Duarte, JM.; González Toñánez, M.; Cernuzzi, L.; Pastor López, O. (2008). Evaluation of software development through an MDA tool: a case study. IEEE Latin America Transactions. 6(3):252-259. doi:10.1109/TLA.2008.4653855.

# Evaluación del desarrollo de software mediante una herramienta MDA: un caso de estudio

J. Duarte, M. González, L. Cernuzzi, O. Pastor

***Resumen*— *Model Driven Architecture* (MDA) surge a principios de esta década como una propuesta de estandarización del enfoque *Model Driven Development* (MDD). MDA adopta a los modelos como elementos esenciales de la cadena de producción de software, proporcionando un conjunto de lenguajes estándares para manejar eficientemente tales modelos. Según los creadores de MDA, la postura de mantener énfasis en la obtención de un grado satisfactorio de productividad, en depreciación de otras características como portabilidad, interoperabilidad, reusabilidad y mantenibilidad, es la que se pretende cambiar con la aplicación de MDA al desarrollo de los sistemas informáticos, independientemente del dominio específico del que se trate una aplicación. El objetivo de este trabajo es verificar la efectividad de tales características, a partir de la experiencia obtenida en el desarrollo de una aplicación real usando por un lado una herramienta MDA, contrastándola con el enfoque tradicional.**

***Palabras clave*— *Model Driven Architecture*, MDA, Desarrollo dirigido por modelos (*Model Driven Development*), MDD, ingeniería de software (*software engineering*), modelado UML (*UML modeling*), generación de código (*code generation*).**

## I. Introducción

*Model Driven Architecture* (MDA) surge a principios de esta década como una propuesta de estandarización del enfoque *Model Driven Development* (MDD) en el desarrollo de software. La regla principal de la propuesta MDA para poder cumplir sus objetivos es la de usar en lo posible estándares, por ello la base (*core*) de la arquitectura lo conforman estándares definidos por el OMG (*Object Management Group*) que permiten trabajar con modelos como componentes esenciales en el proceso de desarrollo de software [1]. MDA adopta a los modelos como elementos esenciales de la cadena de producción de software, proporcionando un conjunto de lenguajes estándares para manejar eficientemente los modelos. En particular se consideran dos principales modelos: el Modelo Independiente de Plataforma (PIM, por sus siglas en inglés), y el Modelo Dependiente de Plataforma (PSM, por sus siglas en inglés).

## II. Arquitectura de MDA

El PIM es el modelo con mayor nivel de abstracción. Describe la estructura, funcionalidad y restricciones del sistema, omitiendo detalles sobre cómo y dónde va a ser implementado.

En el PSM, además de contener la información del PIM, incluye los detalles relacionados a una plataforma de implementación dada (.NET, J2EE, etc.). Este modelo se obtiene partir del PIM por medio de la transformación que lleva a cabo la herramienta MDA teniendo en cuenta la plataforma seleccionada de destino y eventualmente algunos parámetros de ajustes que el desarrollador proporciona. Varios PSM's pueden ser generados a partir de un mismo PIM, cada uno describiendo al sistema desde una perspectiva diferente.

De todo esto podemos ver que un componente importante del enfoque MDA recae sobre una o varias herramientas MDA. En este proceso se intenta lograr el grado máximo posible de automatización de la construcción de los sistemas, haciendo uso intensivo de las transformaciones para una o más plataformas específicas de implementación.

Actualmente es posible contar con herramientas MDD, que ofrecen soporte para una plataforma o dominio de aplicación particular, permitiendo a los desarrolladores la construcción de software con cierta eficiencia y productividad notable en esos entornos. No obstante, la postura de mantener énfasis en la obtención de un grado satisfactorio de **productividad**, en depreciación de otras características como **portabilidad**, **interoperabilidad**, **reusabilidad** y **mantenibilidad**, es la que se pretende cambiar con la aplicación de MDA al desarrollo de los sistemas informáticos, independientemente del dominio específico del que se trate una aplicación. Sin embargo, en la literatura especializada no se cuenta con suficiente documentación de experiencias y casos de estudio que confirmen estas tendencias.

## III. Objetivos del trabajo

A raíz de la necesidad de poner a prueba el desarrollo de aplicaciones en MDA para determinar el alcance de la implementación actual de este nuevo enfoque en las herramientas que dicen conformarse al estándar MDA, surge este trabajo en el que se establecen dos propósitos principales.

Por un lado, la verificación del desarrollo completo de una aplicación con una herramienta MDA concreta, en particular ArcStyler [6], seleccionada a partir de un proceso de análisis comparativo entre un conjunto de herramientas (ArcStyler, Objecteering [11], Enterprise Architect [12], Model-In-Action Generation [13] y OlivaNova [14]). En dicho análisis fueron considerados principalmente las siguientes prestaciones de cada herramienta: 1) que cumpla en lo posible con las especificaciones del estándar MDA, 2) que sea conocida en el ámbito MDA y que haya sido objeto de estudios previos, 3) que la generación automática cubra el mayor número posible de componentes de las capas de la aplicación: base de datos, lógica de aplicación e interfaz de usuario, y 4) que soporte más de una plataforma para la generación de aplicaciones.

Por otro lado, se pretende efectuar la comprobación y evaluación de las características obtenidas del producto realizado por el proceso de desarrollo MDA, contrastando con

los resultados obtenidos por el método tradicional. Esto nos permite establecer el grado efectivo de satisfacción de las características afirmadas en el documento de especificación del estándar MDA (productividad, portabilidad, interoperabilidad, reusabilidad y mantenibilidad) [1], y en varias publicaciones de los distintos propulsores de esta iniciativa [2, 3, 5]. Para realizar el proceso de evaluación, se identificaron una serie de atributos, y eventuales métricas que fueron aplicadas para cada una de las características afirmadas.

