

Una revisión de herramientas MDA

Verónica Bollati

Juan M. Vara

Belén Vela

Esperanza Marcos

Grupo Kybele

Universidad Rey Juan Carlos

C/ Tulipán S/N, 28933 - Móstoles (MADRID)

{veronica.bollati, juanmanuel.vara, belen.vela, esperanza.marcos}@urjc.es

Resumen

En la actualidad la propuesta MDA está tomando cada vez más fuerza en el desarrollo de software. En los últimos años han ido apareciendo numerosas herramientas que dan soporte a esta propuesta, facilitando la tarea del desarrollador de sistemas de información (SI). Siguiendo las líneas de MDA nace MIDAS, que es una arquitectura de modelos para el desarrollo de SI, que propone meta-modelos basados en perfiles UML. En este artículo se presenta un estudio comparativo sobre dos herramientas MDA (AndroMDA y ArgoUML) y un entorno de desarrollo (Eclipse), con el objetivo de determinar, la necesidad de desarrollo de una nueva herramienta o la adaptación de alguna de las existentes, para dar soporte a MIDAS. Para realizar dicha comparativa se ha realizado en primer lugar una caracterización de herramientas MDA, es decir, se han determinado un conjunto de características que, en nuestra opinión, son obligatorias o deseables en una herramienta MDA. Posteriormente se han analizado dichas características en las herramientas seleccionadas, mediante su aplicación a un caso de estudio, permitiendo de esta forma evaluar la funcionalidad de las mismas. Las conclusiones obtenidas en este estudio permitirán decidir la tecnología a usar y la arquitectura de la herramienta MDA que dará soporte a MIDAS.

Palabras claves: MDA, herramientas MDA, MIDAS, meta-modelo, UML, perfil

1. Introducción

MDA [17], propuesto por el grupo OMG, está tomando cada vez más fuerza en el desarrollo de software. Se trata de un marco de trabajo para el desarrollo de software, cuya principal característica es la definición de modelos como

elementos de primer orden en el diseño, desarrollo e implementación del software y las transformaciones entre los diferentes modelos mediante la definición de *mappings*. MDA considera diferentes tipos de modelos, en función del nivel de abstracción de los mismos: los requisitos del sistema son detallados en el Modelo Independiente de Computación (*Computation Independent Model*, CIM); en los Modelos Independientes de Plataforma (*Platform Independent Model*, PIM) se representa la funcionalidad del sistema sin considerar la plataforma final y los Modelos Específicos de Plataforma (*Platform Specific Model*, PSM) se obtienen de combinar las especificaciones contenidas en el PIM con los detalles de la plataforma elegida. A partir de los diferentes PSMs se pueden generar automáticamente distintas implementaciones (código) del mismo sistema.

En los últimos años han aparecido numerosas herramientas MDA que permiten en mayor o menor grado automatizar las transformaciones y que generan, en algunos casos, (semi-) automáticamente código para distintas plataformas.

MIDAS [11][22] es una arquitectura de modelos para el desarrollo de Sistemas de Información (SI) basado en MDA, que propone modelar los sistemas de acuerdo a dos dimensiones ortogonales: el grado de dependencia de la plataforma (CIM, PIM y PSM) y los aspectos, en los que comúnmente se estructura el SI (Web) (contenido, hipertexto y comportamiento).

Con el fin de dar soporte a los modelos y las transformaciones definidas en MIDAS se pretende desarrollar una herramienta MDA. Para ello se realizará una revisión de algunas de las herramientas MDA existentes para determinar:

- Las características *obligatorias* y las *deseables* en una herramienta MDA.

- Las características que recogen la mayoría de las herramientas.
- Y las principales carencias que presentan las mismas.

Existen otros trabajos relacionados con la evaluación de herramientas MDA, como [6][8][16][20]. Tras su análisis, en el presente nos hemos basado en [6] y [20]. En [6] se realiza un estudio comparativo de dos herramientas MDA comerciales (*OptimalJ* y *ArcStyler*). En dicho estudio se han evaluado las herramientas realizando un análisis de propiedades extraídas de la especificación de MDA y desarrollando para ello un caso de estudio, con el fin de detectar cuál de las dos se ajusta mejor a MDA.

En [20] se presenta un marco de referencia para la evaluación de herramientas MDA y se evalúan diez herramientas de estas, logrando realizar una taxonomía de las mismas. Para ello, se propone, una clasificación de herramientas, así como los criterios de evaluación. Estos últimos se basan en características de MDA, criterios de calidad, de entorno y generales.

En el trabajo que aquí se presenta se ha realizado un compendio de las características propuestas en los dos trabajos anteriormente citados, seleccionando las características que se consideran obligatorias o deseables que cumpla una herramienta MDA. Además se han introducido nuevas características a ser evaluadas. Para la selección de las herramientas se ha considerado, por un lado, la propuesta de clasificación de herramientas del segundo trabajo [20] y por otro lado, la conveniencia de abarcar tanto herramientas comerciales como herramientas de libre distribución; teniendo en cuenta en todo momento que es necesario que las herramientas seleccionadas sean extensibles, ya que esto nos permitirá dar soporte a MIDAS. El objetivo principal de este trabajo es evaluar las herramientas para verificar si es posible adaptar alguna de ellas o, si por el contrario, es necesario desarrollar una nueva herramienta para dar soporte a MIDAS.

