Una Arquitectura para la Definición de Metáforas Gráficas para Metamodelos♦

Artur Boronat, Julián Pedrós, José Á. Carsí, Isidro Ramos

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
C/Camino de Vera s/n
E-46071 Valencia- España
{aboronat | jpedros | pcarsi | iramos}@dsic.upv.es

Resumen. El modelado específico a dominio se está consolidando como una forma de desarrollo de software con herramientas como GME (Generic Metamodel Environment) y MetaEdit+, entre otras. La clave del éxito de estas herramientas es la proximidad de la ontología de modelado aportada al problema que intenta solucionar el usuario analista y/o programador, de manera que los modelos definidos son fácilmente interpretables en el dominio del problema. MOMENT es una herramienta que permite definir este tipo de modelos siguiendo una aproximación algebraica. Con el fin de obtener una herramienta de gestión de modelos que utilice metáforas gráficas cercanas al usuario, se presenta una arquitectura que integra MOMENT en herramientas CASE que posean un entorno gráfico, pudiendo así utilizar sus capacidades visuales en la definición de modelos soportados por MOMENT. Con tal fin se indican los elementos gráficos de una herramienta CASE que se asocian con elementos de un metamodelo definido en MOMENT y cómo se almacenan esas asociaciones en el propio repositorio de MOMENT. Se ha realizado una implementación de esta arquitectura para una herramienta CASE concreta, y se presenta cómo se define la vista gráfica de un metamodelo definido algebraicamente utilizando su interfaz.

Palabras clave: gestión de modelos, herramientas CASE, modelado, extensibilidad, especificaciones algebraicas.

1. Introducción

El desarrollo de software implica el estudio del sistema de información que constituye el problema a resolver (espacio del problema), las tecnologías que permiten implementar una solución (espacio de la solución) y el proceso de desarrollo del software que obtiene una solución tecnológica a los requisitos del sistema de información.

♦ Este artículo ha sido financiado por el Proyecto Nacional de Investigación, Desarrollo e Innovación DYNAMICA TIC 2003-07804-C05-01 y el Proyecto Nacional PBC-03-00 de "Metodologías de desarrollo de interfaces de usuario dinámicas".

Desde el punto de vista de la solución, los lenguajes de programación han ido incrementado paulatinamente su nivel de abstracción con el objetivo de aproximarse al espacio del problema. Así pues del lenguaje ensamblador se pasó a los lenguajes procedurales, posteriormente a la tecnología orientada a objeto, desembocando en los lenguajes orientados a aspectos y componentes en estos últimos años. Estos esfuerzos han permitido mejorar el proceso de desarrollo del software incrementando la interoperabilidad, la reutilización de artefactos software y disminuyendo los costes de desarrollo. No obstante, aún no se ha conseguido alcanzar un paradigma que permita representar cualquier problema y que pueda generar su solución de forma automática. Además, difícilmente se alcanzará este punto, puesto que cada dominio dispone de unas propiedades específicas, que se deben tener en cuenta para poder definir el problema con precisión. El hecho de conseguir una tecnología universal suficientemente abstracta como para solucionar cualquier problema equivaldría a afirmar que la lengua "esperanto" será una especie de *lingua franca* en la que nos expresaremos los seres humanos en el futuro. Y esta situación parece improbable puesto que cada lengua posee su riqueza expresiva en el contexto socio-cultural en el que es usada, al igual que cada dominio de problema necesita mantener su vocabulario específico.

Desde el punto de vista del dominio, la Ingeniería del Software también ha conocido una serie de técnicas de modelado que permiten representar el problema aproximándolo hacia el espacio de la solución. Las técnicas de modelado se pueden clasificar en dos tipos, tomando como criterio de decisión la proximidad de la metáfora del modelo respecto al dominio del problema: modelado independiente de dominio y modelado específico a dominio. El modelado independiente de dominio intenta representar cualquier problema mediante una notación gráfica basada en un paradigma determinado, como por ejemplo UML [2] que se centra en el paradigma orientado a objeto. El modelado específico a dominio permite utilizar los conceptos del propio dominio para especificarlo. El problema en ambas aproximaciones es la falta de un soporte formal que garantice un proceso unívoco que permita obtener un producto software consistente y eficiente, y que permita validar los requisitos.

MOMENT es una plataforma [1] que permite definir modelos como especificaciones algebraicas de una teoría expresada en lógica ecuacional condicional [22]. La semántica operacional de estas especificaciones algebraicas se evalúa mediante un sistema de reescritura de términos, lo que permite soportar las siguientes aplicaciones [3]: ejecución (animación) de especificaciones formales, análisis *narrowing*, *model checking* y demostraciones formales, entre otras.

