Patrón para la Especificación de Transformaciones (Pattern for Specification of Transformations)*

Patricia Morantes
Universidad Central de Venezuela
Centro ISYS, Escuela de Computación, Caracas, Venezuela
Universidad Nacional Experimental
"Francisco de Miranda", Área de Tecnología, Coro, Venezuela
pmorantes@unefm.edu.ve

and

Alfredo Matteo
Universidad Central de Venezuela
Centro ISYS, Escuela de Computación
Caracas, Venezuela
almatteo@cantv.net

Resumen

Los modelos y las transformaciones son parte clave en el enfoque de Ingeniería de Modelos, en particular para MDA. La definición y/o especificación de las transformaciones entre modelos es un aspecto fundamental dentro de este enfoque. En tal sentido, en este trabajo se propone un patrón de especificación como marco para la definición de transformaciones. Este patrón se fundamenta en un modelo de características de lenguajes de transformaciones basados en la propuesta MOF/QVT de la OMG. La especificación obtenida es independiente del lenguaje de transformación utilizado, y constituye una ayuda para la documentación y su implementación en el lenguaje de transformación seleccionado.

Palabras claves: Ingeniería de Modelos, MDA, Transformación de modelos, MOF/QVT.

Abstract

The models and transformations are a key part in Model Driven Engineering approach, particularly in MDA. Definitions and/or specification of transformations between models is a fundamental feature in this approach. This paper proposes a pattern specification as a frame for the transformation definitions. This pattern is based on a model of transformations language features based on the MOF/QVT proposition of OMG. The obtained specification is independent of the transformation language used and it is aide documentation and its implementation in the select transformation language.

Keywords: Model Driven Engineering, MDA, Models Transformations, MOF/QVT.

*Esta investigación es financiada por el CDCH - Universidad Central de Venezuela. Proyecto No. PI 03-00-6051-2005 y Programa Alma Mater de la Oficina de Planificación del Sector Universitario (OPSU).

1. Introducción

La aparición de nuevas tecnologías aumenta la complejidad y las exigencias en el desarrollo de sistemas de software, en cuanto a la integración de nuevos elementos, adaptación a cambios de plataformas y coordinación en la integración de aplicaciones heterogéneas. La comunidad de la Ingeniería de Software esta asumiendo nuevos retos en el desarrollo de sistemas complejas adoptando nuevos paradigmas. Actualmente se esta ante un nuevo paradigma denominado Ingeniería de Modelos (IM) o lo que se conoce también por Desarrollo de Software Dirigido por Modelos (DSDM) (en ingles: Model Driven Engineering, MDE).

En la IM el principio base es "todo es modelo", es decir, que una vista particular de un sistema sea capturado por un modelo y luego cada modelo sea escrito en un lenguaje de su metamodelo [3].

Los modelos permiten visualizar cómo es o se quiere que sea el sistema, especifican la estructura y comportamiento de un sistema, documentan las decisiones adoptadas, sirven de guía en la construcción del software y permiten comunicar las características claves del sistema a las partes involucradas.

El enfoque MDA (Model Driver Architecture) de la OMG (Object Management Group) aparece como una de las posibles alternativas para satisfacer los principios de la IM. Este enfoque esta basado en las siguientes especificidades [11]: MOF, UML 2.0, perfiles UML, CWN, SPEM, EDOC, MOF/QVT, entre otros.

El enfoque MDA constituye una aproximación para el desarrollo de software dirigido por modelos. Este enfoque plantea una serie de conceptos básicos como son: el uso de modelos en todas las etapas del desarrollo de software, la definición de transformaciones entre modelos y la separación de las especificaciones correspondiente a las funcionalidades del sistema con respecto a sus implementaciones en una plataforma específica [11].

A través de MDA se definen diferentes tipos de modelos, Modelo Independiente de la Computación (CIM), Modelo Independiente de la Plataforma (PIM) y Modelo Especifico de la Plataforma (PSM), persiguiendo elevar el nivel de abstracción en el desarrollo de software dándole mayor importancia al modelado conceptual y a la definición de transformaciones entre modelos [11, 13].

