Propuestas para la Captura de Requisitos y el Modelado de la Interacción en el marco de MDA

Sergio España, José Ignacio Panach, Nathalie Aquino, Francisco Valverde, Óscar Pastor

Centro de Investigación en Métodos de Producción de Software (ProS)
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, España
{sergio.espana, jpanach, naquino, fvalverde, opastor}@pros.upv.es
www.pros.upv.es
Teléfono: +34 96 387 7007, Fax: +34 96 387 7359

Resumen. El desarrollo de software dirigido por modelos (DSDM) se ha posicionado como una técnica relevante para garantizar la calidad del software producido industrialmente. Propuestas como MDA (*Model-Driven Architecture*) son indicadores de dicha aceptación. En este artículo se presentan unas propuestas de extensión para un método MDA de desarrollo de sistemas informáticos de gestión llamado OO-Method, el cual tiene su implementación industrial en la herramienta OLIVANOVA. Las propuestas de extensión se proponen para afrontar distintos retos a la hora de aplicar OO-Method en las distintas etapas del proceso de DSDM. Más concretamente, este artículo se centra en cómo afrontar la captura de requisitos y el desarrollo de interfaces de usuario avanzadas que garanticen la usabilidad. Para cada uno de estos retos, se discute cómo abordarlos desde la perspectiva MDA.

Palabras clave: Desarrollo de software dirigido por modelos, *Model-Driven Architecture*, requisitos, interfaces de usuario, Web, usabilidad.

1 Introducción

A lo largo de la historia de la ingeniería del software se ha ido aumentado el nivel de abstracción que se emplea para representar los sistemas de información (SIs). Se ha pasado del lenguaje ensamblador al lenguaje de orientación a objetos con el fin de facilitar la labor de los analistas y programadores. Siguiendo esta tendencia de abstracción, algunas empresas hoy en día están adoptando el desarrollo de software dirigido por modelos (DSDM) [19] como un nuevo paradigma para la producción de software que facilita aún más su proceso de desarrollo. El paradigma del DSDM se basa en una serie de modelos conceptuales que representan el SI.

A partir de esos modelos es posible obtener el código final de la aplicación mediante una serie de transformaciones de modelo a modelo y de modelo a código.

Para facilitar la incorporación del proceso de DSDM a la industria del desarrollo del software, el *Object Management Group (OMG)* propuso su propio estándar. Este estándar se conoce con el nombre de *Model-Driven Architecture* (MDA) [18] y actualmente es la aproximación más extendida en el ámbito del DSDM. Tal y como se puede apreciar en la Figura 1 (izquierda), un proceso de desarrollo que cumpla con MDA se estructura en cuatro niveles en los cuales se define un conjunto de modelos conceptuales:

- *Modelo Independiente de Computación* (CIM): representa el entorno y los requisitos del sistema de forma totalmente independiente de cualquier soporte informático.
- *Modelo Independiente de Plataforma* (PIM): en él se especifica el modelo conceptual del sistema que se mantiene constante para cualquier plataforma tecnológica.
- *Modelo Específico de Plataforma* (PSM): el PIM se transforma en uno o varios PSM. El PSM especifica el sistema en términos de una tecnología de implementación específica.
- *Código***:** expresado en un lenguaje de programación específico que implementa el sistema.

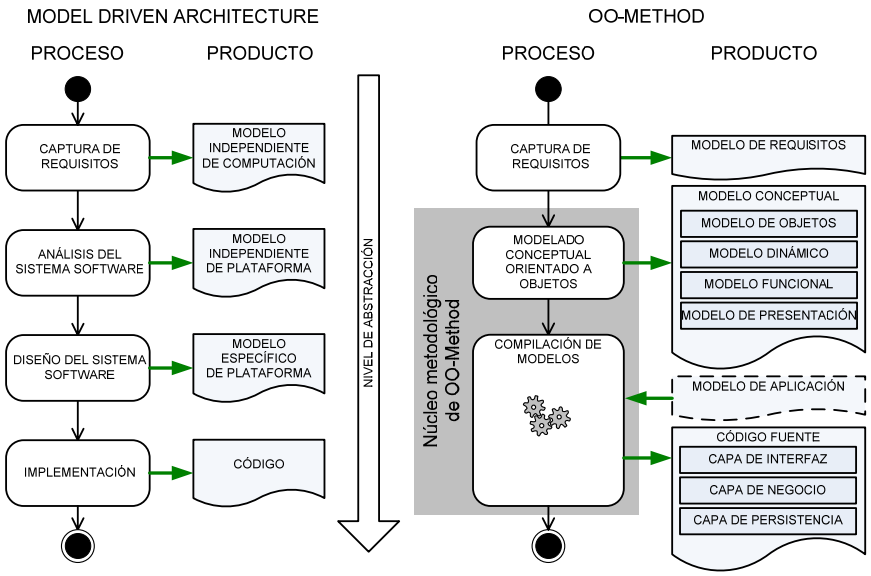


Fig. 1. Modelos genéricos propuestos por MDA y modelos de OO-Method

Actualmente existen varios métodos de DSDM que siguen, en mayor o menor medida, el estándar MDA, como por ejemplo WebML [7]. Existen también herramientas para dar soporte a este estándar, como AndroMDA [1]. En este tra-

bajo hemos optado por enfocarnos a una alternativa MDA llamada OO-Method [24], un método orientado a objetos que permite la generación automática de SIs completos a partir de modelos conceptuales. Esta selección se debe a que, por una parte, OO-Method tiene una herramienta industrial que le da soporte, OLIVANOVA [6], y por otra parte, porque OLIVANOVA es capaz de generar código totalmente funcional a partir de modelos conceptuales. Esto es posible ya que OLIVANOVA incluye un compilador de modelos que realiza las transformaciones de modelo a código de forma totalmente automática. Por lo tanto, OO-Method es un claro ejemplo de que los métodos MDA tienen una aplicación útil en el mundo industrial.

