
Departamento de Tecnologías y Sistemas de Información

Definición de Lenguajes de Modelos
MDA vs DSL

Beatriz Mora, Francisco Ruiz, Félix García, Mario Piattini

Grupo Alarcos. Universidad de Castilla-La Mancha, España.

Informe Técnico
Marzo 2006



DEPÓSITO LEGAL:

ÍNDICE

ÍNDICE

1. INTRODUCCIÓN A MDA (MODEL DRIVEN ARCHITECTURE)	2
1.1. Visión General	2
1.2. Conceptos de MDA	2
1.3. Ciclo de vida de desarrollo de MDA.....	5
1.4. Transformación de modelos	7
1.5. Estándares relacionados	8
1.5.1. UML (Unified Modeling Language).....	8
1.5.2. MOF (Meta Object Facility)	8
1.5.3. XMI (XML Metadata Interchange)	9
1.5.4. QVT (Query, Views and Transformation).....	10
1.5.5. JMI (Java Metadata Interface).....	10
1.5.6. CWM (Common Warehouse Metamodel)	11
1.5.7. ODM (Ontology Definition Metamodel)	11
 2. DEFINICIÓN DE LENGUAJES DE MODELOS DENTRO DE MDA.....	 14
2.1. Introducción.....	14
2.2. Metamodelos UML 2.0 y MOF.....	15
2.2.1. UML 2.0.....	15
2.2.2. MOF	17
2.3. Caso de estudio MOF - MDA	19
2.4. Herramientas	21
2.4.1. OptimalJ.....	22
2.4.2. ArcStyler	25
2.4.3. AndroMDA.....	27
2.4.4. IBM XDE.....	27
2.4.5. Eclipse GMT.....	28
 3. DEFINICIÓN DE LENGUAJES DE MODELOS USANDO DSLS	 30
3.1. Introducción.....	30
3.2. Metodología para el diseño de DSL.....	31
3.3. Caso de estudio	32
3.4. Herramientas	33
3.4.1. Microsoft DSL Tools para Visual Studio 2005.....	33
 4. COMPARATIVA	 38
4.1. Ventajas y desventajas de la alternativa MDA.....	38
4.2. Ventajas y desventajas de la alternativa DSL	39
 5. INTEGRACIÓN – DEFINIENDO DSLS MEDIANTE UML Y/O MOF.....	 42

5.1. Introducción..... 42

5.2. Definición de un DSL..... 42

REFERENCIAS 45

ACRÓNIMOS 48

ÍNDICE
DE FIGURAS

ÍNDICE DE FIGURAS

FIGURA 1. TRANSFORMACIÓN DE MODELOS	5
FIGURA 2. PSM (MODELO DEPENDIENTE DE LA PLATAFORMA)	6
FIGURA 3. PSM Y CÓDIGO AUTOMÁTICO	6
FIGURA 4. INTEROPERABILIDAD MEDIANTE PUENTES	7
FIGURA 5. EJEMPLO DE <i>ESTEREOTIPO</i> , <i>RESTRICCIÓN</i> Y <i>VALOR ETIQUETADO</i> EN UML	16
FIGURA 6. RELACIÓN ENTRE MOF Y XMI	18
FIGURA 7: INSTANCIAS DEL SISTEMA (CAPA M0)	19
FIGURA 8: ENTIDADES DEL MODELO DEL SISTEMA (CAPA M1)	19
FIGURA 9: ENTIDADES DEL METAMODELO DE UML (CAPA M2)	20
FIGURA 10: ENTIDADES DE MOF (CAPA M3)	20
FIGURA 11: OPTIMALJ 4.0 DE COMPUWARE	22
FIGURA 12. TIPOS DE MODELOS Y PATRONES EN OPTIMALJ, EXTRAÍDO DE [20]	23
FIGURA 13: ARCSTYLER	26
FIGURA 14: DIAGRAMA UML DE UNA ARQUITECTURA DE UN SISTEMA BASADO EN DSL	31
FIGURA 15: COMPONENTES DE UN DSL	32
FIGURA 16: MICROSOFT DOMAIN SPECIFIC LANGUAGE (DSL) TOOLS PARA VISUAL STUDIO 2005	34
FIGURA 17: ESQUEMA DSL TOOLS FOR VISUAL STUDIO	34
FIGURA 18: PARTES DE UN DSL. SACADO DE [26]	35
FIGURA 19: DEFINICIÓN DEL MODELO DE DOMINIO	35
FIGURA 20: DEFINICIÓN DE ELEMENTOS VISUALES	36
FIGURA 21: ELEMENTOS DE GENERACIÓN DE TEXTO	36
FIGURA 22: METAMODELO PARA “ <i>FEATURE MODELING</i> ”	42
FIGURA 23: NOTACIÓN UML	43
FIGURA 24: NOTACIÓN DEL MODELO ESPECÍFICO	43
FIGURA 25: EXTENSIÓN DE UN METAMODELO UML	43
FIGURA 26: INSTANCIA DEL COMPONENTE	43

ÍNDICE
DE TABLAS

ÍNDICE DE TABLAS

TABLA 1. CAPAS DE MODELADO DEL OMG..... 15

1. INTRODUCCIÓN A MDA (MODEL DRIVEN ARCHITECTURE)

1. INTRODUCCIÓN A MDA (Model Driven Architecture)

En este apartado se pretende dar una descripción general del MDA y describir algunos de los estándares relacionados.

1.1. Visión General

La arquitectura dirigida por modelos (Model Driven Architecture) es una especificación detallada por el OMG (Object Management Group) que integra diferentes especificaciones y estándares definidos por dicha organización con la finalidad de ofrecer una solución a los problemas relacionados con los cambios en los modelos de negocio, la tecnología y la adaptación de los sistemas de información a los mismos.

El OMG es una organización de compañías de sistemas de información creada en 1990 con el fin de potenciar el desarrollo de aplicaciones orientadas a objetos distribuidas. Esta organización ha definido estándares importantes como UML, CORBA, MOF, entre otros.

En los últimos años se ha creado el interés e importancia al modelado en el desarrollo de cualquier tipo de software, debido a la facilidad que ofrece un buen diseño tanto a la hora de desarrollar como al hacer la integración y mantenimiento de sistemas de software.

Como consecuencia en el año 2001, el OMG definió un marco de trabajo nuevo llamado MDA. La clave del MDA es la importancia de los modelos en el proceso de desarrollo de software. MDA propone la definición y uso de modelos a diferente nivel de abstracción, así como la posibilidad de la generación automática de código a partir de los modelos definidos y de las reglas de transformación entre dichos modelos.

MDA pretende separar, por un lado, la especificación de las operaciones y datos de un sistema, y por el otro, los detalles de la plataforma en la que se construirá el sistema. Para ello, MDA proporciona las bases para:

- Definir un sistema independientemente de la plataforma sobre la que se construye.
- Definir plataformas sobre las que construir los sistemas.
- Elegir una plataforma particular para el sistema
- Transformar la especificación inicial del sistema a la plataforma elegida.

Este enfoque, además, posibilita el desarrollo de herramientas que le den soporte.

Los objetivos principales del MDA son mejorar la **productividad**, la **portabilidad**, la **interoperabilidad** y la **reutilización** de los sistemas.

1.2. Conceptos de MDA

De cara a entender MDA y sus características, su funcionamiento y su aplicación al proceso de desarrollo, se revisarán los conceptos básicos de MDA sacados de [3].

Sistema

Los conceptos de MDA se definen centrados en la existencia o planteamiento de un sistema, que puede variar entre un simple sistema informático, o combinaciones de componentes en diferentes sistemas informáticos, o diferentes sistemas en diferentes organizaciones, etc.

Modelo

Un modelo de un sistema es una descripción o una especificación de ese sistema y su entorno para desempeñar un determinado objetivo. Los modelos se presentan normalmente como una combinación de texto y dibujos. El texto se puede presentar en lenguaje de modelado, o en lenguaje natural.

Dirigido por modelos

MDA es un acercamiento al desarrollo de sistemas, que potencia el uso de modelos en el desarrollo. Se dice que MDA es dirigido por modelos porque usa los modelos para dirigir el ámbito del desarrollo, el diseño, la construcción, el despliegue, la operación, el mantenimiento y la modificación de los sistemas.

Arquitectura

La arquitectura de un sistema es la especificación de las partes del mismo, las conexiones entre ellos, y las normas de interacción entre las partes del sistema haciendo uso de las conexiones especificadas.

MDA determina los tipos de modelos que deben ser usados, como preparar dichos modelos y las relaciones que existen entre los diferentes modelos.

Perspectiva

Una perspectiva es una abstracción que hace uso de un conjunto de conceptos de arquitectura y reglas estructurales para centrarse en aspectos particulares del sistema, obteniendo un modelo simplificado.

Vista

Una vista es una representación del sistema desde un determinado punto de vista.

Plataforma

Una plataforma es un conjunto de subsistemas y tecnologías que aportan un conjunto coherente de funcionalidades a través de interfaces y determinados patrones de uso, que cualquier aplicación que se construya para esa plataforma puede usar sin preocuparse por los detalles de la implementación o como se lleva a cabo la misma dentro de la plataforma.

Aplicación

En MDA se define el término aplicación como una funcionalidad que tiene que ser desarrollada. Por tanto se puede definir un sistema en términos de la implementación de una o más aplicaciones, soportadas por una o más plataformas.

Independencia de la plataforma

La independencia de la plataforma es una cualidad que tienen que presentar los modelos. Lo que significa que un modelo es independiente de las facilidades o características que implementan las plataformas, de cualquier tipo.

Perspectivas de MDA

MDA establece tres perspectivas que se emplearán a lo largo del proceso de ingeniería:

- Punto de vista independiente de la computación: se centra en el entorno del sistema y los requisitos para el mismo. Los detalles de la estructura y procesamiento del sistema no se muestran, o aún no están especificados.
- Punto de vista independiente de la plataforma: se centra en las operaciones del sistema, mientras oculta los detalles necesarios para una determinada plataforma. Muestra aquellas partes de la especificación del sistema que no cambian de una plataforma a otra. En este punto de vista debe emplearse lenguaje de modelado de propósito general, o bien algún lenguaje específico del área en que se empleará el sistema, pero en ningún caso se emplearán lenguajes específicos de plataformas.
- Punto de vista de plataforma específica: combina el punto de vista independiente de la plataforma con un enfoque específico para su uso en una plataforma específica en un sistema.

Modelo independiente de la computación

Un modelo independiente de la computación (CIM) es una vista de un sistema desde el punto de vista independiente de la computación. En algunos casos, se refiere al modelo independiente de la computación como el modelo del dominio, y se usa vocabulario propio de los expertos en el dominio para la especificación.

Modelo independiente de la plataforma

Un modelo independiente de la plataforma (PIM) es una vista del sistema desde el punto de vista independiente de la plataforma. Expone un carácter independiente de la plataforma sobre la que se desplegará, de modo que pudiera ser empleado en diferentes plataformas de carácter similar. Una técnica común para alcanzar el grado de independencia de la plataforma necesario es definir un sistema basado en una máquina virtual que abstraiga los modelos particulares de las plataformas existentes y sea neutral respecto a las mismas.

Modelo específico de la plataforma

Un modelo específico de la plataforma (PSM) es una vista de un sistema desde el punto de vista dependiente de la plataforma. Combina las especificaciones del modelo inde-

pendiente de la plataforma con los detalles que especifican el uso de una plataforma específica por parte del sistema.

Modelo de plataforma

Un modelo de plataforma expone un conjunto de conceptos técnicos que representan las diferentes formas o partes que conforman un sistema, y los servicios que provee. También expone, para su uso en los modelos específicos de la plataforma, conceptos que explican los diferentes elementos que provee una plataforma para la implementación de una aplicación en un sistema.

Una modelo de plataforma incluye también la especificación de requisitos en la conexión y uso de las partes que integran la plataforma, y la conexión de aplicaciones a la plataforma.

Transformación de modelos

La transformación de modelos es el proceso de convertir un modelo en otro modelo del mismo sistema.

La Figura 1 ilustra la transformación del modelo independiente de la plataforma en un modelo específico para una plataforma mediante el uso de información añadida que permita comprobar ambos modelos.

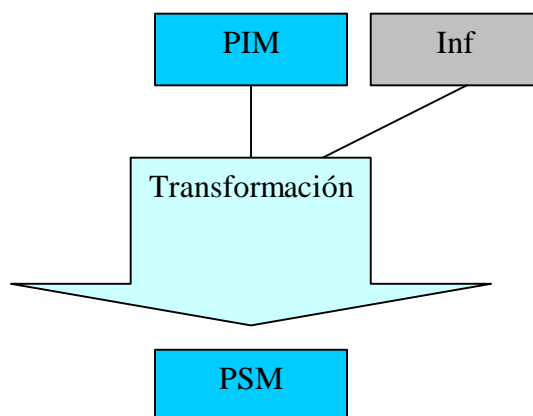


Figura 1. Transformación de modelos

La transformación de un modelo independiente de la plataforma en un modelo dependiente de la plataforma no es necesaria para PIM basados en una máquina virtual. En este caso hay que transformar el PIM correspondiente a la máquina virtual en un modelo de plataforma específico.