A pesar que el uso de formalismos aporta propiedades ventajosas al proceso de desarrollo del software, usualmente no han tenido éxito fuera del ámbito investigador. Esto es debido al grado de preparación teórica que necesitan los usuarios de herramientas formales.

En este artículo, se presenta un enfoque genérico para integrar MOMENT en una herramienta CASE con un entorno gráfico, utilizando su interfaz como mecanismo para asociar una metáfora visual a un metamodelo descrito mediante una especificación algebraica. Como herramienta CASE para realizar la primera implementación de la arquitectura presentada se ha elegido MS Visio [4], porque proporciona buenas propiedades de extensibilidad, tanto en las facilidades gráficas, ya que dispone de un editor gráfico, como en la funcionalidad interna. De esta manera, el

usuario puede especificar modelos de forma algebraica basándose en una metáfora visual conocida a través de una interfaz sencilla.

En esta sección se ha introducido la problemática tradicional existente en el proceso de desarrollo del software que constituye la motivación de nuestro trabajo y hemos presentado nuestro objetivo. En el apartado 2 se indica la estructura de la plataforma de gestión de modelos MOMENT y los elementos principales que se utilizan en ella. En el apartado 3, se describe los elementos principales del entorno gráfico de una herramienta visual de modelado. En el apartado 4, se indica la arquitectura que hemos utilizado para añadir el soporte formal a una herramienta CASE, identificando las asociaciones entre los elementos de su editor gráfico y los sorts (o nombres de tipos) de la teoría algebraica utilizada en MOMENT, y se describe el mecanismo que permite asociar una metáfora gráfica a un metamodelo especificado algebraicamente en MOMENT. En el apartado 5 se discuten trabajos relacionados, y finalmente, en el apartado 6, se resumen una serie de conclusiones y trabajos futuros.

2. La plataforma MOMENT (MOdel manageMENT)

MOMENT es una plataforma de gestión de modelos implementada algebraicamente usando el modelo computacional λ-cálculo mediante el lenguaje funcional F# [5]. MOMENT utiliza también el entorno CafeObj [6] como sistema de reescritura de términos para realizar transformaciones automáticas de modelos entre diferentes metamodelos (transformaciones intermodelo) o de un mismo metamodelo (transformaciones intramodelo).

La plataforma posee cuatro niveles de abstracción, siguiendo la cultura de metamodelado presentada en el estándar MOF [7], aunque no implementa directamente su lenguaje abstracto para definir metamodelos. Cada nivel está formado por un conjunto de esquemas que a su vez están constituidos por un conjunto de conceptos y propiedades. Los conceptos permiten representar entidades descriptibles en un dominio determinado, y las propiedades establecen dichas descripciones, pudiendo asociar dos conceptos mediante una propiedad o bien describir un concepto mediante una propiedad que contiene un valor básico.

Los niveles que consituyen la plataforma MOMENT son los siguientes:

- Nivel M3: contiene los elementos (conceptos y propiedades) necesarios para poder describir cualquier metamodelo en el siguiente nivel. Es el nivel más abstracto de la plataforma.
- Nivel M2: sus esquemas constituyen metamodelos, donde sus respectivos conceptos y propiedades indican como definir un modelo en el siguiente nivel.
- Nivel M1: sus esquemas representan modelos de un determinado metamodelo del nivel M2. Los conceptos y propiedades que forman un modelo permiten definir información en el siguiente nivel.
- Nivel M0: sus esquemas representan instanciaciones de modelos del nivel M1 que contienen datos concretos.

3. Elementos principales del entorno visual de herramientas CASE de modelado

El área de trabajo de las herramientas CASE de modelado visual tienen cuatro elementos en común, que son descritos a continuación:

- *Figuras*: Símbolos que pueden ser prediseñados o pueden ser de creación propia, ajustándose a las necesidades del modelo que el usuario necesita representar. Dichos símbolos pueden contener información de valor añadido mediante propiedades.
- *Hoja de dibujo*: Zona de trabajo donde se depositan (pegan) las figuras, constituyendo un diagrama o modelo gráfico concreto.
- *Primitivas gráficas*: Distintos tipos de figuras que se pueden definir en la hoja de dibujo de la herramienta CASE. Una primitiva gráfica permite definir una figura en la hoja de dibujo. Poseen unos atributos o propiedades por defecto que definen las características de sus figuras.
- *Vistas gráficas*: Plantillas que agrupan a las primitivas gráficas según el modelo o tipo de diagrama al que pertenecen, encapsulando la ontología gráfica que permite definir un modelo.