Este artículo se estructura de la siguiente manera: en el apartado 2 se realiza una caracterización de las herramientas MDA; en el apartado 3 se presenta el caso de estudio que se utilizará para la evaluación; en el siguiente apartado se presenta una breve descripción de las herramientas seleccionadas y el resultado de la implementación del caso de estudio con las

mismas; la sección 5 es una discusión que incluye una comparativa de las herramientas analizadas; finalmente, en el apartado 6 se recogen las principales conclusiones y se plantean trabajos futuros.

2. Caracterización de herramientas MDA

La evolución de las herramientas de apoyo al proceso de desarrollo de SI está ligada a la evolución de la Ingeniería de Software como disciplina. Las primeras herramientas fueron los editores y procesadores de texto, así como las herramientas de dibujo que incorporaban notaciones gráficas. A partir de la consolidación de las metodologías de desarrollo, integrando diferentes técnicas, comenzaron a aparecer las primeras herramientas CASE (*Computer-Aided Software Engineering*), seguidas posteriormente por las herramientas extensibles, herramientas de transformación y herramientas generadoras de código. Por último, se pueden citar las *herramientas MDA*, en las que se centra el presente artículo. Su principal aportación es dar soporte a la transformación de modelos, permitiendo realizar, en la mayoría de los casos, transformaciones verticales y horizontales [21]. Las transformaciones se deben realizar de forma (semi-)automática a través de la definición de reglas de transformación o *mappings*.

Para la selección de las características que se han estudiado en las diferentes herramientas, se han tenido en cuenta las propuestas en los trabajos citados anteriormente, completándose los mismos con otras características extraídas de la especificación de MDA. Éstas se han agrupado en funcionales, técnicas y de calidad, indicándose para cada una de las características el grado de cumplimiento que debería tener la herramienta de acuerdo a las necesidades de MIDAS: obligatoria (*Obl*) o deseable (*Des*).

2.1. Características funcionales

En esta categoría se agrupan todos aquellos requisitos que se correspondan con los aspectos funcionales que debería cumplir una herramienta MDA:

Niveles que cubre (Obl): qué niveles de MDA cubre (CIM, PIM, PSM).

Grado de generación de código (Obl): en qué medida permite la transformación de los PSMs a código. Por ello se debe evaluar si las herramientas generan código y si dicho código puede ser implementado de forma directa o si es necesario realizar modificaciones en el mismo. Además se debe verificar si la generación de código se realiza desde el nivel PSM o desde el nivel PIM.

Transformaciones (Obl): verificar el grado de automatización de las transformaciones.

Interacción con el usuario (Des): verificar el grado de participación del usuario en el proceso de transformación.

Tipo de transformaciones (Obl): evaluar, si permite realizar transformaciones verticales y/u horizontales. Se entiende por transformaciones verticales, las que se realizan entre los diferentes niveles de abstracción (CIM, PIM y PSM) y, por horizontales, las que se realizan entre los diferentes modelos de un mismo nivel de abstracción.

2.2. Características técnicas

En esta categoría se agrupan todos aquellos requisitos que se correspondan con las características técnicas que debería cumplir una herramienta MDA:

Lenguaje de almacenamiento y gestión de modelos (Obl): para realizar las transformaciones de los modelos entre los distintos niveles de abstracción es necesario un lenguaje que permita el almacenamiento y la gestión de estos modelos de forma que éstos se puedan intercambiar entre los distintos niveles (CIM, PIM y PSM).

Plataformas y tecnologías soportadas (Des): verificar qué plataformas de desarrollo soporta la herramienta.

Ámbito de aplicación (Des): se debe determinar el tipo de desarrollo en el que se centran, en particular para el presente artículo interesa determinar si las herramientas están orientadas hacia el desarrollo de SI Web o hacia desarrollos orientados a servicios, o ambos.

2.3. Características de calidad

En esta categoría se agrupan todos aquellos requisitos que se correspondan con las

características de calidad que debería cumplir una herramienta MDA:

Uso de Estándares (Des): es importante determinar si las herramientas estudiadas hacen uso de estándares como UML, XML y MOF, para la definición de los modelos. Además se tendrá en cuenta si permite definir o personalizar perfiles propios del usuario para UML.

Extensibilidad (Obl): capacidad que tiene la herramienta de adaptarse a nuevos requisitos.

Usabilidad (Des): determinar la medida en el que la herramienta puede ser usada por los usuarios para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso especificado.

Interoperabilidad entre herramientas (Des): ver en qué grado la herramienta se puede integrar con otras herramientas con el objetivo de obtener nuevas funcionalidades.

