# Un lenguaje para Transformación de Modelos basado en MOF y OCL

**Roxana S. Giandini** **Claudia F. Pons**

**Universidad Nacional de La Plata, Facultad de Informática**
**LIFIA - Laboratorio de Investigación y Formación en Informática Avanzada**
**La Plata, Argentina, 1900**
**[giandini, cpons]@lifia.info.unlp.edu.ar**

**Resumen**

La iniciativa MDD (*Model Driven Development*) cubre un amplio espectro de áreas de investigación como: lenguajes de modelado, definición de lenguajes de transformación entre modelos y construcción de herramientas de soporte.
Actualmente, algunos de estos aspectos están siendo fundamentados y aplicados, mientras otros están en proceso de definición. Consecuentemente son necesarios esfuerzos que conviertan a MDD en una propuesta coherente, soportada por técnicas y herramientas maduras. Las transformaciones entre modelos requieren de lenguajes específicos para su definición. Estos lenguajes deben tener base formal, por ejemplo tener un metamodelo que los sustente, y permitir un tratamiento automatizado.
En este trabajo presentamos un lenguaje declarativo para transformaciones de modelos inspirado en estándares de OMG. Nuestra propuesta constituye una extensión de especificaciones ya existentes en OMG y utiliza OCL para especificar relaciones de transformación.

**Palabras Claves:** Ingeniería de Software, Desarrollo Dirigido por Modelos, Transformación de Modelos, Extensión de Metamodelos

**Abstract**

MDD (*Model Driven Development*) initiative covers a broad spectrum of research areas such as modelling languages, definition of transformation languages among models, and construction of support tools.
Currently, some of these aspects are being established and applied, while others are still in the process of definition. Consequently, it is necessary to make every effort to turn a MDD into a coherent proposal, supported by mature tools and techniques. Transformations among models require specific languages for their definition. These languages must have a formal base, for example, a metamodel that supports them and allows for an automated treatment.
This paper presents a declarative language for model transformations inspired on OMG standards. Our proposal pretends to be a minimal extension of the already existing OMG specifications, and it basically uses OCL language to specify transformation relations.

**Keywords:** Software Engineering, Model Driven Development, Model Transformation, Metamodel Extension.

## 1. INTRODUCCION

El paradigma MDD [1] tiene dos ejes principales: - por un lado hace énfasis en la separación entre la especificación de la funcionalidad esencial del sistema y la implementación de dicha funcionalidad usando plataformas tecnológicas específicas. Para ello, el MDD identifica dos tipos principales de modelos: modelos con alto nivel de abstracción e independientes de cualquier tecnología de implementación, llamados PIM (Platform Independent Model) y modelos que especifican el sistema en términos de construcciones de implementación disponibles en alguna tecnología específica, conocidos como PSM (Platform Specific Model); - por otro lado, los modelos son considerados los conductores primarios en todos los aspectos del desarrollo de software. Un PIM es transformado en uno o más PSMs, es decir que para cada plataforma tecnológica específica se genera un PSM especifico. La transformación entre modelos constituye el motor del MDD y de esta manera los modelos pasan de ser entidades meramente contemplativas a ser entidades productivas.

La iniciativa MDD cubre un amplio espectro de áreas de investigación: lenguajes para la descripción de modelos, definición de lenguajes de transformación entre modelos, construcción de herramientas de soporte a las distintas tareas involucradas, aplicación de los conceptos en métodos de desarrollo y en dominios específicos, etc.
Actualmente, algunos de estos aspectos están bien fundamentados y se están empezando a aplicar con éxito, otros sin embargo están todavía en proceso de definición. En este contexto son necesarios esfuerzos que conviertan MDD y sus conceptos y técnicas relacionados en una propuesta coherente, basada en estándares abiertos, y soportada por técnicas y herramientas maduras. Las transformaciones entre modelos requieren de lenguajes específicos para su definición. Estos lenguajes deben tener base formal, por ejemplo tener un metamodelo que los sustente, y permitir un tratamiento automatizado.
En este trabajo presentamos un lenguaje puramente declarativo para expresar transformaciones entre modelos que inspira su metamodelo inicial en el lenguaje QVT[4] que es la especificación de OMG[2] para transformaciones, aún en proceso de definición. El lenguaje que proponemos está orientado a ser el mínimo para poder expresar relaciones y *querys* de transformación entre modelos.
La organización del artículo es la siguiente: en la sección 2 se introduce, discute y ejemplifica el concepto de lenguajes de transformación concretando el enunciado de nuestra propuesta. La sección 3 presenta paso a paso la construcción de un perfil para transformaciones, basado en la definición de estereotipos. En la sección 4 se aplica el perfil a un ejemplo. Finalmente presentamos conclusiones y líneas de trabajo futuro.