Una transformación es la generación de un modelo destino a partir de un modelo fuente. Una definición de transformación es un conjunto de reglas de transformación que juntas describen como un modelo en el lenguaje fuente puede ser transformado en un modelo en el lenguaje destino [13].

Las transformaciones de modelo se pueden especificar en lenguajes de programación estándares tales como C, Java o Python, convirtiéndose en una tarea difícil que requiere gran habilidad de parte del desarrollador. Las transformaciones descritas a bajo nivel, son poco reutilizables y carecen de características para facilitar las actualizaciones entre dos modelos transformados.

La OMG propone un estándar denomina MOF 2.0/QVT (Query/View/Transformation) basado principalmente en la definición de un lenguaje para las consultas (Queries) sobre los modelos MOF, la búsqueda de un estándar para generar vistas (Views) que revelen aspectos específicos de los sistemas modelados, y finalmente, la definición de un lenguaje para la descripción de transformaciones (Transformations) de modelos MOF (Meta Object Facility) [13].

El objetivo de este trabajo es la definición de un patrón que permita la especificación de una transformación basado en un modelo de características para lenguajes de transformación MOF/QVT. Este patrón constituirá una ayuda para la documentación y la implementación de una transformación entre modelos.

El artículo a parte de la introducción y conclusiones, esta estructurado en tres secciones: la sección 2 propone un modelo para la caracterización de los Lenguajes de Transformación bajo la propuesta MOF/QVT. La sección 3 describe la propuesta del patrón de especificación para transformaciones. La sección 4 se presenta un caso de estudio para la especificación de una transformación bajo el patrón propuesto.

2. Modelo de Características para Lenguajes de Transformación basados en MOF/QVT

Actualmente se disponen de múltiples lenguajes de transformaciones que son proposiciones a ser utilizadas en la Ingeniería de Modelos. Entre estos lenguajes se pueden mencionar los siguientes: UMLX [7, 16], ATL [8, 10], YATL [14, 15], AndroMDA [1], FUUT-je [6], ArcStyler [9], OptimalJ [4, 5].

Algunos de estos como AndroMDA [1], FUUT-je [6], han sido desarrolladas en código abierto permitiéndose la incorporación de mecanismos como solución al problema de las transformaciones. Otras de corte comercial y orientadas al enfoque MDA son ArcStyler [9] y OptimalJ [4, 5].

Los lenguajes UMLX [7, 16], ATL [8, 10] y YATL [14, 15] siguen la propuesta MOF/QVT, en donde se proporciona una sintaxis concreta a partir de la sintaxis abstracta del metamodelo definido en MOF/QVT; ATL y YATL son lenguajes de transformación textual e híbrido (declarativo e imperativo), UMLX es un lenguaje de transformación gráfico, que a sido definido como un perfil de UML al incorporar diagramas de transformación.

En este trabajo se consideran los lenguajes de transformación definidos bajo la propuesta MOF/QVT [13]. En tal sentido se establece un modelo de características (ver tabla 1) que deben satisfacer los lenguajes definidos bajo esta propuesta. Así dado un lenguaje de transformación, utilizando el modelo de características se determina si este satisface en mayor o menor grado las características MOF/QVT. Si se disponen de varios lenguajes de transformación, el modelo de características permite seleccionar el que mejor se adapte a las necesidades del problema de transformaciones a solucionar.

El modelo propuesto clasifica las características en obligatorias y opcionales. El conjunto de características obligatorias que debe cumplir un lenguaje de transformación basado en MOF/QVT son [13]:

(C1) Permite la definición de transformaciones basadas en MOF 2.0.

(C2) Posee un lenguaje QUERY, es decir, se dispone de un lenguaje de consultas, que permita filtrar y seleccionar elementos de los modelos MOF 2.0.

(C3) Posee un Lenguaje Vista, es decir, se dispone de un lenguaje que permite generar una abstracción de los modelos padres.