MS Visio 2003 es una herramienta visual de modelado fácilmente personalizable y extensible mediante la tecnología .NET. En el vocabulario propio de la herramienta, los elementos que hemos identificado anteriormente de forma genérica reciben los siguientes nombres: las figuras son denominadas *shapes*; la hoja de dibujo *shapesheet*; una primitiva gráfica recibe el nombre de *master*; y la vista que agrupa una serie de primitivas gráficas recibe el nombre de *stencil*.

La personalización del entorno visual de Visio se realiza mediante add-ons, es decir, conjuntos de *stencils* que proporcionan la información que constituye un metamodelo completo. La extensión de la herramienta se consigue mediante add-ins, que son módulos que permiten añadir funcionalidad a la herramienta. Dada la facilidad de extensión aportada por Visio 2003, ha sido la herramienta escogida para construir la interfaz gráfica de usuario de MOMENT.

4. Arquitectura de la solución

Partiendo de la plataforma formal de gestión de modelos MOMENT, se ha desarrollado un módulo, llamado *MOMENT Integrator*, que permite asociar una metáfora gráfica a un metamodelo formal mediante la interfaz de herramientas CASE con entornos visuales que proporcionen cierto grado de extensibilidad, como MS Visio 2003.

4.1. Arquitectura

En la Fig. 1, se muestra la arquitectura que permite integrar la plataforma MOMENT en una herramienta CASE con un entorno gráfico de forma genérica. Aunque se ha contextualizado su implementación en el entorno Visio 2003, se está pensando en

desarrollar la misma arquitectura para integrar MOMENT en la herramienta de desarrollo Eclipse aprovechando su entorno gráfico Graphical Editor Framework [20]. La arquitectura utilizada es la tradicional separación en tres capas: una interfaz que permite representar gráficamente metamodelos y modelos, la funcionalidad que permite asociar la metáfora gráfica a un metamodelo especificado algebraicamente y la capa de persistencia en la que se almacena toda la información utilizada.

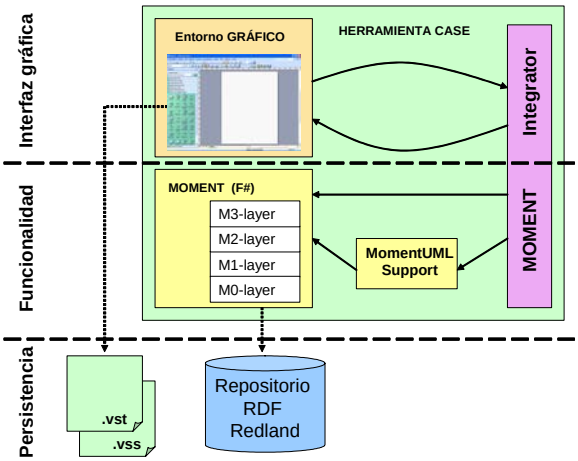

Fig. 1. Arquitectura de la integración de MOMENT en una herramienta CASE.

En la interfaz gráfica, la herramienta CASE ofrece la funcionalidad necesaria para definir los gráficos que se van a utilizar en un metamodelo, y permite acceder a la funcionalidad del módulo *MOMENT Integrator*.

En la capa de funcionalidad, el módulo *MOMENT Integrator* permite definir asociaciones entre los elementos gráficos que se han definido mediante el editor gráfico de la herramienta CASE y los términos algebraicos mediante los cuales MOMENT representa un metamodelo. Como veremos en la siguiente sección, dichas asociaciones se almacenan en la propia plataforma como instancias de clases UML a través de la librería MomentUMLSupport.

En la capa de persistencia se pueden distinguir dos unidades principales de almacenamiento: la de los elementos gráficos de la herramienta CASE y la de la plataforma MOMENT.

Las herramientas CASE suelen almacenar la información gráfica mediante una serie de ficheros. En el caso específico de la herramienta Visio, los modelos gráficos se almacenan mediante dos tipos de ficheros. Los ficheros con extensión *.vss* almacenan los documentos que se crean, o sea el área dibujada con figuras (*shapesheet*), junto con las plantillas o templates a partir de las cuales se ha creado el diagrama o modelo. Los ficheros con extensión *.vst* almacenan las plantillas con *masters* (*stencils*). Por defecto, Visio ya incorpora un gran número de plantillas con todo tipo de formas para todo tipo de modelos, pero un usuario puede crearse y guardar sus propias plantillas con una colección de formas de otras plantillas o con formas creadas por el mismo.