1.3. Ciclo de vida de desarrollo de MDA

El ciclo de vida de MDA no es muy diferente del tradicional y la diferencia reside en la naturaleza de los artefactos creados durante el proceso de desarrollo. Estos artefactos son modelos formales que pueden ser procesados por computadores.

A continuación se describen los tres modelos que son el núcleo de MDA:

1. Modelo de Plataforma independiente (PIM)

El PIM tiene un alto nivel de abstracción y es independiente de la tecnología que se vaya a emplear. Si un sistema se implementa en un ordenador principal con una base de datos relacional o de otro tipo no juega ningún papel en el PIM

2. Modelo de Plataforma Específica (PSM)

En el siguiente paso el PIM se transforma en uno o más PSM. PSM está indicado para especificar el sistema con términos que estén disponibles en una tecnología específica de implementación. Por ejemplo, una base de datos relacional incluirá términos como “table” o “column”.

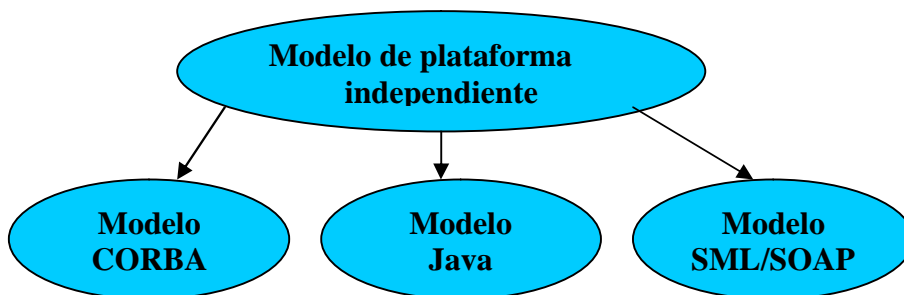


Figura 2. (Modelo dependiente de la plataforma) PSM

3. Código

El último paso en MDA es la transformación de PSM a código, para ello existen ya muchas herramientas lo que hace que la transformación sea prácticamente directa.

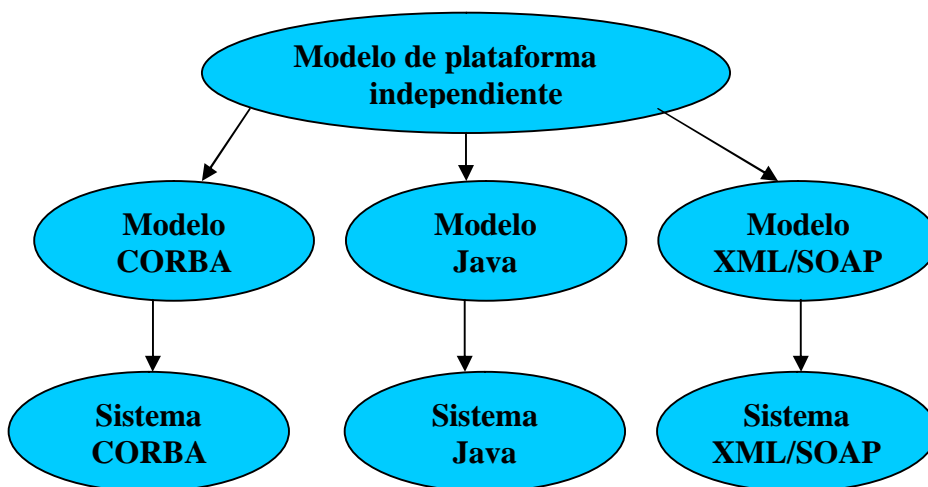


Figura 3. PSM y código automático

La interoperabilidad entre diferentes tecnologías es un punto crucial. MDA dispone de herramientas que generan puentes de forma automática uniendo las diferentes

implementaciones. Se consigue una simplificación substancial en la creación de aplicaciones integradas.

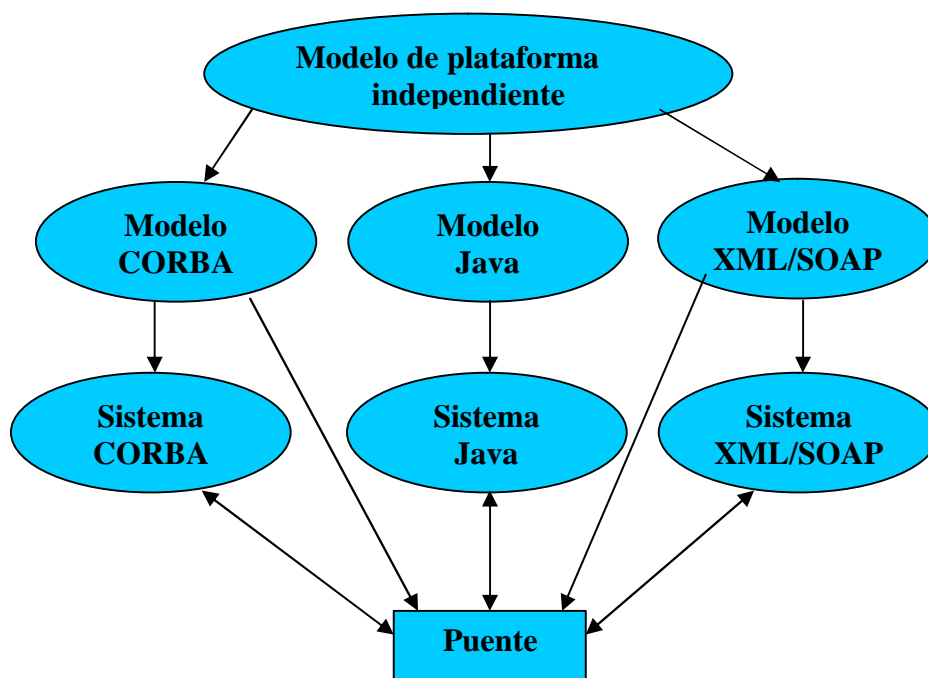


Figura 4. Interoperabilidad mediante puentes

En el modelo tradicional las transformaciones de modelo a modelo o de modelo a código se hacen a mano. En MDA todas las transformaciones se hacen con herramientas: el paso de PSM a código y la transformación de PIM a PSM (novedad en MDA).

1.4. Transformación de modelos

Una definición de transformación o *mapping* MDA proporciona la especificación de la transformación de un PIM en un PSM para una plataforma determinada. Según [2], se distinguen dos tipos de definiciones de transformaciones:

- **Transformaciones de tipos (Model Type Mapping):** según la especificación, “una transformación de tipos especifica una transformación para transformar cualquier modelo construido con tipos del PIM a otro modelo expresado con tipos del PSM”. En el caso de UML, estas reglas pueden estar asociadas a tipos del metamodelo (clase, atributo, relación, etc.) o a nuevos tipos definidos mediante *estereotipos*. También pueden definirse reglas en función de valores de instancias en el PIM.
- **Transformaciones de instancias (Model Instance Mapping):** identifica elementos específicos del PIM que deben ser transformados de una manera particular, dada una plataforma determinada. Esto se puede conseguir mediante **marcas**.

Una marca representa un concepto del PSM, y se aplica a un elemento del PIM para indicar cómo debe ser transformado. Las marcas, al ser específicas de la plataforma, no son parte del PIM. El desarrollador marca el PIM para dirigir o controlar la transformación a una plataforma determinada.

La mayoría de las definiciones de transformación consistirán en alguna combinación de los dos enfoques. Una transformación de tipos sólo es capaz de expresar transformaciones en términos de reglas sobre elementos de un tipo en el PIM que se transforman en elementos de uno o más tipos en el PSM.

Toda transformación de instancias del modelo tiene restricciones implícitas de tipo que deben cumplirse al marcar el modelo para que la transformación tenga sentido. Implícitamente a cada tipo de elemento del PIM sólo pueden aplicarse determinadas marcas, que indican qué tipo de elemento se generará en el PSM. Las transformaciones basadas en marcas deben establecer qué marcas son aplicables a qué tipos del PIM.

1.5. Estándares relacionados

Para llevar a cabo su función, MDA se apoya en estándares de OMG como son: MOF, XMI, UML, JMI, QVT...

A continuación se describirán brevemente los estándares más significativos.

1.5.1. UML (Unified Modeling Language)

El lenguaje unificado de modelado (UML) es un lenguaje gráfico para el modelado de sistemas que permite modelar la arquitectura, los objetos, las interacciones entre objetos, datos y aspectos del ciclo de vida de una aplicación, así como otros aspectos más relacionados con el diseño de componentes incluyendo su construcción y despliegue [5]. Los elementos del modelado de UML, es decir, clases, interfaces, casos de uso, diagramas de actividad, etc. pueden además ser intercambiados entre herramientas del ciclo de vida utilizando XMI. Actualmente se encuentra algunos perfiles UML para diferentes tecnologías como pueden ser CORBA, EJB, EDOC, etc. y otros muchos que están en desarrollo.

Los diferentes elementos del modelado UML permiten especificaciones estáticas y dinámicas de un sistema orientado a objetos. Los modelos estáticos incluyen la definición de clases, atributos, operaciones, interfaces y relaciones entre clases, como pueden ser la herencia, asociación, dependencia, etc. La semántica de comportamiento de un sistema puede representarse por medio del lenguaje UML gracias a sus diagramas de secuencia y colaboración. Para modelados más complejos, UML también proporciona mecanismos para la representación de máquinas de estado. Por último UML también proporciona una notación para representar el agrupamiento del diseño lógico por medio de componentes y el despliegue y ubicación de esos componentes en nodos dentro de una arquitectura distribuida.

El lenguaje UML está formalmente definido por un metamodelo, que está a su vez definido de forma recursiva en UML. Este mecanismo circular permite definir UML en un número reducido de elementos.

1.5.2. MOF (Meta Object Facility)

El OMG ha creado el estándar MOF, Meta Object Facility que extiende UML para que este sea aplicado en el modelado de diferentes sistemas de información [5]. El estándar MOF define diversos metamodelos, esencialmente abstrayendo la forma y la estructura que describe los metamodelos. MOF es un ejemplo de un meta metamodelo o un modelo del metamodelo. Define los elementos esenciales: sintaxis y estructuras de metamo-

delos que se utilizan para construir modelos de sistemas. Además, proporciona un modelo común para los metamodelos de CWM y UML

El gran potencial de MOF reside en que permite interoperar entre metamodelos muy diferentes que representan dominios diversos. Las aplicaciones que utilizan MOF no tienen por qué conocer las interfaces del dominio específico de algunos de sus modelos, pero pueden leer y actualizar los modelos utilizando las operaciones genéricas y reflexivas ofrecidas en las interfaces. La semántica de MOF define generalmente servicios para el repositorio de metadatos, para así permitir la construcción, la localización, la actualización, etc. En concreto una implementación MOF debe proporcionar herramientas para la autorización y la publicación de metadatos contra un repositorio MOF.

1.5.3. XMI (XML Metadata Interchange)

XML Metada Interchange (XMI) es un estándar que permite expresar en forma de fichero XML cualquier modelo (o meta-modelo) definido en MOF. Esta facilidad para la serialización o disposición en forma de flujo de datos (o meta-datos) ofrece un formato adecuado para intercambiar entre diferentes herramientas aquella información cuya semántica ha sido expresado en MOF [13].

La especificación XMI incluye básicamente los siguientes elementos [5]:

- Un conjunto de reglas de producción sobre definición de tipo de documentos (DTD) XML, para transformar metamodelos basados en MOF en XML DTD's.
- Un conjunto de reglas de producción sobre documentos XML, para la codificación y decodificación de metadatos basados en MOF, es decir, para representar los metamodelos en XMI.
- Principios de diseño para los DTD's y flujos XML basados en XMI.
- DTD's específicos del UML y del MOF.

Los metamodelos MOF se convierten en DTD (Document Type Definitions) y los modelos se convierten en documentos XML que son consistentes con su DTD correspondiente. XMI resuelve muchos de los problemas encontrados cuando se intenta utilizar un lenguaje basado en etiquetas para representar objetos y sus asociaciones, y además el hecho de que XMI esté basado en XML significa que tanto los metadatos (etiquetas) y las instancias que describen (elementos) se pueden agrupar en el mismo documento, permitiendo a las aplicaciones entender rápidamente las instancias por medio de los metadatos.

Muchas de las herramientas CASE¹ soportan XMI y el estándar para importar y exportar el formato. XMI no sólo se puede utilizar como un formato de intercambio UML, sino que puede ser utilizado para cualquier formato descrito por medio de un metamodelo MOF. Las herramientas compatibles con MOF permiten definir metamodelos o importarlos, por ejemplo de repositorios y herramientas CASE, y empezar a editar o modificar la información del modelo, por ejemplo instancias concretas de los servicios de negocios. Una vez hecho esto la información del modelo podría ser intercambiada con otra herramienta compatible con MOF por medio de XMI.

¹ Herramientas CASE: conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software.

UML, MOF y XMI son tres tecnologías clave para el desarrollo de software bajo el enfoque de MDA.