(C4) Permite la Transformación automática, es decir, el lenguaje permite expresar toda la información necesaria para generar automáticamente la transformación de un modelo origen a uno destino.

(C5) Soporta consistencia incremental, es decir, el lenguaje permite que los cambios incrementales en el modelo origen sean reflejados en el modelo destino.

El conjunto de características opcionales que debe cumplir un lenguaje de transformación basado en MOF/QVT son [13]:

(C6) Permite la definición de una transformación en forma textual o gráfica.

(C7) Permite la definición de una transformación en forma declarativa, imperativa o híbrida.

(C8) Permite la descripción de dominios, es decir, la definición de un conjunto de variables asociados a un tipo de modelo.

(C9) Permite declaración de variables, es decir, permite la declaración de un conjunto de variables asociadas a tipos de datos OCL 2.0 (Object Constraint Language) [12] o a tipos de datos definidos en lenguaje natural.

(C10) Permite la declaración de los metamodelos fuentes y destinos asociados una transformación.

(C11) Permite la definición de reglas de transformación, es decir, la descripción necesaria para ejecutar una transformación.

(C12) Permite la definición de funciones, es decir, la descripción de un conjunto de operaciones que pro-

ducen el mismo resultado cada vez que son invocadas con los mismos parámetros.

(C13) Permite la ejecución paso a paso de la transformación definida, es decir, se posee un ambiente de ejecución paso a paso (debugger) de la transformación definida.

(C14) Posee mecanismos de reuso y extensión, es decir, posee el mecanismo de la herencia para la extensibilidad, y la reutilización de transformaciones simples para la definición de transformaciones más complejas, así como el uso de plantillas o patrones de transformación.

(C15) Permite en la ejecución de una transformación transaccional, es decir, posee un ambiente de ejecución en el cual se puede realizar acciones de "commit" o "rollback" durante la definición de la transformación.

(C16) Permite la definición de una transformación bidireccional, es decir, se puede definir una transformación de forma simétrica, en donde una única transformación sirva para las dos direcciones. o se definan transformaciones, donde una es la inversa de la otra.

(C17) Permite especificaciones en Lenguaje OCL 2.0, es decir, se pueden describir expresiones de consulta, definir reglas de transformación y funciones utilizando OCL 2.0.

(C18) Permite definir parámetros adicionales, es decir, definición de parámetros adicionales no contenidos en el modelo origen para generar el modelo destino.

Id	Características	Id	Características
C1	Permite la definición de transformaciones basadas en MOF 2.0	C10	Permite la declaración de los metamodelos fuentes y destinos.
C2	Posee un lenguaje QUERY	C11	Permite la definición de reglas de transformación.
C3	Posee un Lenguaje Vista	C12	Permite la definición de funciones.
C4	Permite la Transformación automática	C13	Permite la ejecución paso a paso de la transformación definida.
C5	Soporta consistencia incremental	C14	Posee mecanismos de reuso y extensión.
C6	Permite la definición de una transformación en forma textual o gráfica.	C15	Permite en ejecución realizar una transformación transaccional.
C7	Permite la definición de una transformación en forma declarativa, imperativa o híbrida.	C16	Permite la definición de una transformación bidireccional.
C8	Permite la descripción de dominios.	C17	Permite especificaciones en Lenguaje OCL 2.0
C9	Permite declaración de variables.	C18	Permite definir parámetros adicionales.

Tabla 1: Modelo de Características para Lenguajes de Transformación basados en MOF/QVT

Un lenguaje de transformación cuando satisface todas las características obligatorias descritas en el modelo propuesto esta dentro del grupo de lenguajes de transformación basados en MOF/QVT, de lo contrario se puede decir que el lenguaje no es un lenguaje de transformación basado en MOF/QVT.