Las distintas fases en las cuales se basa OO-Method y su analogía con MDA están representadas en la Figura 1 (derecha):

- *Captura de Requisitos*: este proceso implica la elaboración de una serie de modelos que capturan los requisitos del sistema.
- *Modelado Conceptual Orientado a Objetos*: este proceso consiste en especificar un SI a partir de los requisitos capturados, e implica la derivación de cuatro modelos: (1) *Modelo de Objetos*: especifica la estructura de las clases identificadas en el dominio del problema; (2) *Modelo Dinámico*: determina las posibles secuencias de eventos que pueden ocurrir en la vida de los objetos; (3) *Modelo Funcional*: especifica el efecto que tienen los eventos sobre el estado de los objetos; (4) *Modelo de Presentación* [20]: ofrece una descripción abstracta de la interfaz del sistema.
- *Compilación de Modelos*: este proceso consiste en generar el código a partir de los modelos conceptuales. La generación se hace en base a reglas de transformación de modelo a código.

Actualmente, debido a carencias que se han identificado, estamos trabajando en mejoras relacionadas con las fases de captura de requisitos y de diseño de la interfaz de usuario en el marco del modelado conceptual. Para el proceso de captura de requisitos, estamos proponiendo mecanismos para la especificación de requisitos de interacción que se relacionan con los requisitos funcionales ya existentes. Por otro lado, en el proceso de modelado conceptual, estamos proponiendo extensiones para representar características de las interfaces que dependan de la plataforma destino sobre la que se implementará el sistema. Además, estamos trabajando en enriquecer el modelado conceptual con nuevas primitivas para representar características de usabilidad de forma abstracta. Estas extensiones pretenden hacer frente a retos relacionados a la captura de requisitos y al modelado de la interacción en las aproximaciones MDA, y además conforman las bases de las contribuciones de este artículo, que son las siguientes:

- Se presenta una propuesta para la captura de los requisitos de interacción de los SIs. La propuesta pretende integrarse con la captura de requisitos funcionales ya existente. .
- Se presentan propuestas para la adecuación del proceso de generación de interfaces de usuario en base al contexto.
- Se presenta una propuesta para tratar la usabilidad en todo el proceso de desarrollo del software, desde la captura de requisitos hasta la generación de código.

Todas estas contribuciones se abordan y se presentan desde la perspectiva de OO-Method, pero son perfectamente extrapolables a otros métodos MDA que trabajen con modelos del sistema en distintos niveles de abstracción.

El resto del artículo se estructura con las siguientes secciones: la Sección 2 comenta la captura de requisitos en MDA y las diversas opciones propuestas por OO-Method; la Sección 3 presenta el desarrollo de interfaces de usuario con MDA y las extensiones que se están proponiendo para OO-Method; la Sección 4 aborda la usabilidad desde una perspectiva MDA y la extensión para incorporarla a OO-Method. Por último, la Sección 5 presenta las conclusiones.

2 MDA y la Captura de Requisitos para Sistemas de Información

Los SIs son, eminentemente, sistemas de soporte a las comunicaciones organizacionales [17]. Su informatización entraña a menudo cierta complejidad debido a su tamaño y a factores humanos (p.e. múltiples participantes cuyas visiones entran en conflicto). La ingeniería de requisitos proporciona herramientas claves para el análisis de las necesidades de la organización.

Existen diversas aproximaciones a la captura de requisitos. Podemos encontrar propuestas basadas en metas (p.e. i* [29]); propuestas basadas en el análisis ontológico del universo de discurso, es decir, de la organización y su entorno (p.e. el clásico diagrama entidad-relación y, en general, las orientaciones a objeto); propuestas basadas en el intercambio de objetos de valor (p.e. e3-value [10]); propuestas orientadas a aspectos (p.e. Theme/Doc [3]); etc.

La fase de captura de requisitos estaría dentro del nivel CIM de MDA. Ahora bien, históricamente este nivel se ha dividido en dos subgrupos en los entornos MDA: requisitos funcionales, relacionados con la lógica de negocio, y requisitos de interacción, relacionados con la parte visual de los sistemas. Particularmente para OO-Method (Figura 2), los requisitos funcionales se capturan mediante un árbol de refinamiento de funciones basado en casos de uso y diagramas de secuencia. La captura de los requisitos funcionales ya lleva tiempo incorporada en el proceso de desarrollo de OO-Method, en cambio, no existía ningún mecanismos para capturar los requisitos de interacción. Para suplir esta carencia, en este trabajo proponemos utilizar diagramas de tareas llamados *Concur-Task Trees* (CTT) [25] para la captura de requisitos de interacción. A continuación se brindan más detalles de cómo se capturan los requisitos funcionales y los de interacción en OO-Method.

