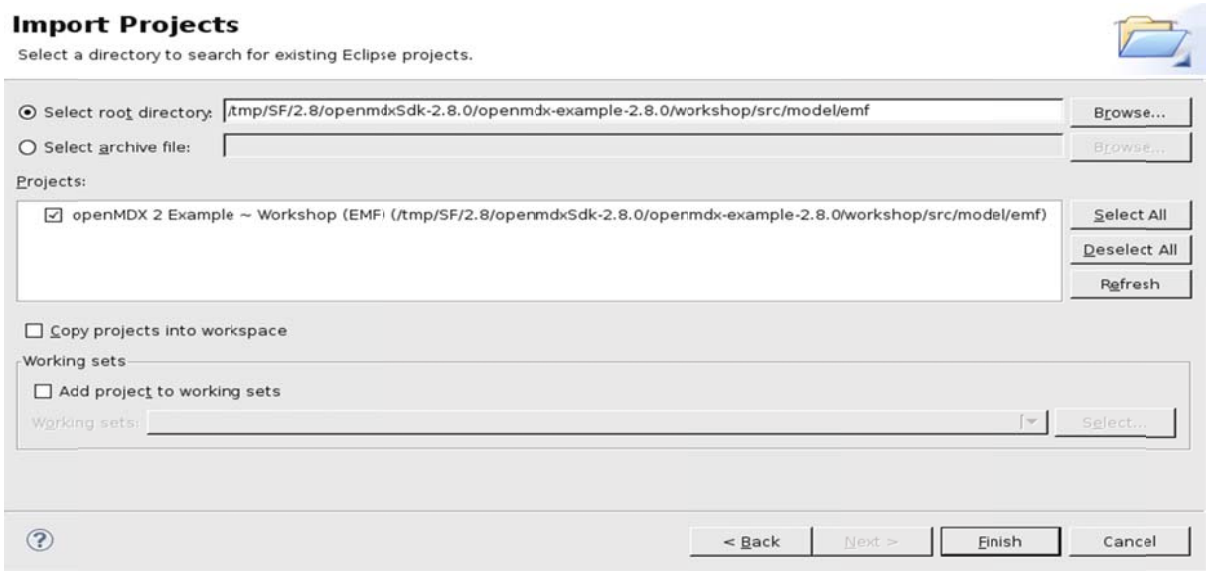
14. Import the projects as shown in the screenshots below:











15. In order to view the class diagram of the HelloWorld projects, expand the project t*openMDX2Example~HelloWorld(EMF)*andopentheclassdiagram*org.openmdx.example.helloworld1-Main.umlclass*.

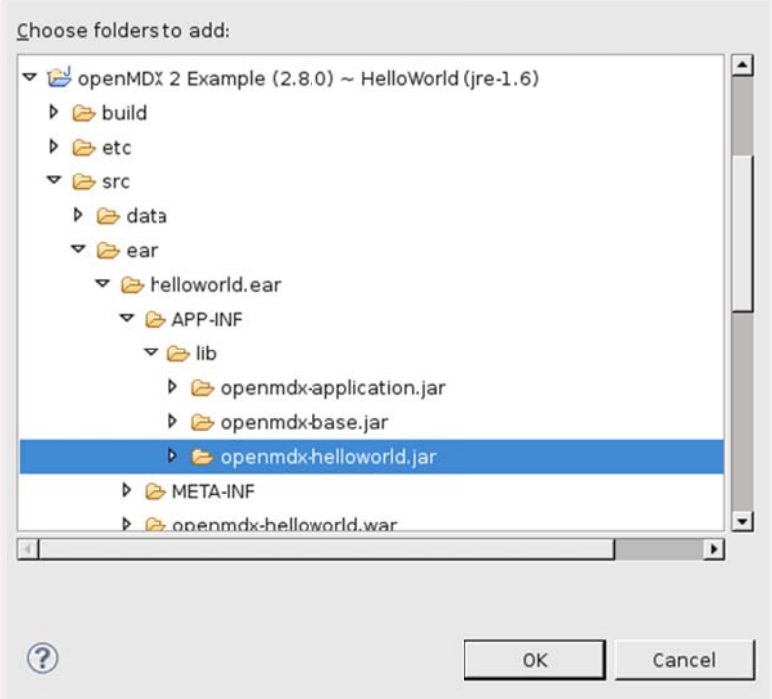
NOTE: the class diagram *models-main.Uml class* does not contain a valid diagram and will be removed in the next version of *openMDXSDK*.

