XI Jornadas de Ingeniería del Software y Bases de Datos
JISBD 2006
José Riquelme - Pere Botella (Eds)

DEFINICIÓN DE OPERACIONES COMPLEJAS CON UN LENGUAJE ESPECÍFICO DE DOMINIO EN GESTIÓN DE MODELOS[1]

Abel Gómez, Artur Boronat, Luis Hoyos, José Á. Carsí e Isidro Ramos

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camí de Vera, s/n
46022 València
e-mail: {agomez,aboronat,lhoyos,pcarsi,iramos}@dsic.upv.es, web: http://issi.dsic.upv.es

Palabras clave: Ingeniería dirigida por modelos, Gestión de modelos, Lenguajes específicos de dominio, *Maude*.

Resumen. *La Ingeniería dirigida por Modelos permite incrementar la productividad en el proceso de desarrollo software, obteniendo herramientas más interoperables y sencillas de mantener mediante técnicas que elevan el nivel de abstracción. En esta dirección ha aparecido la disciplina «Gestión de Modelos», que proporciona un conjunto de operadores genéricos basados en teoría de conjuntos para tratar con modelos. Esta aproximación muestra su potencia en las capacidades de composicionalidad de los operadores que proporciona. Este artículo describe cómo proporciona soporte a la definición de operadores complejos una herramienta del marco de la Gestión de Modelos mediante un lenguaje específico de dominio.*

1. INTRODUCCIÓN.

En la iniciativa MDA, un artefacto software es considerado como un modelo. Un modelo en este contexto es la especificación de la funcionalidad, estructura y/o comportamiento de un sistema o aplicación [12], de forma que permite el desarrollo de éstas de forma automática mediante técnicas de programación generativas [7]. El proceso de desarrollo software en esta filosofía se basa en el refinamiento de los artefactos software desde el espacio del problema (donde se capturan los requisitos de la aplicación), al espacio de la solución (donde se especifica el diseño y desarrollo del producto software final). Durante este proceso de refinamiento se aplican diversas operaciones a los modelos, como por ejemplo, transformaciones o integraciones de distintos modelos. Tradicionalmente, este tipo de tareas

[1] Este artículo ha sido financiado por el Proyecto Nacional de Investigación, Desarrollo e Innovación DYNAMICA TIC 2003-07804-C05-01.

se han resuelto de manera *ad-hoc* para un contexto o metamodelo específico.
Para proporcionar un marco de trabajo genérico se propuso una nueva disciplina llamada Gestión de Modelos en [2]. Ésta considera los modelos y las correspondencias entre ellos como entidades de primer orden, proporcionando un conjunto de operadores independientes de metamodelo y basados en teoría de conjuntos para tratar con ellos (*Merge*, *Cross*, *Diff*, *ModelGen*, etc.). Estos operadores proporcionan una solución reutilizable y componible para las tareas descritas anteriormente.
Empleando nuestra experiencia en la aplicación de la lógica ecuacional de pertenencia para resolver problemas actuales de la ingeniería del software [5], se ha desarrollado una herramienta (llamada MOMENT) que da soporte algebraico a estos operadores mediante un eficiente sistema de reescritura de términos —*Maude* [6]— desde un entorno de modelado industrial.
El entorno elegido para integrar MOMENT ha sido *Eclipse Modeling Framework* (EMF) [15]. La integración de estas tecnologías permite aprovechar las capacidades de modelado de EMF, y la creación de interfaces de usuario amigables, obteniendo interfaces sencillas de emplear ocultando las peculiaridades de *Maude* al usuario.
Para ocultar estas peculiaridades en la declaración de operadores complejos en *Maude*, se ha diseñado en MOMENT un Lenguaje Específico de Dominio (*Domain Specific Language*). En la definición de este lenguaje se ha seguido la propia filosofía de desarrollo software dirigido por modelos, de forma que la declaración de un operador se representa mediante un modelo EMF. El modelo correspondiente a una operación compleja de Gestión de Modelos puede construirse mediante las interfaces gráficas que proporciona Eclipse o mediante una representación textual. La especificación final del operador en *Maude* se obtiene de forma automática mediante técnicas de generación automática de código.
Este artículo muestra cómo se ha resuelto en MOMENT la definición de operadores complejos mediante la aplicación a un caso de estudio no trivial. La estructura del artículo es la siguiente: el apartado 2 proporciona un ejemplo que ilustra la utilidad de la definición de operadores complejos; el apartado 3 describe brevemente MOMENT y algunos de los operadores simples disponibles que son empleados en la solución propuesta al ejemplo de motivación. El apartado 4 describe la arquitectura de la herramienta. El apartado 5 describe la solución propuesta para el problema presentado en el apartado 2, y finalmente, los apartados 6 y 7 describen algunos trabajos relacionados y las conclusiones respectivamente.