## IV. AMBIENTE DE DESARROLLO DEL CASO DE ESTUDIO

La herramienta **ArcStyler** [12] fue la escogida para trabajar y poner a prueba el desarrollo en MDA de un sistema real a modo de caso de estudio. Esta herramienta fue seleccionada por reunir mejor calificación de entre un grupo de herramientas pre-evaluadas como fue indicado en el punto anterior.

Una característica elemental de esta herramienta es la definición de su arquitectura, cuyos componentes centrales son los **cartuchos** o *MDA-cartridges*, que conforman el "motor" o *MDA-engine* de la herramienta. Además, ArcStyler permite que un usuario defina sus propios cartuchos, o extienda algunos de los proporcionados.

ArcStyler logra la transformación de modelos a código a partir de un PIM marcado y estereotipado con información de la plataforma de implementación.

La aplicación de caso de estudio seleccionada es una adaptación de un sistema empleado en un ambiente de trabajo real, que sirve de gestión y despliegue de información contable de los departamentos de una Facultad. Este sistema de gestión cliente-servidor llamado **Sistema de Rubros**, satisface las necesidades propias de gestión de gastos e ingresos y de información de ejecución presupuestaria de la Facultad de Ciencias y Tecnología de la Universidad Católica de Asunción. El desarrollo de este sistema originalmente se ha hecho siguiendo la técnica tradicional en el marco de procesos evolucionarios (*rapid prototyping)* o de crecimiento incremental, en el cual, a partir de diseños de las interfaces ajustadas a las necesidades y requerimientos de los usuarios, se van construyendo gradualmente todos los componentes restantes, haciendo el programador la codificación 'en duro' de la lógica de negocios, de disposición de los elementos de interfaz y de interacción con la base de datos. El tipo de interfaz empleado para esta aplicación es el de ventanas y diálogos típicos de un sistema *stand-alone*. A fin de dimensionar el sistema de rubros original, enumeramos a continuación los artefactos que componen al mismo: 13 entidades de lógica de negocios implementadas en 22 tablas de la base de datos, 50 pantallas de interfaces de usuario y las definiciones adicionales de la base de datos que abarcan 4 vistas, 12 *triggers* y 6 procedimientos almacenados. Estos elementos sirven de soporte para las 38 funcionalidades que ofrece el sistema a sus usuarios.

Para el desarrollo del sistema de caso de estudio planteado, se eligió implementar un número significativo de funcionalidades, las cuales suman 24 y representan las operaciones elementales que satisfacen los requisitos más importantes del sistema. Se decidió trabajar en el desarrollo de esta aplicación en la plataforma Java, específicamente J2EE. También optamos por brindar un entorno web como interfaz de usuario, pues ese es el que dispone de un soporte pleno en la herramienta ArcStyler.

En la Fig. 1 se reúnen los diagramas del modelo de la aplicación necesarios en el desarrollo con la herramienta MDA ArcStyler, que deben contener las especificaciones apropiadas de los componentes implicados de lógica y presentación para poder ser interpretados por los cartuchos MDA correspondientes durante la etapa de generación de artefactos.

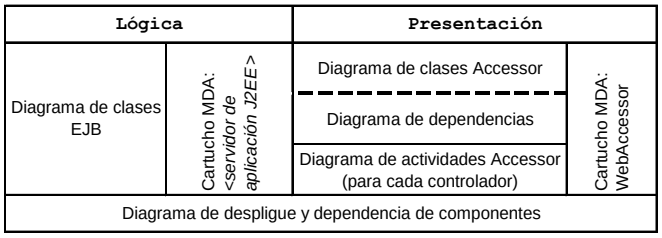


Fig. 1. Diagramas utilizados para el desarrollo en MDA con la herramienta ArcStyler

El modelo de la *Lógica de Negocio* con la herramienta ArcStyler consiste en el modelo de los componentes de la lógica de negocios en un diagrama de clases UML. Esta etapa es relativamente sencilla y permite al desarrollador ir notando la variedad y complejidad de los componentes básicos de la aplicación. Si bien, el enfoque de este diagrama es el de uno independiente de plataforma (PIM), por las características ya mencionadas de la herramienta se deben ir indicando con estereotipos y marcas MDA algunas propiedades distintivas de la plataforma de implementación, en éste caso J2EE.

El modelado de la capa de *Presentación* de la aplicación siguiendo el *framework Accessor* es el trabajo más costoso de todo el proceso de desarrollo en la herramienta ArcStyler. El *framework Accessor* [8] incorporado en ArcStyler provee un proceso bien definido para la construcción de la capa de presentación de las aplicaciones. En el modelo *Accessor*, se establecen el flujo de control y el flujo de datos de la capa de presentación de un modo abstracto. Un modelo *Accessor* es una instancia del metamodelo del mismo nombre: «Accessor», el cual es una extensión del metamodelo UML. El metamodelo *Accessor* resuelve los aspectos de Vista y Control del paradigma Modelo/Vista/Controlador (MVC). Este paradigma se usa para modelar las interacciones de la capa de presentación con la de lógica de negocio, que representa la parte de Modelo de MVC.

Aproximadamente el 80% del tiempo dedicado al modelado se insume en la capa de presentación. Explicando resumidamente, en esta etapa se proceden a identificar y especificar las vistas (*representers*), los controladores (*accessors*) y las transiciones entre las distintas vistas (interfaces) y las acciones esperadas entre cada una de ellas modeladas en un diagrama de estados extendido de UML por cada controlador.

En la etapa de *Generación de Código* del proceso de desarrollo de software en MDA se realiza la generación de código para la plataforma de implementación por medio de los

módulos del Generador y los cartuchos MDA, los cuales permiten crear las secciones mayores de la infraestructura de despliegue, como también la preparación de los entornos de construcción (compilación) y *testing*, todos ellos a partir de los modelos UML.

