UN MOTOR DE TRANSFORMACIÓN DE MODELOS CON SOPORTE PARA EL LENGUAJE QVT RELATIONS

Pascual Queralt, Luis Hoyos, Artur Boronat, José Á. Carsí e Isidro Ramos

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

Palabras clave: QVT Relations, Transformaciones de modelos, Ingeniería de Modelos, Maude.

Resumen. *En la actualidad están apareciendo un gran número de herramientas que proporcionan soporte para transformar modelos. No obstante, hasta el momento no existe ninguna que haya tenido en consideración el lenguaje estándar QVT Relations. En este artículo se presenta la arquitectura de un motor de transformación de modelos que proporciona soporte para este lenguaje estándar dentro del framework de gestión de modelos MOMENT. Este motor de transformaciones ha sido definido mediante la utilización de métodos formales como plugin dentro de Eclipse, haciendo uso de Eclipse Modeling Framework para la manipulación de modelos.*

1. INTRODUCCIÓN

El paradigma de desarrollo dirigido por modelos es una disciplina de la Ingeniería del Software donde los modelos son considerados como artefactos de primera clase, y donde la transformación genérica de modelos realiza un papel fundamental. Han sido numerosos los lenguajes de transformación surgidos con el propósito de especificar transformaciones genéricas entre modelos, pero la dificultad de aprendizaje y la escasa productividad de las herramientas que soportan estos lenguajes ha limitado su repercusión dentro de la comunidad del software. En un intento de aunar los esfuerzos que se están llevando a cabo en este campo, la OMG ha propuesto un lenguaje estándar para la especificación de transformaciones llamada *Query View Transformations* (QVT) [1], que depende a su vez de otros dos estándares: MOF 2.0 y OCL 2.0. De esta manera, se pretende promover el uso estándar de las transformaciones entre modelos, lo que llevará asociado notables mejoras en

la productividad, la interoperabilidad y la calidad, permitiendo que el desarrollo de software se lleve a cabo en un nivel de abstracción más elevado y utilizando conceptos más cercanos al dominio del problema.

Para que la visión del desarrollo dirigido por modelos sea una realidad, las herramientas que lo soportan deben ser capaces de proporcionar la automatización de transformaciones de modelos. Éstas, no sólo deben ofrecer transformaciones de modelos de forma *ad-hoc*, sino también proporcionar un lenguaje que permita a los usuarios definir sus propias transformaciones y ejecutarlas posteriormente, con lo que se podrán automatizar tareas como la aplicación de patrones, el refinamiento o la refactorización.

En este trabajo presentamos un prototipo que ofrece soporte para el lenguaje QVT Relations. El motor de transformaciones está integrado en la herramienta de gestión de modelos MOMENT (*MOdel manageMENT*) basada en una aproximación híbrida entre una herramienta formal y un entrono industrial de modelado, Maude y *Eclipse Modelling Framework* (EMF) [2]. Por un lado, Maude es un sistema de reescritura de términos que proporciona soporte para lógica ecuacional de pertenencia y lógica de reescritura. Maude se ha utilizado como entorno de prototipado rápido para desarrollar la herramienta haciendo uso de algunas de sus propiedades: mecanismo de *pattern matching*, parametrización y reflexión entre otros. Por otro lado, EMF es un estándar industrial que incluye un metamodelo (*ecore*[1]) para describir y soportar la ejecución de modelos, incluyendo notificación de cambios, soporte de persistencia mediante serialización XMI y una API reflexiva muy eficiente para la manipulación de modelos EMF. Con todo esto, MOMENT reaprovecha los esfuerzos realizados tanto en el campo teórico como en el aplicado: en Maude sobre aspectos teóricos y en EMF sobre su aplicación a la Ingeniería del Software.

La estructura del documento es la siguiente: en el apartado 2 se presenta la especificación QVT y se describen brevemente las principales características de MOMENT-QVT, seguidamente en el apartado 3 se muestran los distintos componentes funcionales de MOMENT a través de la secuencia del proceso en la ejecución de transformaciones; en el apartado 4 se ven algunos de los trabajos relacionados en el área y finalmente en el apartado 5 se detallan las conclusiones.

