Un Análisis Crítico de la Aproximación Model-Driven Architecture

UNIVERSIDAD COMPLUTENSE
MADRID

PROYECTO FIN DE MÁSTER EN SISTEMAS INTELIGENTES

Autor: **Pedro Antonio Fernández Sáez**

Director: Antonio Navarro Martín

Máster en Investigación en Informática
Facultad de Informática
Universidad Complutense de Madrid

Curso académico: **2008/2009**

Un Análisis Crítico de la Aproximación Model-Driven Architecture

PROYECTO FIN DE MÁSTER EN SISTEMAS INTELIGENTES

Autor: **Pedro Antonio Fernández Sáez**

Director: **Antonio Navarro Martín**

Máster en Investigación en Informática
Facultad de Informática
Universidad Complutense de Madrid

Curso académico: **2008/2009**

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "UN ANÁLISIS CRÍTICO DE LA APROXIMACIÓN MODEL DRIVEN ARCHITECTURE", realizado durante el curso académico 2008-2009 bajo la dirección de Antonio Navarro Martín en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Madrid, a 14 de septiembre de 2009

Pedro Antonio Fernández Sáez

A Carla, Pedro y David

Abstract

Model Driven Architecture (MDA) emerged as a new approach for software development almost a decade ago, sufficient time to be able to make an analysis of its importance as much in terms of its own evolution, as in its acceptance by the developers' community and the industry. With this objective, a study of the state-of-the-art MDA approach has been made from different points of view, which have allowed the extraction of interesting conclusions.

The study starts with the origins of MDA, where a revision of its concepts is made and the advantages it provides over traditional software development are shown. Likewise, a special emphasis is made on the four-layer architecture defined by the OMG workgroup. Some of the standards defined by the OMG in the context of MDA are studied, detailing their objectives, structure and definition, situating each in the four-layer metamodeling architecture defined in MDA. Another important element is the study of de facto standards defined within the industry. Specifically, the Eclipse standards related to the Eclipse Modeling Project (EMP) are studied. An attempt has been made to understand why Eclipse has decided to define and implement its own standards and a comparison between the OMG and the Eclipse standards has been given.

Another point of view developed was the study of certain tools on the basis of a set of specific characteristics of the MDA approach. The purpose of this is to see to what degree these tools implement these characteristics. The tools studied were: IBM Rational Software Architect, Borland Together and Sparx Systems Enterprise Architect. Finally, a study of the MDA approach's impact and repercussions in the industry has been undertaken, starting with the research efforts that the industry is making into the MDA approach. To that end, a study has been carried out of the papers presented in one of the international conferences devoted to the MDA approach: the MoDELS conference. The papers studied are those given or involving authors from the world of industry, as opposed to those given by authors from an academic environment.

The summary of the conclusions of this study on the success of the MDA approach is not emphatic in regard to its use or implementation. Despite the effort realised by the Eclipse community to simplify and develop specific standards for the industry, the result observed was not very significant.

Key words: Model Driven Architecture (MDA), Model Driven Development (MDD), Object Management Group (OMG), Eclipse Modeling Project (EMP), CASE Tools, Rational Software Architect (RSA), Together, Enterprise Architect (EA), industry, MoDELS conference.

Resumen

Model Driven Architecture (MDA) surgió hace casi una década cómo una nueva aproximación al desarrollo de sistemas software, tiempo suficiente para poder hacer un análisis sobre el alcance de la misma tanto en su propia evolución, como en la aceptación que haya causado en la comunidad de desarrolladores y en la industria. Con este objetivo se ha hecho un estudio sobre el estado del arte de la aproximación MDA desde diferentes puntos de vista, que han permitido extraer interesantes conclusiones.

El estudio parte de los orígenes de MDA, donde se hace un repaso de sus conceptos y se indican las ventajas que proporciona frente al desarrollo de software tradicional. Así mismo, se hace un especial hincapié en la arquitectura de metamodelado de cuatro capas definidas por el grupo de trabajo OMG. Se estudian algunos de los estándares definidos por OMG en el contexto de MDA detallando sus objetivos, su estructura y su definición, situando cada uno de ellos en la arquitectura de cuatro capas definida en MDA. Otro elemento importante a tener en cuenta es el estudio de estándares de facto definidos con la industria. En concreto se estudian los estándares Eclipse relacionados con el proyecto *Eclipse Modeling Project* (EMP). Se ha intentado comprender por qué Eclipse ha decidido definir e implementar sus propios estándares y se ha indicado la equiparación entre los estándares OMG y los de Eclipse.

Otro punto de vista desarrollado ha sido el estudio de determinadas herramientas, sobre la base de un conjunto de características específicas de la aproximación MDA. La finalidad de esto es ver en qué grado estas herramientas implementan estas características. Las herramientas estudiadas han sido: *IBM Rational Software Architect, Borland Together* y *Sparx Systems Enterprise Architect*. Finalmente, se ha realizado un estudio del impacto y calado en la industria, partiendo del esfuerzo investigador que hace ésta en la aproximación MDA. Para ello, se ha realizado un estudio de las ponencias presentadas en una de las conferencias internacionales centradas en la aproximación MDA: la conferencia MoDELS. Se han estudiado las ponencias realizadas o participadas por autores provenientes del mundo de la industria, en contraposición a las realizadas por autores provenientes de un entorno académico.

El resumen de las conclusiones de este estudio, apunta a que el éxito de la aproximación MDA, no ha sido rotundo en cuanto a su utilización e implantación. A pesar del esfuerzo realizado por la comunidad Eclipse para simplificar y desarrollar estándares específicos para la industria, se observa que el resultado no ha sido especialmente significativo.

Palabras clave: *Model Driven Architecture* (MDA), *Model Driven Development* (MDD), *Object Management Group* (OMG), *Eclipse Modeling Project* (EMP), Herramientas CASE, *Rational Software Architect* (RSA), *Together*, *Enterprise Architect* (EA), industria, conferencia MoDELS.

Agradecimientos

Me gustaría en primer lugar, agradecer a Antonio Navarro Martín el tiempo que me ha dedicado desde el primer día que le planteé la posibilidad de realizar un trabajo dentro del ámbito de la Ingeniería del Software. Por entender mi situación personal y poner todos los medios posibles a su alcance para facilitarme la difícil tarea de compaginar los estudios con mi vida laboral, tanto en la asignatura de Requisitos del Software como en la dirección de este trabajo. Gracias por darme la idea, por atender todas mis consultas, adaptarte a mi difícil horario laboral y sobre todo por escuchar atentamente mis problemas y darme consejos incluso de aspecto estrictamente profesionales. Agradecer también a Narciso Martí Oliet el que me hubiera animado a no dejar escapar la oportunidad de estudiar el Máster en Investigación en Informática, a pesar del tiempo transcurrido desde que acabara mis estudios de Ingeniería Técnica en Informática de Gestión.

Por supuesto, no puedo dejar de agradecer la infinita paciencia de mi mujer Carla y de mis hijos Pedro y David, por el tiempo que he dejado de dedicarles desde que hace tres años me reenganché a la vida de estudiante. Vida más apasionante si cabe, cuando se observa desde el punto de vista que te aporta una larga experiencia profesional y unos cuantos años más a tus espaldas. El esfuerzo ha sido enorme, más si cabe porque es difícil exigirle a tus hijos un rendimiento máximo en las notas y que las tuyas puedan llegar a ser mediocres. Gracias por estar ahí, por apoyarme en los momentos de sobrecarga de trabajo y por exigirme cuando bajaba el ritmo.

Por último, agradecer a la dirección de la empresa GFI Informática, en general y a mis responsables directos en particular, tanto a Pilar Rubio como a Ubaldo Serna, por haberme permitido compaginar mis responsabilidades en la empresa con las de la realización del máster y de este trabajo.

Hay muchas más personas que directa o indirectamente me han ayudado, animado o han servido de apoyo psicológico en estos años de esfuerzo y dedicación. Mis padres, mis hermanos, mis amigos y mis compañeros de trabajo. Una extensa lista de personas, que no detallo por miedo a cometer una injusticia con alguno de ellos, al no nombrarles.

Índice general

Índice de figuras

Índice de tablas

Definiciones y Acrónimos

- ***Aplicación***: funcionalidad que se está desarrollando. Un sistema se define en términos de una o varias aplicaciones soportadas en una o varias plataformas.
- ***Arquitectura de un sistema***: especificación de las partes y conexiones de un sistema y de las reglas de interacción entre sus partes.
- ***Eclipse***: Fundación que lidera el desarrollo de la plataforma de código abierto para el desarrollo de aplicaciones en Java que lleva el mismo nombre.
- ***Implementación***: es la especificación que provee toda la información necesaria para la construcción del sistema y su puesta en funcionamiento.
- ***Independiente de la plataforma***: cualidad que un modelo puede exhibir.
- ***Interoperabilidad***: Capacidad de sistemas software diferentes de intercambiar datos y posibilitar la puesta en común de información y conocimientos.
- ***CASE*** (*Computer Aided Software Engineering*): Ingeniería de Software Asistida por Ordenador.
- ***Data warehouse***: colección de datos orientada a un determinado ámbito (empresa, organización, etc.), integrado y variable en el tiempo, que ayuda a la toma de decisiones.
- ***Dirigido por modelos*** *(model-driven)*: aproximación que incrementa el poder de los modelos.
- ***EC-MDA***: *European Conference on Model Driven Architecture Foundations and Applications.*
- ***ICSE***: *International Conference of Software Engeenering*
- ***JISBD***: Jornadas de Ingeniería del Software y Bases de Datos.
- ***MDA*** (*Model Driven Architecture*): Arquitectura dirigida por modelos.
- ***MDD*** (*Model Driven Development*): Desarrollo dirigido por modelos.
- ***Metamodelo***: definición precisa de los constructores y reglas necesarios para definir la semántica de los modelos. Define un lenguaje de modelado para modelos.
- ***Meta-metamodelo***: definición de un lenguaje de modelado para metamodelos.
- ***Modelo***: descripción o especificación de un sistema realizado con un lenguaje determinado. Abstracción semánticamente completa de un sistema. Representación abstracta de un sistema.
- ***Modelo de comportamiento***: modelo que representa un sistema desde un punto de vista dinámico.
- ***Modelo de plataforma***: proporciona un conjunto de conceptos técnicos, representando las diferentes clases o partes que forman una plataforma y sus servicios.
- ***Modelo de plataforma específica*** (PSM – *Platform Specific Model*): vista del sistema desde un punto de vista específico de la plataforma.
- ***Modelo estructural***: modelo que representa aspectos estáticos de un sistema.
- ***Modelo independiente del cómputo*** (CIM – *Computation Independent Model*): vista del sistema desde un punto de vista independiente del cómputo. No muestra detalles de la estructura del sistema.
- ***Modelo independiente de la plataforma*** (PIM – *Platform Independent Model*): vista del sistema desde un punto de vista independiente de la plataforma.

- ***MoDELS***: *International Conference on Model Driven Engineering Languages and Systems*.
- ***OLAP*** (*On-Line Analytical Processing*): Procesamiento analítico en línea. Solución utilizada en el ámbito del *data warehouse*, para la consulta de grandes cantidades de datos.
- ***OMG:*** *Object Management Group.*
- ***Patrón de diseño***: solución de diseño a problemas comunes en el desarrollo de software, que permita aplicarse de forma repetitiva en diferentes circunstancias.
- ***Perfil***: mecanismo para adaptar un metamodelo existente con constructores de una metodología, dominio o plataforma específica.
- ***Plataforma***: conjunto de subsistemas y tecnologías que proveen un conjunto coherente de funcionalidades mediante interfaces y patrones.
- ***Punto de vista***: técnica de abstracción usando un conjunto selecto de conceptos de arquitectura y reglas de estructura focalizados en conceptos específicos del mismo.
- ***Puntos de vista MDA****:*
  - Independiente del cómputo: focalizado en el entorno del sistema y sus requerimientos
  - Independiente de la plataforma: focalizado en la operación del sistema ocultando los detalles específicos de la plataforma.
  - De la plataforma: combina el anterior con los detalles específicos de la plataforma.
- ***Sistema***: conjunto de programas, computadores, combinaciones de partes de diferentes sistemas, personas, empresas, etc.
- ***Transformación de Modelos***: proceso que convierte un modelo en otro del mismo sistema, mediante un conjunto de reglas.
- ***Vista***: representación de un sistema desde la perspectiva dada por un punto de vista.

1. INTRODUCCIÓN

Model Driven Architecture (MDA) [OMG 2003a][Mellor 2004] es una aproximación definida por el *Object Management Group* (OMG) [OMG], mediante la cual el diseño de los sistemas se orienta a modelos. En ocasiones, el término MDA se intercambia con el de MDD (*Model-Driven Development*). MDD se refiere a las actividades que llevan a cabo los desarrolladores, mientras que MDA se refiere a su definición formal. Definición creada por el grupo de trabajo OMG, que se centra en la creación de un marco de trabajo formal, en el que puede operar MDD [Gardner 2006]. A pesar de estas sutiles diferencias, ambos términos se utilizan de manera indistinta en este trabajo.

El desarrollo orientado a modelos permite una alta flexibilidad en la implementación, integración, mantenimiento, prueba y simulación de los sistemas. Una de las ideas principales por la que surge MDA es separar la especificación de los sistemas de los detalles de su implementación en una determinada plataforma. MDA provee un conjunto de herramientas para especificar un sistema independientemente de la plataforma de implementación, especificar dichas plataformas, elegir una determinada plataforma para el sistema, y transformar las especificaciones de los sistemas a la plataforma elegida. Todo esto se complementa con los objetivos de portabilidad, interoperabilidad y reusabilidad.

La independencia propuesta por MDA se consigue mediante una catalogación de modelos que permiten especificar el sistema desde diferentes puntos de vista. Los tipos más destacables de modelos son los siguientes:

- *Computational Independent Model* (CIM): son visiones de los sistemas desde el punto de vista del problema a resolver, es decir, un modelo simplificado que se abstrae de detalles específicos.
- *Platform Independent Model* (PIM): muestra una vista del diseño del sistema obviando detalles de plataformas concretas.
- *Platform Specific Model* (PSM): muestra un diseño del sistema incluyendo detalles específicos de la plataforma.

Asimismo, dentro de la aproximación MDA, tiene especial relevancia la existencia de transformaciones entre modelos. De esta forma, se definen los mecanismos necesarios para convertir un modelo de un tipo a otro, siendo ambos representaciones del mismo sistema. Se puede transformar por ejemplo, un modelo PIM en uno o varios modelos PSM.

OMG ha definido un conjunto de estándares de acuerdo con la aproximación MDA. Se pueden destacar algunos de los más relevantes: UML (*Unified Model Language*) [OMG UML], MOF (*Meta Object Facility*) [OMG 2006a], QVT (*Query View Transformation*) [OMG 2008a], OCL (*Object Constraint Language*) [OMG 2006b] o XMI (*XML Metadata Interchange*) [OMG 2007c], pero existen muchos estándares muy extensos y complejos.

En la actualidad, ha pasado casi una década desde la presentación de MDA como una propuesta para la especificación y el desarrollo de sistemas software, lo que puede situarnos en un momento oportuno para hacer un análisis sobre esta aproximación. En [CEPIS 2008] se presenta un monográfico sobre MDA, donde se hace un repaso de los objetivos iniciales y futuros, los objetivos alcanzados y pendientes de cumplir, una visión retrospectiva del camino andado y las dificultades encontradas, trabajos que hay actualmente en marcha, estrategias de investigación, así como el trabajo pendiente para cumplir los objetivos iniciales. Existe también un estudio realizado mediante encuesta [Forward 2008], cuyo objetivo es identificar las actitudes y experiencias frente al desarrollo dirigido por modelos y los argumentos que utilizaban los desarrolladores para seguir utilizando un desarrollo basado en el código, en vez de en modelos.

Estos trabajos son muy interesantes, pero no son capaces de darnos una idea global del estado actual de la aproximación MDA. En efecto, desde el estudio de sus estándares, la versión industrial

de éstos, el soporte que le dan las herramientas CASE y la repercusión de MDA en la industria, se puede obtener una visión más completa de esta aproximación. Precisamente, este trabajo pretende revisar los retos y objetivos alcanzados hasta ahora, hacer un repaso de la situación actual y realizar un análisis de la repercusión real que ha tenido esta aproximación en el desarrollo de sistemas software. En definitiva, ver si el gran esfuerzo realizado en la estandarización e investigación sobre MDA, ha repercutido en la comunidad de desarrolladores. Se intentará medir el grado de aceptación y utilización en el diseño y desarrollo de sistemas reales implantados, más allá de prototipos o estudios teóricos.

Existen diferentes puntos de vista en los que fijarse para abordar este análisis. Por un lado habría que estudiar y conocer a fondo los conceptos y la arquitectura definida por la aproximación MDA, así como los principales estándares definidos por OMG. El estudio de los principales estándares puede aportar un interesante punto de vista teórico sobre la amplitud y complejidad de su implementación. Otro elemento de medida podría ser realizar un estudio sobre las herramientas existentes en el mercado que den soporte a la aproximación MDA. Se puede estudiar si abarca completa o parcialmente los estándares, qué tipo de transformaciones realiza, qué calidad de código genera, etc. Sería también un elemento de medida interesante, el impacto que tiene MDA en comunidades de desarrollo de código *open source*, viendo si se desarrollan herramientas que implementan los estándares definidos. Un caso representativo es el de Eclipse que creó el proyecto *Eclipse Modeling Project* [Eclipse EMP], que se centra en la evolución y promoción de tecnologías de desarrollo basadas en modelos.

Otro análisis relevante sería ver el impacto que tiene MDA en la industria, ya que es precisamente la industria diseña y desarrolla la mayoría de los sistemas software. Un elemento de medida para este objetivo podría ser observar y estudiar qué presencia hay de la industria en las conferencias relacionadas con MDA. Que la industria se dedique a escribir ponencias de investigación para determinadas conferencias puede llevarnos a sacar la conclusión de que invierte en ese esfuerzo porque puede sacarle rentabilidad. La investigación, no se ha quedado atrás en este sentido y existe un gran volumen de publicaciones y estudios sobre MDA que aportan conocimiento y experiencias de aplicación práctica de la misma. No obstante, se observa que aparentemente predominan los autores del entorno académico, existiendo en proporción, poca presencia de la industria como impulsores de estos estudios. Esto queda de manifiesto en las conferencias cuyo eje central es el desarrollo dirigido por modelos. Se observa que la mayoría de las ponencias se presentan por autores de entornos académicos. Un ejemplo de ellas es la conferencia MoDELS [MoDELS] que se realiza de de forma anual y se centra en la ingeniería dirigida por modelos impulsada por MDA. Las conferencias MoDELS son un punto de encuentro para el intercambio y la innovación tecnológica entorno a ideas y experiencias relativas a la arquitectura MDA en desarrollo de sistemas software. Por esto es interesante repasar las ponencias de los últimos años analizando la presencia de la industria en las mismas, así como las características de las empresas que participan, con el fin de poder extraer conclusiones.

La estructura de este trabajo pretende ordenar, repasar y referenciar todos estos elementos, de manera que se puedan obtener conclusiones sobre la repercusión de MDA, en particular en el ámbito industrial. En el capítulo 2 se hace una presentación sobre la aproximación MDA, sus orígenes, qué problemas resuelve y cuáles son sus conceptos básicos y su arquitectura. En el capítulo 3 se realiza un completo estudio de los principales estándares definidos por OMG. El capítulo 4 se centra en el estudio del proyecto *Eclipse Modeling Project*, cuyo objetivo es la evolución y promoción de tecnologías de desarrollo basadas en modelos. A continuación, en el capítulo 5, se analizan algunas de las principales herramientas comerciales basadas en el desarrollo dirigido por modelos. Las herramientas estudiadas son: *IBM Rational Software Architect, Borland Together*, y *Sparx Systems Enterprise Architect*. En el capítulo 6, se hace un estudio de la presencia de la industria en una de las conferencias del ámbito de MDA. En concreto se ha elegido la conferencia MoDELS. Finalmente, en el capítulo 7 se exponen las conclusiones de este análisis crítico sobre la aproximación MDA.

Aportaciones principales de este trabajo

Aunque resulta poco ortodoxo incluir una sección con aportaciones dentro de una introducción, la experiencia demuestra que facilita la revisión y evaluación práctica de este tipo de trabajos.

Este trabajo presenta cuatro grandes aportaciones, coherentes con la naturaleza de un estado del arte:

- Recopila en un mismo documento una gran cantidad de información, que se encuentra dispersa, sobre la aproximación MDA. De esta forma, sirve como punto de partida para cualquier persona que quiera iniciarse en dicha aproximación. Esta información incluye: estándares OMG, sus homólogos industriales, como los que están en el contexto del proyecto Eclipse, y el soporte CASE de la aproximación MDA.
- Contextualiza la información presentada en términos de la jerarquía de metamodelos en los cuatro niveles de OMG, facilitando así su asimilación [OMG 2007a]. Además, compara la relación existente entre los estándares OMG y sus equivalentes industriales.
- Analiza la repercusión de la aproximación MDA en la industria.
- Proporciona una visión crítica de la aproximación MDA en base a la información procesada y analizada en el trabajo.

Este trabajo también puede tener repercusiones prácticas en proyectos reales de informática, ya que debido a la situación de crisis que vivimos, y por los estímulos que provienen del gobierno, las empresas buscan nuevas tecnologías para aplicar a sus proyectos y optimizar su productividad. De hecho, una de las principales consultoras ya ha mostrado su interés por el mismo.

Además, sus aportaciones son fundamentales para proyectos de investigación como el proyecto *Integración de Plataformas y Servicios en el Campus Virtual* (IPS-CV) (TIN2008-06708-C03-01/TSI) o el proyecto *Arquitecturas Avanzadas en Campus Virtuales* (AACV) (TIN2009-14317-C03-01). Desde la experiencia en el desarrollo, operación y mantenimiento de campus virtuales de grandes universidades se concluye que éstos evolucionan hacia complejas aplicaciones web que necesitan resolver problemas de integración de plataformas y servicios, mantenibilidad, interoperabilidad y autoría de contenidos y calidad. El objetivo global de estos proyectos es investigar, desarrollar y perfeccionar la infraestructura tecnológica necesaria para el funcionamiento eficiente de un campus virtual universitario que aúne investigación y docencia. En concreto, pretenden desarrollar una investigación en ingeniería web, centrada en la integración de aplicaciones web de dimensión industrial según diversas arquitecturas (multicapa, de integración y orientadas a servicios). Además, debido a la constante evolución a la que están expuestos los campus virtuales, es necesario investigar sobre los mecanismos que faciliten el modelado de estas aplicaciones y su transición a código, tales como la aproximación MDA. Por lo tanto, dicha investigación en ingeniería web comprende investigación en ingeniería del software y sistemas de información y en sistemas distribuidos. Esta investigación va a ser aplicada a la construcción de campus virtuales que permitan la integración y prueba de plataformas de gestión de cursos (históricos, en uso y en prueba), y de herramientas y servicios complementarios (por ejemplo autoría y gestión de contenidos, intercambio de datos con el resto de sistemas de la universidad, gestión de espacios virtuales de trabajo, etc.).

2. MDA - *MODEL DRIVEN ARCHITECTURE*

2.1 *Object Management Group* (OMG)

El *Object Management Group* [OMG] es un consorcio internacional abierto y sin ánimo de lucro de la industria de la informática creada en 1989. En él están incluidas cientos de organizaciones dedicadas al desarrollo de software para usuarios finales, especializados en docenas de mercados de negocio diferentes y organizaciones dedicadas a la industria de los computadores.

En sus orígenes [Watson 2008], OMG se constituyó para especificar elementos *middleware* que ayudaran a resolver los problemas de integración entre sistemas de información multiplataforma. Con esta idea surgió la especificación de CORBA (*Common Object Request Broker Architecture*) [CORBA] y otros estándares como el *Interface Definition Language* (IDL) [OMG 2002], que permite definir las interfaces de los servicios que se van a utilizar en la comunicación entre las distintas plataformas a integrar. Hace casi una década, OMG decidió dar el salto desde una aproximación basada en servicios de integración a una aproximación independiente de la plataforma, que definiera situaciones y aspectos estructurales de la interoperabilidad mediante modelos. Estos modelos independientes de la plataforma podrían ser convertidos, mediante reglas de transformación, a cualquier plataforma específica. Un poco antes, a mediados de los años 90, basándose en un incipiente modelado orientado a objetos, OMG crea el lenguaje de modelado Unified Modeling Language (UML) [OMG UML] que permite representar aspectos dinámicos y estáticos de un sistema de información software. Aplicando UML y MOF (Meta Object Facility) [OMG 2006a], el metamodelo que define UML, al problema de la creación de modelos independientes de la plataforma, permite la creación de la aproximación MDA.

OMG desarrolla estándares de integración para un amplio rango de tecnologías: tiempo real, sistemas empotrados, entre otros y una amplia variedad de industrias diferentes: modelado e integración de negocios, finanzas, administración pública, salud, etc.

Los estándares de modelado de OMG facilitan un diseño visual, un desarrollo y un mantenimiento de software potentes. Toda la documentación y los estándares definidos por OMG se encuentran disponibles al público en general.

OMG también toma la iniciativa y promueve conferencias y eventos internacionales sobre los avances en las investigaciones sobre esta disciplina y la experiencia de la aplicación de los estándares definidos en la industria. Un ejemplo de ellas son las conferencias MoDELS [MoDELS] que congregan profesionales e investigadores para presentar y discutir tanto la experiencia práctica, como técnicas innovadoras en relación a MDA. Sobre la conferencia MoDELS se hablará más en profundidad en el capítulo 6.

2.2 *Model-Driven Architecture* (MDA)

2.2.1 Desarrollo de Software Tradicional

El desarrollo de sistemas software siempre ha sido una labor intensa, pero a medida que ha ido evolucionando la tecnología, esta labor se ha complicado cada vez más. La evolución de los lenguajes, entornos y técnicas de programación provoca que haya que desarrollar los mismos sistemas una y otra vez, que éstos utilicen e integren diferentes tecnologías o que exista una necesidad de comunicación entre sistemas dispares. A todo esto hay que añadirle los continuos

cambios en los requerimientos, ya sean impuestos por los usuarios de los sistemas o derivados de los cambios tecnológicos [Schach 2005].

El enfoque tradicional del desarrollo de software no es capaz de absorber toda esta casuística de una forma eficiente, por problemas que vienen derivadas de su propio planteamiento, que se resumirán a continuación.

Uno de los principales problemas es la *productividad*. Los desarrollos como los conocemos están dirigidos por un diseño y una codificación a bajo nivel. El proceso típico de un desarrollo de software, tal y como se muestra en la Figura 2-1 ([Kleppe 2005]), incluye las siguientes fases:

- Conceptualización y toma de requisitos
- Análisis y descripción funcional
- Diseño
- Codificación
- Pruebas
- Implantación

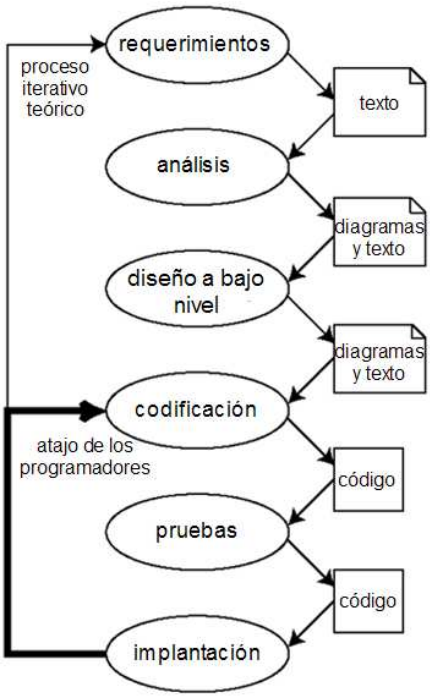

Figura 2-1. Ciclo de vida del desarrollo de software tradicional

Como se expresa en la figura, las tres primeras fases se centran en generar la documentación y los diagramas que definen el sistema, e incluso en muchos casos, se utiliza UML [OMG UML] para definir los casos de uso, diagramas de clases, de interacción o de actividades. Por el contrario, las tres últimas fases se centran básicamente en la codificación, utilizando como punto de partida la documentación generada previamente. En el momento de iniciar la codificación del software, toda la documentación generada hasta el momento, la utilizan los programadores para poder iniciar su trabajo. Incluso, en el caso de utilizar UML en la fase de análisis, muchas herramientas CASE generan código a partir de los diagramas.

El desarrollo tradicional de software trata las fases del ciclo de vida como fases independientes y completas. [Schach 2005], explica gráficamente que si un sistema es un modelo de la realidad, si

ésta cambia, el sistema debe cambiar, por lo que los requerimientos pueden modificarse constantemente. Este y otros factores como los errores que se puedan producir por el equipo de desarrollo en las diferentes fases del ciclo de vida, lo que provocan el que haya un proceso iterativo que obligue a volver a la fase de requisitos para volver a revisar todas las fases anteriores. Las iteraciones deben ser completas por muchas razones, pero la principal es el mantenimiento de la documentación, ya que ésta es uno de los valores fundamentales de los sistemas. Un sistema no es sólo el código.

Lo que puede ocurrir en ocasiones, es que la documentación se quede desactualizada y por lo tanto, la conexión que pueda existir entre la documentación y la codificación, se puede perder de forma rápida [Kleppe 2005]. Resulta tentador para los programadores resolver los problemas o los cambios de requisitos, incidiendo directamente en el código, en vez de partiendo de los diagramas y la documentación funcional. Probablemente, se termine por modificar la documentación, pero esto suele ocurrir una vez finalizado el desarrollo y su validez puede ser cuestionable. Por lo tanto, cabría preguntarse si vale la pena perder un tiempo precioso en una definición a alto nivel de la especificación del sistema, una vez iniciada la codificación.

Evidentemente, la respuesta negativa sería impensable, ya que incumpliría todos los principios de modelado de la ingeniería del software, pero en los casos en los que no se actúe conforme a las metodologías definidas, se podría llegar a una conclusión de que se dedica mucho tiempo en mantener y que por lo tanto ver como más productivo la codificación.

A pesar de que exista la tentación de plantearse esta pregunta, lo que no se puede negar es que, en proyectos de desarrollo, ambas tareas son necesarias y tienen su importancia, por lo que hay que se plantea la necesidad de encontrar un mecanismo que facilite la generación y el mantenimiento de la documentación de los sistemas software.

Con el desarrollo de software tradicional se plantea también el problema de la *portabilidad*. La tecnología avanza y los sistemas se ven obligados a adaptarse a los cambios tecnológicos por muy diversas razones:

- Los clientes demandan las nuevas tecnologías.
- Las nuevas tecnologías solucionan problemas que se vienen detectando en los sistemas actuales.
- Los proveedores descatalogan los productos, dejando de dar soporte a las tecnologías obsoletas.

En estos casos los desarrolladores se deben adaptar a las nuevas tecnologías, de forma rápida, quedando desperdiciado el tiempo dedicado a la investigación de la tecnología anterior. La situación es un tanto más compleja, ya que no es necesario cambiar de tecnología, en una misma tecnología aparecen asiduamente nuevas versiones, sobre las que habitualmente sólo se da soporte a las más recientes.

En el caso de que no se evolucione el software heredado a las nuevas tecnologías, sin duda aparecerá el problema de la *interoperabilidad*, ya que sistemas con tecnologías diferentes deben comunicarse y entenderse para el buen funcionamiento de los negocios.

El ejemplo típico en los últimos años es el de los sistemas web que tienen un *front-end* que se ejecuta sobre un navegador (usando tecnologías como HTML, ASP, JSP, etc) y que necesita integrarse con un sistema *back-end* basados en tecnologías *mainframe* o que acceden a grandes bases de datos.

2.2.2 Aproximación MDA

La arquitectura dirigida por modelos (MDA) [Mellor 2004] es una aproximación para el desarrollo de software definido por el OMG. La clave de MDA es la importancia que le da a los modelos en el los procesos de desarrollo.

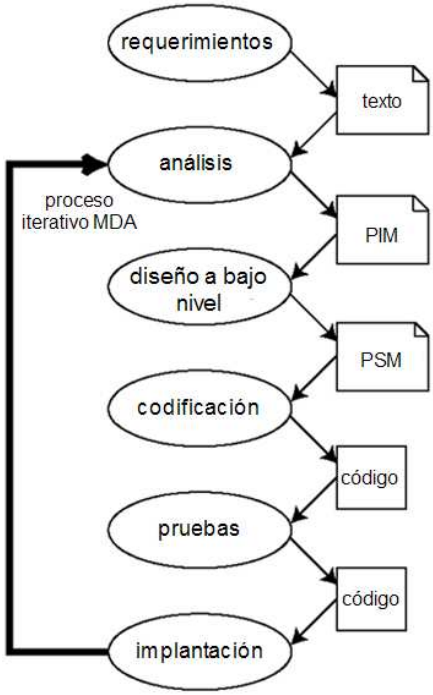

Figura 2-2. Ciclo de vida del desarrollo de Software MDA

Se puede observar en la Figura 2-2 ([Kleppe 2005]), que el ciclo de vida de los desarrollos MDA, no es muy diferente del visto en la sección anterior. En realidad, el enfoque de Kleppe se centra en las fases de desarrollo, pero en según el proceso unificado de desarrollo [Jacobson 2001], el ciclo de vida del desarrollo de software se basa en una combinación de incrementos de estas fases. Como ya se ha comentado, se pueden cometer errores en cada una de estas, por lo que es conveniente detectarlos de forma temprana para evitar costes y desviaciones posteriores. Para facilitar esta tarea, lo ideal es abordar los diferentes componentes de un sistema de información (o artefactos) de forma incremental, de manera que en cada incremento se van a realizar todas las fases del ciclo de vida en mayor o menor medida. Cada incremento, será como un pequeño proyecto en el que se ejecutarán las fases de requisitos, análisis, diseño implementación y pruebas. Dentro de cada una de ellas, existirán las iteraciones necesarias para revisar cada artefacto hasta que se complete el incremento y se pueda continuar con el siguiente.

Los incrementos que fija el proceso unificado de desarrollo son: iniciación, elaboración, construcción y transición. La Figura 2-3 muestra gráficamente el desglose de estos incrementos en el tiempo, en donde se indica el impacto de horas-persona de cada fase en cada incremento y las posibles iteraciones que en mayor o menor medida, tiene cada uno de ellos.

Con independencia del enfoque dado al ciclo de vida la diferencia fundamental del ciclo de vida de desarrollo MDA con el tradicional es que la comunicación entre las fases del proceso de desarrollo, se hace mediante modelos que un computador es capaz de entenderlos.

Básicamente los modelos tres tipos de modelos explicados a continuación, son la base de MDA [OMG 2003a]:

- Modelos independientes del computo (CIM): asimilable a los modelos del dominio y/o del negocio del proceso unificado de desarrollo [Jacobson 2001]. Así, por ejemplo, un CIM de una biblioteca hablaría de las entidades persistentes Usuario, Ejemplar, Préstamo, etc.