2. EJEMPLO DE MOTIVACIÓN: PROPAGACIÓN DE CAMBIOS.

Como ejemplo de motivación, utilizaremos un escenario de propagación de cambios semejante al introducido en [13], donde en primer lugar se define un diagrama de clases UML que modela un sistema de información para una aplicación.
Para construir la aplicación que almacene la información en una base de datos relacional, se utiliza la información descrita en el diagrama *Uml*. Aplicando un mecanismo de transformación (paso 1), obtenemos el nuevo esquema relacional *(Rdb)*. El mecanismo de transformación también genera un conjunto de enlaces entre el nuevo esquema relacional generado (*Rdb*) y el diagrama de clases origen (*Uml*) para proporcionar soporte a la

trazabilidad (*MapUml2Rdb*).

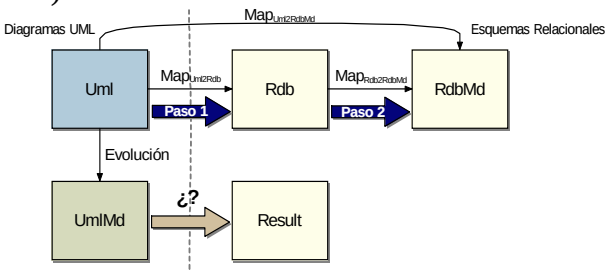

Figura 1. Ejemplo de propagación de cambios.

Tras obtener un esquema relacional a partir del diagrama UML original, se continúa con el desarrollo del nuevo sistema. Esto puede implicar cambios en la aplicación y en el esquema de la base de datos (paso 2), obteniendo el esquema relacional (*RdbMd*[2]). Estos cambios son trazados y almacenados por la herramienta que gestiona la manipulación del modelo, o introducidos directamente por el usuario (*MapRdb2RdbMd*).
Una vez se ha desarrollado el nuevo sistema, pueden producirse cambios en los requisitos del sistema, requiriendo modificaciones sobre el modelo original *Uml*. Es más sencillo modificar el diagrama *Uml* que modificar el esquema de la base de datos *RdbMd* de forma que se mantenga la consistencia entre el diseño de la aplicación y la capa de persistencia. En este punto, la aplicación de nuevo del mecanismo de transformación aplicado en el paso 1 descartaría los cambios aplicados de *Rdb* a *RdbMd*.
Una solución a este ejemplo de propagación de cambios puede proporcionarse mediante operadores de Gestión de Modelos.

3. MOMENT: UN FRAMEWORK PARA LA GESTIÓN DE MODELOS.

MOMENT es la herramienta de Gestión de Modelos que nos permitirá abordar la solución del problema del caso de estudio mediante la definición de una operación compleja de Gestión de Modelos. El siguiente apartado presenta brevemente los fundamentos en los que se apoya esta herramienta, así como los operadores simples sobre los que construiremos la solución en el apartado 5.

3.1. Espacios tecnológicos en MOMENT: EMF y *Maude*.

En MOMENT se emplean los espacios tecnológicos [11] *Maude* y EMF. El álgebra de operadores propuesta por Bernstein para tratar con modelos se ha especificado directamente como un álgebra genérica usando el formalismo de especificaciones algebraicas *Maude*. Mediante un plug-in que integra *Maude* dentro del entorno de desarrollo Eclipse podemos usar este sistema para nuestros propósitos.