Utilizadas de forma conjunta proporcionan grandes ventajas que hacen que los modelados sean más claros y fácilmente mantenibles. Estas tecnologías definen una forma estándar de almacenar e intercambiar modelos, bien sean de negocio o de diseño. Las herramientas que implementan estos estándares permiten automatizar y estandarizar numerosos procesos del desarrollo que facilitan muchas tareas, que antes eran manuales o que se realizaban de forma automática por medio de alguna característica propietaria de la herramienta, que en muchos casos hacía imposible el intercambio con otras herramientas del mercado.

1.5.4. QVT (Query, Views and Transformation)

QVT [10] es un estándar que define el modo en que se llevan a cabo las transformaciones entre modelos cuyos lenguajes han sido definidos usando MOF. Consta de tres partes:

- Un lenguaje para crear vistas de un modelo.
- Un lenguaje para realizar consultas sobre modelos.
- Un lenguaje para escribir definiciones de transformaciones.

La última parte es la más relevante para MDA, pues actualmente no existe un modo estándar de definir transformaciones entre modelos [1].

La propuesta distingue entre dos tipos de transformaciones:

- **Relaciones:** especificaciones de transformaciones multidireccionales. No son ejecutables en el sentido de que son incapaces de crear o modificar un modelo. Permiten comprobar la consistencia entre dos o más modelos relacionados. Se utilizan normalmente en la especificación del desarrollo de un sistema o para comprobar la validez de un *mapping*.
- **Mappings:** implementaciones de transformaciones. A diferencia de las relaciones, los *mapping* son unidireccionales y pueden devolver valores. Un *mapping* puede *refinar* una o varias relaciones, en cuyo caso el *mapping* debe ser consistente con las relaciones que refina.

La propuesta también incluye un lenguaje estándar para definir relaciones y *mapping*, llamado Model Transformation Language o MLT. MLT utiliza el *pattern matching* como uno de sus factores clave para permitir definir transformaciones potentes. La idea esencial del *pattern matching* es permitir expresar brevemente restricciones complejas sobre un tipo de dato de entrada, los datos que cumplen el patrón se seleccionan y se retornan al invocador.

1.5.5. JMI (Java Metadata Interface)

JMI es un estándar que surge del Java Community Process (JCP) que define una implementación dinámica e independiente de la plataforma que permite la creación, acceso, almacenamiento e intercambio de metadatos. Esta tecnología está basada en la especificación MOF y define la transformación de MOF a Java. Este estándar especifica una forma de generar interfaces Java para acceder a metadatos descritos en MOF.

JMI proporciona el marco de trabajo (the metadata framework) para capturar la semántica específica del sistema [22]. Dispone de las reglas necesarias para generar la interfaz java para cualquier modelo MOF y trabajar con la información contenida en el modelo. Permite modelar sistemas dispares, aplicaciones y servicios en un modelo común. Este modelo común puede utilizarse para integrar estos sistemas dispares. JMI proporciona modelos de alto nivel de sistemas y dominios de negocios y oculta las complejidades de los sistemas. Proporciona un modelo común de programación que está en la forma de la interfaz generada automáticamente desde el metamodelo. Las interfaces proporcionan un formato común de intercambio basado en XML, se trata del XMI. JMI lleva el MDA hacia la tecnología J2EE.

JMI define plantillas para generar las interfaces de java, estas interfaces se utilizan para acceder y modificar los metadatos. La interfaz se genera a partir de un modelo MOF.

La herramienta MDA ArcStyler (descrita en el apartado 2.4.2) se basa íntegramente en JMI. Es la base común para el acceso a cualquier modelo a través de la herramienta. El principal beneficio de esto es que un gran número de repositorios son conformes a JMI, con lo que sólo es necesario un pequeño esfuerzo para integrar esos repositorios en ArcSyler.

1.3.6. CWM (Common Warehouse Metamodel)

La especificación del CWM [11] contiene las directivas necesarias para poder almacenar la meta-información de cualquier modelo mediante un formato estándar y fácilmente intercambiable. CWM se compone de varios sub-modelos como el sub-modelo multi-dimensional y el sub-modelo OLAP que permiten representar información relativa al modelado multidimensional (MD).

CWM fue concebido como un estándar muy general para asegurar un amplio consenso para su utilización por la comunidad científica y empresarial. Así, los sub-modelos MD y OLAP mencionados anteriormente adolecen en la representación de algunas características básicas en el modelado MD.

CWM utiliza UML, MOF y XMI para modelar, manipular e intercambiar respectivamente almacenes de datos (*data warehouse*). CWM es una especificación de sintaxis y semántica en la que herramientas de *data warehousing* y *Business Intelligence* pueden apoyarse para intercambiar metadatos. También se dice que es un *framework*² para especificar representación externa de *metadata* de *data warehouse* con propósitos de intercambio.

1.3.7. ODM (Ontology Definition Metamodel)

ODM es un metamodelo de definición de ontologías³ que pertenece a la familia de metamodelos de MOF [12]. Permite tanto la transformación entre dichos metamodelos como desde ellos a UML y viceversa.

² **Framework:** Modelado de un caso de uso, operación o mecanismo.

³ **Ontología:** especificación de una conceptualización. Mediante la definición de ontologías se pretende reunir y formalizar el conocimiento sobre un determinado problema de problema.

Los metamodelos que incluye el ODM reflejan una sintaxis abstracta para representar el conocimiento de forma estándar y lenguajes de modelado conceptual.

2. DEFINICIÓN DE LENGUAJES DE MODELOS DENTRO DE MDA

2. DEFINICIÓN DE LENGUAJES DE MODELOS DENTRO DE MDA

En este apartado se explicará el significado de metamodelado y el por qué de su relevancia dentro de MDA. Se centrará el estudio en los metamodelos UML 2.0 y MOF así como en herramientas dentro de MDA y casos de uso.

2.1. Introducción

El metamodelado es un mecanismo que permite definir formalmente lenguajes de modelado [1]. Un metamodelo de un lenguaje es una definición precisa de sus elementos mediante conceptos y reglas de cierto metalenguaje, necesaria para crear modelos en ese lenguaje. Se trata de usar modelos para describir otros modelos.

El OMG utiliza una arquitectura de cuatro niveles o capas de modelado para sus estándares. En la terminología del OMG estas capas se llaman **M0**, **M1**, **M2** y **M3**.

Capa M0. Instancias

En el nivel M0 están todas las instancias reales del sistema, es decir, los objetos de la aplicación. Aquí no se habla de clases ni atributos, sino de entidades que existen en el sistema.

Capa M1. Modelo del sistema

Por encima de la capa M0 está la capa M1, que representa el modelo de un sistema software. Los conceptos del nivel M1 representan categorías de las instancias de M0. Es decir, cada elemento de M0 es una instancia de un elemento de M1.

Capa M2. Metamodelado

Al igual que en los casos anteriores, los elementos del nivel M1 son a su vez instancias del M2. En este nivel aparecerán conceptos como clase, atributo o relación.

Capa M3. Meta-metamodelo

Al igual que en los casos anteriores, los elementos del nivel M2 son a su vez instancias del M3. Dentro del OMG, MOF es el lenguaje estándar de la capa M3. Esto supone que todos los metamodelos de la capa M2, por ejemplo, el metamodelo de UML, es instancia de MOF. Es decir, UML se define utilizando MOF (estándar que se detallará en el apartado 2.2.2).

En la siguiente tabla se muestra un resumen de las distintas capas de modelado:

Capa	Contenido	Ejemplo
M0 Instancias	Instancias reales del sistema.	Alumno con nombre="Dario Ruiz" y DNI=48496589.
M1 Modelo	Entidades del modelo del sistema.	Clases "Alumno" con atributos "nombre" y "DNI".
M2 Metamodelo	Entidades de un lenguaje de modelado.	Entidad "UML Class" del metamodelo de UML.
M3 Meta-metamodelo	Entidades para definir lenguajes de modelado.	Entidad "MOF Class" de MOF

Tabla 1. Capas de modelado del OMG

El metamodelado es importante en MDA porque actúa como mecanismo para definir lenguajes de modelado, de forma que su definición no sea ambigua. Esta no ambigüedad permite que una herramienta pueda leer, escribir y entender modelos como los construidos con UML.

El metamodelado es importante en la transformación de lenguajes. Las reglas de transformación utilizadas en la transformación de lenguajes utilizan los metamodelos de los mismos.

2.2. Metamodelos UML 2.0 y MOF

2.2.1. UML 2.0

La versión 2.0 de UML viene acompañada de nuevas versiones de los estándares MOF, XMI y OCL (Object Constraint Language) [13]. Es un intento de coordinación general de varios estándares para dar un soporte más formal y completo a la filosofía general de MDA. Concretamente, MOF 2.0 se basa en el núcleo de UML y permite la creación formal de un número indeterminado de niveles de abstracción. XMI 2.0 permite la serialización automática de los modelos UML en forma de ficheros XML y de los metamodelos en forma de XML Schemas. Está recomendado su uso para el intercambio estándar de información entre herramientas UML y para el almacenamiento de los modelos en repositorios XML.

Mientras que UML 1.4 tenía como objetivo principal satisfacer a las necesidades clásicas de la industria del software, UML 2.0 se plantea cubrir otros campos como el modelado de negocio, de sistemas de tiempo real, basado en componentes, etc. El mecanismo estándar para extender las capacidades de UML a cada uno de estos campos específicos es el de los perfiles y este mecanismo se ve mejorado en la nueva versión. En general, se mejora la escalabilidad y usabilidad del lenguaje en todos los ámbitos.

Características generales de UML 2.0:

- Simplificación de la sintaxis y semántica del lenguaje.
- Alineación formal con MOF.

- Soporte del modelado de la arquitectura software y del modelado basado en componentes, lo que conlleva el uso de una nueva semántica.
- Se refina el modelo de desarrollo y el de proceso, gracias a la mejora de los diagramas de actividad, de interacciones y máquinas de estados.
- Nuevo mecanismo de extensión para añadir metaclasses propias y mejorar así la definición de perfiles.
- Soporte de arquitecturas en tiempo de ejecución para modelar flujo de datos y objetos entre diferentes partes del sistema.
- Mejor representación de las relaciones.
- Mejor modelado de comportamiento con encapsulado y escalabilidad.
- Modelos ejecutables (nueva semántica de acción).

UML 2.0 promete convertirse en una solución general, haciendo innecesario el uso de extensiones de su metamodelo para tratar los problemas específicos de dominios específicos.

Perfiles de UML

Un **perfil UML** se define como un conjunto de *estereotipos*, *restricciones* y *valores etiquetados* [1]. Estos conceptos forman parte del mecanismo de extensión de UML. A continuación se explican cada uno de ellos:

- Mediante **estereotipos** se pueden crear nuevos tipos de elementos de modelado basados en elementos ya existentes en el metamodelo de UML. Por lo tanto, un estereotipo será un nuevo tipo de elemento de modelado que extiende la semántica del metamodelo.
- Las **restricciones** imponen condiciones que deben cumplir determinados elementos del modelo para que éste esté “bien formado”, según un dominio de aplicación específico. Una restricción generalmente se define mediante una expresión en OCL⁴ (Object Constraint Language).
- Un **valor etiquetado** es una extensión de las propiedades de un elemento de UML, permitiendo añadir nueva información en la especificación del elemento.

En la Figura 5 se muestra un ejemplo de la representación gráfica de estos tres elementos en UML.

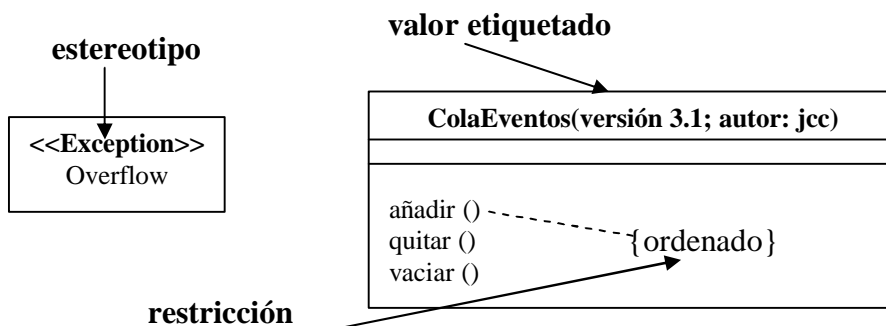


Figura 5. Ejemplo de *estereotipo*, *restricción* y *valor etiquetado* en UML

⁴ OCL es un lenguaje notacional (subconjunto de UML estándar), que permite escribir restricciones sobre modelos de objetos.

Para obtener un perfil se tendrá que especializar un subconjunto de UML a través de estereotipos, restricciones y valores etiquetados. Como resultado de crear un *perfil UML* se obtiene una variante de UML para un propósito específico, que se puede usar para completar un modelo con detalles específicos de una tecnología o plataforma determinada, o lo que es lo mismo, se puede usar como **lenguaje para definir PSMs**.

Los perfiles UML permiten:

- Disponer de una terminología y vocabulario propio de un dominio de aplicación o de una plataforma de implementación concreta.
- Definir una nueva notación para símbolos ya existentes, más acorde con el dominio de la aplicación final.
- Añadir cierta semántica que no existe o no aparece determinada de forma precisa en el metamodelo.
- Añadir restricciones a las existentes en el metamodelo, restringiendo su forma de utilización.
- Añadir información que puede ser útil a la hora de transformar el modelo a otros modelos o a código.