3. Caso de estudio

Con el fin de evaluar las distintas herramientas seleccionadas, se ha usado como caso de estudio el perfil UML para el meta-modelo Objeto-Relacional (OR) [12] [22] del estándar SQL:2003 [7]. Como se ha dicho anteriormente, MIDAS contempla varios aspectos: contenido, hipertexto y comportamiento. En el aspecto de contenido, que es en el que se centra el presente artículo, propone dos meta-modelos para la implementación del nivel PSM, el meta-modelo del *XML Schema* y el meta-modelo OR. Este último es el que se ha utilizado para el desarrollo del presente caso de estudio.

El perfil OR permitirá la especificación de modelos de datos OR, mediante extensiones de UML, para cada una de las herramientas elegidas. Posteriormente, para realizar la validación de este meta-modelo, se creará una base de datos para una empresa de arquitectura utilizando para ello, el perfil OR previamente creado.

Con este caso de estudio se pretende determinar el grado de cumplimiento de las herramientas para cada una de las características detalladas anteriormente.

3.1. Meta-modelo objeto-relacional

El meta-modelo OR para el estándar SQL:2003, que aquí se presenta, recoge sólo las extensiones para objetos del estándar, no se incluyen los artefactos correspondientes al meta-modelo relacional. Los tipos de datos se clasifican en *Array*, *Multiset*, *Row*, tipo referencia y tipos estructurados. Una tabla tipada está basada en un

tipo estructurado. Un tipo estructurado puede contener atributos y métodos. Una descripción detallada de dicho meta-modelo puede encontrarse en [21].

En la Figura 1 se muestra el perfil UML [4] para el meta-modelo OR del estándar SQL:2003. Dicho perfil permitirá el modelado de un esquema OR (nivel PSM de la arquitectura de MIDAS) usando un diagrama de clases extendido.

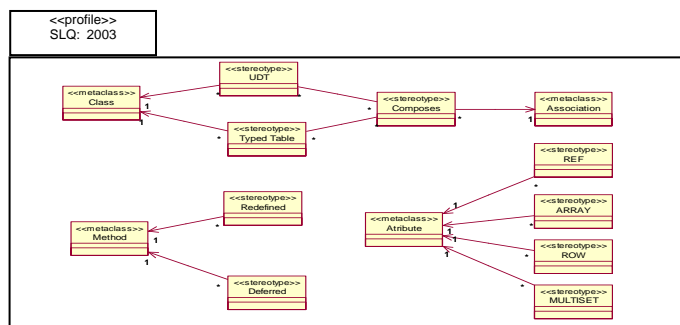


Figura 1. Perfil OR SQL 2003

4. Evaluación de herramientas

Como se ha mencionado anteriormente, en la actualidad existe una amplia gama de herramientas que permiten dar soporte a MDA. Se han seleccionado alguna de ellas para realizar una primera evaluación de las mismas.

Para realizar la evaluación se han llevado a cabo las siguientes tareas:

1. Instalación de las distintas herramientas.
2. Instalación de herramientas y componentes adicionales necesarios para la ejecución de la misma.
3. Implementación del caso de estudio seleccionado:

- a. Definición del perfil UML para BD OR en la herramienta.
- b. Validación del mismo usando una base de datos del OR (un estudio de arquitectura). En cada una de las herramientas se ha comprobado además el código generado en la definición de las clases, atributos y relaciones.

A continuación, se explica brevemente el comportamiento de las herramientas seleccionadas en el desarrollo del caso de estudio propuesto. En

la Tabla 1 de la sección 5, se resumirán las características analizadas para cada una de ellas.

4.1. Eclipse

Eclipse [5] nace como un entorno de desarrollo para JAVA y se ha convertido en una plataforma que permite el desarrollo y la integración de herramientas de desarrollo. Si bien no puede ser considerada como una herramienta MDA, es una herramienta con la que se pueden generar herramientas MDA. Fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas *VisualAge*. En el año 2001 se forma un consorcio para el desarrollo futuro de Eclipse como código abierto. Finalmente, en el año 2003 este consorcio se independiza de IBM convirtiéndose en la fundación Eclipse.

Se ha incluido Eclipse en este estudio, porque puede ser utilizado, no sólo en el desarrollo de productos software, sino también en el desarrollo de herramientas que permitan construir productos software [5]. Se puede decir también que Eclipse es un marco de trabajo para el modelado y la integración de datos permitiendo almacenar meta-modelos y meta-datos.

La principal ventaja de Eclipse radica en que su arquitectura está diseñada de forma que la mayoría de la funcionalidad proporcionada está localizada en *plug-ins* o en un conjunto de *plug-ins* relacionados. Esto hace a Eclipse una herramienta extensible.

Del conjunto de *plug-ins* de Eclipse el presente artículo se centra en *Eclipse Modeling Framework* (EMF) [15]. EMF es una herramienta para la generación de código a partir de modelos definidos por el usuario. A través de ella se pueden generar *plug-ins* que proporcionen editores de modelos. Caben destacar dos *plug-ins* en concreto, el *plug-in* UML2 permite definir estereotipos de UML y el *plug-in Graphical Modeling Framework* (GMF) basado en EMF, que permite generar editores gráficos de modelos.