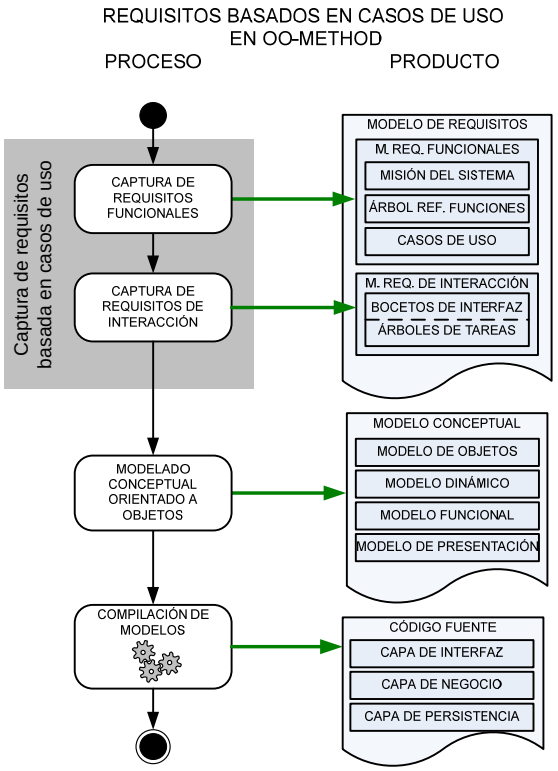


Fig. 2. Proceso de generación de código en OO-Method

El *Modelo de Requisitos Funcionales* se encarga de capturar los requisitos relacionados con la lógica de negocio y la persistencia de información. Este modelo está formado por las siguientes vistas [12]:

- La *Misión del Sistema* es una descripción concisa del propósito del sistema en construcción.
- El *Árbol de Refinamiento de Funciones* descompone jerárquicamente las funciones del negocio. Cada hoja del árbol se considera un caso de uso.
- El *Modelo de Casos de Uso* define las relaciones entre los actores del sistema y los casos de uso.
- Las *Plantillas* de casos de uso describen los flujos de tareas asociadas a cada caso de uso, así como otros detalles del caso.
- Los *Diagramas de Secuencia* describen la interacción del usuario con el sistema y ya supone una preconcepción de las clases de objetos en que se descompondrá el sistema. Por esto, se trata de una vista del Modelo de Requisitos Funcionales que pertenece al nivel del PIM.

A partir de las vistas que capturan los requisitos funcionales se puede derivar parte del Modelo de Objetos perteneciente al Modelo Conceptual (Figura 3). Para

ello se han definido transformaciones modelo a modelo. Por ejemplo, cada caso de uso se corresponde con un servicio de una clase del modelo de objetos.

Por otro lado, el *Modelo de Requisitos de Interacción* es una de nuestras propuestas de extensión de OO-Method apropiada para aquellos casos de uso que requieren una interacción compleja con el sistema. Se compone de dos vistas interrelacionadas [22]:

- Los *Árboles de Tareas* se basan en la notación CTT [25] para describir con detalle la interacción del usuario. Cada árbol de tareas representa la interacción que se lleva a cabo en un caso de uso. Existe una equivalencia de un árbol de tareas por cada una de las interfaces que tendrá el sistema. Como desventaja del uso de CTTs, cabe destacar la complejidad que conlleva el uso de la notación. Casos de uso muy sencillos requieren de árboles de tareas muy grandes que son inmanejables por su tamaño y complejidad. Para facilitar la construcción de los árboles CTT, se ha propuesto el uso de bocetos de interfaz como una capa más abstracta para la captura de requisitos de interacción.
- Los *Bocetos de Interfaz* ofrecen una primera versión bosquejada de la futura interfaz gráfica de usuario. El analista se encarga de dibujar los bocetos de la interfaz mientras que de forma totalmente oculta para el analista, los árboles CTT se van construyendo de manera sincrónica. Para ello se han definido una serie de transformaciones que generan los CTTs a partir de los bocetos.

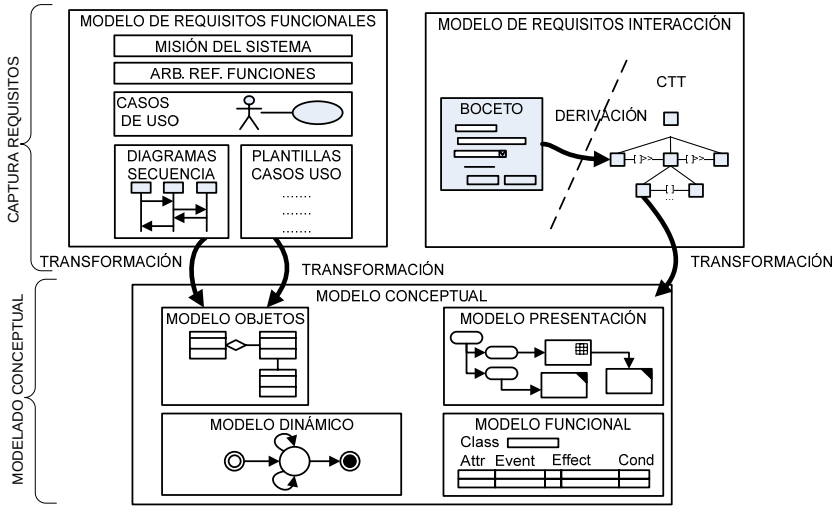


Fig. 3. Transformaciones a partir de las capturas de requisitos en OO-Method

A simple vista, parece lógico pensar que bastaría con utilizar solamente los bocetos para capturar los requisitos de interacción. Sin embargo, el uso de CTTs como notación formal aporta la ventaja de que los modelos se pueden validar. Esta validación no se puede llevar a cabo a nivel de boceto. Además, se pueden definir

reglas de transformación modelo a modelo para generar parte del Modelo de Presentación de OO-Method (Figura 3) a partir de árboles CTT.

La Figura 3 representa gráficamente las transformaciones que se hacen desde las plantillas de los casos de uso y los diagramas de secuencia para generar parte del Modelo de Objetos. Además, muestra la derivación de árboles CTT (de manera oculta para el analista) a partir de bocetos y la transformación de dichos árboles a parte del Modelo de Presentación.

