Revista EIA, ISSN 1794-1237 Número 8, p. 131-146. Diciembre 2007
Escuela de Ingeniería de Antioquia, Medellín (Colombia)

MDA Y EL PAPEL DE LOS MODELOS EN EL PROCESO DE DESARROLLO DE *SOFTWARE*

Juan Bernardo Quintero*
Raquel Anaya**

RESUMEN

El papel de los modelos es fundamental en el desarrollo de *software* para potenciar el reúso de los diferentes elementos del *software* y facilitar la labor de los diferentes 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). Para explicar el papel de los modelos en el proceso de desarrollo de *software* este artículo explora los principales conceptos presentados en la propuesta de MDA.

PALABRAS CLAVE: reúso; modelo; lenguaje de modelado unificado (UML); arquitectura dirigida por modelos (MDA); lenguaje de restricciones de objetos (OCL); transformación; perfil UML; mapeo; marcado.

ABSTRACT

The role of models is critical in software development to enable the reuse of different software elements and to aid the work of several roles involved in the process. Model Driven Architecture (MDA) suggests a development process based on models realization and transformation. The principles in which MDA is based are abstraction, automation, and standardization. The central process of MDA is the transformation of

* Ingeniero de Sistemas Universidad de Antioquia y Magíster en Ingeniería Informática, Universidad EAFIT. Docente e investigador del Grupo de Investigación en Ingeniería de Software, Universidad EAFIT. jquinte1@eafit.edu.co

** Doctora en Ingeniería de la Programación e Inteligencia Artificial, Universidad Politécnica de Valencia. Directora del Grupo de Investigación en Ingeniería de Software, Universidad EAFIT. ranaya@eafit.edu.co

Artículo recibido 27-IX-2007. Aprobado 19-XI-2007
Discusión abierta hasta junio de 2008

models from the problem space (CIM) to platform specific models (PSM), passing across models describing a platform independent solution (PIM). In order to explain the model role in the software process development, this paper explores the main concept presented in the MDA proposal.

KEY WORDS: reuse; model; Unified Modeling Language (UML); Model Driven Architecture (MDA); Object Constraint Language (OCL); transformation; UML profile; mapping; marked.

1. INTRODUCCIÓN

El reúso de *software* es una de las estrategias que se considera promisoria para que la industria de *software* pueda enfrentar el reto de desarrollar productos con niveles de calidad y productividad adecuados en un contexto de negocio altamente complejo y dinámico y con acelerados cambios tecnológicos. El uso de plantillas, componentes de granularidad gruesa, patrones de diseño, arquitecturas de referencia, *frameworks,* entre otros, son mecanismos cada vez más utilizados por los desarrolladores de *software*. El objetivo de dichas prácticas es lograr que el reúso se integre de forma sistémica en las diferentes etapas del desarrollo, de tal manera que su impacto en los diferentes artefactos resultantes del proceso de desarrollo sea efectivo y, en lo posible, medible (Jacobson *et al.,* 1997).

Al plantear el interrogante: ¿Cuál es el gran reto de las industrias que desarrollan *software*? nos encontramos con una diversidad de respuestas que parecen convergir a la necesidad de mejorar el desempeño para maximizar las ganancias. El Human Performance Center agrupa las tendencias que sigue la comunidad informática para lograr tal propósito, en tres enfoques básicos (HPC, 2002):

- *Trabajando más rápidamente*. Mejorando las herramientas que apoyan el desarrollo de *software* (IDE[1], compiladores, generadores de código, etc.), se pueden mejorar magnitudes de hasta una unidad porcentual.
- *Trabajando más ágilmente*: analizando, evaluando y mejorando la forma de trabajar (SPI[2]), se pueden mejorar magnitudes de hasta dos dígitos porcentuales.
- *Trabajando menos*: cambiando la forma de trabajar, maximizando el reúso, no desgastándose en diseño, codificación y pruebas exhaustivas, realizando programación en el nivel de ingeniería de modelos y requisitos, de esta forma se pueden mejorar magnitudes de hasta tres dígitos porcentuales.

El planteamiento anterior evidencia la importancia de proponer estrategias de trabajo que potencien el reúso a un alto nivel de abstracción. Es aquí donde las aproximaciones avanzadas en esta área, como la arquitectura dirigida por modelos –MDA– (OMG-MDA, 2003), las líneas de productos de *software* –SPL[3]– (SEI-SPL, 2006), los lenguajes específicos de dominio –DSL[4]– (Fowler, 2006) y el desarrollo orientado por aspectos –AOSD[5]–) (AOSA, 2006) proponen pensar tempranamente en formas de reúso basadas en la manera como se hace abstracción de un problema y se especifica una propuesta de solución.

Dada la complejidad del desarrollo de una solución de *software,* las herramientas CASE (Computer Aided Software Engineering) son

1 IDE: Integrated Development Environment.

2 Tendencia conocida como Software Process Improvement

3 SPL: Software Product Line.

4 DSL: Domain Specific Language.

5 AOSD: Aspect Oriented Software Development.

imprescindibles como apoyo al trabajo del grupo de desarrollo. Una de las características que motivan la adquisición de una herramienta CASE es agilizar el proceso de desarrollo, tratando de automatizar el proceso de generación del producto final a partir de los modelos construidos. Dos de los principales inconvenientes que se presentan en el uso de las herramientas CASE son la generación limitada de código (por ejemplo, *scripts* de base de datos y esqueleto de clases en un lenguaje determinado) y la rápida obsolescencia de los modelos, debido a que, una vez realizada la generación del código, el desarrollador se concentra en completar el código generado y pierde de vista los modelos construidos.

El reto que en la actualidad motiva a la comunidad de investigadores y generadores de tecnología es proponer esquemas de desarrollo en los cuales los modelos, antes que el código, son los actores centrales del proceso de desarrollo y donde se proveen mecanismos y herramientas de trabajo integradas que asisten al desarrollador en la construcción y transformación progresivas de modelos hasta llegar a la solución final. Esta corriente de trabajo, liderada por el OMG (Object Management Group), se conoce como arquitectura dirigida por modelos (MDA).

El objetivo de este artículo es explorar los principales conceptos que rigen MDA; para tal propósito se organizó de la siguiente manera: se trabajarán los conceptos generales de MDA en la sección 2; luego, en la sección 3, se presentará el papel de los estándares en MDA; se tratarán los fundamentos de la transformación en la sección 4; después, en la sección 5, se bosquejará una propuesta para el proceso de desarrollo de *software* dirigido por modelos; y, finalmente, se mostrarán las conclusiones y trabajos futuros.

2. LA ARQUITECTURA DIRIGIDA POR MODELOS

La propuesta que planteó el OMG cuando presentó MDA centra su atención en los modelos como el elemento básico que, por medio de su evolución, permiten de manera sistemática obtener el producto final.