La generación de código, si bien está disponible en cualquier etapa de desarrollo y puede hacerse para un solo elemento en particular, no es segura ni completa hasta no concluir la etapa del modelado de los componentes *desplegables*[1], en especial al primer componente esencial, el que contiene las entidades de la lógica de la aplicación. Además, como comprobación de la validez del modelo, es recomendable utilizar el verificador de modelos proporcionado por el cartucho MDA de la plataforma J2EE para el servidor de aplicaciones definido (JBoss, WebLogic, etc.).

En cuanto al aprendizaje en el manejo de la herramienta, podemos afirmar que ésta presenta una curva de aprendizaje de esfuerzo moderado, pues a pesar de constituir un entorno de desarrollo integrado, con una documentación incluida más o menos extensa, y con ejemplos de proyectos simples, es necesaria una cuidadosa lectura de cada parte de la documentación. Además es importante también el ensayo con tutoriales para comprender tanto el entorno de trabajo como la serie de pasos que deben seguirse en cada fase de trabajo. Esto se debe a que algunos detalles, aunque simples pero importantes en el proceso de desarrollo, no resultan demasiado obvios para el desarrollador principiante.

## V. Análisis de los resultados

Aprovechamos la experiencia ganada en el desarrollo de dos sistemas que ofrecen un conjunto de funcionalidades equivalentes pero realizados por procesos diferentes para determinar el nivel de satisfacción de las cualidades seleccionadas para ambos procesos involucrados. En base a ello presentamos un análisis de las cualidades de la aplicación obtenida siguiendo el enfoque MDA con el apoyo de la herramienta ArcStyler. Además establecemos una comparación de los puntos principales del desarrollo de la aplicación de caso de estudio entre el sistema construido por el proceso tradicional y el sistema realizado con la herramienta MDA para éste trabajo. Es importante tener en cuenta que los resultados derivados de la evaluación del enfoque MDA en el desarrollo del sistema de caso de estudio corresponden a particularidades de las partes implicadas. Por un lado, el desarrollo hecho con el soporte de una herramienta específica está sujeto a la implementación propia de la herramienta ArcStyler del enfoque MDA y las prestaciones o funciones que ofrece a sus usuarios. Por otra parte, tenemos que la aplicación modelada y generada por la herramienta se trata de un sistema típico de información y gestión, en el que las tareas más realizadas se refieren a funciones *CRUD* (crear, recuperar, actualizar y borrar datos). Por lo tanto, las afirmaciones dadas en este análisis no necesariamente deben extenderse a otros tipos de sistemas más complejos o especiales (sistemas de tiempo real, sistemas embebidos, etc.), si bien los investigadores y expertos en la arquitectura MDA y la misma herramienta ArcStyler afirman poder aplicarse plenamente a la construcción de varios tipos de sistemas. No obstante, consideramos que los resultados hallados son de interés y relevancia para los desarrolladores y demás personas involucradas en la industria del software, ya que gran parte de la demanda de software a medida son sistemas que contienen principalmente funciones *CRUD*.

En el siguiente cuadro se resumen las características y los atributos empleados en la evaluación de las cualidades obtenidas del sistema de caso de estudio desarrollado. Como fue mencionado en la introducción, estas características se definen a partir de las cualidades que el enfoque MDA pretende satisfacer, mencionados explícitamente en el documento de especificación [1], y también indicado por algunos autores que promueven la arquitectura [2, 3, 5]:

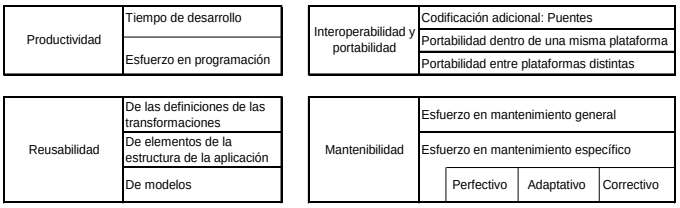


Fig. 2. Atributos de evaluación empleados en el caso de estudio.

Los atributos definidos por cada característica, fueron identificados a partir de lo siguiente:

- Selección de atributos que puedan ser fácilmente cuantificables a partir de métricas conocidas: tiempo de desarrollo, esfuerzo en programación, esfuerzo en mantenimiento general, codificación adicional de puentes.
- Atributos, que si bien no son cuantificables, son fácilmente comparables: los atributos definidos para la característica de reusabilidad y la portabilidad dentro de una misma plataforma.
- Atributos deseables para una característica en particular que pueden ser comparables de manera cualitativa: la portabilidad entre plataformas distintas, y los atributos indicados para el esfuerzo de mantenimiento específico.

La definición de las métricas y evaluación pretende más bien ayudar a analizar las características deseables, de una manera más bien cualitativa, utilizando, en aquellos casos en que fueron posibles, valores cuantitativos. Si bien, esta experiencia no representa un proceso totalmente riguroso de validación, consideramos que puede ser muy útil para realizar un análisis del grado de satisfacción de tales características.

### A. Productividad

#### 1) Tiempo de desarrollo

Con el desarrollo del sistema por el proceso enmarcado en la arquitectura MDA, pudimos notar que prácticamente todas las funcionalidades consideradas de la aplicación de caso de estudio pudieron lograrse eficazmente, previa definición de ajustes manuales en el código fuente. Por funcionalidad definimos como una tarea indivisible que debe llevar a cabo el sistema, pudiendo exactamente corresponder a un caso de uso, o formar parte de un conjunto de tareas que inscribe un caso

[1] En una aplicación típica, usualmente se tendrán dos módulos componentes principales: el de la lógica de negocios y el de la capa de presentación. Este mismo enfoque es el aplicado para el modelado de los componentes desplegables en la herramienta.

de uso.

Con el fin de dimensionar el tamaño del sistema, utilizamos la métrica del número de funcionalidades que implementa la aplicación. Conociendo el tiempo empleado en implementar todas las funcionalidades, desde el inicio del desarrollo hasta la disposición de la aplicación para su comunidad de usuarios, podemos emplear otra métrica a la que llamaremos **índice de productividad** *p*, que es igual al la cantidad de funcionalidades ***F*** efectivas que presta la aplicación, dividido el tiempo total ***T*** de desarrollo e implementación: $p = F/T$.