Actualmente existen perfiles para CORBA, Java, EJB o C++, lo que permitirá construir PSMs específicos para estas tecnologías.

2.2.2. MOF

MOF es el estándar definido por OMG (Object Management Group) para la definición, representación y gestión de metadatos [12]. Es uno de los estándares más importantes usados en MDA. Define un lenguaje común y abstracto para definir lenguajes de modelado. MOF y sus especificaciones asociadas incluyen:

- La **arquitectura MOF de 4 niveles de metadatos**, que proporciona un patrón genérico para la construcción de sistemas centrados en los metadatos.
- El **modelo MOF** es el lenguaje estándar y abstracto para los metamodelos que definen diferentes clases de metadatos.
- El **repositorio de metamodelos MOF** proporciona un servicio estándar para crear, modificar, validar y acceder a los metamodelos.
- La **especificación XMI** define las correspondencias que soportan el intercambio de metadatos y metamodelos basados en MOF.

El marco de trabajo establecido por el estándar MOF se describe típicamente como una arquitectura conceptual de cuatro capas de metadatos descrita anteriormente en la Tabla 1.

Como se puede observar en la Tabla 1, el nivel tres está formado por el modelo MOF, que constituye el metalenguaje para la definición de metamodelos y que es lo que distingue esta especificación de otras basadas en este tipo de arquitecturas conceptuales. Las cinco construcciones básicas para definir un lenguaje de modelado son:

- **Clases:** usadas para definir tipos de elementos en un lenguaje de modelado
- **Generalización:** define herencia entre clases.
- **Atributos:** usados para definir propiedades de elementos del modelo. Los atributos tienen un tipo y una multiplicidad.

- **Asociaciones:** definen relaciones entre clases. Una asociación tiene dos extremos, cada uno de los cuales puede tener definido un nombre de rol, navegabilidad y multiplicidad.
- **Operaciones:** definen operaciones dentro del ámbito de una clase, junto con una lista de parámetros.

Atendiendo a las 4 capas de modelado del OMG (descrito en el apartado 2.1), MOF estaría situado en el nivel M3 (nivel de meta-metamodelado). Mediante MOF puede definirse cualquier lenguaje de modelado, incluido UML.

Para la representación e intercambio de metadatos, MOF se basa en el estándar XMI. En la Figura 6 se puede observar la relación entre MOF y XMI.

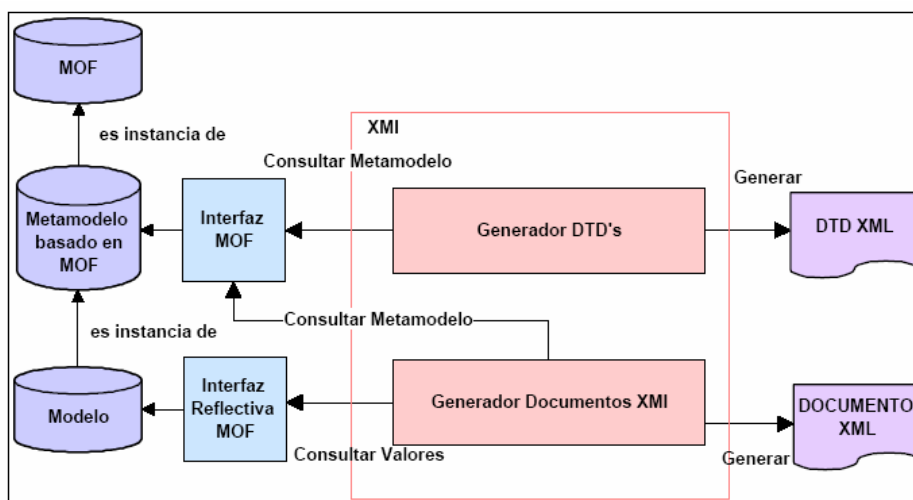


Figura 6. Relación entre MOF y XMI

Como se puede apreciar en la Figura 6, los dos módulos que conforman la especificación XMI acceden a las interfaces proporcionadas por MOF (Interfaz MOF e Interfaz Reflectiva MOF) para obtener la información sobre los modelos y metamodelos, a partir de los cuales es posible construir los documentos XML (que representan los modelos) y los DTD's asociados (que describen los metamodelos basados en MOF).

A partir de la versión de XMI 2.0, la gramática de los lenguajes XML, es decir, la estructura y elementos permitidos en los documentos, además de definirse mediante DTD, se definen mediante XSD (XML Schema Definition). Estos mejoran los DTD's porque están escritos en XML y permiten nuevas características como: definir tipos de datos, utilizar espacios de nombre, definir intervalos de valores para los atributos y elementos, características OO...

Con el uso de los estándares MOF/XMI se facilita la gestión, representación e intercambio de metadatos de forma abierta y efectiva.

El papel principal de MOF dentro del MDA es proporcionar los conceptos y herramientas para razonar sobre lenguajes de modelado.

2.3. Caso de estudio MOF - MDA

En este apartado se va a realizar un estudio de un Lenguaje para entidades de negocios. El lenguaje tiene que ser capaz de capturar la información de una aplicación bancaria.

Para ello se seguirá la arquitectura de cuatro niveles.

Capa M0. Instancias

En el nivel M0 están todos los objetos de la aplicación bancaria, es decir clientes, cuentas bancarias y transacciones, con sus correspondientes atributos indicados en la siguiente figura.

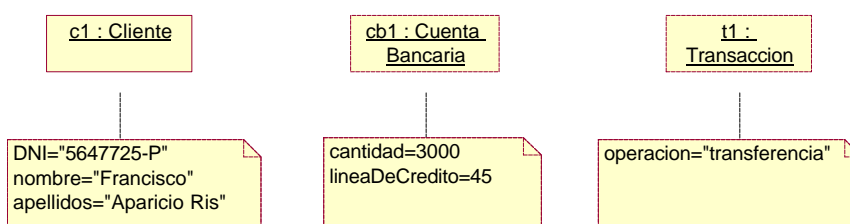


Figura 7: Instancias del sistema (capa M0)

Capa M1. Modelo del sistema

Los conceptos del nivel M1 representan categorías de las instancias de M0, en este nivel aparecería la entidad Cliente con los atributos DNI, nombre y apellidos, la entidad Cuenta Bancaria con los atributos cantidad y lineaDeCredito y la entidad Transacción con el atributo operación.

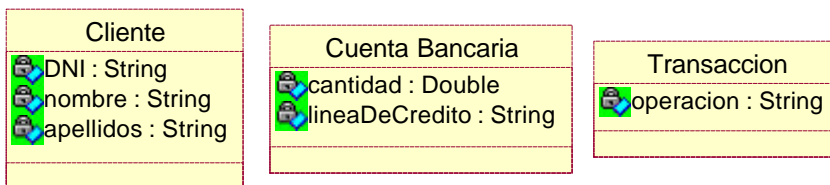


Figura 8: Entidades del modelo del sistema (capa M1)

Capa M2. Metamodelo

Los elementos del nivel M1 son instancias del nivel M2. La entidad Cliente sería una instancia de la metaclass UML Class del metamodelo de UML junto a la metaclass UML Attribute. Las relaciones entre clases serían instancias de la metaclass UML Association.

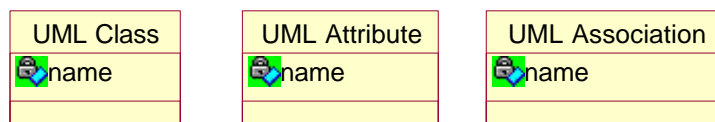


Figura 9: Entidades del metamodelo de UML (capa M2)

Capa M3. Meta-Metamodelo

Los elementos de la capa M2 son instancias de la capa M3, es decir UML se define usando MOF.

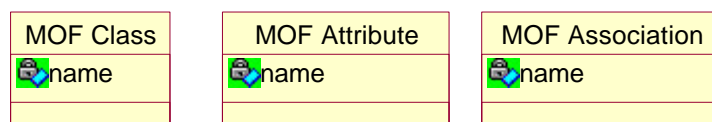
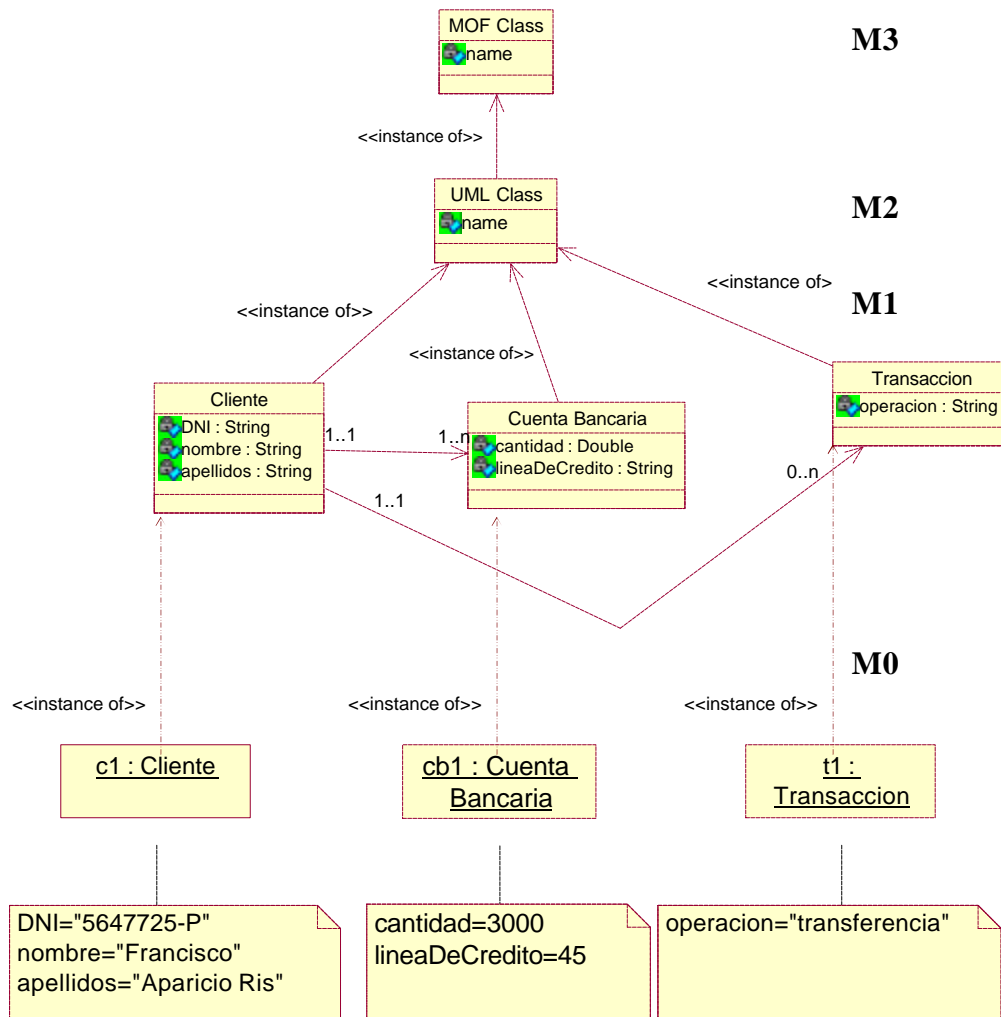


Figura 10: Entidades de MOF (capa M3)

Visión completa de las capas

En la siguiente figura se muestra un ejemplo completo donde se aprecia la relación existente entre las capas del modelado definidas por el OMG.



2.4. Herramientas

Para cumplir las promesas de MDA se necesitan herramientas que den soporte a este nuevo framework: transformación de modelos, verificación de modelos, generación de código, etc.

El soporte para MDA se puede dar en distintas variantes. La generación de código a partir de modelos encaja bien dentro del marco de MDA. Además de la generación de código, se necesita otros tipos de transformación, fundamentalmente transformaciones de modelos, para dar soporte completo a MDA. Estas transformaciones pueden ser implementadas por distintas herramientas.

Por lo tanto se encuentran las siguientes herramientas:

- Herramientas de transformación de PIM a PSM.
- Herramientas de transformación de PSM a código.

- Herramientas de transformación de PIM a código.
- Herramientas de definición de transformaciones.