## 2. METAMODELOS Y LENGUAJES DE TRANSFORMACIÓN DE MODELOS

El metamodelado es un mecanismo que permite construir formalmente lenguajes de modelado, como lo son el UML y el RDBMS. La Arquitectura 4 capas de Modelado es la propuesta de OMG orientada a estandarizar conceptos relacionados al modelado, desde los más abstractos a los más concretos.
Los niveles definidos en esta arquitectura se denominan comúnmente: M3, M2, M1, M0:
El **nivel M3** (el meta-metamodelo) es el nivel más abstracto, donde se encuentra el MOF [3], que permite definir metamodelos concretos (como el del lenguaje UML)
El **nivel M2** (el metamodelo), sus elementos son lenguajes de modelado, por ejemplo UML. Los conceptos a este nivel podrían ser Clase, Atributo, Asociación.
El **nivel M1** (el modelo del sistema), sus elementos son modelos de datos, por ejemplo entidades como "Persona", "Auto", atributos como "nombre", relaciones entre estas entidades.
El **nivel M0** (instancias) modela al sistema real. Sus elementos son datos, por ejemplo "Juan López", que vive en "Av. Libertador 345"

### 2.1 La Transformación de Modelos en la Arquitectura 4 capas de Modelado

En este contexto, la definición de lenguajes para transformación de modelos puede pensarse en la capa M3 de la Arquitectura de modelado 4 capas, ya que una instancia específica de transformación se ubica en la capa M2 para poder relacionar instancias genéricas de metamodelos concretos (que se ubican en M2) como el de UML y el de RDBMS, entre cuyas instancias se produce la transformación (por ejemplo una Class de UML y una Table de RDBMS). Es decir, los modelos que concretamente están involucrados en la transformación (capa M1) son parámetros para el lenguaje de transformación.
Es lógico pensar que el metamodelo para transformaciones y su instanciación no pueden convivir en la misma capa, ya que representan distintos niveles de abstracción.

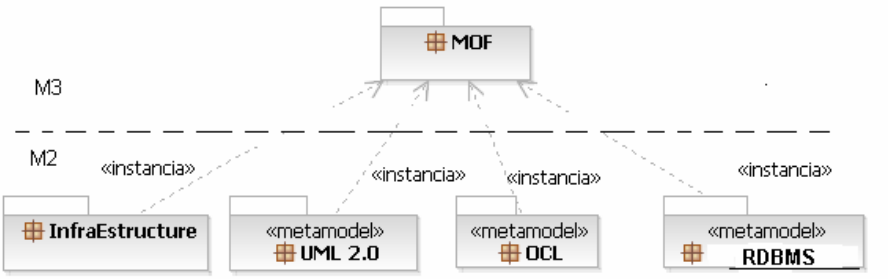


Figura 1. Capas M3 y M2 de la Arquitectura 4 capas

Ahora bien, en la capa M3 como ya dijimos, se ubica MOF, que representa un Meta-metamodelo cerrado sobre el que se instancian metamodelos (instancias de MOF). En consecuencia, el metamodelo para Transformaciones debería ubicarse

en la capa M2, junto con el resto de los metamodelos (por ejemplo el de UML, de OCL, etc.) como puede verse en la Figura 1. Analicemos otras especificaciones, estrechamente vinculadas a MOF, que define OMG: una de estas especificaciones es la Infraestructura [7] para lenguajes de modelado. La Infraestructura es la especificación más simplificada que define los constructores básicos y conceptos comunes para lenguajes de modelado. Podemos decir que es independiente al lenguaje UML en sí. El metamodelo de UML se complementa con la especificación de la Superestructura [6], que define los constructores a nivel usuario de UML 2.0.
El caso de la Infraestructura es interesante por su definición recursiva respecto a MOF: por un lado, puede verse definida como instancia de MOF (en la capa M2 como muestra la Figura 1); por otro lado el MOF mismo se basa, o bien usa elementos del paquete Core de la Infraestructura para su definición, situación que nos permite identificar a la Infraestructura como un meta-metamodelo. Si definimos el nuevo metamodelo para transformaciones como una extensión de elementos del paquete Core, es discutible pensar que subyace en el nivel M3, junto a la Infraestructura (ver Figura 2). Luego, una instancia del metamodelo Transformación en la capa M2 relaciona elementos de metamodelos, mientras que en la capa M1 existirá un link (enlace de correspondencia) entre modelos, instancias de dichos metamodelos, que son las que concretamente se transforman.

**2.2 Nuestra Propuesta: Transformaciones Declarativas puras**

En el documento de especificación de QVT, que es la propuesta estándar de OMG para transformaciones, el metamodelo se define como "extensión de MOF y de OCL [5]". Desde nuestro punto de vista, según lo analizado anteriormente y según las definiciones de OMG no es técnicamente correcto extender a MOF ya que representa un Meta-metamodelo cerrado sobre el que se instancian metamodelos. Por lo tanto proponemos *extender* la Infraestructura 2.0 y *usar* OCL 2.0 para implementar el metamodelo de las transformaciones.
Gran parte de los lenguajes propuestos para transformación de modelos [8, 9, 10], se basan en el lenguaje QVT. La definición de QVT consta de 3 paquetes: 2 declarativos (Relations y Core) y uno imperativo. En su mayoría, estas propuestas de lenguajes son híbridas: declarativas e imperativas. Para definir una relación de transformación entre modelos, bastaría especificarla declarativamente y posteriormente elegir en que lenguaje imperativo se podrá ejecutar y recién entonces escribir sentencias ejecutables que se correspondan a las declaraciones. Por lo tanto optamos proponer un lenguaje declarativo puro para transformaciones, cuyo metamodelo inicial se inspira en el paquete Relations de QVT. Luego lo implementamos como una extensión (perfil) de la Infrastructura y utilizamos OCL para completar dicha implementación. Este metamodelo pretende ser el mínimo para poder expresar relaciones y *querys* de transformación entre modelos.