2.1 Importancia de los modelos en MDA

El concepto del modelado, como una de las estrategias básicas del desarrollador para entender un problema y proponer una solución, es ampliamente aceptado en la ingeniería de *software,* mucho antes del surgimiento de MDA. Sin embargo, la aplicación del modelado en la práctica diaria presenta problemas como los enunciados a continuación:

- Los modelos se usan solo como documentación. Los modelos no funcionan como un artefacto activo que contribuya en el proceso de desarrollo.
- Existen vacíos entre el modelo y la implementación de los sistemas. Los cambios en el modelo no se reflejan en el código y los cambios en el código no se reflejan en los modelos, sólo se genera el código de los modelos la primera vez y nunca se actualiza.
- No hay una adecuada mezcla de modelos. Vistas desconectadas de un sistema (desconexión horizontal) y grupos de modelos desconectados (desconexión vertical).
- No hay transformación de modelos. Pocos lenguajes de transformación populares y pocas herramientas que soporten estas transformaciones.

Lo anterior les atribuye a los modelos la fama de una costosa y pesada carga que complica la labor de los participantes en el proceso de desarrollo. La MDA rescata la importancia de los modelos como estrategia clave para entender y especificar una solución de *software* y progresivamente obtener la solución final. Las siguientes son algunas definiciones de modelo de la comunidad de MDA (OMG-MDA, 2003; Kleppe *et al.,* 2003).

- Un modelo es la descripción de un sistema (o de una parte) en un lenguaje bien definido.

- Un lenguaje bien definido es un lenguaje con una forma definida (sintaxis) y significado (semántica) que sea apropiado para ser interpretado automáticamente por un computador (Kleppe *et al.*, 2003).
- Un modelo se presenta con frecuencia como una combinación de dibujos y de texto (OMG-MDA, 2003).

2.2 El proceso de desarrollo basado en modelos

El esquema planteado en la figura 1 toma como referente la propuesta de desarrollo basada en modelos planteada por Kleppe *et al.* (2003), con el propósito de compararla con el desarrollo tradicional. En los dos enfoques del proceso, cada etapa del desarrollo produce artefactos que sirven como insumo para la siguiente etapa. 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. Es decir, en el enfoque tradicional, la consistencia del proceso de transformación depende de la habilidad de 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 tal propósito MDA caracteriza los siguientes tipos de modelos:

- CIM. Representa los modelos independientes de la computación (Computationally-Independent Model) que caracterizan el dominio del problema. Este tipo de modelos surge ante todo en procesos de modelado de negocio e idealmente se conciben antes del levantamiento de requisitos para una aplicación particular.
- PIM. Representa 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 (Platform-Independent Models). Estos modelos surgen como resultado del análisis y diseño.
- PSM. Son los modelos derivados de la categoría anterior, que contienen los detalles de la plataforma o tecnología con que se implementará la solución, de ahí su nombre de modelos específicos de la plataforma (Platform-Specific Models).

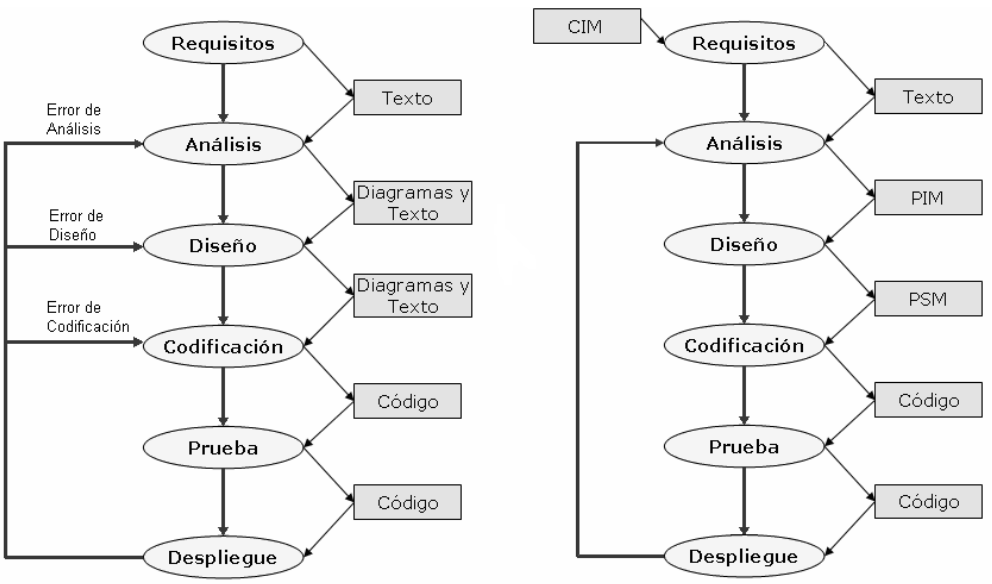

Figura 1. Proceso tradicional y Proceso con MDA

Estos modelos se construyen entre el diseño y la codificación.

- Y, finalmente, el código mismo que se genera después de la codificación y las pruebas.

A la luz de este enfoque, surgen nuevos esquemas de trabajo en los que se distinguen los diferentes roles de los participantes del proceso (figura 2). El analista de negocio, experto en el dominio del problema, que modela una realidad por medio de CIM; el arquitecto y el diseñador que definen una propuesta de solución (transformación del CIM en PIM) y progresivamente la concretan (transformación de PIM a PIM), hasta llegar a un diseño detallado; el desarrollador o el probador que tiene amplio conocimiento de la tecnología y decide la manera más adecuada como el diseño detallado se implementará en una plataforma particular (transformación del PIM a PSM) (OMG-BP, 2004).

2.3 Principios de MDA

La MDA aparece como respuesta a dos problemas fundamentales dentro de la industria informática.

- *La diversidad de plataformas y tecnologías*: en la actualidad se oye hablar con frecuencia de los objetos distribuidos, los componentes, los aspectos o los *web services,* entre otras tantas estrategias tecnológicas en las que no hay mucha interoperabilidad y que tienen tendencia a aumentar.

- *La acelerada evolución tecnológica*: esto ocasiona que las plataformas muy pronto sean obsoletas. Surgen, entonces, interrogantes como: ¿Cuál tecnología va a salir mañana? ¿Cuánto va a durar la última versión de una plataforma? ¿Cómo protejo mi inversión?

Por consiguiente, nunca se tiene un verdadero estándar en el nivel de sistema operativo, servidores, plataformas o *middleware,* lo cual dificulta considerablemente el reúso de los artefactos *software,* y más aun en etapas tempranas del proceso. Las estrategias para alcanzar beneficios fundamentales, como productividad, interoperabilidad, "portabilidad" y facilidad de mantenimiento, se plantean en las ideas del manifiesto MDA (Booch *et al.,* 2004):

- ***Representación directa.*** Esta estrategia se basa en el principio de abstracción, que hace énfasis en