The HelloWorld project only contains three classes:

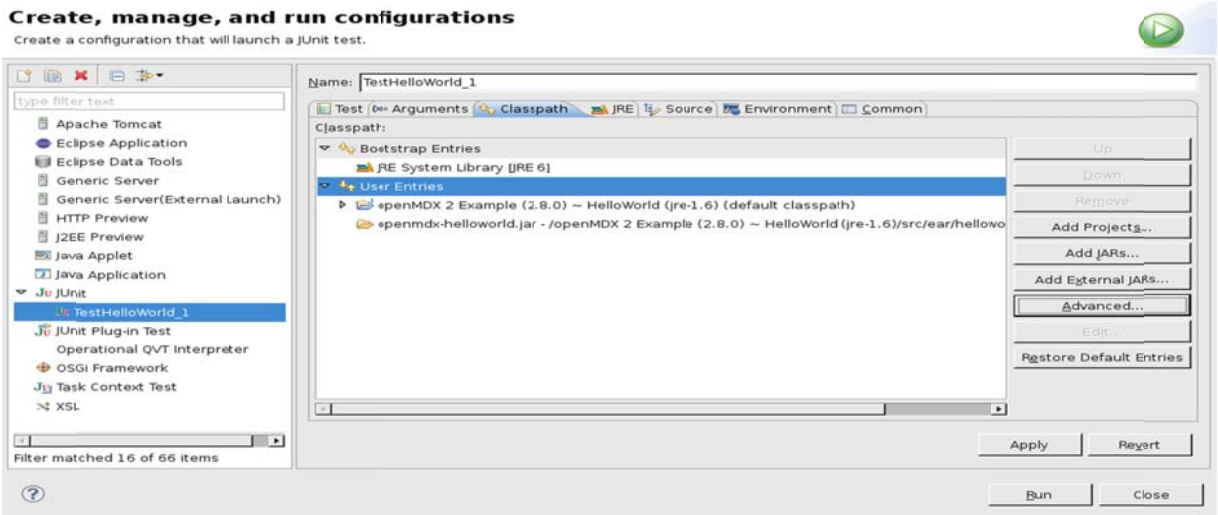
- The class *HelloWorld* defining the operation *sayHello()*
- The operation's *sayHello()* input parameter *HelloWorldSayHelloParams*
- The operation's *sayHello()*output parameter *HelloWorldSayHelloResult*

The openMDX application consists of a client and a service. The HelloWorld client invokes the operation *sayHello()*,the HelloWorld service implements the operation.

16. We are now ready to run the *!Helloworld* JUnits. Open the folder *src/java* in the project *openMDX/Example!Helloworld* and navigate to the package *org.openmdx.example.helloworld1.program*. Right click on *TestHelloWorld_1* and select *RunAs >JUnit Test*.
17. TheJUnit will report errors. To fix the problem open the run dialog with *Run>Open Run Dialog* and select *TestHelloWorld_1* in the tab *JUnit*. Select the tab *Class path* and click the button *Advanced...*.Select the option *Add Folders...*:
18. Navigate to the folder *openmdx-helloworld.jar* and select it:



The class path should now look as follows:



19. Run the test again. It should now complete successfully.

20. Do the same with the JUnit *TestExample_1* from the Workshop project located in the package *org.openmdx.test.example.workshop1* in the project *openMDX/ExampleWorkshop*. Do not forget to

    - Add the folder *openmdx-workshop.jar* to the classpath
    - Add the project *openMDXThirdpartyHSQLDB2* to the classpath
    - Start the workshop database as explained inopenMDXSDK2.8forAntStep-by-Step

Congratulations!!. You have successfully prepared *openMDXSDK2.8* for *Eclipse*.