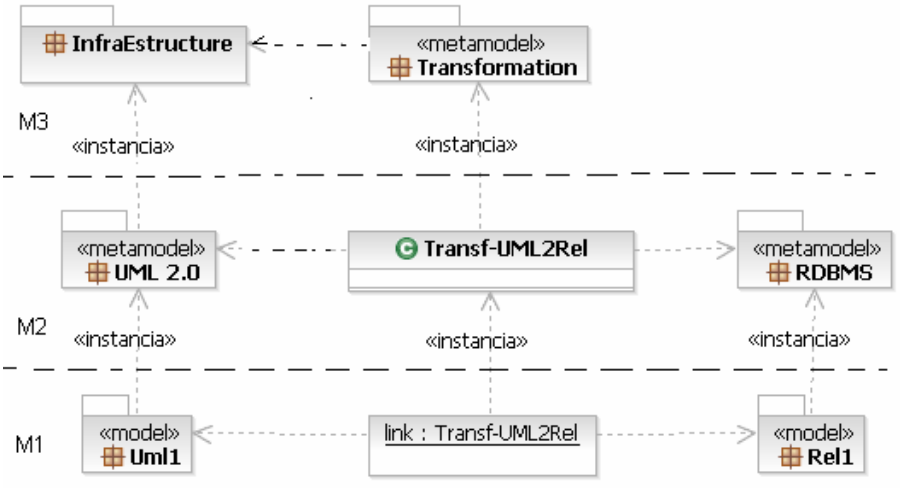


Figura 2. La Transformación de Modelos en la Arquitectura 4 capas

**2.3 Ejemplo de Transformación de Modelos**

El siguiente ejemplo extraído del manual de QVT, muestra la especificación textual de una transformación llamada UMLToRdbms que define una relación UML2Rel que transforma clases UML (que cumplan la restricción de ser persistentes, indicado en la cláusula **when** de esta relación) a tablas del Modelo Relacional con el mismo nombre. Además la transformación tiene otra relación Attr2Col (aparece indicada, sin completar), que especifica que los atributos de una clase, se corresponden con columnas del mismo nombre en la tabla correspondiente. Esta relación es invocada en la cláusula **where** de la relación UML2Rel y significa que cada vez que UML2Rel se cumple para una clase y una tabla, la relación Attr2Col para sus atributos y columnas respectivamente, se cumple también:

```
Transformation UMLToRdbms (Uml1: UML2.0, Rel1: RDBMS)
{
    Relation UML2Rel {
    checkonly domain Uml1 c: Class {name = n}
    checkonly domain Rel1 t: Table {name = n}
    when {c.isPersistent( )}
    where { Attr2Col(c, t)}
    }
    Relation  Attr2col (c, t ) { …
    }
}
```

Gráficamente, una instanciación de esta transformación podría ser, por ejemplo, la que se muestra en la Figura 3 entre dos modelos concretos Uml1 (de UML) y Rel1 (de Rdbms), donde n = 'Persona' y un nombre de atributo podria ser 'nombre'. O sea, para la clase Persona del modelo Uml1, existe una Tabla en el modelo relacional Rel1 con el mismo nombre. Lo mismo sucede con los atributos: para cada atributo de la clase Persona existe una columna en la tabla Persona con el mismo nombre. O sea, para el atributo `nombre´, existe la columna `nombre´.

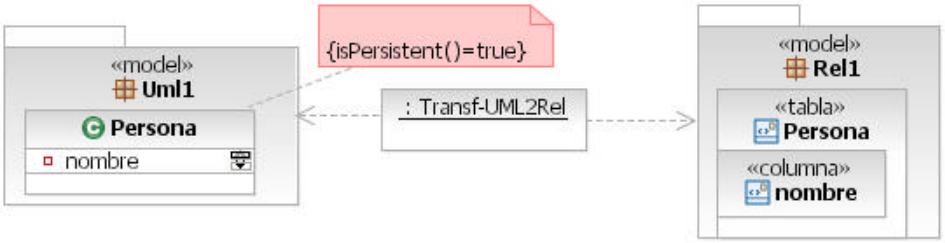


Figura 3. Una instanciación gráfica de la Transformación.

## 3. CONSTRUCCIÓN DE UNA EXTENSIÓN DE LA INFRAESTRUCTURA Y OCL PARA TRANSFORMACIÓN DE MODELOS.

Debido a lo expuesto en la sección anterior, nuestra propuesta es extender la Infraestructura, creando un perfil específico (instancia de la Clase Profile de la Infraestructura) que permita expresar características particulares de la transformación de modelos.
La extensión se define en las siguientes subsecciones en cinco etapas, como se sugiere en otros trabajos que definen perfiles [11, 12, 13, 14].