En cualquier caso, sigue siendo un reto para la comunidad de ingeniería de requisitos, ofrecer y perfeccionar métodos de captura de requisitos que combinen una robustez teórica, gran expresividad y facilidad de uso.

3 Desarrollo de Interfaces de Usuario con MDA

Los métodos tradicionales de desarrollo de interfaces de usuario implican habitualmente un trabajo artesanal que es llevado a cabo, típicamente, por los mismos programadores de la lógica y persistencia de los SIs. Por tanto, la calidad de las interfaces de usuario resultantes es altamente dependiente de la habilidad y experiencia de los programadores.

La calidad de las interfaces de usuario es un factor determinante tanto para la implantación exitosa de un SI como para la aceptación y satisfacción del usuario final. Además, un alto porcentaje de la actividad de desarrollo de software se invierte en el diseño e implementación de las interfaces de usuario. Dicho porcentaje es especialmente alto cuando un SI debe proveer interfaces de usuario multiplataforma. Por tanto, se puede afirmar que el desarrollo de las interfaces de usuario precisa de un proceso bien definido, automatizable y reusable que asegure la calidad.

La aplicación de MDA al desarrollo de interfaces de usuario sienta las bases para dicho proceso. Por una parte, ayuda a disminuir la variabilidad que introduce la experiencia de los programadores. Por otra parte, introduce un proceso automatizable en el cual es posible introducir buenas prácticas de desarrollo de interfaces de usuario. Estas prácticas se aplicarán automáticamente al utilizarse el método y redundarán en beneficio de las interfaces de usuario generadas.

Diversas aproximaciones se han propuesto para el desarrollo de interfaces de usuario dirigido por modelos (Teallach [11], TERESA [21], USIXML [16], entre otras). De manera general, las diversas aproximaciones utilizan todos o algunos de los modelos que se ilustran en la Figura 4.

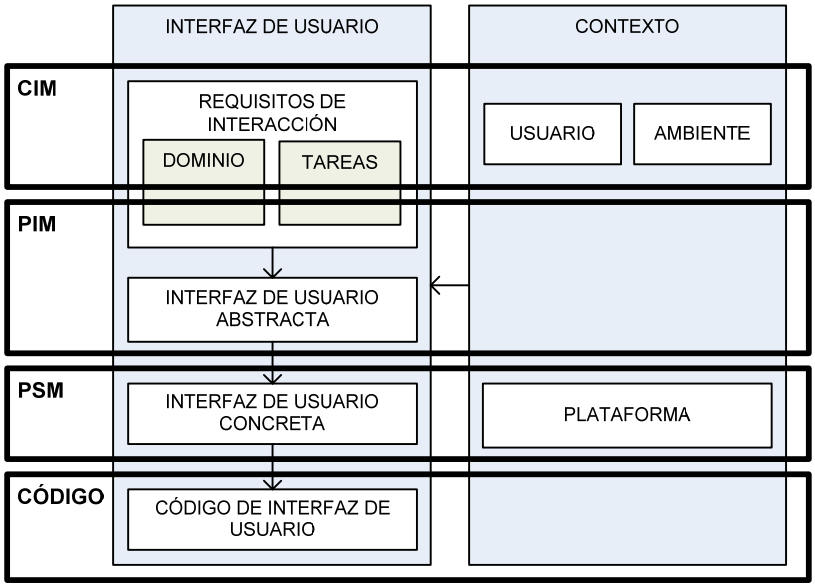


Fig. 4. Esquema del desarrollo de interfaces de usuario dirigido por modelos

Un modelo de interfaz de usuario típicamente incluye modelos de *dominio* y de *tareas*. Estos modelos describen las tareas que deben ser realizadas por los usuarios finales de un sistema interactivo y los conceptos del dominio relacionado a esas tareas. También se cuenta con el modelo de *interfaz de usuario abstracta*, el cual es independiente de la tecnología de implementación. Por otra parte, se define el modelo de *interfaz de usuario concreta*, en el que se afinan los detalles para una plataforma en concreto. Finalmente, se tiene el *código de la interfaz de usuario* en algún lenguaje de programación o de marcado [28].

Por otra parte, las aproximaciones de desarrollo de interfaces de usuario dirigidas por modelos también suelen especificar modelos del *contexto* de uso de un SI, a fin de que las interfaces resultantes sean apropiadas para dicho contexto. La especificación del contexto implica la descripción de los *usuarios* finales del sistema, y del *ambiente* y las *plataformas* hardware y software en las que el SI será utilizado.

En el caso específico de OO-Method, el desarrollo de las interfaces de usuario implica la elaboración de los modelos que capturan los requisitos de interacción (ver Sección 2), y la definición del *Modelo de Presentación* [20]. Este modelo está basado en patrones de interfaces de usuario y permite especificar una interfaz de manera abstracta en relación al contexto de uso de un SI. Por lo tanto, el Modelo de Presentación se corresponde a un modelo de interfaz de usuario abstracta.