Por otro lado, MOMENT almacena toda la información en un repositorio RDF puesto que los conceptos y propiedades manejados en la plataforma son un reflejo de

los recursos y propiedades RDF. El repositorio utilizado es Redland [19] y permite almacenar la información sobre esquemas de todos los niveles de la plataforma y asociaciones entre elementos gráficos del entorno visual y términos de MOMENT, explotando las facilidades que nos ofrece el entorno gráfico de la herramienta CASE.

4.2. Diseño de la capa de funcionalidad

Mediante el módulo *MOMENT Integrator*, los elementos gráficos de la herramienta CASE se vinculan con los elementos de la plataforma MOMENT. Cómo nuestro objetivo consiste en la definición de la metáfora gráfica asociada a un metamodelo, nos centramos en el uso de los esquemas del nivel M2 de la plataforma, que son los metamodelos. De manera que, un esquema de este nivel se asocia con una vista gráfica (*stencil* en Visio). Por otro lado, para representar gráficamente los conceptos y las propiedades de los esquemas que constituyen un metamodelo en el nivel M2 de la plataforma, hacemos uso de las primitivas gráficas que componen la vista gráfica elegida (*masters* en el contexto de Visio).

Para asociar la información de cada elemento de un metamodelo específico de MOMENT a las figuras gráficas que proporciona el entorno visual, se ha definido una composición, expresada en notación UML en la Fig. 2. En esta composición participan las siguientes clases:

– *GraphicViewWrapper*: es la clase agregada y encapsula información sobre una vista gráfica determinada y sobre el esquema correspondiente del nivel M2 de la plataforma. Es en esta clase, por tanto, donde se asocia el esquema del metamodelo a la plantilla gráfica. Una instancia de la clase *GraphicViewWrapper* está formado por un conjunto de instancias de la clase *GraphicPrimitiveWrapper* que definen las primitivas gráficas del metamodelo.

  La información que contiene esta clase es el identificador o nombre del contenedor, el de la vista gráfica (*stencil* en el contexto de Visio) al que se le asocia el metamodelo junto con la ruta completa donde se encuentra el fichero en el que se almacena la vista gráfica, y finalmente los atributos que identifican la definición del metamodelo en la plataforma, es decir, los identificadores del nivel y del esquema. El tipo de datos *Repository* permite abstraer el mecanismo de almacenamiento de una herramienta CASE específica en el módulo coordinador *MOMENT Integrator*.

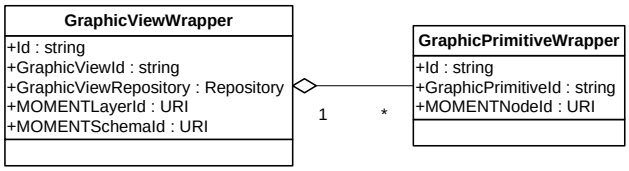

Fig. 2. Diagrama de clases UML que modela las asociaciones de elementos gráficos con elementos de la plataforma MOMENT.

– *GraphicPrimitiveWrapper*: es la clase componente que encapsula la información sobre una primitiva gráfica específica (*master* en el contexto de Visio) y el concepto correspondiente de un metamodelo de la plataforma MOMENT. Este

concepto puede disponer de una serie de propiedades, definidas como términos en MOMENT.

Los atributos de esta clase son un identificador o nombre para el contenedor, el identificador de la primitiva gráfica que pertenece a la vista gráfica especificada y el identificador del nodo MOMENT (concepto o propiedad) que se asocia.

La asociación entre elementos gráficos de la herramienta CASE con elementos de la plataforma definida en el diagrama de clases UML de la Fig. 2, se almacena utilizando la propia plataforma MOMENT como repositorio orientado a objetos, puesto que permite definir modelos UML y sus extensiones.

Para llegar a tal fin, se aprovecha la parte del metamodelo UML, especificado como un esquema en el nivel M2 de la plataforma, que permite la definición de clases y de asociaciones. Por tanto, se ha definido el modelo UML de la Fig. 2 como un esquema del nivel M1 de la plataforma. De esta manera, MOMENT ya dispone de la información necesaria para almacenar instancias de las clases del diagrama de clases definido.