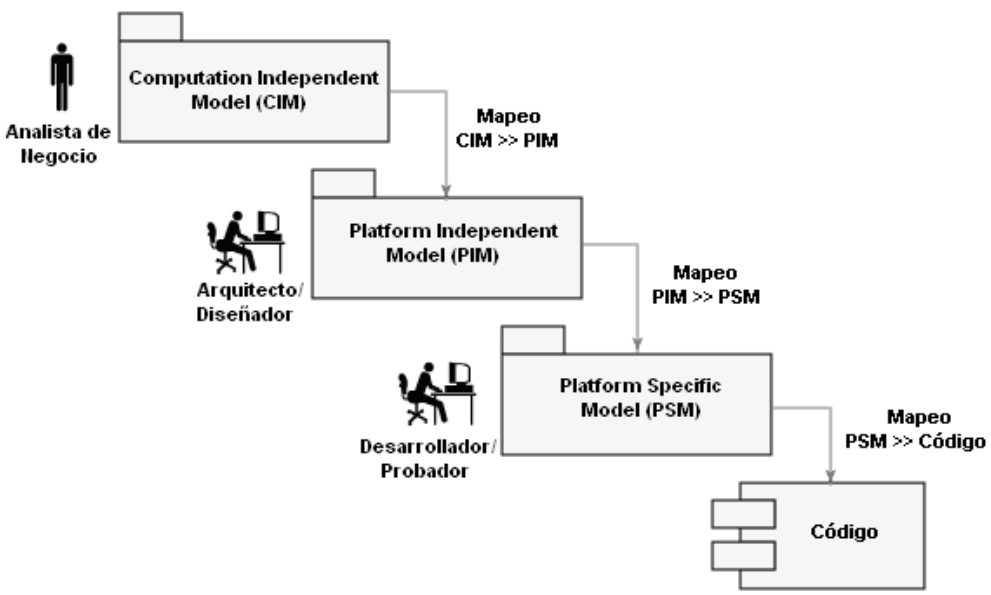

Figura 2. Proceso y roles en MDA

el dominio del problema más que en la tecnología. Los diferentes tipos de modelos mencionados buscan precisar una semántica que claramente separe los aspectos relevantes del problema de las decisiones de tecnología. Esta estrategia parte de la hipótesis de que los impactos considerables en el desarrollo y mantenimiento de una solución de *software* se dan por cambios en el negocio, más que por la diversidad de plataformas y la evolución tecnológica.

- ***Automatización.*** La propuesta de MDA fortaleció y dinamizó el papel que las herramientas CASE tienen en el desarrollo de soluciones. Surgen nuevas funcionalidades que deben ser soportadas por las herramientas como el intercambio de modelos, verificación de consistencia, transformación de modelos y manejo de metamodelos, entre otras. Desde la perspectiva de MDA, el papel de las herramientas es esencial para apoyar de forma consistente y sistémica el proceso de desarrollo visto como un proceso de transformación de modelos.
- ***Estándares abiertos.*** El uso de estándares se ha constituido en el medio que ha posibilitado el reto de integrar herramientas robustas de apoyo al desarrollo. Por ejemplo, los estándares como UML deben expresarse en XML, de esta forma resultan de gran utilidad en mecanismos de transformación de modelos que utilizan otros estándares como XSLT[6].

2.4 Reúso de modelos

Mientras más temprano se piense en el reúso, más esfuerzo se debe realizar, por ejemplo, para conseguir herramientas que soporten transformaciones a estos niveles, o para definir acuerdos con el grupo de desarrollo con respecto a los elementos de granularidad gruesa. Sin embargo, esto se verá recompensado con el impacto positivo en el desempeño del grupo, puesto que diminuirán las labores repetitivas, los errores por intervención humana y el tiempo para completar algunas tareas.

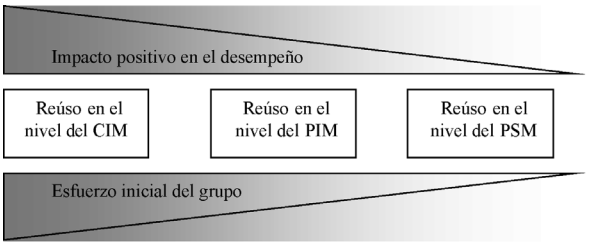

Figura 3. Impacto del reúso en el desarrollo basado en MDA

La figura 3 muestra cómo impacta en términos generales el reúso dentro del proceso de desarrollo de *software*. Con el tiempo el esfuerzo inicial realizado por el grupo de desarrollo disminuirá, pues empezará a familiarizarse con las herramientas y los acuerdos tempranos realizados. Todo esto evidencia la importancia de que los modelos estén correctos, pues un error en alguno de los modelos tempranos tendría graves consecuencias, por el efecto de propagación que se presentaría en las operaciones de transformación.

2.5 Niveles de madurez de los modelos

Dada la importancia de los modelos en el contexto de la propuesta de MDA, Jos Warmer (2002) presenta los "niveles de madurez de los modelos" (MML), lo cual se asemeja a los niveles de capacidad planteados en CMMI (Capability Maturity Model Integration) (SEI-CMMI, 2002). Por medio del MML, se puede descubrir en qué estado de madurez se encuentra un grupo de desarrollo con respecto al uso de modelos. Además se puede observar que estos niveles de madurez están estrechamente relacionados con el uso de UML como lenguaje estándar de especificación de sistemas intensivos en *software*. Los niveles planteados en MML y su estado con respecto al uso de UML son:

[6] XSLT: Extensible Style Language Transformation.

- **MML 0 Sin especificar.** La especificación del *software* se conserva en la cabeza de los desarrolladores. En este nivel no se utiliza UML.
- **MML 1 Especificación textual.** La especificación del *software* está escrita en uno o más documentos en lenguaje natural. Tampoco se hace uso de UML en este nivel.
- **MML 2 Texto con diagramas.** La especificación del *software* está en uno o más documentos en lenguaje natural y adicionalmente en varios diagramas de alto nivel para explicar la arquitectura global. Aquí se empieza a hacer uso moderado de UML.
- **MML 3 Modelos con texto.** La especificación del *software* está escrita en uno o más modelos. En forma adicional el texto en lenguaje natural se usa para explicar el trasfondo y la motivación de los modelos. El uso de UML se realiza de manera extensiva por todos los desarrolladores.
- **MML 4 Modelos precisos.** La especificación del *software* está escrita en uno o más modelos. Además el texto en lenguaje natural se usa para explicar el trasfondo y la motivación de los modelos. Los modelos son precisos y alcanzan a tener una relación directa con el código real. En este nivel se hace uso extensivo de UML acompañado por lenguajes como OCL que permiten especificar restricciones de manera formal.
- **MML 5 Solo modelos.** Los modelos son precisos y detallados lo suficientemente para permitir una completa generación de código. El código es invisible. El lenguaje de modelado equivale a un lenguaje de programación de alto nivel. En este nivel el lenguaje de especificación transciende el uso de UML para llegar al concepto de lenguajes de transformación o lenguajes de dominio específico que automatizan de forma completa todas las fases del desarrollo.

3. EL PAPEL DE LOS ESTÁNDARES EN MDA

Los diversos estándares en los que se apoya la propuesta de MDA tienen el propósito de lograr la interoperabilidad de las herramientas y plataformas, posibilitando evadir los problemas por la diversidad de plataformas y la evolución tecnológica. A continuación se describen los principales estándares en los que se soporta MDA.

3.1 Arquitectura de cuatro niveles (MOF)

El OMG plantea una arquitectura de cuatro niveles para la definición de sus estándares, en donde cada capa se define como instancia de la anterior. Esta arquitectura de modelos denominada MOF (Meta Object Facility) representa el nivel meta más general (M3) y tiene el objetivo de permitir la incorporación de nuevos lenguajes de modelado (metamodelos) para propósitos específicos (OMG-MOF, 2003). Un metamodelo es un modelo que define el lenguaje para expresar un modelo. A continuación se definen las diferentes capas especificadas en la arquitectura del OMG que sustenta MOF.