3. Patrón para la Especificación de Transformaciones

En este artículo se presenta un Patrón de Especificación de Transformaciones de Modelos llamado PETM (Ver tabla 2), el cual es descrito utilizando la sintaxis EBNF (Extended Backus Naur Form). Este patrón aporta

un lenguaje común de referencia para personas implicadas en el desarrollo dirigido por modelos, evitando ambigüedades o malas interpretaciones, mejorando la comunicación y comprensión entre los miembros del equipo de desarrollo.

El PETM se apoya en el Template de Especificación ATL [2] y en el modelo de características mostrado en el tabla 1. A continuación se describe este patrón y se muestra la relación con el modelo características. En el PETM se específica el nombre y resumen de la transformación que definen de manera textual la transformación, la declaración de los metamodelos fuentes y destinos, a través de una representación textual y/o gráfica (C10). Adicionalmente pueden definirse precondiciones al metamodelo fuente y postcondiciones aplicadas al metamodelo destino, que complementan la descripción de los metamodelos. El dominio de la transformación es descrito a través de la declaración de modelos (C8). Las librerías y transformación simples que se van a importar en la transformación son llamadas mecanismos de reuso y extensión (C14). La declaración de variables globales se especifica usando los tipos de datos OCL 2.0 o a través de pseudo código (C9). Las funciones son declaradas con un nombre y una lista de parámetros opcionales, además se especifica el tipo de dato a retornar y el cuerpo de la función (C12). Las reglas de transformación son declaradas con un nombre y un cuerpo con la información necesaria para realizar la transformación entre modelos (C11). Por último todas las especificaciones pueden ser descritas en pseudo código o lenguaje OCL 2.0.(C6).

Transformación: <Id_transf>
Resumen: <Descripción de la transformación>
Metamodelos Fuentes: <nombre_mm> <descripción_textual_mm> | <descripción_gráfica_mm> Pre-conditions: <Descripción_pseudos_código> | <Descripción_condicional_OCL>
Metamodelos Destinos: <nombre_mm> <descripción_textual_mm> | <descripción_gráfica_mm> Post-conditions: <Descripción_pseudos_código> | <Descripción_condicional_OCL>
Dominio: { <nombre_m> } : <nombre1_mm>
Mecanismos de rehusó y extensión: { <Nombre_Lib>} | { <Nombre_TS> }
Variables Globales: { <nombre_V:Tipo> }
Funciones: <Función> ::= <Función> <Nombre_F> <Parámetros de la función>: <Tipo_Retorno> <Descripción_pseudos_código> | <Descripción_OCL>
Reglas de Transformación <Regla de Transformación> ::= <Regla> <nombre_R> <Desde> <Descripción_pseudos_código> | <Descripción_OCL> <Para> <Descripción_pseudos_código> | <Descripción_OCL>

Tabla 2: Patrón de Especificación de Transformaciones PETM

Este patrón puede ser utilizado para la definición transformaciones basadas en la propuesta MOF/QVT ya que se fundamenta en el modelo de características definido en la sección 2. Sin embargo, también puede ser usado para especificar transformaciones que serán implementadas en cualquier otro lenguaje de transformación que no este basado en MOF/QVT, ya que el patrón contempla algunas de las características que debe tener un lenguaje de transformación en el marco de la IM. Esta características son la definición de metamodelos, librerías, variables, dominios, funciones y reglas de transformación.

4. Caso de Estudio

El caso de estudio presenta una transformación donde el modelo fuente consiste de dos clases que representan a un conjunto de alumnos y las notas de las materias que están cursando y se desea obtener un modelo destino formado por una clase que representa a los alumnos que tengan promedio de notas mayor o igual a 18 puntos, los cuales son considerados alumnos sobresalientes.

El objetivo es usar el PETM en la especificación de la Transformación Alumnos del Cuadro de Honor (Ver tabla 3) donde el modelo Alumno es transformado en el modelo Alumno Sobresaliente.

El metamodelo Alumno (Figura 1) contiene la clase Alumno que esta relacionada con las notas de los alumnos por materia.