Aplicación del Caso de Estudio

Para el desarrollo del caso de estudio con Eclipse se ha realizado lo siguiente:

- Se ha utilizado la versión 3.2.2 de Eclipse con el *plug-in* de EMF versión 2.3.0.

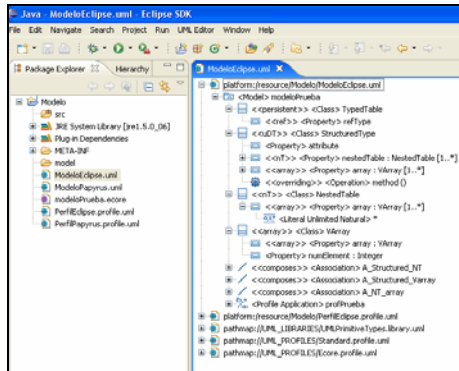


Figura 2. Meta-modelo de SQL: 2003 definido con UML2.

- Para definir el meta-modelo OR en la herramienta ha utilizado el *plug-in* UML2, para poder representar el perfil, ya que el mismo se define utilizando estereotipos. Una vez definido el perfil con UML2 (ver Figura 2) se ha convertido a un modelo *Ecore* para así poder generar el editor de meta-modelos con EMF. Como EMF no admite la representación de estereotipos en la conversión de UML2 a *Ecore* éstos no han podido ser contemplados. También se ha utilizado el *plug-in* de GMF para generar de forma gráfica editores de modelos.
- Se ha generado el editor de modelos sin los estereotipos (ver Figura 3). Para corroborar su funcionamiento se ha creado el diagrama de clases correspondiente con la BD del estudio de arquitectura, obteniéndose las clases, atributos y relaciones sintácticamente acordes al meta-modelo OR definido.

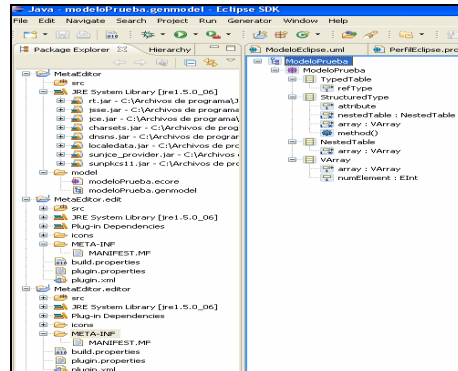


Figura 3. Meta-editor Ecore del perfil SQL:2003

Evaluación de características

Conjuntamente con el desarrollo del caso de estudio se han evaluado cada una de las características seleccionadas en el apartado 2

Niveles que cubre: Utilizando EMF se pueden definir todos los niveles de abstracción definidos en MIDAS, ya que *Ecore* es una implementación de eMOF (*Essential MOF*), por lo que nos permite definir CIMs, PIMs y PSMs

Grado de generación de código: El lenguaje en el que se genera código por defecto, con Eclipse es JAVA, si bien existen otros *plug-ins*

que permiten generar código en otros lenguajes. El código generado puede ser verificado y, en caso necesario, se pueden redefinir los modelos y generar el código de nuevo.

Transformaciones: Se pueden obtener diferentes PSMs a partir del mismo PIM de forma automática. Estos PSMs se pueden pasar al generador de código y se puede generar el código automáticamente [2]. Para realizar las transformaciones entre modelos se debe utilizar *plug-ins* que implementen dichas transformaciones. Para definir las se puede utilizar

cualquier lenguaje que entienda *Ecore*, por ejemplo ATL [3] y QVT [14].

Grado de interacción con el usuario: el usuario participa activamente en las etapas de definición de los meta-modelos y los meta-datos. En las etapas de definición de modelos y datos disminuye la participación del usuario ya que permite tomar como base los meta-modelos definidos con anterioridad, trasladándose de esta manera las características definidas.

Tipo de Transformaciones: únicamente se pueden realizar transformaciones verticales (de PIM a PSM y de PSM a código). Para realizar las transformaciones horizontales se podrían definir nuevos *plug-ins* que las implementen.

Lenguaje de almacenamiento y gestión de modelos: se almacenan en formato XMI. Esto permite definir los diferentes modelos en cualquier herramienta, por ejemplo *Rational Rose*, almacenarlo con formato XMI y exportarlo a EMF.

Plataformas y tecnologías soportadas: soporta la plataforma de JAVA.

Ámbito de aplicación: es una herramienta orientada al desarrollo de sistemas de propósito general. En particular, como hemos dicho anteriormente, en el presente trabajo interesan las facilidades que brinda Eclipse para el desarrollo de software orientado a servicios.

Uso de estándares: utiliza XMI, UML y MOF.

Extensibilidad: se puede extender fácilmente por medio de *plug-ins*, ya que la ventaja principal de Eclipse reside en que cada *plug-in* puede definir puntos de extensión a los que otros *plug-ins* se pueden conectar, permitiendo de esta manera incrementar la funcionalidad. Así el usuario podría crear el *plug-in* necesario para cubrir nuevos requerimientos.