2. QVT Y MOMENT

La especificación QVT se define a través de dos dimensiones ortogonales: la dimensión del lenguaje y la dimensión de la interoperabilidad, donde cada una de las cuales tiene una serie de niveles. La dimensión del lenguaje define los diferentes lenguajes de transformación presentes en la especificación QVT. Concretamente son tres: Relations, Core y Operational, y la principal diferencia entre ellos es su naturaleza declarativa o imperativa. En la dimensión de la interoperabilidad se encuentran aquellas características que permiten a una herramienta que cumple el estándar QVT interoperar con otras herramientas. La intersección de niveles de las dos dimensiones especifica un punto de cumplimiento QVT (*QVT-compliance*).

Para integrar QVT en MOMENT se ha elegido el lenguaje Relations para aprovechar su

[1] Ecore es un lenguaje común basado en EMOF, que es parte de la especificación MOF 2.0 usado para la definición de metamodelos

naturaleza declarativa, su especificación de trazabilidad transparente al usuario, así como la sintaxis bastante sencilla, en comparación con otros lenguajes declarativos, que ofrece para definir transformaciones y relaciones de equivalencia entre modelos. En el lenguaje Relations las transformaciones quedan descritas a través de la enumeración de los metamodelos participantes, un conjunto de reglas que especifican la relación existente entre términos de los metamodelos, un conjunto de dominios por regla que se ajusta al conjunto de términos para los que se expresan relaciones y un conjunto de patrones que cumplen con la estructura de los términos y a los que se aplican operaciones OCL. El que una regla sea de transformación o no viene determinado por el dominio destino, que estará marcado como *checkonly* o *enforce*. La marca *checkonly* comprueba si existe una correspondencia válida entre los modelos, mientras que la marca *enforce* fuerza a que los términos de los respectivos modelos cumplan la relación descrita en la regla.

Respecto a la interoperabilidad en MOMENT, se permite la ejecución de modelos que conforman al metamodelo QVT Relations, la ejecución de cualquier especificación textual de una transformación expresada con el lenguaje Relations, así como la importación y exportación de modelos serializados con formato XMI (véase Tabla 1).

En MOMENT, a través de QVT Relations, es posible la especificación de transformaciones endógenas y exógenas. Las transformaciones endógenas son aquellas transformaciones que comparten el mismo metamodelo, mientras que las transformaciones exógenas se definen entre modelos que conforman a metamodelos diferentes[2] [3].

		Interoperabilidad			
Lenguaje		Sintaxis Ejecutable	XMI Ejecutable	Sintaxis Exportable	XMI Exportable
	Core				
	Relations	**X**	**X**		**X**
	Operational				

Tabla 1. Cumplimiento de QVT.

Por un lado, el hecho de que MOMENT se base en estándares como MOF, QVT, OCL y XMI, conlleva implícitamente un alto nivel de interoperabilidad con herramientas similares. Por otro, la utilización y aplicación de Maude como método formal subyacente permitirá el estudio formal de ciertas propiedades de las transformaciones de modelos: como la terminación o el indeterminismo.

En cuanto a la cardinalidad de modelos participantes en una transformación, el número máximo de modelos origen no está limitado. Por otra parte, el número de modelos destino queda restringido a uno tal y como se especifica en el estándar QVT, aunque es posible

[2] Si en una transformación se utilizan varios modelos origen y/o varios modelos destino, entonces pueden estar involucrados varios metamodelos diferentes.

alcanzar cardinalidad múltiple en el resultado aplicando composición de transformaciones que tengan diferentes modelos resultantes y que pasen a formar parte del resultado final. La forma con la que actualmente se lleva a cabo la composición de transformaciones es manual y mediante interacción del usuario, pero en un futuro próximo se tiene previsto que este proceso se lleve a cabo de manera automática. Dado que en MOMETN-QVT las transformaciones se proyectan como funciones en Maude, la composición de transformaciones se basa en la composición algebraica de funciones. En definitiva, son posibles las transformaciones 1-1, n-1, 1-n y n-n, bien aplicando una sola transformación o mediante composición de transformaciones.