- Modelos independientes de la plataforma (PIM): son modelos con un alto nivel de abstracción que son independientes de la tecnología en la que se van a implantar. Describen el sistema desde el punto de vista de los procesos de negocio que van a soportar. Así, un PIM de una biblioteca hablaría de servicios de la aplicación de la biblioteca, de los objetos del negocio Usuario, Ejemplar, Préstamo, etc
- Modelos específicos de plataforma (PSM): especifican el sistema en términos de las construcciones que se van a implementar a la hora de desarrollar. Un modelo PIM puede generar distintos modelos PSM, en función de las tecnologías utilizadas. Así, un PSM de una biblioteca hablaría de JSP [SUN JSP], conexiones JDBC [SUN JDBC], etc.
- Código: la fase final del desarrollo es transformar cada PSM en código. Como cada PSM es relativo a una tecnología determinada esta transformación es, teóricamente, relativamente sencilla.

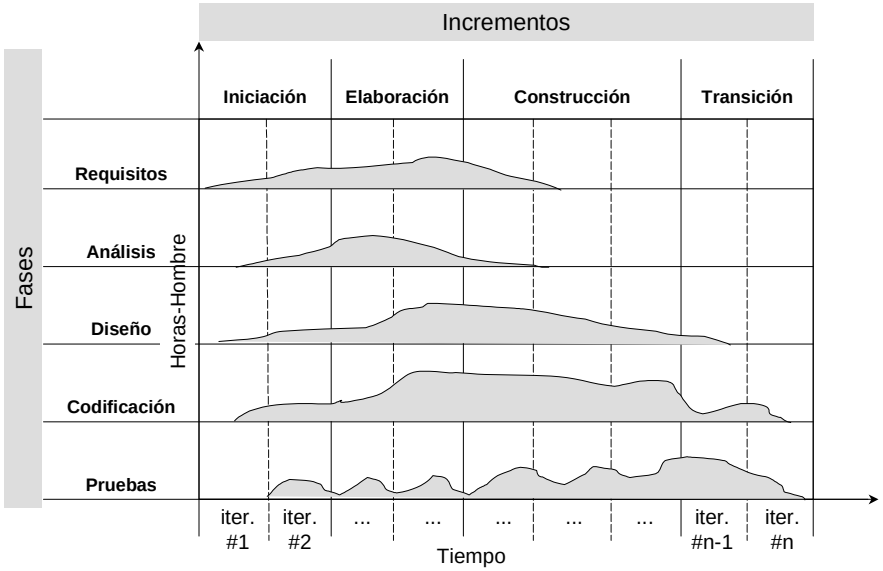

Figura 2-3. Proceso unificado de desarrollo. Diagrama de incrementos e iteraciones

MDA define los modelos PIM, PSM y el código, así como la manera de relacionarse unos con otros. Los modelos PIM se deben crear, después se deben transformar en uno o varios PSM (el paso más complejo en los desarrollos MDA) y finalmente transformarlo en código.

La novedad de MDA frente al desarrollo tradicional, es que las transformaciones se hacen se pueden hacer mediante herramientas que las ejecutan de forma automática. En concreto, la mayor aportación de MDA y su mayor beneficio, es la transformación de modelos PIM a modelos PSM. Las transformaciones, aunque es deseable que se realicen de forma automática mediante herramientas, no siempre se pueden realizar. El propio OMG, indica en los diferentes métodos de transformación que éstas pueden realizarse de forma manual, mediante el uso de perfiles, a través de marcas y patrones o de forma automática [OMG 2003a]. En general se utiliza una mezcla entre todas ellas y el programador va a tener que especificar algunas de forma manual, pero la idea de las transformaciones automáticas van a permitir a los desarrolladores tener retroalimentación de forma rápida de un modelo PIM, ya que va a poder generar prototipos de forma inmediata.

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* también, 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. Así mismo, los modelos PIM no se abandonan al comenzar a codificar. Todo lo contrario. Cada vez que se modifican los requisitos del sistema, no se va a modificar el código, como ocurría en el enfoque tradicional, si no que se va a modificar directamente el PIM y se podrían regenerar desde éste el PSM y el código. Esta es la teoría, ya que partir desde el PIM, se suele hacer sólo cuando existen cambios de requisitos de cierta relevancia, ya que como hemos visto antes, las transformaciones no son siempre automáticas.

La *portabilidad* en MDA se consigue gracias a su propio planteamiento. Siempre se va a partir del mismo PIM y en el caso de tener que migrar el sistema a otra tecnología, sólo será necesario generar el PSM apropiado para la nueva plataforma. Sólo es necesario tener una herramienta que realice la transformación, que puede encontrarse en el mercado para tecnologías con una alta tasa de uso o que haya que construirla, en caso de ser una tecnología poco utilizada.

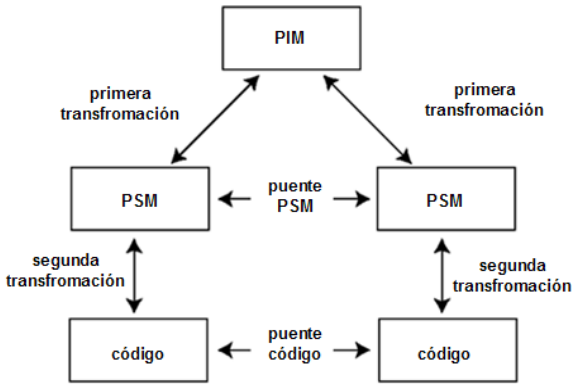

Figura 2-4. Interoperabilidad MDA mediante puentes

Como hemos visto más arriba, un PIM puede generar uno o varios PSM, en función de las plataformas en las que se vaya a implantar el sistema. El conjunto resultante de PSM, sin embargo, no estarán directamente comunicados entre ellos. Para conseguir la *interoperatibilidad*, hay que transformar conceptos de una plataforma en los de otra, construyendo lo que en terminología MDA se llaman puentes (*bridges*). Esta idea está mostrada en la Figura 2-4 ([Kleppe 2005]).

En un caso como el del ejemplo, se pueden deducir los elementos relacionados de los PSM, mediante las transformaciones que se deben realizar desde el PIM a cada uno de los PSM resultantes. Como conocemos los detalles técnicos de ambas plataformas, ya que sin ellos no se podrían definir las transformaciones, podemos obtener toda la información necesaria para construir los puentes entre los dos PSM. Lo mismo ocurre a nivel de código.

2.2.3 Arquitectura de Metamodelado de Cuatro Capas de MDA

2.2.4 Introducción al metamodelo

Se ha visto anteriormente que la aproximación MDA se basa en la generación de modelos que especifican sistemas desde diferentes puntos de vista. También se ha visto que se deben definir las transformaciones necesarias para que, partiendo de un modelo PIM podamos obtener modelos desde otros puntos de vista diferentes (PSM) y generar el código de la aplicación. Para poder llevar a la práctica esta teoría, los modelos deberán definirse en base a un lenguaje que nos proporcione la posibilidad de escribir, de una cierta manera formal nuestros modelos.

Al ser un lenguaje de modelado, y en aras a favorecer su usabilidad, a diferencia de los lenguajes basados en texto, la sintaxis concreta debería estar definida de forma gráfica. No podemos basarnos en gramáticas que describan las expresiones correctas de la sintaxis, como por ejemplo BNF. Necesitamos pues mecanismos diferentes que permitan definir lenguajes en el contexto de la arquitectura MDA. Este mecanismo es lo que se define como metamodelo.

> *Un metamodelo es un lenguaje específico de dominio que se orienta hacia una representación de metodologías de desarrollo de software.[González 2008].*

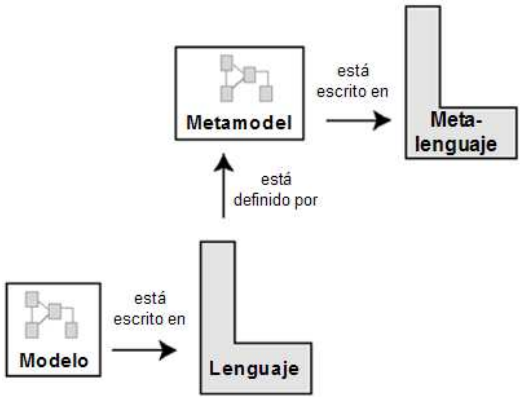

Figura 2-5. Modelos, lenguajes, metamodelos y metalenguajes

Un modelo define qué elementos pueden existir en un sistema determinado. Si definimos una clase `Ratón` en un modelo, podremos tener diferentes instancias de `Ratón` tales como `Mickey`, `Jerry`, `Pixie`, `Dixie` y un largo etcétera. Sin embargo un lenguaje sólo define qué elementos pueden existir, los que pueden ser utilizados en un modelo. En el ejemplo anterior, mi lenguaje de modelado tendrá el concepto "Clase", mediante la que podemos definir los ratones y las demás especies relacionadas con el mundo de los dibujos animados. En un lenguaje más concreto y conocido (UML), existen conceptos como "Clase", "Atributo" o "Estado". Viendo esta similitud, podemos definir un lenguaje mediante un modelo superior o metamodelo: el modelo del lenguaje describe los elementos que pueden ser usados por el lenguaje. En definitiva, cada elemento que un desarrollador puede utilizar en su modelo, es definido por el metamodelo del lenguaje que se utilice para modelar.

Lo expuesto, se resume en la Figura 2-5 ([Kleppe 2005]) que ilustra las relaciones entre los conceptos de modelo, lenguaje, metamodelo y metalenguaje.

No obstante, a esta descripción se le pueden hacer algunas matizaciones. Un metalenguaje juega un papel diferente al del lenguaje, ya que se especializa en describir lenguajes de modelado, por lo

que usa diferente simbología. Por otro lado, un metamodelo define completamente un lenguaje. Por esto, no sería necesario hacer la distinción entre lenguaje y el metamodelo que define el lenguaje, pudiéndose resumir el esquema anterior como se muestra en la Figura 2-6 ([Kleppe 2005]).

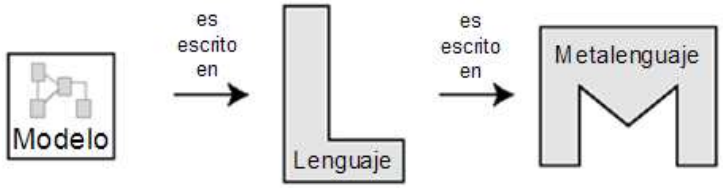

Figura 2-6. Modelos, lenguajes y metalenguajes

Observando este gráfico, se deduce que la arquitectura MDA se puede jerarquizar en una serie de niveles o capas que van del más particular al más general. En concreto, OMG define los cuatro niveles, detallados a continuación.

2.2.5 Arquitectura de metamodelado de cuatro capas

La arquitectura de metamodelado de cuatro capas [OMG 2007a] se define, no sólo como consecuencia del metamodelado, sino para entender con posterioridad qué papeles juegan los estándares MDA definidos por OMG. Cada uno de ellos (UML, MOF, QVT, etc) tiene su lugar y su sentido esta definición de capas. Las capas definidas por OMG son las siguientes:

- *Capa M0: Las instancias*. Es la capa en donde se ejecuta el sistema, donde están las instancias reales que se han creado durante la ejecución. A este nivel existirán las instancias de un gato llamado "Tom" o un ratón llamado "Jerry". Como es lógico podrán existir varias instancias de cualquier objeto. "Mickey", por ejemplo, será otro ratón de nuestro sistema.
- *Capa M1: El modelo del sistema*. A este nivel se define el modelo del sistema o la aplicación propiamente dicha. Es donde se reflejarán los conceptos existentes en nuestro sistema, tales como Ratón y Gato, junto con sus propiedades (en el caso de la Figura 2-7, sólo tienen el nombre). Los conceptos a este nivel son categorizaciones o clasificaciones de las instancias del nivel M0. Esto es, cada elemento del nivel M0 es una instancia del nivel M1 (por ejemplo "Jerry" es una instancia de Ratón).
- *Capa M2: Metamodelo*. Este nivel contiene los elementos del lenguaje de modelado o metamodelo. El lenguaje tendrá elementos como Clases y Atributos que permitirán definir el modelo del sistema de la capa M1. Al igual que en el nivel anterior, todos los conceptos definidos en el nivel M1 son instancias de los elementos definidos en este nivel. Así tenemos que los conceptos Ratón y Gato, son instancias del elemento Clase del metamodelo. Igualmente, cada propiedad es una instancia de Atributo.
- *Capa M3*: *Meta-metamodelo*. Siguiendo la misma línea que los anteriores, podemos definir los elementos existentes en la capa M2, mediante instancias de elementos existentes en esta capa. Realmente esta capa define un lenguaje de modelado para el metamodelo de la capa inmediatamente anterior. En definitiva, en este nivel está definido el meta-metalenguaje o meta-metamodelo.

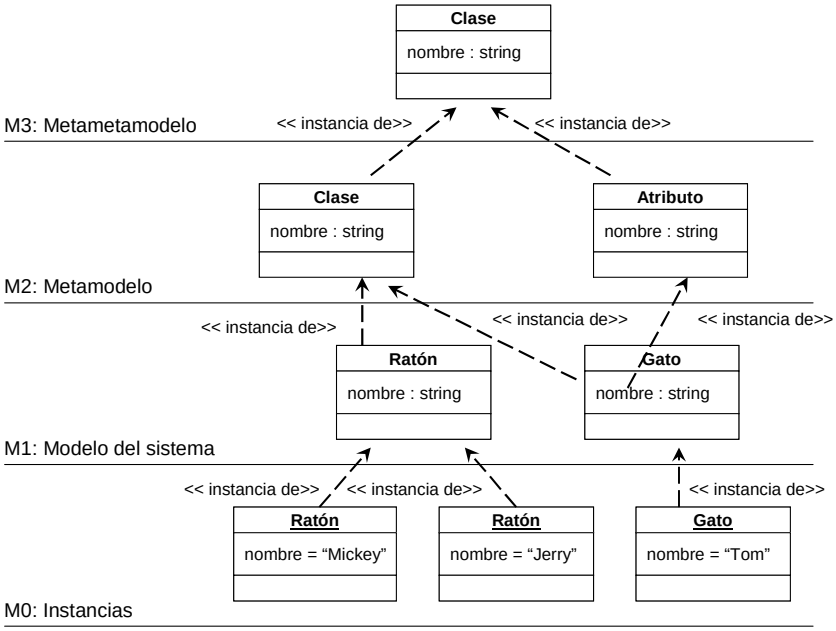

Figura 2-7. Jerarquía de Metamodelos en la Arquitectura de Cuatro Capas

La idea fundamental que hay detrás de esta definición de niveles jerárquicos es que existen elementos que cuyas instancias están asociadas a instancias de otros elementos. Si nos centramos en los estándares de OMG, estos elementos serán clases MOF que se podrán instanciar en clases UML, que a su vez se instanciarán en las clases del modelo del sistema. En definitiva con esta jerarquía de niveles se observa que M3 describe el paradigma del modelado.

Es razonable contemplar la posibilidad de definir quinto nivel M4 que fuera el meta-meta-metamodelo. Igualmente, se podría definir una jerarquía de modelos de forma infinita, al menos desde el punto de vista teórico. El nivel M3 es un lenguaje de modelado, por lo que se podría definir un metamodelo de M3. No obstante, para definir este hipotético nivel M4, podemos usar el propio lenguaje definido en M3, ya que M3 es autodescriptivo. En base a esto, OMG limita esta recursividad a un número discreto, definiendo cuatro niveles ya expuestos.

Los estándares MDA definidos por OMG tienen un lugar dentro de esta jerarquía de niveles, aunque esto se va a explicar en el próximo capítulo, como anticipo al mismo, comentar que en esta arquitectura UML se situará en el nivel M2, como lenguaje de modelado, mientras que MOF se encontraría en el nivel M3, definiendo UML o cualquier lenguaje de modelado que se desee definir.

2.3 Conceptos MDA

Llegado a este punto, se debe profundizar un poco en los conceptos de MDA que han ido apareciendo a lo largo de los epígrafes anteriores. Se han comentado por encima conceptos como modelos o transformaciones entre modelos, pero es conveniente verlos con un poco más de detalle e introducir otros, que no se han mencionado directamente, pero que se derivan de éstos.

2.3.1 Modelos

En apartados anteriores, ya se han hecho referencias al concepto de modelo. No obstante esta sección, se va a centrar un poco más en la definición de este concepto que, en definitiva es el elemento principal de la arquitectura MDA.

Un modelo es una representación en un cierto medio de algo en el mismo u otro medio. El modelo capta los aspectos importantes de lo que se está modelando, desde un cierto punto de vista, y simplifica u omite el resto [Rumbaugh 2007]. Es decir, es una representación abstracta de algo que existe en la realidad, diferente de lo representado, ya que no contempla todos sus detalles, y que se usa como un ejemplo para reproducir una realidad determinada. En el contexto del desarrollo del software, la realidad representada no es más que el sistema que se pretende desarrollar. Por eso se puede resumir un modelo como una abstracción semánticamente completa de un sistema determinado [Jacobson 2001].

Como se ha comentado más arriba, un modelo se debe escribir en un lenguaje, que para que se puedan realizar las transformaciones entre los diferentes modelos, este lenguaje debe ser un lenguaje bien definido en el sentido de que pueda ser interpretado automáticamente por un computador. Partiendo de estas ideas [Kleppe 2005] hace las siguientes definiciones:

> *Un modelo es una definición de un sistema, o de una parte de él, escrita en un lenguaje bien definido.*
>
> *Un lenguaje bien definido es un lenguaje con una sintaxis y semántica bien definida que es adecuado para la interpretación automática por un computador.*

En general, como dice la definición, no va a existir un único modelo que represente todo el sistema. Existirán distintos modelos que representen parcialmente al mismo, incluso que se solapen o que, aunque representen la misma parte del sistema, pretendan modelar conceptos diferentes. Se puede hacer una clasificación de los tipos de modelos en base a la información que aporte. Por ejemplo, un modelo puede indicar en qué fase del ciclo de vida se usa (análisis o diseño), si es un modelo abstracto o detallado, si describe un modelo de negocio o un modelo software o si está asociado a una determinada tecnología o es independiente de cualquier plataforma. Todas estas distinciones de uso de los modelos, deben estar claras a la hora de realizar las transformaciones, permitiéndonos hacer una clasificación de los mismos.

- *Modelos de negocio y modelos software*. Un modelo de negocio describe los procesos de negocio que se pretenden implementar en un sistema. Al definir un modelo, no existe la necesidad de mencionar ningún elemento de los sistemas que lo van a soportar. Por otro lado, los sistemas software que especifican los modelos de negocio, se definen mediante los modelos software. Para la mayoría de los modelos de negocio existirán varios modelos software que describen los diferentes sistemas que van a soportar las distintas partes del negocio. En definitiva, los modelos de negocio se corresponden con los modelos CIM y los modelos software a los PIM.
  Existe el inconveniente de que no se puede derivar de forma automática un CIM en su totalidad en un PIM. Para elegir qué parte del modelo de negocio se va a representar mediante un modelo PIM se requiere intervención por parte del desarrollador y el propio modelo PIM debe desarrollarse previamente para poder indicar qué parte del CIM describe.
- *Modelos estructurales y modelos de comportamiento*. La diferencia entre ambos tipos de modelos es el punto de vista mediante el que se describen los diferentes modelos. Cuando el modelo describe aspectos estáticos del sistema – como pueden ser las clases – se está hablando de un modelo estructural, mientras que los modelos de comportamiento muestran aspectos del sistema mediante un punto de vista dinámico, como la secuencia de estados de un sistema.

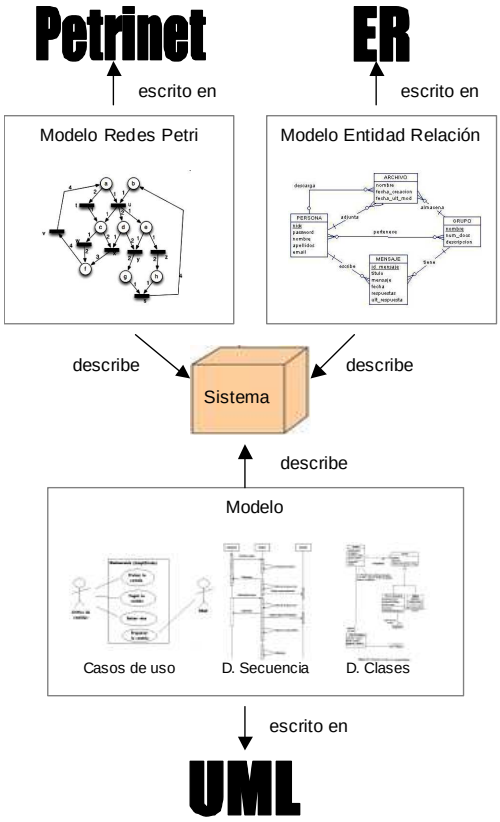

Figura 2-8. Diferentes acepciones de los modelos en función del lenguaje utilizado

No obstante, hay que hacer una apreciación sobre el concepto de modelo de un sistema, ya que en función del lenguaje de modelado utilizado, podemos estar indicando dos cosas diferentes. Por un lado los modelos pueden representar elementos estructurales o dinámicos en sí mismos y por otro se puede estar hablando de diferentes diagramas que representan estos mismos aspectos y que, en su conjunto, representan el modelo completo del sistema. Por ejemplo, como se representa en la Figura 2-8, mediante UML se puede definir el modelo de un sistema mediante un conjunto de diagramas que muestran ambos puntos de vista. Sin embargo, un modelo entidad-relación describe un único aspecto del sistema. Éste se puede complementar con un modelo basado en redes Petri, que especifica otro aspecto diferente del mismo sistema. Por lo tanto, la característica esencial de un modelo es el lenguaje en el que está escrito, ya que algunos lenguajes son más expresivos que otros, a la hora de representar determinados aspectos del sistema.

- *Modelos independientes de la plataforma y modelos específicos de plataforma*. Es difícil poder decidir, dado un modelo, si es un modelo PIM o PSM. La línea que divide ambos tipos de modelos es muy delgada. Realmente, no se puede aseverar con certeza el tipo de modelo, sino que sólo podremos identificar el grado en el que un modelo es PIM o PSM, de manera que en una transformación MDA, realmente se transformará un modelo, en un alto grado independiente de la plataforma, en otro modelo que tiene un alto grado de modelo específico de una plataforma. Por ejemplo, se podría ver como un modelo PIM algo descrito en términos de la máquina virtual de Java, pero si en ese modelo se vieran conceptos como "JSP" o "PHP", se asociaría a un modelo PSM.
- *Modelos de diferentes plataformas*. Otro elemento diferenciador de los modelos, en concreto los PSM, serán las plataformas a las que están enfocadas. Cabe preguntarse también, si dados

dos modelos de dos plataformas diferentes, se pueden diferenciar claramente. La respuesta es que sí, ya que los constructores que se utilizan van a ser muy diferentes. No tiene nada que ver la estructura de un modelo enfocado a EJB y uno enfocado a Smalltalk. En el ejemplo anterior, para conseguir ambos modelos PSM desde un mismo PIM, se necesitarán aplicar diferentes reglas de transformación. Es importante conocer la plataforma objetivo de los modelos y el grado en el que el modelo es específico de esa plataforma. Esto facilitará la definición de las transformaciones.

2.3.2 Transformaciones de Modelos

Ya se ha comentado que una de las grandes ventajas de MDA es que, partiendo de modelos escritos en un lenguaje determinado, se puedan obtener diferentes modelos mediante transformaciones. Se ha visto en la sección 2.2.2, cómo mediante transformaciones sucesivas, se puede pasar de un modelo PIM definido por el desarrollador, al código de la aplicación. Para ello, se deben definir las transformaciones que se deben realizar y deben existir herramientas que, aplicando estas definiciones, ejecuten las mismas.

[OMG 2003a] define la transformación de modelos como el proceso que convierte un modelo en otro del mismo sistema, mediante un conjunto de reglas. El modelo a transformar suele identificarse como "modelo fuente", mientras que el transformado, se le suele denominar "modelo objetivo". En general, las transformaciones no suelen ser de modelos completos, sino que se van realizando transformaciones parciales de ellos. Lo importante es que las definiciones de las reglas de transformación deben estar especificadas sin ambigüedad, para que la transformación sea única. Así mismo uno de los retos más importantes de las transformaciones es lograr la trazabilidad de los requisitos definidos en los modelos CIM, para poder verificar que, mediante las transformaciones sucesivas, el código resultante cumpla con todos ellos.

Partiendo de estas ideas, en [Kleppe 2005] se definen los siguientes conceptos:

> *Transformación: generación automática de un modelo objetivo desde un modelo fuente, de acuerdo con una definición de transformación.*
>
> *Definición de transformación: conjunto de reglas de transformación que juntas describen cómo un modelo en un lenguaje fuente puede ser transformado en un modelo en el lenguaje objetivo.*
>
> *Regla de transformación: descripción de cómo una o varias construcciones descritas en un lenguaje fuente, pueden ser transformadas en una o varias construcciones en un lenguaje objetivo*

Lo que se deduce de estas definiciones, es que las transformaciones se definen a nivel de los metamodelos. Mediante las reglas de transformación, se van creando las correspondencias entre las construcciones de los diferentes lenguajes implicados (es decir, entre los metamodelos de estos lenguajes, los cuales serán instancias de sus meta-metamodelos correspondientes). Podemos por ejemplo, especificar una definición de transformación entre el metamodelo UML y la gramática de C# que describa qué código C# será generado para un modelo UML. Esta situación está descrita en la Figura 2-9 ([Kleppe 2005]).

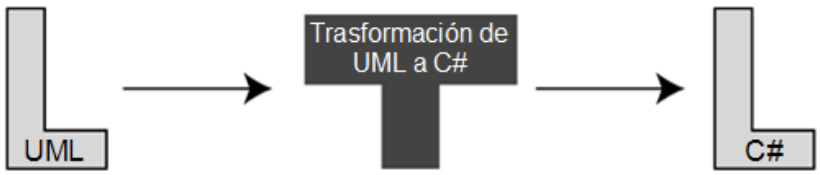

Figura 2-9. Definición de transformaciones entre lenguajes

Para que estas definiciones se puedan llevar a cabo, deben existir herramientas que las implementen. Estas herramientas, aplicando las reglas definidas, consiguen realizar la transformación de cualquier modelo fuente en un modelo objetivo.

En [OMG 2003a] se muestra un diagrama más intuitivo a la hora de explicar las transformaciones. La Figura 2-10 ([OMG 2003a]) muestra de forma intuitiva el planteamiento de que la transformación se debe apoyar en una serie de reglas que la definen. Cabe preguntarse cómo se definen estas reglas y qué elementos utiliza la arquitectura MDA como apoyo a su definición.

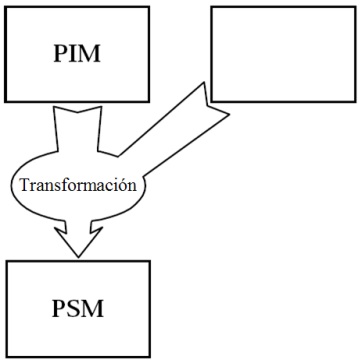

Figura 2-10. Modelo de transformación

En el caso concreto de las transformaciones de PIM a PSM, se utilizan mapas de transformación de modelos, mediante los cuales se definen las transformaciones particulares a realizar sobre los modelos de análisis, para conseguir un modelo de diseño específico de una plataforma. El mapeo de un modelo se define en base a funciones, que son colecciones de reglas o algoritmos que definen el mismo. Estas funciones, como se muestra en la Figura 2-11 ([OMG 2003a]), se definen siempre a nivel del metamodelo, aunque operan al nivel de los modelos. El objetivo de la aproximación MDA es poder realizar las transformaciones de forma automática, por lo que es necesario que las funciones de mapeo se definan con cierto formalismo para lograr que no sean ambiguas. El lenguaje de transformación *Query, Views and Transformations* (QVT) provee el formalismo necesario para su definición.

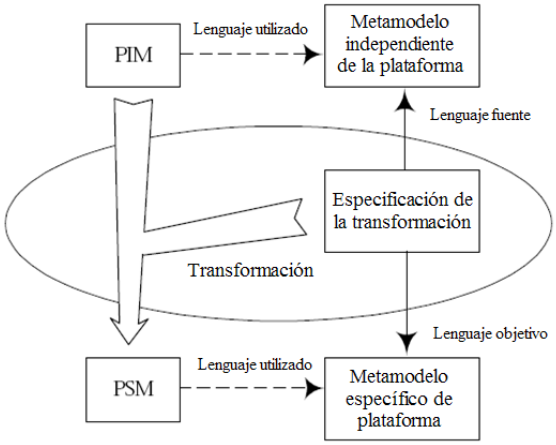

Figura 2-11. Transformación de metamodelos

Una manera de facilitar la transformación es identificar en el modelo PIM, qué elementos deben transformarse en la plataforma destino mediante marcas. Éstas indican qué mapa de transformación concreto debe ser utilizado para transformar el elemento en uno o más elementos de un modelo PSM. Estas marcas alejan al modelo PIM de su independencia de la plataforma por lo que es importante tratar el marcado como una capa transparente que se superpone al modelo. La Figura 2-12 ([OMG 2003a]), representa el uso de marcar como ayuda a la transformación y su situación intermedia entre la independencia y la dependencia de la plataforma. La representación es bastante intuitiva, ya que se ve cómo, partiendo de una plataforma elegida, existe un mapa de transformación que incluye un conjunto de marcas que se usan para marcar los elementos del PIM como guía para la transformación de modelos.

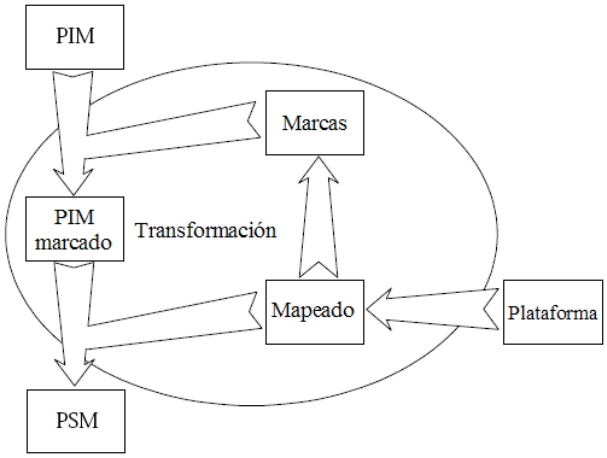

Figura 2-12. Marcado de un modelo

Existen otros elementos que facilitan las transformaciones, como son el uso de patrones (o plantillas) y el uso de información adicional, que complementan la definición de los mapas de transformación. Los patrones son modelos parametrizados que especifican tipos concretos de transformaciones. A éstos se les puede asociar un conjunto de marcas para identificar instancias en un modelo, que deben ser transformadas de acuerdo a las indicaciones del patrón.

En base a todo esto, [OMG 2003a] especifica los métodos de transformación que se pueden llevar a cabo, insistiendo en que no son métodos excluyentes, ya que las herramientas de transformación pueden combinarlos para conseguir su máxima efectividad. Los diferentes métodos existentes son:

- *Transformaciones manuales*: se suelen dar cuando hay que tomar decisiones sobre el diseño durante la fase de desarrollo en función del contexto de una implementación específica.
- *Transformación de un PIM preparado usando perfiles*: se utilizan perfiles UML independientes de la plataforma para preparar modelos PIM, de manera que el modelo preparado se transforme en un modelo PSM. Este tipo de transformaciones requieren que se marque el modelo PIM con marcas del perfil específico de la plataforma
- *Transformaciones utilizando patrones y marcas*: se realizan mediante mapas de trasformación que incluyen marcas y patrones, de manera que se genere un modelo PIM con las marcas específicas de la plataforma sobre el que se aplica el patrón para generar el modelo PSM.
- *Transformaciones automáticas*: se pueden realizar cuando el modelo PIM proporciona toda la información necesaria para la implementación, por lo que no se necesita ninguna marca ni el uso de perfiles adicionales para generar el código.

También se pueden clasificar las transformaciones como en [France 2001], que las categorizan en base al nivel de abstracción de los modelos fuentes y objetivos de la misma:

- *Transformaciones verticales*: son aquellas en las que los niveles de abstracción del modelo fuente y objetivo son diferentes. Refinar un modelo o implementarlo en un lenguaje de programación concreto son ejemplos de transformaciones verticales. En el ámbito de MDA se contemplan varias transformaciones verticales. Así por ejemplo, cuando un PIM se transforma en uno o varios PSM, se produce una transformación vertical; igualmente, cuando partiendo de una implementación en una plataforma concreta, se pretende una abstracción de los detalles concretos de dicha plataforma, se está aplicando una transformación vertical para pasar del PSM al PIM.
- *Transformaciones horizontales*: son aquellas en las que el modelo fuente y objetivo corresponden al mismo nivel de abstracción. Estas transformaciones se pueden aplicar para evolucionar modelos ya sea para perfeccionarlo y mejorar su diseño, para corregir errores de diseño bien para adaptarlo a nuevos requisitos o restricciones. La aplicación de transformaciones horizontales en MDA permite además obtener modelos de diferentes vistas del sistema, pasando de un PIM a otro; o bien obtener también modelos específicos para distintas plataformas pasando de un PSM a otro.

3. ESTÁNDARES MDA

Este capítulo va a repasar de forma breve algunos de los estándares más representativos de la aproximación MDA, definidos por OMG. Estos estándares son las herramientas que facilitarán a los desarrolladores plasmar materializar los conceptos explicados en el capítulo anterior al utilizar las herramientas que los implementen.

El análisis se centrará en los estándares: UML [OMG UML], Perfiles UML [OMG 2007a], MOF [OMG 2006a], CWM [OMG 2003b], QVT [OMG 2008a], MOF2TEXT [OMG 2008b], OCL [OMG 2006b] y XMI [OMG 2007c]. Otros estándares que no van a ser analizados en este trabajo, pero que caben mencionar son:

- *Metadata Interchange Patterns* (MIP) [OMG 2004a]
- *UML Human-Usable Textual Notation* (HUTN) [OMG 2004b]
- *OMG Systems Modeling Language* (SysML) [OMG 2008e]
- *Ontology Definition Metamodel* (ODM) [OMG 2008d]
- *Reusable Asset Specification* (RAS) [OMG 2005b]
- *Semantics of a Foundational Subset for Executable UML Models* (FUML) [OMG 2008f]
- *Software Process Engineering Metamodel* (SPEM) [OMG 2008c]

3.1 Infraestructura común

Antes de entrar en el detalle de los diferentes estándares definidos por OMG, es importante centrarse en la definición de la infraestructura común que va a dar soporte a todos ellos. En [OMG 2007a] se especifica esta infraestructura como núcleo del lenguaje UML, pero al ser una infraestructura común a varios de los estándares, se comentan generalidades que se han decidido incluir en este apartado.

La infraestructura UML se define por la *InfrastructureLibrary* que especifica el núcleo (*core*) del metalenguaje. Define constructores básicos y conceptos comunes que se reutilizan para definir varios metalenguajes, tales como MOF o CWM, a parte del propio UML. También tiene el objetivo de alinear arquitectónicamente MOF y UML con el fin de poder reutilizar los mismos metamodelos para ambos lenguajes. Por último, esta librería también permite una personalización de UML mediante perfiles que facilitan la creación de nuevos lenguajes basados en el mismo *core*. La *InfrastructureLibrary* es un paquete que está compuesto a su vez por los paquetes *Core* y *Profiles*.