- **Capa M3 (Metametamodelo).** Corresponde a MOF, es una especificación que define un lenguaje abstracto para especificar, construir y manejar elementos comunes a cualquier metamodelo.
- **Capa M2 (Metamodelos).** Especifica las entidades de un lenguaje de modelado. Los lenguajes que se han definido como instancias de MOF son: UML, CWM[7] y MOF en sí mismo. Adicionalmente se definen metamodelos para otros propósitos como objetos de negocio, *workflows* y modelos de componentes (OMG-MOF, 2003).

[7] CWM: Common Warehouse Metamodel.

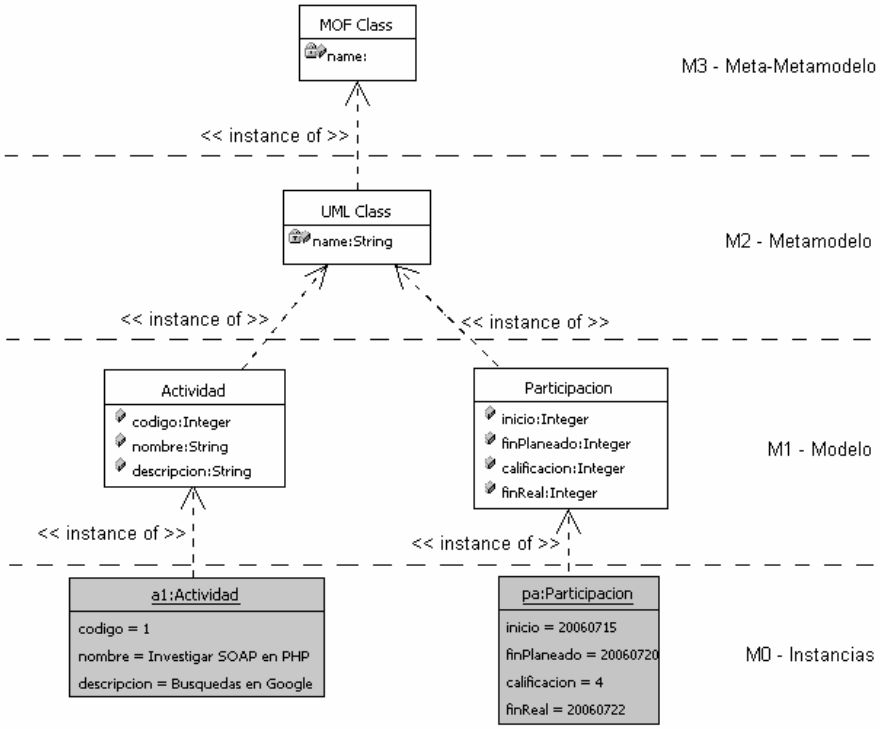

Figura 4. Ejemplo de las capas de modelado

- **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).

La figura 4 ejemplifica cada capa de la arquitectura para los estándares de OMG e ilustra la relación entre cada capa.

Aunque esta arquitectura de cuatro niveles abre la posibilidad de definir a partir de M3 cualquier lenguaje para especificación de soluciones, UML representa el lenguaje estándar preestablecido, definido en la capa M2, a partir del cual se pueden especificar los modelos definidos por MDA en cualquiera de sus categorías (CIM, PIM, PSM). A continuación se describe brevemente el rol que desempeñan los principales estándares alrededor de UML como una de las capas de MOF (OMG-UML, 2003).

3.2 Lenguaje de restricciones de objetos (OCL)

El OCL es un lenguaje para especificar restricciones sobre los objetos de un modelo UML (Cernuda, 2002; OMG-RfPOCL, 2003). Usualmente existen algunos detalles de la especificación que no pueden expresarse vía modelos UML; resulta, entonces, necesario llenar los faltantes que se presentan utilizando un lenguaje como OCL (OMG-OCL, 2003). Uno de los principales objetivos que se persiguen con OCL es precisar la especificación de los modelos escritos en UML, evitando ambigüedades en su interpretación y garantizando su consistencia semántica.

Los principales tipos de restricciones que se pueden construir utilizando OCL son (Kleppe y Warmer, 1999): *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) Postcondición*: condición que debe ser cierta después de ejecutar una operación de una clase. Es importante recalcar que usando OCL se pueden escribir restricciones en el nivel de los metamodelos; de esta forma, por ejemplo, se definen las reglas de consistencia de un lenguaje como UML.

3.3 Perfiles

Desde la aparición de UML 2.0 (Shekhar y Prabhakar, 2003), el concepto de perfil ha tomado un gran auge, sobre todo por su gran utilidad en los procesos de transformación de modelos (Fuentes y Vallecillo, 2004). Un perfil está constituido por estereotipos, valores etiquetados y restricciones, y sirve para extender la semántica de los modelos construidos con UML a un dominio específico o a una plataforma particular.

La figura 5 muestra un ejemplo de aplicación de un perfil para el diseño de bodegas de datos seguras (Villarroel *et al.*, 2003) que usa los tres elementos básicos de los perfiles, de la manera como sigue.

- ***Estereotipos.*** Representa una distinción de uso de un elemento en UML que se utiliza para añadir nuevos elementos de modelado. Se representa con una palabra encerrada entre dobles símbolos "<" y ">", así: <<estereotipo>>. En el ejemplo el estereotipo UserProfile se le aplica a la clase PerfilUsuario que contiene la información de todos los usuarios que tendrán acceso a este sistema.
- ***Valores etiquetados.*** Si con los estereotipos se pueden añadir elementos a UML, con los valores