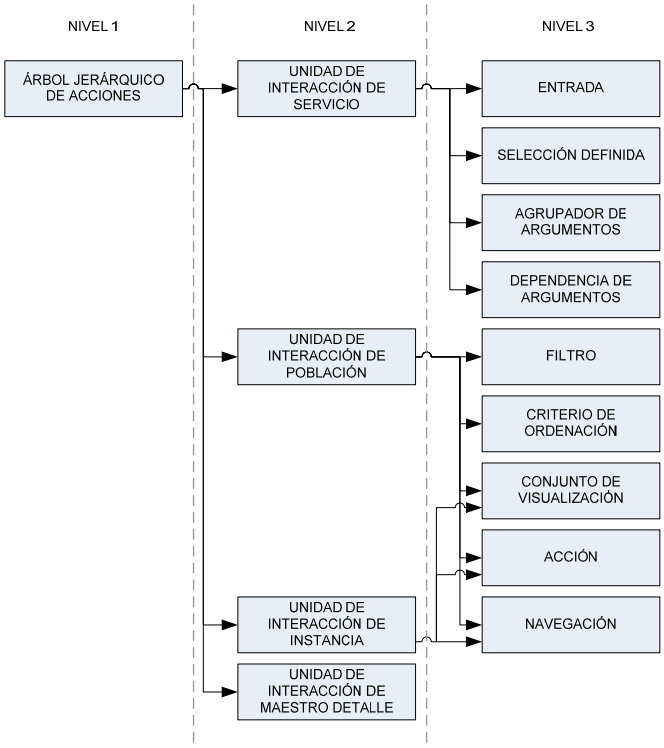


Fig. 5. Modelo de Presentación de OO-Method

La Figura 5 ilustra el Modelo de Presentación de OO-Method. El mismo se encuentra estructurado en tres niveles. En el primer nivel, el *Árbol Jerárquico de Acciones* define la estructura de acceso a la funcionalidad del sistema. En el segundo nivel se definen las unidades de interacción. Una unidad de interacción describe un escenario particular de la interfaz a través del cual los usuarios serán capaces de realizar tareas específicas. OO-Method provee cuatro tipos de unidades de interacción:

- La *Unidad de Interacción de Servicio*: define un escenario de ejecución de servicio.
- La *Unidad de Interacción de Población*: define un escenario apropiado para la manipulación de un conjunto de objetos de una clase.
- La *Unidad de Interacción de Instancia*: define un escenario de manipulación de un único objeto.
- La *Unidad de Interacción de Maestro Detalle*: define un escenario para la interacción con múltiples colecciones de objetos pertenecientes a diferentes clases relacionadas.

En el tercer nivel se encuentran los elementos básicos a partir de los cuales se construyen las unidades de interacción. Por ejemplo, una unidad de interacción de

población debe incluir mecanismos apropiados para seleccionar y ordenar objetos (elementos básicos *Filtro* y *Criterio de Ordenación*), para elegir la información y los servicios a ser desplegados (*Conjunto de Visualización* y *Acción*), y para listar otros escenarios a los que se podrá acceder a partir del actual (*Navegación*). En [20] y [24] se pueden encontrar más detalles sobre los patrones de los tres niveles del Modelo de Presentación de OO-Method.

Hoy en día, OLIVANOVA genera el código de las interfaces de usuario directamente partiendo del Modelo de Presentación, y no se define de manera explícita un modelo de interfaz de usuario concreta, así como tampoco se definen modelos de contexto.

Sin embargo, actualmente estamos trabajando en dos extensiones para OO-Method a fin de poder adaptar las interfaces a contextos específicos. Por lo tanto, estas extensiones están relacionadas a la definición de la interfaz de usuario concreta y de los modelos de contexto. Estas extensiones se describen brevemente a continuación.

3.1 Plantillas de Transformación

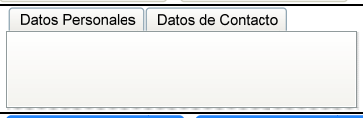
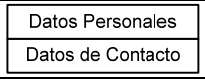
Algunas de las aproximaciones actuales para el desarrollo de interfaces de usuario dirigido por modelos utilizan herramientas de generación en las que el conocimiento de diseño y las guías de presentación están implícitos en el código mismo de las herramientas (p.e. Teallach [11], TERESA [21]). En estos casos, si el diseñador de la interfaz necesita hacer modificaciones, las mismas deberán realizarse de manera manual sobre el código generado. Otras aproximaciones, hacen que el conocimiento de diseño y guías de presentación sean explícitos, y los representan utilizando modelos de correspondencias y de transformaciones (p.e. USIXML [16]). En este caso, si se necesitan modificaciones, el diseñador debe editar las reglas de transformación existentes o crear nuevas, y ese es un proceso relativamente complejo, más apropiado para un especialista en transformaciones de modelos que para un diseñador de interfaces de usuario. Por otra parte, también existen aproximaciones que se basan en plantillas (p.e. CSS [5]), pero éstas, típicamente, se limitan a especificar valores de atributos físicos de un elemento de la interfaz (p.e. el color de los botones).

A fin de proveer una aproximación en la que el conocimiento de diseño y las guías de presentación sean explícitos, fácilmente configurables por los diseñadores, y que no se limiten a especificar atributos de un elemento de interfaz, estamos trabajando con la especificación de Plantillas de Transformación [2]. Estas plantillas tienen el objetivo de especificar aspectos concretos de las interfaces de usuario en base al contexto en el que las mismas serán utilizadas.

Una Plantilla de Transformación está compuesta por parámetros que tienen un valor que define un aspecto de la estructura, disposición o estilo de la interfaz de usuario. Así mismo, cada parámetro tiene un selector que define los elementos del modelo de interfaz sobre los cuales el valor del parámetro tendrá efecto. A manera

de ejemplo, la Tabla 1 presenta los valores posibles para un parámetro de agrupación de elementos de interfaz.