El paquete *Core* es un metamodelo completo diseñado para una alta reusabilidad, de manera que permita extender metamodelos descritos en MOF en el mismo metanivel, mediante la utilización o especialización de sus metaclases. Tal es el caso de UML, CWM y MOF que todas dependen de un núcleo común, tal y como se muestra en la Figura 3-1 ([OMG 2007a]). El paquete *Core*, por lo tanto, es el corazón que sustenta toda la arquitectura de la aproximación MDA.

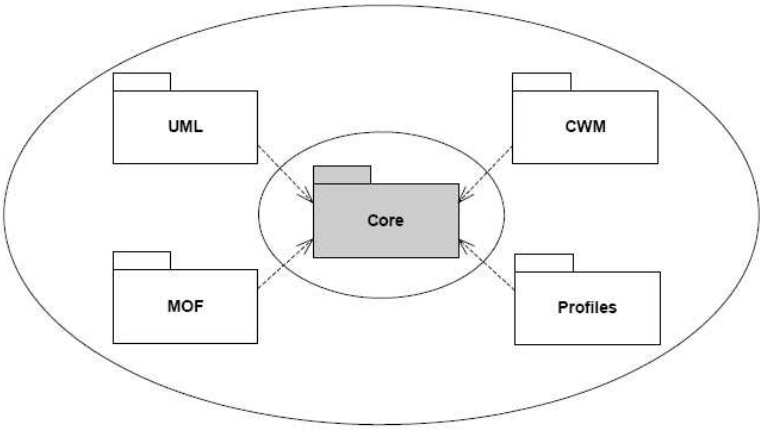

Figura 3-1. Papel del núcleo común

El paquete *Profiles* depende del paquete *Core* y define los mecanismos que se usan para personalizar metamodelos existentes centrándose en plataformas específicas o dominios particulares. Si se desea extender un metamodelo, como es el caso de UML, se puede especificar un perfil partiendo de este paquete y generar un nuevo lenguaje de modelado específico y personalizado. Se puede considerar como un mecanismo de extensión ligero de los metamodelos definidos con MOF alternativo al metamodelado, ya que es mucho más cómodo ampliar el metamodelo que crear uno completamente nuevo. Además la ventaja es que, como se indica en la Figura 3-2 ([OMG 2007a]), con *Profiles* se puede extender un metamodelo definido en MOF sin variar su definición original.

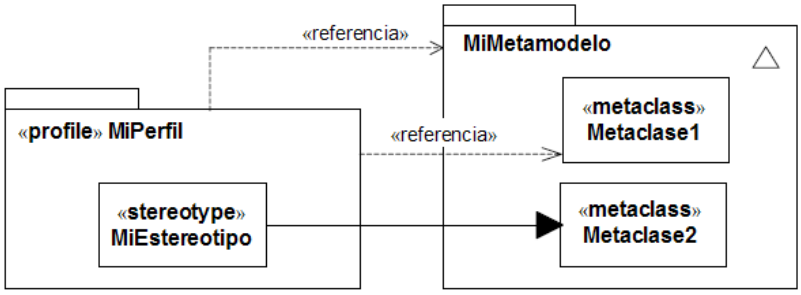

Figura 3-2. Extensión de un metamodelo mediante *Profiles*

En este ejemplo se ve que MiMetamodelo es un metamodelo que contiene dos metaclases. MiPerfil es un perfil que referencia a MiMetamodelo y a una de sus clases (Metaclase1). Sin embargo hay una referencia explícita a Metaclase2 que anula la referencia al metamodelo. La aplicación de MiPerfil sobre algún modelo basado en MiMetamodelo, mostrará instancias de Metaclase2 (porque está explícitamente referenciada mediante una referencia de metaclase). También estarán visibles las instancias de Metaclase1 que son extendidas por una instancia de MiEstereotipo. Sin embargo, las instancias de Metaclase1 que no se extienden por MiEstereotipo permanecerán ocultas.

3.2 *Unified Model Language* (UML)

UML [Booch 2005] es un lenguaje estándar para visualización, especificación, construcción y documentación de sistemas software y otros sistemas no software. Representa una colección de buenas prácticas que proporcionan un éxito acreditado en el modelado de grandes y complejos sistemas [OMG 2005a]. Fusiona conceptos de las metodologías de *Booch* [Booch 2007], de OMT (*Object Modeling Technique*) [Rumbaugh 1996] y de OOSE (*Object-Oriented Software Engineering*) [Jacobson 1996] consiguiendo como resultado un lenguaje de modelado común, sencillo y ampliamente utilizado por los usuarios de éstos y otros métodos de desarrollo, ampliando sus posibilidades. Por ejemplo, UML puede modelar sistemas concurrentes y distribuidos.

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

- Proveer a los usuarios de un lenguaje de modelado fácil de usar y visual 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.

La especificación del lenguaje UML está basada en la aproximación de metamodelado, por lo que se sitúa en el nivel M2 de la arquitectura MDA. La definición de UML está basada en los siguientes principios de diseño [OMG 2007a]:

- *Modularidad*: que se consigue agrupando constructores en paquetes y organizando aspectos comunes en metaclases.
- *Estratificación*: la división en capas se aplica de dos formas en el metamodelo UML. La primera es la estructura de los paquetes que se estratifica para separar los constructores, que forman el núcleo (*core*) del metalenguaje, de los constructores de alto nivel que los usan. En segundo lugar, el patrón de la arquitectura de cuatro capas se aplica para separar aspectos en diferentes niveles de abstracción.
- *División en partes*: se organizan en partes diferentes áreas conceptuales dentro de la misma capa.
- *Extensibilidad*: UML se puede extender de dos maneras.
  - Mediante el uso de perfiles (*profiles*) se pueden crear nuevos dialectos, adaptando construcciones a plataformas o dominios específicos.
  - Reutilizando parte del paquete *InfraestructureLibray* para aumentarlo con nuevas metaclases y metarrelaciones y así crear un nuevo lenguaje de modelado relacionado con UML, que estrictamente sería ya un metamodelo diferente.
- *Reusabilidad*: la reusabilidad se consigue mediante librerías de metamodelado flexibles que permitan definir metamodelos como el UML o como otros relacionados, tales como MOF (*Meta Object Facility*) o CWM (*Common Warehouse Metamodel*), que se verán más adelante.

El metamodelo de UML se define en UML *Superstructure* [OMG 2007b], metamodelo descrito en MOF y que a su vez está basado también el paquete *Core*, como se ha comentado en el apartado anterior. El paquete UML proporciona los constructores a nivel de usuario y se compone de

diferentes paquetes que se encargan de gestionar los diferente modelos estáticos (o de estructura) y dinámicos (o de comportamiento).

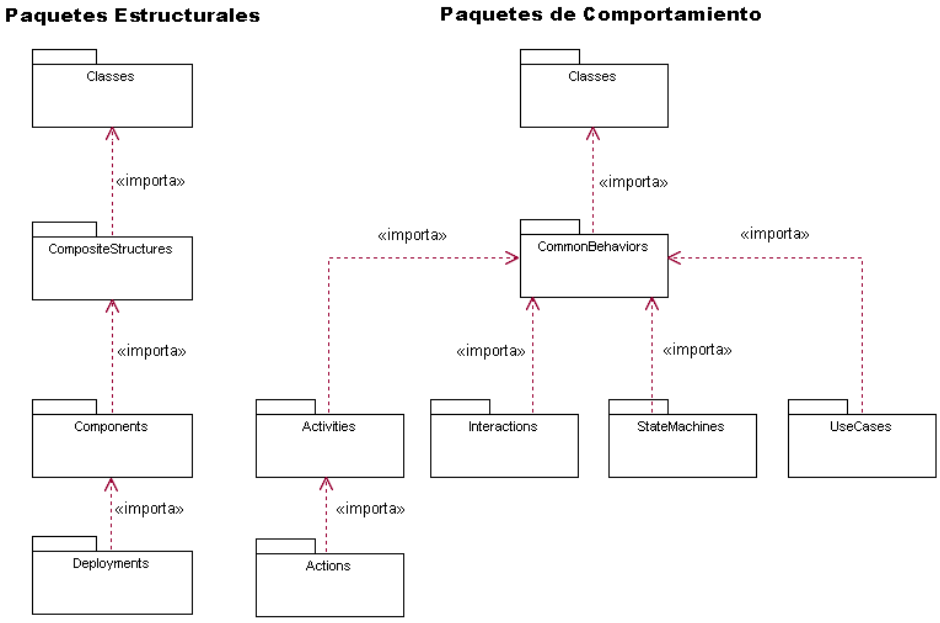

Figura 3-3. Paquetes UML que soportan modelos estructurales y de comportamiento

Los constructores de estructura y los de comportamiento están definidos para dar soporte a los diferentes tipos de diagramas que proporciona UML [OMG 2005a]. Los diagramas estáticos: diagramas de clases, de componentes, de objetos y de despliegue, los soportan los constructores de estructura, mientras que para los dinámicos: diagrama de estados, de actividades, de secuencia, de colaboración y de casos de uso, están los constructores de comportamiento. Tal y como se indica en la Figura 3-3 ([OMG 2007b]), los constructores relacionados con estos tipos de diagramas se definen con los siguientes paquetes:

- Estructura:
  - *Classes*
  - *CompositeStructures*
  - *Components*
  - *Deployments*
- Comportamiento:
  - *CommonBehaviors*
  - *Activities*
  - *Interactions*
  - *StateMachines*
  - *UseCases*
  - *Actions*

3.3 UML *Profiles*

Como ya hemos comentado, *UML Infrastructure* define la posibilidad de extender UML de manera que se pueda crear un nuevo lenguaje de modelado sin necesidad de definirlo modificando su metamodelo definido en MOF. Este mecanismo lo proporcionan los perfiles UML (*Profiles*) y lo que permiten es aplicar a UML nuevas especificaciones añadiendo nuevos tipos de elementos del lenguaje o restricciones al mismo [OMG 2003a].

Este mecanismo lo proporciona el paquete *Profiles* el cual define los mecanismos para extender y adaptar las clases de un metamodelo cualquiera descrito en MOF a diferentes propósitos o necesidades concretas, tales como los demandados por diferentes plataformas (como pueden ser J2EE o .NET) o dominios de aplicación (como los de tiempo real, modelado de procesos de negocio, etc.).

[OMG 2007a] señala varias razones por las que se puede necesitar personalizar un metamodelo:

- Disponer de una terminología propia que se adapte a una plataforma particular o un dominio de aplicación.
- Dotarse de una sintaxis para construcciones que no cuentan con una notación propia (como es el caso de las acciones).
- Obtener nuevas notaciones para símbolos ya existentes para adaptarlas al dominio de aplicación.
- Añadir semántica no especificada en el metamodelo
- Añadir restricciones al metamodelo de manera que acoten el uso del metamodelo y sus constructores.
- Añadir información útil para la realización de transformaciones entre modelos.

Un perfil se define en un paquete UML estereotipado «*profile*» que extiende a un metamodelo o a otro perfil. Los mecanismos para definir los perfiles son tres: estereotipos (*stereotypes*), restricciones (*constraints*) y valores etiquetados (*tagged values*) [Fuentes 2004].

Un estereotipo viene definido por un nombre y por una serie de metamodelos sobre los que puede asociarse. Gráficamente, los estereotipos se definen dentro de cajas estereotipas «*stereotypes*» que se asocian con las metaclases que va a extender, según la Figura 3-4 ([Fuentes 2004]), el perfil UML `WeighthsAndColors` define dos estereotipos: `Colored` y `Weighed` que se asocian a las metaclases `Class` y `Association` el primero y sólo a `Association` el segundo.

Las restricciones son condiciones que se aplican a los estereotipos. En concreto a los elementos del metamodelo que han sido estereotipados. En el ejemplo de la Figura 3-4, se puede imponer la restricción de que si dos o más clases están unidas por una asociación coloreada, el color de las clases debe coincidir con el de la asociación. Estas restricciones suelen escribirse con el lenguaje OCL.

Un valor etiquetado es un meta-atributo adicional que se asocia a una metaclase del metamodelo extendido por un perfil. Todo valor etiquetado deberá tener un nombre y un tipo y deberá estar asociado a un determinado estereotipo. En el ejemplo de la Figura 3-4 el estereotipo `Weighed` tiene un valor etiquetado `Weight` de tipo `integer` que indicará el peso de cada asociación que haya sido estereotipada como `Weighed`. Los valores etiquetados se representan como atributos de la clase que define el estereotipo.

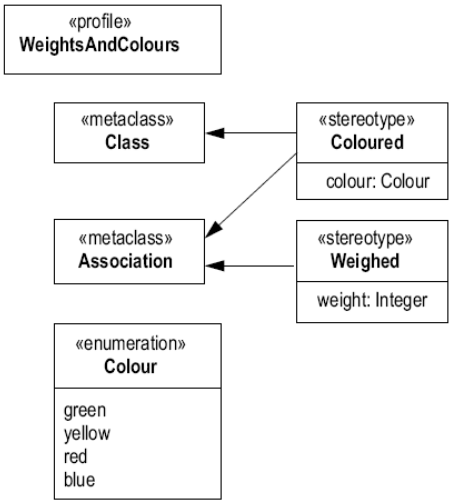

Figura 3-4. Ejemplo de Perfil UML

3.4 *Meta Object Facility* (MOF)

MOF es un estándar de OMG que provee de un marco de trabajo de gestión de metadatos y de un conjunto de servicios para permitir el desarrollo de la interoperabilidad de sistemas dirigidos por modelos y metadatos [OMG 2006a]. Muchas de las tecnologías estandarizadas por OMG (UML, el propio MOF, CWM, SPEM, XMI y varios perfiles UML) usan MOF y sus tecnologías derivadas para un intercambio dirigido por metadatos y la manipulación de los mismos. Así mismo, MOF introduce el concepto de los modelos independientes de plataforma de metadatos, además del mapeo de estos PIM a plataformas específicas.

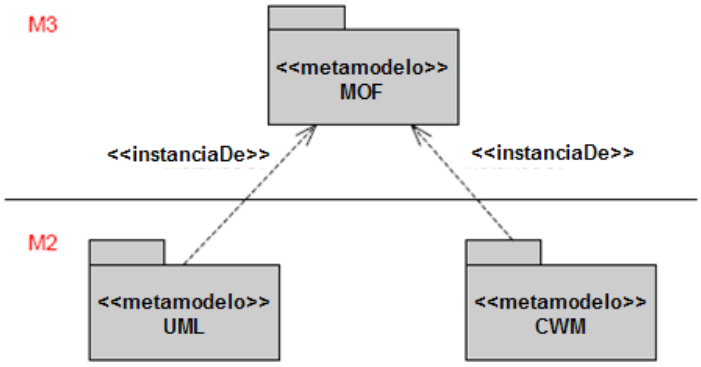

Figura 3-5. Metaniveles de MOF y UML

Como se muestra en la Figura 3-5 ([OMG 2007a]), en la arquitectura de metamodelado de cuatro capas MOF estaría situado a nivel del meta-metamodelo (M3), ya que con él, como hemos

viso, se definen los metamodelos situados en el nivel M2. No obstante, desde su primera versión, MOF ha estado íntimamente ligado a UML, debido al alineamiento arquitectónico derivado de compartir el mismo *core*. Uno de los mayores éxitos de la infraestructura común es precisamente este alineamiento [OMG 2007a]. Mediante el paquete *Core* se consigue que todos los elementos del metamodelo sean compartidos por UML y MOF. No obstante, UML se define como un modelo descrito mediante un metamodelos MOF. Es decir, cada elemento de UML es una instancia de un elemento del modelo de MOF. Este alineamiento es posible porque la *InfrastructureLibrary* UML se usa en ambos metaniveles.

El hecho de que tanto MOF como UML tengan en común la *InfrastructureLibrary,* incluye los siguientes beneficios[OMG 2006a]:

- 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

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 autodescriptivo.
- *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

3.5 *Common Warehouse Metamodel* (CWM)

El estándar CWM [OMG 2003b][Poole 2003] también es una instancia de MOF, lo cual le permite usar otros estándares MDA dependientes de MOF. En particular permite utilizar XMI para el intercambio de almacenes de metadatos que se representan utilizando el propio metamodelo y permite el uso de IDL (*Interface Description Language*) y otros lenguajes de programación para implementar el acceso a almacenes de metadatos basados en metamodelos CWM.

El tratamiento de los grandes volúmenes de información en las organizaciones actuales es una necesidad de vital importancia. El almacenamiento de datos (*data warehousing*) aporta una excelente aproximación para la transformación de datos en información real y útil que soporte la toma de decisiones y permita ejecutar procesos de *business intelligence*. Uno de los más importantes aspectos del *data warehousing* son los metadatos. Los metadatos son usados para construir, mantener, gestionar y usar los almacenes de datos. Desafortunadamente, existen en el mercado muchos gestores de datos y herramientas de análisis que han dado como resultado la existencia de muchas representaciones y tratamientos de metadatos. La solución al problema de tener diferentes metadatos y diferentes metamodelos, sería proporcionar un repositorio único de metadatos que implemente un metamodelo simple para todos los metadatos en una organización. En definitiva, un estándar de intercambio de almacenes de metadatos.

CWM responde a esta necesidad y provee de un marco de trabajo para la representación de metadatos sobre datos fuentes, datos objetivos, transformaciones y análisis y procesos y operaciones que crean y mantienen almacenes de datos y proveen una línea de información sobre su uso.

Dentro de la gestión del *data warehousing*, otro elemento importante a tener en cuenta es la gestión de la seguridad y auditoría de los datos. Curiosamente, CWM no proporciona constructores de modelado que permitan representar la seguridad de datos tales como, los derechos de acceso para usuarios o roles, pero sí proporciona un mecanismo de extensión que permita extender el metamodelo relacional de CWM para representar las medidas de seguridad y auditoría de los almacenes de datos a nivel lógico [Soler 2007].

El metamodelo CWM consiste en un número de submetamodelos que representan un almacén común de metadatos en las siguientes áreas de interés para los procesos de *data warehousing* y de *business intelligence* :(ver Figura 3-6 extraída de [OMG 2003b])

- Recursos de datos – incluye metamodelos que representan recursos de orientación a objetos, relacional, registros, multidimenional y XML.
- Análisis de datos – incluye metamodelos que representan transformaciones de datos, OLAP (*On-line Analytical Processing*), *data mining*, visualización de información y nomenclatura de negocio.
- Mantenimiento de almacenes – incluye metamodelos que representan procesos de almacenes y resultados de operaciones de almacenamiento.

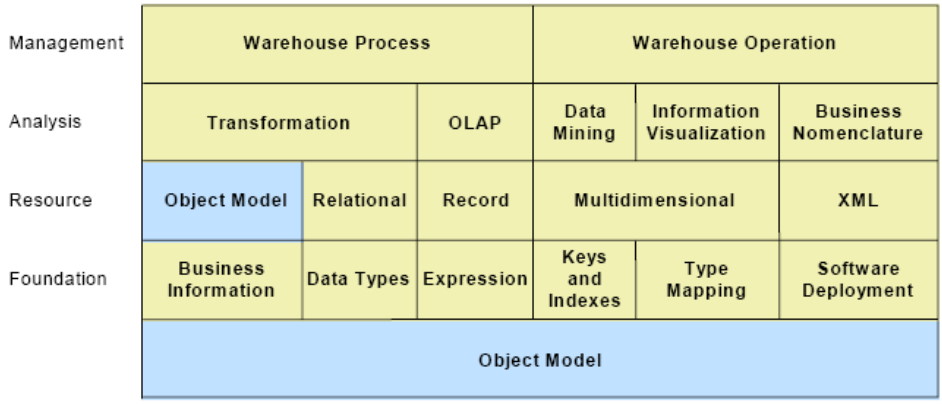

Figura 3-6. El metamodelo CWM

El metamodelo de CWM usa paquetes y una estructura jerárquica de paquetes para controlar la complejidad, fomentar entendimiento y soportar la reutilización. Los elementos del modelo siguen la estructura de paquetes indicada en la Figura 3-6: *ObjectModel package, Foundation package, Resource package, Analysis package y Management package*.

3.6 *Query, Views and Transformations* (QVT)

QVT es un lenguaje que permite definir transformaciones entre modelos cuyos lenguajes han sido definidos basados en MOF. Los requerimientos impuestos por OMG a la hora de definir el lenguaje de transformación QVT son las que le dan su nombre [Kleppe 2005]:

- *Query*: el lenguaje debe permitir la creación modelos de consultas sobre elementos de los modelos, de manera que permita seleccionar y filtrar los elementos que sirvan de entrada para una transformación.

- *View*: el lenguaje debe permitir la creación de vistas de metamodelos MOF sobre los que definen las transformaciones.
- *Transformation*: el lenguaje debe permitir la definición de las transformaciones desde un metamodelo fuente a un metamodelo destino, ambos basados en MOF.

La especificación de QVT tiene una naturaleza hibrida declarativa – imperativa [OMG 2008a], de manera que proporciona dos lenguajes de transformación, uno de naturaleza imperativa y otro de naturaleza declarativa. La parte declarativa está dividida en una arquitectura de dos niveles que proporciona un marco de trabajo para la ejecución de la semántica de la parte imperativa.

Los dos niveles de la arquitectura declarativa mostrados en la Figura 3-7 ([OMG 2008a]), se definen a continuación:

- Un lenguaje de relaciones (*Relations*) que soporta comparación de patrones (*pattern matching*) de objetos complejos y permite crear plantillas (*templates*) de objetos. Es una especificación declarativa de relaciones entre modelos MOF. Así mismo, realiza una trazabilidad de los objetos involucrados en la transformación de forma implícita.
- Un metamodelo *Core* y un lenguaje definido mediante una mínima extensión de EMOF y OCL, soporta la comparación de patrones sobre un conjunto plano de variables mediante la evaluación de condiciones sobre esas variables.

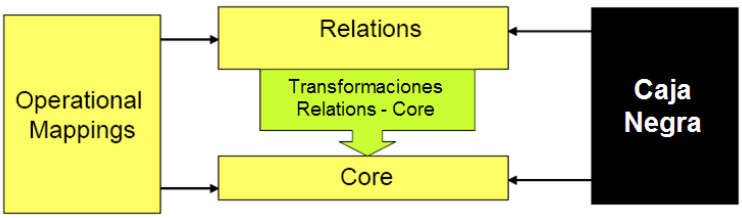

Figura 3-7. Relaciones entre metamodelos QVT.

Además de los lenguajes *Core* y *Relations*, que personalizan la misma semántica en dos diferentes niveles de abstracción, hay dos mecanismos para invocar implementaciones imperativas de transformaciones desde *Relations* y *Core*: un lenguaje estándar llamado *Operational Mappings* y una implementación mediante operaciones MOF de caja negra (*Black Box*), no estándar. Mediante estos lenguajes se define, para cada relación, que mantiene las trazas entre los elementos relacionados en la transformación.

Operational mappings proporciona un estándar que provee implementaciones imperativas. Es una extensión de OCL con un estilo procedimental y una sintaxis concreta parecida a los lenguajes imperativos. Se usa para implementar relaciones cuando es difícil hacer especificaciones puramente declarativas.

Las implementaciones de caja negra permiten realizar implementaciones de partes de una transformación de forma opaca, utilizando otro tipo de lenguajes. En determinadas situaciones puede ser más sencillo recurrir al uso de determinadas librerías de dominios específicos, que definan transformaciones de gran complejidad, muy difíciles de expresar mediante OCL. Esto puede resultar peligroso, ya que la implementación podría tener acceso a las referencias de los objetos de los modelos, pudiendo manipularlos arbitrariamente.

En [Koch 2006] se propone una aproximación mediante transformaciones QVT, para obtener un método que construya modelos de diseño desde especificaciones de requisitos de sistemas web. Un ejemplo ilustrativo del uso de QVT *Relational*, se muestra en [Gutiérrez 2008], donde se propone una transformación de casos de uso textuales (mediante un metamodelo definido por los

autores) a diagramas de actividad (UML). En la Figura 3-8 ([Koch 2006]) se muestra el código necesario para realizar la transformación de un requerimiento funcional a una actividad.

```
transformation FR_2_AD (frm: FunctionalRequirementMetamodel,
                        adm: ActivityDiagramMetamodel){

key Activiy(name);
key Action(effect, incoming, outgoing);
key ControlFlow(source, target);
key ActivityPartition(name);

top relation FunctionalRequirement_2_Activity {
    checkonly domain frm fr: FunctionalRequirement {
        name = fr_name,
        precondition = pre,
        postcondition = pos
    };
    enforce domain adm a: Activity {
        name = fr_name,
        precondition = pre,
        postcondition = pos
    };
    where {
        Actor_2_ActivityPartition (fr, a);
    }
}
```

Figura 3-8. Transformación de requerimiento funcional a actividad.

3.7 MOF *Model to Text*

Al igual que QVT es un estándar definido para especificar transformaciones entre modelos, *MOF Model to Text* (mof2text) [OMG 2008b] es un estándar de OMG que permite definir transformaciones desde un modelo a código en una representación del mismo basada en texto: código, listados, informes, etc.

Un camino intuitivo para conseguir este requisito es mediante una aproximación basada en plantillas (*templates*), donde el texto a generar es especificado como un conjunto de plantillas de texto que se parametrizan con elementos del modelo. Se especifican plantillas de texto con la transformación de los elementos que se van a extraer de los modelos. Las plantillas, básicamente están compuestas por expresiones especificadas sobre las entidades del metamodelo mediante consultas. Éstas inicialmente seleccionan y extraen valores de los modelos, que posteriormente son convertidos en fragmentos de texto utilizando las expresiones del lenguaje.

En la Figura 3-9 ([OMG 2008b]) se muestra un ejemplo de trasformación mediante plantillas. Partiendo de una plantilla (a) que especifica la generación de una clase Java desde una clase UML, se puede obtener el código correspondiente de la clase `Employee` (b) del diagrama indicado en la Figura 3-10 ([OMG 2008b]).

```
[template public classToJava(c : Class)]
class [c.name/]
{
      // Constructor
      [c.name/]()
      {
      }
}
[/template]
```

⇒

```
class Employee
{
     // Constructor
     Employee()
     {
     }
}
```

Figura 3-9. (a) plantilla de clase UML a clase Java (b) ejemplo de clase transformada

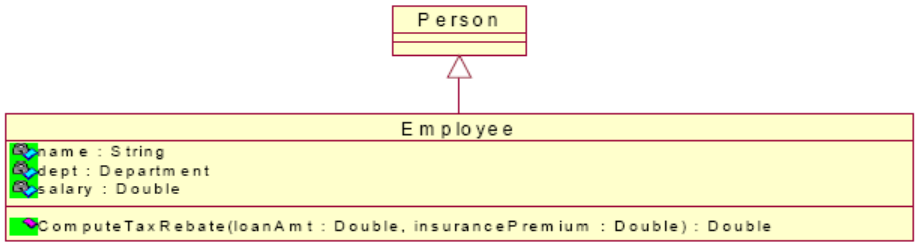

Figura 3-10. Ejemplo de diagrama de clases

Las plantillas se pueden componer entre ellas para conseguir transformaciones complejas. Cuando las transformaciones son muy extensas, se estructuran mediante módulos teniendo sus partes privadas y públicas. Una plantilla puede invocar a otras, insertando en el lugar donde se invocan, el texto generado por la plantilla invocada. Por ejemplo, una plantilla que transforme una clase a Java, puede invocar a su vez a una plantilla que transforme un atributo en Java. En este caso será necesario recorrer la lista de atributos de una clase, por lo que las plantillas podrán definir procesos iterativos sobre un conjunto de elementos, mediante un bloque *for*. En la Figura 3-11 [OMG 2008b] se muestra un ejemplo de una plantilla (a) y el resultado (b) de una transformación de este tipo.

```
[template public classToJava(c : Class)]
class [c.name/]
{
    // Attribute declarations
    [for(a : Attribute | c.attribute)]
    [a.type.name/] [a.name/];
    [/for]
    // Constructor
    [c.name/]()
    {
    }
}
[/template]
```

⇒

```
class Employee
{
     // Attribute declarations
     String name;
     Department dept;
     Double salary;
     // Constructor
     Employee()
     {
     }
}
```

Figura 3-11. (a) Plantilla que invoca a otra (b) ejemplo de clase trasformada

En definitiva, existen una serie de mecanismos que permiten recorrer la navegabilidad de los modelos, controlar las declaraciones de tipos, mantener la trazabilidad de los elementos trasformados o el destino del fichero en el que se va a almacenar el código.

3.8 *Object Constraint Language* (OCL)

OCL es un lenguaje formal usado para describir expresiones sobre modelos UML [OMG 2006b]. Estas expresiones suelen utilizarse con el fin que 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).

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.

Un diagrama UML normalmente no está lo suficientemente definido para mostrar todos los elementos relevantes de una especificación. Entre otros aspectos, existe la necesidad de describir restricciones sobre elementos del modelo. Normalmente las restricciones se escriben en lenguaje natural y están llenas de ambigüedades. Para poder escribir restricciones no ambiguas, se debía definir un lenguaje formal que lo permitiera, pero la desventaja de utilizar un lenguaje tradicional es que resultaría demasiado complejo de aplicar a los sistemas de modelado.

OCL se desarrolló para resolver esta dificultad. Es un lenguaje formal fácil de escribir y de leer. Se desarrolló como un lenguaje de modelado de negocios, que garantiza que no provoca efectos sobre los modelos. Cuando una expresión OCL se evalúa sólo devuelve el valor solicitado en la misma. No modifica nada del modelo y por lo tanto, no cambia el estado del sistema. Por esta razón, OCL no es un lenguaje de programación ya que no se pueden invocar procesos o actividades que no sean consultas.

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 *Classifier* [OMG 2007a] 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:

- 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 poscondiciones 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.

Se han estudiado varios ejemplos [Zubcoff 2008][Akehurst 2008][Cabot 2008][Cabot 2009], pero uno representativo, es el presentado en la Figura 3-12 [OMG 2006b]. Partiendo de este se pueden definir, mediante OCL, varias restricciones concretas:

```
1. context Company inv:
    Self.numberOfEmployees > 50

2. context Person::getCurrentSpouse() : Person
    pre: self.isMarried = true
    body: self.mariages->select( m | m.ended = false ).spouse

3. context Person::income : Integer
    init: parents.income->sum() * 1%          -- paga semanal
    derive: if underAge
           then parents.income->sum() * 1%    -- paga semanal
           else job.salary                    -- sueldo de un trabajo
           endif
```

Cada expresión OCL se escribe en el contexto de una instancia de un tipo específico, por eso todas las expresiones se inician con la palabra reservada `context`. En el primer ejemplo muestra, dentro del contexto de la clase `Company`, una restricción para el atributo `numberOfEmployees`, en la que no se puede superar los 50 empleados. En este ejemplo y los siguientes, la palabra `self` se refiere siempre a la clase del contexto.

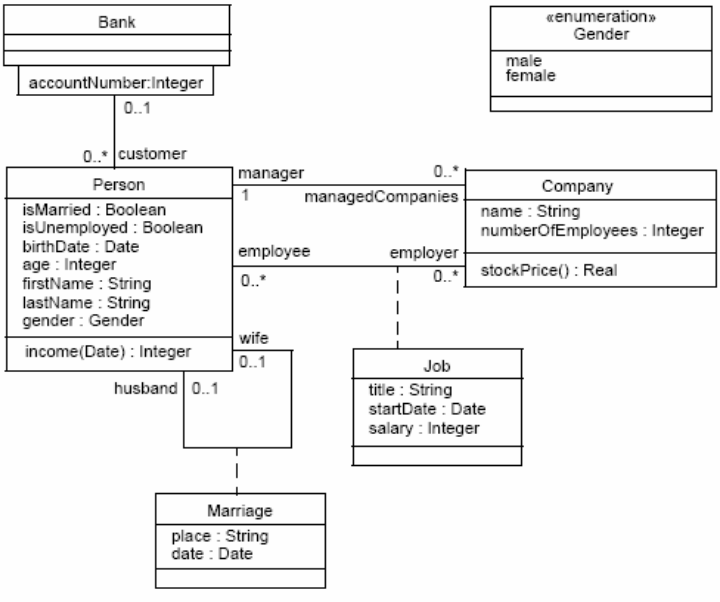

Figura 3-12. Ejemplo de diagrama de clases

En el segundo ejemplo se define una función sobre la clase persona, que devuelve el cónyuge de una persona concreta. Para ello se define un precondición, mediante la cláusula `pre`, que debe cumplir la persona para poder tener un cónyuge. La persona debe estar casada. La especificación del cuerpo de la función se define tras la cláusula `body`.

Mediante expresiones OCL también se pueden indicar los valores iniciales o derivados de un atributo o una asociación. El ejemplo mostrado en tercer lugar, muestra que el valor inicial (`init`) de los ingresos de una persona es su paga semanal, que se define en base al sueldo de sus progenitores. En este caso el 1% de la suma de los ingresos de ambos. Este sería su valor inicial, pero después de podría obtener (`derive`) su renta en base a su edad. Mientras no sea mayor de edad mantendrá su asignación semanal. Una vez que la supere, sus rentas corresponderán con el sueldo del puesto de trabajo que desempeñe.

3.9 *XML Metadata Interchange* (XMI)

XMI es otro estándar OMG que permite el intercambio de modelos definidos mediante MOF entre diferentes herramientas. Permite expresar en XML cualquier modelo o metamodelo que se haya definido en MOF. Se utiliza en herramientas de integración, repositorios, aplicaciones y almacenes de datos, ya que XMI proporciona reglas por las que permite expresar en XML cualquier modelo o metamodelo que se haya definido en MOF [OMG 2007c].

Mediante el estándar XMI, se define cómo se deben crear los documentos XML así como los esquemas XML que se pueden utilizar su validación. Cada documento XML que defina un modelo debe contener los elementos requeridos en la especificación XMI, los elementos que contienen los datos del modelo representado y, opcionalmente, elementos que contienen metadatos que representan extensiones del modelo. Determinada información del modelo puede ser codificada en esquemas XML que permitan realizar validaciones de un documento XMI. Mediante los esquemas de XML se puede validar sobre todo la sintaxis de los elementos del modelo que se definen en el documento y algunos aspectos de la semántica. También se pueden validar extensiones del modelo

ya que éstas de definen como elementos del modelo y por lo tanto pueden estar incluidos en el esquema como parte de los elementos a verificar.

Sobre el ejemplo del diagrama de clases de un modelo GIS de la Figura 3-13, se muestra en el código expuesto a continuación, la representación del esquema relativa a la clase `Road`, como extracto de la definición completa del esquema que se puede consultar en [OMG 2007c].

```
../..

<xsd:annotation>
   <xsd:documentation>CLASS: Road</xsd:documentation>
</xsd:annotation>

<xsd:complexType name="Road">
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="classification" type="xsd:string" nillable="true"/>
      <xsd:element name="number" type="xsd:string" nillable="true"/>
      <xsd:element name="linearGeometry" type="xsd:string" nillable="true"/>
      <xsd:element ref="xmi:Extension"/>
   </xsd:choice>
   <xsd:attribute ref="xmi:id"/>
   <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
   <xsd:attribute name="classification" type="xsd:string" use="optional"/>
   <xsd:attribute name="number" type="xsd:string" use="optional"/>
   <xsd:attribute name="linearGeometry" type="xsd:string" use="optional"/>
</xsd:complexType>

../..
```

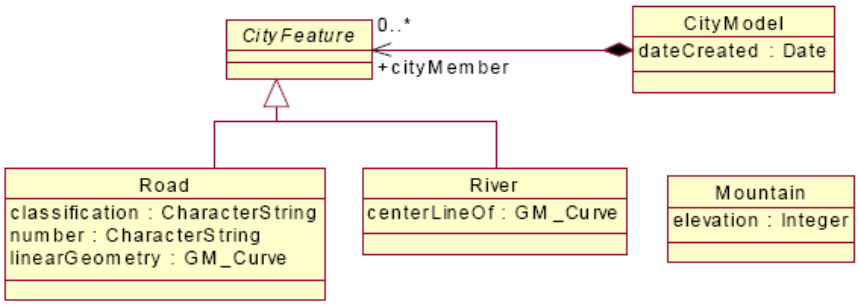

Figura 3-13. Ejemplo de Modelo GIS.