[2] Los sufijos «–Md» y «–Unmd» son contracción de «Modificado» y «No modificado» respectivamente. Se ha adoptado esta convención en todo el texto para evitar el uso de nombres de variables excesivamente largos en el ejemplo de aplicación al caso de estudio.

3.2. Operadores en MOMENT.

En MOMENT cada operador se encuentra definido de forma genérica en un módulo paramétrico. Para aplicar estos operadores sobre unos modelos concretos, estos módulos deben ser instanciados con los metamodelos implicados en la operación. Toda esta tarea la realiza automáticamente los mecanismos de control de ejecución de MOMENT.

3.2.1. Operadores comunes.

Los operadores comunes son aquellos aplicables a cualquier tipo de modelo y se basan en teoría de conjuntos[3]. Los operadores de interés para nuestro caso de estudio son los siguientes:

1. *Cross* y *Merge*. Estos operadores corresponden a operaciones de conjuntos bien definidas: intersección y unión disjunta respectivamente. Ambos operadores reciben dos modelos (A y B) como entradas, y producen un tercer modelo (C). También devuelven sendos modelos de enlaces (*mapAC* y *mapBC*) que relacionan los elementos de cada modelo de entrada con los elementos del modelo resultante.
2. *Diff*. Este operador realiza la diferencia entre dos modelos de entrada (A y B) devolviendo un modelo resultado (C) y un único modelo de enlaces (*mapAC*), —un modelo *mapBC* no es interesante porque en C no habrá elementos de B—.
3. *ModelGen*. *ModelGen* realiza la traducción de un modelo (A) (o un conjunto de modelos), definidos mediante sus respectivos metamodelos[4], a un metamodelo MMB destino, obteniendo el modelo B, según una transformación determinada especificada en el lenguaje de transformaciones QVT [14]. Este operador produce a su vez un modelo de correspondencias relacionando los elementos de cada uno de los modelos de entrada con los elementos del modelo generado.

3.2.2. Operadores de soporte para la navegación:

Son operadores más específicos que los anteriores, diseñados para tratar con modelos de trazabilidad (modelos de enlaces que permiten relacionar elementos entre dos modelos distintos). En las siguientes definiciones encontramos estos elementos: dos modelos, uno denominado *dominio*¸ y otro modelo *rango* (A y B); un modelo de trazabilidad (mapAB) que relaciona los elementos de dos modelos de entrada; un modelo (A') que es un submodelo de A; y un modelo (B') que es un submodelo de B. Los operadores de trazabilidad considerados son:

1. *Range*. Dados un modelo de trazabilidad (mapAB), un modelo dominio (A'), y un modelo rango (B), obtiene un submodelo del modelo rango B (B') cuyos elementos se relacionan con los de A'.
2. *RestrictDomain*. Dado un modelo dominio (A') y un modelo de trazabilidad (mapAB), devuelve un modelo con los enlaces de trazabilidad de «mapAB» que tienen elementos

[3] La semántica de los operadores se asemeja a los operadores de teoría de conjuntos. No obstante, como se muestra en [4], esta semántica puede refinarse en función del metamodelo sobre el que se aplican.

[4] O como se expresaría según la terminología inglesa, «un modelo conforma a su respectivo metamodelo».

del modelo A' como elementos dominio.

3. *Compose*. Dados tres modelos A, B y C, y los modelos de trazabilidad «mapAB» y «mapBC» que relacionan respectivamente A con B y B con C; el operador *Compose* obtiene el modelo «mapAC» haciendo explícita la relación entre los elementos de A y C por la propiedad transitiva.

4. EJECUCIÓN DE OPERADORES COMPLEJOS EN MOMENT.

El desarrollo de MOMENT ha sido realizado siguiendo la propia filosofía de desarrollo de software dirigido por modelos. Se ha desarrollado un modelo de la aplicación empleando EMF que describe todos los elementos involucrados en la ejecución de operadores. Posteriormente, mediante las capacidades de generación de código que proporciona EMF se ha obtenido el código Java de la aplicación.