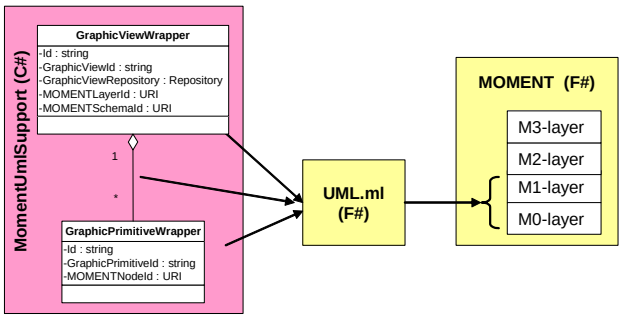

Fig. 3. Estructura de la librería MomentUMLSupport

Para facilitar el uso de MOMENT como repositorio orientado a objetos, se ha desarrollado una API que se comporta como la interfaz directa entre la plataforma de gestión de modelos y el módulo *MOMENT Integrator*. Esta librería, llamada MomentUMLSupport, ha sido desarrollada en C# y utiliza las funciones de un módulo F#, llamado UML.ml, que utiliza la funcionalidad de MOMENT para insertar instancias de las clases de un modelo del nivel M1, en una instanciación de modelo del nivel M0. La Fig. 3 muestra la estructura de la librería MomentUMLSupport.

El API de la librería MomentUMLSupport permite manipular instacias de clases de cualquier modelo UML, involucrando la instanciación, consulta o destrucción de objetos en el nivel M0 de la plataforma. De manera que cuando se instancia una clase mediante esta librería, se está almacenando la información en la plataforma.

4.3. Definición de la metáfora gráfica de un metamodelo

Para construir un metamodelo mediante una herramienta CASE asociamos un metamodelo MOMENT (un esquema del nivel M2) a una vista gráfica. Una vez definido el significado de esa vista, se especifica cada uno de sus primitivas gráficas con conceptos y propiedades de un esquema del nivel M2 de la plataforma, completando la metáfora gráfica asociada al metamodelo. A continuación se detallan

estos dos pasos en el contexto de la implementación que se ha realizado para la herramienta MS Visio 2003:

1. En primer lugar, para definir el *stencil* se accede a la interfaz de definición que se muestra en la Fig. 4, donde se selecciona el *stencil* que se desea asociar y el esquema del nivel M2 de la plataforma que define el metamodelo.

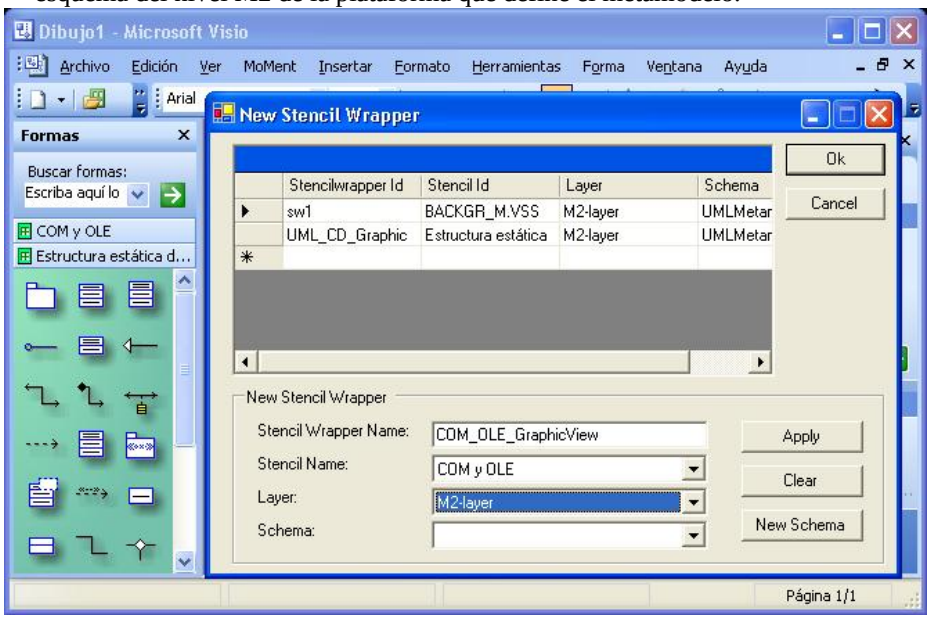

Fig. 4. Creación de un metamodelo mediante VisioMoment

2. Una vez asociada una vista gráfica a un metamodelo MOMENT mediante una instancia *GraphicViewWrapper*, ya sea por haber seguido los pasos anteriores o por haber cargado la plataforma MOMENT de un repositorio Redland, se definen gráficamente los conceptos y propiedades que posee el metamodelo en MOMENT.

   Para ello se accede a la interfaz de la Fig. 5, donde se selecciona un metamodelo definido gráficamente. Acto seguido, se selecciona una primitiva gráfica de la vista seleccionada y un nodo (concepto o propiedad) del metamodelo MOMENT especificado.