Para poder utilizar esquemas XML, como instrumento de validación en intercambios mediante ficheros XMI, todos los elementos y atributos incluidos en la especificación XMI tienen que estar importados en el esquema XML. Los constructores del modelo tienen que tener sus declaraciones de elementos correspondientes y en ocasiones algunos constructores pueden tener declaraciones de tipo complejas. Por último, los elementos que representan extensiones, también deben ser definidas en el esquema.

Uno de los logros importantes de XMI es que no es necesario siempre transmitir modelos completos, sino que no se pueden trasmitir modelos parcialmente. En estos casos hay que solventar problemas de sincronización, ya que esos modelos incompletos pueden transmitir elementos modificados que, al recibirse pueden afectar al modelo receptor. En función de las diferencias encontradas, hay que ser capaz de eliminar, añadir o reemplazar los elementos del modelo receptor.

El intercambio de información entre diferentes herramientas puede verse dificultado por los diferentes requisitos de identificación (ID) que tenga cada una de ellas. El intercambio entre los ID de los elementos de los diferentes modelos, se realiza mediante las etiquetas UUID (*Universal Unique Identifier*). Las herramientas pueden utilizar como ID uno propio, pero preservando el UUID de XMI para garantizar el correcto intercambio de información. En relación a esta

dificultad, [Lundell 2006] muestra un estudio sobre el intercambio de modelos de información mediante XMI en herramientas heterogéneas y cómo XMI 2.0 resuelve muchos de las carencias de versiones anteriores.

3.10 Estándares OMG y la arquitectura de metamodelado 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 definidos en los apartados anteriores, se van a ubicar básicamente en las capas de meta-metamodelo (M3) y metamodelo (M2), aunque alguno de ellos pueden 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 representada en varios niveles a la vez.

Para no complicar en exceso las ilustraciones, se ha decidido plasmar los diferentes estándares en varias figuras, utilizando un criterio basado en la relación de los conceptos MDA que representa cada uno de ellos.

Como se ha comentado más arriba, el núcleo de toda la arquitectura dirigida por modelos es la *InfrastructureLibrary*, 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 3-14). El paquete *UML Profiles* pertenece a la *InfrastructureLibrary*, 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). Cabe destacar que, tanto los modelos UML como los extendidos mediante un perfil, son instancias de UML, ya que los perfiles extienden al metamodelo de UML, sin que por ello transforme su metamodelo.

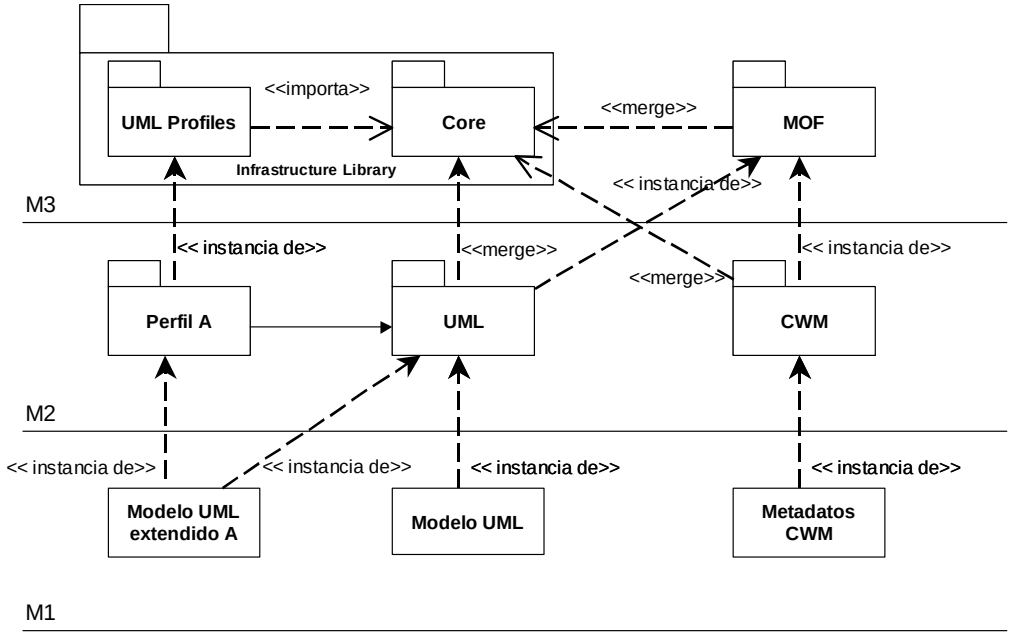

Figura 3-14. Infraestructura común

El estándar XMI (Figura 3-15), 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.

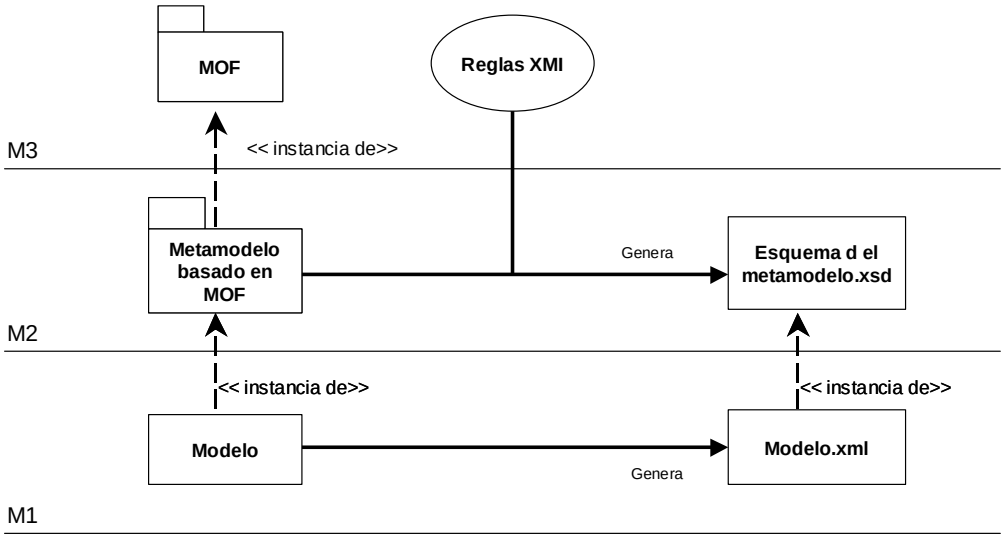

Figura 3-15. Intercambio entre modelos

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 y M2). 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. En la Figura 3-16 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.

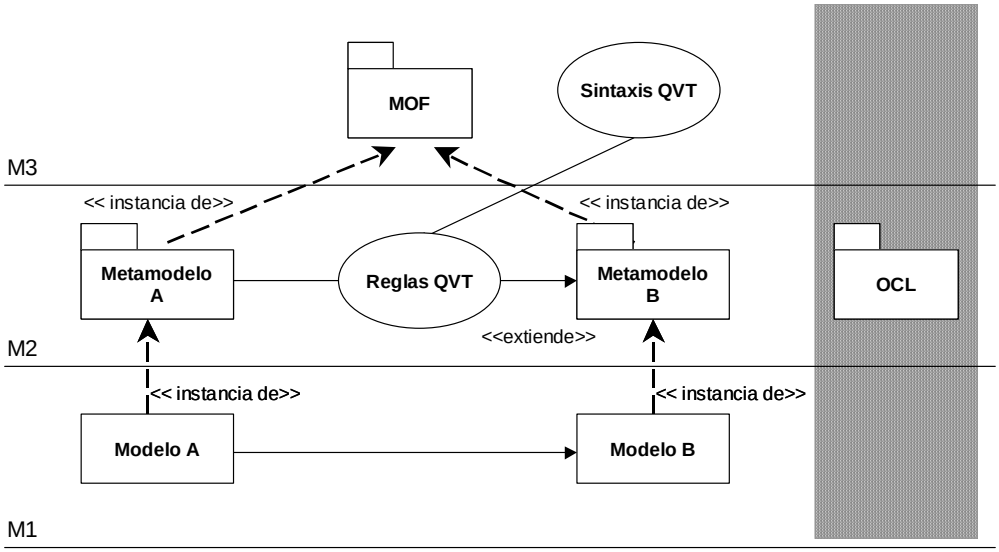

Figura 3-16. Transformación de modelos

4. PROYECTOS ECLIPSE

4.1 Eclipse

Eclipse es un proyecto de desarrollo de software de código abierto cuyo principal objetivo es proporcionar una plataforma de herramientas altamente integrada [Budinsky 2003]. El trabajo en Eclipse consiste en un núcleo central del proyecto, que incluye un marco de trabajo para integración de herramientas, y un entorno de desarrollo Java construido basado en él. Hay otros proyectos que extienden el marco de trabajo para soportar tipos específicos de herramientas y entornos de desarrollo. Los proyectos en Eclipse, se implementan en Java y pueden ejecutarse en diferentes sistemas operativos.

El proyecto Eclipse lo soporta eclipse.org, un consorcio de compañías que han asumido el compromiso de proveer un soporte en términos de tiempo, técnica, tecnología y conocimiento al proyecto.

Los proyectos operan con distintos participantes con una definición clara de funciones y responsabilidades. Hay usuarios, desarrolladores y un comité de gestión. El trabajo de desarrollo en Eclipse se divide en tres proyectos fundamentales: el proyecto *Eclipse*, el proyecto *Tools* [Eclipse Tools] y el proyecto *Technology* [Eclipse Tech].

El Proyecto Eclipse tiene los componentes básicos necesarios para desarrollar utilizando Eclipse. Sus componentes son fijos y se descargan en un solo bloque denominado Eclipse SDK (*Software Development Kit*). Los componentes de los otros dos proyectos son utilizados para fines específicos y generalmente son independientes, por lo que se descargan individualmente. Constantemente se están agregando nuevos componentes a los proyectos *Tools* y *Technology*.

El proyecto *Tools* define y coordina la integración de diferentes categorías de herramientas basadas en la plataforma Eclipse. Dentro del contexto de este proyecto se encuentra el *Graphical Editing Framework* (GEF) y los modelos basados en procesos, como el *Eclipse Modeling Framework* (EMF). El proyecto *Technology* proporciona una oportunidad a los investigadores, educadores y el mundo académico, para participar en la evolución constante del proyecto Eclipse. Es un proyecto que no contiene código, pero abarca una gran variedad de temas.

De forma general, los proyectos siguen un ciclo de vida definido principalmente por las siguientes fases [Eclipse Ciclo]:

- *Propuesta*. Define y refina el interés sobre un determinado proyecto.
- *Incubación*. Si en la fase de propuesta no se desecha el proyecto, se crea el proyecto pasando a esta fase, en donde se define la funcionalidad completa basada en código abierto, mediante diferentes revisiones.
- *Incubación, no conforme*. Si no supera la fase de incubación
- *Madurez*. Se pasa a la fase de madurez cuando se demuestra que es un proyecto completamente abierto con procesos abiertos y transparentes, tienen una participación activa de la comunidad Eclipse y una buena calidad tecnológica. Una vez alcanzada la madurez se van creado diferentes versiones del proyecto hasta que se decida su terminación y se archive.

La Tabla 4-1 [Eclipse PRO] presenta la relación de los estados de todos los subproyectos de EMP. En ella se especifica el estado actual en el que se encuentra cada uno de ellos.

Proyecto	Estado
Eclipse Modeling Project (EMP)	Madurez
Modeling Amalgamation Project (MAP)	Incubación - no conf.
Generative Modeling Technologies (GMT)	Incubación - no conf.
Graphical Modeling Framework (GMF)	Madurez
Model Development Tools (MDT)	Incubación
Eclipse Model Framework Technology (EMFT)	Incubación
Model-to-Model Transformation (M2M)	Incubación
Model To Text (M2T)	Incubación
Eclipse Modeling Framework (EMF)	Madurez
Presentation Modeling Framework (PMF)	Incubación - no conf.
Textual Modeling Framework (TMF)	Incubación

Tabla 4-1. Estados de los proyectos Eclipse EMP

4.2 *Eclipse Modeling Project*

El Proyecto de Modelado de Eclipse (*EMP – Eclipse Modeling Project*) [Eclipse EMP][Gronback 2009], se centra en la evolución y promoción de tecnologías de desarrollo basadas en modelos, dentro de la comunidad Eclipse, proporcionando un conjunto unificado de marcos de trabajo de modelado, de herramientas y estándares de implementación. Este proyecto se divide a su vez en un conjunto de subproyectos que van a ir desgranando los diferentes aspectos que proporciona la aproximación MDA. La descomposición en subproyectos es la siguiente:

- *EMF (Eclipse Modeling Framework)*: proporciona un marco de trabajo de modelado y un sistema de generación de código para construir herramientas y otras aplicaciones basadas en un modelo de datos estructurado [Eclipse EMF].
- *EMFT (EMF Technology)*: pretende extender y complementar el subproyecto EMF [Eclipse EMFT].
- *GMF (Graphical Modeling Framework)*: provee una infraestructura de componentes para el desarrollo de editores gráficos basados en *Graphical Editing Framework* (GEF) [Eclipse GEF] y EMF. Estos dos proyectos han servido como inspiración para la creación del proyecto GMF [Eclipse GMFa].
- *GMT (Generative Modeling Technologies)*: proyecto de investigación que se centra en producir prototipos en el área de la ingeniería dirigida por modelos [Eclipse GMT].
- *MDT (Model Development Tools)*: este proyecto se centra en proveer a la industria de estándares de metamodelado y de un conjunto de herramientas para el desarrollo de modelos basados en estos estándares [Eclipse MDT]. Bajo este proyecto se encuentran herramientas como OCL, UML2, UML2 *Tools* y XML *Schemas Definition* (XSD) entre otras.
- *M2M (Model to Model Transformation):* marco de trabajo extensible para lenguajes de transformación entre modelos. Las transformaciones se ejecutan mediante motores de transformación que se integran dentro de la infraestructura [Eclipse M2M]. Dentro del ámbito del proyecto se han creado tres motores de transformación; *Atlas Transformation Language* (ATL), *Procedural* QVT (*Operational*) y *Declarative* QVT (*Core & Relational*).
- *M2T (Model to Text Transformation):* proyecto que se centra en la transformación de modelos en artefactos textuales. Provee estándares de lenguajes de transformación de modelos a texto para la industria, herramientas de desarrollo para estos lenguajes y una infraestructura común para los mismos [Eclipse M2T].
- *TMF (Textual Modeling Framework):* proporciona herramientas y marcos de trabajo para construir sintaxis textuales y sus correspondientes editores basados en EMF [Eclipse TMF].

Existe otro proyecto dentro de EMP, que pretende mejorar el empaquetado, la integración y la usabilidad de todos los componentes de modelado del proyecto EMP. Este proyecto se llama *Modeling Amalgamation Project* (MAP) que proporciona un lenguaje específico de dominio basado en Eclipse para el desarrollo dirigido por modelos [Eclipse MAP]. La mayoría de estos subproyectos están en fases iniciales [Eclipse PRO], según el ciclo de vida de los proyectos Eclipse [Eclipse Ciclo]. Sólo dos de ellos se encuentran en una fase madura: EMF y GMF. Por esta razón, este capítulo se va a centrar principalmente en estos dos proyectos. Además, también se detallarán las herramientas más destacadas que se integran en el proyecto *Model Development Tools* (MDT) cuyo objetivo es proveer implementaciones de los metamodelos estándares de la industria y herramientas ejemplares para desarrollar modelos basados en dichos metamodelos [F. de Alba 2009].

4.3 *Eclipse Modeling Framework* (EMF)

4.3.1 Proyecto EMF

El proyecto *Eclipse Modeling Framework* [Steinberg 2008] es una herramienta creada para facilitar el modelado de sistemas y la generación automática de código Java mediante patrones y buenas prácticas de diseño. EMF proporciona herramientas y entornos de ejecución que permiten generar un conjunto de clases del modelo. Los modelos pueden ser especificados mediante anotaciones Java [SUN Annotations], documentos XML o herramientas de modelado como *Rational Software Architect* [IBM RSA], y después ser importados a EMF. Con EMF se consigue tener un modelo de representación de alto nivel que permite combinar estos tres lenguajes. Los desarrolladores pueden definir un modelo en una de estas alternativas y mediante EMF conseguir generarlo en las otras dos. Lo más importante es que EMF suministra las bases para la interoperabilidad con otras herramientas y aplicaciones basadas en EMF.

EMF permite de una forma ágil y rápida, transformar modelos en código Java eficiente, correcta y fácilmente parametrizable [Eclipse SDK]. Una vez especificado el modelo en EMF, el generador EMF puede crear el conjunto de clases correspondientes. Construir aplicaciones utilizando EMF proporciona a su vez otros beneficios, tales como notificaciones de modificaciones de modelos, serialización basada en esquemas XML, un marco de trabajo para validación de modelos y un interfaz para la manipulación de objetos EMF eficiente.

El núcleo básico de EMF lo compone el subproyecto *EMF Core*, que contiene: (i) el metamodelo a partir del cual podemos describir nuestros modelos, (ii) un entorno de edición que nos dota de clases reutilizables para construir editores para nuestros modelos y (iii) la herramienta para la generación de código a partir del modelo creado por el usuario.

Este apartado se va a centrar en el estudio de *EMF Core*. No obstante, antes de entrar en el detalle del mismo, se van a enumerar los otros subproyectos constituidos alrededor de este núcleo principal [Alonso 2009]:

- *Service Data Object* (SDO): es un marco de trabajo que se encarga de aplicar las mejores prácticas y patrones de diseño, así como la simplificación de nuestros modelos de datos J2EE.
- *Connected Data Object* (CDO): se encarga de dar soporte a la generación distribuida de modelos y la gestión de la persistencia de dichos modelos. Esta gestión incluye la notificación de los cambios al resto de clientes que procederán a actualizar las partes del modelo que quedan afectadas por el cambio inicial del modelo. Este proyecto usa la tecnología del subproyecto Net4j.

- *Net4j*: este subproyecto consta de un sistema cliente/servidor originalmente creado para facilitar el compartir y la persistencia de modelos EMF. Actualmente se usa para multiplexar diversos protocolos de usuario a través del la misma conexión.
- *Teneo*: Es una solución dentro de EMF para gestionar la persistencia de datos. Puede utilizar *Eclipselink* [Eclipse link] o *Hibernate* [RED Hibernate]. Soporta la creación automática de mapeos de los esquemas de base de datos utilizados y de las relaciones establecidas. Los objetos EMF se pueden almacenar y recuperar mediante consultas escritas en *Hibernate Query Language* (HQL) o *Enterprise Java Beans Query Language* (EJB-QL).
- *Model Query*: este subproyecto da la capacidad para recuperar modelos guardados anteriormente, mediante un lenguaje estructurado de datos específico para EMF.
- *Model Transaction*: da soporte a la lectura/escritura de modelos EMF en múltiples hilos mediante la encapsulación de los cambios en transacciones, se encarga de verificar la integridad de los modelos, soporte para notificación de eventos en modo *batch* y gestión de versiones de los modelos.
- *Validation Framework*: este subproyecto se encarga de dotar a los modelos de diversas características que modifican el comportamiento del meta-modelo básico de EMF a partir del cual se generan. Entre las modificaciones que podemos realizar se encuentran las definición de nuevas restricciones, personalizar algoritmos para incluir nuevas estrategias para generar código, inclusión de *listeners* a los eventos de validación, etc.

EMF se divide en dos marcos de trabajo fundamentales: *EMF Core* y *EMF.Edit*. *EMF Core* incluye un metamodelo (*ECore*) para describir modelos y un motor de ejecución para los modelos que soporta la creación de las clases Java de un modelo. *EMF.Edit* proporciona clases genéricas reutilizables para construir editores para modelos EMF.

4.3.2 *Ecore*

EMF empezó como una implementación de la especificación de MOF, pero ha evolucionado basado en la experiencia que la comunidad Eclipse ha adquirido al utilizarlo, para implementar un amplio conjunto de herramientas. EMF se puede considerar como una implementación Java eficiente del subconjunto central (o *core*) de la interfaz de MOF. No obstante, para no llevar a confusiones el equivalente al *Core* de MOF, en EMF se le denomina *Ecore*, el cual sería equiparable al paquete EMOF de la especificación MOF. *Ecore* es por tanto un modelo mediante el que representar modelos en EMF. Es en sí mismo, un modelo EMF, así que es de hecho, su propio metamodelo. En definitiva, es un meta-metamodelo [Budinsky 2003]. Con él se van a poder definir metamodelos, así como funcionalidades para definir los cambios en el modelo, para gestionar su persistencia mediante la exportación del formato XMI y una API eficiente para manipular los objetos EMF de forma genérica [Alonso 2009].

En la Figura 4-1 (extraída de [Budinsky 2003]), se muestra un subconjunto del modelo del *Ecore*, que además se encuentra simplificado. En el modelo completo de *Ecore*, las clases *EClass*, *EAttribute* y *EReference* comparten una clase base llamada *ENamedElement*, la cual define el atributo *name* que aparece en las clases de forma explícitamente.

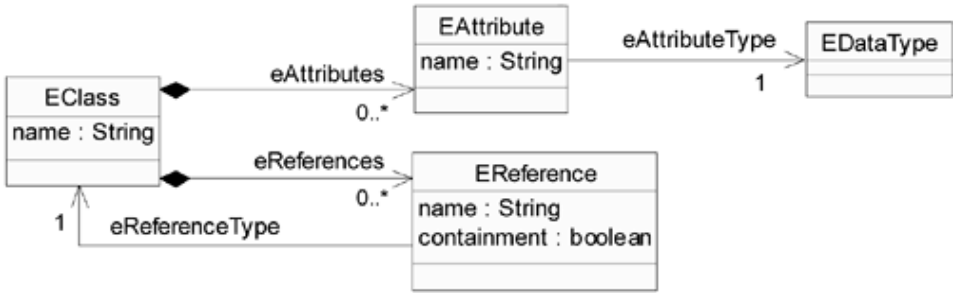

Figura 4-1. Diagrama de clases simplificado del metamodelo *Ecore*

Tal y como se muestra en la figura, se necesitan cuatro clases *ECore* para representar los modelos [Magariño 2009]:

- *EClass*: se usa para representar las clases. Contiene *EAttributes* y *EReferences*. Las instancias de *EClass* pueden extender otras instancias de *Eclass*, heredando los atributos.
- *EAttribute*: se usa para representar los atributos modelados. Contiene valores de tipos primitivos (enteros, cadenas de caracteres, etc), representados mediante la clase *EDataType*.
- *EReference*: representa relaciones binarias entre dos instancias de *EClass*. La primera instancia de *EClass* contiene una instancia de *EReference* que señala a la otra instancia de *EClass*. Si la instancia de *EReference* es múltiple, quiere decir que se podrá instanciar varias veces en la *EClass* contenedora. Las relaciones pueden ser de dos tipos:
  - Contenedoras: el elemento que representa el contendor, incluye al elemento referenciado por la relación.
  - No contenedoras: el contenedor tiene el camino hacia el elemento referenciado por la relación.

Se puede apreciar que los nombres de las clases corresponden con términos UML. De hecho cabría preguntarse por qué UML *Core* no es el metamodelo de EMF o por qué EMF necesita su propio metamodelo. Esto es así porque EMF es un subconjunto de UML *Core*. UML soporta modelados mucho más ambiciosos de los que puede soportar EMF. Por ejemplo, permite modelar el comportamiento de una aplicación así como su estructura de clases.

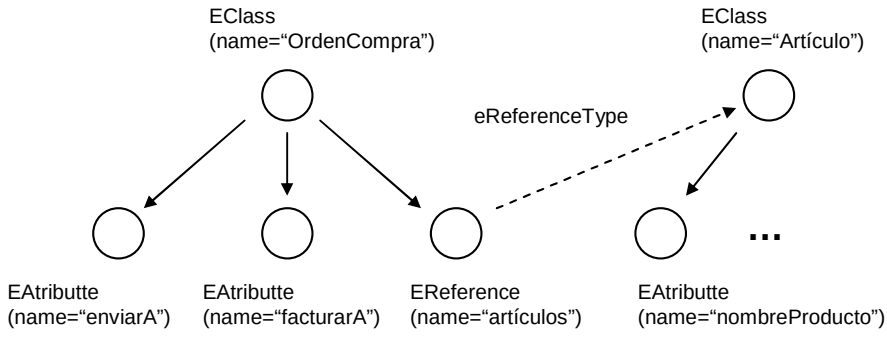

Figura 4-2. Ejemplo de instancias *Ecore*

Se pueden usar instancias de las clases definidas en *Ecore* para describir estructuras de un modelo de aplicación. Por ejemplo, se puede describir una clase de orden de compra como una instancia de *EClass* llamada `OrdenCompra`. Ésta contiene dos atributos (instancias de *EAttribute*) llamada `enviarA` y `facurarA` y una asociación (instancia de *EReference*) llamada `artículo`,

por cada *eReferenceType*, que sea igual a otra *EClass* llamada `Artículo`. Esto se muestra en la Figura 4-2, también de [Budinsky 2003].

4.3.3 Creación y edición de modelos

Como se ha comentado más arriba, EMF permite generar modelos desde diferentes lenguajes, permitiendo así a los desarrolladores elegir el lenguaje más asequible a sus conocimientos o experiencia: XML, Java o UML. Partiendo de uno de ellos, y mediante EMF, se podrían generar los modelos en los otros dos lenguajes, evitando tener que duplicar el trabajo.

Partiendo de las tres posibilidades, la forma de importar a EMF sería la siguiente:

- Partiendo de *Java Interfaces*, el generador EMF lo examinará y generará el modelo *Core*.
- Si se parte de un esquema XML, se generará de la misma manera.
- Si se parte de un modelo UML, se puede actuar de tres formas:
  - Exportar el modelo UML a XMI, de manera que se pueda importar así en EMF.
  - Importarlo directamente mediante un *wizard* que proporciona EMF para importar directamente desde *Rational Rose*.
  - Exportar directamente desde herramientas UML que permitan exportar a EMF.

Existe otra opción que es utilizar un editor que cree directamente el modelo en EMF. De esta manera se evitan los pasos de importaciones o exportaciones, que deban estar sincronizadas con el modelo original. Se pude utilizar el *Ecore editor* o el *Eclipse UML Graphical Editor*.

4.3.4 Generación de código

El beneficio más importante de EMF, como la aproximación MDA en general, es la capacidad de generar código de forma automática, aumentando así la productividad en el desarrollo. *EMF Core* facilita la generación automática del esqueleto de la aplicación y en algunos casos la implementación completa de partes de nuestros modelos. Para ello se tienen en cuenta las mejores prácticas y patrones de diseño más eficaces para resolver las distintas tareas con las que tratan los elementos de la arquitectura [Alonso 2009].

La generación del código Java, la realiza de una manera distinta en función de los elementos del modelo. Para las entidades, *EMF Core* crea un interfaz que hereda del interfaz *EObject* del metamodelo de *EMF Core* (*ECore*) que contiene todas las funciones y crea una clase que implementa dicho interfaz y a su vez hereda de la clase *EObjectImpl* de *ECore* (que también implementa el interfaz *EObject*).

Para las relaciones entre las entidades se genera el código de las funciones que generan dicha relación teniendo en cuenta sobre las cardinalidades de cada uno de los sentidos de estas relaciones para elegir el patrón de diseño más apropiado en cada caso.

EMF Core también trata de manera especial los atributos enumerados los que convierte en entidades, si bien es cierto que tienen funcionalidad limitada, pero forman parte del sistema de herencia del nuevo modelo y como tales pueden acceder a las factorías y paquetes que *EMF Core* genera automáticamente para modelo.

Como último paso falta tratar la herencia de clases dentro de los modelos, debido a que las clases generadas son clases Java, como tal no pueden tener herencia múltiple por eso en EMF *Core* se tiene que decidir qué clase base va a ser de la que se va a heredar y para el resto de las clases padre del modelo la nueva clase simplemente se limitará a implementar cada uno de sus interfaces.

Otra característica del generador automático de *EMF Core* es la posibilidad de personalizar la implementación generada automáticamente, sin que se pierdan los cambios si se decide modificar el modelo y volver a regenerarlo. Esto se puede de dos maneras [Eclipse SDK]:

- Eliminando del *javadoc*, el *tag* (`@generated`) que incluye EMF al generar el código. Cada vez que el generador encuentre un conflicto entre un método del modelo y del código y en éste no exista ese *tag*, entenderá que se debe respetar la versión modificada por el desarrollador.
- Modificando el nombre de los métodos, añadiéndole el sufijo `Gen`. El generador descartará aquellos métodos que en el modelo se llamen igual que el del código, pero sin el sufijo.

4.4 *Graphical Modeling Framework* (GMF)

4.4.1 Proyecto GMF

El proyecto *Graphical Modeling Framework* [Beca 2009][Eclipse GMFb] nace de la necesidad de establecer un puente entre *Eclipse Modeling Framework* y el *Graphical Editing Framework* con el objetivo de generar herramientas de edición gráfica. La función principal de este marco de trabajo es proporcionar a los desarrolladores la posibilidad de construir editores gráficos que facilite a los diseñadores el desarrollo de aplicaciones basadas en Eclipse. Permite crear editores propios para el modelado de diagramas UML, como por ejemplo un editor para un lenguaje específico de dominio. El resultado final será un *plug-in* que instalado en cualquier proyecto basado en la arquitectura de Eclipse, proporciona la capacidad para editar gráficamente un modelo que se ajuste a un metamodelo previamente especificado.

GMF se divide básicamente en dos grandes componentes. Un componente de instrumentación (*GMF Tooling*), mediante el cual se describen los aspectos de notación y semánticos y funcionalidades del editor gráfico, así como la parametrización del generador que se va a encargar de construir el código del mismo y generar el *plug-in* correspondiente. Este *plug-in* será ejecutado por el otro componente de GMF (*GMF Runtime*) que permitirá el uso del editor basado el GMF.

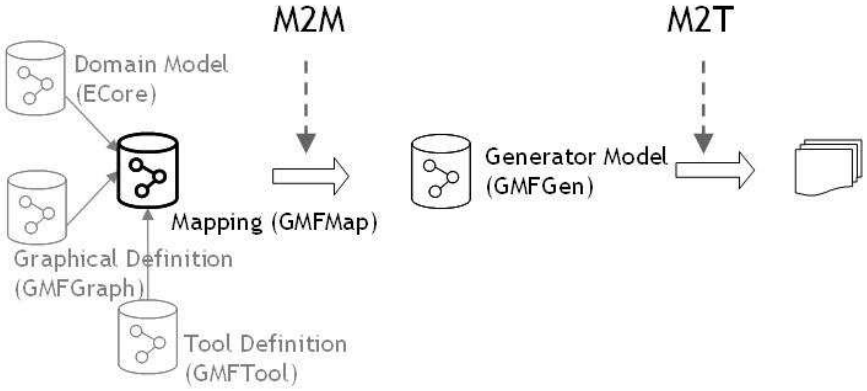

Figura 4-3. Arquitectura del desarrollo basado en GMF

La arquitectura para el desarrollo basado en GMF se detalla en la Figura 4-3 (extraída de [Beca 2009]) y consta de varios modelos en donde se definen el modelo de dominio, la definición gráfica de los elementos que van a componer el editor y la definición de las herramientas que van a estar asociadas a cada objeto (menús de contexto, paletas, barra de herramientas, etc).

Mediante el modelo de mapeo se especifica el enlace entre los tres modelos. Definiendo el modelo de mapeo, se usará como entrada para la transformación modelo a modelo que se ejecuta

para obtener el modelo generador. Este último permite generar el código que va a permitir ejecutar el editor, mediante la correspondiente transformación.

Los beneficios obtenidos con GMF son [Hunter 2007]:

- Proveer de una aproximación dirigida por modelos para la generación de editores gráficos
- Mantener la separación entre el dominio y las definiciones gráficas
- Proporcionar flexibilidad en la generación de salida, ya que facilita un *runtime* propio extensible y permite *runtimes* alternativos.
- Promueve el uso de lenguajes específicos de dominio

4.4.2 GMF *Tooling*

De lo explicado en la sección anterior se observa que los pasos a dar para construir un editor gráfico basado en GMF son [Tikhomirov 2008]: (i) diseñar un metamodelo (*domain model*), (ii) crear una representación gráfica de para los nodos y los enlaces (*graphical definition* + *tool definition*) y (iii) definir la estructura del diagrama (*mapping*) y generar el código mediante transformaciones sucesivas. El diagrama de la Figura 4-4 ([Kolovos 2009]) ilustra con algo más de detalle los componentes y modelos utilizados durante el desarrollo basado en GMF.

El núcleo de GMF es el concepto de la definición gráfica de modelo. Este modelo contiene información relacionada con los elementos gráficos que aparecen en el *runtime* basado en GEF, pero que no tiene una conexión directa con el modelo de dominio al que proporciona representación.

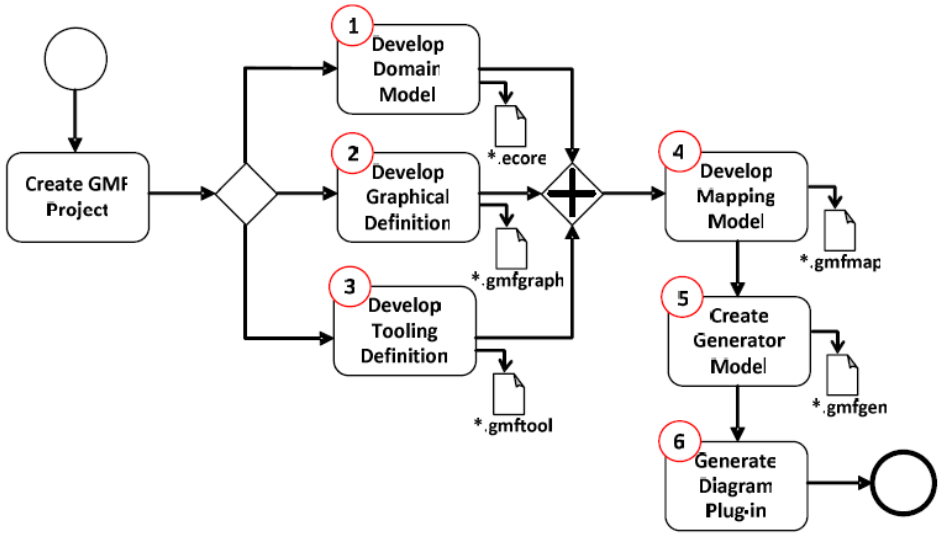

Figura 4-4. Componentes y Modelos GMF

El modelo de desarrollo de dominio define la información no gráfica gestionada por el editor. Define el metamodelo sobre el que se va a basar todo el desarrollo y se define en base a EMF. Se puede conseguir mediante (i) la importación de un meta-modelo *Ecore* de EMF, (ii) la utilización del editor estándar de EMF o (iii) mediante el uso del editor *Ecore* que está incorporado en GMF.