2.4.1. OptimalJ

OptimalJ de Compuware es una herramienta para dar soporte a MDA [15]. Se trata de un entorno de desarrollo de aplicaciones empresariales que permite generar con rapidez aplicaciones J2EE completas directamente a partir de un modelo de alto nivel (PIM), utilizando patrones para codificar adecuadamente las especificaciones de J2EE y haciendo uso de los estándares del MDA. OptimalJ implementa el framework MDA haciendo uso de tecnologías estándar como MOF, UML, XMI, XML, WSDL y J2EE. Actualmente se encuentra en la versión 4.0

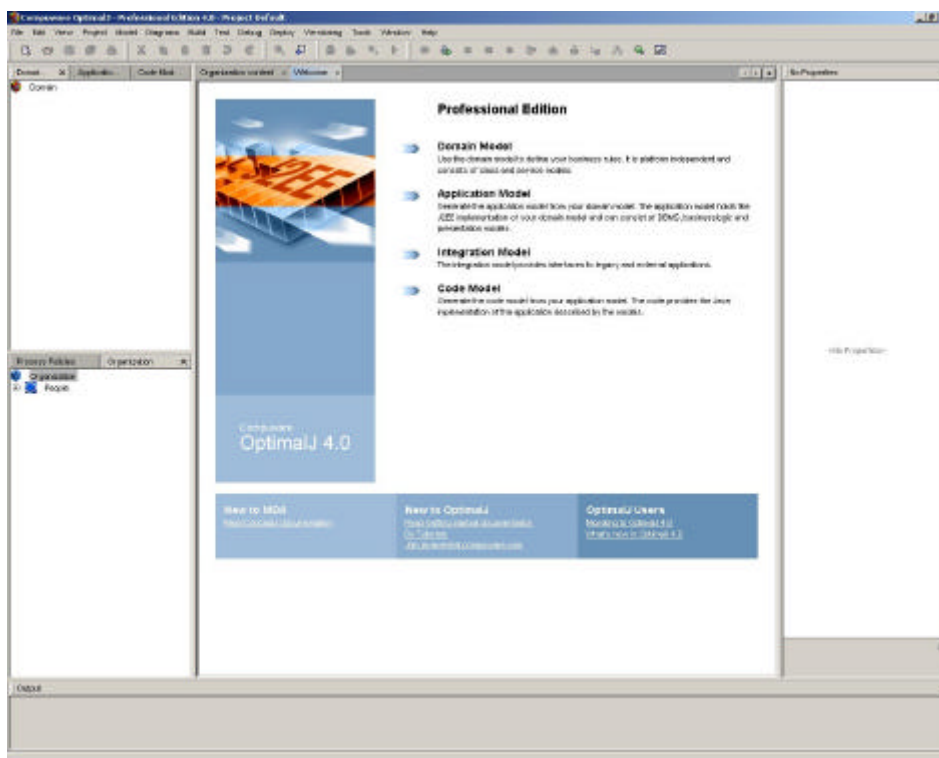


Figura 11: OptimalJ 4.0 de Compuware

En OptimalJ existen tres tipos de modelos [16]:

- **Modelo del Dominio (Domain Model):** modelo que describe el sistema a un alto nivel de abstracción, sin detalles de implementación. Corresponde al PIM de la aplicación y su elemento principal es un modelo de clases del negocio.
- **Modelo de la Aplicación (Application Model):** modelo del sistema desde el punto de vista de una tecnología determinada (J2EE). Contiene los PSM de la aplicación. Se genera automáticamente a partir de un modelo del dominio y está formado por tres modelos: modelo de base de datos, modelo de interfaz web y modelo EJB.
- **Modelo de Código (Code Model):** código de la aplicación, generado a partir de un modelo de la aplicación.

OptimalJ usa dos tipos de patrones:

Patrones de transformación **entre** modelos:

- **Patrones de tecnología** (*Technology patterns*): transforma el modelo del dominio (PIM) en el modelo de aplicación (PSMs)
- **Patrones de implementación** (*Implementation patterns*): transforman el modelo de aplicación (PSMs) en código.

Patrones funcionales para hacer transformaciones **dentro** de un modelo. Estos patrones promueven *best practices*⁵, reducen los errores y aceleran el desarrollo de la aplicación. Optimal J ofrece:

- **Patrones de dominio** (*Domain patterns*): patrones definidos por el usuario que permiten a los desarrolladores capturar, reutilizar y distribuir modelos del dominio. Un patrón de dominio es un modelo del dominio que puede ser reutilizado, de manera que podemos construir un nuevo modelo del dominio rápidamente basado en el conocimiento existente.
- **Patrones de aplicación** (*Application patterns*): usan el mismo principio que los patrones del dominio, pero aplicados a los modelos de aplicación.
- **Patrones de código** (*Code patterns*): patrones de bajo nivel aplicados al código.

En la Figura 12 se puede ver un esquema de los tipos de modelos y de patrones disponibles en OptimalJ.

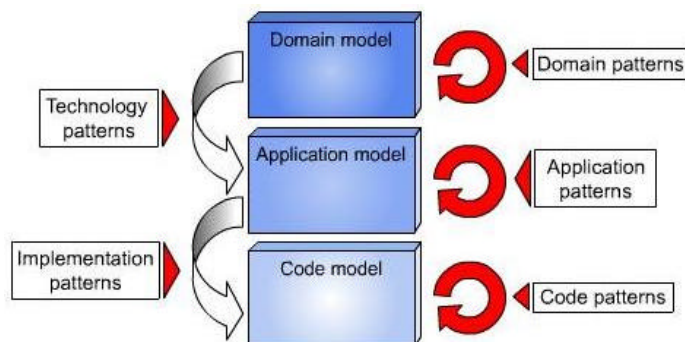


Figura 12. Tipos de modelos y patrones en OptimalJ, extraído de [20]

A continuación se explicarán cada tipo de modelo con más detalle.

Modelo del dominio (*Domain Model*)

En OptimalJ el PIM está incluido dentro del **Modelo del dominio** (*Domain Model*). Este modelo del dominio está formado a su vez por dos modelos: el **Modelo de Clases** y el **Modelo de Servicios**.

- **Modelo de Clases** (*Domain Class Model*): se define la estructura de la información con la que trabaja la aplicación mediante un diagrama de clases UML. Este modelo es el **PIM** de la aplicación.

⁵ Mejor solución encontrada para un problema determinado

- **Modelo de servicios (*Domain Service Model*):** permite definir vistas sobre las clases definidas en el modelo de clases. La funcionalidad en este modelo consiste en limitar el acceso a las acciones de crear, leer, modificar o eliminar instancias, y ocultar la aparición de determinados atributos en la interfaz de la aplicación generada.

Modelo de Aplicación (*Applicatin Model*)

Una vez construido el modelo del dominio, OptimalJ permite generar automáticamente el Modelo de Aplicación, modelo que consta de varios PSMs orientados hacia la plataforma J2EE. Esta transformación la llevan a cabo los patrones de tecnología.

El modelo de aplicación generado contiene tres tipos de modelos: el **Modelo de Presentación**, el **Modelo de Negocio** y el **Modelo de Base de Datos**.

- **Modelo de Presentación (Web):** este modelo contiene la información necesaria para generar una capa Web para la aplicación, según la plataforma J2EE: Módulos Web, Web Data Schemas, Componentes Web, Componentes de Autenticación Web y Tipos de Presentación Web.
- **Modelo de Negocio (EJB):** el modelo *Enterprise Java Bean* es una capa *middleware* encargada, entre otras cosas, de las transacciones, la seguridad, la persistencia y la escalabilidad de la aplicación. OptimalJ usa las definiciones contenidas en el modelo del dominio para generar automáticamente el modelo de EJB y los componentes de tipo entidad y sesión (*entity beans* y *session beans*). El modelo EJB está formado por componentes de tipo entidad, *Data Schemas*, *Key Classes* y otros componentes relacionados con la tecnología EJB.
- **Modelo de Base de Datos:** la persistencia de los datos es un aspecto importante en el desarrollo de aplicaciones J2EE. La aproximación más común es usar bases de datos relacionales para almacenar los datos. Para hacer esto, es necesario una conversión objeto-relacional entre el modelo de objetos, representado por el modelo del dominio, y la base de datos. OptimalJ maneja esta correspondencia generando automáticamente un **Modelo de Base de Datos** a partir del modelo del dominio usando **Patrones de Tecnología**. El modelo de base de datos de OptimalJ se usa para modelar todas las definiciones relevantes de la base de datos. Este modelo soporta esquemas, tablas, columnas, filas, claves ajenas, claves primarias y restricciones de unicidad.

Modelo de Código (Code Model)

Una vez generado el modelo de aplicación a partir del modelo del dominio, OptimalJ automáticamente transforma el Modelo de la Aplicación en código haciendo uso de los **Patrones de Implementación**.

El código para la lógica de negocio (EJB), base de datos y presentación (Web) puede generarse automáticamente de forma separada, o todos a la vez. Este proceso de generación automática a partir del modelo de aplicación asegura la consistencia con el modelo del dominio, ahorrando una gran cantidad de tiempo y reduciendo el riesgo potencial de errores en la programación. Naturalmente, el modelo Web depende del modelo EJB, y éste depende del Modelo de Base de Datos. OptimalJ mantiene esta interdependencia entre modelos y la traslada también al código generado.

El código que genera OptimalJ es totalmente operativo y conforme a las especificaciones de J2EE. El código generado consiste en ficheros Java, JSPs y XML, *beans* de tipo entidad y de tipo sesión, y código SQL.

Todo el código generado por OptimalJ se sitúa en bloques protegidos. Esto significa que los desarrolladores no pueden modificarlo. No obstante, OptimalJ deja disponibles bloques libres donde los desarrolladores pueden añadir sus propias extensiones al código generado. Cuando la aplicación es regenerada, por ejemplo, por un cambio en el modelo del dominio, la herramienta preserva el código escrito en los bloques libres.

OptimalJ 4.0 amplía sus capacidades para el desarrollo de aplicaciones Java mediante MDA (Model Driven Architecture), y con la que se ofrece también soporte para la plataforma de desarrollo “open source” Eclipse⁶ [19]. Las nuevas capacidades de OptimalJ 4.0 incluyen un nuevo modelo de proceso, que permite a las organizaciones de TI desarrollar aplicaciones Java empresariales de mayor calidad y de manera extremadamente rápida, proporcionando así una gran productividad en el desarrollo de aplicaciones en este entorno.

OptimalJ es la pieza central de las soluciones avanzadas de desarrollo de Compuware, que combinan metodología probada, tecnología líder y servicios profesionales para permitir a los equipos de desarrollo en Java conseguir de manera extremadamente rápida aplicaciones orientadas a servicios. Con esta nueva solución Compuware continua extendiendo las capacidades de automatización de OptimalJ e incorpora funcionalidades basadas en roles que posibilitan a todo el equipo de desarrollo implantar soluciones usando una aproximación MDA altamente pragmática.

La nueva solución OptimalJ 4.0 incluye una nueva funcionalidad de desarrollo orientado a procesos con la que avanza un paso más en la MDA, al presentar un modelo de proceso basado en el diagrama de actividad UML 2.0. Con ello, los desarrolladores pueden definir el flujo de una aplicación y transformarlo automáticamente en componentes de una aplicación J2EE, pudiendo así generar aplicaciones con una productividad y calidad sin precedentes.

Por último, OptimalJ 4.0 ofrece soporte para la plataforma Eclipse, el “framework” de Java de más rápido crecimiento en la actualidad. Compuware ha anunciado que todas las versiones futuras de OptimalJ serán construidas sobre la plataforma Eclipse.

2.4.2. ArcStyler

ArcStyler de iO-Software [21] es una herramienta MDA que también utiliza MOF para soportar estándares como UML y XMI, y además JMI para el acceso al repositorio de modelos. Integra herramientas de modelado (UML) y desarrollo (ingeniería inversa, explorador de modelos basado en MOF, construcción y despliegue) con la arquitectura CARAT que permite la creación, edición y mantenimiento de cartuchos MDA (MDA-Cartridge) que definen transformaciones. También incluye herramientas relacionadas con el modelado del negocio y el modelado de requisitos por lo que cubre todo el ciclo de vida. Se comentará brevemente los aspectos más relevantes de ArcStyler y se indica-

⁶ Eclipse: es una poderosa herramienta independiente de cualquier plataforma con un código *open source* y extensible mediante *plugins*

rá cómo construir una aplicación de tres capas como la que soporta OptimalJ. Actualmente ArcStyler está disponible en la versión 5.1.

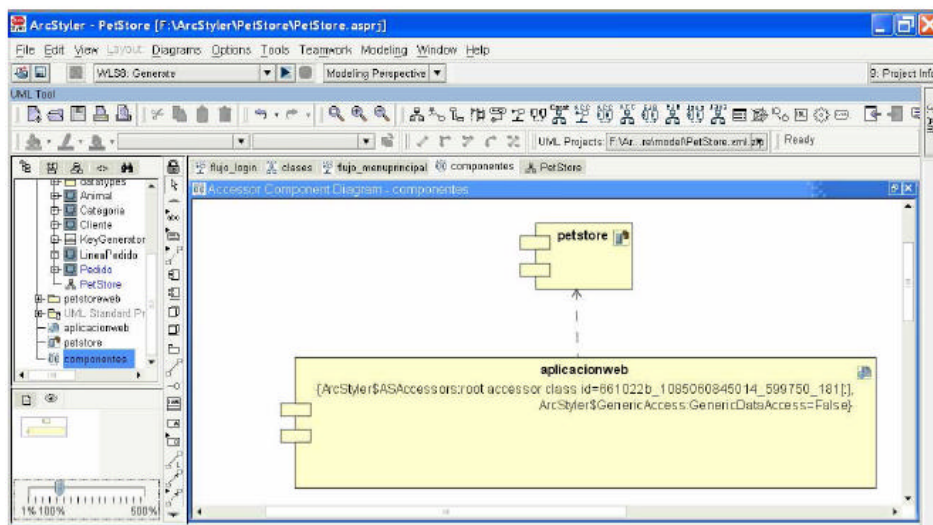


Figura 13: ArcStyler

Arquitectura CARAT

Un cartucho contiene un conjunto de reglas de transformación y se instala como un plugin. El lenguaje de script Jpython es utilizado para la creación de cartuchos. Actualmente existen numerosos cartuchos para diferentes plataformas, como J2EE, .NET, servicios web, etc. Los cartuchos utilizan Perfiles UML para incluir en los modelos aspectos específicos de una plataforma, por ejemplo un perfil para EJB o Java 2. Una funcionalidad muy potente relacionada con los cartuchos es la herencia de cartuchos: siendo posible definir un cartucho a partir de otro existente. A través de cartuchos podríamos definir todos los modelos de OptimalJ.