   Después de haber definido una primitiva gráfica, se pueden consultar las propiedades ocultas que describen al nodo MOMENT, es decir, aquéllas que no están definidas gráficamente. La información de esas propiedades aparece en la lista inferior, y además se puede navegar recursivamente por las propiedades mediante el árbol situado en la parte izquierda inferior de la ventana.

Una vez se ha definido un metamodelo en la herramienta CASE, se puede definir un modelo de la forma convencional mediante la técnica *drag-and-drop* característica de este tipo de entornos de modelado, soltando primitivas gráficas de la vista gráfica sobre la hoja de dibujo, creando una nueva figura. El módulo *MOMENT Integrator* también enriquece esta funcionalidad definiendo los términos del esquema correspondiente del nivel M1 de forma automática y transparente al usuario. De manera que cuando un nodo del metamodelo tiene asignado un conjunto de

propiedades ocultas, éstas se instancian tomando como valor el valor por defecto especificado para la propiedad o uno asignado de forma automática por el módulo *MOMENT Integrator*. De manera que no sólo definimos un modelo gráfico, sino que además utilizamos la información semántica asociada a un metamodelo, tanto para los conceptos como para las propiedades.

Además, cuando disponemos de un metamodelo en MOMENT que ya ha sido asociado a una metáfora gráfica, se puede cargar un modelo de ese metamodelo, directamente del nivel M1 de MOMENT. Automáticamente se asocian las figuras, que son instancias de las primitivas gráficas de la vista asociada al metamodelo, a los términos concepto y propiedad del modelo formal, dando lugar a un diagrama que representa gráficamente el modelo cargado mediante la metáfora del metamodelo.

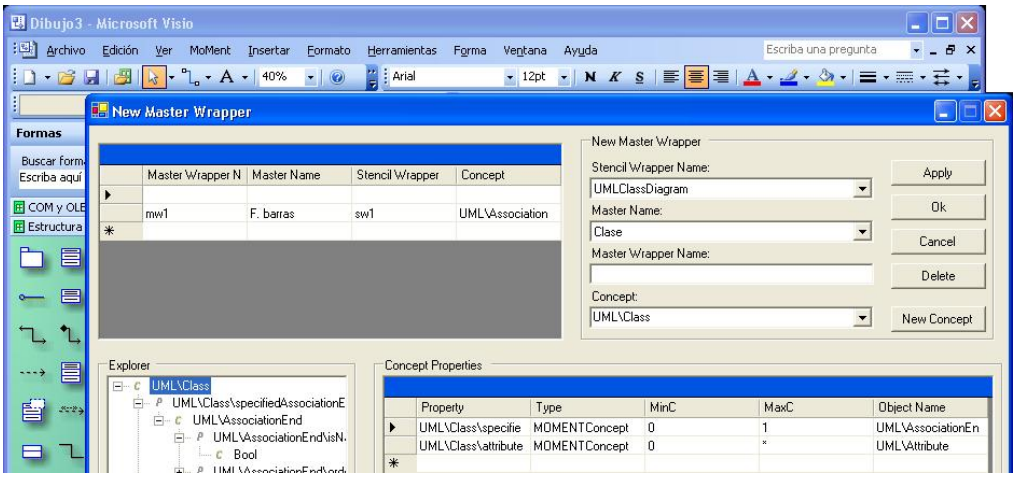

Fig.5. Creación de los conceptos y propiedades de un metamodelo

5. Trabajos relacionados

La definición de metáforas gráficas para técnicas de modelado no es una idea nueva en Ingeniería del Software como se indica en [8], pero pocas son las herramientas que permiten especificaciones formales, por no decir ninguna de las herramientas CASE más conocidas. En esta sección tratamos las herramientas CASE más conocidas en dos ámbitos de modelado: el independiente de dominio y el específico a dominio.

La técnica de modelado más representativa de las independientes de dominio es, sin duda alguna, UML [2]. UML aporta una notación gráfica para diseñar productos software siguiendo el paradigma orientado a objetos, pero no proporciona una especificación formal y sólida que garantice la producción de software de calidad y consistente.

La falta de una definición formal de los elementos del estándar UML provoca que el propio estándar vaya evolucionando sin garantía alguna sobre compatibilidad entre las diferentes versiones. De esta manera, una herramienta CASE que soporte UML, perfectamente puede quedar desfasada tras la publicación de una nueva versión del estándar. La existencia de una definición formal permitiría utilizar herramientas para

demostrar que la nueva versión es compatible con la anterior o incluso facilitar la migración automática hacia la nueva versión.