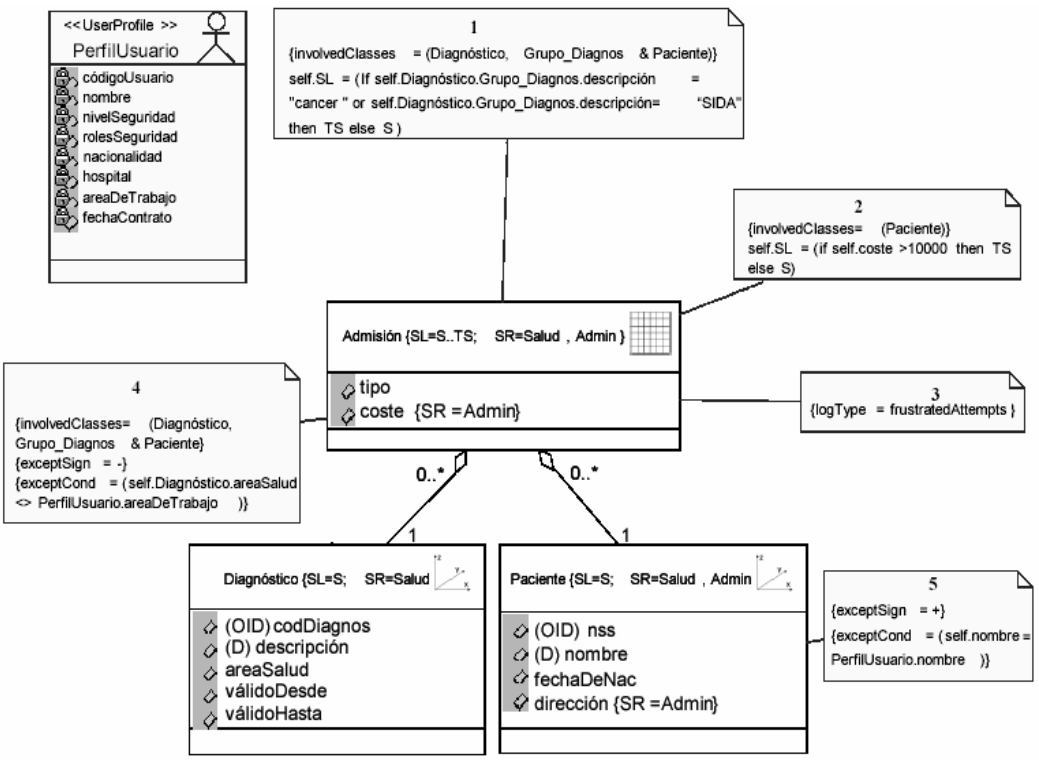

Figura 5. Diagrama UML en el que usa un perfil

etiquetados se añaden nuevas propiedades, agregando nueva información en la especificación de un elemento. Se representan con una etiqueta seguida del símbolo "=" y el valor, encerrados entre llaves, acompañando el nombre del elemento. En el ejemplo aparecen los valores etiquetados de SL SecurityLevel (con los posibles valores U=Unclassified, C=Confidencial, S=Secret y TS=TopSecret) y SR SecurityRoles (Salud y Admin).

- ***Restricciones.*** Como se explicó en el numeral 3.2, las restricciones se escriben usando OCL y se suelen representar en una nota asociada al objeto sobre el que se aplica la restricción. Para el ejemplo se está empleando un "dialecto" particular de OCL llamado OSCL (Fernández *et al.*, 2001), que se usa específicamente en el diseño de sistemas de información seguros. Obsérvese como en algunas de las restricciones se usan los valores etiquetados dentro de la definición de las reglas.

3.4 XMI (XML Metadata Interchange)

El XMI es un lenguaje para permitirles a los usuarios desarrolladores de *software* el intercambio de modelos UML. Es una especificación del OMG definida con el objetivo de facilitar el intercambio de modelos entre herramientas de modelado y repositorios de metadatos basados en la especificación (OMG-XMI, 2003).

El XMI provee un mecanismo de intercambio de modelos entre herramientas CASE utilizando XML. Su importancia se evidencia en que las herramientas CASE más conocidas utilizan XMI como especificación estándar para la funcionalidad de intercambio (Iyengar, 1999). La figura 6 muestra cuáles son las ventajas de usar XMI, comparando un ambiente desarrollado con 6 herramientas que utilizan UML contra un ambiente que usa UML y XMI en el contexto de MDA. Como se observa en la parte derecha de la figura 6, el uso de XMI ha facilitado enormemente la distribución de las diversas responsabilidades que se exigen en un entorno de desarrollo con el enfoque MDA: las funcionalidades como edición de modelos, repositorio de modelos, verificación de modelos y transformación utilizan, producen o modifican documentos XMI que sirven para los propósitos de MDA.

Una de las razones de la aceptación de XMI como lenguaje de intercambio de modelos es la tecnología ya desarrollada alrededor de XSLT, un modelo de procesamiento de documentos XML que incluye a la vez el lenguaje para describir las reglas de transformación. De esta forma se pueden construir diversas plantillas, que permiten generar documentos, recorrer su estructura y realizar transformaciones.

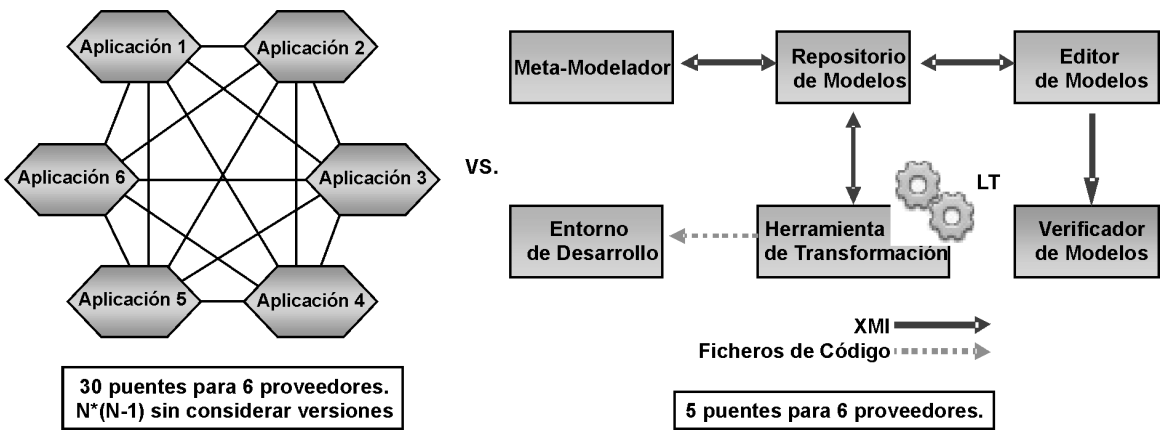

Figura 6. Importancia de XMI en el proceso con MDA

4. TRANSFORMACIÓN DE MODELOS

La transformación de modelos se considera el proceso central de MDA. Con el propósito de lograr un estándar para la transformación, OMG inicia un proceso de estandarización que favorece la presentación de propuestas (RFP Request for Proposal) por parte de toda la comunidad informática alrededor del estándar denominado QVT (Queries/Views/ Transformations). Este estándar está basado en MOF y pretende establecer un lenguaje para la transformación de modelos (T), un lenguaje para consulta de modelos (Q) y un lenguaje para la definición y generación de vistas (V) que facilite el análisis de modelos desde diferentes perspectivas de los desarrolladores. En esta sección se presentan las consideraciones más importantes alrededor del proceso de transformación.

4.1 Transformación

La transformación es el proceso que, basado en una serie de reglas, define los mecanismos para el paso de un modelo origen a un modelo destino. A partir de la estandarización promovida por QVT y atendiendo a las necesidades prácticas de la transformación de modelos, surgen diversas estrategias de transformación que van más allá del uso de elementos del metamodelo y definen mecanismos de transformación basados en tipos de elementos, patrones, información de la plataforma y otros modelos e información adicional (OMG-MDA, 2003).

La figura 7 ilustra la arquitectura de transformación de modelos; en ella se muestra cómo se definen y aplican las transformaciones que se basan en el uso de metamodelos (Jezequel, 2005).