Con los valores de ésta unidad correspondientes a los procesos de desarrollo tradicional y en MDA para el caso de estudio expuesto, podemos establecer una primera comparación objetiva para la característica de *productividad* en la elaboración de sistemas, y verificar la conformidad o no de ésta propiedad obtenida utilizando el enfoque MDA en el desarrollo, en confrontación con los métodos tradicionales de *Rapid Prototyping* / *Code and fix* (prototipado rápido / codificar y ajustar).

En la tabla I se recogen los datos de las métricas descritas arriba concernientes a ambos métodos de desarrollo del sistema adoptado como caso de estudio:

TABLA I
TIEMPOS DE DESARROLLO DEL SISTEMA DE RUBROS FCYT

| MÉTODO DE DESARROLLO | ENFOQUE TRADICIONAL | ENFOQUE MDA |
|---|---|---|
| Cantidad de funcionalidades (F) | 24 | |
| Tiempo total de desarrollo (días trabajados) (T) | 90 | 15 |
| Tiempo total de desarrollo (días calendario para referencia) | 140 días [4 meses] | 20 días [3 semanas] |
| Valor $p$ = F/T | 0,267 | 1,6 |
| Tiempo de ahorro | -------- | **600%** |

Otra observación importante es que las funcionalidades implementadas en el sistema original incluyen las tareas relativas a la validación de datos introducidos por el usuario, en todas las interfaces concernientes a la ejecución de una tarea, cuidados que no fueron incorporados en la aplicación desarrollada por el enfoque MDA, por tratarse meramente de un trabajo de codificación adicional, que no tiene contribución significativa en la satisfacción de los requisitos funcionales del sistema, claves para establecer criterios de comparación entre ambos métodos de desarrollo.

Como puede apreciarse, los tiempos insumidos en el desarrollo de ambos sistemas por una sola persona difieren significativamente, notándose una reducción de tiempo de desarrollo con el enfoque MDA de hasta 6 veces menos que el empleado sin utilizar enfoque alguno de MDD. Esta diferencia hallada, que muestra marcadamente un acentuado aumento en el índice de productividad del desarrollo de sistemas a favor del primer enfoque, no obstante puede ser matizado con otros factores relevantes del sistema hecho con la herramienta MDA tales como: la falta de tiempo consignado en tareas que confieren la adecuación de requisitos no funcionales de la aplicación como módulos de validación, ajustes manuales de las interfaces a requerimientos visuales, así como los entornos de presentación (entorno *stand-alone* en el sistema original vs. entorno *web* de la aplicación de caso de estudio en MDA).

El esfuerzo en el modelado incide mayormente en el componente de presentación, debido al número de elementos que deben diseñarse. De acuerdo a observaciones del desarrollo del sistema de caso de estudio, estimamos que el tiempo dedicado al modelado de este componente supera ligeramente el 80% del tiempo total de diseño y especificación, perteneciendo la fracción restante al modelado de la lógica de negocios, que no pasa el 20% del tiempo empleado. Estos datos sugieren que la lógica de negocio para este tipo de aplicaciones CRUD es relativamente simple y con un bajo grado de complejidad. Sin embargo, como se puede apreciar en el siguiente indicador, gracias al enfoque MDA es posible una fuerte reducción del esfuerzo manual de desarrollo de la lógica de negocio lo cual no deja de ser una ventaja significativa. En el modelado de la presentación, gran parte del trabajo es dedicado al diseño de las vistas, en donde se esbozan los elementos de interfaz de usuario y los atributos de los *representers*.

El tiempo total dedicado a la presentación, podría extenderse un poco más de lo indicado en la tabla anterior, ya que luego de la generación de las interfaces pueden requerirse ajustes manuales en la apariencia de algunas interfaces de usuario.

En la experiencia con la aplicación de caso de estudio, se determinó efectuar ciertos ajustes a determinadas interfaces que constituyen las esenciales para realizar las operaciones más importantes del sistema de rubros, conforme a las necesidades de los usuarios. Se editaron en total 12 vistas de interfaz, que incluyen 9 *webforms* y 3 reportes. Este conjunto permite realizar eficientemente 8 funcionalidades, de las 24 que implementa el sistema desarrollado en MDA. Los formularios web sirven de implementación a 5 funcionalidades, y cada uno de los reportes implementa una funcionalidad. El tiempo empleado en los ajustes de los formularios web fue de 3 horas, y de los tres reportes en total fue de 3 horas. Haciendo una estimación de lo que llevaría completar los ajustes en las interfaces relacionadas a las funcionalidades restantes (2/3), obtenemos que se necesitarían 12 horas adicionales para tener completo los ajustes más importantes de las interfaces (páginas web) de la aplicación de acuerdo a los requisitos de los usuarios. A una razón de 6 horas de trabajo en desarrollo por día, tenemos que se debe añadir 2 días de trabajo aproximadamente al tiempo expresado en la tabla 1, lo que nos da un valor de 17 días. Si consideramos esta cifra como nuevo tiempo total de desarrollo, tenemos que 3 de los 17 días, o el 17,6% del tiempo total de desarrollo correspondería a cambios manuales en los artefactos de interfaces generados automáticamente por la herramienta.

*2) Esfuerzo en programación*

Podemos determinar además otro índice de productividad utilizado desde los primeros tiempos de la aplicación de ingeniería de software: el número de líneas de código (***LOC***). Haciendo una adaptación para su empleo en el desarrollo con MDA, definimos la métrica que indica la cantidad de líneas de código efectivo generado por la herramienta, y la cantidad de código necesario que debe ser programado manualmente. De esta manera se puede obtener una estimación interesante del "*trabajo*" hecho por la herramienta y por el programador, esperando obtener como resultado una proporción de esfuerzo realizado en términos de línea de código mucho mayor para la herramienta que el producido por el programador, siguiendo las expectativas de MDA. En la tabla II se recogen los resultados del cálculo aplicado a los archivos fuentes indicados[2].