3. EJECUCIÓN QVT RELATIONS EN MOMENT

El proceso de transformación que se lleva a cabo en MOMENT-QVT consta de tres fases: análisis, proyección a Maude y ejecución y proyección a EMF (Figura 1). Todas ellas quedan ocultas al usuario, siendo únicamente necesaria la especificación de la transformación por parte del mismo.

Para llevar a cabo estas tres fases, son necesarios un conjunto de componentes funcionales que forman la arquitectura de MOMENT-QVT.

- QVT Parser: encargado de analizar un programa QVT Relations de entrada y generar su modelo correspondiente.
- MOMENT Registry: repositorio de artefactos generados y/o utilizados por MOMENT.
- MOMENT Relations: encargado de generar y proyectar código Maude a partir de la especificación de una transformación.
- OCL Parser: encargado de analizar expresiones OCL y generar su código Maude correspondiente.

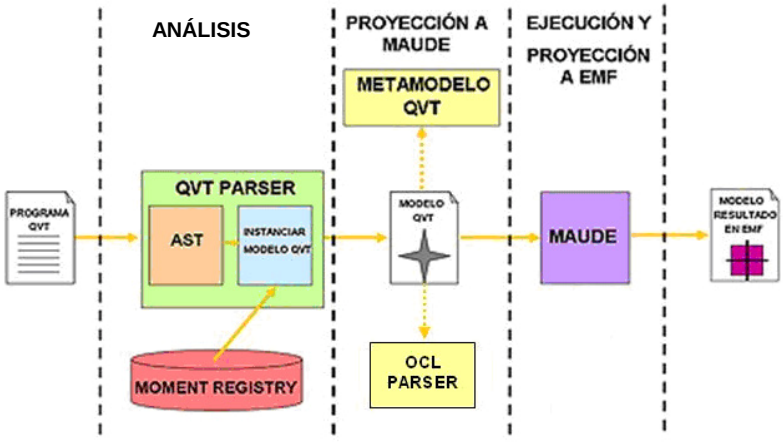

Figura 1. Proceso de ejecución de un programa QVT Relations en MOMENT

3.1. Fase de análisis

Esta etapa comienza con un análisis léxico y sintáctico del programa QVT Relations.

Recorriendo el árbol de sintaxis abstracta (AST) generado por esta etapa, se crea un modelo instancia del metamodelo de QVT Relations que se encuentra integrado en la plataforma MOMENT. Durante este recorrido es necesario consultar los metamodelos respectivos a los modelos participantes en la transformación y obtener de este modo los tipos y las propiedades de sus elementos. Por consiguiente, es prerrequisito imprescindible que estos metamodelos estén registrados en el MOMENT *Registry*.
Para el ejemplo, nos hemos basado en la transformación que aparece en el apéndice de la especificación del estándar MOF-QVT, que especifica una transformación de un diagrama de clases UML a su correspondiente modelo relacional. En la parte izquierda de la Figura 2, se muestra el segmento de código de la regla de transformación "ClassToTable". Esta regla establece la correspondencia entre una clase perteneciente al metamodelo origen (UML) y una tabla perteneciente al metamodelo destino (relacional), de forma que a partir del término clase perteneciente a uno de los dominios de la regla se obtienen los términos tabla, columna y clave primaria, que pertenecen al dominio destino de la regla tal como indica la palabra reservada *enforce*. En la parte derecha de la Figura 2 y tras llevar a cabo la fase de análisis, se muestra la parte del modelo QVT Relations correspondiente a la regla "ClassToTable".