No obstante, hoy en día, existen herramientas CASE que permiten generar un producto software total o parcial a partir de un modelo UML, como Rational Rose [9], MS Visio [10], ArgoUML [11]. Estas herramientas únicamente suelen generar el esqueleto de la aplicación software final y las herramientas que proporcionan productos finales suelen ser difícilmente extensibles, lo que no permite seguir la contiua evolución de UML, y se suelen centrar en la generación de un tipo específico de aplicaciones. La funcionalidad de algunas de estas herramientas ha sido embebida en entornos de desarrollo (como Rational Rose XDE Developer [12] para Visual Studio .NET y EMF (Eclipse Modeling Framework) [13] para Eclipse) explotando los modelos UML para proporcionar facilidades durante la implementación del producto software. En [14] se realiza un amplio estudio de las propuestas de formalización de UML, pero ninguna de ellas ha llegado a sensibilizar a OMG para enriquecer el estándar UML.

Por otra parte, las técnicas de modelado específicas a dominio se están consolidando como una fuerte alternativa a las independientes de dominio. Esto es debido a que permiten mejorar el proceso de producción de software especializando a los usuarios en el espacio del problema (su área de trabajo) y no en el espacio de la solución (implementación tecnológica). La computación integrada en modelos (MIC – Model Integrated Computing) [15] es una aproximación de desarrollo de sistemas y de software que permite el uso de modelos específicos a dominio para representar aspectos relevantes de un sistema. El ciclo de desarrollo MIC consiste en identificar los conceptos de un dominio, sus atributos y relaciones obteniendo un metamodelo concreto. Este metamodelo se traduce a un entorno de diseño específico del dominio (DSDE – Domain Specific Design Environment) que permite definir modelos en un dominio. Además los DSDE suelen obtener código ejecutable, realizar análisis o animar los modelos. Algunas de las herramientas que destacan en este campo son GME [16], Atom3 [17] y MetaEdit+ [18]. Todas estas herramientas constituyen un marco de metamodelado basadas en un lenguaje abstracto para definir la sintaxis, semántica y visualización de lenguajes específicos a dominio.

En cambio, muchas de ellas utilizan lenguajes abstractos propietarios o UML (como GME) para definir metamodelos en lugar de utilizar un formalismo adecuado. De esta manera, los intérpretes de modelos obtenidos a partir de un metamodelo concreto suelen tener una complejidad considerable. Además, los transformadores de modelos se basan en las capacidades de estos lenguajes para realizar dichas traducciones de modelos a metamodelos diferentes. Por otra parte, ninguna de ellas consta de un mecanismo para definir metáforas gráficas como el que proporciona el entorno gráfico de MS Visio.

Mediante el *add-in* que hemos desarrollado para dicha herramienta CASE, no somos sólo capaces de definir metamodelos específicos a dominio de una forma sencilla y consiguiendo una buena representación gráfica de los conceptos relevantes de un dominio, sino que también ofrecemos soporte formal para notaciones gráficas utilizadas masivamente para el modelado independiente de dominio, como UML. Esta aportación se consigue mediante las especificaciones algebraicas que representan artefactos software en MOMENT.

6. Conclusiones y trabajo futuro

El modelado constituye un punto clave en el proceso de desarrollo del software, contando con diferentes aproximaciones. El modelado independiente de dominio que permite diseñar soluciones basándose en un determinado paradigma, como UML se centra en la orientación a objetos. Por otra parte, el modelado específico a dominio permite diseñar un producto software utilizando los conceptos del dominio del problema. Ambos tipos de técnicas de modelado suelen proporcionar una representación gráfica y una serie de facilidades para generar el producto software, total o parcialmente, y para transformar modelos que representan una determinada implementación tecnológica.

La teoría de grafos es el formalismo más utilizado para representar modelos y realizar transformaciones entre ellos pero su aceptación viene limitada por la complejidad de los algoritmos que proporcionan una solución a este tipo de problemas. En cambio, el enfoque algebraico de la plataforma de gestión de modelos MOMENT, permite abordar el problema de una forma genérica utilizando sistemas de reescritura de términos para realizar dichas transformaciones de una forma sencilla puesto que la implementación consiste únicamente en definir un conjunto de reglas de reescritura.

En este artículo presentamos un mecanismo de representación gráfica para los metamodelos especificados algebraicamente en MOMENT. De esta manera, permitimos definir metáforas gráficas asociándolas a metamodelos formales utilizando una herramienta CASE de gran difusión y fácilmente extensible. Este hecho permite acercar la gestión formal de modelos al usuario final a través de una interfaz gráfica fácilmente personalizable.