Tabla 1. Valores posibles para el parámetro Agrupador

Valor posible	Descripción gráfica
Cuadros de grupo	
Fichas	
Wizard	
Acordeón	

La incorporación de la aproximación de Plantillas de Transformación a OO-Method implica la incorporación de un Modelo o Plantilla de Transformación que cumpliría las funciones de un modelo concreto de interfaz. Además, como los parámetros y sus valores posibles pueden estar disponibles, o no, en diferentes contextos, se deberán incorporar los modelos relacionados al contexto: usuarios, plataforma y ambiente.

3.2 Interfaces Avanzadas en el marco de las aplicaciones Web

En el campo del desarrollo Web se ha producido en los últimos años una evolución considerable en las interfaces de usuario. Dicha transición ha sido provocada por las denominadas *Rich Internet Applications* (RIA), las cuales han introducido una serie de tecnologías para implementar interfaces de usuario más interactivas y visualmente atractivas. Las RIA han definido una nueva arquitectura en la cual, la interfaz de usuario es implementada mediante tecnologías que son desplegadas en el navegador ya sea mediante plug-ins o scripts de JavaScript. De esta manera ha sido posible evitar el procesamiento de la interfaz de usuario en el servidor Web y proporcionar lenguajes de interfaz de usuario más avanzados que HTML.

Actualmente los métodos dirigidos por modelos no se encuentran adaptados para soportar este nuevo paradigma de aplicación [26]. Fundamentalmente, se detectan dos carencias: 1) el número de componentes gráficos o *widgets* soportados por los modelos de presentación resulta insuficiente; y 2) no es posible definir la respuesta de la interfaz como consecuencia de los eventos producidos por los usuarios.

El problema reside en que dichas carencias deben ser resueltas de manera específica para cada tecnología. Por ejemplo, el conjunto de widgets y eventos que posee la tecnología Adobe Flex dista mucho del proporcionado por los distintos *frameworks* basados en JavaScript. Aunque es posible definir un modelo con las características comunes a cada tecnología, si realmente se quiere hacer uso del verdadero potencial de una tecnología RIA la solución es la de definir un modelo tecnológico específico para cada una.

Sin embargo, a la hora de aplicar esta aproximación surgen diversos problemas. En primer lugar, diversos métodos MDA ya han definido ciertos aspectos relacionados con la interfaz de usuario en otros modelos de carácter más abstracto. Por lo tanto, la nueva expresividad introducida para soportar el desarrollo de RIA debe ser coherente con los modelos previos. Esta situación se produce en el marco de OO-Method.

Al igual que en otros métodos MDA, en OO-Method la interfaz se genera a partir de un modelo de presentación basado en patrones de interfaz abstractos. Para especificar aspectos concretos de una tecnología RIA, en primer lugar deben establecerse las relaciones entre las primitivas de los modelos OO-Method actuales y del modelo RIA seleccionado. Por ejemplo, si definimos en OO-Method un patrón de interfaz para la recuperación de información, debemos modelar qué widget RIA soporta en la interfaz dicha interacción. El problema reside en que dado un modelo abstracto de la interfaz de usuario, existe un gran número de transformaciones modelo-a-modelo posibles. Por ejemplo en la tecnología Adobe Flex, el patrón de recuperación de información de OO-Method puede ser representado utilizando quince widgets diferentes. El widget más adecuado depende de las necesidades de información, de usabilidad o de los requisitos definidos, pero en un principio no puede definirse una transformación por defecto. Para agravar más si cabe dicha situación, este conjunto de transformaciones debe definirse para cada par método-tecnología RIA.

La solución planteada en el marco de OO-Method, la cual puede ser extendida a otros métodos MDA, es la de aplicar el concepto de *metamodel weaving* [8]. Dicho metamodelo se encarga de definir las relaciones entre el metamodelo de OO-Method actual y un metamodelo para el desarrollo de RIA. De esta manera cuando se crea un *model weaving* se asocia a cada primitiva del modelo OO-Method una primitiva de un modelo RIA. Por lo tanto, se delega al analista la tarea de seleccionar los componentes que son más adecuados para la interfaz RIA y se evita la definición de transformaciones. A la hora de generar el código de la interfaz, los tres modelos son tenidos en cuenta a la vez. De esta forma se evita tener que especificar un nuevo metamodelo en el método, que tenga que soportar tanto la expresividad actual como la necesaria para el desarrollo de RIA. Esta solución, además, permite tener en cuenta las ventajas específicas que cada tecnología RIA ofrece al desarrollo de interfaces.

4 Modelado de la Usabilidad en Entornos MDA

Históricamente, los trabajos de la comunidad de la ingeniería del software se han centrado en modelar aspectos de persistencia y funcionalidad, dejando en un segundo plano aspectos de interacción y en especial aspectos de usabilidad. La ISO 9126-1 [13] define la usabilidad como *la capacidad del producto software de ser entendido, aprendido y usado por el usuario bajo condiciones específicas*. Además, la ISO 9126-1 declara que la usabilidad es una de las características de la calidad del software. Es decir, para conseguir desarrollar software de calidad, es necesario que éste sea usable.

El vacío de propuestas para tratar la usabilidad por parte de la comunidad de la ingeniería del software ha sido cubierto por los trabajos de la comunidad de interacción persona-ordenador. Esta comunidad ha propuesto una serie de recomendaciones para mejorar la usabilidad de los sistemas. Las recomendaciones se pueden clasificar en tres grupos [14]: con impacto en la interfaz; con impacto en el proceso de desarrollo; y con impacto en el diseño arquitectural.