Marcas

En la especificación de MDA se distingue dos modos de anotar un PIM con información específica de una plataforma: Correspondencia de tipos (Model type mapping) y Correspondencia de instancias (Model instance mapping) [2]. ArcStyler proporciona los dos modos a través de estereotipos y marcas MDA. Las marcas son anotaciones sobre elementos de un modelo que proporcionan información a los cartuchos y que junto con los estereotipos dirigen una transformación. Un cartucho proporciona una definición del conjunto de marcas permitido para una determinada plataforma. Una marca se define con la siguiente información: tipo de dato asociado, elementos del modelo a los que se aplica y valor por defecto. Un editor inteligente de marcas evalúa las definiciones en tiempo de modelado y proporciona el soporte apropiado para marcar los modelos para una plataforma específica.

Framework Accesor

Establece un proceso bien definido para elaborar la capa de presentación de las aplicaciones, mediante la definición de un modelo *accesor* se define el flujo de control y el

flujo de datos de la capa de presentación de un modo abstracto. En el proceso de generación de código, el modelo *Accessor* se implementa para una tecnología determinada. Un modelo *Accessor* incluye dos tipos de elementos:

- Un *accessor* juega el rol de un Controlador en la arquitectura MVC, y describe el flujo de control de una interfaz externa. Este comportamiento se modela en un dia-grama de estados extendido de UML.
- Un *representer* juega el rol de una Vista en la arquitectura MVC. Se usa para describir un componente básico de la interfaz de usuario, por ejemplo una página JSP o ASP.NET. Se modela mediante hojas de propiedades especiales proporcionadas por ArcStyler.

Un elemento *accessor* puede corresponder a un caso de uso del sistema expresado de forma detallada y concreta. El diagrama de estados asociado a un *accessor* establece la secuencia de acciones que caracteriza a un determinado caso de uso. Esta analogía resulta muy interesante en un desarrollo dirigido por casos de uso.

2.4.3. AndroMDA

AndroMDA es una herramienta de generación de código que sigue el paradigma de arquitectura MDA (Model Driven Architecture). Recibe un modelo UML de una herramienta CASE y genera las clases y los componentes (J2EE u otros), específicos para la arquitectura de la aplicación. Debido a que su generador de código soporta plataformas actuales, se ha convertido en la principal herramienta Open Source de MDA para el desarrollo de aplicaciones empresariales.

AndroMDA incluye un conjunto de cartuchos enfocados a los Kits de desarrollo actuales como son Axis, jBPM, Struts, JSF, Spring and Hibernate. AndroMDA también incluye un Kit para desarrollar propios cartuchos generadores de código o personalizar los existentes (el cartucho Meta). Utilizando dicho cartucho se puede construir un generador de código propio mediante una herramienta UML.

El equipo de AndroMDA acaba de anunciar la versión 3.1. Esta nueva versión incluye nuevas funcionalidades, correcciones de errores y mejor documentación.

Esta nueva versión AndroMDA se puede ejecutar en modo servidor que monitoriza los cambios en los modelos y los mantiene cargados y validados de manera que no tiene que hacerlo en cada generación de código.

AndroMDA se configura con un único fichero de configuración (normalmente llamado *andromda.xml*). Esto desacopla AndroMDA de Maven⁷ lo que permite empotrarlo más fácilmente en diferentes herramientas como son maven1, maven2, Eclipse, Ant, etc.

3.4.4. IBM XDE

IBM Rational Rose XDE Modeler es un entorno de modelado UML para arquitectos y diseñadores de software y está construido sobre la plataforma abierta y Extensible de Eclipse, lo que le proporciona una capacidad de extensión. Permite crear modelos UML independientes del lenguaje de arquitectura de software, necesidades empresariales, activos reutilizables y comunicación a nivel de gestión. Dispone de un único entorno de

⁷ Maven: herramienta software para la gestión y comprensión de proyectos Java. Estaba integrado dentro del proyecto *Jakarta* pero ahora ya es un proyecto de nivel superior de la *Apache Software Foundation*.

diseño. Está basado en modelos con soporte para UML y da soporte para múltiples modelos para la Arquitectura basada en modelos (MDA). Se ejecuta de forma independiente o integrado con Microsoft Visual Studio .NET. Permite la creación de arquitecturas independientes de la plataforma que se pueden implementar en plataformas Java y .NET. Dispone de Patrones definibles por el usuario para crear, personalizar y aprovechar patrones de diseño arquitectónico. Las referencias cruzadas entre modelos y el control de versiones a nivel de clase y diagrama permiten una estructuración que se ajusta a cualquier proyecto. Permite conservar la capacidad de rastreo entre los modelos de análisis, diseño e implementación.

3.4.5. Eclipse GMT

Generative Model Transformer (GMT), se un proyecto de eclipse que proporciona tecnología (la herramienta FUUT-je y un generador de código) para la transformación de modelos bajo la plataforma eclipse. Está formado por un conjunto de subproyectos que implementan las diferentes partes de una herramienta MDA. El mayor inconveniente es que los diferentes proyectos no están muy integrados unos con otros.

3. DEFINICIÓN DE LENGUAJES DE MODELOS USANDO DSLs

3. DEFINICIÓN DE LENGUAJES DE MODELOS USANDO DSLs

3.1. Introducción

Existe una estrategia de desarrollo del software paralela al MDA llamada Factorías del Software. Las Factorías del Software constituyen un enfoque para el desarrollo de software promovido principalmente por Microsoft. Una factoría de Software, tal y como se define en [17], *es una línea de productos software que configura herramientas extensibles, procesos y contenido [...] para automatizar el desarrollo y mantenimiento de variantes de un producto arquetípico mediante la adaptación, en ensamblaje y configuración de componentes basados en frameworks*. Las Factorías de Software se centran en el desarrollo de sistemas similares promoviendo la reutilización de arquitecturas, componentes software y conocimiento. Para alcanzar este objetivo, las Factorías Software integran varias técnicas conocidas en la Ingeniería del Software. Una de las principales actividades que promueve es el **desarrollo de lenguajes de modelado y herramientas específicas para el dominio**. Los desarrolladores utilizan estos lenguajes para describir los requisitos específicos de un miembro de la familia de sistemas. A partir de estas especificaciones se genera automáticamente el código de la parte específica del sistema. Es en esta actividad donde aparece la importancia de los DSLs, que son una de las guías metodológicas utilizadas por las Factorías de Software.

Los DSLs (*Domain Specific Language*) son lenguajes específicos del dominio. Un lenguaje específico del dominio es un lenguaje de programación que está diseñado para utilizarse en dominios o problemas específicos, a diferencia de los lenguajes de programación de propósito general. Tienen un mayor nivel de abstracción que los lenguajes base y expresan los conceptos de dominio específico en un nivel de representación más alto. Representan los elementos de plataformas de implementación.

DSL debe considerarse como un lenguaje de tamaño pequeño, muy centrado en resolver algunos problemas claramente identificables a los que debe enfrentarse un analista, arquitecto, responsable de pruebas o administrador del sistema.

Una buena forma de encontrar DSL candidatos consiste en identificar los patrones que utilizan los desarrolladores y encapsularlos en un lenguaje de modelado, o bien incluir como abstracciones los conceptos de un marco de software en un lenguaje de modelado que podrá después generar pequeñas cantidades de código que amplíen el marco. Estas técnicas permiten controlar la cantidad y complejidad del código generado, ofreciendo un valor real a los desarrolladores sin las complicaciones que caracterizan a los productos CASE.

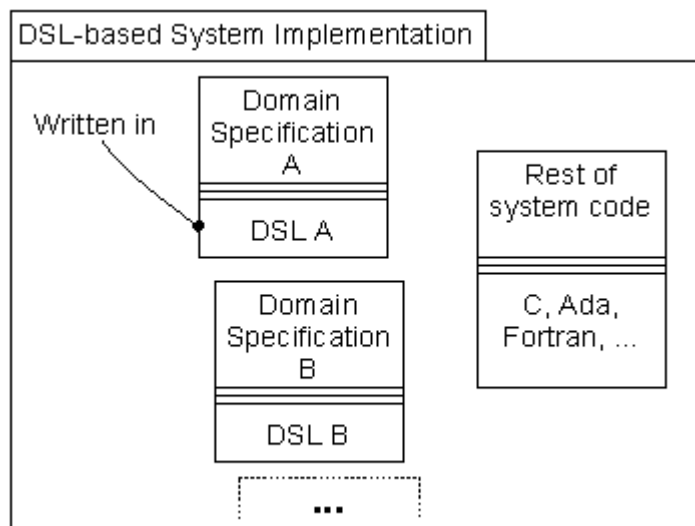


Figura 14: Diagrama UML de una arquitectura de un sistema basado en DSL

Se recomienda utilizar DSL y las herramientas basadas en DSL definidas de forma precisa para:

- Abstracciones precisas con las que generar código.
- Abstracciones precisas que asignan puntos de variación en los marcos y componentes.
- Asignaciones precisas entre lenguajes DSL.
- Dibujos conceptuales que tengan asignaciones que se puedan especificar de forma precisa en otros lenguajes DSL o en artefactos de código.

Existen cientos de DSLs hoy en día. Los ejemplos clásicos más conocidos son PIC, SCATTER, CHEM, LEX, YACC, and Make. Existen además otros como BNF, yacc, Spice, VHDL, Verilog, LATEX, SQL y HTML. Todos proporcionan lenguajes especializados para manipular conceptos dentro de su dominio. Por ejemplo en el caso de SQL se trabaja con tablas, registros y columnas, y en el caso de HTML con elementos, tablas y formas.

3.2. Metodología para el diseño de DSL

El desarrollo de un DSL se divide en las siguientes etapas [23]:

- **Análisis:** el objetivo del análisis es agrupar todo el conocimiento del dominio del problema. Para ello se siguen los siguientes pasos:
 1. Identificación del dominio del problema.
 2. Recolectar todo el conocimiento relevante del dominio.
 3. Agrupar este conocimiento en un conjunto de conceptos semánticos y operaciones sobre ellos.
 4. Diseñar un DSL que describa los usos en el dominio
- **Implementación:**
 1. Construir una librería que implemente las nociones semánticas.

- 2. Diseñar e implementar un compilador que traduzca los programas DSL en una secuencia de las llamadas de la librería.
- **Utilización:** Escribir el programa DSL para todos los usos deseados y compilarlo

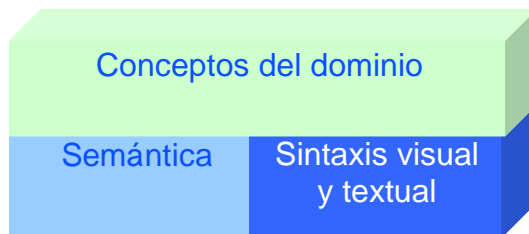


Figura 15: componentes de un DSL

3.3. Caso de estudio

Al igual que se ha hecho en el apartado 2.3 se va a realizar el estudio de un Lenguaje para entidades de negocios (ejemplo extraído de [24]). El lenguaje tiene que ser capaz de capturar la información de una aplicación bancaria.

El lenguaje de dominio específico en cuestión especificará los requisitos de negocio. Lo primero será crear un diagrama que nos muestre las entidades involucradas en los procesos de negocio.

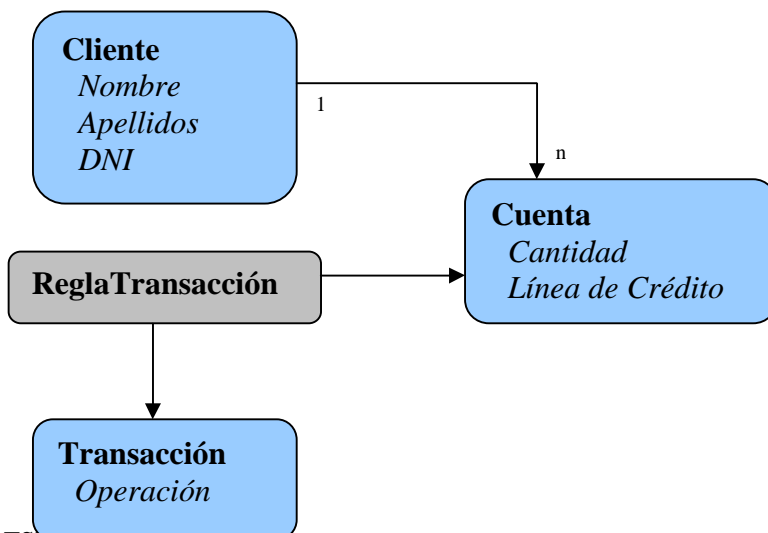
Para empezar se definirán las **entidades de negocio** implicadas con sus atributos y las **relaciones** entre dichas entidades. Es decir se diseña los conceptos del dominio.