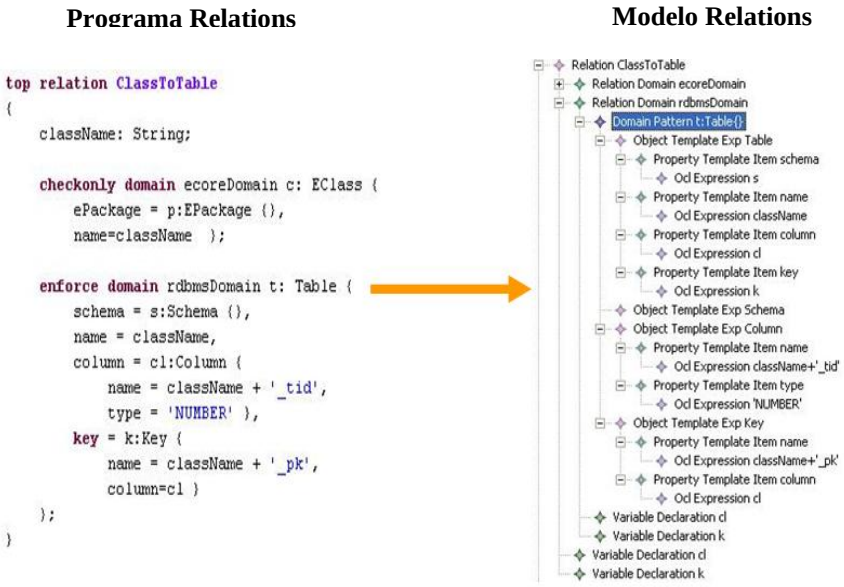

Figura 2. Vistas de programa Relations y modelo asociado.

3.2. Fase de proyección a Maude

Los metamodelos origen y destino son compilados en teorías basadas en lógica ecuacional [4]. Los conceptos del metamodelo se proyectan como *sorts*[3] y las propiedades como

[3] Un sort es el nombre que recibe un tipo den Maude.

constructores[4] y operadores de consulta. Así, se crea una especificación algebraica en el espacio tecnológico de Maude para cada metamodelo participante en la transformación. Además, se toman todos los modelos origen (en la capa M1 de MOF) y se obtiene para cada uno de ellos el término algebraico que conforma la especificación algebraica definida para su respectivo metamodelo [5].

Seguidamente, el modelo QVT Relations resultado de la etapa de análisis se proyecta en otra especificación algebraica. Una parte muy importante de esta proyección es la traducción de las expresiones OCL utilizadas en la especificación de la transformación, y que se realiza a través del componente OCL Parser [6].

La nueva especificación algebraica obtenida extiende a las especificaciones ya existentes de los metamodelos participantes en la transformación, mediante un conjunto de reglas de reescritura que describen las guías para realizar la transformación, constituyendo el contexto donde se definen las relaciones semánticas entre los metamodelos origen y destino.

La Figura 3 muestra el resultado de proyectar a Maude la regla de transformación "ClassToTable" obtenida en la fase anterior. En ella se aprecia la ecuación "TransformElements", que lleva a cabo su ejecución a través de los mecanismos de *pattern-matching* y de recursión. La generación de los términos resultantes "Table", "Column", "Key" se realiza a través del operador "New", el cálculo de sus respectivos identificadores se consigue con el operador "AddOID" y la asignación de valores y contenido a cada una de las propiedades de estos términos se obtiene por medio del operador asignación "←" y las navegaciones del modelo origen partiendo de su respectivo dominio en la regla como es el caso de "ecoreEClass0 :: name".

```
*** regla 1: ClassToTable
   eq TransformElements( ClassToTable , ? Set{ecoreEClass0} ? ecoreModel0 ,  TargetModel, Tuple2 ) =
          Set(
          AddOID(
             ( (New("Table", MM((empty-set).Set{rdbms}))).rdbmsNode
                 :: schema <-- ((p1
                         ((ModelGenRule ( PackageToSchema ,
                             ? ( Set{((ecoreEClass0 :: ePackage ( ecoreModel0 ))} -> flatten ) ? ecoreModel0 ,
                              TargetModel , Tuple2)
                         )) ))
                 :: name <-- (((ecoreEClass0 :: name)))
                 :: column <-- (Set ( AddOID(
                                 (New("Column", MM((empty-set).Set{rdbms}))).rdbmsNode
                                 :: name <-- (((((ecoreEClass0 :: name) + "_tid"))))
                                 :: type <-- ("NUMBER") )))
                 :: key <-- (Set ( AddOID(
                             (New("Key", MM((empty-set).Set{rdbms}))).rdbmsNode
                             :: name <-- (((((ecoreEClass0 :: name) + "_pk"))))
                             :: column <-- (Set ( AddOID(
                                         (New("Column", MM((empty-set).Set{rdbms}))).rdbmsNode
                                         :: name <-- (((((ecoreEClass0 :: name) + "_tid"))))
                                         :: type <-- ("NUMBER") )))
                          )))
             )))
   .
```