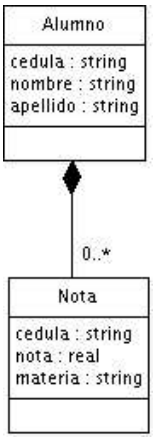


Figura 1: Metamodelo Alumno Regulare

El metamodelo Alumno Sobresaliente (Figura 2) consiste en una clase Alumno_S el cual contiene cédula, nombre, apellido y la nota promedio solo de aquellos alumnos que tengan un promedio mayor o igual a 18 puntos.

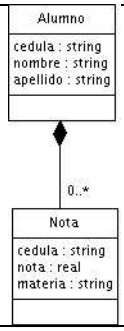
Figura 2: Metamodelo Alumno Sobresaliente

Transformación: AlumnoCuadroHonor

Resumen: El objetivo es generar un modelo Alumno Sobresaliente desde un modelo Alumno. Los distintos elementos del modelo Alumno Sobresaliente son generados por cada elemento Alumno solo si tienen el promedio de notas mayor o igual a 18.

Metamodelos Fuentes:

```
paquete Alumno {
    clase Alumno {
        cedula: string
        nombre: string
        apellido:string }
    clase Notas {
        cedula: string
        nota: real }
        materia: string }}
```



Metamodelos Destinos:

```
paquete Alumno_S {
    clase Alumno_S {
        cedula: string
        nombre: string
        apellido:string
        promedio: real }}
```

Dominio:

```
alumno: Alumno
alumno_s: Alumno_S
```

Variables Globales:

```
Promedio: real
```

Funciones:

```
Funcion sumarnotas() : Real =
    self.notas->collect(b|b.notas)
        ->iterate(nnotas; n : Integer = 0 |
            (n + nnotas))
Funcion contarnotas() : Real =
    self.notas->collect(b|b.notas)
        ->iterate(nnota; cont : Integer = 0 |
            (cont+1))
```

Reglas de Transformación:

```
Regla AlumnoCuadroHonor
    Desde: a: Alumno
        promedio = a.sumarnotas()/a.contarnotas()
        if promedio>=18
    Para: ch: Alumno_S
        ch.cedula = a.cedula
        ch.nombre = a.nombre
        ch.apellido = a. Apellido
        ch.promedio = promedio
```

Tabla 3: Transformación AlumnoCuadroHonor

A continuación se presenta la implementación de la Transformación AlumnoCuadroHonor en el lenguaje ATL [8] usando como referencia el PETM:

El Metamodelo Alumno:

```
package Alumno {
    class Alumno {
        attribute cedula:String;
        attribute nombre:String;
        attribute apellido:String;
        }
    class Nota {
        attribute cedula:String;
        attribute nota:Integer;
        attribute materia:String;
        }
    }
package PrimitiveTypes {
    datatype String;
        datatype Integer
    }
```

El Metamodelo Alumno Sobresaliente:

```
package Alumno_S {
    class Alumno_S {
        attribute cedula:String
        attribute nombre:String;
        attribute apellido:String;
        attribute promedio:Integer;
        }
package PrimitiveTypes {
        datatype String;
        datatype Integer
    }
```

Transformación AlumnoCuadroHonor:

```
module AlumnoCuadrohonor;
create OUT : Alumno_S from IN : Alumno;
helper context Alumno!Alumno def : getSumarNotas() : Integer =
    self.nota− >collect(b|b.nota)
        − >iterate(nnota; n : Integer = 0 |
        n + nnota
        );
helper context Alumno!Alumno def : getContarNotas() : Integer =
    self.nota− >collect(b|b.nota)
        − >iterate(nnota; cont : Integer = 0 |
        cont+1
        );
rule AlumnoCuadrohonor {
        from
            a : Alumno!Alumno (
                promedio = a.getSumarNotas()/a.getContarNotas(),
                if promedio>=18
```

```
        )
    to
        ch : Alumno_S!Alumno_S (
            cedula <- a.cedula,
            nombre <- a.nombre,
            apellido <- a. apellido,
            promedio <- promedio
        )
}
```

Los metamodelos de la transformación AlumnoCuadroHonor fueron implementados en el formato KM3 (Kernel MetaMetaModel), esta es una notación textual simple de la herramienta de desarrollo ATL [8]. La implementación de la transformación AlumnoCuadroHonor fue desarrollada siguiendo la sintaxis y semántica de ATL y las instrucciones del lenguaje de restricción OCL 2.0. [12].