“Cuenta bancaria” con atributos *cantidad* y *línea de crédito*.

“Cliente” con atributos *nombre*, *apellidos*, *dni*,...

“Transacción” con atributo *operación*.

Se definirán también **reglas de negocio** que contienen la lógica de negocios, describen cómo se utilizan las entidades de negocio. En este caso, la entidad de negocio cuenta tiene una entidad llamada transacción, para describir cómo se utilizan estas dos entidades se describe la regla de negocio “ReglaTransacción”.



El siguiente paso es construir la **notación** del dominio. En el caso del Visual Studio 2005 la **notación** que se utilizará para el DSL se describe en un fichero XML donde se indican los elementos utilizados para la creación del lenguaje de dominio específico.

Después de la notación se realiza la **transformación** de la notación al modelo del dominio. Esto incluye la transformación de las formas (*shapes*) en clases y de las líneas de conexión en relaciones.

A continuación se lleva a cabo la **generación del código** a partir del modelo del dominio y como consecuencia se obtiene el componente deseado.

Se puede resumir el diseño de un DSL en 3 pasos: creación del metamodelo, generación del código y creación del componente.

3.4. Herramientas

Interesan herramientas visuales para facilitar el diseño de los DSL. Existen herramientas para DSL que permiten diseñar, generar soluciones, especificar el diseño de los patrones, visualizar sistemas existentes, personalizar aplicaciones, etc. todo en una misma herramienta.

Los diseñadores de DSLs presentan muchas similitudes: todas tienen una superficie para dibujar, propiedades de Windows, barras de herramientas, explorador de modelos.

En el momento de construir un DSL, se necesitan herramientas para:

1. Definir el modelo: metamodelado
2. Definir la notación: conjunto de herramientas para entender las formas y complementos. Notación definida en un fichero XML.
3. Visualización definida del modelo del dominio.

3.4.1. Microsoft DSL Tools para Visual Studio 2005

Microsoft dispone de un CTP (Community Technology Preview) de las herramientas para DSL (Domain Specific Language). Con este *add-in* de Visual Studio 2005 se pueden definir nuevos lenguajes que sean específicos para resolver problemas de un dominio específico así como los diseñadores gráficos y los generadores de código correspondientes.

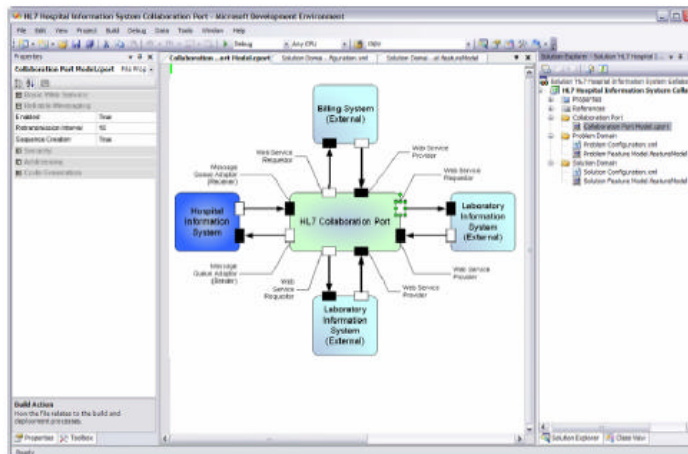


Figura 16: Microsoft Domain Specific Language (DSL) Tools para Visual Studio 2005

Las herramientas de Microsoft para lenguajes específicos de dominio permiten describir los conceptos relevantes de un dominio con un problema específico que sienten las bases para crear diseñadores visuales personalizados en Visual Studio 2005. Con estas herramientas se podrá crear propios diseñadores integrados en Visual Studio. Las herramientas ayudan a definir el DSL y generan el código del diseñador gráfico. El diseñador resultante utiliza la misma tecnología de modelado de las herramientas de modelado *Class Designer* y *Distributed System Designers* del Visual Studio 2005. El diseñador generado es una herramienta completa y un punto de partida del desarrollo del diseñador (*SDK, artifact generation*)

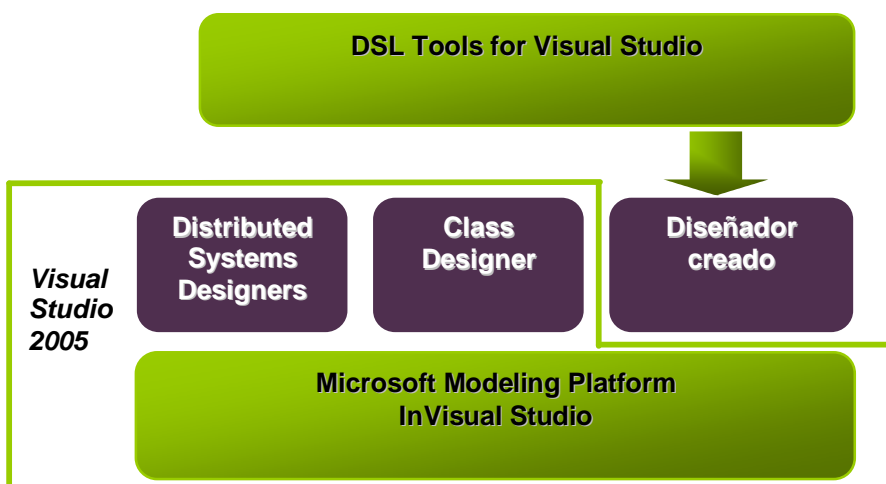


Figura 17: esquema DSL Tools for visual Studio

El DSL resultante, tiene que estar compuesto por las partes que se indican en la siguiente figura:

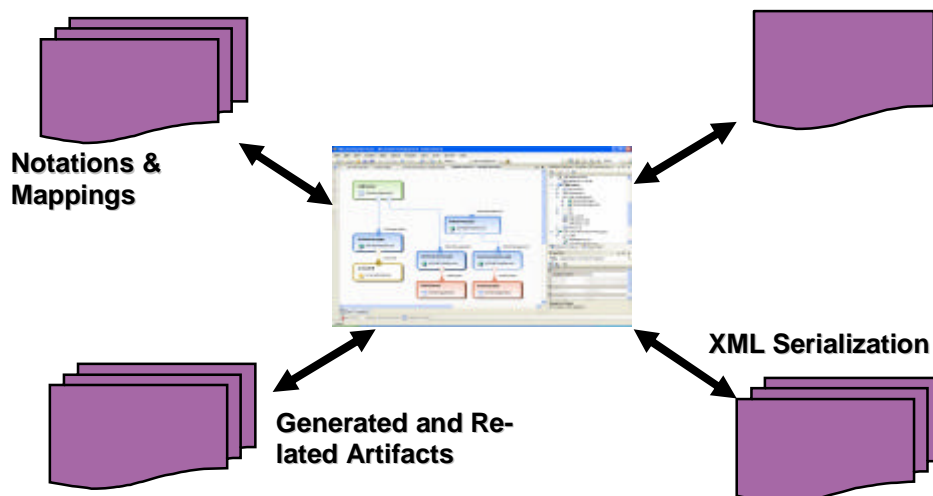


Figura 18: partes de un DSL. Sacado de [26]

La construcción de un diseñador (*designer*) para visual Studio se realiza mediante los siguientes pasos:

- Definición del modelo de dominio mediante clases, relaciones, roles, valores de propiedades...

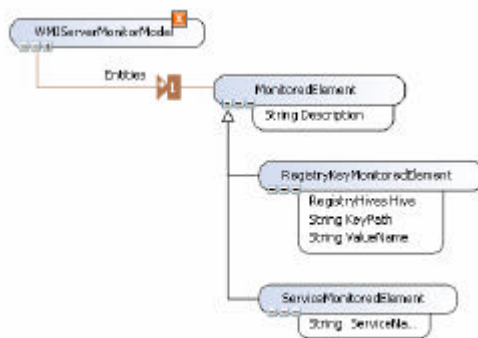


Figura 19: definición del modelo de dominio

- Definición de los elementos visuales, utilizando shapes, conectores, mapeo del modelo de dominio...

```
<geometryShape name="TaskShape" initialWidth="1.5" initialHeight="0.6"
    geometry="RoundedRectangle">
  <decorators>
    <shapeText name="Title" position="Center" defaultTextId="Title"/>
  </decorators>
  <fillColor color="gray" variability="User"/>
  <outlineColor color="black" variability="Fixed"/>
</geometryShape>
```

Figura 20: definición de elementos visuales

- Definición de los elementos de generación de texto

```
<#@ template inherits=
"Microsoft.VisualStudio.TextTemplating.VSHost.
ModelingTextTransformation"#>
<#@ output extension=".cs" #>
<#@ WMIEnvironment
processor="WMIInstanceViewerLanguageDirectiveProcessor"
requires="fileName='Empty.wmimondef' "
provides="WMIEnvironment=WMIEnvironment" #>
```

Figura 21: elementos de generación de texto

- Personalizar el diseñador añadiendo restricciones, validaciones, personalizando el comportamiento, serializando un XML personalizado...

4. COMPARATIVA

4. COMPARATIVA

El lenguaje utilizado a la hora de definir lenguajes de modelos varía dependiendo si se aplica un enfoque de desarrollo del software u otro. El enfoque de las Factorías del Software promueven el uso de Lenguajes de Dominio Específico, DSLs, mientras que MDA promueve el uso del UML, que es de propósito general. Esta es una de las diferencias más importantes entre los dos enfoques y la que, sin duda, es más polémica. Las Factorías de Software recomiendan la creación de lenguajes (de modelado) que permitan a los desarrolladores utilizar las primitivas más adecuadas para cada tipo de sistema. La OMG defiende que debe usarse UML, utilizando sus capacidades de extensión (profiles), cuando sea necesario, para expresar conceptos que no pueden representarse de forma directa en UML.

Como se observará, cada enfoque posee unas ventajas que resultan muy interesantes para la creación de métodos de desarrollo de software, pero ninguno de las dos resulta claramente mejor que la otra. A la hora de construir un método de desarrollo de software, la elección de un enfoque u otro dependerá de las características requeridas por el método:

El enfoque MDA es especialmente adecuado cuando la interoperabilidad con otras herramientas o el uso de herramientas existentes (que sigan los estándares de OMG) sea un factor clave.

El enfoque de las Factorías de Software es especialmente adecuado cuando existe la intención de construir una serie de sistemas similares y/o se va trabajar dentro de un dominio determinado.

4.1. Ventajas y desventajas de la alternativa MDA

Ventajas

- Define de manera clara la tecnología que recomienda utilizar para aplicar su enfoque: UML, MOF, QVT, etc. Esto facilita la tarea de construcción de un método a los desarrolladores que quieren aplicar este enfoque. Se puede contar con lenguajes ampliamente conocidos que, en gran parte, no necesitan de descripción y disponen de abundante documentación.
- Puesto que está más tiempo “*en el mercado*” MDA ha sido estudiado, discutido y aplicado. Este aspecto facilita la aceptación de un método por parte de la comunidad de la Ingeniería del Software.
- Ofrece un soporte de herramientas amplio que están empezando a madurar. Esto sucede por dos razones: la primera es que OMG está compuesto por muchas empresas que desean ofrecer productos que sigan las especificaciones de OMG. La segunda es que MDA es más conocido desde hace más tiempo. Por ejemplo, Eclipse es un entorno de desarrollo extensible, libre y promovido por IBM. Este entorno tiene una amplia comunidad que desarrolla extensiones para proporcionar funcionalidad muy diversa. Entre estas extensiones destaca EMF, una extensión para el metamodelo basado en el estándar MOF de OMG capaz de almacenar los modelos en formato XMI 2. También existe una extensión que proporciona una implementación del metamodelo de UML 2 utilizando EMF, disponiendo en estos casos de implementaciones libres de los estándares de OMG.

- Por una parte MDA mejora la productividad desde el punto en que los desarrolladores de PIM hacen menos trabajo porque pueden trabajar independientemente de los detalles de la plataforma que se vaya a utilizar.
- Por otro lado aumenta la productividad puesto que se puede pasar de código a PIM lo cual se ajusta mucho mejor a las necesidades del usuario final. Este incremento de productividad se consigue mediante el uso de herramientas que automaticen el paso de un PIM a un PSM
- La portabilidad se centra en el desarrollo de las PIMs que son plataformas independientes. El mismo PIM se puede transformar automáticamente en múltiples PSMs para diferentes plataformas, por tanto, todo lo que se especifique a nivel de PIM es portable.
- Mejora la interoperatividad puesto que múltiples PSM generados a partir de un mismo PIM pueden tener relaciones entre sí, estas relaciones se llaman en MDA puentes. Cuando los PSM están orientados a distintas plataformas no se pueden comunicar directamente. Es necesario transformar conceptos de una plataforma en los conceptos de la otra (En eso consiste la interoperatividad). En MDA no sólo se generan PSM sino que también los puentes necesarios entre ellos.
- El mantenimiento y la documentación presenta una notable mejora ya que hay que centrarse en el PIM, dado que el PIM se usará para transformarse en PSM que a su vez se transformará en código. Además el PIM no es abandonado una vez escrito, los cambios que se quieran hacer al sistema se harán en el PIM, que actualizará el PSM que a su vez actualizará el código. Y como consecuencia la documentación de alto nivel será consecuente con el código actual.