### 3.1 Definición del metamodelo propuesto

El metamodelo que proponemos (ver Figura 4) minimiza al metamodelo del paquete Relations de QVT, basándose en la sintaxis concreta de dicho paquete (gramatica BNF) con algunas adaptaciones: fueron suprimidas algunas clases abstractas (como Rule, RelationDomain, Pattern, PatternDomain, etc.) de la sintaxis abstracta de Relations QVT y agregadas otras (como Query y Helper).
En nuestro metamodelo, una Transformación se compone de: **modelos** tipados (typedModel: el tipo es algún metamodelo, instancia de MOF) que participan de la transformación como sources y/o targets, **relaciones** y **querys**. Las Relations pueden incluir declaraciones de **variables**, cláusulas **when** y **where**, predicados que deben cumplirse para que se produzca la relación de Transformación y predicados que deben cumplirse al finalizar la ejecución de la transformación, respectivamente y **helpers**. En nuestro caso, un **helper**, elemento no considerado en el metamodelo de QVT, define operaciones adicionales para realizar navegación sobre los modelos que participan de la transformación. Los helpers pueden tener parámetros de entrada y pueden usar recursión. Se componen de declaraciones de variables y de expresiones que especifican dichas operaciones adicionales de consulta/navegación. La aplicación de un **helper** no produce efectos laterales.
Además las **relaciones** se definen en uno o mas **Dominios**; cada Dominio se corresponde a un modelo y puede ser sólo *chequeado* o bien, *forzado*. Cuando un dominio sólo se chequea, sus restricciones y/o expresiones booleanas serán evaluadas con respecto a los valores actuales del modelo asociado. Cuando un dominio es forzado también es chequeado, pero en este caso, el modelo asociado puede ser actualizado (es decir puede producirse la creación y /o borrado de objetos) para concretar la transformación.

En el dominio se pueden especificar **TemplateExp**, expresiones complejas que deben conformar el tipo de la variable del dominio que están especificando y, opcionalmente otras restricciones (predicados).
Una TemplateExp tiene como propósito definir *templates* de objetos y/o colecciones arbitrariamente anidados para *pattern matching* e instanciación. Contiene un número de declaraciones de variables que permiten hacer el *binding* a todos los elementos en un modelo el cual encaja con la forma estructural del tipo de nivel más alto del *template*. La Figura 5 muestra la jerarquía propuesta para las TemplateExp . Este modelo es una simplificación del propuesto en el documento QVT y completa el metamodelo propuesto. Una TemplateExp puede ser un **ObjectTemplateExp** o una **CollectionTemplateExp**. Un ObjectTemplateExp refiere a una metaclase para la cual es su template, contiene PropertyTemplate, que permiten matchear valores contra Properties de la metaclase en cuestión. El atributo value de la property template puede ser cualquier expresión del tipo correcto, incluyendo otros ObjectTemplateExp en el caso de propiedades object-valuadas, o bien CollectionTemplateExp en el caso de cualquier propiedad multi-valuada).
Por su parte, los **querys** especifican transformaciones entre instancias, al igual que las relaciones, pero con un formato más simple. La idea es que son funciones libres de efectos laterales, con parámetros formales y una OCLExpression como cuerpo.