Para llegar a tal fin, hemos descrito la plataforma MOMENT y los principales elementos del entorno gráfico de una herramienta CASE. A continuación hemos descrito la arquitectura de un módulo que permite extender dichas herramientas, las asociaciones entre sus elementos gráficos y los elementos de la plataforma MOMENT, contextualizando la implementación en la herramienta Visio 2003. Finalmente, detallamos la interfaz que permite especificar dichas asociaciones para un metamodelo específico.

En los trabajos relacionados hemos citado herramientas que soportan el modelado independiente de dominio y específico a dominio, con mecanismos de manipulación de modelos mucho más complejos que los que ofrece MOMENT. Además con la aproximación seguida, MOMENT se beneficia del entorno gráfico de herramientas CASE, evitando el coste de desarrollo que sería necesario invertir para obtener un editor gráfico de las mismas características. Siguiendo este enfoque, se está estudiando la posibilidad de desarrollar un plug-in para la herramienta Eclipse que se beneficie de su módulo Graphical Editor Framework [20] y que utilice MAUDE [21] como sistema de reescritura de términos para manipular artefactos software en MOMENT.

7. Bibliografía

1. Boronat A., Carsí J.A., Ramos I., *Una plataforma semántica para la gestión de modelos*. Jornadas de Ingeniería del Software y Bases de Datos, JISBD 2003. Alicante, 2003.
2. Object Management Group, *Unified Modeling Language (UML) 1.4 draft*, February 2001. http://www.omg.org/cgi-bin/doc?ad/2001-02-11
3. Clavel M., Durán F., Eker S., Lincoln P., Martí-Oliet N., Meseguer J. y Quesada J., *Maude: Specification and programming in rewriting logic*. Theoretical Computer Science, 2001.
4. Wideman G., *Microsoft Visio 2003 Developer's Survival Pack*, 2004.
5. F# web site: http://research.microsoft.com/projects/ilx/fsharp.aspx
6. Diaconescu R., Futatsugi K., Ishisone M., Nakagawa A. T. y Sawada T. *An Overview of CafeObj*. In: C. Kirchner and H. Kirchner (eds), Proceedings of WRLA '98, Electronic Notes in Theoretical Computer Science, Vol. 13, 1998.
7. Object Management Group, *Meta-Object Facility (MOF) version 1.4*, April 2003. http://www.omg.org/technology/documents/formal/mof.htm
8. Bruno G., *Model Based Software Engineering*, Chapman & Hall, 1995.
9. Rational Rose Developer. Official web site: http://www-306.ibm.com/software/rational/
10. MS Office Visio 2003. Official web site: www.microsoft.com/office/visio/
11. ArgoUML. Official web site: http://argouml.tigris.org/
12. Rational Rose XDE Developer. Official web site: http://www-306.ibm.com/software/awdtools/developer/rosexde/
13. Eclipse Modeling Framework. Official web site: http://www.eclipse.org/emf/
14. Fernández J.L., *Una Propuesta de Formalización de la Arquitectura en Cuatro Capas de UML*. Tesis Doctoral, Departamento de Informática y Sistemas, Univ. de Murcia, 2001.
15. Sztipanovits J. y Karsai G., *Model-Integrated Computing*, Computer, Apr. 1997, pp. 90-92.
16. Ledeczi. A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason C., Nordstrom G., Sprinkle J. y P. Volgyesi. *The Generic Modeling Environment*. In Proc. Workshop on Intelligent Signal Processing. 2001.
17. Lara J., Vangheluwe H., Alfonseca M., *Using Meta-Modelling and Graph Grammars to create Modelling Environments*. Workshop on Graph Transformations and Visual Modellig Techniques GT-VMT at ICGT'2002. Barcelona, Octubre 2002. ENTCS Vol.72, (3).
18. Kelly S., Lyytinen K. y Rossi M., *Meta Edit+: A Fully configurable Multi-User and Multi-Tool CASE Environment.* In Proceedings of CAiSE'96, LNCS 1080, Springer-Verlag, Heraklion, Creta, Grecia, Mayo 1996, pp. 1-21.
19. Beckett D.. *The Design and Implementation of the Redland RDF Application Framework*. Tenth International World Wide Web Conference. Mayo 2-5, 2001, Hong Kong. http://www10.org/cdrom/papers/490/
20. Graphical Editing Framework. Official web site: http://www.eclipse.org/gef
21. MAUDE. Official web site: http://maude.cs.uiuc.edu/
22. Boronat A. Carsí J. A., Ramos I., An Algebraic Baseline for Automatic Transformations in MDA. Workshop Software Evolution Through Transformations: Model-based vs. Implementation-level Solutions (SETra'04), 2nd International Conference on Graph Transformation (ICGT2004), ENTCS, Roma (Italia). Octubre 2004.