Como podemos apreciar en los datos, el esfuerzo en el código agregado manualmente, alcanza aproximadamente al 10% de todo el código fuente. Este esfuerzo (en programación) es modestamente superior cuando se toma en cuenta solo los controladores de la capa de presentación, llegando cercanamente al 15% del código fuente correspondiente a los controladores de la arquitectura MVC de *Accessor*, que como sabemos, proporcionan la "inteligencia" o dinámica del sistema.

TABLA II
RESUMEN DE LA PROPORCION DE CÓDIGO GENERADO Y CÓDIGO PROGRAMADO

| | TOTAL | MANUAL | GENERADO |
|---|---|---|---|
| Lógica de negocios | 5.289 | 152 | 5.137 |
| | 100% | 2,87% | 97,13% |
| *Controladores de la lógica de presentación* | | | |
| `Login` | 551 | 42 | 509 |
| `MenuPrincipal` | 7.007 | 1.075 | 5.932 |
| | 100% | 14,78% | 85,22% |
| *Resumen de la aplicación* | | | |
| Lógica de negocios | 5.289 | 152 | 5.137 |
| Controladores de la presentación | 7.558 | 1.117 | 64.441 |
| | 12.847 | 1.227 | 11.620 |
| | 100% | 9,55% | 90,45% |

*B. Interoperabilidad y Portabilidad*

*1) Codificación adicional: Puentes*

El trabajo de las herramientas MDA no debería limitarse a transformar modelos y generar código sino también de generar pasarelas o puentes entre las diferentes capas de la aplicación para que puedan interoperar satisfactoriamente. A las conexiones entre los componentes de una aplicación componentes (base de datos, lógica de la aplicación, interfaz de usuario) se lo llaman "puentes", y son necesarios para completar la implementación de las funcionalidades elementales de la aplicación. En ArcStyler encontramos que el esfuerzo "manual", indispensable para tener listos los fuentes de la aplicación de manera a llevar a cabo la construcción completa e interoperable de la aplicación, concierne básicamente a la programación manual de esas conexiones o pasarelas, concretamente a las interconexiones entre las entidades (*EJB's* desplegados) del modelo de la lógica, y los elementos de interfaz de las vistas (páginas), coordinadas adecuadamente desde las operaciones de los controladores (*accessor*) pertinentes. Si bien se podría esperar algún grado mayor de generación automática de tales puentes haciendo uso del rico mecanismo de marcas y patrones de MDA, hallamos que la adición y edición de código es racionalmente manejable, ágil y sin mayores dificultades.

*2) Portabilidad dentro de una misma plataforma*

La portabilidad dentro de una misma plataforma como el caso de J2EE y sus distintos servidores de aplicaciones es coherentemente convincente y rápida, quizás considerando que los modelos definidos estén orientados a una misma definición de arquitectura (J2EE) y que el cambio de servidor de aplicación sólo conlleva a ajustes específicos de configuración propios del servidor de destino para el cual se elige la generación, mediante las marcas ofrecidas por el cartucho MDA del servidor J2EE elegido. Particularmente en esta investigación se ha trabajado con los servidores Jboss 4 y WebLogic 8.1, y en lo que respecta a la experiencia del desarrollo para estos productos con la herramienta ArcStyler podemos hacer las siguientes observaciones:

- El modelo de lógica de negocios que comprende la descripción completa de *beans* no sufre de alteraciones importantes cuando se trata del cambio de proveedores de implementación J2EE. No obstante, el verificador de modelos avisa las incompatibilidades encontradas cuando no reconoce alguna especificación del modelo para el cartucho MDA del servidor de destino. Por ejemplo, las clases utilizadas como enumeraciones, modeladas adecuadamente con el estereotipo respectivo, no son soportadas por el servidor WebLogic 8.1, lo que hace que salte un aviso de error si existe en el modelo tal instancia. Sin embargo, en el servidor JBoss son reconocidas perfectamente. De acuerdo a la misma documentación de ArcStyler [10], el modelo Accessor y su debido conjunto de artefactos generados, no es 100% compatible con los servidores distintos a WebLogic 8.1.
- El otro framework disponible en la herramienta para la presentación, Struts, no tiene un soporte completo y funcional como Accessors, ya que los artefactos generados están incompletos y requieren de mucho trabajo adicional en la manipulación de código. El modelado en Struts lleva aproximadamente el mismo tiempo de diseño que el realizado en su par Accessor, lo que determina un innecesario trabajo adicional si se escoge primeramente el

[2] Las líneas de código consideradas sólo corresponden a los archivos fuentes en los que se definen clases e interfaces de la lógica de la aplicación. Por lógica de aplicación incluimos a las definiciones de las entidades del sistema, que para la plataforma J2EE son los *beans* y sus interfaces, además de la lógica de los controladores *accessors* del modelo web, que no abarcan a las clases *representers* que enmarcan las vistas o interfaces de usuario de la aplicación

framework Struts para el diseño de la capa de presentación, y luego se cambia por Accessor al ver que los artefactos generados para Struts son improductivos.

- El trabajo con una base de datos específica (distinta a la establecida por defecto en el servidor de aplicación) no requiere de cambios especiales en el modelo. Sólo se necesitan establecer en las marcas de la configuración de transformaciones los datos relativos al servidor de base de datos, como autenticación de usuario, ubicación y nombre de base de datos, *driver* JDBC, etc. No obstante, el tiempo necesario para llevar a cabo los ajustes del servidor J2EE con el servidor de base de datos específico varía según el motor de base de datos y de la experiencia del desarrollador en llevar a cabo dichos ajustes.