4.1. Soporte para la definición de operadores complejos.

Esta filosofía, además de incrementar la mantenibilidad del código, permite manipular todos los elementos representados en el modelo de la aplicación a un mayor nivel de abstracción. Además proporciona un mecanismo automático de persistencia y recuperación de datos (por defecto el formato de persistencia es XMI, lo que también proporciona interoperabilidad con otras aplicaciones).

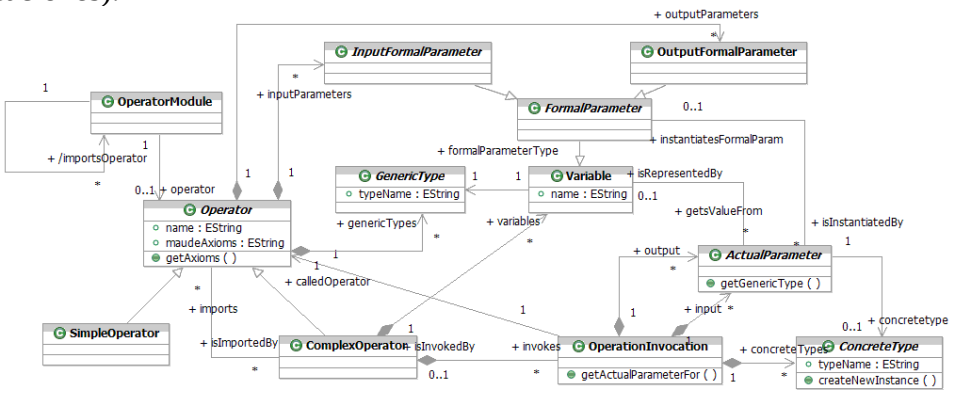

Figura 2. Modelo simplificado para la especificación de operadores complejos en MOMENT.

El diagrama de clases de la Figura 2 muestra la parte del modelo de la herramienta que permite la declaración y ejecución de operadores de gestión de modelos. La clase *Operator* captura la información de la declaración de un operador. Esta clase se especializa en operadores simples *(SimpleOperator)* y complejos *(ComplexOperator)*. Los parámetros formales tanto de entrada *(InputFormalParameter)* como de salida *(OutputFormalParameter)* son especializaciones de la clase *Variable*; disponiendo todos ellos tanto de un nombre como de un tipo *GenericType*. Los tipos en la declaración de un operador se denominan genéricos puesto que, como se comentó en el apartado 3.2, la declaración de un operador se realiza de forma genérica independiente del metamodelo concreto.

La clase *OperationInvocation* captura la información sobre la ejecución de un determinado

operador (referenciado mediante el rol *calledOperator*) con unos datos concretos, representados en la figura mediante la clase *ActualParameter* (parámetro actual). Cada parámetro actual es el dato concreto con el que se invoca un operador e instancia un parámetro formal de la declaración de un operador (rol *instantiatesFormalParameter*). De igual forma que un parámetro actual, también dispone de un tipo *(ConcreteType)*. Por ejemplo, un parámetro actual podría ser un diagrama de clases UML concreto (por ejemplo el modelo *Uml* del caso de estudio), y su tipo concreto sería el metamodelo UML.
Esta representación de los operadores complejos como instancias del modelo de la aplicación permite tratar con ellos a un mayor nivel de abstracción, ya que las manipulaciones y consultas sobre un operador se realizan sobre los conceptos modelados en la figura, y no sobre un árbol de sintaxis abstracta construido en tiempo de compilación a partir de una gramática. Finalmente, dada esta representación del operador y mediante técnicas generativas, se obtiene el código *Maude* en la forma de un módulo paramétrico independiente de metamodelo.