Usabilidad: es una herramienta amigable. Permite definir los meta-modelos y meta-datos utilizando otras herramientas, por lo que se debe tener en cuenta que el usuario deberá tener conocimiento de las mismas.

Interoperabilidad entre herramientas: se puede adaptar a otras herramientas, por medio de *plug-ins* existentes o desarrollar nuevos *plug-ins*.

4.2. AndroMDA

AndroMDA[9] [10] es una herramienta de generación de código que toma modelos UML en

formato XMI como entrada y, genera código como salida, en cualquier lenguaje de programación. Nace en el año 2002 como una iniciativa de Matthias Bohlen [1]. En el año 2003 adquiere el nombre de AndroMDA debido a que toma las bases del paradigma MDA.

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

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

Aplicación del Caso de Estudio

Para el desarrollo del caso de estudio con AndroMDA se ha realizado lo siguiente:

- El funcionamiento AndroMDA depende del lenguaje con el que se vaya implementar el SI. Para el presente artículo se ha seleccionado como lenguaje de implementación *Visual Studio 2005*, por ello se ha instalado el *cartridge* Android/VS. Además ha sido necesaria la instalación de herramientas adicionales como *MagicDraw 9.5* para realizar los diferentes diagramas y *Apache's Maven* [23] que permite el acceso a las librerías necesarias de AndroMDA de forma transparente para el usuario.
- Para definir el meta-modelo OR se ha utilizado *Magic Draw*, y posteriormente se ha exportado a .Net. En este punto se presentó el mismo inconveniente que se explico en el caso de Eclipse, AndroMDA toma como meta-modelo el meta-modelo de UML. Como el meta-modelo OR utiliza extensiones UML usando estereotipos, que si bien pueden ser representados con la herramienta *Magic Draw*, cuando se exporta a .Net se eliminan los estereotipos. Esto ocurre porque en el *cartridge* de AndroMDA para .Net no están definidos los estereotipos ni la forma de realizar la transformación de los mismos a .Net. Como se ha dicho anteriormente los meta-modelos y las reglas de transformación se encuentran definidas en los *cartridges*, por lo cual para poder desarrollar modelos de clases acordes al

meta-modelo OR se debería definir el *cartridge* correspondiente con el meta-modelo y el lenguaje de desarrollo que se desee utilizar.

Evaluación de características

Conjuntamente con el desarrollo del caso de estudio se han evaluado cada una de las características seleccionadas en el apartado 2:

Niveles que cubre: implementa los niveles PIM y PSM, permitiendo realizar transformaciones desde un PIM a varios PSMs. Las transformaciones entre los diferentes modelos se realizan de forma automática.

Grado de generación de código: utiliza la tecnología de *cartridge*, que le permite obtener modelos en diferentes plataformas. A partir de los PSMs se puede generar código en el lenguaje de programación que se desee siempre que se tenga el *cartridge* correspondiente. Además se debe tener en cuenta que AndroMDA también permite generar código desde un PIM directamente.

Transformaciones: una vez que se tienen definidos los *cartridges* correctamente, las transformaciones entre los modelos de diferentes niveles son automáticas. El lenguaje utilizado para definir los *cartridges* en AndroMDA es JAVA.

Grado de interacción con el usuario: se puede decir que la interacción del usuario es alta, ya que debe participar activamente en todas las etapas del desarrollo.

Tipos de transformaciones: únicamente se pueden realizar transformaciones verticales (de PIM a PSM y de PSM a código). Para realizar las transformaciones horizontales se podría, o bien modificar un *cartridge* existente, o definir nuevos *cartridges* que los implementen.

Lenguaje de almacenamiento y gestión de modelos: los modelos generados son almacenados en XMI, permitiendo realizar el intercambio de modelos entre los diferentes niveles [9].

Plataformas y tecnologías soportadas: utiliza herramientas adicionales como *Apache's Maven*, que simplifica su uso, ya que permite a los *cartridges* de AndroMDA para los diferentes lenguajes. Además se pueden utilizar herramientas CASE que permitan generar los diagramas en formato XMI. Utiliza *Nhibernate* para asegurar la persistencia de los objetos. También posee *cartridges* que permiten generar código en plataformas como Java, .Net o PHP.

Ámbito de aplicación: está pensada para el desarrollo de software orientado a servicios y SI Web.

Uso de estándares: es *MOF-complaint*, permitiendo así soportar estándares como UML y XMI.

Extensibilidad: se extiende creando nuevos *cartridges* que soporten nuevos requisitos.

Usabilidad: en cuanto a la facilidad de uso, se debe considerar que AndroMDA se implementa a través de *cartridges* que se pueden integrar con otras herramientas de desarrollo, por ejemplo Visual .Net o JAVA, por ello es necesario que el usuario posea los conocimientos necesarios en el lenguaje seleccionado.