*3) Portabilidad entre plataformas distintas*

Por otra parte, viendo la portabilidad a otra plataforma distinta a Java, tenemos que la herramienta ofrece el soporte de modelado y generación de artefactos de la plataforma .NET y el lenguaje C# a través de los cartuchos MDA respectivos. De manera similar a la arquitectura J2EE, la generación de artefactos para la plataforma .NET está orientada a la arquitectura *enterprise* a través del modelado de los *Enterprise Distributed Objects* con la definición de las 4 capas de la arquitectura (*bussiness facade*, *bussiness rules*, *data access*, *common*), y la generación de los artefactos correspondientes. No obstante, el modelado y generación de la aplicación no se ha realizado para esta plataforma por lo que se encuentra fuera del alcance de los resultados analizados.

*C. Reusabilidad*

*1) De las definiciones de las transformaciones*

ArcStyler no sólo permite al usuario la elección de la plataforma de trabajo, y la selección de un proveedor *middleware*. Al no ser una herramienta "cerrada", el soporte para nuevas plataformas no depende exclusivamente del fabricante de la herramienta, sino del desarrollo de los mismos usuarios de nuevos cartuchos MDA (*MDA-Cartridge*) ó de la extensión de los cartuchos MDA ya existentes con el fin de adaptarlos a los requisitos específicos para luego utilizarlos en proyectos concretos.

Mediante la arquitectura CARAT [4, 7, 9] y un entorno de desarrollo exclusivo para la edición y creación de cartuchos brindados por la herramienta, se pueden definir nuevos cartuchos con los ajustes necesarios para realizar las transformaciones buscadas utilizando el mecanismo de herencia de cartuchos MDA. Estas tareas, aunque requieren un conocimiento profundo de la arquitectura CARAT, pueden proporcionar definiciones de transformación ajustadas a los requisitos comunes del dominio de los sistemas, como ser una empresa o una industria de software, en donde las aplicaciones desarrolladas presentan con regularidad patrones similares en funcionalidades y detalles de presentación. Dicha utilidad podría ser interesante si se aplica eficazmente en la construcción de sistemas, pues se puede llegar a ahorrar tiempo sustancial en el modelado y en la codificación manual de secciones comunes de la aplicación mediante la utilización de patrones por ejemplo.

*2) De elementos de la estructura de la aplicación*

Podemos señalar que como consecuencia del enfoque MDA y de su lógica subyacente que es la orientación a objetos, contamos con modelos y artefactos cuyos elementos son fácilmente reusables en distintas partes de la aplicación. Además, al momento de agregar código adicional para completar la implementación de funcionalidades, el esfuerzo del programador es mínimo ya que gran parte del código fuente es generado por la herramienta en base a reglas de transformación adecuados a estándares de plataformas y *frameworks* (como *Accessor* por ejemplo). Esta reducción de esfuerzo para el programador es debida fundamentalmente al mecanismo de generación (cartuchos MDA, generador), que reutiliza las definiciones de las transformaciones, por lo que contamos con artefactos que contienen porciones de código generado muy similares, observándose patrones de código a lo largo del código fuente (reusabilidad de definiciones de transformaciones). En el enfoque de desarrollo tradicional, la disponibilidad de elementos reusables depende de un mayor esfuerzo del desarrollador, quien es responsable del diseño e implementación de componentes reusables de la aplicación.

*3) De modelos*

La reusabilidad de modelos en ArcStyler está disponible básicamente para la lógica de negocios en arquitecturas *enterprise*, como J2EE y .NET EDS, permitiendo en cualquier etapa de desarrollo la inclusión del cartucho MDA relativo a una plataforma específica en el proyecto de trabajo, y de ese modo consignar las marcas necesarias para la transformación a esa plataforma. Sin embargo, el modelo de la capa de presentación está orientado a un ambiente particular (web) y para una plataforma determinada como J2EE. Esta limitación nos demuestra que ArcStyler trabaja con modelos que son más dependientes de plataforma para la capa de presentación de una aplicación.

*D. Mantenibilidad*

*1) Esfuerzo en mantenimiento general*

El índice de productividad indicado en la sección V-A-1 también nos sirve para hallar el costo de mantenimiento de agregar una nueva funcionalidad. Proponemos interpretar el valor obtenido de $p$=1,6 diciendo que se invierte en promedio 1 o 2 días a lo sumo para implementar una funcionalidad, que contempla desde el diseño, la generación, codificación manual y construcción de la aplicación actualizada que ofrece la nueva funcionalidad agregada.

*2) Esfuerzo en mantenimiento perfectivo*

Esta cualidad bien puede ser una de las ventajas más constatadas a lo largo del todo el trabajo de la aplicación de caso de estudio con la herramienta MDA ArcStyler, pues en todo el período de producción se ha realizado una implementación gradual de las funcionalidades. Dicha incorporación gradual de nuevas funcionalidades es posible gracias al mecanismo del generador de la herramienta, que permite mantener una consistencia entre los modelos y el código generado, de tal modo a que cualquier cambio hecho al modelo, como la creación de un nuevo elemento, o la eliminación de alguna pieza del modelo (tareas típicas de un

mantenimiento perfectivo), queda reflejado en el código fuente generado, sea con la adición o eliminación de la porción de código correspondiente.

Los cambios manuales introducidos en el código fuente, son conservados en las sucesivas regeneraciones. A esta propiedad facilitada por la herramienta se le conoce con el nombre de "consistencia incremental". A menos que intencionalmente se eliminen uno o más archivos fuentes que contengan porciones de código escritos manualmente, tales modificaciones no son afectadas y son compiladas junto al código bruto generado por la herramienta, formando parte del módulo de ensamblado respectivo de la aplicación.