Una de las ventajas de esta arquitectura es lograr que de un mismo PIM se pueda generar más de un PSM. Para permitir la interoperabilidad entre los diferentes PSM o las diferentes porciones

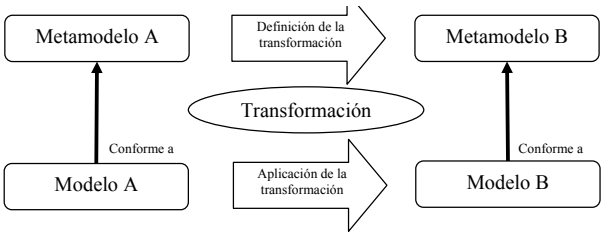

Figura 7. Arquitectura de la transformación

de código generados, surgen los llamados puentes (bridges). Estos puentes podrían, por ejemplo, definir las relaciones entre un PSM que representa una solución en una base de datos relacional y un PSM que representa una solución en una plataforma J2EE; el puente estaría representado por la lógica de acceso a datos de la aplicación, y de esta manera se podrían coordinar y tener en cuenta en cada uno de los modelos los cambios realizados en el otro. A continuación se describen las dos principales operaciones que deben ser realizadas sobre los modelos en su proceso de transformación.

4.2 Mapeo (mapping)

El mapeo es la correspondencia que se establece entre cada uno de los elementos de un tipo de modelo origen y otro destino. En el momento de establecer un mapeo se puede realizar una mezcla de las siguientes estrategias (OMG-MDA, 2003).

- ***Por tipo***: especifica un mapeo desde un modelo construido usando tipos específicos de un lenguaje origen, para expresar un modelo con tipos específicos en el lenguaje destino.
- ***Por metamodelo***: es un mapeo de tipos, donde tanto los tipos del modelo origen como los del destino están especificados a MOF.
- ***Por instancia***: se especifican elementos del modelo origen para transformarse de una forma particular, dada la selección de una plataforma específica, en el modelo destino.

4.3 Marcado (marking)

Identificación de cada uno de los elementos del modelo origen, para saber qué reglas y qué tipo de mapeo se les aplicará en la transformación. Las marcas para un modelo suelen provenir de diferentes fuentes (OMG-MDA, 2003), por ejemplo:

- Tipos desde un modelo, especificados por clases, asociaciones u otros elementos
- Roles de un modelo, por ejemplo, desde patrones
- Estereotipos de un perfil UML
- Elementos de un modelo MOF
- Elementos de un modelo especificados en algún metamodelo

4.4 Vista integrada del proceso de transformación

Para la mejor comprensión de los conceptos propios de la transformación, se ilustra con un caso la forma como el mapeo y el marcado participan en la transformación de un PIM a un PSM y a código, involucrando los conceptos propios de la plataforma, que para el ejemplo se supone J2EE[8].

Los enfoques para la transformación de modelos pueden agruparse en dos categorías (Haywood, 2004). El enfoque *elaboracionista* que parte de modelos ejecutables hasta en un 70 % que no contienen toda la información (por ejemplo, el comportamiento), la información perdida se adiciona refinando el PSM o el código y posibilitan la ingeniería de ida y vuelta. El enfoque *traslacionista* que parte de modelos completos ejecutables 100 %; los compiladores de modelos tienen toda la responsabilidad de generar el sistema completo; generalmente este enfoque se apoya en lenguajes formales, como los ASL[9] (OMG-UAS, 2002), que recogen toda la semántica necesaria para hacer una transformación completa.

En la figura 8 se muestra una comparación del proceso de transformación desde los enfoques elaboracionista y traslacionista. Se parte de un modelo conceptual (PIM) representado por un diagrama de clases de alto nivel que define los conceptos relevantes en el espacio del problema (cliente y pago); se realiza una operación de marcado sobre el modelo, en este caso ambos términos se corresponden con el estereotipo <<*entity*>>; este modelo marcado pasa por un proceso de transformación. Obsérvese como la marca <<*entity*>> puesta como estereotipo en las clases del PIM representa una semántica que consigue impactar transversalmente el proceso de transformación hasta llegar al código. Mientras que en el enfoque elaboracionista la transformación se realiza de forma progresiva por medio de un proceso asistido, el enfoque traslacionista toma la especificación inicial y a partir de ella genera el código final.

5. MDA Y EL PROCESO DE DESARROLLO DE *SOFTWARE*

En esta sección se plantean los principales interrogantes que surgen con respecto a la manera de articular MDA en la práctica del desarrollo de *software*. Hoy se plantean interrogantes alrededor de lo que finalmente es MDA: ¿Una nueva aproximación de desarrollo de *softwa*re?, ¿un conjunto de estándares?, ¿una propuesta de transformación de modelos? Desafortunadamente la respuesta no es categórica. Diversos autores tienen opiniones diferentes con respecto al alcance de la propuesta MDA (Muñoz y Pelechado, 2004).

Atkinson y Kühne (2003) no hablan directamente de MDA sino del concepto de desarrollo dirigido por modelos (MDD[10]); los métodos que sigan esta aproximación deben especificar (a) los metamodelos