4.2. Arquitectura de la herramienta.

MOMENT emplea un proceso *Maude* desde un programa Java. Para ello se ha hecho uso de *Maude Development Tools* [8], un conjunto de herramientas que extienden Eclipse y proporcionan una API para el uso de *Maude*. El diagrama de componentes UML de la Figura 3 muestra los elementos de MOMENT relacionados con la ejecución de operaciones de Gestión de Modelos. Los principales módulos son los denominados «Lanzador de operaciones» *(Operator Launcher)* y «Cargador de módulos» *(Module Loader).*

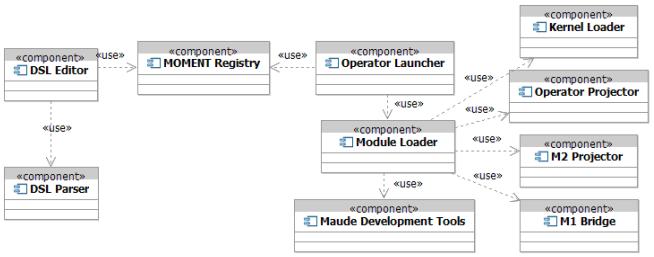

Figura 3. Componentes relacionados con la ejecución de operaciones de gestión de modelos.

El primero de ellos es el que proporciona la interfaz de MOMENT al usuario. Permite, dada la declaración de un operador (sea simple o complejo), especificar sus parámetros actuales así como dónde se guardarán los resultados devueltos. El segundo componente, el cargador de módulos, se encarga de controlar de forma transparente al usuario el proceso *Maude* sobre el que se ejecutarán las operaciones.
MOMENT se sustenta sobre un conjunto básico de módulos *Maude* (denominado *kernel*) que proporcionan la funcionalidad básica del *framework*. Entre estos módulos encontramos aquellos que implementan los operadores simples comentados en el apartado 3.2. El *Module Loader*, se ocupa de la preparación del contexto de ejecución (carga de los módulos del kernel por parte del *Kernel Loader*) y de la proyección a código *Maude* de todos los elementos que intervendrán en la ejecución. Esto incluye la especificación algebraica de los metamodelos

implicados *(M2 Projector)*, el código del operador a ejecutar *(Operator Projector)*, y los términos correspondientes a los parámetros de entrada en el momento de la invocación de la operación *(M1 Bridge)*. Finalmente, el puente a nivel M1 *(M1 Bridge)* es también el componente encargado de procesar los términos resultantes de una ejecución y recuperarlos en el espacio tecnológico de EMF. Es de destacar en este punto la expresividad de *Maude* puesto que a diferencia de otros lenguajes (como Java), la composicionalidad funcional de operaciones es inherente al formalismo de la lógica ecuacional de pertenencia subyacente.

4.3. Definición textual de operadores complejos.

La representación de los operadores complejos como instancias de un modelo aporta numerosas ventajas. No obstante, las interfaces gráficas proporcionadas por defecto por Eclipse resultan poco intuitivas para la definición de nuevos operadores. En este caso, la opción más natural para el usuario es ofrecer una representación textual. Por ello se ha definido una gramática [8] para soportar la definición de operadores complejos de esta forma.
Tal como se observa en la Figura 3, existen en MOMENT dos componentes adicionales: *DSL Editor* y *DSL Parser*. El primero corresponde a un editor de texto con coloreado de sintaxis que permite la declaración textual de operadores complejos. Mediante el *DSL Parser*, obtiene automáticamente la representación EMF equivalente a la declaración textual de un operador, que finalmente, se incluye en el repositorio (*MOMENT Registry*). Es el *Operator Launcher* quien recupera los operadores de este repositorio para ejecutarlos sobre *Maude*.