Desventajas

- Presenta carencias en transformación de modelos puesto que QVT aún está en fase de aprobación como estándar OMG y no tiene soporte de herramientas.
- No proporciona suficientes guías metodológicas, sólo se define la estrategia general para la transformación PIM → PSM
- No garantiza la separación de conceptos

4.2. Ventajas y desventajas de la alternativa DSL

Ventajas

- El principal promotor es Microsoft por lo que su posición dominante en el mercado puede conseguir que el enfoque de las Factorías de Software acabe implantándose en la industria. Pese a ello, se debe tener en cuenta que OMG también une a un número importante de multinacionales del desarrollo de software, por lo que esta ventaja puede ser relativa.
- Los lenguajes de aspectos de dominio específico fuerzan la separación de conceptos.
- La representación del dominio del problema es más intuitivo.
- DSLs permite soluciones para expresarse en el lenguaje y el nivel de abstracción del dominio del problema. Como consecuencia, los expertos del dominio pueden entender, validar, modificar y a menudo desarrollar programas DSL
- DSLs mejora la calidad, la productividad, la confiabilidad, la capacidad de mantenimiento, la portabilidad y la reutilidad.
- Permite la documentación del código automática.
- Los DSLs permiten la validación del nivel del dominio

- DSLs permite la validación en el nivel del dominio. Puesto que las construcciones del lenguaje son seguras también lo son las sentencias del mismo.

Desventajas

- Tiene un alcance y un diseño limitado ya que con los lenguajes de dominio específicos, pasar de una aplicación a otra, significa comenzar de cero nuevamente, otros lenguajes, otras restricciones, que pueden que sean similares o completamente distintas.
- Está poco tiempo en el mercado en comparación al enfoque de MDA.
- Al contrario que MDA, las Domain-Specific Language Tools que Microsoft propone para construir Factorías de Software todavía se encuentran en fase de desarrollo, por lo que no son muy robustas y no pueden aplicarse en entornos de producción industrial. No existen aún herramientas maduras o difundidas.
- Alto coste de diseño, implementación y mantenimiento.
- Alto coste en la educación a los usuarios.
- Pérdida potencial de eficacia en comparación con el software codificado a mano.

5. INTEGRACIÓN – DEFINIENDO DSLs MEDIANTE UML Y/O MOF

5. INTEGRACIÓN – DEFINIENDO DSLs MEDIANTE UML Y/O MOF

5.1. Introducción

Los dos puntos de vista de desarrollo controlados por modelos, MDA y DSL, son opuestos desde el punto de vista de sus promotores, OMG y Microsoft respectivamente. A pesar de ello las dos estrategias no son incompatibles, están relacionadas en algunos aspectos:

- DSL es un Lenguaje de Dominio Específico que por sus características se encuentra en el nivel M2 de metamodelado de la arquitectura multicapa del OMG. Es decir, parejo con UML.
- MDA utiliza DSLs para expresar cualquier modelo (o meta-modelo) definido en MOF. Este DSL en concreto es XML.
- PSM es un modelo de específico de la plataforma, por tanto de cierta manera es un DSL.

5.2. Definición de un DSL

Existen dos alternativas para definir DSLs. Por una parte se puede definir un DSL mediante un meta-metamodelo predefinido como MOF y por otro lado mediante la extensión de un metamodelo capaz como es UML. Para distinguir las dos alternativas veremos un ejemplo de cada estrategia sacada de [25].

Estrategia 1: Metamodelo para “*Feature Modeling*” basado en MOF

La idea es crear un DSL que modele un problema específico, es decir, obtener un metamodelo para tal fin. A partir del meta-metamodelo MOF se crean las instancias para el problema específico, especialización que se puede ver en la Figura 22, diferenciando los 2 niveles M3 y M2 de la arquitectura multicapa delOMG.

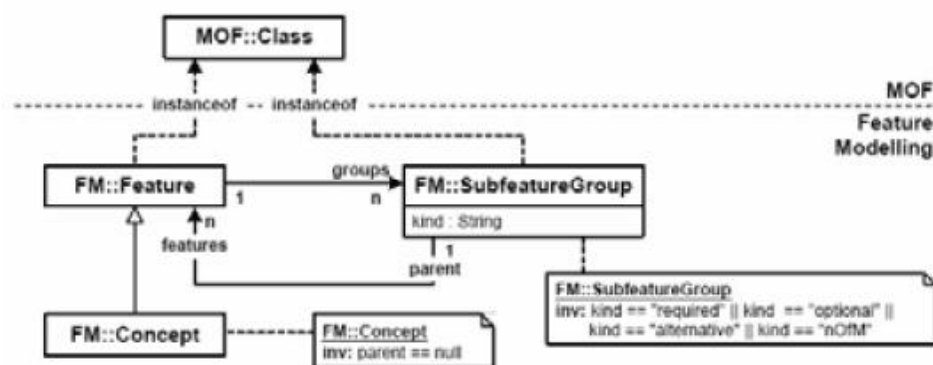


Figura 22: Metamodelo para “*Feature Modeling*”

La notación del dominio específico se obtiene a partir de la notación del metamodelo como se muestra en Figura 23 y Figura 24.

UML Notation:

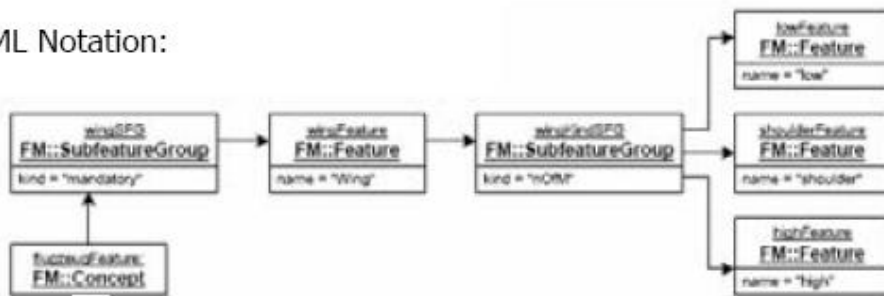


Figura 23: notación UML

Feature Model Notation:

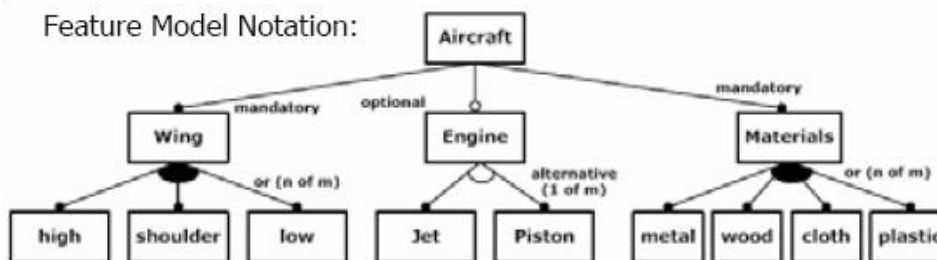


Figura 24: notación del modelo específico

Estrategia 2: Extensión UML

Esta segunda alternativa consiste en la extensión de un metamodelo UML en una clase específica del dominio del problema.

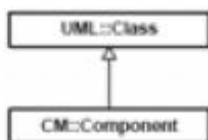


Figura 25: extensión de un metamodelo UML

El componente resultante es el DSL, un metamodelo capaz de definir cualquier modelo del problema específico a tratar.

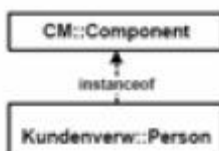


Figura 26: instancia del componente27: instancia del componente

REFERENCIAS

REFERENCIAS

1. Rodríguez, J., García, J., *Ingeniería de Modelos con MDA. Estudio comparativo de OptimalJ y ArcStyler*. 2004, España, Murcia, <http://www.dis.um.es/~jmolina/pfcs/proyecto-mda.pdf>.
2. Object Management Group. *MDA Guide Version 1.0.1*. 2003. <http://www.omg.org/docs/omg/03-06-01.pdf>.
3. Corredera, L.E., *Arquitectura dirigida por modelos para J2ME*. 2005. España, Murcia, http://personal.telefonica.terra.es/web/lencorredera/mda_j2me.pdf
4. García, P.F, Carrión, J., *Arquitectura Dirigida por Modelos*. 2005. España, Ciudad Real, <http://alarcos.inf-cr.uclm.es/doc/aplicabdd/DASBD-MDA.pdf>.
5. Caramazana, A., *Tecnologías MDA (Model Driven Architecture) para el desarrollo de software*. España, Madrid. <http://albertocc.tripod.com/pdf/TecnologiasMDA.pdf>.
6. Object Management Group 2003. *OMG Model-Driven Architecture Home Page*, <http://www.omg.org/mda/index.htm>.
7. Object Management Group. 2003. *MetaObjectFacility(MOF) Specification version 1.4*, <http://www.omg.org/mda>.
8. Object Management Group. 2003. *XML Metadata Interchange (XMI) Specification version 2.0*, <http://www.omg.org>.
9. Randall M. Hauch. 2002. *Enterprise Information Integration and the OMG's MDA and MOF*, <http://www.omg.org/mda>.
10. Object Management Group (OMG), *Revised submission for MOF 2.0 Query/View/Transformation rfp (ad/2002-04-10) (2005)*.
11. Object Management Group (OMG), *Common Warehouse Metamodel (CWM)*, <http://www.omg.org/cgi-bin/doc?ad/2001-02-01>.2000.
12. Object Management Group (OMG), *Ontology Definition Metamodel. Second Revised Submission to OMG*, http://codip.grci.com/odm/draft/submission_text/ODMPrelimSubAug04R1.pdf. 2003.
13. Portillo, J., Marcos, M., *Entorno Multidisciplinar para el desarrollo de Sistemas de Control Distribuido con Requisitos de Tiempo Real*, España, Bilbao. 2004.
14. García, F.,O., Piattini, M.,G., Ruiz, F., *FMESP: Marco de Trabajo Integrado para el Modelado y la Medición de los Procesos software*, España, Ciudad Real. 2004.

15. García, J., Rodríguez, J., Menárguez, M., Ortín, M.,J., Sánchez, J., *Un estudio comparativo de dos herramientas MDA: OptimalJ y ArcStyler*, España, Murcia. 2003,
<http://www.dsic.upv.es/workshops/dsdm04/files/09-Garcia.pdf>
16. Compuware, *Manuales de OptimalJ 3.0*. 2003.
<http://www.compuware.com/products/optimalj>
17. Jack Greenfield, Keith Short, Steve Cook and Stuart Kent. *Software Factories*. Wiley Publishing Inc., 2004
18. Muñoz, J., Pelechano, V., *MDA vs Factorías de Software*, España, Valencia.
<http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-157/paper01.pdf>
19. <http://www.compuware.es/pressroom/news/2005/2005063001.htm>
20. Compuware. *Model Driven Architecture in OptimalJ*. 2003.
<http://javacentral.compuware.com/demos/mda.html>
21. Interative Object, *ArcStyler 4.0.90*. 2004.
<http://www.arcstyler.com>
22. Java Boutique,
<http://javaboutique.internet.com/articles/JMI/index-3.html>
23. Deursen, A., V., Klint, P., Visser, J., *Domain-Specific Languages: An Annotated Bibliography*
<http://homepages.cwi.nl/~arie/papers/dslbib/>
24. MSDN Virtual Labs, *Build a Domain Specific Language with DSL Tools*,
<http://www.aspfree.com/c/a/BrainDump/Build-a-Domain-Specific-Language-with-DSL-Tools/>
25. Falb, J., Radinger, W., *IT Vertiefung*, Institute of Computer Technology
26. Matthewman, A., “*Domain Specific Languages Workshop. Microsoft EMEA*”

ACRÓNIMOS

ACRÓNIMOS

CASE	<i>Computer-Aided Software Engineering</i>
CIM	<i>Computation Independent Model</i>
CORBA	<i>Common Object Request Broker Architecture</i>
CWM	<i>Common Warehouse Metamodel</i>
DSL	<i>Domain Specific Language</i>
DTD	<i>Document Type Definitions</i>
EJB	<i>Enterprise Java Bean</i>
GMT	<i>Generative Model Transformer</i>
J2EE	<i>Java 2 Platform, Enterprise Edition</i>
JCP	<i>Java Community Process</i>
JMI	<i>Java Metadata Interface</i>
MDA	<i>Model Driven Architecture</i>
MLT	<i>Model Transformation Language</i>
MOF	<i>Meta Object Facility</i>
OCL	<i>Object Constraint Language</i>
ODM	<i>Ontology Definition Metamodel</i>
OLAP	<i>Online Analytical Processing</i>
OMG	<i>Object Management Group</i>
PIM	<i>Platform Independent Model</i>
PSM	<i>Platform Specific Models</i>
QVT	<i>Query, Views and Transformation</i>
UML	<i>Unified Modeling Language</i>
XMI	<i>XML Metadata Interchange</i>
XSD	<i>XML Schema Definition</i>