8 J2EE: Java 2 Enterprise Edition.

9 ASL: Action Specification Language.

10 MDD: Model Driven Development.

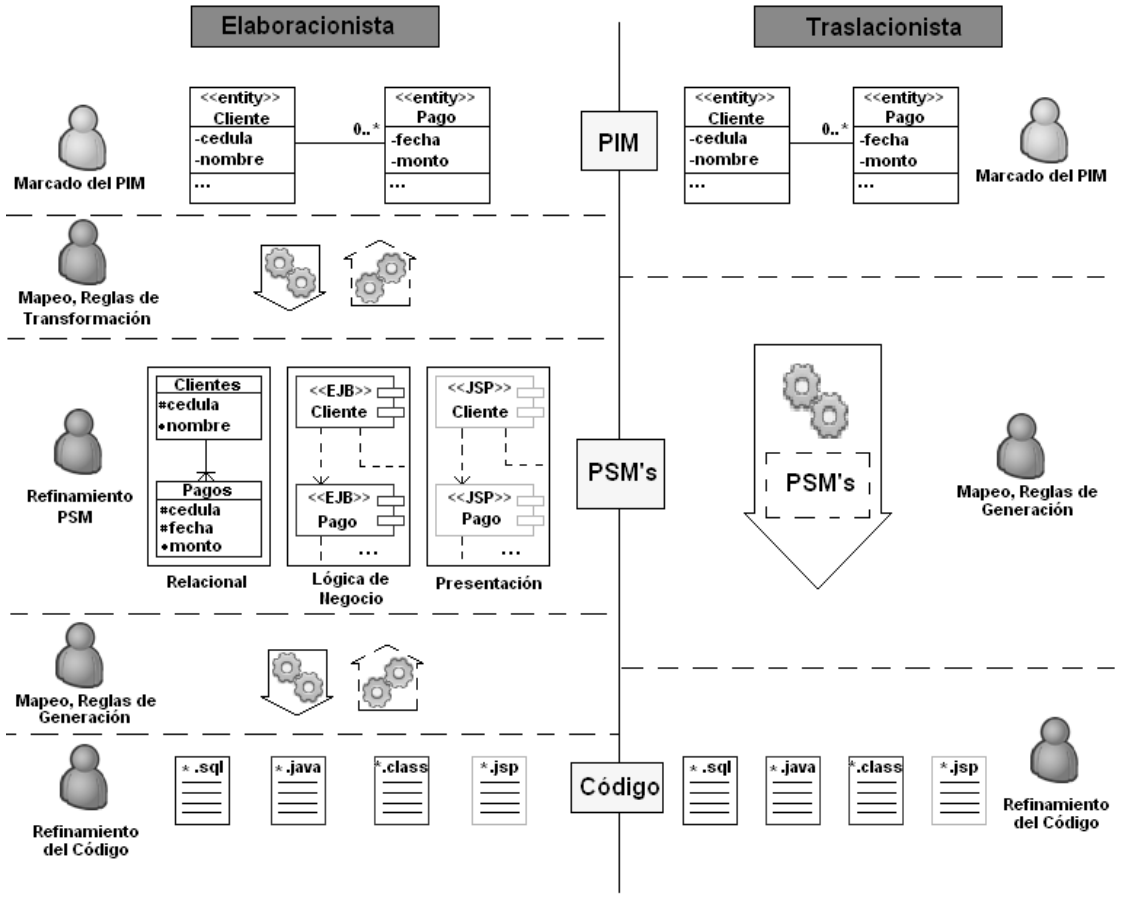

Figura 8. Mapeo y el marcado en enfoques elaboracionistas y traslacionistas

de los modelos, (b) la sintaxis de los modelos y (c) las transformaciones de los modelos a código fuente. Al ser MDD una aproximación, sólo proporciona una estrategia general para seguir en el desarrollo de *software* sin prescribir las técnicas por utilizar, ni fases del proceso que se siguen. Por su parte, Mellor *et al*. (2003) y los editores de IEEE Software definen MDA como un conjunto de estándares de OMG que permiten seguir la aproximación MDD. La MDA, por lo tanto, no es por sí un método que define técnicas, etapas, artefactos, etc., solamente proporciona la infraestructura tecnológica y conceptual con la cual construir estos métodos MDD. Esto significa que MDA no puede utilizarse directamente para desarrollar *software;* es necesario definir métodos precisos que proporcionen a los equipos de desarrollo pautas y técnicas que puedan seguir y utilizar.

Desde una perspectiva práctica es importante analizar qué tan factible es integrar MDA con marcos metodológicos robustos como RUP[11] o livianos como las propuestas de los modelos ágiles. Mientras que MDA provee los estándares y mecanismos para concebir una arquitectura alrededor de modelos y su proceso de transformación, los marcos metodológicos aportan las pautas y características del equipo de trabajo y la gestión del proyecto. Dadas las metas ambiciosas de MDA, se puede afirmar que dicho enfoque, por lo general, se tiende a asociar a procesos

[11] RUP: Rational Unified Process.

de desarrollo robustos y entornos de desarrollo altamente complejos; no obstante, existen corrientes que encuentran oportunidad de integrar los enfoques de MDA con las propuestas de modelos ágiles. Agile MDA es una propuesta basada en la noción de que el código y los modelos ejecutables son operativamente lo mismo y, por lo tanto, los principios de la alianza ágil pueden ser igualmente aplicados a los modelos (Mellor *et al.*, 2003). Por su parte, Ambler (2003) se apropia del término de AMDD[12] para referirse al desarrollo ágil dirigido por modelos, que promueve el uso de herramientas de apoyo simples y diagramas UML orientados a entender y analizar las necesidades de los usuarios, siguiendo un proceso iterativo e incremental.

Al igual que los estándares y las herramientas, conceptos como el mapeo y el marcado cumplen un papel determinante en el desarrollo dirigido por modelos. La tabla 1 ilustra la propuesta para la secuencia de pasos para seguir cuando se desarrolla *software* y se aplican los principios de MDA, complementando cada paso con el rol que desempeña cada tarea y los tipos de herramientas que podrían llegar a apoyar su automatización.

6. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se ha realizado una revisión general de los principios, estándares y mecanismos en los que está soportada la propuesta de MDA. Una

Tabla 1. Propuesta para el desarrollo dirigido por modelos

No.	Tarea	Rol que la desempeña	Herramientas que la apoyan
1	Definir los metamodelos base (origen y destino)	Metamodelador	Entornos de metamodelado
2	Definir reglas de transformación PIM-PSM (mapeo)	Transformador de modelos	Herramientas de transformación
3	Construir el PIM (generalmente UML)	Analista	Herramientas de modelado con soporte XMI
4	Marcar el PIM (marcado)	Arquitecto	Herramientas de transformación y herramientas de modelado con soporte XMI
5	Probar el PIM (validación y verificación)	Verificador de modelos	Entornos de verificación de modelos con soporte OCL
6	Transformar PIM a PSM (transformación)	Diseñador	Herramientas de transformación y entornos de despliegue de aplicaciones
7	Probar PSM (pruebas funcionales)	Probador	*Frameworks* de pruebas unitarias y entornos de verificación de modelos con soporte OCL
8	Generar código (generación)	Codificador	Herramientas de transformación y herramientas de modelado con apoyo a la generación de código
9	Desplegar la aplicación (despliegue)	Desarrollador	Entornos de despliegue de aplicaciones
10	Probar la aplicación (pruebas de integración)	Probador de integración	Entornos de despliegue de aplicaciones

[12] AMDD: Agile Model Driven Development.

de las ideas que consideramos más prometedoras de este enfoque es lograr una separación del modelado del espacio del problema y los modelos del espacio de la solución, de tal manera que los detalles tecnológicos e ingenieriles que son irrelevantes para la funcionalidad del negocio se encuentren plasmados en un modelo de descripción de la plataforma (PDM[13]) que se puede fundir con el PIM para generar un PSM. Esta separación clara de los niveles de abstracción les permite a los desarrolladores enfocarse en el dominio del negocio y el cumplimiento de los requisitos más que en el dominio de la tecnología.

La abstracción, la estandarización y la automatización son los tres pilares en los que se apoya MDA. Los estándares promovidos por medio de QVT han favorecido la definición de diversas aproximaciones de transformación y han dinamizado el papel de las herramientas CASE alrededor de funcionalidades particulares que requiere este enfoque.

La transformación de modelos es el mecanismo central que promueve MDA, sin embargo, no es exclusivo de MDA. Este tema se está estudiando desde la perspectiva de la Ingeniería de Modelos (MDE[14]) y abarca enfoques que van más allá de MDA como lenguajes de dominio específico o lenguajes de transformación (Bézivin, 2004). Las diversas corrientes alrededor del tema de transformación se mueven en los enfoques elaboracionista o traslacionista que fueron mencionados.

Surgen interesantes áreas de investigación y trabajos futuros que podrían agruparse en los siguientes frentes:

- Profundizar en aspectos como los roles, los flujos de trabajo y las etapas, en el marco de la propuesta para el desarrollo dirigido por modelos bosquejada en este trabajo. Para lograr esto, es necesario tomar como referentes los planteamientos expuestos en las propuestas del MDSD[15] (Völter y Stahl, 2006). También se podrían plantear las estrategias y el esquema de integración de *frameworks,* que sirvan de punto de partida para la construcción de una herramienta MDA que apoye dicha propuesta.
- Estudiar las propuestas de modelado de negocio y plantear estrategias para que las herramientas de transformación le den una amplia cobertura al soporte del CIM, y de esta forma permitir el reúso desde etapas más tempranas.
- Definir un *framework* que provea mecanismos para que MDA sirva de eje integrador de las diferentes aproximaciones avanzadas en el desarrollo de *software,* como lo son la programación orientada por aspectos, las líneas de productos de *software* y los lenguajes específicos de dominio, entre otros.