5. APLICACIÓN DE MOMENT AL CASO DE ESTUDIO.

Dado el problema del caso de estudio (apartado 2), el modelo *MapUml2RdbMd* puede ser fácilmente obtenido de los modelos *MapUml2Rdb* y *MapRdb2RdbMd* mediante el operador *Compose*. Por tanto, el problema puede enunciarse de la siguiente manera:
«Dados los siguiente modelos: un diagrama de clases UML original (Uml); un diagrama de clases UML (UmlMd) evolucionado de Uml; un esquema de base de datos relacional RdbMd, generado a partir del diagrama de clases UML y modificado posteriormente; y un modelo de trazabilidad entre UML y RDB' (MapUml2RdbMd); deberemos obtener el esquema relacional del diagrama de clases UmlMd que conserve los cambios realizados en RdbMd.»
Este problema puede ser resuelto por el siguiente operador complejo:

```
operator PropagateChanges(MM1 Uml, MM1 UmlMd, MM2 RdbMd,
                TraceabilityMetamodel MapUmlIni2RdbMd, Transformation Uml2Rdbms)
        : <MM2, TraceabilityMetamodel>
{   // Obtención del modelo resultado
    <UmlUnmd, MapUml2UmlUnmd, MapUmlMd2UmlUnmd> = Cross(Uml, UmlMd);                (1)
    <RdbUnmd> = Range(mapUml2RdbMd , UmlUnmd, RdbMd);                               (2)
    <UmlNew, MapUmlMd2UmlNew, MapUmlUnmd2UmlNew> = Diff(UmlMd, UmlUnmd);            (3)
    <RdbNew, MapUmlNew2RdbNew> = ModelGen(Uml2Rdbms, UmlNew);                       (4)
    <Result, MapRdbUnmd2Result, MapRdbNew2Result> = Merge(RdbUnmd, RdbNew);         (5)
    // Generación del modelo de trazabilidad
    <MapUmlUnmd2RdbUnmd> = RestrictDomain(UmlUnmd , MapUml2RdbMd);                  (6)
    <MapUmlUnmd2Result> = Compose(MapUmlUnmd2RdbUnmd, MapRdbUnmd2Result);           (7)
    <MapUmlNew2Result> = Compose(MapUmlNew2RdbNew, MapRdbNew2Result);               (8)
    <MapUmlMd2Result, Map1, Map2> = Merge(MapUmlUnmd2Result, MapUmlNew2Result);     (9)
    return <Result, MapUmlMd2Result>;
}
```

Este operador está construido a partir de operadores simples del álgebra de MOMENT y realiza las siguientes operaciones (ver Figura 4): (1) «UmlUnmd» es la parte del modelo UML que permanece sin modificar en el modelo *UmlMd*. (2) «RdbUnmd» es el submodelo de *RdbMd* que corresponde a la parte no modificada de *UmlMd*. (3) «UmlNew» es la parte de *UmlMd* que ha sido añadida al modelo *Uml*. (4) «RdbNew» es el esquema relacional obtenido de la traducción de *UmlNew* al metamodelo relacional. (5) «Result» es el modelo final obtenido de la integración de las bases de datos obtenidas en los pasos 2 y 4.

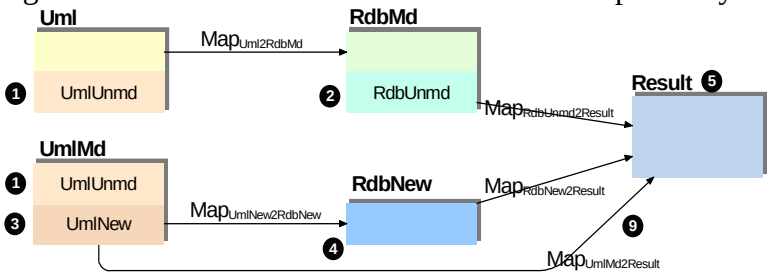

Figura 4: Solución al problema del caso de estudio.

«MapUmlUnmd2RdbUnmd» (6) es el modelo de trazabilidad que enlaza los elementos de *Uml* con elementos de *RdbMd* que han permanecido sin cambios. (7) *MapUmlUnmd2Result* es el modelo de trazabilidad que relaciona la parte no modificada de *Uml (UmlUnmd)* y *Result* combinando *MapUmlUnmd2RdbUnmd* y *MapRdbUnmd2Result*. (8) Obtiene el modelo de trazabilidad entre la parte nueva de *UmlMd* y *Result*. Finalmente, (9) une los modelos obtenidos en 7 y 8 mediante el operador *Merge*, obteniendo *MapUmlMd2Result*.