De los tres tipos de recomendaciones, el último es el que sería más interesante tratar dentro de un entorno de desarrollo MDA, ya que afecta a la arquitectura del sistema. En la literatura, este tipo de recomendaciones se llaman FUF (*Functional Usability Features)* [15]. Los FUFs deben estar presente desde las primeras etapas del proceso de desarrollo a fin de evitar el mayor número posible de modificaciones de la arquitectura del sistema, tal y como proponen Bass [4] y Folmer [9]. Por ejemplo, la incorporación de FUF *deshacer la última acción realizada*, implica modificaciones profundas en el diseño arquitectural del sistema. Por ejemplo, en el código fuente se deben añadir clases y librerías que soporten la funcionalidad. Cuanto antes se detecte la necesidad de estas clases y se defina cómo se incorporan en la arquitectura, menos cambios habrá que hacer en el diseño.

El principio de tratar la usabilidad en fases tempranas del desarrollo es perfectamente aplicable a MDA. De esta manera, la incorporación de FUFs a los sistemas se beneficiaría de algunas de las ventajas del paradigma MDA:

- La incorporación de FUFs a un sistema implica normalmente mayor carga de trabajo para el analista. En un proceso MDA, esta carga de trabajo se reduce. El analista sólo tendría que dedicar mayor esfuerzo a la fase de modelado conceptual, ya que la generación de código se puede automatizar con un compilador de modelos que realice las transformaciones de modelo a código.
- Hay errores de usabilidad que sólo se detectan tras la implementación. En un proceso MDA, este tipo de errores se podría solucionar modificando sólo el modelo conceptual. Una vez corregidos, el sistema podría volver a generarse automáticamente en base al nuevo modelo conceptual.

A la hora de incorporar la usabilidad en un entorno MDA, ésta afecta a varios niveles:

- *A nivel de CIM***:** se deben definir elementos para capturar los requisitos de usabilidad. Por ejemplo, se deben detectar las interfaces que necesitan la función de *deshacer* según los requisitos del usuario.

- *A nivel de PIM*: se deben modelar las características de la usabilidad de forma lo suficientemente abstracta como para no depender de la plataforma destino. Por ejemplo, se indica en el modelo conceptual las interfaces que tendrán la función *deshacer*.
- *A nivel de PSM*: se deben especificar las características de usabilidad dependientes de la plataforma. Por ejemplo, la característica que determina la forma de acceso a la funcionalidad deshacer es dependiente de la plataforma destino. El acceso a la funcionalidad *deshacer* puede ir implícita en el navegador para el caso de las aplicaciones Web, pero no para las aplicaciones de escritorio.

Se han hecho propuestas recientemente para incorporar características de usabilidad (FUFs) a OO-Method [23]. Estas propuestas abarcan desde la captura de requisitos hasta la generación de código. Para la captura de requisitos (a nivel CIM) se propone utilizar las guías de captura de requisitos textuales definidas por Juristo [15], por lo tanto, no es necesario incorporar cambios a OO-Method. En cambio, a nivel PIM y PSM sí que es necesario incorporar cambios. Se deben enriquecer los modelos conceptuales con nuevas primitivas que representen la usabilidad de forma abstracta. Por ejemplo, para incorporar la característica de usabilidad de *deshacer* se deberían hacer los siguientes cambios:

- *En el Modelo de Objetos* [24]: se debe añadir una nueva primitiva conceptual para representar los servicios que tendrán la capacidad de deshacer sus acciones.
- *En el Modelo de Presentación* (Figura 5): se deben añadir nuevas primitivas conceptuales para representar de forma abstracta dónde se ubicará la funcionalidad de deshacer.

Además, es necesario modificar el compilador de modelos para que reconozca las nuevas primitivas conceptuales añadidas al modelo conceptual y genere el código que las implemente.

Se puede apreciar que la propuesta para incorporar la usabilidad en un entorno MDA es costosa y requiere cambios desde la fase de captura de requisitos hasta la fase de generación de código. En cambio, una vez incorporados los cambios, se pueden generar aplicaciones usables y por lo tanto, aplicaciones de mayor calidad que mejorarán la satisfacción del usuario, su efectividad y su eficiencia.

5 Conclusiones

No cabe duda que MDA es una de las aproximaciones más aceptadas y prometedoras para mejorar el desarrollo de software. Sin embargo, aún existen algunos retos por cubrir en este dominio, como la conjugación de los requisitos de interacción con requisitos funcionales. Muchas de las propuestas que existen hoy en día para capturar requisitos tratan de forma totalmente independiente los requisitos funcionales de los requisitos de interacción, cuando para el usuario final no existe

dicha distinción. Además, para el usuario es más fácil participar en la captura de requisitos mediante bocetos que mediante otras técnicas como los casos de uso.

Otro de los grandes retos de las propuestas MDA es el modelado conceptual para representar la interacción y la usabilidad de los sistemas. No existe un modelo en UML [27] con el cual representar estos aspectos. Normalmente, la ingeniería del software ha basado sus esfuerzos en el modelado de la persistencia y de la funcionalidad, dejando de lado la parte visual del sistema. Sin embargo, hoy en día se ha demostrado que la calidad del software depende en gran medida no sólo de aspectos funcionales, sino también de aspectos de interacción [13]. En este trabajo se ha presentado una serie de propuestas para tratar el modelado de la interfaz en plataformas de desarrollo innovadoras como pueden ser las RIAs, así como para incorporar la usabilidad dentro del proceso MDA.

Como ejemplo para aplicar todas estas ideas, hemos elegido OO-Method, un método de desarrollo MDA que tiene su aplicación industrial en la herramienta OLIVANOVA. Por cada uno de estos retos se han descrito, brevemente, propuestas para abordarlos. Las propuestas han sido presentadas en términos de OO-Method, aunque también pueden ser aplicables a otros métodos MDA de desarrollo de software.