El modelo de definición gráfica define los elementos gráficos que se van a visualizar con el editor tales como figuras, nodos, conexiones, etiquetas, etc. El modelo de definición de herramientas permite definir los elementos como la paleta, cuadro de herramientas, menú principal, *pop-ups* y resto de acciones que se va a asociar a cada elemento.

Mediante el modelo de mapeo define la correlación que existe entre los elementos del modelo de dominio y los elementos gráficos. Los elementos del dominio vienen representados en el diagrama *.ecore* y los elementos gráficos vienen representados en los diagramas *.gmfgraph* y *.gmftool*. El mapeo por lo tanto, debe definirse mediante la relación entre los tres diagramas. Uno de los logros de GMF es la reutilización de las definiciones gráficas para varios dominios diferentes. Esto viene derivado de la utilización de mapeos separados para enlazar las definiciones gráficas y de herramienta y los dominios concretos. Finalmente, el modelo de generación permite definir los detalles de implementación para la fase de generación de código, de forma similar al modelo de generación de código de EMF. El código generado es un *plug-in* que se puede personalizar y mejorar y que mediante su ejecución permite a los desarrolladores modelar software basándose en un metamodelo concreto. La generación de código se realiza con el motor de generación Xpand [Eclipse Xpand] perteneciente al proyecto *Model to Text*.

4.4.3 GMF *Runtime*

El *Runtime* de GMF permite trabajar al desarrollador con la herramienta *plug-in* de edición gráfica desarrollada en el apartado anterior. La Figura 4-5, extraída de [Plante 2006], muestra las dependencias entre el componente de editor gráfico generado, el *runtime* de GMF y los componentes GEF y EMF, que todos ellos están soportados por la plataforma Eclipse. Se observa que la dependencia del editor con el *GMF Runtime*, ya que es el encargado de ejecutar el *Plug-in*, pero a su vez, el editor tiene una dependencia con el modelo de dominio definido en EMF y con los componentes gráficos definidos en GEF.

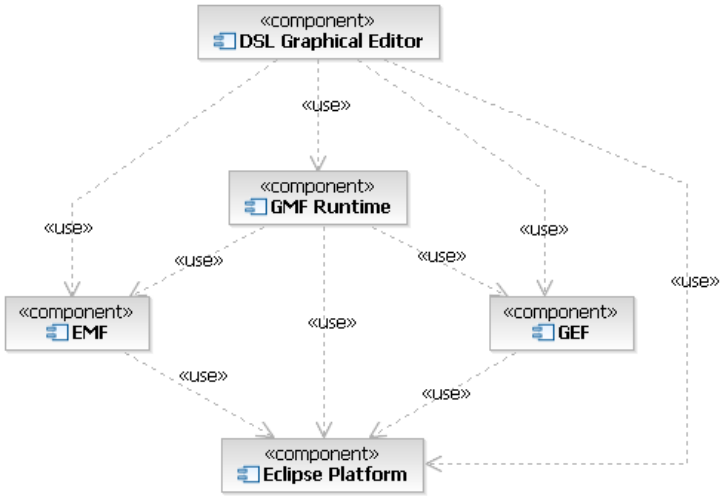

Figura 4-5. Dependencias de la plataforma GMF

Algunos de los beneficios de que se pueden apuntar sobre la utilización de los editores basados en el *runtime* de GMF son los siguientes [Plante 2006]:

- Proporcionan una apariencia y comportamiento común a otros editores basados en GMF.
- Los editores pueden ser creados desde cero o partiendo de elementos previamente creados como parte del SDK de GMF.

- Gestiona el diagrama de persistencia, permitiendo al desarrollador centrarse en la lógica de negocio del sistema.
- Su naturaleza extensible permite a los clientes del *runtime* de GMF, ser editores abiertos que a su vez pueden ser extendidos por terceros.
- Está completamente integrado con varios de los componentes EMFT como la definición de restricciones y validaciones mediante OCL.
- Define un meta-modelo de notación ampliable para permitir la total separación entre la semántica y la notación.
- Sus características están bien diseñadas, codificadas, probadas y desplegadas.
- Se realizan nuevos desarrollo por la comunidad encaminados a mejorar el *runtime* de manera que sean fáciles de integrar en los editores generados.

Si se construye un editor gráfico basado en GMF, se van a tener la posibilidad de explotar las características que proporciona el *runtime* de GMF, tales como:

- Expandir o contraer compartimentos de objetos (el compartimento de atributos de una clase por ejemplo).
- Edición directa de las etiquetas de los objetos sobre el elemento gráfico, sin necesidad de abrir la ventana de propiedades.
- Asistencia en el modelado mediante menús contextuales.
- Aparte de definir elementos propios en la paleta de herramientas y de los menús, disponer de elementos comunes de base a todos los editores generados.
- Proporciona un conjunto de propiedades comunes para figuras y conectores.
- Proporciona un componente de previsualización e impresión.
- Permite exportar los modelos a diferentes formatos de imagen.
- Soporta la utilización del portapapeles del sistema, para operaciones de copiado.

4.5 *Model Development Tools* (MDT)

4.5.1 Proyecto MDT

Este proyecto se centra en proveer a la industria de estándares de metamodelado y de un conjunto de herramientas para el desarrollo de modelos basados en estos estándares [Eclipse MDT]. Bajo este proyecto se encuentran herramientas como *Objects Constraint Language* (OCL), UML2, UML2 *Tools* y XML *Schemas Definition* (XSD) entre otras.

MDT se concentra en la evolución y promoción de las tecnologías de desarrollo basadas en modelos dentro de la comunidad Eclipse, proporcionando un conjunto unificado de infraestructuras de modelado, herramientas e implementaciones estándar. Algunos de los subproyectos dependientes de MDT son los siguientes:

- *UML2*: es la implementación basada en EMF de la especificación del metamodelo UML de OMG para la plataforma Eclipse [Eclipse UML2].
- *BPMN2*: es un componente *open source* de MDT que proporciona una implementación basada en el metamodelo de *Business Process Model and Notification* (BPMN) de OMG [Eclipse BPMN2].
- *IMM*: es un componente de MDT que proporciona implementaciones basadas en el metamodelo/perfil de la especificación de *Information Management Metamodel* (IMM) de OMG [Eclipse IMM].

- *OCL*: es la implementación basada en EMF de la especificación del metamodelo OCL de OMG para la plataforma Eclipse [Eclipse OCL].
- *UML2 Tools*: Es un conjunto de editores basados en GMF para la edición de modelos basados en UML. Se focaliza en la generación automática de editores para todos los tipos de diagramas UML [Eclipse UML2T].
- *XSD*: es una librería que proporciona un interfaz para la manipulación de componentes de esquemas XML [Eclipse XSD].

Como subproyectos más representativos del proyecto MDT, en este capítulo se van a estudiar básicamente los productos UML2 y UML2 Tools.

4.5.2 UML2

UML2 es una implementación basada en EMF (*Eclipse Modeling Framework*) para la plataforma Eclipse del metamodelo de UML 2.x de OMG [F. de Alba 2009]. En definitiva, es el metamodelo de Eclipse para OMG UML2 *superstructure* descrito en *Ecore*. Incluye un generador EMF de código caracterizado para manejar los distintos aspectos de UML, como la redefinición, subconjuntos y uniones derivadas.

Los objetivos del componente UML2 son proporcionar [Hussey 2005]:

- Una implementación utilizable del metamodelo de UML para soportar el desarrollo de Herramientas de modelado.
- Un esquema XMI común p/ara facilitar el intercambio de modelos semánticos.
- Casos de prueba como formas de validar la especificación.
- Reglas de validación como formas de definir e imponer los niveles de conformidad.

Sin embargo, UML2 sólo trata con la sintaxis abstracta de UML. La notación concreta se trata en el proyecto UML2 Tools, el cual consiste en un conjunto de editores gráficos para visualizar y editar modelos de UML2 con la notación UML. UML2 Tools está basado en el *framework* GMF.

UML2 se organiza en unidades del lenguaje que son una colección de conceptos de modelado estrechamente relacionados que tratan con un aspecto particular de un sistema. Están divididos en paquetes (incrementos de mezcla) y se usan como fundamento para definir la conformidad en UML.

Los niveles de conformidad están divididos en diferentes niveles horizontales referidos como puntos de conformidad. Un nivel de conformidad es finalmente mezclado en un único paquete "UML" que define un espacio de nombres compartido por todos los niveles de conformidad. UML define cinco niveles de conformidad predefinidos y UML2 se basa en el nivel L3 de UML, agregando la funcionalidad de Eclipse *Ecore* para la generación de código [OMG 2007b]:

- L0: es el nivel más básico de conformidad de UML contiene la capacidad básica para modelar estructura tiene la misma capacidad de modelado que EMOF (*Essential* MOF), pero no incluye las otras capacidades de MOF como la reflexión, identidad o extensiones.
- LM: es el nivel de conformidad de las construcciones de metamodelo de la *InfrastructureLibrary* añade una unidad de lenguaje extra para estructuras basadas en clases más avanzadas usadas para construir metamodelos (usando CMOF), como UML mismo.
- L1: extiende las capacidades proporcionadas por el nivel LM añade unidades de lenguaje para casos de uso, interacciones, estructuras, acciones y actividades.
- L2: extiende las unidades de lenguaje proporcionadas en el nivel L1 añade unidades de lenguaje para despliegue, modelado de máquinas de estados y perfiles.

- L3: representa el UML completo extiende las unidades de lenguaje proporcionadas en el nivel L2 añade nuevas unidades de lenguaje para modelar flujos de información, plantillas y empaquetado de modelos.

UML2 de Eclipse implementa las características de UML 2.0, cuyas principales características son las que se describen a continuación. La *mezcla de paquetes* (*package merge*) es una relación dirigida entre dos paquetes que indica que los contenidos de los dos paquetes se combinan. Se puede ver como una operación que toma los contenidos de dos paquetes y produce un nuevo paquete que combina los contenidos de los paquetes involucrados en la mezcla. Seleccionando qué incrementos mezclar, es posible obtener una definición a medida de un concepto para un fin específico.

La terminología en relación a la característica de la mezcla de paquetes es la siguiente:

- Elemento mezclado: contenido en el paquete mezclado antes de la mezcla
- Elemento receptor: contenido en el paquete receptor antes de la mezcla
- Elemento resultante: contenido en el paquete resultante después de la mezcla
- Paquete mezclado: objetivo de la mezcla, contiene los elementos mezclados
- Paquete receptor: fuente de la mezcla, contiene los elementos receptores
- Paquete resultante: resultado de la mezcla, contiene los elementos resultantes

La semántica de la mezcla se define mediante un conjunto de reglas de transformación y restricciones. Cuando un elemento mezclado y un elemento receptor representan la misma entidad, sus contenidos son conceptualmente mezclados en un único elemento resultante de acuerdo con las reglas de la mezcla de paquetes. Básicamente lo que se hace es añadir las restricciones de todos los elementos mezclados que representan a la misma entidad al elemento receptor.

UML2 proporciona una utilidad que impone esas restricciones y aplica esas transformaciones. Un ejemplo de mezcla de paquetes se ilustra en la Figura 4-6 ([OMG 2007b]), en donde se muestra las clases que componen determinados paquetes antes y después de realizar una mezcla. En concreto el paquete `R` se mezcla con los paquetes `P` y `Q`, que tras la mezcla, tendrá la unión de todas las clases de los tres paquetes. En este caso la clase A, que está en los tres paquetes, la clase `B` del paquete `P` y la clase `C` del paquete `Q`, manteniendo las relaciones definidas entre las clases en los paquetes originales. Lo mismo ocurre con la mezcla de los paquetes `S` y `Q`.

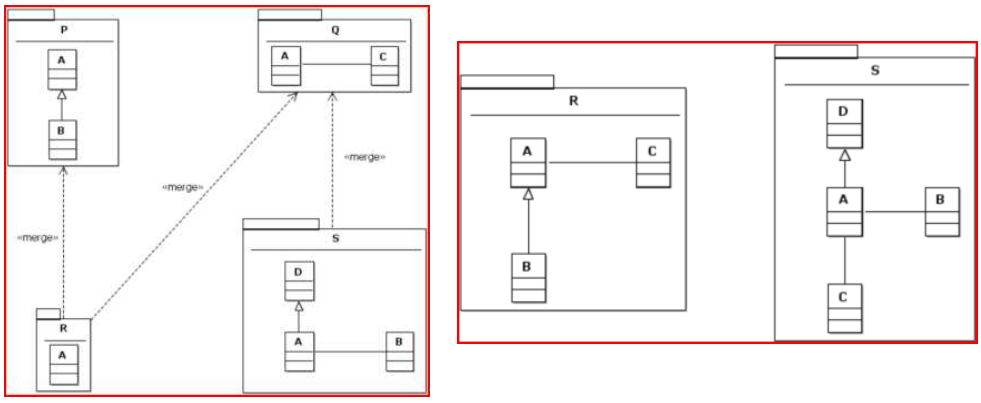

Figura 4-6. Ejemplo de mezcla de paquetes

Otra de las características implementadas es la de la *redefinición,* que se refiere a la capacidad de un elemento de redefinir a otro en el contexto de una jerarquía de generalización. Un elemento

que redefine, debe ser consistente con el elemento redefinido, pero puede añadir restricciones específicas u otros detalles particulares al clasificador que especializa.

La característica del *subconjunto*, permite marcar una propiedad como subconjunto de otra, siempre que cada elemento, en el contexto de la propiedad subconjunto, se ajuste al elemento correspondiente en la propiedad original. Para que esto ocurra, básicamente las propiedades deben tener tipos compatibles (uno subtipo del otro).

Cuando una propiedad se marca de este modo, se genera una restricción: la colección de valores asociada con una instancia de la propiedad subconjunto, debe estar incluida o ser igual a la colección de valores asociada con una instancia de la propiedad original correspondiente. Estas restricciones se definen para luego ser capaz de validar modelos.

Por último, implementa la propiedad *derivada de unión*, de manera que si una propiedad es derivada, sus valores pueden calcularse a partir de otra información. La colección de valores denotada por la propiedad en algún contexto, se deriva haciendo la unión estricta de todos los valores denotados en el mismo contexto, por sus propiedades subconjunto. Esta restricción sirve para delimitar la estructura de la información.

UML2, al igual que UML, admite ser extendido para adaptarse a las necesidades de un determinado dominio de aplicación. Para ello, en UML se describen dos tipos "oficiales" de adaptación: (i) extensiones de peso ligero, utilizando perfiles y (ii) extensiones de peso pesado, modificando el metamodelo de UML con MOF. Sin embargo, en UML2 se definen dos tipos más que permiten ajustarse más a la radicalidad de los cambios [Bruck 2008].

- *Extensiones de peso pluma*: se hacen adornando los elementos del modelo con palabras clave. Una palabra clave es una etiqueta para un elemento del modelo que se muestra típicamente entre comillas angulares. UML2 almacena estas palabras clave en anotaciones *Ecore*. La Tabla 4-2 muestra las ventajas y desventajas de este mecanismo de extensión.

Ventajas	Desventajas
pueden cambiar la IU textual	no pueden cambiar la IU gráfica
	no pueden agregar estructura
	no pueden agregar comportamiento
	no pueden agregar restricciones
sin coste de desarrollo	
se pueden reutilizar herramientas	las herramientas son independientes del dominio
	la API es independiente del dominio
	el formato del fichero no es estándar

Tabla 4-2. Ventajas y desventajas de las extensiones de peso pluma

- *Extensiones de peso ligero*: se hacen creando perfiles/estereotipos y aplicándolos a elementos del modelo. Los estereotipos se pueden usar para añadir palabras clave, restricciones, imágenes y propiedades (valores etiquetados) a los elementos del modelo. Los metadatos para los perfiles deben definirse y desplegarse. UML2 almacena aplicaciones de estereotipo como objetos EMF dinámicos. La Tabla 4-3 muestra las ventajas y desventajas de este mecanismo de extensión.

Ventajas	Desventajas
pueden cambiar la IU gráfica y textual	
Pueden agregar estructura	no pueden modificar estructura
	no pueden agregar comportamiento
Pueden agregar restricciones	no pueden modificar restricciones
bajo coste de desarrollo	
se pueden reutilizar herramientas	las herramientas son independientes del dominio
	la API es independiente del dominio
El formato del fichero es estándar	el formato del fichero es recargado

Tabla 4-3. Ventajas y desventajas de las extensiones de peso ligero

- *Las extensiones de peso medio*: se hacen creando un metamodelo que extienda el de UML2 vía especialización. Debe generarse, implementarse y desplegarse una nueva API y esquema para el nuevo metamodelo. Las clases de implementación en el metamodelo especializado extienden las de UML2. Como este tipo de extensiones se hace por especialización, el ajuste a la API de UML debería mantenerse. La Tabla 4-4 muestra las ventajas y desventajas de este mecanismo de extensión.

Ventajas	Desventajas
pueden cambiar la IU grafica y textual	
pueden agregar estructura	
pueden agregar comportamiento	no pueden modificar comportamiento
pueden agregar restricciones	no pueden modificar restricciones
	alto coste de desarrollo
se pueden reutilizar herramientas	las herramientas son independientes del dominio
la API es específica del dominio	
el formato del fichero es compacto	el formato del fichero no es estándar

Tabla 4-4. Ventajas y desventajas de las extensiones de peso medio

- *Las extensiones de peso pesado*: se hacen creando un metamodelo que mezcla paquetes de UML. Se debe generar, implementar y desplegar una API y un esquema para el nuevo metamodelo. Las clases de implementación en el metamodelo especializado no extienden las clases de implementación de UML2. Como se está modificando el metamodelo de UML2 de raíz, el ajuste a la API ya no es necesario (o mejor dicho, no es posible). La Tabla 4-5 muestra las ventajas y desventajas de este mecanismo de extensión.

Ventajas	Desventajas
pueden cambiar IU gráfica y textual	
pueden agregar y modificar estructura	
pueden agregar y modificar comportamiento	
	alto coste de desarrollo
las herramientas son específicas del dominio	no se pueden reutilizar herramientas
la API es específica del dominio	
el formato del fichero es completo	el formato del fichero no es estándar

Tabla 4-5. Ventajas y desventajas de las extensiones de peso pesado

4.5.3 UML2 Tools

UML2 Tools es un conjunto de editores basados en GMF para la edición de modelos basados en UML. Se focaliza en la generación automática de editores para todos los tipos de diagramas UML Eclipse, proporcionando editores para diagramas estructurales, de comportamiento o de interacción. UML2 Tools se basa en GMF para agregar la sintaxis típica de UML al metamodelo de UML2. Mediante UML2 Tools se pueden crear los siguientes diagramas [Eclipse UML2]. Se observa en la Figura 4-7, un ejemplo de diagrama de estados realizado con UML2 Tools en el que se muestran el comportamiento del acceso de un usuario a un sistema, en donde el usuario puede acceder o registrarse en ese mismo momento.

- Diagramas de estructura:
  - Clases
  - Definición de perfiles
  - Estructura compuesta
  - Componentes
  - Despliegue

- Diagramas de comportamiento:
  - Actividades
  - Estados
  - Casos de uso
- Diagramas de interacción:
  - Secuencias
  - Tiempos (en desarrollo)

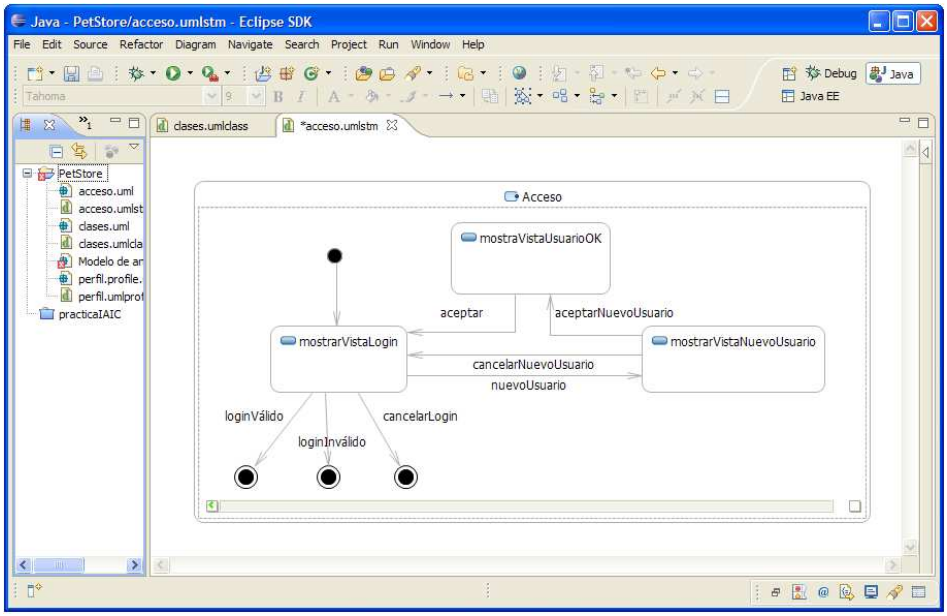

Figura 4-7. Ejemplo de diagrama de estados con UML2 Tools

UML2 Tools permite crear perfiles que permitan extender el estándar UML y conseguir nuevos posibles dominios [Eclipse UML2P]. Para crear un perfil con UML2 Tools, se debe primero crear un diagrama de definición de perfiles. Los elementos principales de estos diagramas son los perfiles, los estereotipos, las metaclases y las relaciones de la extensión. Tras crear el diagrama, hay que realizar la definición del perfil, que lo genera como una estructura *Ecore* estática en el modelo UML y permitirá su uso. Para poder aplicar los estereotipos definidos en el perfil a elementos del modelo, primero hay que ejecutar las operaciones de aplicar perfil y aplicar los estereotipos definidos. De esta manera se encuentra el elemento extendido. La Figura 4-8 [Eclipse UML2P], muestra el flujo necesario para extender un elemento UML mediante UML2 Tools.

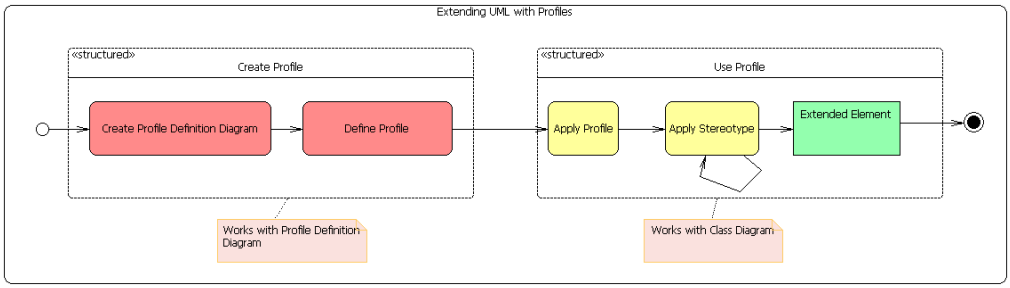

Figura 4-8. Extensión de UML mediante perfiles

4.6 Relación entre Estándar OMG y Eclipse

Según [Gronback 2006] la importancia de que se soporten los estándares de la industria es fundamental para el éxito del proyecto *Eclipse Modeling Project*. El papel de este proyecto en el soporte de estos estándares, es su creación y mantenimiento, además de facilitar la comunicación más allá de los propios miembros de la comunidad Eclipse y extenderla a organizaciones externas. Eclipse ha contribuido al éxito de la ejecución de MDA, proporcionando una plataforma *open source* y una implementación de facto de las especificaciones MDA.

El proyecto *Model Development Tools* (MDT) se centra en proporcionar implementaciones de los metamodelos estándares de la industria y proporcionar herramientas que permitan definir modelos basados en estos metamodelos. En definitiva, El proyecto *Modeling* de Eclipse, proporciona tecnologías alternativas a muchas de las especificaciones MDA.

En la Tabla 4-6 se muestra la relación de los estándares más relevantes OMG, con sus correspondientes implementaciones en Eclipse [Gronback 2008].

OMG	Eclipse	Comentarios
Meta Object Facility (MOF)	EMF	En concreto, el alineamiento existe entre la especificación EMOF y el metamodelo *Ecore* de EMF
Unified Modeling Language (UML)	UML2	Componente de MDT
Object Constraint Language (OCL)	OCL	Componente de MDT
UML *Diagram Interchange* (DI)	No tiene	
XML *Metadata Interchange* (XMI)	MDT	Inicialmente estaba soportado por EMF, pero ahora lo soporta el subproyecto XSD de MDT.
Query/View/Transformation (QVT)	M2M	QVT es parte del proyecto *Model-to-Model transformation*, al cual también pertenece el lenguaje ATL. A un nivel de detalle mayor, la equivalencia estaría entre *Operational Mappings* de OMG y *Procedural QVT* de Eclipse y entre *Core* y *Relational* de OMG y *Declarative QVT* de Eclipse
MOF2Text	M2T/GMT	M2T tiene varias herramientas, como JET o Xpand. GMT proporciona el componente MOFScript.
Human-Usable Textual Notation (HUTN)	No tiene	Aunque está relacionado con el proyecto *Textual modeling Framework*
Software Process Engineering Metamodel (SPEM)	EPF	No hay conexión entre el proyecto *Eclipse Process Framework* y el proyecto *Modeling*. Realmente la implementación de facto de SPEM la hace EMF.

Tabla 4-6. Equivalencia entre Estándares OMG y Eclipse

Según esta tabla, queda reflejado que los estándares de Eclipse son un espejo de los de OMG. No es sólo esto. Tras su estudio, se ha observado que son una simplificación de los mismos. Eclipse ha contribuido significativamente a la implementación de la aproximación de MDA, proporcionando una plataforma *open source* y una implementación de muchas de las especificaciones de MDA. Esto ha sido así, a pesar de que no haya existido una colaboración real con OMG. Hasta hace relativamente poco tiempo no ha existido una relación formal entre ambas organizaciones. Parece que esto ha llevado a Eclipse a simplificar los estándares de OMG y ha conseguido, que de alguna manera, tengan cierto impacto en la industria, convirtiéndose en estándares de facto.

5. SOPORTE MDA EN HERRAMIENTAS CASE

5.1 Introducción

Este capítulo se centra en analizar y comparar el soporte MDA proporcionado por diferentes herramientas CASE. Al contrario que lo visto en el capítulo anterior, las herramientas estudiadas en éste, son productos comerciales de pago, que pretenden proporcionar a la industria medios para el desarrollo de sistemas software. El objetivo principal de este estudio es ver hasta qué punto cubren el ciclo de vida de desarrollo definido por la aproximación MDA, así como sus características más importantes.

Las herramientas que se han evaluado en este estudio son:

- *IBM Rational Software Architect* [IBM RSA]
- *Borland Together* [Borland 2008]
- *Sparx Systems Enterprise Architect* [Sparx EAGuide]

Se han considerado estas herramientas por ser algunas de las más conocidas en los entornos profesionales de desarrollo de software.

5.1.1 Criterios de estudio

Existen diferentes artículos que, desde distintos puntos de vista, proporcionan un estudio de algunas herramientas que soportan la aproximación MDA [Cabot 2006][López 2007][Génova 2006][Bollati 2007]. A diferencia de las herramientas analizadas en estos artículos, basadas en herramientas *Open Source*, este estudio se ha centrado en aquellas de corte más comercial y que dan soporte total a OMG UML 2. El objetivo de este capítulo es abordar un estudio de las herramientas mencionadas, aportando una visión general del grado de cumplimiento en el soporte de MDA.

El método desarrollado, se basa en la identificación de un conjunto de criterios que serán evaluados para cada una de las herramientas. De esta manera, de una forma objetiva, se podrá medir hasta qué punto se implementa la aproximación MDA en cada una de ellas y tener a su vez, una homogeneidad de datos que permita realizar una comparativa de las mismas.

Los criterios de evaluación elegidos están inspirados en los utilizados por [Rodríguez 2004] y [King's 2003], en los que realizan sendos estudios de herramientas que soportan MDA. Para simplificar el estudio, se han elegido de entre estos criterios algunas de las propiedades más relevantes de la aproximación MDA.

Las propiedades evaluadas para cada herramienta son los que se detallan en la Tabla 5-1 mostrada a continuación:

Id	Propiedad	Descripción
P01	Interoperabilidad y estandarización	La herramienta puede importar y exportar información a otras herramientas, mediante estándares XMI. Así mismo se analizan la utilización de estándares básicos de MDA.
P02	Acceso a la definición de las transformaciones	La herramienta provee de un mecanismo de definición de transformaciones entre modelos y permite al usuario crear nuevas transformaciones o modificar las existentes para satisfacer sus requisitos específicos. Se tendrá en cuenta tanto en transformaciones de modelos, como en la generación de código.
P03	Uso de patrones	La herramienta aplica o permite aplicar patrones de diseño en la construcción de modelos PIM, PSM o código y pueden definirse otros nuevos o modificar los existentes.
P04	Control y refinamiento de las transformaciones	La aplicación permite dirigir o controlar las transformaciones entre modelos o entre el modelo PSM y el código. Por ejemplo, dispone de parámetros en las transformaciones, permite seleccionar los elementos a ser transformados o establecer condiciones para las transformaciones
P05	Trazabilidad	Incorpora un mecanismo para seguir el rastro de determinadas transformaciones desde su origen hasta su destino.
P06	Soporte para regeneración de modelos	La herramienta proporciona soporte para rehacer modelos, por ejemplo, regenerar el PIM a partir de los modelos PSM y viceversa. También debe permitir conservar los cambios efectuados "manualmente" tanto a nivel de modelo como de código.
P07	Verificador de Modelos	Incluye algún mecanismo para chequear la corrección de los modelos incluidos PIM y PSM.

Tabla 5-1. Criterios de estudio de las herramientas MDA elegidas

Para cada una de las herramientas, se indicará mediante los iconos que se muestran en la Tabla 5-2, el grado de cumplimiento de las propiedades expuestas.

Puntuación	Descripción
	La herramienta estudiada no contempla la propiedad evaluada.
	La propiedad evaluada es soportada aunque no de una forma completa.
	La herramienta soporta de forma excelente la propiedad evaluada.

Tabla 5-2. Criterios de puntuación aplicables a las propiedades.

5.1.2 Ejemplo práctico

Para facilitar el estudio, a la hora de evaluar las propiedades, se ha partido de un sencillo ejemplo de aplicación web, de manera que la comparativa no se desvirtúe por características propias de ejemplos diferentes. Al igual que el método elegido, se ha utilizado el mismo ejemplo empleado en [Rodríguez 2004] por su simplicidad y como ejemplo académico utilizado por Sun para probar nuevas herramientas y tecnologías [SUN PET].

Concretamente, se trata de un sistema de gestión de venta de mascotas (*Pet Store*) a través de Internet. La idea es tener una web que permita realizar las operaciones típicas de este tipo de negocios: consultar el catálogo de animales disponibles, poder incluirlos en el carrito de la compra y finalmente realizar un pedido. El objetivo de este estudio no es cubrir todas las fases del desarrollo de MDA y conseguir desarrollar una aplicación, sino tener una definición de partida que permita especificar los ejemplos prácticos para verificar si características evaluadas son contempladas por las diferentes herramientas.

El diagrama de clases de la Figura 5-1 representaría el conjunto de requisitos de este sistema. En él que se puede observar la clase `Cliente` que representa a los compradores o futuros compradores, ya que según la relación `Cliente-Pedidos`, no es necesario para ser cliente haber realizado ningún pedido previo, representados por la clase `Pedido`. Los pedidos se detallan mediante la clase `LineaPedido` por este motivo existe una relación de agregación o composición entre ellas. En las líneas de pedido se especifican las unidades que se desean adquirir de cada animal representado mediante `Animal`. Como suele ser habitual en este tipo de definiciones, un pedido debe estar compuesto al menos de una línea de pedido, que a su vez, de forma obligatoria tiene que tener asociado un animal. La clase `Categoria` identifica la clasificación de los animales de manera que todo animal tiene que pertenecer a una sola categoría. Este diagrama representa el modelo PIM del sistema *PetStore* que va a servir de base para el estudio de las diferentes herramientas propuestas.

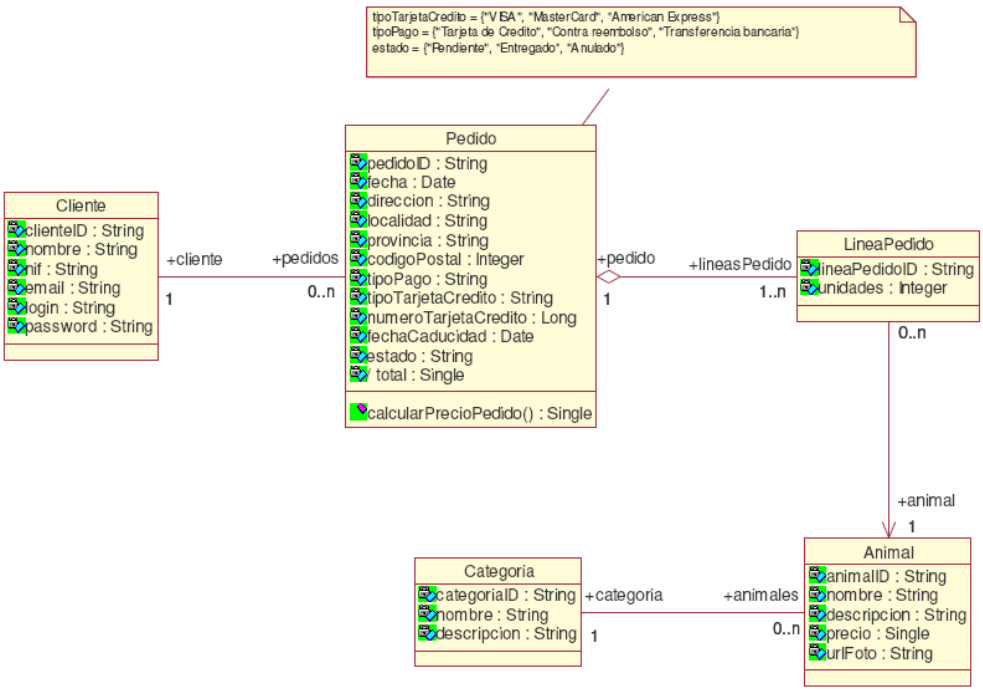

Figura 5-1. Diagrama de Clases de *Pet Store*

En general, el procedimiento utilizado para el estudio de las herramientas ha sido partir de la documentación del producto. Se han localizado las referencias a las características estudiadas y en los casos en los que se encontraba dificultad en la comprensión de la documentación o no estaba claramente explicado, se procedía a hacer un pequeño ejemplo que permitiera comprobar la característica concreta.

5.2 *IBM Rational Software Architect* (RSA)

5.2.1 Descripción de IBM RSA

IBM RSA es una herramienta colaborativa que provee de un soporte integral para el diseño y el desarrollo enfocado en un desarrollo dirigido por modelos con el lenguaje UML. Permite realizar un diseño visual de los sistemas a desarrollas, realizar transformaciones desde los diagramas a código y continuar con el desarrollo de la aplicación. Cubre el ciclo de vida del desarrollo basándose en la metodología *Rational Unified Process* [IBM RUP].

Permite transformaciones de modelo a modelo y de modelo a texto, abarcando diferentes lenguajes de programación. Permite también la creación de patrones que reduzca el esfuerzo de rehacer trabajos repetitivos, reduciendo costes de desarrollo y aumentando la calidad del software resultante.