Interoperabilidad entre herramientas: acepta como entrada los PIMs generados con otras herramientas, como por ejemplo ArgoUML. Además, como se ha dicho anteriormente, tiene *cartridges* definidos para la mayoría de los lenguajes de programación.

4.3. ArgoUML

ArgoUML [19] es una herramienta para el desarrollo de SI. Nace a finales de la década de los 90 como una iniciativa del grupo liderado por Nora Koch. Fue concebido como una herramienta CASE para realizar el análisis y el diseño en el desarrollo de sistemas orientados a objeto y evolucionó hacia los SI Web.

Se ha seleccionado ArgoUML para este estudio, ya que es una herramienta que ha nacido como propuesta de la comunidad de desarrolladores *Open Source*. Está desarrollado utilizando el lenguaje JAVA, permitiendo así que funcione en cualquier plataforma que tenga instalada JAVA2.

Aplicación del Caso de Estudio

Para el desarrollo del caso de estudio con ArgoUML se ha realizado lo siguiente:

- Se ha utilizado la versión 0.24 de ArgoUML. No se requiere ninguna herramienta adicional para el desarrollo del caso de estudio, los diagramas de clases se generan utilizando ArgoUML.
- Se ha comenzado definiendo el perfil OR utilizando la herramienta. En este caso también se ha tenido el mismo inconveniente que en las herramientas anteriores: no se han podido definir los estereotipos del perfil. Como

ArgoUML es una herramienta *OpenSource* se puede agregar la funcionalidad correspondiente para poder definir estos estereotipos con la herramienta e implementar las transformaciones correspondientes.

Evaluación de características

Conjuntamente con el desarrollo del caso de estudio se han evaluado cada una de las características seleccionadas en el apartado 2:

Niveles que cubre: cubre todas las etapas del ciclo de vida de un sistema, por lo que se pueden definir modelos para el nivel CIM, PIM y PSM y posteriormente su paso a código.

Grado de generación de código: genera código en el lenguaje JAVA por defecto, aunque existen módulos auxiliares que permiten generar código en lenguajes como C#, C++, PHP4 y PHP5.

Transformaciones: brinda soporte para realizar transformaciones de PIM a PSM y de PSM a código. Las transformaciones entre los diferentes modelos se realizan de forma automática de acuerdo a lo definido por el usuario. Como se ha dicho anteriormente, ArgoUML está definido en JAVA, por lo que las transformaciones deben definirse utilizando ese lenguaje.

Grado de interacción con el usuario el usuario debe participar activamente en la definición de los diferentes modelos.

Tipo de transformaciones: únicamente se pueden realizar transformaciones verticales (de PIM a PSM y PSM a código). Para realizar transformaciones horizontales se deberían definir las transformaciones, agregándolas como funcionalidad a la herramienta.

Lenguaje de almacenamiento y gestión de modelos: la información de los diferentes modelos se almacena en formato XMI y los gráficos en PGML (*Precision Graphics Markup Language*) basado en XML.

Plataformas y tecnologías soportadas: para su funcionamiento no necesita ninguna herramienta adicional, utiliza la plataforma de JAVA.

Ámbito de aplicación: ArgoUML está pensada para el desarrollo de SI, concretamente en los SI Web. Posteriormente, con la evolución de los SI Web se adaptó al desarrollo de software orientado a servicios.

Uso de estándares: el repositorio de la herramienta está implementado con JMI (*Interface Metadata* de JAVA) y basado en

MOF, tomando la especificación de UML 1.4. También tiene soporte para OCL y XMI.

Extensibilidad: Es un proyecto *Open Source*, al que se le puede seguir agregando funcionalidad.

Usabilidad: Tiene una interfaz gráfica muy fácil de entender y manejar, es muy intuitiva. A través de la herramienta se pueden realizar todas las etapas del desarrollo.

Interoperabilidad entre herramientas: los PIMs realizados con ArgoUML pueden ser utilizados como entrada para AndroMDA como se ha indicado en el apartado anterior.

5. Discusión

Tras haber desarrollado el caso de estudio con las diferentes herramientas y haber evaluado las características definidas, se ha llegado a las siguientes conclusiones:

- Al utilizar Eclipse con el *plug-in* EMF se pueden generar editores de meta-modelos de forma fácil si éstos se basan en UML. Sin embargo, no soporta la definición de estereotipos para definir extensiones UML. Teniendo en cuenta que la metodología MIDAS propone el uso de perfiles UML para la mayoría de sus meta-modelos, es necesario utilizar el *plug-in* UML2 para poder definir los meta-modelos de MIDAS. Además, con el uso del *plug-in* GMF se pueden definir los meta-modelos y los modelos correspondientes facilitando así el modelado al usuario. Se debe tener en cuenta que para definir las transformaciones para los diferentes meta-modelos de MIDAS será necesario realizar *plug-ins* que las implementen utilizando para ello el lenguaje que el usuario desee como se ha explicado anteriormente.
- AndroMDA tiene definidos los *cartridges* para la mayoría de los lenguajes de desarrollo, por lo que la implementación en distintas plataformas es muy fácil de realizar. Los *cartridges* se basan en el meta-modelo de UML. Si se desea realizar modelos basados en perfiles de UML que utilicen estereotipos, se deben modificar los *cartridges* o crear nuevos. Hay que tener en cuenta que se deberían crear *cartridges* para cada uno de los lenguajes de desarrollo con los que se desee trabajar y para cada meta-modelo que se desee introducir. La modificación o creación de nuevos *cartridges* no es trivial, se

debe tener conocimiento del lenguaje para el cual se está desarrollando el *cartridge* y de la manera de definir las transformaciones en el mismo.