Referencias

[1] AndroMDA, http://www.andromda.org/, Última visita: Septiembre 2009.

[2] Aquino, N., Vanderdonckt, J., Valverde, F., and Pastor, O. Using Profiles to Support Model Transformations in the Model-Driven Development of User Interfaces. In *Computer-Aided Design of User Interfaces VI, Proc. of 7th Int. Conf. on Computer-Aided Design of User Interfaces CADUI2008, (June 11-13, 2008, Albacete, Spain)* (2008), V. Lopez Jaquero, F. Montero Simarro, J. Molina Masso, and J. Vanderdonckt, Eds., Springer, pp. 35–46.

[3] Baniassad, E. and S. Clarke (2004). Theme: an approach for aspect-oriented analysis and design. Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), IEEE Computer Society: 158-167.

[4] Bass, L., Bonnie, J.: Linking usability to software architecture patterns through general scenarios. The journal of systems and software 66 (2003) 187-197

[5] Bos, B., Çelik, T., Lie, H. W., and Hickson, I. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Tech. rep., World Wide Web Consortium (W3C), July 2007.

[6] CARE Technologies S.A. www.care-t.com . Última visita: junio 2009.

[7] Ceri, S. Fraternali, P., Bongio, et al. (2003). Designing Data-Intensive Web Applications.Morgan Kaufman

[8] Fabro, M.D.D., Valduriez, P.: Semi-automatic model integration using matching transformations and weaving models. Proceedings of the 2007 ACM symposium on Applied computing. ACM, Seoul, Korea (2007)

[9] Folmer, E., Bosch, J.: Architecting for usability: A Survey. Journal of Systems and Software, Vol. 70 (1) (2004) 61-78

[10] Gordijn, J., Wieringa, R.J.: A value-oriented approach to e-business process design. 15th Conference on Advanced Information Systems Engineering. Klagenfurt, Austria, Springer, pp. 390--403 (2003)

[11] Griffiths, T., Barclay, P. J., Paton, N. W., McKirdy, J., Kennedy, J. B., Gray, P. D., Cooper, R., Goble, C. A., and da Silva, P. P. Teallach: a Model-Based User Interface Development Environment for Object Databases. *Interacting with Computers 14*, 1 (2001), 31–68.
[12] Insfrán, E., Pastor, Ó. and Wieringa, R. (2002). "Requirements engineering-based conceptual modelling." Requirements Engineering 7(2): 61-72.
[13] ISO/IEC 9126-1 (2001): Software engineering - Product quality - 1: Quality model.
[14] Juristo, N., Moreno, A.M., Sánchez, M.I.: Analysing the impact of usability on software design. Journal of Systems and Software, Vol. 80 (2007) 1506-1516
[15] Juristo, N., Moreno, A.M., Sánchez, M.I.: Guidelines for Eliciting Usability Functionalities. IEEE Transactions on Software Engineering, Vol. 33 (2007) 744-758
[16] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L. and López-Jaquero, V. USIXML: A Language Supporting Multi-path Development of User Interfaces. In Bastide, R., Palanque, P., and Roth, J., editors, *EHCI/DS-VIS 2004*, vol. 3425 of *LNCS*, pp. 200–220. Springer, 2004.
[17] Lockemann, P.C., Mayr H.C.: Information System Design: Techniques and Software Support. In: Kugler, H.-J. (ed.) IFIP 86. North-Holland, Amsterdam (1986)
[18] MDA Guide V1.0.1: http://www.omg.org/docs/omg/03-06-01.pdf, Última visita: junio 2009
[19] Mellor, S.J., Clark, A.N., Futagami, T.: Guest Editors' Introduction: Model-Driven Development. IEEE Software, Vol. 20 (2003) 14-18
[20] Molina, P.J.: Especificación de interfaz de usuario: de los requisitos a la generación automática.: Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, Valencia (2003) 382
[21] Mori, G., Paternò, F., and Santoro, C. Tool Support for Designing Nomadic Applications. In *IUI '03: Proceedings of the 8th International Conference on Intelligent User Interfaces*, pp. 141–148, New York, NY, USA, 2003. ACM.
[22] Panach, J. I., España, S., Pederiva, I. and Pastor, O. (2007). OO-Sketch: una herramienta para la captura de requisitos de interacción. X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS 2007), Isla Margarita, Venezuela.
[23] Panach, J.I., España, S., Moreno, A., Pastor, Ó. Dealing with Usability in Model Transformation Technologies. ER 2008. Springer LNCS 5231, Barcelona (2008) 498-511.
[24] Pastor, O., Molina, J.: Model-Driven Architecture in Practice. Springer, Valencia (2007)
[25] Paternò F, Mancini C, et al. (1997). ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In Proc. of the IFIP TC13 International Conference on Human-Computer Interaction, Chapman & Hall, Ltd.: 362-369.
[26] Preciado, J.C., Linaje, M., Sánchez, F., Comai, S.: Necessity of methodologies to model Rich Internet Applications. 7th IEEE International Symposium on Web Site Evolution. IEEE, Budapest, Hungary (2005) 7-13
[27] UML: http://www.uml.org/, Última visita: Septiembre 2009.
[28] Vanderdonckt, J. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008, (Iasi, 18-19 September 2008)* (2008), S. Buraga and I. Juvina, Eds., Matrix ROM, Bucarest, pp. 1–10.
[29] Yu, E. and J. Mylopoulos (1994). From E-R to "A-R" - Modelling strategic actor relationships for business process reengineering. Proceedings of the13th International Conference on the Entity-Relationship Approach. Manchester, Springer-Verlag: 548-565.