RSA soporta algunos de los estándares definidos por OMG, tales como UML, XMI y OCL. Tiene integrado un entorno de desarrollo Eclipse que permite refinar el código generado de forma cómoda. De hecho, los productos de RSA están construidos sobre una arquitectura extendida proporcionada por Eclipse, por lo que su integración es total a través de un conjunto de *Plug-ins*. Dentro de los componentes Eclipse integrados en RSA, el que más estrechamente relacionado está con el modelado es EMF, marco de trabajo que permite generar código eficiente y adaptable desde los propios modelos. También integra *Eclipse Modeling Framework Technologies* (EMFT) que contempla diferentes tecnologías que complementan la manipulación de modelos basados en EMF y otros subproyectos de *Eclipse Modeling Project* como GMF y GEF.

5.2.2 Análisis de RSA

A continuación se describe, para cada una de las características propuestas, el análisis realizado sobre la herramienta *Rational software Architect* de *IBM*. La versión estudiada es la 7.5 y el análisis se ha basado en el estudio de la documentación de la herramienta [IBM RSA], a partir de la que se han realizados diferentes ejemplos para su mejor comprensión y poder observar el grado de cumplimiento de las garantías.

P01 – Interoperabilidad 

La interoperabilidad conseguida por RSA con otras herramientas, la consigue mediante la importación o exportación de los modelos de dos formas diferentes: (i) una mediante formato XMI y (ii) mediante formato UML 2.1 (ficheros *.uml*). En el caso de la exportación, se pueden también exportar los perfiles UML aplicados.

Se han realizado pruebas de importar el modelo del ejemplo de *PetStore*, desde un fichero XMI generado desde EA, pero no se ha conseguido importar. Aparentemente se procesaba correctamente el fichero y en el explorador de proyecto aparecía el diagrama, pero debajo de él no aparecían los elementos. No se ha podido investigar sobre este tema concreto a fondo.

En cuanto a otro tipo de interoperabilidad, RSA también permite importar y exportar los modelos directamente en formato *Ecore* de Eclipse. Cabe destacar, que RSA está construida sobre la arquitectura extensible de Eclipse, por lo que la integración con componentes de eclipse, es muy alta.

En cuanto al uso de estándares, RSA implementa UML de forma completa, permitiendo el modelado de sistemas con todas las posibilidades que permite este estándar. Igualmente permite extender el modelo UML mediante perfiles y crear restricciones mediante el estándar OCL. Según la documentación estudiada no se soporta el estándar MOF, de manera que no se podría definir un

metamodelo directamente como instancia de MOF y sólo existiría el mecanismo de las extensiones. Tampoco se implementa el estándar QVT.

P02 - Acceso a la definición de las transformaciones

Las transformaciones en RSA se definen mediante la creación de una configuración de la transformación. Una configuración de una transformación es una instancia de una transformación, que incluye información que va a utilizar al ejecutarse. En ella se especifican la información que va a utilizar la transformación para generar la salida esperada. También puede incluir propiedades específicas de una transformación específica como elementos del lenguaje destino, en transformaciones modelo a Código.

En definitiva, RSA permite el acceso a la definición de las transformaciones, aunque en el fondo es una configuración de parámetros de transformaciones ya existentes. El hecho de no tener soporte a QVT, no permite definir nuevas transformaciones, si no que sólo se proporciona la posibilidad de configurar un conjunto de transformaciones ya definidas de antemano.

P03 - Uso de patrones

Mediante el uso de *Rational design patterns* [IBM RDP] se puede integrar soluciones de diseño en los modelos UML definidos. Los diseños de patrones van a capturar procesos y estructuras complejas o frecuentemente utilizadas para aplicarles una solución de diseño que al ser utilizada permitirá eliminar el rediseño repetitivo y proporcionará consistencia al software desarrollado. RSA permite dentro de la definición de los patrones, el uso a su vez de patrones previamente definidos. La definición de los patrones empieza en un modelo UML y acaba con un patrón RAS (*Reusable Asset Specification*) [OMG 2005b] en forma de *Plug-in* que facilita su reutilización.

En la Figura 5-2, se muestra un ejemplo de aplicación de patrón en el modelo de ejemplo PetStore. El caso concreto es el de la aplicación de un patrón `Interface` que lo que hace al aplicarse es asegurarse que una clase que implementa la interfaz tenga definidas todas las operaciones definidas en la interfaz. En este caso se ha añadido la operación `crearPedido` a la interfaz y al aplicar el patrón, ésta se ha añadido a la clase `Pedido`.

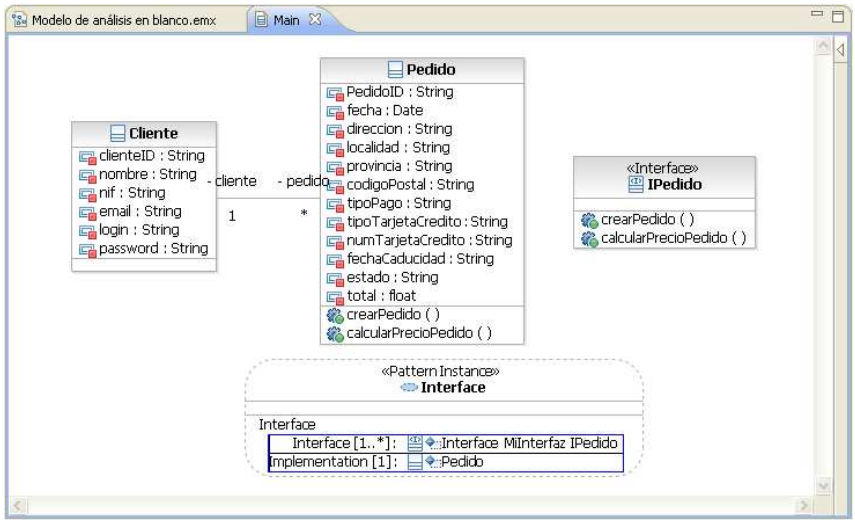

Figura 5-2. Ejemplo de Patrón RSA

P04 - Control y refinamiento de las transformaciones 

RSA permite un refinamiento de las transformaciones en el sentido de que se pueden elegir los elementos a transformar tanto en la definición de las transformaciones como en el momento de ejecutarlas. En todo momento se pueden modificar los elementos a transformar del modelo y las propiedades de los objetivos de la transformación. Así mismo, desde las definiciones de las transformaciones, sobre todo en el caso de las transformaciones a código se puede especificar el nombre del fichero destino y las carpetas en donde se van a generar. Esto de puede definir mediante un modelo de mapeo de nombre que permita crear convenciones de nombres para aplicarse de forma automática en las transformaciones.

RSA también permite generar un *log* en las ejecuciones de las transformaciones que proporciona información sobre las reglas de ejecución, los elementos fuentes y objetivo, en formato XML.

P05 – Trazabilidad 

La trazabilidad que gestiona RSA es la relativa a los requerimientos definidos para el sistema, con el fin de poder comprobar que los artefactos generados cumplen con todos ellos. Esta trazabilidad se gestiona mediante la herramienta *Rational RequisitePro* [IBM RRP], integrada en RSA. No se ha encontrado en la documentación estudiada ninguna referencia a la trazabilidad en cuanto a las relaciones existentes entre modelos fuentes y objetivos de las transformaciones.

P06 - Soporte para regeneración de modelos 

RSA permite generar los modelos o código partiendo de las transformaciones definidas, pero en el caso de que se efectúen modificaciones sobre el resultado de las transformaciones, éstas no se mantendrán si se vuelve a ejecutar la misma. Permitiría hacer ingeniería inversa. Por ejemplo modificando el código fuente generado y realizando una generación del modelo partiendo con el nuevo código, pero no tiene elementos que permitan identificar los objetos modificados y que sean respetados en las transformaciones.

No obstante, la integración que tiene RSA con EMF, permite mantener las modificaciones realizadas sobre el código generado mediante una transformación, mediante los mecanismos que tiene EMF, explicados en la sección 4.3.4 de este trabajo.

P07 - Verificador de Modelos 

RSA permite realizar en cualquier momento la validación de los modelos y sus diagramas, de manera que se verifique la correcta compilación de los mismos y de las restricciones implementadas. Permite validar modelos, diagramas o elementos concretos de los modelos, permitiendo localizar fácilmente los errores detectados para su corrección.

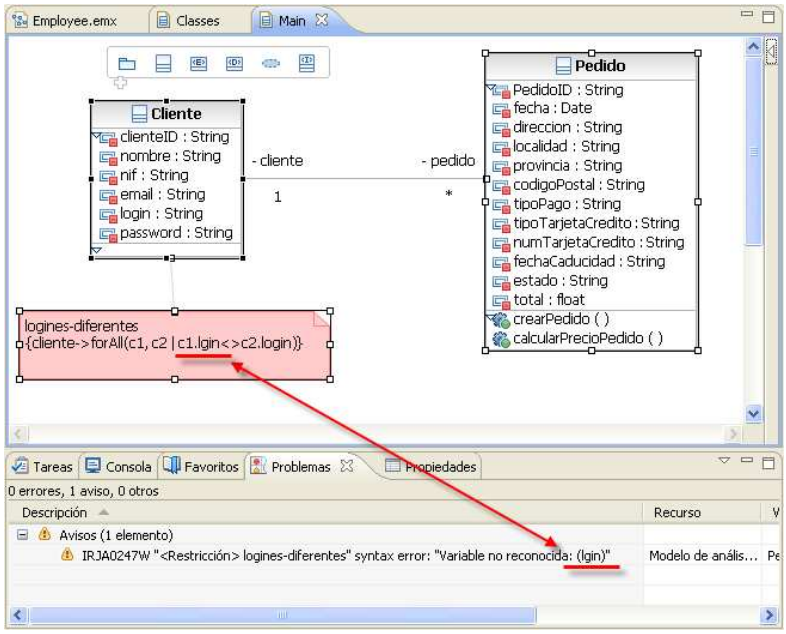

Figura 5-3. Ejemplo de Validación RSA

En el ejemplo de la Figura 5-3 se muestra el resultado de una validación ejecutada sobre parte del diagrama de clases de PetStore. Sobre la clase `Cliente` se ha definido una restricción OCL que especifica que dos clientes nunca deben tener el mismo login. Al ejecutar la validación Se detecta un error, ya que se ha escrito mal el nombre del atributo `login`.

5.3 Borland Together

5.3.1 Descripción de Borland Together

Borland Together es una herramienta de modelado que permite el análisis diseño e implementación de arquitecturas software de forma flexible, fácilmente mantenibles y adaptables a los requisitos impuestos. Es una herramienta colaborativa, que permite a los equipos de desarrollo elaborar sus proyectos con un bajo coste y consiguiendo a la vez una alta calidad. Mediante *Together* se puede crear nuevas aplicaciones o realizar reingeniería inversa del diseño de un sistema partiendo del código de una aplicación existente. Permite la creación de patrones que reduzca el esfuerzo de rehacer trabajos repetitivos revirtiendo en los tiempos del ciclo de vida de desarrollo. Se adapta a las características MDA en cuanto a la posibilidad de definición de modelos computacionalmente independientes y modelos independientes de cualquier plataforma, soportando transformaciones a diferentes lenguajes de programación y plataformas específicas.

Together soporta algunos de los estándares definidos por OMG, tales como UML, XMI, QVT y OCL y tiene una alta integración con Eclipse y sus proyectos relacionados con *Eclipse Modeling Project*, ya que implementa la totalidad de sus subproyectos. Así mismo, permite la definición de flujos de los procesos de negocio mediante otro mediante el estándar BPMN (*Business Process Modeling Notation*) [OMG BPMN] y sus convenientes herramientas de importación y exportación a BPEL (*Business Process Execution Language*) [OASIS BPEL].

Cabe destacar también el *toolkit* que proporciona *Together* para el desarrollo de lenguajes específicos de dominio que faciliten el desarrollo de sistemas software de un problema específico de dominio concreto.

5.3.2 Análisis de Together

A continuación se describe, para cada una de las características propuestas, el análisis realizado sobre la herramienta *Together* de *Borland*. La versión estudiada es la 2008 y el análisis se ha basado en el estudio de la documentación de la herramienta [Borland 2008], por no poder ejecutar prácticamente ninguna opción del producto al tener problemas al licenciar el producto. En determinados casos, se ha utilizado material complementario. En tal caso se hace una referencia explícita al mismo.

P01 – Interoperabilidad 

Together permite importar y exportar modelos entre diferentes herramientas de varias formas. Está por un lado el método basado en el estándar OMG mediante el formato XMI, de manera que permite la interoperabilidad con todas aquellas herramientas que soporten este estándar. Por otro lado, Together permite otros procedimientos de importación y exportación de modelos tales como (i) bases de datos mediante JDBC y ficheros DDL, (ii) Generando IDL de un proyecto o (iii) directamente desde modelos Rational Rose tanto ficheros ".mdl" como ".mdx".

En cuanto a los estándares utilizados Together soporta todos los diagramas UML y los estándares QVT y OCL 2.0, aunque cabe destacar que en la versión estudiada, la generación de código no soporta OCL al 100% según se especifica en las notas de versión del producto [Borland 2009]. No se especifica en la documentación consultada, la implementación de otros estándares OMG, sin embargo sí que implementa algunos de los estándares de los estándares definidos por Eclipse (UML2 Tools, GMF y EMF entre otros).

P02 - Acceso a la definición de las transformaciones 

Together soporta un conjunto completo de capacidades de transformación basadas en la aproximación MDA, mediante el *Together Model Transformation Framework* (TMF). Este Marco de trabajo implementa la mayoría de los conceptos que se definen en la especificación de QVT de OMG y está basado en el *Eclipse Modeling Framework*. *Together* permite transformaciones de modelo a modelo, pudiendo ser los modelos origen y objetivo definidos en *Together* o EMF, y transformaciones de modelo a texto, para generar el código de la aplicación. Mediante QVT se realizan las transformaciones entre modelos desde modelos CIM a modelos PIM y desde éstos a modelos PSM. Las transformaciones para la generación de código se pueden realizar mediante un script de transformaciones QVT o transformaciones XSL/OCL. Para poder realizarlas, la herramienta *Together* viene provista con una serie de componentes tales como los editores QVT y XSL/OCL y depuradores correspondientes que van a permitir la ejecución paso a paso y con puntos de control para permitir detectar errores o malos comportamientos.

Así mismo, *Together* permite definir y modificar nuevas transformaciones, tanto de modelo a modelo como de modelo a texto, proporcionando así facilidades a los desarrolladores para implementar sus propias transformación sin depender de fabricante externos. La generación de código que viene definida por defecto es a los lenguajes Java, C++, C#, BPEL y SQL DDL.

P03 - Uso de patrones 

El uso de patrones que proporciona *Together* permite a los desarrolladores la reutilización de soluciones diseñadas para un determinado problema. *Together* proporciona una serie de patrones definidos por defecto y permite crear al desarrollador sus propios patrones. La jerarquía de los

patrones definidos se realiza mediante el registro de patrones (*pattern registry*) y la gestión de los mismos se realiza desde la ventana de organización de patrones (*pattern organizer*), desde la que se pueden crear y modificar patrones, así como implementar soluciones de código fuente asociadas al proyecto.

Cada patrón describe un conjunto de elementos de modelo, relaciones entre ellos y restricciones que se les aplican, siendo independientes a cualquier marca de lenguaje de programación concreto.

P04 - Control y refinamiento de las transformaciones

En cuanto a la posibilidad de controlar los elementos a transformar y poder refinar determinados parámetros en la ejecución de las transformaciones, *Together* permite realizar las mismas sobre uno o varios elementos de un diagrama o sobre diagramas completos. Al aplicar una transformación determinadas, se lanza un asistente que permite definir ciertos parámetros de ejecución que cada vez pueden ser diferentes. Cabe mencionar que al no poder ejecutar la aplicación y al ser la documentación un tanto escueta sobre este punto no se ha podido valorar convenientemente esta característica sobre *Together*.

P05 – Trazabilidad

Durante las transformaciones se genera un modelo de trazabilidad desde los parámetros de transformación hasta los resultados de la misma. Los datos de las trazas son almacenados en un fichero que se puede abrir con la opción de análisis de trazabilidad de transformación de modelos que proporciona *Together* [Borland 2006]. Las entradas y salidas del resultado de la transformación y del método QVT responsable de la misma, se muestran en el fichero de trazas.

También se pueden crear y mantener trazas entre los elementos de los diagramas *Together* y los requisitos definidos en las herramientas *Borland CaliberRM* [Borland CRM] o *Rational RequisitePro* [IBM RRP]. La trazabilidad de estos requisitos no la soporta directamente *Together* sino que la soportan ambas herramientas. Se pueden gestionar trazas mediante requerimientos y elementos de diagramas *Together*, navegar fácilmente entre los elementos trazados y los requerimientos relacionados y sincronizar las trazas entre *Together* y el repositorio *CaliberRM* o *RequisitePro*.

P06 - Soporte para regeneración de modelos

Together permite generar los modelos o código partiendo de las transformaciones definidas, pero en el caso de que se efectúen modificaciones sobre el resultado de las transformaciones, éstas no se mantendrán si se vuelve a ejecutar la misma. Permitiría hacer ingeniería inversa. Por ejemplo modificando el código fuente generado y realizando una generación del modelo partiendo con el nuevo código, pero no tiene elementos que permitan identificar los objetos modificados y que sean respetados en las transformaciones.

Sólo tiene la posibilidad de respetar las modificaciones realizadas en el código que ofrece EMF, que al estar integrada en *Together*, permite modificarlo salvaguardándolo de futuras ejecuciones de las transformaciones, tal y como se ha explicado en la sección 4.3.4 de este trabajo.

P07 - Verificador de Modelos

Las posibilidades que ofrece *Together* en cuanto a validación y verificación, se acota sólo a la verificación de sentencias OCL, definición de perfiles y diagramas BPMN [OMG BPMN], pero en cuanto a diagramas UML, no se ha encontrado en la documentación estudiada ninguna referencia.

5.4 *Sparx Systems Enterprise Architect* (EA)

5.4.1 Descripción de *Sparx Systems EA*

Sparx Systems Enterprise Architect es una plataforma colaborativa de modelado, diseño y mantenimiento basada en UML 2.1 y estándares relacionados, que permita desarrollar sistemas software mediante la metodología dirigida por modelos, ya que cumple con muchos de los aspectos definidos en MDA. Permite el modelado del sistema mediante PIM, ejecutar transformaciones sucesivas para generar diagramas PSM y la codificación de los diferentes artefactos que componen el sistema. Además de la posibilidad de las transformaciones, EA cumple con varias de las características definidas en MDA: trazabilidad, extensión de UML, validación de modelos, creación de modelos MOF, etc. También permite realizar ingeniería inversa permitiendo generar un modelo del sistema desde el código fuente de una aplicación o desde un esquema de base de datos concreta mediante conexión ODBC.

Así mismo, EA proporciona un SDK que aporta un entorno de desarrollo que facilita la tarea de definición y construcción del sistema y una herramienta de gestión de proyectos que permite planificar tareas y sus lanzamientos, gestionar los recursos asignados al proyecto y realizar el seguimiento oportuno.

Esto lo convierte en una herramienta completa que integra gestión, entorno de desarrollo y metodología basada en MDA para el desarrollo de sistemas software.

5.4.2 Análisis de EA

A continuación se describe, para cada una de las características propuestas, el análisis realizado sobre la herramienta *Sparx System Enterprise Architect*. La versión estudiada es la 7.5 y el análisis se ha basado principalmente en el estudio de la guía de usuario de EA [Sparx EAGuide]. Partiendo de la información de esta guía de referencia se han realizado ejemplos prácticos que permitieran comprender mejor el grado de cumplimiento de la característica estudiada. En determinados casos, se ha utilizado material complementario. En tal caso se hace una referencia explicita al mismo.

P01 – Interoperabilidad y estandarización 

Enterprise Architect permite la exportación e importación la especificación de modelos entre con otras herramientas que soporten este estándar. Así mismo, el proceso de importación y exportación, puede ser validado mediante definiciones DTD (*Data Type Definition*), de manera que se chequea y valida la corrección de los modelos y se evitan errores sintácticos en la importación. A pesar de la utilización del Estándar XMI, EA también permite importar y exportar modelos mediante ficheros *.csv* e importar directamente de ficheros *.emx* y *.uml2* generadas por *Rational Software Architect* (RSA) y *Rational Software Modeler* (RSM) o esquemas de bases de datos de diferentes motores tales como Ingres, SQL Server, Oracle y Sybase, entre otros.

Se han hecho pruebas de exportación en importación entre EA y RSA, para verificar el correcto funcionamiento de las mismas y se observa que EA es capaz de importar correctamente los modelos desde ficheros *.emx* de RSA y las exportaciones XMI. En sentido contrario, como se ha mencionado más arriba, no se consiguió importar en RSA un modelo exportado con EA, por lo que da la impresión de que la generación de ficheros XMI de EA, no es del todo correcta.

Enterprise Architect permite también un modelado rápido mediante ingeniería inversa y directa, utilizando esquemas XML [XSD] y servicios web de descripción de lenguajes [WSDL].

En cuanto al uso de los estándares, además de los mencionados, EA implementa los estándares UML, MOF y OCL. UML es el centro de EA, de hecho la propia *Sparx Systems* la define como una herramienta UML, más que como una herramienta MDA. EA aprovecha la flexibilidad de

UML para poder definir modelos de sistemas concretos. Implementa todos los tipos de diagramas, permite la definición de modelos específicos de dominio extendiendo UML mediante perfiles. También permite generar nuevos metamodelos soportados por MOF [Sparx 2005b]. El estándar OCL es utilizado para la validación de modelos.

P02 - Acceso a la definición de las transformaciones

De acuerdo con la aproximación MDA, EA proporciona los métodos de necesarios para realizar las conversiones de elementos de un modelo en fragmentos de modelos de uno o varios dominios diferentes. EA permite realizar transformaciones definidas por defecto, pero también permite configurar las ya existentes o definir nuevas transformaciones según la necesidad. Para ello utiliza un lenguaje de generación de plantillas que mediante la utilización de macros convierte los elementos elegidos del diagrama fuente en elementos del diagrama destino.

En el caso de querer realizar una transformación de un diagrama de clases a un diagrama relacional, la transformación de los atributos de la clase a columnas de la tabla, se definiría mediante la siguiente plantilla:

```
Column
{
   %TRANSFORM_CURRENT("type", "stereotype", "collection", "constant",
   "containment", "ordered", "static", "volatile")%
   type=%qt%%CONVERT_TYPE(genOptDefaultDatabase,attType)%%qt%
}
```

Básicamente utiliza dos macros. La primera es `TRANSFORM_CURRENT` que realiza una copia de todas las propiedades del ámbito de la plantilla actual (atributos), pudiendo decidir qué propiedades se quedan fuera de la transformación (en el ejemplo sería la lista de parámetros excepto el primero). La transformación que realiza de cada atributo es sobre el tipo de dato (el primer parámetro) y ésta se especifica en la siguiente instrucción en donde se usa la macro `CONVERT_TYPE` que transforma el tipo de dato especificado en el atributo al tipo de dato correspondiente, definido en la base de datos especificada por defecto para el modelo (`genOptDefaultDatabase`). Esta correspondencia se define dentro de la opción de parametrización de los tipos de datos para las diferentes bases de datos. En el ejemplo de la Figura 5-4, se observa que se ha decidido transformar el tipo de dato común `char` al tipo `VARCHAR2` de Oracle, con una longitud por defecto de 50 posiciones.

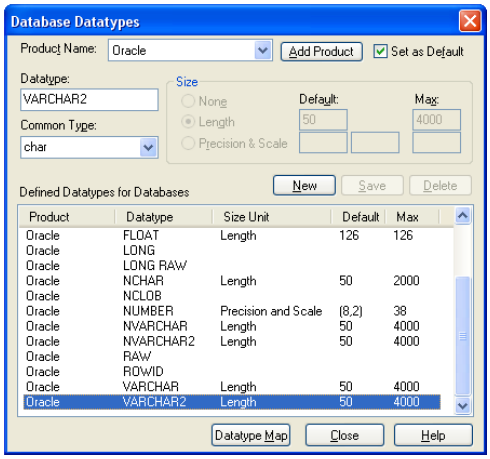

Figura 5-4. Transformación de tipos de datos en EA

Con esta definición de transformación, se observa según la Figura 5-5, que todos los atributos de la clase Cliente, de tipo `char`, se han convertido a columnas con tipo `VARCHAR2(50)`.

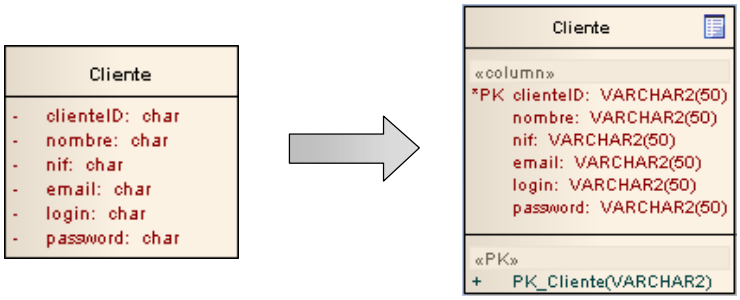

Figura 5-5. Transformación de diagrama de clases a diagrama relacional en EA

Esto es aplicable tanto a las transformaciones *modelo-a-modelo*, como las de *modelo-a-código*. Por el mismo procedimiento se puede generar el código fuente que implementen los modelos definidos a cualquiera de los lenguajes por defecto o los que se definan nuevos en la herramienta. Cabe destacar de este método de transformaciones que el sistema de plantillas y macros utilizado, no utiliza el estándar QVT, ni un sistema de marcado de modelos PIM.

La herramienta, en cuanto a la generación de código, tiene un abanico de lenguajes de programación destino aceptable, aunque destaca el que no da soporte a la creación de JSP ni de ASP [Sparx 2005b]. En cuanto a la generación de scripts de base de datos, destaca el amplio espectro de diferentes lenguajes de definición de datos (DDL) que soporta por defecto EA.

P03 - Uso de patrones

Enterprise Architect permite la utilización de patrones de diseño en la construcción de diferentes modelos o en la generación de código, de manera que puedan aplicarse repetidamente y sin esfuerzo soluciones encontradas a problemas específicos. Existe una plantilla básica de patrón en EA, que puede ser reutilizada modificando el nombre de las variables de cada proyecto concreto.

El patrón se debe crear como un diagrama UML estándar y salvarlo como XML. Para poder ser utilizado, se debe importar el XML como un recurso UML, en el diagrama que se desee utilizar.

P04 - Control y refinamiento de las transformaciones

Aparte de la posibilidad que existe de la modificación de las plantillas para efectuar las transformaciones, es posible también, sin necesidad de realizar un esfuerzo en el aprendizaje del lenguaje de generación de plantillas, que permiten al usuario refinar o influir en determinados parámetros de la transformación. Uno de ellos es mediante la ventana de configuración de tipos de datos de la base de datos, vista con anterioridad. Así mismo, en el momento de realizar la transformación, EA permite elegir tanto los elementos a transformar como qué tipo de transformación se quiere. En el ejemplo de la Figura 5-6 se muestra cómo se han seleccionado sólo tres de las cinco clases del diagrama de clases de *PetStore* y se pretende realizar la transformación a EJB de entidades y a tablas del modelo de datos.

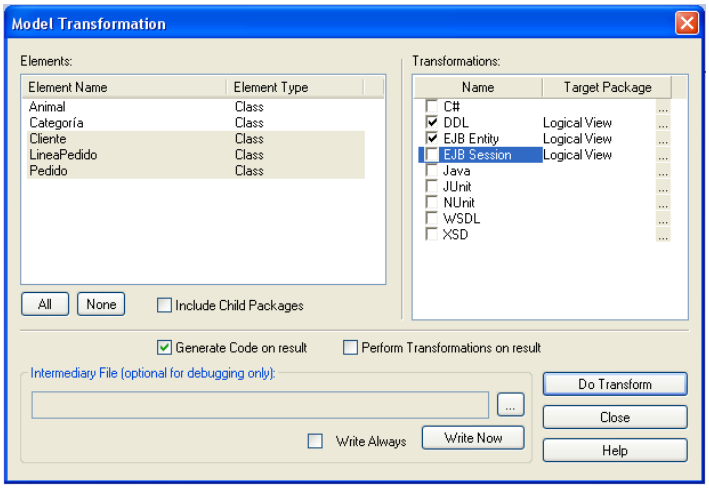

Figura 5-6. Facilidad de transformaciones parciales en EA

A la hora de definir el diagrama, en las ventanas de propiedades, se pueden elegir determinados aspectos, que a modo de marca del modelo PIM, van a servir para facilitar las transformaciones. En el caso de las clases se puede definir el lenguaje destino de la transformación, que permitirá a la hora de definir los atributos utilizar tipos de datos específicos del lenguaje indicada, en vez de unos generales.

P05 – Trazabilidad

La trazabilidad de *Enterprise Architect* se consigue mediante la ventana de jerarquía de los objetos. En esta ventana se puede ver el camino que ha seguido un determinado objeto del modelo en sus transformaciones desde el PIM original hasta la generación de código y también la dependencia con otros objetos en forma de árbol, tal y como se muestra en la Figura 5-7. Cada vez que se ejecuta una transformación sobre un objeto determinado, EA crea automáticamente una dependencia de transformación que permite trazar el camino que recorre.

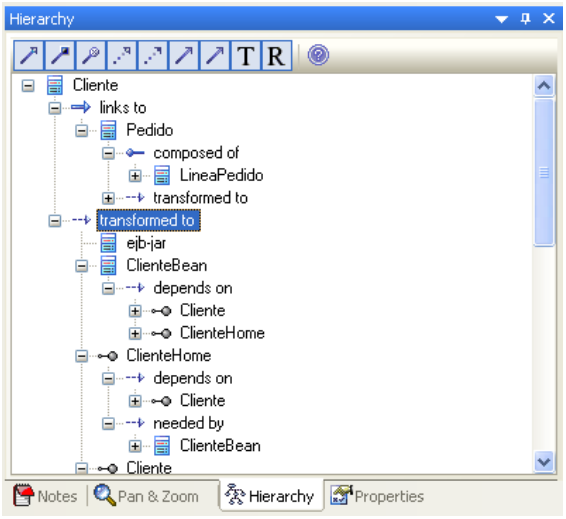

Figura 5-7. Ventana de Jerarquía de EA

Si se desean definir otro tipo de dependencias entre elementos como requisitos, casos de uso, clases, paquetes y otros artefactos, EA permite crear diagramas específicos de trazabilidad. Estos diagramas se crean mediante un tipo de diagrama especial (*Custom Diagram*), extensión de los diagramas de clase.

Otro tipo de herramienta que permite la trazabilidad es el informe de dependencias. Mediante este informe se muestran las dependencias entre los diferentes elementos definidos en un paquete, mostrando de cada uno de ellos de quien depende. Igualmente mediante la matriz de relaciones, EA permite consultar y definir dependencias entre diferentes objetos del modelo. La ventaja de la matriz es que desde ésta se pueden definir directamente las dependencias. Por ejemplo se podría una nueva dependencia *Realization* entre un requisito y un caso de uso. En la Figura 5-8, se muestra un ejemplo de una matriz de relaciones y del informe de dependencias del ejemplo estudiado de *PetStore*.

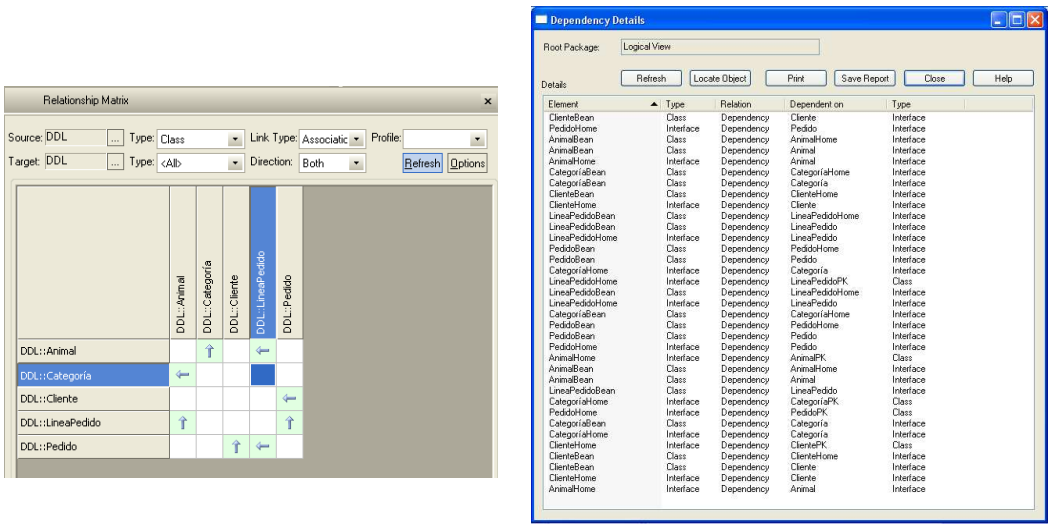

Figura 5-8. Matriz de relaciones e informe de dependencias de EA

P06 - Soporte para regeneración de modelos

El soporte para la regeneración de modelos no está soportado por EA. Permite realizar una sincronización de un modelo con el código generado y viceversa. Es decir, si se modifica el código generado en una transformación, EA permite actualizar el modelo partiendo del código modificado. Mediante la sincronización, las modificaciones realizadas en el código se reflejan en el modelo equiparando ambos. La sincronización se puede hacer en ambos sentidos. Se comprueba que esto se pude realizar con la generación de los lenguajes soportados por EA, aunque se observa que EA no soporta la sincronización con un script especifico de Oracle.

A nivel de transformación entre modelos, esto no se da. Si se modifica un modelo resultado de una transformación, no se puede sincronizar con el modelo fuente y tampoco se respetan las modificaciones realizadas en el modelo destino, si se vuelve a ejecutar la transformación. Más arriba se ha puesto el ejemplo de la transformación de la clase Cliente de un modelo PIM a la tabla Cliente de un modelo PSM relacional. En el ejemplo mostrado se observaba como el tipo común se convertía en un tipo de datos `VARCHAR2(50)` de Oracle. Se ha probado a modificar la definición de las longitudes de las columnas de la tabla y volver a ejecutar la transformación, observándose que no se respetan las modificaciones.

P07 - Verificador de Modelos

Enterprise Architect permite realizar una validación de modelos en base a unas reglas de modelado y sobre la corrección de restricciones OCL definidas en el mismo [Sparx 2005b]. Permite realizar validaciones sobre un elemento del modelo, un diagrama o un paquete completo. En el ejemplo de la Figura 5-9 se muestra un ejemplo de error de validación sobre un de diagrama en la que la clase `Cliente` se generaliza a sí misma, algo que no está permitido según la sintaxis UML.