- ArgoUML se basa en el meta-modelo de UML. Por tanto, al igual que ocurre con AndroMDA, si los modelos que se desean generar son acordes a UML, éstos son fáciles de implementar. Pero si se desea realizar modelos basados en perfiles UML, se debe codificar la funcionalidad necesaria que permita recoger los estereotipos. Si bien ArgoUML únicamente genera código JAVA, los modelos generados con esta herramienta pueden servir de entrada a otras herramientas, como por ejemplo, AndroMDA.

De las tres opciones evaluadas hasta el momento se considera que Eclipse es la que mejor se adapta a las necesidades de MIDAS como

arquitectura de modelos, ya que combinando los *plug-ins* EMF, GMF y UML2 se podrá generar una herramienta que se adapte fácilmente a la arquitectura de modelos de MIDAS. Sin embargo hay que tener en cuenta que se debe realizar *plug-ins* para implementar las transformaciones entre los distintos modelos y el usuario deberá poseer los conocimientos necesarios del lenguaje en el que implemente las transformaciones. También se podrían adaptar AndroMDA y ArgoUML, pero en ambos casos la adaptación de las herramientas implica un esfuerzo de desarrollo adicional, ya que se implementarían no solo los meta-modelos sino también las transformaciones entre los modelos.

En la siguiente tabla se resumen las características evaluadas en cada una de las herramientas.

	<i>Eclipse EMF</i>	<i>AndroMDA</i>	<i>ArgoUML</i>
<i>Características Funcionales</i>			
<i>Niveles que cubre</i>	CIM – PIM – PSM	PIM – PSM	CIM - PIM – PSM
<i>Grado de Generación de código</i>	JAVA principalmente	Cualquier lenguaje. Se debe tener el <i>cartridge</i> adecuado	JAVA + AndroMDA
<i>Transformaciones</i>	Completamente	Completamente	Completamente
<i>Grado de Interacción con el Usuario</i>	Alto	Alto	Alto
<i>Tipos de Transformaciones</i>	Verticales. Se pueden implementar las horizontales	Verticales. Se pueden implementar las horizontales	Verticales. Se pueden implementar las horizontales
<i>Características Técnicas</i>			
<i>Lenguaje de almacenamiento y gestión de modelos</i>	XMI	XMI	XMI
<i>Plataformas y Tecnologías soportadas</i>	Ecore – GenModel – JAVA - H. Case para generar diagramas UML	H. Case generen diagramas UML con formato XMI – Nhibernate – Maven – JBoss	JAVA
<i>Ámbito de Aplicación</i>	Desarrollos orientados a servicios	SIW - Desarrollos orientados a servicios	SIW - Desarrollos orientados a servicios
<i>Características de Calidad</i>			
<i>Uso de estándares</i>	XMI – UML – MOF	XMI – UML – MOF	UML – XMI – SVG – OCL
<i>Extensibilidad</i>	Por medio de <i>pluggins</i>	Por medio de <i>cartridge</i>	Integración con herramientas
<i>Usabilidad</i>	Fácil de usar	Difícil de usar	Fácil de usar
<i>Interoperabilidad entre herramientas</i>	Integración con otras herramientas	Integración con otras herramientas	Integración con otras herramientas

Tabla1: Resultado del análisis comparativo de herramientas MDA

6. Conclusiones

En este artículo se han analizado las siguientes tres herramientas: Eclipse, AndroMDA y ArgoUML. El objetivo ha sido determinar si alguna de estas puede ser adaptada para dar soporte a la arquitectura de modelos MIDAS o, si por el contrario, es necesario el desarrollo de una nueva herramienta MDA. Para ello, se han seleccionado características funcionales, técnicas y de calidad. Se han analizado todas las características determinando el cumplimiento de las mismas. Además, para evaluar el funcionamiento de las herramientas se ha desarrollado el caso de estudio definido con cada una de ellas.

Este estudio ha permitido concluir, como se ha dicho en el apartado anterior, que de las herramientas evaluadas hasta el momento, Eclipse es la que mejor se adapta a nuestras necesidades. Eclipse complementado con los *plug-ins* de EMF, GMF y UML2 permite de manera sencilla y gráfica generar editores de modelos en base a los meta-modelos definidos por MIDAS.

Actualmente se está trabajando en la evaluación de las herramientas con otros casos de estudio, es decir, con el resto de los meta-modelos propuestos por MIDAS para completar el análisis que se está llevando a cabo, además se está modificando el proceso de evaluación de las herramientas incluyendo un ranking numérico de cumplimiento de cada una de las características.