Figura 3. Proyección a Maude.

[4] Un constructor en Maude es un operador que permite definir información. En nuestro caso tendríamos un constructor para definir la instancia de una clase.

3.3. Ejecución y proyección a EMF

Llegado este punto, se dispone en el espacio tecnológico Maude de toda la información necesaria para lanzar la transformación a ejecución. Desde Maude, se aprovecha el mecanismo de reducción modulo las ecuaciones generadas a partir de un programa QVT Relations, como brevemente se indica en el apartado anterior y de *pattern-matching*, obteniendo el término resultante que en el ejemplo anterior se corresponde con un esquema relacional. Finalmente, este término es proyectado de vuelta a EMF utilizando los puentes definidos a nivel M1 entre ambos espacios tecnológicos. Se pueden encontrar más detalles del proceso de compilación a código Maude en [5].

4. TRABAJOS RELACIONADOS

Diversas son las herramientas y lenguajes que están apareciendo en el área de transformaciones de modelos. Estas aproximaciones generalmente se diferencian según la naturaleza del lenguaje (declarativo o imperativo) y la representación de los modelos que utilice. Como representación de modelos existen dos tendencias principales: grafos y conjuntos. A continuación detallamos alguna de las características de las herramientas más extendidas.

4.1. XSLT

XSLT es un lenguaje estándar de la *World Wide Web Consortium* (W3C) y cuenta con un extenso soporte tecnológico [7]. A pesar de satisfacer la gran mayoría de los requisitos necesarios por un lenguaje de transformación usado en *Model Driven Engineering* (MDE), resulta bastante complicado escribir transformaciones XSLT porque el usuario ha de tener en cuenta los esquemas XSD de entrada [8].

Como principales características, podemos decir que XSLT tiene una naturaleza híbrida pues permite tanto construcciones imperativas como declarativas. En su parte declarativa, la aplicación del *pattern-matching* se hace de forma ordenada y recursiva, por lo que XSLT es determinista. Además, desde la perspectiva de metamodelado, efectúa la transformación de grafos, que son árboles, como representación de modelos, y utiliza XPath como lenguaje de consulta.

Algunas de las restricciones existentes son: no tener disponibles transformaciones bidireccionales, ausencia de soporte de trazabilidad implícita, carencia de mecanismos automáticos de composición y no tener definida su sintaxis como instancia de MOF. Asimismo, XSLT no tiene una equivalencia directa con QVT y es difícil establecer una correspondencia entre ambos.

4.2. ATL

ATL forma parte del *framework* de gestión de modelos AMMA [9] que se encuentra integrado en Eclipse y EMF. Utiliza un lenguaje propietario llamado ATLAS para definir transformaciones que se ejecuta sobre JAVA y que proporciona un entorno de depuración. La naturaleza de ATLAS es declarativa, aunque lleva mucho tiempo utilizando también

construcciones imperativas, y las transformaciones son expresadas como reglas de transformación ya que ATL cumple el estándar MOF.
ATL posee un algoritmo de ejecución preciso y determinista, sin embargo no se apoya sobre ningún método formal. No proporciona mecanismos para la validación de las transformaciones. Para llevar a cabo transformaciones compuestas se necesita ejecutar cada una de las transformaciones participantes una a una.
Aunque la sintaxis de ATL es muy similar a la de QVT, ésta no es interoperable con QVT. No obstante, es posible definir transformaciones entre ambos espacios tecnológicos tal como se explica en [10].