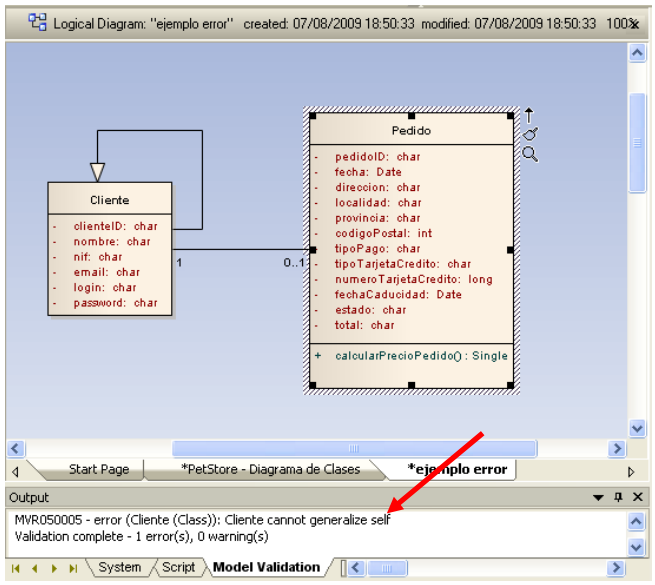

Figura 5-9. Ventana de errores de validación de modelos de EA

Durante la validación revisa el modelo según los criterios especificados a continuación, siempre que se activen en la herramienta para ser utilizados:

- Elementos, relaciones, características y diagramas bien formados
- Composición de elementos
- Validez de las propiedades de las relaciones y elementos
- Corrección de las expresiones OCL

Las restricciones OCL se pueden definir a nivel de elemento, relación y de atributos. EA valida la corrección de la sintaxis de las expresiones OCL utilizadas.

5.5 Comparativa RSA, *Together* y EA

A la vista de los resultados obtenidos mostrados en la Tabla 5-3, se observa que en general, las herramientas estudiadas cumplen en alguna medida con la mayoría de las características analizadas. En los tres casos, sólo se deja de cumplir una de ellas.

La característica de la interoperabilidad y estandarización, se cumple en todas ellas de una forma limitada. La interoperabilidad la cubren todas, permitiendo la exportación mediante XMI, a pesar de que soportan determinados estándares OMG, el conjunto de los soportados es reducido.

De hecho, llama la atención de que las dos herramientas de las dos grandes compañías como son RSA y *Together* hayan optado por la estandarización proporcionada por Eclipse, en vez de basarse en los de OMG.

La posibilidad de acceso a la definición de las transformaciones se cumple en todas ellas, aunque EA lo hace de una forma muy particular, mediante plantillas definidas en un lenguaje propio, lo que no permite reutilizar transformaciones QVT de otras herramientas. Lo mismo ocurre con RSA, que tiene un conjunto de transformaciones definidas, que permite configurar, pero no deja crear nuevas transformaciones de forma estándar mediante QVT. Así mismo, todas ellas permiten refinar y modificar las transformaciones existentes, así como la creación de nuevas. Sólo *Together*, en base a la documentación estudiada, permite decidir las transformaciones a ejecutar de forma algo limitada, pero al no poder ejecutar la herramienta no se ha podido comprobar este punto en detalle.

Id	Propiedad	RSA	Together	EA
P01	Interoperabilidad y estandarización	☞	☞	☞
P03	Acceso a la definición de las transformaciones	☞	👍	☞
P03	Uso de patrones	👍	👍	👍
P04	Control y refinamiento de las transformaciones	👍	☞	👍
P05	Trazabilidad	👎	☞	👍
P06	Soporte para regeneración de modelos	☞	☞	👎
P07	Verificador de Modelos	👍	👎	👍

Tabla 5-3. Tabla resultado del análisis de las herramientas MDA

La trazabilidad en las transformaciones tiene un comportamiento desigual. La única que la cumple es EA, que permite realizar un seguimiento de las dependencias de los modelos, derivadas de las transformaciones. *Together*, según su documentación, cumple con esta característica, aunque no se ha podido comprobar, por eso se ha indicado que soporta esta característica, aunque no de forma completa. La que no la soporta es RSA, que al igual que *Together*, sólo soporta trazabilidad en cuanto a requerimientos del sistema.

En cuanto al resto de características, todas las herramientas soportan satisfactoriamente el uso de patrones y sólo *Together* no cumple con la característica de verificación de modelos.

Tras el estudio del soporte de esta propiedades y teniendo en cuenta que la amplitud de las herramientas dificulta tener un control completo sobre ellas, se puede concluir que todas soportan razonablemente bien la aproximación MDA. Como apreciación personal, cabe destacar que tanto *Together* como RSA, son herramientas mucho más completas, que abarcan otro tipo de elementos que complementan la metodología de desarrollo dirigido por modelos, pero por ello son mucho más difíciles de profundizar en ellas y son mucho menos intuitivas que EA, que tiene un alcance mucho menor.

6. APLICACIONES PRÁCTICAS EN LA INDUSTRIA

6.1 Introducción

La investigación con respecto a MDA, es muy extensa. Existe un gran volumen de publicaciones y estudios sobre MDA que aportan conocimiento y experiencias de aplicación práctica de la misma, pero como se ha mencionado en la introducción, aparentemente predominan los autores del entorno académico, existiendo en proporción, poca presencia de la industria como impulsores de estos estudios. Se observa que la mayoría de las ponencias presentadas en conferencias o talleres realizados en el ámbito de MDA, suelen ser de autores del ámbito académico.

Una de las conferencias existentes a nivel internacional es la conferencia MoDELS [MoDELS] que se realiza de de forma anual y se centra en la ingeniería dirigida por modelos propulsada por MDA. Inicialmente se llamaban *Unified Modeling Language Conferences* [UML Conf], pero desde el 2005 se han pasado a denominar *MoDELS Conferences,* que amplían y redireccionan las anteriores. Las conferencias MoDELS son un punto de encuentro para el intercambio y la innovación tecnológica entorno a ideas y experiencias relativas a la arquitectura MDA en desarrollo de sistemas Software. No es la única. Existen diferentes conferencias a nivel internacional como la *European Conference on Model Driven Architecture* (EC-MDA) [ECMDA] enfocada en la difusión del conocimiento y el impulso de la industrialización de la metodología MDA, procurando conseguir una industria fuerte y más eficiente en cuanto al desarrollo de software fiable. También existe la *Modeling in Software Engineering* (MiSE), taller que se celebra dentro del marco de la *International Conference of Software Engeenering* [ICSE], centrada sobre todo en la promoción del uso de modelos en los sistemas software. A nivel nacional, también existen conferencias como el Taller sobre el Desarrollo de Software Dirigido por Modelos enmarcada en conferencia JISBD (Jornadas de Investigación en Ingeniería del Software y Bases de Datos) [JISBD].

El objetivo de este capítulo es analizar las ponencias de una conferencia concreta, de manera que, partiendo del número de autores provenientes de la industria, del las empresas que participan y el sector al que pertenecen o de la temática de las mismas, se puedan extraer conclusiones sobre si la industria ha acogido con interés la aproximación MDA.

Dentro del conjunto de conferencias existentes, se ha elegido la conferencia MoDELS, frente a EC-MDA, MiSE o JISBD, por varias razones. La primera de ellas es por su dimensión internacional. Resulta mucho más objetivo realizar este estudio sobre una conferencia que tenga un amplio espectro en cuanto a la filiación de sus ponentes, que una de ámbito nacional. Por otro lado, es una conferencia con cierto arraigo. En el año 2009 se va a celebrar la décimo segunda edición, por lo que la convierte en una conferencia con una mayor experiencia frente a las otras candidatas internacionales. Por último, es una conferencia con una doble vertiente de la exposición de ponencias del ámbito de la investigación y de la innovación tecnológica y de aquellas que repasan experiencias concretas donde se aplica MDA, sin estar exclusivamente enfocada en la industria, como es el caso de EC-MDA.

6.2 Conferencia MoDELS

Durante la pasada década hemos sido testigos de una explosión por el interés en técnicas avanzadas para expresar el diseño con un alto nivel de abstracción. Los lenguajes de modelado específicos de dominio, la ingeniería dirigida por modelos, las transformaciones de modelos y la estandarización de lenguajes de modelado, son algunas de las más significativas nuevas

tendencias. La conferencia MoDELS congrega profesionales e investigadores para presentar y discutir tanto experiencia práctica, como técnicas innovadoras en relación a MDA.

Dentro de las ponencias que se presentan en MoDELS, existen dos categorías diferenciadas: (i) ponencias científicas que muestran investigaciones sobre la ingeniería dirigida por modelos y otros aspectos del modelado en los procesos de desarrollo; y (ii) ponencias experimentales que cuentan experiencias reales de proyectos concretos.

La conferencia MoDELS suele incluir ponencias, documentos y presentaciones en diversas categorías relacionadas con la MDA. Dichas categorías pueden verse por años en las en las cuatro primeras tablas.

En este trabajo se han estudiado las ponencias de las últimas cuatro ediciones de MoDELS (es decir, desde que cambio su denominación de *UML Conference* a la actual), con el objeto de ver la presencia de estudios empíricos o de científicos donde esté representada la industria. Para ello se han elaborado una serie de tablas donde, agrupadas por categorías, se muestra el recuento de las ponencias que han sido elaboradas por investigadores o grupos de trabajo pertenecientes a la universidad o realizadas por personas relacionadas con empresas. Se han contado como ponencias de la industria, aquellas en las que al menos uno de sus autores pertenezca a una empresa privada. También se han considerado aquellas ponencias que, aunque escritas únicamente por miembros del mundo académico, describen desarrollos MDA en la empresa.

En la Tabla 6-1 se muestra la relación de ponencias de la conferencia realizada en el año 2005 en Montego Bay, Jamaica [Briand 2005]. En esta conferencia de las 51 presentaciones realizadas 12 de ellas (23,5%) están relacionadas con la industria.

Categoría	**Academia**	**Industria**
Procesos	3	0
Familias de Productos, Reutilización	3	0
Estado / Situación del Modelado	2	1
Aspectos	3	0
Estrategias de Diseño	3	0
Transformación de Modelos	3	0
Refactorización de Modelos	3	0
Control de Calidad	1	2
MDA	4	1
Automatización	4	2
UML 2.0	1	2
Experiencia de la Industria	0	3
Crosscutting Concerns	3	0
Estrategias de Modelado	5	0
Keynote	1	1

Tabla 6-1. Ponencias MoDELS 2005

En Montego Bay, la industria elaboró estudios sobre el estado o *situación actual de modelado*, sobre el *control de calidad*, sobre *UML 2.0* y sobre *automatización*. Cómo no, bajo la categoría de la *experiencia de la industria*, las tres ponencias existentes son de autores del ámbito de la industria. En las áreas de mayor presencia industrial, se presentan ponencias donde se exponen casos prácticos llevados a cabo por Siemens, Motorola e IBM y estudios sobre lenguajes de modelado y herramientas.

En la Tabla 6-2 se muestra la relación de ponencias de la conferencia realizada en el año 2006 en Génova, Italia [Nierstrasz 2006]. En esta conferencia de las 53 ponencias las relacionadas con la empresa fueron 10 (un 18,9%).

Categoría	Academia	Industria
Evaluando UML	3	0
MDA en el Desarrollo de Software	1	2
Sintaxis Concreta	2	0
Aplicando UML a la Interacción y Coordinación	1	1

Categoría	Academia	Industria
Aspectos	2	0
Integración de Modelos	3	0
Semánticas Formales de UML	3	0
Seguridad	3	0
Herramientas de Transformación de Modelos	3	0
Analizando Modelos Dinámicos	2	1
Especificando Transformaciones	3	0
MOF	2	1
Puentes entre Modelos	3	0
Riesgos, Confianza y Fiabilidad	1	2
Entornos de Herramientas	2	1
OCL	3	0
Roundtrip Engineering	3	0
Sistemas Empotrados y de Tiempo Real	1	2
Keynote	2	0

Tabla 6-2. Ponencias MoDELS 2006

En Génova destacaron sobre todo las ponencias relacionadas con *MDA en el desarrollo de software*, *riesgos confianza y fiabilidad* y *sistemas empotrados*. Así, por ejemplo dentro de esta última categoría la empresa C_LAB, coparticipada por Siemens y la Universidad de Paderborn en Alemania, describe un escenario de herramientas de construcción para sistemas empotrados. Otro ejemplo es el de BMW, que presenta una semántica formal del lenguaje de modelado UML-RT, dialecto de UML.

En la Tabla 6-3 se muestra la relación de ponencias de la conferencia realizada en Nashville, USA en el 2007 [Engels 2007]. Este año se realizaron 12 ponencias relacionadas con la industria de un total de 45. Esto supone el 26,7% del total.

Categoría	Academia	Industria
Transformación de Modelos	3	0
Restricciones al Modelado	3	0
Meta-Modelado	3	0
Modelos Consistentes	1	2
Soporte al Modelado	3	0
Diseño de Interfaz de Usuario	2	1
Definición de Lenguajes	1	2
Métodos de Modelado	0	3
Modelado de Servicios y Procesos	2	1
Análisis de Modelos	3	0
Procesos de Modelado	1	2
Aspectos	3	0
Características de Nuevos Lenguajes	2	1
Soporte a la Depuración	3	0
Diagramas de Estado	3	0

Tabla 6-3. Ponencias MoDELS 2007

Este año, destacan sobre todo las ponencias sobre *consistencia de modelos*, *definición de lenguajes*, *métodos de modelado* y *procesos de modelado*. Gran parte de las ponencias de estas categorías son aproximaciones o definición de metodologías basadas en experiencias previas. Cabe destacar también la presencia de grandes empresas como IBM, Ericsson, Motorola o General Dynamics.

Finalmente, en la Tabla 6-4 se muestra la relación de ponencias de la conferencia realizada en el año 2008 en Toulouse, Francia [Czarnecki 2008]. En esta conferencia de las 61 ponencias realizadas, 13 de ellas (21,3%) están relacionadas con la industria.

Categoría	Academia	Industria
Transformación de Modelos: Principios y Técnicas	6	0
Modelado de Requisitos	2	1
Modelado Específico del Dominio	1	2
Composición y Análisis de Modelos de Comportamiento	3	0
Comprensión de Modelos	3	0
Gestión de Modelos	3	0
Aprobación de Comportamiento y Refinamiento	2	1
Metamodelado y Modularidad	3	0
Restricciones	3	0
Análisis de Modelos	3	0
Arquitecturas Orientadas a Servicios	1	2
Sistemas Adaptables y Autónomos	2	0
Estudios Empíricos	2	0
Evolución e Ingeniería Inversa	2	1
Semánticas de Lenguaje de Modelado	3	0
Análisis de Fiabilidad y Testeo	3	0
Modelado Orientado a Aspectos	2	1
Modelado Estructural	2	1
Sistemas Empotrados	0	3
Keynote	2	1

Tabla 6-4. Ponencias MoDELS 2008

En la última edición celebrada en Toulouse, destaca la presencia de la industria en *sistemas empotrados*, *modelado específico del dominio* y *arquitecturas orientadas a servicios* que ha despuntando últimamente en la industria. Otra vez cabe destacar la participación de empresas centradas en investigación como SINTEF, y grandes empresas como IBM, Siemens, Mitsubishi o Lockheed Martin.

6.3 Análisis

Teniendo en cuenta la información presentada en las tablas anteriores, se observa que el número de ponencias relacionadas con la industria en las diferentes conferencias MoDELS asciende a 47 de las 210 presentadas en total, dejando una media de 23,4%.

Categoría	Ponencias
Sistemas empotrados	7 (15%)
Modelos de comportamiento	5 (11%)
Análisis de modelos	4 (9%)
Desarrollo de Software	4 (9%)
UML 2.0	4 (9%)
Requisitos	4 (9%)
Herramientas	3 (6%)
Aspectos	3 (6%)
Casos de estudio industriales	2 (4%)
Metamodelado	2 (4%)
Seguridad	2 (4%)
Arquitectura orientada a servicios	2 (4%)
Pruebas	2 (4%)
Automatización de diseños	1 (2%)
Modelado específico de dominio	1 (2%)
HCI	1 (2%)
Calidad	1 (2%)
Seguridad en sistemas críticos	1 (2%)

Tabla 6-5. Distribución de Ponencias de la Industria por Categorías

Debido a que las categorías específicas de cada año difieren sobremanera, se ha optado por definir una serie de categorías que permitan agrupar convenientemente las ponencias de las distintas conferencias. La Tabla 6-5 muestra esta categorización.

Se observa que donde más destacan son las relativas a *sistemas empotrados*, así mismo también se observa que también destacan las ponencias centradas en *Modelos de comportamiento*, *análisis de modelos*, *desarrollo de software* y *UML 2.x*. En definitiva todas estas categorías aglutinan 18 de las conferencias presentadas por la industria.

La Tabla 6-6 analiza las ponencias en base a los autores de las mismas. Nótese que al haber distintos autores de distintas empresas en la misma ponencia, el número de ponencias por empresa sobrepasa el número de ponencias totales.

Empresa	Ponencias
IBM	6
SINTEF	6
Siemens	5
BMW	3
Ericsson	3
ABB	2
CEA LIST	2
Motorola	2
Simula	2
Aerospace Corporation	1
Airbus	1
Amazon.com	1
ASML	1
Boeing	1
C_LAB	1
Combitech	1
Elektrobit	1
France Telecom	1
General Dynamics	1
Hexacta S.A	1
Infovide	1
IT-Telecom	1
Lockheed Martin	1
MetaCase	1
Mitsubishi	1
Nokia	1
Pragma	1
Sodifrance	1
SuperOffice	1
Telenor	1

Tabla 6-6. Distribución de Ponencias por Empresas

Las empresas que participan más en este tipo de conferencias son grandes multinacionales y empresas centradas en investigación. Las pequeñas o no tan conocidas, son las que, por norma general participan poco. Hay una curiosidad destacable y es que, en cuanto a las que a lo largo de los últimos cuatro años, sólo han participado una vez, suelen ser ponencias conjuntas de autores pertenecientes a la industria junto con algunos del ámbito académico. Esto es una norma, salvo en el caso de las aeronáuticas, Nokia, CEA LIST, ABB y C_LAB.

Sacar alguna conclusión de este hecho es complicado. Puede que sean estudios conjuntos derivados de subvenciones a proyectos de investigación en las cuales sea obligatoria la presencia de la industria, por una simple relación existente entre ambas organizaciones o por una búsqueda de publicidad en el ámbito de la investigación para las empresas. La verdad es que no hemos encontrado ninguna conclusión de peso en este punto.

Se puede también hacer una comparativa por los sectores a los que pertenecen estas empresas. Aquí también resulta complicado dividirlas, ya que muchas de ellas, sobre todo las grandes multinacionales, abarcan muchos ámbitos de negocio. La Tabla 6-7 muestra una posible distribución.

Sector	Ponencias
Empresas investigación en TIC	11
Informática	9
Telecomunicaciones	9
Aeroespacial	6
Consultoría	3
Semiconductores	3
Motor	3
Conglomerados industriales (no especifica sector)	2
Robótica	2
Almacén online	1
Relaciones empresariales	1
Transporte	1

Tabla 6-7. Distribución de Ponencias por Sectores Empresariales

Se observa precisamente que las empresas dedicadas a la investigación en Tecnologías de la Información y Comunicación (TIC) son las más prolíficas en ponencias MDA. Este sector está seguido muy de cerca por las empresas de informática y telecomunicaciones. El sector aeroespacial, también está bien representado.

Una vez visto las empresas que están presentes en las conferencias, cabría preguntarse cuales son las que no están. Las que se echan en falta sobretodo, son las grandes consultoras. Cabe por tanto preguntarse, por qué no están presentes las empresas cuyo principal activo es el proporcionar servicios a sus clientes [Fernández 2009].

7. CONCLUSIONES Y TRABAJO FUTURO

Como se ha comentado en la introducción, la aproximación MDA nació hace casi una década, por lo que es un buen momento para poder hacer una visión retrospectiva y ver hasta qué punto ha tenido calado en el desarrollo de software. Con este objetivo, se han ido desgranando a lo largo de este trabajo, los elementos que faciliten extraer conclusiones del análisis realizado sobre esta aproximación.

7.1 Aproximación MDA

En primer lugar, del estudio de la aproximación MDA podemos extraer varias conclusiones. La principal de ellas es la abstracción promovida por la aproximación. Los sistemas se pueden definir y diseñar desde un alto grado de abstracción, lo que permite a los desarrolladores aislarse de los detalles técnicos. El procedimiento de diseño se hace mediante modelos especificados en un lenguaje de modelado preferentemente no textual. Esto simplifica la generación, entendimiento y modificación de los modelos a desarrollar. Además, estos modelos facilitan la comunicación con los usuarios y a la vez son el germen del código de la aplicación, ya que éste se deriva de las transformaciones sucesivas sobre los mismos.

La característica fundamental de los modelos es que pueden ser procesados por un computador, por lo que las trasformaciones se pueden hacer de forma automática. Esta es una de las características por las que MDA resuelve el problema de la productividad, derivado del desarrollo de software tradicional. MDA permite aumentarla básicamente por dos motivos: (i) la agilidad en la generación y actualización de los modelos (ii) la generación automática de transformaciones sucesivas cuyo resultado final es el código de la aplicación en un lenguaje determinado. También se ha visto que se resuelven los problemas de la portabilidad y de la interoperabilidad, en ambos casos sustentándose una vez más en las transformaciones. La portabilidad se resuelve mediante transformaciones que generan modelos PSM de la nueva plataforma a la que se desee portar el sistema. La interoperabilidad, mediante la construcción de puentes entre modelos de plataformas diferentes, que permiten la comunicación entre ambos.

En definitiva, las transformaciones juegan un papel de peso en la aproximación MDA, pero no todo son ventajas. Las transformaciones no son siempre automáticas. Dependiendo del grado de automatismo de las mismas, la ventaja de la productividad puede verse mermada. Así mismo, las transformaciones son bastante complejas de definir, principalmente porque hay que conseguir una ausencia total de ambigüedad. Normalmente las herramientas que dan soporte a MDA tienen definidas las reglas de transformación. No obstante, en el momento en que aparezca una nueva versión de un lenguaje de programación o nazca uno nuevo, las herramientas CASE quedarían desactualizadas. Los desarrolladores quedarían entonces supeditados a que las herramientas sean adaptadas por los fabricantes o a tener expertos que sean capaces de definir las nuevas transformaciones necesarias.

En cuanto a la definición de la arquitectura de metamodelo de cuatro capas que define MDA, es importante destacar el concepto de meta-metamodelo asociado al nivel M3 de la arquitectura. Éste permite definir nuevos metamodelos que resuelvan cualquier tipo de problema específico de dominio. Otra de las ventajas que proporciona esta arquitectura en cuatro capas es que subiendo a los niveles superiores de la arquitectura se pueden relacionar tipos de metadatos diferentes o intercambiar modelos y metadatos entre elementos que usen el mismo meta-metamodelo.

7.2 Estándares MDA

En el capítulo 3, mediante el estudio de los estándares definidos por OMG, se observan y entienden mejor las características de MDA. Una de las aportaciones de este trabajo ha sido la de analizar y contextualizar la jerarquía de los estándares OMG en la arquitectura de cuatro capas, viendo claramente cómo se relacionan entre sí. Como núcleo fundamental, se encuentra la *InfrastructureLibrary* que define los constructores básicos y conceptos comunes que se van a reutilizar para definir los distintos metalenguajes y que alinea arquitectónicamente a MOF y UML. MOF es un lenguaje para la especificación de metamodelos, mientras que UML es un metamodelo basado en MOF para modelar sistemas. La *InfrastructureLibrary* está situada a nivel del meta-metalenguaje y va a permitir definir los metalenguajes como instancias suyas. A su vez, los paquetes *Core* y *Profiles* que la componen, permite dos mecanismos diferentes para la creación de metamodelos. Mediante *Core* se pueden extender metamodelos con la utilización o especialización de sus metaclases. Mediante *Profiles* se pueden personalizar metamodelos existentes consiguiendo una especialización en una plataforma específica o en un dominio particular. Esta es una característica importante ya que facilita la generación de metamodelos partiendo de la especialización de otros, sin tener que crear uno desde cero y sin modificar la sintaxis ni la semántica del metamodelo original. Este mecanismo tiene las limitaciones inherentes a la definición del propio metamodelo de partida. Se va a poder especializar un metamodelo, pero no se va a poder definir uno radicalmente diferente. No obstante, en función de las necesidades, siempre existe la posibilidad de definir un nuevo lenguaje basado en MOF.

Cabe destacar que tanto éstos, como el resto de estándares explicados en el capítulo 3, tienen una gran complejidad y extensión. No son estándares sencillos y esto puede resultar poco atractivo a la hora de implementar herramientas CASE que lo soporten. Un claro ejemplo de su complejidad es la simplificación que Eclipse ha hecho al implementarlos, con el objetivo de facilitar su uso. De hecho, se ha visto en este mismo estudio que la dos grandes herramientas que dan soporte a MDA, *IBM Rational Software Architect* y *Borland Together*, han optado por basarse en la implementación de los estándares Eclipse y no implementar los de OMG.

7.3 *Eclipse Modeling Project*

El estudio de *Eclipse Modeling Project* ha sido un elemento importante en cuanto al interés que puede suscitar en la industria. El interés de Eclipse en determinadas tecnologías o aproximaciones es un indicador de referencia de lo beneficioso que estas pueden resultar para las comunidades de desarrolladores. En el caso de MDA, llama la atención el estado de los subproyectos que están abiertos actualmente y del tiempo que llevan en marcha. Destaca que sólo se encuentran en un estado maduro los subproyectos EMF y GMF, el resto están todavía en estado de incubación. Parece razonable que se dé prioridad a estos dos subproyectos. Por un lado, EMF (en particular *Ecore*) es el equivalente a MOF, el núcleo de toda la arquitectura dirigida por modelos, y GMF es la herramienta que permite editar los modelos que representen los sistemas.

Lo que más llama la atención del proyecto Eclipse es que ha optado por la definición de sus propios estándares, que aunque se encuentran alineados con los estándares OMG, no son una simple implementación de los mismos, sino que son en sí unos estándares de facto. En el capítulo 4 se han relacionado éstos estándares con los definidos por OMG y se puede apreciar que son un espejo de los mismos. Lo que se ha visto en el estudio del proyecto Eclipse es la simplificación de los estándares respecto a los de OMG.

Se podría hacer un paralelismo entre los estándares SGML/XML [W3C 1997] y la respuesta de la industria en relación a MDA. SGML [ISO 1986] se podría equiparar a los estándares OMG, mientras que XML [W3C 2006] sería equiparable a los estándares de facto Eclipse. A la vista está

que la simplificación propuesta en XML ha sido ampliamente utilizada. Aunque si se equipara XML con EMF, se observa que el primer ha tenido un éxito mucho más arrollador que el segundo.

Otro elemento a tener en cuenta es la operatividad de Eclipse para la generación de los modelos. Para poder generar un modelo desde el que realizar la generación del código de una aplicación, es necesario generar, mediante cualquier otra aplicación CASE, el metamodelo sobre el que se desea trabajar. Hay que exportarlo a XMI para que EMF lo entienda. Una vez hecho esto, hay que mapear el metamodelo con las definiciones de los elementos gráficos para poder crear y editar los modelos con GMF. Esto no parece muy operativo, ya que si partimos de una herramienta CASE, se podría generar el código directamente desde ésta, sin tener que dar todos estos pasos. En definitiva, esto hace pensar que el proyecto de modelado de Eclipse, todavía no esté lo suficientemente avanzado e implantado. En el aspecto positivo, GMF, permite definir herramientas visuales para la construcción de modelos para cualquier metamodelo descrito en EMF.

Lo que se observa es que Eclipse ha contribuido significativamente a la implementación de la aproximación de MDA, proporcionando una plataforma *open source* y una implementación de muchas de las especificaciones de MDA. Curiosamente, a pesar de existir un alineamiento entre cada uno de los subproyectos del *Eclipse Modeling Framework*, no parece que haya existido este alineamiento entre OMG y la comunidad Eclipse. Hasta hace poco tiempo, no existe una relación formal entre ambas organizaciones, ya que Eclipse no era miembro de OMG y ésta a su vez, tampoco era miembro de Eclipse [Gronback 2006]. No se puede extraer la conclusión de que si hubiera habido una colaboración formal más temprana entre ambos, se habría incrementado el éxito en la adopción del desarrollo dirigido por modelos, pero probablemente habría tenido un efecto positivo. Hoy por hoy, esta relación existe, aunque cabe preguntarse qué frutos va a dar esta colaboración [Gronback 2008].

7.4 Herramientas MDA

El análisis de las tres herramientas CASE estudiadas, apunta a que todas ellas cumplen con la práctica totalidad de las características analizadas. Estas características son algunas de las más representativas de MDA y aunque no son todas, se puede concluir que soportan en alto grado esta aproximación.

Como se ha mencionado en el estudio, todas ellas en mayor o menor medida soportan la interoperabilidad entre herramientas e implementan algunos de los estándares MDA, aunque no todos. También tienen un alto grado de soporte, tanto en la generación de transformaciones como en la redefinición de las mismas. Lo único destacable es que salvo *Together*, que implementa QVT, las otras no soportan este estándar y han definido mecanismos propios para poder definirlas.

En cuanto a los estándares, llama la atención de que dos grandes empresas, como son IBM y Borland, hayan decidido desarrollar sus productos basados en la plataforma Eclipse, utilizando los estándares implementados por ésta ajo el proyecto EMP. Esto permite focalizar los esfuerzos de desarrollo de la herramienta, en el cumplimiento de los requisitos funcionales de la misma, sin tener que invertir en la implementación de las especificaciones de los estándares OMG. La reutilización de las implementaciones Eclipse, permite que el inconveniente mencionado sobre la operatividad de los subproyectos Eclipse en el apartado anterior, sea solventado. La construcción de una herramienta CASE sobre la plataforma Eclipse, tal y como ocurre con RSA y *Together*, permite que todos los subproyectos del proyecto *Modeling*, queden encapsulados bajo éstas y así se pueda gestionar la posible falta de operatividad con elementos de la misma.

La gran paradoja que se produce es que las herramientas basadas en una plataforma *open source*, son precisamente herramientas comerciales de gran envergadura y, por su puesto, de un alto coste, tanto en la adquisición como en el soporte posterior.

7.5 Aplicaciones prácticas en la industria

Viendo las estadísticas mostradas en el capítulo 6 sobre la conferencia MoDELS se pueden extraer como resumen, varias ideas básicas. En primer lugar, la industria está presente en conferencias sobre MDA, aunque se detecta que en muchos casos van de la mano de colaboradores del mundo académico. Así, se presentan ponencias en donde los autores no son exclusivamente del ámbito industrial.

Las categorías en las que más se centra la industria son en los sistemas empotrados, modelos de comportamiento, análisis de modelos y UML 2.x. Además, existe una gran fragmentación en los intereses de la industria. De hecho, fueron necesarias 18 categorías distintas para clasificar los artículos provenientes de este ámbito.

Tal vez, lo que más destaca, es la presencia de grandes multinacionales que se dedican a procesos productivos, aparte de otras posibles tareas. Muchas participan varias veces, no sólo en años diferentes, sino en la misma conferencia dentro de diferentes categorías.

Partiendo de este análisis, se pueden extraer una serie de conclusiones sobre la presencia de la industria en la conferencia MoDELS. En primer lugar, se concluye que la presencia de la industria es aceptable para una conferencia que tiene una dedicación en parte a la investigación. Además, el hecho de que exista una relación cercana entre la industria y el mundo académico, es un elemento representativo de que en principio, la investigación académica es tenida en cuenta por parte de la industria.

Otra conclusión, que se puede tener como la más importante, es el tipo de empresa sobre la que ha tenido más repercusión la arquitectura dirigida por modelos. Las más interesadas en esta aproximación, son las grandes empresas multinacionales involucradas en procesos productivos. Estas empresas dedican un gran esfuerzo y tiempo a la investigación, por dos grandes motivos. Primero, estas empresas se pueden permitir invertir parte de su presupuesto en investigación debido a su envergadura. Segundo, la investigación que realizan repercute directamente en su propio beneficio, ya que sus propios procesos productivos se ven beneficiados por estas investigaciones.

Esta última puede ser una de las razones por las que se echa en falta a las grandes consultoras o empresas de servicios. Estas empresas dan soporte a una infinidad de clientes, que en muchos casos, tienen sus propias normativas y sus metodologías de trabajo. Así, cabe pensar que este hecho les impediría poder aplicar los resultados de una metodología específica de forma homogénea a sus distintos clientes, y por lo tanto, no sería rentable la inversión en MDA. Por último, cabe destacar que la heterogeneidad de las empresas participantes en MoDELS puede ser el motivo causante de la gran fragmentación en el tema de los artículos enviados a dicha conferencia.

7.6 Conclusiones finales

Partiendo de las conclusiones parciales expuestas en este capítulo, se puede concluir que el éxito de la aproximación MDA se ha circunscrito básicamente al ámbito académico, teniendo una repercusión en menor en entornos industriales. Bajo un punto de vista del impacto de las nuevas tecnologías, hoy por hoy las tecnologías aplicadas por la industria, evolucionan de forma vertiginosa. Las comunidades de desarrolladores trabajan de forma colaborativa, avanzando, depurando y mejorando las mismas en un tiempo relativamente corto. En el caso de MDA, se observa que la comunidad Eclipse enseguida entendió que podía ser una metodología interesante. Crearon sus propios estándares, consiguiendo simplificar los de OMG, pero viendo el tiempo transcurrido desde que iniciaron el *Eclipse Modeling Project* y estado de incubación de la mayoría de sus subproyectos, se concluye que no ha tenido el impacto esperado, por lo menos en la

comunidad *open source*, ya que sí se han aprovechado de la iniciativa Eclipse las grandes empresas, para desarrollar sus herramientas CASE. Tal vez, si los proyectos Eclipse estuvieran mejor integrados, el éxito habría sido algo más contundente.

De la misma manera, se puede llegar a pensar que si hubiera habido una alineación mayor entre OMG y Eclipse desde hace más tiempo, se podrían haber logrado más éxitos. Sólo cabe esperar si el alineamiento recientemente realizado entre las dos organizaciones, le dé el impulso necesario.

Por otro lado el impacto que la aproximación MDA ha tenido sobre la industria no ha sido especialmente relevante. Sólo las grandes empresas dedican un esfuerzo en investigar para aplicar esta metodología a sus propios procesos productivos, sobre todo en el ámbito de sistemas empotrados o sistemas hardware, pero las ideas más importantes de la aproximación MDA, el metamodelado y las transformaciones, no parece que despierten interés en la industria.

Otro detalle importante, sería el grado de aplicación de la aproximación MDA por parte de las grandes empresas consultoras. Estas podrían tener presupuesto suficiente para dedicarlo a la investigación, pero sin embargo no están presentes en la conferencia analizada. Como se ha comentado más arriba puede que sea la dependencia de sus clientes, en cuanto a metodologías de trabajo, que les obliga a adaptarse a múltiples formas de trabajo. No obstante, es cierto que podría ser interesante definir diferentes lenguajes específicos de dominios con los que resolver problemas comunes en las diferentes áreas de negocio existentes (administración pública, banca, seguros, telecomunicaciones, etc), pero esto parece no ocurrir.