Esto nos permitirá especificar la arquitectura de la herramienta e implementar un meta-editor de los meta-modelos propuestos por MIDAS.

Agradecimientos

Este artículo está enmarcado en el proyecto GOLD (TIN-2005-0010) financiado por el Ministerio de Ciencia y Tecnología de España y en el proyecto FOMDAS (URJC-CM-2006-CET-0387) financiado por la Comunidad de Madrid y la Universidad Rey Juan Carlos.

Referencias

- [1] *An Interview with Matthias Bohlen*. <http://www.codegeneration.net>. Marzo 2004.
- [2] Backansky, V., *Mastering Eclipse modelling Framework, Tutorial*. EclipseCON'2005. Febrero 2005.
- [3] Bézivin, J., Valduriez, P., Jouault, F. *The ATL home page*. <http://www.sciences.univnantes.fr/lina/atl>.
- [4] Fuentes, L., Vallecillo, A. *Una introducción a los perfiles UML*. Novatica Nro. 168. Marzo – Abril 2004.
- [5] Gallardo, D. Burnette, E. y McGorven, R. *Eclipse in action, a guide for Java Developers*. Manning Publications. Septiembre 2002.
- [6] García Molina, J. Rodríguez, J. Menárguez, M. Ortín, M.J. y Sánchez, J. *Un estudio comparativo de dos herramientas MDA: OptimalJ y ArcStyler*. I Taller sobre desarrollo dirigido por modelos, MDA y aplicaciones (DSDM'2004), noviembre 2004. España.
- [7] ISO / IEC 9075 Standard, Information Technology – Database Languages – SQL:2003. International Organization for Standardization, 2003.
- [8] King's College London. *An Evaluation of Compuware OptimalJ Professional Edition as an MDA Tool*. University of York, 2003.
- [9] Kontio, M. *Architectural manifiesto: MDA in Action*. <http://www-128.ibm.com/developerworks/library/wiarch19/index.html>. Octubre 2005.
- [10] Kozikowski, J. *A bird's eye view of AndroMDA*. <http://galaxy.andromda.org/docs-3.1/contrib/birds-eye-view.html>. Noviembre 2005.
- [11] Marcos, E. Vela, B., Cáceres, P. y Cavero, J.M. *MIDAS/DB: a Methodological Framework for Web Database Design*. DASWIS 2001. LNCS 2465, Springer-Verlag, pp. 227-238, septiembre, 2002.
- [12] Marcos, E., Vela, B. y Cavero J.M. *Methodological Approach for Object-Relational Database Design using UML*. Journal on Software and Systems Modeling (SoSyM). Springer-Verlag. Ed.: R. France and B. Rumpe. Vol. SoSyM 2, pp.59-72, 2003.
- [13] Moore, E., Vela, B. y Cavero J.M. *Methodological Approach for Object-Relational Database Design using UML*. Journal on Software and Systems Modeling (SoSyM). Springer-Verlag. Ed.: R. France y B. Rumpe. Vol. SoSyM 2, pp.59-72, 2003.
- [14] MOF QVT Standard Specification. <http://www.omg.org/docs/ptc/05-11-01.pdf>
- [15] Moore, B. Dean, D. Gerber, A. Wagenknecht, G. y Vanderheyden, P. *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. <http://ibm.com/redbooks>. Febrero 2004.
- [16] Naveed Ahsan Tariq and Naeem Akhter. *Comparison of Model Driven Architecture (MDA) based tools (A Thesis document)*. Junio, 2005
- [17] OMG. *MDA Guide Version 1.0. Document number omg/2003-05-01*. Ed. Millar, J y Mukerji, J. <http://www.omg.com/mda>.
- [18] OMG, *UML Superstructure 2.0*. OMG Adopted Specification ptc/05-07-04. <http://www.uml.org/>, 2005.
- [19] Ramírez, A. Vanpeperstraete, P. Rueckert, A. Odutola, K. Bennett, J. Tolke, L. y van der Wulp, M. *ArgoUML User Manual: A tutorial and reference description*. <http://www.opencontent.org/openpub/> 2006
- [20] Quintero, J., Anaya, R. Marco de Referencia para la Evaluación de Herramientas Basadas en MDA. IDEAS 07, Mayo 2007, Venezuela.
- [21] Vara, J. M., Vela, B., Cavero, J. M., y Marcos, E. *Model Transformation for Object-Relational Database development*. ACM Symposium on Applied Computing 2007 (SAC 2007). Seul (Korea), Marzo, 2007.
- [22] Vela, B., Cáceres, P., de Castro, V., Marcos, E. *Midas: una aproximación dirigida por modelos para el desarrollo ágil de sistemas de información web*. Ingeniería de la web y patrones de diseño. Capítulo 4. Pearson – Prentice Hall. 2005
- [23] What is Maven?. <http://maven.apache.org/what-is-maven.html>.