5. Conclusiones

La Ingeniería de Modelos es un nuevo paradigma en la Ingeniería de Software que tiene como principio el uso de modelos y transformaciones en el desarrollo de sistemas de software. El enfoque MDA de la OMG es una aproximación al desarrollo dirigido por modelos. En este artículo se presentó el Patrón de Especificación de Transformación denominado PETM el cual esta basado en un modelo de características para lenguajes de transformación MOF/QVT. El patrón contempla algunas de las características que debe tener un lenguaje de transformación en el marco de la IM, como son la definición de metamodelos, librerías, variables, dominios, funciones y reglas de transformación. El PETM permite especificar las transformaciones entre modelos antes de su implementación, aportando una solución al problema de documentación y comunicación de experiencias. Este patrón puede ser utilizado y/o incorporado en el marco de un ambiente de desarrollo de software bajo el enfoque de IM, en particular para MDA, como un artefacto del proceso que eventualmente pudieran ser reutilizados para la definición de transformaciones mas complejas.

Referencias

[1] AndroMDA. [Página Web en línea]. Disponible: http://www.andromda.org/

[2] ATLAS group. LINA INRIA. *ATL Transformation Description Template*. Version 0.1. Nantes. (December 2005).

[3] Bézivin, J. In Search of a Basic Principle for Model Driven Engineering, *Novática* No. 1, (June 2004), pp. 21-24.

[4] Compuware. OptimalJ, [Página Web en línea]. Disponible: http://www.compuware.com/products/optimalj/

[5] Dept. Computer Science, King's College London. An Evaluation of Compuware OptimalJ Professional Edition as a MDA Tool. Disponible: http://www.lcc.uma.es/~av/MDD-MDA/publicaciones/P_13kings_mda.pdf

[6] Eclipse GMT. Fuut-je. [Página Web en línea]. Disponible: http://www.eclipse.org/gmt/

[7] Eclipse GMT. UMLX. [Página Web en línea]. Disponible: http://www.eclipse.org/gmt/

[8] INRIA. The Atlas Transformation Language (ATL). [Página Web en línea]. Disponible: http://modelware.inria.fr/rubrique12.html

[9] Interactive Object. ArcStyler. [Página Web en línea]. Disponible: http://www.interactive-objects.com/products/download/arcstyler-for-ibm-rsm-edition

[10] Jouault, F. and Kurtev, I. Transforming Models with ATL. MoDELS 2005 Conference. *Lecture Notes in Computer Science*, Springer-Verlag. Volume 3844, (2006), pp. 128-138.

[11] Miller, J. and Mukerji, J. *MDA. Guide Version 1.0.1*, Object Management Group OMG. Document omg/2003-06-01, (June 2003).

[12] Object Management Group. *UML 2.0 OCL Specification*. OMG Document ptc/03-10-14, (October 2003).

[13] Object Management Group: QVT-Partners. Revised Submission for MOF 2.0 Query/Views/Transformations RFP version 2.0. OMG document ad/2005-03-02, (March 2005).

[14] Patrascoiu, O. Yet Another Transformation Language. *Proceedings of the 1st European MDA Workshop, MDA-IA*. (January 2004). pp. 83-90.

[15] Patrascoiu, O. and Rodgers P. Embedding OCL expressions in YATL. *Software and Systems Modelling Journal*. Vol.4, No.3, (August 2005).

[16] Willink, E. UMLX: A graphical transformation language for MDA. A. Rensink (Ed.), Model Driven Architecture: Foundations and Applications (2003), *CTIT Technical Report TR-CTIT-03-27*, (2003).