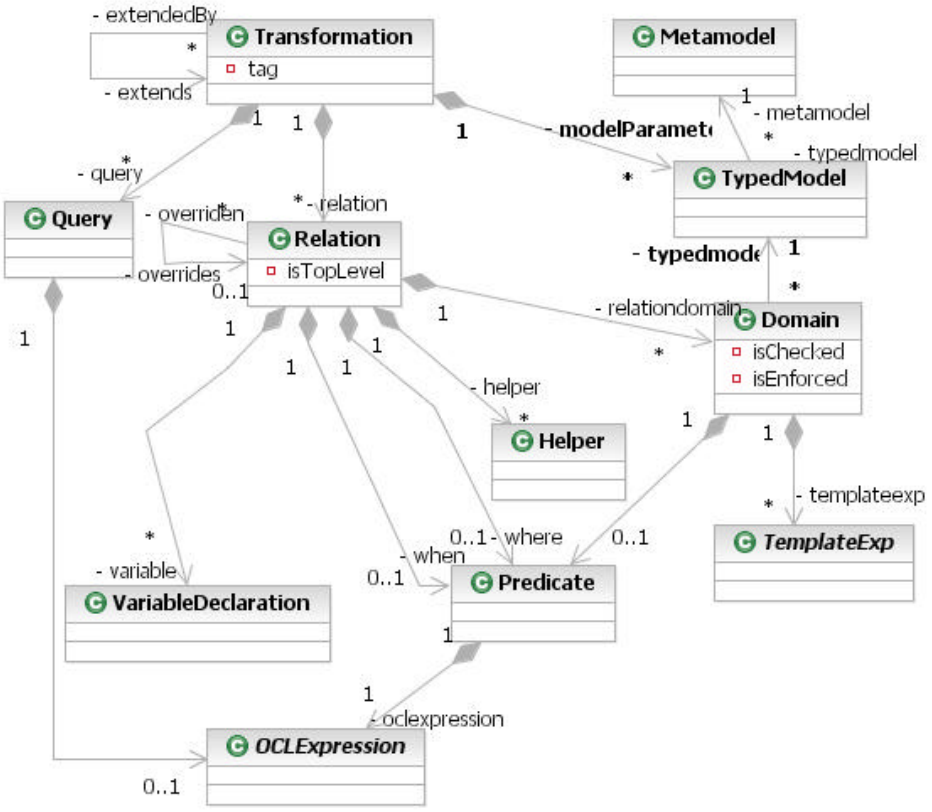


Figura 4. Metamodelo propuesto para Transformación de Modelos

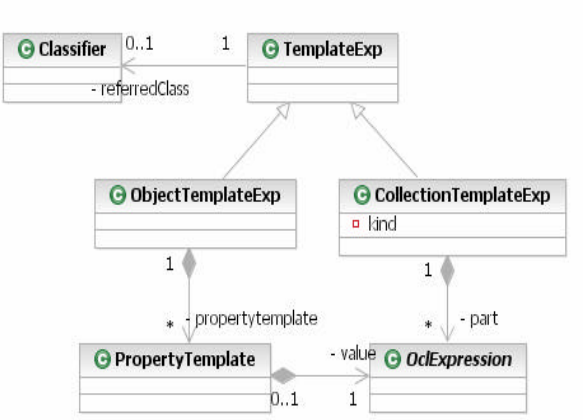


Figura 5. Metamodelo propuesto para Expresiones Template

### 3.2 Definición de los estereotipos correspondientes al metamodelo propuesto

Por cada metaclase y relaciones entre metaclases del paso anterior, definimos un estereotipo nuevo. Los estereotipos en la Infraestructura a diferencia que en UML, son simplemente clases y los llamados taggedValues de UML son Property (ownedAttribute). La Relationship entre el estereotipo y la metaclase que es su base se denomina Extension y es subclase de Association.

Notacionalmente no hay diferencia para describir estereotipos en la Infraestructura y en UML. La próxima subsección muestra a los nuevos estereotipos con sus bases.

Respecto al metamodelo para las TemplateExp presentado en la Figura 5, no creemos necesario definir nuevos estereotipos, aún cuando en el documento QVT, TemplateExp es una especialización de LiteralExp de OCL y fueron agregadas nuevas metaclases a OCL.

Luego de analizar la sintaxis abstracta del metamodelo de OCL 2.0, podemos concluir que éste incluye metaclases suficientes para cumplir con la jerarquía de TemplateExp, es decir podemos ver una clara correspondencia entre las metaclases:

1. TemplateExp con VariableDeclaration de OCL ( con la restricción de que el Classifier, referenciado por *type*, de la variable debe coincidir con algún type del metamodelo del typedModel correspondiente en la transformación);
2. La VariableDeclaration de OCL se compone de una initExp (una OCLExpression) y LiteralExp es subclase de OCLExpression, por lo tanto, para completar la especificación de TemplateExp vemos que:
3. ObjectTemplateExp se corresponde con TupleLiteralExp de OCL, que es subclase de LiteralExp y a su vez se compone de varias VariableDeclaration (las PropertyTemplate de nuestro modelo), y
4. CollectionTemplateExp se corresponde con CollectionLiteralExp de OCL, que es también subclase de LiteralExp y se compone de varias partes que tienen nombre, eventualmente tipo y una OCLExpression como item, que a su vez puede ser otra TupleLiteralExp, con sus VariableDeclaration (se mantiene la definición recursiva original)

Por lo tanto, no es necesario crear nuevos estereotipos para esta parte del metamodelo. Es suficiente aplicar un paso de *refactoring* sobre la jerarquía de TemplateExp de la Figura 5, y logramos correspondencia con las metaclases de OCL. Esta correspondencia entre metaclases de cada metamodelo, aparece marcada con el mismo número en la Figura 6.

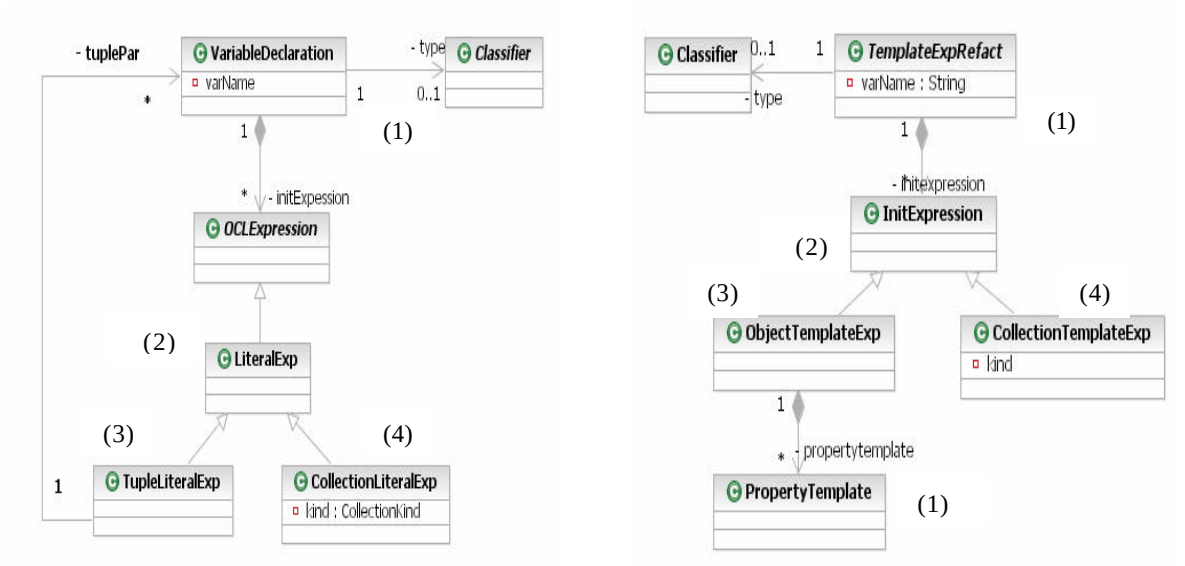


Figura 6. Correspondencia entre el metamodelo TemplateExp Refactoreado (derecha) y OCL (izquierda)

### 3.3 Identificación de las bases para los estereotipos propuestos

En esta subsección se identifican las metaclases de la Infraestructura que pueden ser extendidas para representar el metamodelo propuesto en 3.1.