4.3. VIATRA

Viatra actualmente forma parte del framework VIATRA2 [11], que ha sido escrito en Java y que se encuentra integrada en Eclipse. Viatra provee un lenguaje textual para describir modelos y metamodelos, y transformaciones llamados VTML y VTCL respectivamente.
La naturaleza del lenguaje es declarativa y está basada en técnicas de descripción de patrones, sin embargo es posible utilizar secciones de código imperativo. Se apoya en métodos formales como la transformación de grafos (GT) y la máquina de estados abstractos (ASM) para ser capaz de manipular modelos y realizar tareas de verificación, validación, seguridad, así como una temprana evaluación de características no funcionales como fiabilidad, disponibilidad y productividad del sistema bajo diseño.
Como puntos débiles podemos resaltar que Viatra no se basa en los estándares MOF y QVT. No obstante, pretende soportarlos en un futuro mediante mecanismos de importación y exportación integrados en el *framework*.

4.4. EPSILON

Epsilon es una plataforma desarrollada como un conjunto de *plug-ins* (editores, asistentes, pantallas de configuración, etc) sobre Eclipse. Presenta el lenguaje metamodelo independiente *Epsilon Object Language* que se basa en OCL [12]. Puede ser utilizado como lenguaje de gestión de modelos o como infraestructura a extender con nuevos lenguajes específicos de dominio. Tres son los lenguajes definidos en la actualidad: *Epsilon Comparison Language* (ECL), *Epsilon Merging Language* (EML), *Epsilon Transformation Language* (ETL), para comparación, composición y transformación de modelos respectivamente. Se da soporte completo al estándar MOF mediante modelos EMF y documentos XML a través de JDOM.
La finalidad perseguida con la extensión del lenguaje OCL es: dar soporte al acceso de múltiple modelos, ofrecer constructores de programación convencional adicionales (agrupación y secuencia de sentencias), permitir modificación de modelos, proveer depuración e informe de errores, así como conseguir una mayor uniformidad en la invocación. Soporta mecanismos de herencia, trazabilidad e introduce mecanismos de comprobación automática del resultado en composición y transformación de modelos [13].

5. CONCLUSIÓN

Las transformaciones de modelos juegan un papel importante en la iniciativa MDA [14]. Dadas estas necesidades, OMG ha propuesto el estándar QVT que proporciona diferentes aproximaciones para lenguajes de transformación de modelos. Nuestra aproximación constituye un primer prototipo ejecutable de transformaciones especificadas a través del lenguaje QVT Relations, proporcionando soporte para el *pattern matching* de reglas de transformación, trazabilidad y navegación mediante expresiones OCL. De este modo, consideramos que MOMENT-QVT es una buena alternativa en el campo de las transformaciones, ya que al estar basada en estándares aprovecha la reutilización de tecnología y se consigue reducir la curva de aprendizaje de la herramienta.

De acuerdo con nuestra experiencia trabajando con transformaciones en MOMENT-QVT, el lenguaje Relations es capaz de expresar transformaciones tanto sencillas como complejas; eso sí, es imprescindible conocer el lenguaje OCL con el propósito de añadir más potencia y funcionalidad a la propia transformación en su aplicación tanto a modelos origen como destino. Diversos ejemplos de transformaciones nos han permitido conocer el rendimiento actual del prototipo, concluyendo que la primera ejecución que tiene lugar en MOMENT-QVT tiene asociado un coste temporal cuatro veces superior al resto con un coste aceptable e inmediato, debido a la carga de módulos del *kernel*[5].