Este operador resuelve el problema de la propagación de cambios de forma independiente de los metamodelos implicados, por lo que puede ser aplicado a cualquier combinación de metamodelos (en lugar de UML y relacional). Tras obtenerse su representación como modelo EMF mediante el *parser* implementado, se obtiene el código *Maude* que encontramos en [8].

6. TRABAJOS RELACIONADOS.

En el campo de la ingeniería dirigida por modelos pocas son las aproximaciones que proporcionan composicionalidad en sus operaciones, especialmente en la disciplina de Gestión de Modelos, dado el reducido número de herramientas que proporcionan un conjunto de operadores tal y como la Gestión de Modelos los define.

RONDO [13] es la plataforma de Gestión de Modelos basada en los trabajos de P. Bernstein [13]. Ésta proporciona un lenguaje de *script* similar al que proporciona MOMENT mediante el cual se pueden definir operadores complejos. No obstante, en esta herramienta, el operador *Merge* por ejemplo, recibe como entradas dos modelos (A y B) y un modelo de *mappings* entre ellos (*mapAB*); produciendo el modelo combinado C y dos nuevos modelos de correspondencias (*mapAC* y *mapBC*): *<C, mapAC, mapBC> = Merge(A, B, mapAB).*

En MOMENT, sin embargo, las relaciones de equivalencia entre los elementos de dos modelos (necesarias para aplicar un operador), se definen entre los elementos de sus metamodelos de forma axiomática. Esta colección de relaciones de equivalencia entre dos

metamodelos constituye un morfismo que puede ser reusado por todos los operadores del álgebra de MOMENT permitiendo una especificación más clara de los operadores complejos.
Existen también diversas plataformas para la manipulación de modelos que se han implementado sobre Eclipse y su *framework* de modelado *(EMF)* como por ejemplo *ATL* [3] (*Atlas Transformation Languaje*). Esta plataforma proporciona un lenguaje que permite especificar transformaciones de modelos de forma declarativa o imperativa, de forma dependiente del metamodelo. Este lenguaje carece de la potencia para la composición de operadores de MOMENT. En este sentido ATL únicamente proporciona la capacidad para definir *librerías* de funciones auxiliares que no son ejecutables de forma independiente [1]. En MOMENT, al contrario, un operador siempre es una unidad ejecutable de forma independiente y susceptible de ser incluida en la definición de un operador complejo mayor.
En el marco de EMF también se encuentre *Epsilon Object Language* [10]. Esta aproximación proporciona un lenguaje de bajo nivel para la manipulación de modelos, con el objetivo de construir lenguajes de fines más específicos sobre éste. EOL está inspirado en los mecanismos de navegación de OCL [17]. Esta forma de navegación, basada en la estructura de los modelos, impide la definición de operaciones genéricas fácilmente componibles.

7. CONCLUSIONES.

La Gestión de Modelos es un campo de investigación emergente en la ingeniería dirigida por modelos que destaca por su potencia en la composición de operaciones de una forma natural, intuitiva, genérica y reutilizable.
En este artículo hemos presentado la herramienta MOMENT desde el prisma de la definición de operadores complejos.
Maude proporciona una implementación de la lógica ecuacional de pertenencia, que ha sido utilizada para definir las operaciones de gestión de modelos mediante módulos funcionales. En estos módulos funcionales, las operaciones son descritas como funciones y dependiendo de las propiedades algebraicas que son añadidas a cada operación (asociatividad, conmutatividad, etc) se pueden componer fácilmente. Estas facilidades de composición de funciones han sido reflejadas en el lenguaje de definición de operadores complejos.
Por otra parte, el uso de una herramienta como Eclipse y EMF proporciona a esta aproximación una gran interoperabilidad y extensibilidad, a la vez que permite aplicar la propia filosofía de ingeniería dirigida por modelos al diseño del lenguaje de definición de operadores complejos. La representación de un operador a un mayor nivel de abstracción permite el aprovechamiento de técnicas de programación generativas para la obtención del código ejecutable final y facilita el diseño de diversas interfaces de usuario para la definición y modificación de operadores complejos con independencia del lenguaje empleado por el usuario. Ejemplo de esta independencia es que un operador complejo puede ser definido mediante un editor gráfico proporcionado por defecto por Eclipse, o mediante el editor textual implementado en MOMENT que proporciona una sintaxis más intuitiva.
Pero también cabe destacar que esta representación como instancia de un modelo, permite aprovechar el esfuerzo de terceros en el campo de la ingeniería dirigida por modelos en, por ejemplo, la creación semi-automática de lenguajes visuales específicos de dominio.