Según este metamodelo, una *transformation* podria definirse:

- como una relación (*Relationship*) entre los modelos. Esta opción predispone a definir sentido de navegabilidad (o bien source y target), multiplicidad para la transformación (atributos para las *relationships*), con lo que agregamos elementos que nos alejan de una especificación declarativa.
- como una operación definida en el contexto de uno de los modelos participantes (una operación se debe especificar en un determinado contexto), que tenga como parámetros a los otros modelos. Esta opción nos lleva a definir direcciones para la transformación, por lo que se debe elegir a priori sobre que modelo definir la transformación. Similarmente a la primer opción, resulta más imperativa.
- O bien como una extensión de Class (de la Infraestructura) compuesta por los modelos que transforma y por las *relations* y *querys* definidos entre ellos. Así, todos los modelos son referenciados sin indicar a cual se le aplica la transformación ni en que dirección se realiza.

Elegimos este último caso por su estilo declarativo ya que se ajusta más a nuestra propuesta. En tal sentido, una *Relation* también puede representarse con base en Class, compuesta por los *Domains* que intervienen en ella y las condiciones y *helpers* que la restringen. *Domain*, que agrega características a los modelos, también tiene base en Class.

El estereotipo Query, puede tener base en Operation. Su atributo isQuery ya tiene representación en Operation; isQuery es un metaAtributo booleano de Operation.

En general, los modelos se pueden representar mediante paquetes. Por lo tanto, TypedModel y Metamodel se definen con base en Package ya que especifican respectivamente a los modelos que participan en la transformación y a los metamodelos correspondientes de los cuales son instancias cada uno de ellos.

Por otro lado, la relación *extends* entre transformaciones como la de *overrides* entre *relations* y la de *Composite* (entre *transformation* y *relation* y entre *relation* y *domain*) pueden tener base en Association, ya que relacionan instancias de Class, que son Classifiers

Para la relación entre Transformation y Query (que tiene base en *Operation*), el dueño de la *operation* <<Query>> debe ser una clase <<Transformation>>. La Figura 7a) muestra los estereotipos nombrados con sus bases (metaclases que estereotipan). En el caso de las relaciones entre TypedModel y Transformation (modelParameters), entre TypedModel y Metamodel (Type) y entre Domain y TypedModel (DomainModel), como los elementos relacionados son Paquetes (no Classifiers) no puede elegirse Association como base ya que ésta se define entre Classifiers. Por lo tanto se toma como base DirectedRelationship, como puede verse en la Figura 7b).

Nota: en la Infraestructura no existe Dependency, ni otra Relationship que conecte elementos distintos a Classifier.