La aplicación constante de cambios en el modelo conduce a un ciclo de generación-construcción permanente, por lo que es importante mantener también una consistencia entre el código fuente y el código objeto compilado, a fin de que en la construcción se obtenga con certeza los componentes de despliegue correspondientes a la última generación de código. Para ello es necesario aplicar las tareas de "limpieza" de código compilado, que para la plataforma Java se trata de clases compiladas (.class), antes de cada inicio de secuencia de construcción de la aplicación, mediante la ejecución de la operación de limpieza facilitada por la herramienta de construcción ANT.

*3) Esfuerzo en mantenimiento adaptativo*

En cuanto al mantenimiento adaptativo de la aplicación, encontramos que este se realiza sobre todo en los elementos de la capa de presentación, es decir, controladores *accessor* y vistas o *representers*. La dedicación a ajustes en las interfaces, es decir páginas JSP-HTML, es inevitable si se requiere de una estructura apropiada de los elementos GUI o controles de los formularios web conforme a los intereses de los usuarios. En otros casos, la introducción de cambios manuales es imperativa porque el generador produce código incompleto como en la conversión de tipos de datos, en particular para el tipo fecha. Generalmente la aplicación de formato en la presentación de datos de cualquier tipo en las interfaces debe ser manejado manualmente en el código fuente. En la sección V-A-1 pudo verse un detalle de los cambios aplicados en las interfaces generadas por la herramienta de la aplicación de caso de estudio. Otro aspecto referente al mantenimiento adaptativo tiene que ver con el trabajo de aplicar ajustes adicionales para un eventual cambio de plataforma, el cual se realiza de manera ágil y no involucra costo importante cuando se trata de cambios en una misma plataforma, ó entre plataformas diferentes pero dentro de una arquitectura *enterprise* (J2EE, .NET EDS), para el componente de lógica de negocios, tal como se ha mostrado en la sección V-A-2.

*4) Esfuerzo en mantenimiento correctivo*

El código introducido por el programador en distintas partes del código fuente puede no estar exento de errores, lo que podría advertirse en tiempo de compilación o en tiempo de ejecución. Las habilidades del programador en la depuración de código entran en juego, porque facilitan la detección temprana de errores para su corrección adecuada, que pueden influir más o menos en los costos de mantenimiento correctivo del sistema. Los errores que pudieran contener los modelos suelen ser indicados en el momento de generación de artefactos o en la verificación de modelos, por lo que la probabilidad de trasladar errores desde el modelo al código fuente es relativamente menor que aquellos introducidos involuntariamente por el programador en la manipulación del código. Al centrarse más en los modelos el diagnóstico de errores posibles, se facilita considerablemente su corrección y se logra obtener un código fuente generado más fiable, puesto que antes de proceder a la generación de artefactos los eventuales errores son detectados y manejados apropiadamente en su gran mayoría.

Una propiedad deseable que debería soportar toda herramienta MDA es la **trazabilidad**, y se refiere a la capacidad de conocer el destino en el código generado de los elementos del modelo (seguimiento de una *traza*). Esta característica es útil para las tareas de mantenimiento porque contribuye en la optimización en el trabajo con el código fuente. Pero ArcStyler sólo permite ver detalles sobre el modelo indicando los diagramas que contienen un determinado elemento, como una clase o relación. La herramienta no facilita información que permita saber rápidamente y en cualquier momento el origen de los archivos fuentes o el destino en el código fuente de un elemento del modelo.

## VI. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo hemos comprobado el alcance de los beneficios planteados por el enfoque MDA con el desarrollo del sistema de gestión e información presentado en una herramienta particular como *ArcStyler*. Evidentemente, los resultados alcanzados son significativos para el ámbito de la herramienta utilizada y para el tipo de aplicaciones CRUD de pequeño tamaño y complejidad, como es el caso del sistema de rubros. Sin embargo, cabe mencionar que sistemas con estas características son bastante comunes en muchas organizaciones.

Analizando los resultados obtenidos, especialmente en lo que se refiere a ahorro de costos de producción, la experiencia ganada nos ha mostrado un índice de productividad obtenido muy ventajoso, ya que se ha llegado a ahorrar hasta seis veces el tiempo desarrollo para un sistema corriente de información y una importante reducción de esfuerzo en codificación siendo que el programador se ha ocupado sólo del 10% del código fuente de la aplicación. En cuanto al mantenimiento, se ha constatado cierta facilidad en las tareas de mantenimiento perfectivo de la aplicación. No obstante, todavía hay aspectos que pueden mejorarse, principalmente los referidos a las propiedades cualitativas de una aplicación (como ser los requisitos no funcionales que incluyen robustez, seguridad, interfaz del usuario adaptable, etc.), y a las cuestiones de portabilidad entre distintas plataformas, ya que estas se aplican más bien al componente de lógica de negocios en arquitecturas Enterprise, como J2EE y .NET EDS, y no tanto al modelo de la presentación, ya que estos son solo útiles para un entorno y plataforma específicos: aplicaciones Web basadas en Java. Es importante recordar que el logro de los beneficios planteados inicialmente en *MDA* no responde a un único factor, que es la herramienta *MDA*, el cual se considera uno de los más importantes dentro de este enfoque. Existen otros factores

también valiosos que contribuyen a la satisfacción de los objetivos de *MDA* en el desarrollo de un producto, por lo que es importante mencionarlos: el manejo correcto de los lenguajes estándares *MDA* por los desarrolladores, así como los no estándares (por ejemplo, aquellos usados para la definición de transformaciones), el modelado adecuado (diagramas UML y otras extensiones de UML) y la completitud de las especificaciones, la habilidad en la aplicación de ajustes de configuración de las herramientas a las necesidades de un proyecto específico. La combinación apropiada de estos factores, junto con el soporte de una buena herramienta *MDA*, logrará finalmente la satisfacción plena de los objetivos planteados por *Model Driven Architecture* en cada proyecto de desarrollo de sistemas informáticos.