En cuanto al soporte actual, MOMENT-QVT únicamente soporta transformaciones unidireccionales. Sin embargo, es posible soportar bidireccionalidad en transformaciones de carácter endógeno o en aquellas de carácter exógeno, siempre que no haya pérdida de información, siendo además necesario que las manipulaciones que se realicen mediante expresiones OCL sean reversibles; por ejemplo la concatenación de *strings* no es reversible. También se está trabajando en el soporte de ejecución de transformaciones QVT Relations en modo *checkonly*, que se encargan de comprobar que las relaciones establecidas por los modelos participantes en la transformación se sostienen en todas direcciones. Así pues, en un breve espacio de tiempo, MOMENT-QVT será uno de los primeros prototipos ejecutables que cumplan completamente el lenguaje Relations de la especificación QVT. Como trabajos futuros pretendemos implementar mecanismos que permitan la herencia de transformaciones, así como comprobar y preservar la semántica de los modelos resultantes de las transformaciones.

REFERENCIAS

[1] MOF QVT Standard Specification. http://www.omg.org/docs/ptc/05-11-01.pdf

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

[3] Tom Mens, Pieter Van Gorp : *A Taxonomy of Model Transforamtion.* Proc. Int'l Workshop on Graph and Model Transformation (GraMoT 2005)

[5] Núcleo de MOMENT que contiene todos los módulos necesarios para ejecución.

[4] Artur Boronat, José Iborra, José A. Carsí, Isidro Ramos, Abel Gómez: *Utilización de Maude desde Eclipse Modeling Framework para la Gestión de Modelos.* Desarrollo de Software Dirigido por Modelos - DSDM'05 (Junto a JISBD'05). September 2005. Granada, Spain.

[5] Artur Boronat, José Á. Carsí, Isidro Ramos : *Algebraic Specification of a Model Transformation Engine.* LNCS. Fundamental Approaches to Software Engineering (FASE'06). ETAPS'06. Vienna (Austria). March 27-29. 2006.

[6] Artur Boronat, Joaquin Oriente, Abel Gómez, Isidro Ramos, José A. Carsí : *An Algebraic Specification for Generic OCL Queries within the Eclipse Modeling Framework.* Proceedings of Second European Conference on Model Driven Architecture. Bilbao (Spain). Springer LNCS. July 10th-13th 2006 (to appear)

[7] Anna Gerber, Michael Lawley, Kerry Raymond, Jim Steel and Andrew Wood: *Transformation: The Missing Link of MDA.* First International Conference, ICGT 2002, Barcelona, Spain, October 7-12, 2002

[8] Boronat A.,Carsí J.A., Ramos I.: *An Algebraic Baseline for Automatic Transformations in MDA.* Software Evolution Through Transformations: Model-based vs. Implementation-level Solutions Workshop (SETra'04), Second International Conference on Graph Transformation (ICGT2004), Electronic Notes in Theoretical Computer Scienc.

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

[10] Frédéric Jouault, Ivan Kurtev: *On the ArchitecturalAlignment of ATL and QVT*. Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 06). ACM Press, Dijon, France, chapter Model transformation (MT 2006)

[11] Sitio web VIATRA: http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/subprojects/VIATRA2/index.html

[12] Dimitiros S. Kolovos, Richard F. Paige, and Fiona A.C Polack: *The Epsilon Object Language (EOL).* Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 2006.

[13] Dimitiros S. Kolovos, Richard F. Paige, and Fiona A.C Polack: *Model Comparison: A Foundation for Model Composition and Model Transformation Testing*. First International Workshop on Global Integrated Model Management (G@MMA) 2006, co-located with ICSE'06, Shanghai, China, May 2006.

[14] Shane Sendall and Wojtek Kozaczynski. *Model Transformation – the Heart and Soul of Model-Driven Software Development.* IEEE Software, Special Issue on Model Driven Software Development, pages 42--45, Sept/Oct 2003.