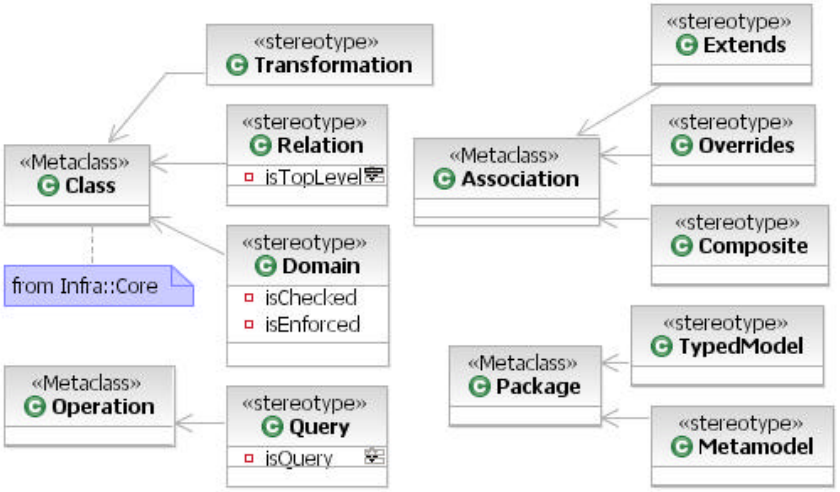


Figura 7a) Estereotipos con base en Class, Association, Package y Operation

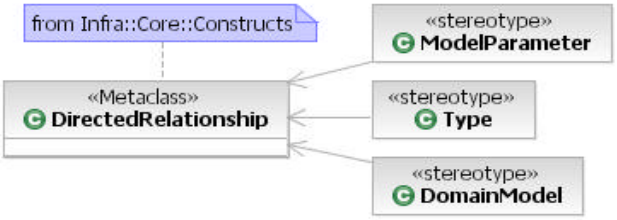


Figura 7b) Estereotipos con base en DirectedRelationship

Finalmente, resta definir bases para las expresiones (*helpers* y *predicates*) que restringen una *relation* de nuestro metamodelo. En los lenguajes de modelado basados en MOF, en muchos de los casos que se utiliza el término *expression*, puede usarse una expresión escrita en OCL. El significado de la evaluación de la expresión OCL depende de su contexto y del objeto al que referencia dentro del modelo. Dado que en la sintaxis abstracta de OCL una OCLExpression se define recursivamente, necesitamos una nueva metaclase para representar la raíz del árbol sintáctico abstracto que representa una OCLExpression. Esta metaclase es llamada ExpressionInOCL. En el metamodelo OCL, una ExpresionInOCL aparece como subclase de Expression, pero no para todas las Expression se permite definir una especificación (*body*) en un determinado lenguaje. Por esto creemos que ExpressionInOCL debería ser subclase de OpaqueExp de la Infraestructura, ya que ésta si tiene un atributo *body* (que sería la OCLExpression) y un atributo *language* el cual debería tener para este caso el valor "OCL". Un estereotipo ya existente para una ExpressionInOCL es <<definition>>, que restringe a la ExpressionInOCL indicando que debe estar definida en el contexto de un Classifier y puede definir operaciones adicionales y contener expresiones Let. O sea, los helpers pueden verse como una expresión <<definiton>> de OCL, es decir una subclase del estereotipo <<definition>>, con base en ExpressionInOCL (ver Figura 8) que restringe elementos de los modelos de la *relation* y está definido en el contexto de una clase <<relation>> de nuestro metamodelo (lo cual es correcto, por tener <<relation>> base en Class).

A su vez un **predicate** puede verse como un invariante de la *relation*, por lo tanto lo definimos como subclase del estereotipo <<invariant>> (ver Figura 8) de ExpressionInOCL (una ExpressionInOCL <<invariant>> es una Expresión cuyo cuerpo es una OCLExpression de tipo Boolean, que restringe a un classifier y no puede contener el postfijo @pre). En nuestro metamodelo, un **predicate** restringe sólo una clase <<relation>> y puede referir y anidar a otras *relations* salvo en el caso que sea topLevel. A su vez, **when** (precondiciones de la *relation*) y **where** (post condiciones de la *relation*), son esencialmente predicados etiquetados. La gramática BNF no presenta diferencias sintácticas entre ellos, por lo que se definen como atributos. Ambos pueden referir a *helpers* de la *relation* y a otras *relations*. Cuando un predicado tiene el atributo isWhen en true, se están especificando restricciones previas a la ejecución de la *relation*.

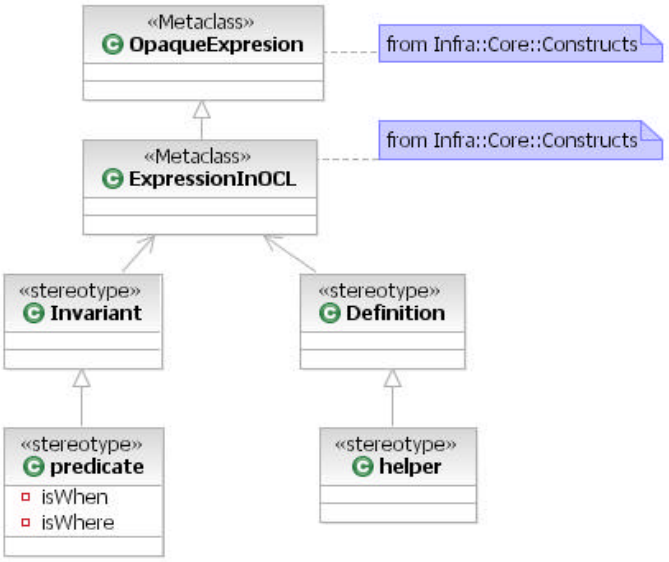


Figura 8 Estereotipos **helper** y **predicate**, con base en ExpressionInOCL

Cuando un predicado tiene el atributo isWhere en true, se están especificando condiciones que deben cumplirse luego de la ejecución de la *relation.*

### 3.4 Definición de atributos para los estereotipos.

Por cada atributo presente en el metamodelo propuesto en 3.1, se define un atributo en el estereotipo correspondiente. Las Figuras 7a), 7b) y 8 muestran los estereotipos con sus atributos, por ejemplo *IsTopLevel* de tipo Boolean en <<Relation>>; *isChecked*, *isEnforced* de tipo Boolean, en <<Domain>>; *isWhen* e *isWhere*, de tipo Bolean en <<predicate>>, etc.
En la sección 4 presentamos ejemplos de uso de los estereotipos definidos.

### 3.5 Definición en forma completa de la extensión propuesta

La extensión propuesta constituye una instanciación de la clase Profile dentro del paquete Profile de la Infraestructura 2.0. Este paquete se relaciona con el paquete Core y especifica entre otras, a las metaclases Profile, Stereotype y Extension.
Presentamos aquí una definición formal de algunos (no incluimos a todos por limitaciones de espacio) de los nuevos estereotipos mencionados previamente, indicando a que metaclase estereotipan (cual es su base), sus atributos y algunas de las restricciones, expresadas en OCL, que deben cumplirse al utilizarlos.