De esta manera, como trabajo futuro encontramos el desarrollo de un editor visual para la definición de operadores complejos aprovechando los esfuerzos del proyecto *Graphical Modeling Framework* (GMF) [16], que permite la generación de editores gráficos para la edición de modelos EMF mediante una metáfora gráfica específica de dominio.

REFERENCIAS.

[1] ATLAS group, LINA & INRIA, Nantes. *ATL User Manual. Version 0.7*. http://www.eclipse.org/gmt/atl/doc/ATL_User_Manual[v0.7].pdf

[2] Bernstein, P.A., Levy, A.Y., Pottinger, R.A.: *A Vision for Management of Complex Models*. Microsoft Research Technical Report MSR-TR-2000-53, June 2000, (short version in SIGMOD Record 29, 4 (Dec. '00)).

[3] Bézivin, J., Valduriez. P., Jouault, F. *The ATL home page*. http://www.sciences.univ-nantes.fr/lina/atl/

[4] Boronat, A., Carsí, J.Á., Ramos, I., Letelier, P. *Formal Model Merging Applied to Class Diagram Integration*. Int. ERCIM Workshop on Soft. Evolution. 2006. Lille.

[5] Boronat, A., Pérez, J., Carsí, J. Á., Ramos, I.: *Two experiencies in software dynamics*. Journal of Universal Science Computer. Special issue on Breakthroughs and Challenges in Software Engineering. Vol. 10 (issue 4). April 2004.

[6] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: *Maude: specification and programming in rewriting logic*. Theoretical Computer Science, 285(2):187-243, 2002.

[7] Czarnecki, K., Eisenecker, U.: *Generative Programming: Methods, Tools, and Applications*. Addisson-Wesley (2000). ISBN 0-201-30977-7, pag. 267-304.

[8] Gómez, A., Boronat, A., Carsí, J.Á., Ramos, I. *Integración de un sistema de reescritura de términos en una herramienta de desarrollo software industrial*. IV Jornadas de trabajo DYNAMICA. Archena (Murcia), 2005.

[9] Gramática para la definición de operadores complejos y código Maude proyectado para el caso de estudio. http://moment.dsic.upv.es/Default.aspx?tabid=86

[10] Kolovos D. *Website de EOL*. http://www-users.cs.york.ac.uk/~dkolovos/epsilon/

[11] Kurtev, I., Bézivin, J., Aksit, M. *Technological Spaces: An Initial Appraisal*. Int. Federated Conf. (DOA, ODBASE, CoopIS). Irvine, 2002.

[12] *MDA Guide Version 1.0.1*. http://www.omg.org/docs/omg/03-06-01.pdf

[13] Melnik, S., Rahm E., Bernstein P. A., *Rondo: A Programming Platform for Generic Model Management*. SIGMOD 2003. Extended ver. in Web Semantics, vol. 1, Num. 1.

[14] OMG: MOF 2.0 QVT final adopted specification (ptc/05-11-01). (2005).

[15] Sitio web de EMF: http://www.eclipse.org/emf/

[16] Sitio web de GMF: http://www.eclipse.org/gmf/

[17] Warmer, J., Kleppe, A.: *The Object Constraint Language, Second Edition, Getting Your Models Ready for MDA*. Addison-Wesley (2004).