Cabe señalar que no se han encontrado en la literatura especializada trabajos cuyo objetivo sea evidenciar la ventaja de usar herramientas MDA. Sin embargo, otros trabajos de comparación más general [15, 16] ofrecen ideas interesantes para este estudio.

En futuros trabajos sería interesante profundizar aspectos tales como la ampliación de la experiencia a aplicaciones de otro dominio y mayor complejidad; la creación de un metamodelo para la capa de presentación independiente de plataforma; y la aplicación del trabajo de evaluación con otras herramientas MDA (androMDA, Borland Together, etc.).

## REFERENCIAS


[1] Object Management Group. MDA Guide Version 1.0.1. (2003) www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf

[2] Kleepe, J. Warmer, W. Bast. MDA Explained. The Model Driven Architecture practice and promise. Addison-Wesley Pearson Education. (2003)

[3] The Middleware Company. Model Driven Development for J2EE utilizing Model Driven Architecture (MDA) Approach. Productivity Analysis. (2003).
www.omg.org/mda_files/MDA_Comparison-TMC_final.pdf

[4] Marquina Muñoz, F. Arquitecturas de Transformación MDA: ArcStyler y OptimalJ. Proyecto Informático de Fin de Carrera. Departamento de Informática y Sistemas, Universidad de Murcia (España). (2005)

[5] Rodríguez, V. J. Ingeniería de modelos con MDA: Estudio comparativo de OptimalJ y ArcStyler. Proyecto Informático de Fin de Carrera. Departamento de Informática y Sistemas, Universidad de Murcia (España). (2004)

[6] ArcStyler (Interactive Objets).
www.interactive-objets.com/products/arcstyler

[7] Lasheras Velazco, J. Cartuchos MDA en ArcStyler 4.0. Curso de Doctorado: Arquitectura del Software. Universidad de Murcia (España). (2004)

[8] Interactive Objects. ArcStyler Accessor Guide for ArcStyler Version 5.5. (2006)

[9] Interactive Objects. ArcStyler Carat Guide for ArcStyler Version 5.5. (2006)

[10] Interactive Objects. ArcStyler Release Notes for ArcStyler Version 5.5. (2006)

[11] Objecteering (Objecteering Software). www.objecteering.com

[12] Enterprise Architect (Sparx Systems). www.sparxsystems.com

[13] Mia-Generation (Mia-Software). www.mia-software.com

[14] OlivaNova (Care Technologies). www.care-t.com

[15] Mellor, S.J., Balcer, M.J.: Executable UML : a foundation for model-driven architecture. Addison-Wesley, Boston ; San Francisco ; New York (2002)

[16] Pastor, O., Molina, J.C.: Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. Springer (2007)



### José Duarte

Ingeniero en Informática en 2007 por la Facultad de Ciencias y Tecnología de la Universidad Católica "Nuestra Señora de la Asunción", Asunción, Paraguay. Desde 2004, se desempeñó como investigador y desarrollador del Laboratorio de Electrónica Digital (LED) de la misma Universidad, etapa en la cual centró sus estudios en las notaciones y métodos para el análisis y diseño de software. Entre 2004 y 2006 fue profesor del Departamento de Ingeniería Electrónica e Informática de la misma Universidad. Profesionalmente a la vez ha trabajado en proyectos de desarrollo de sistemas de gestión, y más recientemente, en el área de *datawarehousing*. Actualmente se desempeña como consultor informático y desarrollador para el gobierno, en el marco del proyecto "Reingeniería del Sistema de Administración Financiera", Programa Umbral, USAID-Ministerio de Hacienda.
Correo e.: joseph.duartev@gmail.com

### Magalí González

Ingeniera en Informática en 2000 por la Facultad de Ciencias y Tecnología de la Universidad Católica "Nuestra Señora de la Asunción" (Asunción-Paraguay). Actualmente, realizando un doctorado en la Universidad Politécnica de Valencia, España. Desde 2002, se desempeña como investigadora y desarrolladora del Departamento de Ingeniería Electrónica e Informática (DEI) de la Universidad Católica "Nuestra Señora de la Asunción" (Asunción-Paraguay), y como docente para la carrera de Ingeniería Informática del DEI. Sus áreas de investigación se centran en la Ingeniería Web, calidad del Software, MDA para entornos Web. Cuenta con varias publicaciones internacionales.
Correo e.:mgonzalez@uca.edu.py

### Luca Cernuzzi

Doctor en Ciencias Informáticas ("Università degli Studi di Milano" - Italia, 1990). PhD en Ingeniería (Università di Modena e Reggio Emilia – Italia, 2007). Desde 1991, es Profesor del Departamento de Ingeniería Electrónica e Informática (DEI) de la Universidad Católica "Nuestra Señora de la Asunción" (Asunción-Paraguay) y actualmente Profesor Titular en la disciplina de Ingeniería de Software. Desde Julio de 1996 es Director del DEI. Sus áreas principales de investigación se centran en: Ingeniería de Software, en particular en las áreas de Ingeniería Web y metodologías para el diseño de Sistemas de Agentes, e Informática Educativa (en particular para niños con necesidades especiales). Cuenta con más de 60 publicaciones internacionales.
Correo e: lcernuzz@uca.edu.py

### Oscar Pastor

Catedrático de Universidad en el Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia, donde es Director del Departamento y Responsable del Grupo de Investigación en Métodos Orientados a Objetos de Producción de Software (OO-METHOD). Autor de más de un centenar de publicaciones de I+D en el ámbito nacional e internacional, y participante en numerosos Comités Científicos relevantes en el area, sus temas de interés incluyen Modelado Conceptual, Generación Automática de Software a partir de Esquemas Conceptuales, Patrones de Análisis y Diseño, UML, Tecnología Software para Ambientes Web y Calidad de Proceso y Producto Software. Sus trabajos han dado lugar a la creación de la empresa CARE Technologies, que comercializa el producto Oliva Nova Model Execution, un Compilador de Modelos Conceptuales .
Correo e: opastor@dsic.upv.es