7.7 Trabajo futuro

Evidentemente, se puede seguir ampliado este análisis, profundizando en determinados aspectos que se han pasado por alto o que no se han estudiado en detalle. Como trabajo futuro se pueden abordar las líneas de trabajo expuestas a continuación. En el estudio de los estándares MDA de OMG, se han dejado mencionados algunos de ellos y sería interesante completar el estudio de los mismos:

- *Metadata Interchange Patterns* (MIP) [OMG 2004a]
- *UML Human-Usable Textual Notation* (HUTN) [OMG 2004b]
- *OMG Systems Modeling Language* (SysML) [OMG 2008e]
- *Ontology Definition Metamodel* (ODM) [OMG 2008d]
- *Reusable Asset Specification* (RAS) [OMG 2005b]
- *Semantics of a Foundational Subset for Executable UML Models* (FUML) [OMG 2008f]
- *Software Process Engineering Metamodel* (SPEM) [OMG 2008c]

De igual forma existen proyectos Eclipse que se han dejado al margen de este estudio y que sería interesante analizar. Algunos de ellos serían:

- *EMF Technology* (EMFT) [Eclipse EMFT]
- *Graphical Editing Framework* (GEF) [Eclipse GEF]
- *Generative Modeling Technologies* (GMT) [Eclipse GMT]
- *Model Development Tools* (MDT) [Eclipse MDT]
- XML *Schemas Definition* (XSD) [Eclipse XSD]
- *Model to Model Transformation* (M2M) [Eclipse M2M] y dentro de éste proyecto los subproyectos:
  - *Atlas Transformation Language* (ATL) [Eclipse ATL]
  - *Procedural* QVT (*Operational*) [Eclipse QVTO]
  - *Declarative* QVT (*Core & Relational*) [Eclipse QVTR]
- *Model to Text Transformation* (M2T) [Eclipse M2T]
- *Textual Modeling Framework* (TMF) [Eclipse TMF]

En cuanto a las herramientas estudiadas, se puede optar por ampliar el número y el tipo de éstas. Como elemento de interés para ampliar el análisis, se deben estudiar herramientas *open source* que permitan conocer el nivel de cumplimiento con las características MDA. Así mismo, se podría ampliar el número de características estudiadas intentando igualar a las del estudio de *King's College* [King's 2003], reevaluando las herramientas estudiadas (en particular *Borland Together* con ejemplos prácticos) y evaluando las nuevas herramientas a estudiar.

Dentro del apartado del estudio de herramientas, parece interesante estudiar en concreto la herramienta OLIVANOVA [OLIVA][Molina 2004], por ser un herramienta coparticipada entre la Universidad Politécnica de Valencia y CARE *Technologies* S.A. Es un aspecto importante para el análisis de la repercusión de MDA, estudiar qué motivos han llevado a unificar esfuerzos del mundo académico e industrial, para desarrollar y comercializar una herramienta que de soporte a MDA y saber hasta qué punto soporta esta aproximación.

Se podría también hacer un estudio sobre otros trabajos relacionados con MDA en diferentes revistas especializadas o aportaciones en otras conferencias, que aparte de MoDELS, se centran en el desarrollo dirigido por modelos. De esta forma se analizaría la presencia de la industria en ellas, así como sus casos prácticos expuestos. Se podría repetir el estudio realizado en las conferencias MiSE, EC-MDA y JISDB.

Por último y como elemento que pueda resultar interesante, se puede abordar un estudio de *Business Process Modeling Notation* (BPMN), estándar de OMG que, aunque no se engloba dentro de la aproximación MDA, pueden verse relacionados. Llama la atención que la herramienta RSA soporta el modelado de procesos mediante esta notación y que Eclipse haya incluido la implementación de este estándar como parte del proyecto *Modeling Development Tools*.

APENDICE A

En este apéndice se relacionan las ponencias que se han tenido en cuenta en el análisis realizado en capítulo 6, en las diferentes conferencias MoDELS estudiadas.

MoDELS 2005

- "Computing Refactorings of Behavior Models" – A. Pretschner (Information Security, ETH) and W. Prenninger (BMW Group)
- "Invited Presentation I: Lessons Learned, New Directions, and Migration Plans for Model-Driven Development of Large Scale Software Based Systems" – M. J. Marich and H. F. Krikorian (The Boeing Company)
- "Invited Presentation II: Experiences in Applying Model Based System Testing Generation" - Marlon Vieira (Siemens Corporate Research)
- "Invited Presentation III: The Architects' Workbench — Research in the Trenches" - Doug Kimelman (IBM)
- Keynote address: "Domain-Specific Modeling: No One Size Fits All" – Juha-Pekka Tolvanen (MetaCase).
- "Lessons Learned from Automated Analysis of Industrial UML Class Models (An Experience Report)" – Betty H.C. Cheng , Ryan Stephenson (Michigan State University) and Brian Berenbach (Siemens Corporate Research)
- "Model-Based Scalability Estimation in Inception-Phase Software Architecture" - Steve Masticola (Siemens Corporate Research, Inc), Andre Bondi (Siemens Corporate Research, Inc), and Mark Hettish (Siemens Communications, Inc)
- "Model-Driven Engineering in a Large Industrial Context — Motorola Case Study" - Paul Baker Shiou Loh and Frank Weil (Motorola)
- "Scenario Construction Tool Based on Extended UML Metamodel" – M. Smialek (Infovide), J. Bojarski, W. Nowakowski, T. Straszak (Warsaw University of Tech.)
- "The Impact of UML 2.0 on Existing UML 1.4 Models" - Julie A. Street, Robert G. Pettit IV (The Aerospace Corporation)
- "Using a Domain-Specific Language and Custom Tools to Model a Multi-tier Service-Oriented Application — Experiences and Challenges" - Marek Vokàc (Simula Research Laboratory) and Jens M. Glattetre (SuperOffice ASA / ICT)
- "Using UML 2.0 Collaborations for Compositional Service Specification". R. Torbjorn (SINTEF ICT) and H. N. Castejón, F. A. Kreemer, R. Braek (NTNU)

MoDELS 2006

- "A Formal Semantics of UML-RT" - Michael von der Beeck (BMW Group)
- "A Graphical Approach to Risk Ideintification, Motivated by Empricial Investigations" – I. Hogganvik and Ketil Stolen (SINTEF and University of Oslo)
- "A Metamodeling Approach to Pattern Specification" - Maged Elaasar (Software Quality Engineering Laboratory and IBM Canada Ltd), Lionel C. Briand (Software Quality Engineering Laboratory and Simula Research Laboratory, Department of Software Engineering), and Yvan Labiche (Software Quality Engineering Lab.)
- Adopting Model Driven Software Development in Industry – A Case Study at Two Companies. M. Staron (University of Goteborg)
- "Analysis and Visualization of Behavioral Dependencies Among Distributed Objects Based on UML Models" - Vahid Garousi (Software Quality Engineering Laboratory), Lionel C. Briand

(Software Quality Engineering Laboratory and Simula Research Laboratory), and Yvan Labiche (Software Quality Engineering Laboratory)
- "Compositional MDA" - Louis van Gool (Technische Universiteit Eindhoven), Teade Punter (Technische Universiteit Eindhoven), Marc Hamilton (ASML), and Remco van Engelen (ASML)
- "Reusable MDA Components: A Testing-for-Trust Approach" - Jean-Marie Mottu (IRISA), Benoit Baudry (IRISA), and Yves Le Traon (France Télécom)
- "UML Model Interchange in Heterogeneous Tool Environments: An Analysis of Adoptions of XMI 2" - Björn Lundell, Brian Lings and Anna Persson (University of Skövde), and Anders Mattsson (Combitech)
- "Use Case Driven Iterative Development: Hurdles and Solutions" - Santiago Ceria (Hexacta SA) and Juan José Cukier (Pragma Consultores)
- "Using UML Activities for System-on-Chip Design and Synthesis" – T. Schattkowsky (C-Lab), J. H. Hausmann, and G. Engels (University of Paderborn)

MoDELS 2007

- "A Modelling Method for Rigorous and Automated Design of Large-Scale Industrial Systems" – K. Leppänen, S. Leppänen, and M. Turunen (Nokia Research Center)
- "A Practical Perspective on the Design and Implementation of Service-Oriented Solutions" - Alan W. Brown, Marc Delbaere, and Simon K. Johnston (IBM)
- "A UML Profile for Developing Airworthiness-Compliant (RTCA DO-178B), Safety-Critical Software" – Gregory Zoughbi (General Dynamics Canada and Calerton University) and Lionel Briand and Yvan Labiche (Carleton University)
- "Architectural Aspects in UML" – Jon Oldevik and Oysteien Haugen (University of Oslo and SINTEF)
- "Automated Semantic Analysis of Design Models" - Frank Weil, Brian Mastenbrook, David Nelson, Paul Dietz, and Aswin van den Berg (Motorola)
- "Domain Specific Modeling Methodology for Reconfigurable Networked Systems" - Gabor Batori, Zoltan Theisz (Network Management Research Centre, Ericsson Ireland Ltd), and Domonkos Asztalos (Software Engineering Group, Ericsson Hungary)
- "Enhancing UML Extensions with Operational Semantics. Behaviored Profiles with Templates"- A. Cuccuru, C. Mraidha, F. Terrier, and S. Grard (CEA LIST).
- "i^{2}MAP: An Incremental and Iterative Modeling and Analysis Process" - Sascha Konrad (Siemens Corporate Research, Inc), Heather J. Goldsby (Michigan State University), and Betty H.C. Cheng (Michigan State University)
- "Improving Inconsistency Resolution with Side-Effect Evaluation and Costs" - Jochen M. Küster and Ksenia Ryndina (IBM Zurich Research Laboratory)
- "Model-Driven Approach for Managing Human Interface Design Life Cycle" - Noi Sukaviriya, Vibha Sinha, Thejaswini Ramachandra, and Senthil Mani (IBM)
- "Model-Driven Engineering for Software Migration in a Large Industrial Context" – F. Fleurey (IRISA/INRIA and Sodifrance), E. Breton (Sodifrance), B. Baudry (IRISA/INRIA), A. Nicolas (Sodifrance), and J.-M.Jézéquel (IRISA/INRIA),
- "Relating Navigation and Request Routing Models in Web Applications" - Minmin Han (amazon.com) and Christine Hofmeister (Lehigh University)

MoDELS 2008

- "3D Parametric Models for Aeroplanes - From Idea to Design" – P. Rauhut (Airbus)
- "A Model-Based Framework for Security Policy Specification, Deployment and Testing" - Tejeddine Mouelhi (IT- Telecom), Franck Fleurey (SINTEF), Benoit Baudry (INRIA/IRISA) and Yves Le Traon (IT- Telecom)

- "An Aspect-Oriented and Model-Driven Approach for Managing Dynamic Variability"- B. Moring, F. Fleurey, N. Bencomo, J.M. Jezequel, A. Solberg, V. Dehlen and G. Blair (ISIA/INRIA, SINTEF and Lancaster University)
- "Automatability of Coupled Evolution of Metamodels and Models in Practice" – M. Herrmannsdoerfer (Institut für Informatik Technische Universität München), S. Benz (BMW) and E. Juergens (Institut für Informatik Technische Universität München)
- "General Mode Controller for Software on Artificial Satellite with Model-Based Validation Tool" – T. Obata and T. Inoue (Mitsubihi Electric Corporation)
- "Meaningful Composite Structures On the Semantics of Ports in UML2" - Arnaud Cuccuru, Sébastien Gérard, and Ansgar Radermacher (CEA LIST)
- "Model-Based Quality Assurance of Automotive Software"- J. Jürjens, D. Reib, D. Trachtenherz.
- "NAOMI – An Experimental Platform for Multi–modeling" - Trip Denton, Edward Jones, Srini Srinivasan, Ken Owens, and Richard W. Buskens (Lockheed Martin Advanced Technology Laboratories)
- "Ontology Guided Evolution of Complex Embedded Systems Projects in the Direction of MDA" - Lars Pareto (IT University of Göteborg), Miroslaw Staron (IT University of Göteborg), and Peter Eriksson (Ericsson Software Research)
- "Requirements Modeling and Validation Using Bi-layer Use Case Descriptions" - Avik Sinha, Matthew Kaplan, Amit Paradkar, and Clay Williams (IBM)
- "Specifying Service Composition Using UML 2.x and Composition Policies" - Judith E.Y. Rossebø (Department of Telematics, NTNU and Telenor R&I) and Ragnhild Kobro Runde (Department of Informatics, University of Oslo)
- "Sufficient Criteria for Consistent Behavior Modeling with Refined Activity Diagrams" - Stefan Jurack (Philipps-Universität Marburg), Leen Lambers (Technische Universität Berlin), Katharina Mehner (Siemens, Corporate Technology), and Gabriele Taentzer (Philipps-Universität Marburg)
- "The Future of Train Signaling" - Andreas Svendsen (SINTEF), Gøran K. Olsen (SINTEF), Jan Endresen (ABB), Thomas Moen (ABB), Erik Carlson (ABB), Kjell-Joar Alme (ABB), and Øystein Haugen (SINTEF)

REFERENCIAS

[Akehurst 2008] David H. Akehurst, Steffen Zschaler y Gareth Howells. *OCL: Modularising the Language. Electronic Communication of the European Association of Software Science and Technology*, volumen 9, 2008.

[Alonso 2009] Luis María Alonso Martín. *Eclipse Modeling Framework (EMF): Una visión general*. Trabajo asignatura Requisitos del Software. Máster en investigación en informática. Universidad Complutense de Madrid. Febrero de 2009.

[Beca 2009] Santiago Beca Moreno. *Análisis del Graphical Modeling Framework del Proyecto Eclipse*. Trabajo asignatura Requisitos del Software. Máster en investigación en informática. Universidad Complutense de Madrid. Febrero de 2009.

[Bollati 2007] Verónica Bollati, Juan M. Vara, Belén Vela y Esperanza Marcos. *Una revisión de herramientas MDA*. DSMD, Actas del IV taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones, 2007.

[Booch 2005] Grady Booch, James Rumbaugh, Ivar Jacobson. *The unified modeling language user guide 2nd Edition*. Addison-Wesley, 2005.

[Booch 2007] Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young Ph.D., Jim Conallen y Kelli A. Houston. *Object-oriented analysis and design with applications, 3rd Edition*. Addison-Wesley, 2007.

[Borland 2006] Borland. Quick Start Guide to MDA - A Primer to Model-Driven Architecture Using Borland Together Technologies. Borland Software Corporation, 2006.

[Borland 2008] Borland. *Borland Together 2008 - Borland Together Modeling Guide*. Borland Software Corporation, 2008.

[Borland 2009] Borland. *Borland Together 2008 Service Pack 1 Release Notes*. Borland Software Corporation, 2009.

[Borland CRM] Borland. Borland Caliber Requirements Management. http://www.borland.com/us/products/caliber/index.html (consulta: agosto de 2009).

[Briand 2005] L. Briand y C. Williams (eds.) *MoDELS 2005, LNCS 3713*. Springer-Verlag Berlin Heidelberg 2005.

[Bruck 2008] James Bruck, Christian Damus. *Creating Robust Scalable DSLs with UML* . Eclipse Conference, 2008.

[Budinsky 2003] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick y Timothy J. Grose. *Eclipse Modeling Framework: A Developer's Guide.* Addison-Wesley, 2003

[Cabot 2006] Jordi Cabot y Ernest Teniente. *Constraint Support in MDA Tools: A Survey*. Lecture notes in Computer Science nº 4066, 2006

[Cabot 2008] Jordi Cabot, Cristina Gómez, Elena Planas y M. Elena Rodríguez. *Reverse Engineering of OO constructs in Object-Relational*. JISBD 2008, 13th Conference on Software Engineering and Databases, 2008.

[Cabot 2009] Jordi Cabot, Robert Clarisó y Daniel Riera. *Verifying UML/OCL Operation Contracts*. IFM 2009, *7th International conference of Integrated Formal Methods*, 2009.

[CEPIS 2008] Council of European Professional Informatics Societies. *Monograph: Model-Driven Software Development*. UPGRADE. *The European Journal for Informatics Professional*. Abril 2008.

[CORBA] OMG. *Common Object Request Broker architecture (CORBA/IIOP) Version 3.1*, 2008. http://www.omg.org/spec/CORBA/3.1/

[Czarnecki 2008] K. Czarnecki et al. (eds.) *MoDELS 2008, LNCS 5301*. Springer-Verlag Berlin Heidelberg 2008.

[Eclipse ATL] Eclipse. *ATLAS Transformation Language*. http://www.eclipse.org/m2m/atl/ (consulta agosto de 2009)

[Eclipse BPMN2] Eclipse. *MDT/BPMN2*. http://wiki.eclipse.org/MDT-BPMN2 (consulta: agosto de 2009)

[Eclipse Ciclo] Ciclo de vida de los proyetos Eclipse. http://www.eclipse.org/projects/dev_process/development_process.php (consulta: mayo de 2009)

[Eclipse EMF] Eclipse. *EMF: Eclipse Modeling Framework*. http://www.eclipse.org/modeling/emf/ (consulta: julio de 2009)

[Eclipse EMFT] Eclipse. *EMFT: EMF Technology*. http://www.eclipse.org/modeling/emft/ (consulta: mayo de 2009)

[Eclipse EMP] Eclipse. *EMP: Eclipse Modeling Project*. http://www.eclipse.org/modeling/ (consulta: mayo de 2009)

[Eclipse GEF] Eclipse. *GEF: Graphical Editing Framework*. http://www.eclipse.org/gef/ (consulta: mayo de 2009)

[Eclipse GMFa] Eclipse. *GMF: Graphical Modeling Framework*. http://www.eclipse.org/modeling/gmf/ (consulta: julio de 2009)

[Eclipse GMFb] Eclipse. *GMF Tutorial*. http://wiki.eclipse.org/index.php/GMF_Tutorial (consulta: julio 2009)

[Eclipse GMT] Eclipse. *GMT: Generative Modeling Technologies.* http://www.eclipse.org/gmt/ (consulta: julio de 2009)

[Eclipse IMM] Eclipse. *MDT/IMM*. http://wiki.eclipse.org/MDT-IMM (consulta: agosto 2009)

[Eclipse link] Eclipse. Eclipse Persistence Services Project. http://www.eclipse.org/eclipselink/ (consulta: agosto de 2009)

[Eclipse M2M] Eclipse. *M2M: Model to Model Transformation.* http://www.eclipse.org/m2m/ (consulta: julio de 2009)

[Eclipse M2T] Eclipse. *M2T: Model to Text Transformation.* http://www.eclipse.org/modeling/m2t/ (consulta: julio de 2009)

[Eclipse MAP] Eclipse. *MAP: Modeling Amalgam Project.* http://www.eclipse.org/modeling/amalgam/ (consulta: julio de 2009)

[Eclipse MDT] Eclipse. *MDT: Model Development Tools.* http://www.eclipse.org/modeling/mdt/ (consulta: julio de 2009)

[Eclipse OCL] Eclipse. *MDT/OCL*. http://wiki.eclipse.org/MDT/OCL (consulta: agosto de 2009)

[Eclipse PRO] Eclipse. *Estados de los Proyectos Eclipse.* http://www.eclipse.org/projects/listofprojects.php (consulta: mayo de 2009)

[Eclipse QVTO] Eclipse. *Operational QVT Documentation*. http://www.eclipse.org/m2m/qvto/doc/ (consulta: agosto 2009)

[Eclipse QVTR] Eclipse. *Relational QVT Language*. http://wiki.eclipse.org/M2M/Relational_QVT_Language_(QVTR) (consulta: agosto 2009)

[Eclipse SDK] Eclipse. *Eclipse Documentation*. http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.emf.doc/references/overview/EMF.html (consulta: julio de 2009)

[Eclipse Tech] Eclipse. *Eclipse Tehcnology Project*. http://www.eclipse.org/technology (consulta: julio de 2009)

[Eclipse TMF] Eclipse. *TMF: Textual Modeling Framework.* http://www.eclipse.org/modeling/tmf/ (consulta: mayo de 2009)

[Eclipse Tools] Eclipse. *Eclipse Tools Project*. http://www.eclipse.org/tools (consulta: julio de 2009)

[Eclipse UML2] Eclipse. *MDT/UML2*. http://wiki.eclipse.org/MDT-UML2 (consulta: agosto de 2009)

[Eclipse UML2P] Eclipse. MDT-UML2Tools How To Use UML Profiles. http://wiki.eclipse.org/MDT-UML2Tools_How_To_Use_UML_Profiles (consulta: agosto 2009)

[Eclipse UML2T] Eclipse. *MDT/UML2 Tools*. http://wiki.eclipse.org/MDT-UML2Tools (consulta: agosto 2009)

[Eclipse Xpand] Eclipse. *M2T/Xpand*. http://wiki.eclipse.org/Xpand (consulta: agosto 2009)

[Eclipse XSD] Eclipse. *MDT/XSD*. http://www.eclipse.org/modeling/mdt/?project=xsd#xsd (consulta: agosto 2009)

[ECMDA] *European Conference on Model Driven Architecture Foundations and Applications.* http://www.ecmda-fa.org (consulta: julio 2009)

[Engels 2007] G. Engels et al. (eds.) *MoDELS 2007, LNCS 4735*. Springer-Verlag Berlin Heidelberg 2007.

[F. de Alba 2009] José María Fernández de Alba López de Pablo. *Eclipse UML2 y UM2 Tools*. Trabajo asignatura Requisitos del Software. Máster en investigación en informática. Universidad Complutense de Madrid. Febrero de 2009.

[Fernández 2009] Pedro Fernández Sáez. *Análisis Sobre la Repercusión de la Aproximación MDA en la Industria*. Trabajo asignatura Requisitos del Software. Máster en investigación en informática. Universidad Complutense de Madrid. Febrero de 2009.

[Forward 2008] Andrew Forward, Timothy C. Lethbridge. *Problems an Oportunities for Model-Centric Versus Code-Centric Software Development: A Survey of Software Professional*. *Models in Software Engineering*, 2008.

[France 2001] Robert France, James M. Bieman. *Multi-view Software Evolution: A UML based Framework for Evolving*. *International Conference on Software Maintenance*, 2001.

[Fuentes 2004] L. Fuentes, A. Vallecillo. *An Introduction to UML Profiles*. UPGRADE, *The European Journal for the Informatics Professional*, 5(2), 2004. http://www.lcc.uma.es/~av/Publicaciones/04/UMLProfiles-Upgrade04.pdf

[Gardner 2006] T. Gardner, L. Yusuf. *Explore model-driven development (MDD) and related approaches: A closer look at model-driven development and other industry initiatives*, 2006. http://www.ibm.com/developerworks/library/ar-mdd3/

[Génova 2006] Gonzalo Génova Fuster, José Miguel Fuentes Torres y María Cruz Valiente Blázquez. *Evaluación comparativa de herramientas CASE para UML desde el punto de vista notacional*. Novática, Revista de la Asociación de Técnicos de Informática, mayo-junio de 2006.

[González 2008] César González-Pérez, Brian Henderson-Sellers. *Metamodelling for Software Engineering*. Wiley, 2008.

[Gronback 2006] Richard C. Gronback. *Eclipse Modeling Project and OMG Standards*. Eclipse Modeling Symposium, 2006.

[Gronback 2008] Richard C. Gronback. *Model-Driven Architecture and Eclipse*. UPGRADE, *The European Journal for the Informatics Professional*, abril de 2008.

[Gronback 2009] Richard C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Professional, 2009.

[Gutiérrez 2008] Javier J. Gutiérrez, Clémentine Nebut, María J. Escalona, Manuel Mejías y Isabel M. Ramos. *Visualization of Use Cases through Automatically Generated Activity Diagrams*. MoDELS *International Conference on Model Driven Engineering Languages and Systems*, 2008.

[Hunter 2007] Anthony Hunter y Mohammed Mostafa. *Extending your DSM by leveraging the GMF Runtime*. Eclipse Conference, 2007.

[Hussey 2005] Kenn Hussey. *UML2 Project Sprint*. Eclipse Conference, 2005.

[IBM RDP] IBM. Rational Design Patterns. http://publib.boulder.ibm.com/infocenter/rsdvhelp/v6r0m1/topic/com.ibm.xtools.pttrn.author.doc/topics/c_design_ptrns.html (consulta: agosto 2009).

[IBM RRP] IBM. Rational RequisitePro. http://www-01.ibm.com/software/awdtools/reqpro/ (consulta: agosto 2009).

[IBM RSA] IBM. IBM Websphere Help System – Rational Sotware Architect Information Center. http://publib.boulder.ibm.com/infocenter/rtnlhelp/v6r0m0/index.jsp?topic=/com.ibm.rational.rsa.books/icwelcome_product_rsa.htm (consulta: agosto 2009).

[IBM RUP] IBM. Rational Unified Process. http://www-01.ibm.com/software/awdtools/rup/ (consulta: agosto 2009)

[ICSE] International Conference on Software Engeenering. http://www.icse-conferences.org (consulta: julio 2009).

[ISO 1986] ISO 8879: 1986. *Standard Generalized Markup*. International Organization Standardization, 1986.

[Jacobson 1996] Ivar Jacobson, Michael Blaha, William Premerlani, Frederick Eddy y William Lorensen. *Object-oriented software engineering: a use case driven approach*. Addison-Wesley, 1996

[Jacobson 2001] Ivar Jacobson, Grady Booch, James Rumbaugh. *El proceso unificado de desarrollo de software.* Traducción, Salvador Sánchez... [et al.] ; coordinación de la traducción y revisión técnica, Luis Joyanes, Ernesto Pimentel. Addison Wesley, 2001.

[JISBD] Jornadas de Ingeniería del Software y Bases de Datos. http://www.informatik.uni-trier.de/~ley/db/conf/jisbd/index.html (consulta: junio 2009)

[King's 2003] King's College London, *An Evaluation of Compuware OptimalJ Professional Edition as an MDA Tools*. 2003

[Kleppe 2005] Anneke Kleppe, Jos Warmer y Win Bast. *MDA Explained – The Model Driven Architecture: Practice And Promise*. Addison-Wesley, 2005

[Koch 2006] Nora Koch, Gefei Zhang y María José Escalona. *Model Transformations from Requirements to Web System Design*. ICWE: International Conference on Web Engineering, 2006.

[Kolovos 2009] Dimitrios S. Kolovos, Louis M. Rose, Richard F. Paige, Fiona A.C. Polack. *Raising the level of abstraction in the development of GMF-based graphical model editors*. ICSE: International Conference on Software Engineering, 2009.

[López 2007] Edna D. López, Moisés González, Máximo López y Erik L. Iduñate. *Proceso de desarrollo de software mediante herramientas MDA*. CISCI 6ª Conferencia Iberoamericana en sistemas, cibernética e informática, 2007

[Lundell 2006] Björn Lundell, Brian Lings, Anna Persson, y Anders Mattsson. *UML Model Interchange in Heterogeneous Tool*. MoDELS: International Conference on Model Driven Engineering Languages and Systems, 2006.

[Magariño 2009] Iván García-Magariño, Rubén Fuentes-Fernández, Jorge J. Gómez-Sanz. *Guideline for the definition of EMF metamodels using an Entity-Relationship approach.* Elsevier, 2009.

[Mellor 2004] Stephen J. Mellor, Kendall Scott, Axel Uhl y Dirk Weise. *MDA Distilled – Principles of Model-Driven Architecture*. Addison-Wesley, 2004. www.MDADistilled.com

[MoDELS] ACM/IEEE *International Conference on Model Driven Engineering Languages and Systems*. http://www.informatik.uni-trier.de/~ley/db/conf/models/index.html (consulta: febrero 2009).

[Molina 2004] Juan Carlos Molina, Oscar Pastor. *MDA, OO-Method y la Tecnología OLIVANOVA Model Execution*. I Taller sobre desarrollos dirigidos por modelos, MDA y aplicaciones. Málaga, 2004.

[Nierstrasz 2006] O. Nierstrasz et al. (eds.) *MoDELS 2006*, *LNCS 4199*. Springer-Verlag Berlin Heidelberg 2006.

[OASIS BPEL] OASIS. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[OLIVA] CARE Technologies. *OLIVANOVA the Programming Machine*. http://www.care-t.com/products/index.asp (consulta: agosto de 2009)

[OMG] Object Management Group. http://www.omg.org/ (consulta: abril 2009).

[OMG 2002] OMG. *OMG IDL Syntax and Semantics*, 2002. http://www.omg.org/cgi-bin/doc?formal/02-06-39.pdf

[OMG 2003a] OMG. *MDA Guide Version 1.0.1*, 2003. http://www.omg.org/docs/omg/03-06-01.pdf

[OMG 2003b] OMG. *Common Warehouse Metamodel, version 1.1*, 2003. http://www.omg.org/docs/formal/03-03-02.pdf

[OMG 2004a] OMG. *Common Warehouse Metamodel (CWM) Metadata Interchange Patterns (MIP) Specification, v1.0,* 2004. http://www.omg.org/docs/formal/04-03-25.pdf

[OMG 2004b] OMG. *Human-Usable Textual Notation (HUTN) Specification, v1.0,* 2004 http://www.omg.org/docs/formal/04-08-01.pdf

[OMG 2005a] OMG. *Unified Modeling Languaje Specification version 1.4.2*, 2005. http://www.omg.org/docs/formal/05-04-01.pdf

[OMG 2005b] OMG. *Reusable Asset Specification (RAS), version 2.2*, 2005. http://www.omg.org/docs/formal/05-11-02.pdf

[OMG 2006a] OMG. *Meta Object Facility (MOF) Core Specification*, 2006. http://www.omg.org/docs/formal/06-01-01.pdf

[OMG 2006b] OMG. *Object Constraint Language, Version* 2.0, 2006. http://www.omg.org/docs/formal/06-05-01.pdf

[OMG 2007a] OMG. *OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2*, 2007. http://www.omg.org/docs/formal/07-11-04.pdf

[OMG 2007b] OMG. *OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2*, 2007. http://www.omg.org/docs/formal/07-11-02.pdf

[OMG 2007c] OMG. *MOF 2.0/XMI Mapping, Version 2.1.1*, 2007. http://www.omg.org/docs/formal/07-12-01.pdf

[OMG 2008a] OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, 2008. http://www.omg.org/docs/formal/08-04-03.pdf

[OMG 2008b] OMG. *MOF Model to Text Transformation Language, v1.0*, 2008. http://www.omg.org/docs/formal/08-01-16.pdf

[OMG 2008c] OMG. Software *& Systems Process Engineering Metamodel specification (SPEM) v2.0*, 2008. http://www.omg.org/docs/formal/08-04-01.pdf

[OMG 2008d] OMG. *Documents associated with Ontology Definition Metamodel(ODM) Version 1.0-Beta 3,* 2008. http://www.omg.org/docs/ptc/08-09-07.pdf

[OMG 2008e] OMG. *OMG Systems Modeling Language (OMG SysML)*. 2008. http://www.omg.org/docs/formal/08-11-01.pdf

[OMG 2008f] OMG. *Semantics of a Foundational Subset for Executable UML Models (FUML) v1.0-Beta 1*, 2008. http://www.omg.org/docs/ptc/08-11-03.pdf

[OMG BPMN] OMG. *Business Project Management Notation*. http://www.bpmn.org/ (consulta: agosto 2009).

[OMG UML] OMG. *Unified Modeling Language Version 2.2*. http://www.omg.org/spec/UML/2.2/ (consulta: abril 2009)

[Plante 2006] Frederic Plante. *Introducing the GMF Runtime*. IBM Corporation, 2006. http://www.eclipse.org/articles/Article-Introducing-GMF/article.html

[Poole 2003] John Poole, Dan Chang, Douglas Tolbert, David Mellor. *Common Warehouse Metamodel Developer's Guide*. New York, John Wiley & Sons Inc, enero 2003.

[RED Hibernate] Red Hat. *Hibernate*. https://www.hibernate.org/ (consulta: abril 2009)

[Rodríguez 2004] Jesús Rodríguez Vicente. *Ingeniería de Modelos con MDA: Estudio comparativo de OptimalJ y ArcStyler*. Proyecto informático, Departamento de Informática y Sistemas, Universidad de Murcia, junio de 2004

[Rumbaugh 1996] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy y William Lorensen. *Modelado y Diseño Orientado a Objetos. Metodología OMT.* Traducción: José Rafael García-Bermejo Giner; revisión técnica: Luis Joyanes Aguilar. Prentice Hall, 1996

[Rumbaugh 2007] James Rumbaugh, Ivar Jacobson, Grady Booch. *El lenguaje unificado de modelado : manual de referencia* (2ª Edición). Traducción, Héctor Castan Rodriguez, Oscar Sanjuan Martínez, Mariano de la Fuente Alarcón ; coordinacion general y revisión técnica, Luis Goyanes Aguilar. PEARSON – Addison Wesley, 2007

[Schach 2005] Stephen R. Schach. *Análisis y diseño orientado a objetos con UML y el proceso unificado*. Traducción, Lorena Peralta Rosales ; revisión técnica, Humberto Cárdenas Anaya, Martha Rosa Cordero López, Marco Antonio Dorantes González. Mc Graw Hill, 2005

[Soler 2007] E. Soler, J. Trujillo, E. Fernández-Medina y M. Piattini. *An extension of the Relational Metamodel of CWM to represent Secure Data Warehouses at the Logical Level*. JISBD 2007, 12th Conference on Software Engineering and Databases, 2007

[Sparx EAGuide] Sparx Systems. *Enterprise Architect User Guide version 7.5*, 2009.

[Sparx 2005b] Sparx Systems. *Implementing Model Driven Architecture using Enterprise Architect - MDA in Practice*, 2005.

[Sparx 2005b] Sparx Systems. *Implementing Model Driven Architecture using Enterprise Architect - Mapping MDA Concepts to EA Features*, 2005.

[Steinberg 2008] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks. *EMF: Eclipse Modeling Framework, 2nd Edition*. Addison-Wesley Professional, 2008

[SUN Annotations] SUN Microsystems. *Java Programming Languages Annotations*. http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html

[SUN JDBC] SUN Microsystems. *Java DataBase Connectivity*. http://java.sun.com/javase/technologies/database/index.jsp (consulta: abril de 2009)

[SUN JSP] SUN Microsystems. *Java Server Pages*. http://java.sun.com/products/jsp/ (consulta: abril de 2009)

[SUN PET] Sun Microsystems. *Java Pet Store Demo*, https://blueprints.dev.java.net/petstore/ (consulta: julio de 2009)

[Tikhomirov 2008] Artem Tikhomirov, Alexander Shatalin. *Introduction to Graphical Modeling Framework*. Eclipse Conference, 2008.

[UML Conf] UML Conference. http://www.informatik.uni-trier.de/~ley/db/conf/uml/index.html (consulta: febrero de 2009)

[W3C 1997] W3C. *Comparison of SGML and XML*. World Wide Web Consortium Note 15 de diciembre de1997. http://www.w3.org/TR/NOTE-sgml-xml-971215 (consulta: julio de 2009)

[W3C 2006] W3C. *Extensible Markup Language* (XML) 1.1 (Second Edition). W3C Recommendation 16 de agosto de 2006. http://www.w3.org/TR/2006/REC-xml11-20060816/ (consulta: julio de 2009)

[Watson 2008] Andrew Watson. *A brief History of MDA*. UPGRADE, The European Journal for the Informatics Professional, abril de 2008.

[WSDL] W3C. *Web Services Definition Language (WSDL)*. http://www.w3.org/TR/2001/NOTE-wsdl-20010315 (consulta: julio 2009)

[XSD] W3C. *XML Schema Specification*. http://www.w3.org/XML/Schema (consulta: julio 2009)

[Zubcoff 2008] Jose Zubcoff, Jesús Pardillo, y Juan Trujillo. *Análisis de series temporales dirigido por modelos conceptuales*. JISBD: 13th Conference on Software Engineering and Databases, 2008.