REFERENCIAS

AMBLER, Scott W. Agile model driven development is good enough. En: IEEE Software, vol. 20, no 5: (Sep.-Oct., 2003), p. 71-73.

ASPECT-ORIENTED SOFTWARE ASSOCIATION (AOSA). Aspect-oriented software development community & conference. [Documento Electrónico]. AOSA, 2006. (Citada: 5 agosto 2006) http://aosd.net/

ATKINSON, Colin and KÜHNE, Thomas. Model-driven development: A metamodeling foundation. En: IEEE Software, vol. 20, no 5: (Sep.-Oct., 2003), p. 36-41.

BÉZIVIN, Jean. In search of a basic principle for model driven engineering. En: Upgrade, vol. V, no. 2, (April 2004), p. 21-24. ISSN 1684-5285.

BOOCH, G., BROWN A. and RUMBAUGH, J. An MDA manifesto. IBM Rational Software, 2004.

CERNUDA, Agustín. Sistema de verificación de componentes software. Departamento de Informática, Universidad de Oviedo, 2002. 230 p.

[13] PDM: Platform Description Model.

[14] MDE: Model Driven Engineering.

[15] MDSD: Model Driven Software Development.

FERNÁNDEZ-MEDINA, Eduardo; TOVAL, Ambrosio y PIATTINI, Mario. Lenguaje de especificación de restricciones de seguridad: OSCL V 1.1. Dolmen, 2001. 9 p.

FOWLER, Martin. Domain specific language. [Documento electrónico] MartinFowler.com, 2006. (Citada: 22 agosto 2006) http://www.martinfowler.com/bliki/DomainSpecificLanguage.html

FUENTES, Lidia y VALLECILLO, Antonio. Una introducción a los perfiles UML. Málaga: Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, 2004. 13 p.

HAYWOOD, Dan. MDA: Nice idea. [Documento electrónico]. The Server Side, 2004. (Citada: 4 septiembre 2006). http://www.theserverside.com/tt/articles/article.tss?l=MDA_Haywood

HUMAN PERFORMANCE CENTER (HPC). Software architectura, Technology Area. [Documento electrónico]. HPC Spider, 2002. (Citada: 4 agosto 2006) https://www.spider.hpc.navy.mil/index.cfm?RID=TTE_OT_1000025

IYENGAR, Sridhar. XML Metadata interchange (XMI) from UML object models to XML DTDs and Documents. s.l.: Unisys Corporation, 1999. 45 p.

JACOBSON, Ivar; GRISS, Martin and JONSSON, Patrik. Software reuse: Arquitecture, process and organization for business success. Addison Wesley, 1997.

JÉZÉQUEL, Jean-Marc. Model transformation techniques. [Documento electrónico]. Universidad de Rennes, 2005. (Citada: 15 agosto 2006) http://www.irisa.fr/prive/jezequel/enseignement/ModelTransfo.pdf

KLEPPE, Anneke and WARMER, Jos. The Object Constraint Language: precise modeling with UML. s.l.: Addison Wesley, 1999. ISBN 0 201 37940 6.

KLEPPE, Anneke; WARMER, Jos and BAST, Win. MDA explained: The practice and promise of model-driven architecture. Addison-Wesley, 2003.

MELLOR, Stephen J.; CLARK, Anthony N. and FUTAGAMI Takao. Model-driven development. En: IEEE Software, vol. 20, no 5: (Sep.-Oct., 2003), p. 14-18.

MUÑOZ, Javier y PELECHADO, Vicente. MDA a debate. En: Taller sobre Desarrollo Dirigido por Modelos, MDA y Aplicaciones de las IX Jornadas de Ingeniería del Software y Bases de Datos. (Málaga, España: noviembre 2004).

OMG-BP, 2004. OBJECT MANAGEMENT GROUP. Business processes and the OMG: An overview. [Documento electrónico]. (Citada: 8 agosto 2006) http://www.omg.org/bp corner/bp files/The_OMG_BP_Corner_INTRO_Paper_3-2-04.pdf

OMG-MDA, 2003. OBJECT MANAGEMENT GROUP. Model driven architecture (MDA) Guide Version 1.0.1 [Documento electrónico]. (Citada: 8 junio 2006) http://www.omg.org/docs/omg/03-06-01.pdf

OMG-MOF, 2003. OBJECT MANAGEMENT GROUP. A review of OMG MOF 2.0 Query/Views/Transformations submissions and recommendations towards the final standard. [Documento electrónico]. (Citada: 8 junio 2006) http://www.omg.org/docs/ad/03-08-02.pdf

OMG-RfPOCL, 2003. OBJECT MANAGEMENT GROUP. Response to the UML 2.0 OCL RfP (ad/2000-09-03) [Documento electrónico]. OMG, 2003. (Citada: 8 junio 2006) http://www.omg.org/docs/ad/03-01-07.pdf

OMG-OCL, 2003. OBJECT MANAGEMENT GROUP. UML 2.0 OCL specification. [Documento electrónico]. (Citada: 9 agosto 2006) http://www.omg.org/docs/ptc/03-10-14.pdf

OMG-UML, 2003. OBJECT MANAGEMENT GROUP. UML 2.0 superstructure specification. [Documento electrónico]. (Citada: 10 agosto 2006) http://www.omg.org/docs/formal/05-07-04.pdf

OMG-UAS, 2002. OBJECT MANAGEMENT GROUP. Unified modeling language specification (Action Semantics). [Documento Electrónico]. (Citada: 22 agosto 2006) http://www.omg.org/docs/ptc/02-01-09.pdf

OMG-XMI, 2003. OBJECT MANAGEMENT GROUP. XML metadata interchange (XMI) specification. [Documento electrónico]. (Citada: 10 mayo 2006) http://www.omg.org/docs/formal/03-05-02.pdf

SEI-CMMI, 2002. SOFTWARE ENGINEERING INSTITUTE. Capability maturity model integration (CMMI), Version 1.1. Carnegie Mellon University, 2002. 725 p.

SEI-SPL, 2006. THE SOFTWARE ENGINEERING INSTITUTE. Software product line. [Documento electrónico]. SEI, 2006. (Citada: 5 agosto 2006) http://www.sei.cmu.edu/productlines/index.html

SHEKHAR, C. and PRABHAKAR, T. V. An overview of UML 2.0. En: Technical Meeting (Paris, June 2003).

VILLARROEL R.; FERNÁNDEZ-MEDINA, E.; TRUJILLO, J. y PIATTINI, M. Un profile de UML para diseñar almacenes de datos seguros. CYTED, 2003. 9 p.

VÖLTER, M. y STAHL, T. Model-driven software development. Technology, Engineering, Management (Julio 2006), p. 444. ISBN: 978-0-470-02570-3

WARMER, Jos. The role of OCL in the model driven architecture. Springer, Berlin and Heidelberg 2002. 202 p.