- STEREOTYPE **Transformation**

*Base*: Class
*Constraints*:
1. Una *transformation* debe contener relaciones o querys.

```
self.relations -> notEmpty() or  self.querys -> notEmpty()
```

2. Una *transformation* debe tener al menos dos modelos como parámetros (un input y un output)

```
self.modelParameter -> size() > 1
```

- STEREOTYPE **Relation**

*Base*: Class
*OwnedAttributes:*
Name: isTopLevel, Type: Boolean
*Constraints*:
1. Una *relation* debe definirse en al menos un dominio.

```
self.domains -> notEmpty()
```

2. Una *relation* debe ser parte de una *Transformation*

```
self.owningPackage.ownedMember-> exists ( a:Association | a.estereotype=<<composite>> and
a.memberEnd-> exists (e| e.isComposite=true and e.type.stereotype= <<Transformation>> and
e.opposite = self ) )
```

3. una *relation* topLevel no puede estar anidada en otra.

```
(self.owningPackage.ownedMember-> select ( r:Class | r. stereotype= <<relation>>)) -
>forAll (r| self.owningPackage.ownedMember-> exists ( a:Association | a.memberEnd->
exists (e| e.isComposite=true and e.type =r and e.opposite.stereotype =<<relation>>
implies e.opposite.isTopLevel=false ) ) )
```

- STEREOTYPE **Domain**

*Base*: Class
*ownedAttributes:*
Name: isEnforced, Type: Boolean
Name: isChecked, Type: Boolean
*Constraints*:
1. Un Domain debe hacer referencia a un TypedModel (existe una DirectedRelationship <<domainModel>> entre Domain y TypedModel)

```
self.owningPackage.ownedMember-> exists ( d:DirectedRelationship | d.estereotype=<<
domainModel>> and d.source=self and d.target.type.stereotype= <<TypedModel>> )
```

- STEREOTYPE **typedModel**

*Base*: Package
*Constraints*:
1.un modelo tipado debe ser instancia de un metamodelo, o sea debe ser el source de una DirectedRelationship con stereotipo <<type>> y target un paquete <<metamodel>>

```
self.owningPackage.ownedMember-> exists ( d:DirectedRelationship | d.estereotype=<<type>>
and d.source=self and d.target.type.stereotype= <<metamodel>> )
```

2.un modelo tipado solo debe contener instancias del metamodelo con el que se relaciona.

```
self.ownedMember-> forAll (e| self.metamodel -> includes (e.type))
```

donde la operación *metamodel* retorna el paquete que contiene al metamodelo del modelo tipado (*self*):
*metamodel:* Package -> Package

```
metamodel = self.owningPackage.ownedMember ->select (d:DirectedRelationship|
d.stereotype=<<type>> and d.source=self) ->collect (d:DirectedRelationship| d.target)
)
```

- STEREOTYPE **type**

*Base*: DirectedRelationship
*Constraints*:
1.se establece entre TypedModel y un único metamodelo

```
self.source-> forAll (m| m.stereotype= <<TypedModel>>)and self.target.stereotype=
<<metamodel>>
```

- STEREOTYPE **predicate,** subclase del stereotype *<<invariant>>*

*Base*: ExpresionInOCL
*ownedAttributes:*
Name: isWhen, Type: Boolean
Name: isWhere, Type: Boolean
*Constraints*:
1. un *predicate* debe restringir una *relation*

```
self.constraint.constrainedElement.stereotype=<<relation>>
```

2. un predicate no puede etiquetarse como when y where al mismo tiempo

```
Self.isWhen=true implies Self.isWhere=false and Self.isWhere= true implies
Self.isWhen=false
```

- STEREOTYPE **helper,** subclase del stereotype <<*definition*>>

*Base*: ExpresionInOCL
*Constraints*:
1. un helper se define (tiene contexto) en una <<relation>>.

```
self.constraint.context.stereotype= <<relation>>
```

2. un helper debe restringir a elementos de los modelos (a través de los dominios) que participan de la relación de transformación.

```
self.owningPackage.ownedMember-> exists ( a:Association | a.estereotype=<< composite>>
and a.memberEnd->exists (e| e.isComposite=true and e.type=self.constraint.context and
e.type.stereotype =<<Relation >> and e.opposite. stereotype =<<Domain>> and
typedModel(e.opposite).metamodel. ownedMember ->includes
self.constraint.constrainedElement.type) ) )
```

donde la operación *typedModel* retorna el paquete que contiene el modelo correspondiente al domino dado como parámetro (dominio es una Clase con stereotype <<domain>>)

*typedModel :* Class -> Package

```
typedModel (dom)= dom.owningPackage.ownedMember ->select (d:DirectedRelationship|
d.stereotype=<<DomainModel>> and d.source= dom)->collect(d: DirectedRelationship|
d.target)
```

## 4 APLICACIÓN DEL PERFIL EN EL EJEMPLO

En esta sección mostramos el metamodelado, utilizando el perfil propuesto, para el ejemplo introducido en la sección 2. La Figura 9 muestra este ejemplo utilizando la notación gráfica y estereotipos propuestos. Este modelo presenta una instanciación de la transformación que define patrones de transformación sobre instancias genéricas de las metaclases (clases y tablas, por ejemplo). Podemos observar que una nueva instanciación de esta transformación, será sobre elementos particulares de modelos concretos a nivel M1 de la arquitectura 4 capas. Las cláusulas when y where del ejemplo son ahora predicados (constraints de OCL) que restringen a la relación UML2Rel, mientras que el patrón de transformación dentro del dominio Uml está expresado también en OCL mediante una VariableDeclaration que, como fue expuesto en la subsección 3.2, se corresponde con la metaclase TemplateExp del metamodelo inicial.

La especificación del dominio Rel (que no aparece en el modelo por limitaciones de espacio) es similar a la del dominio Uml.

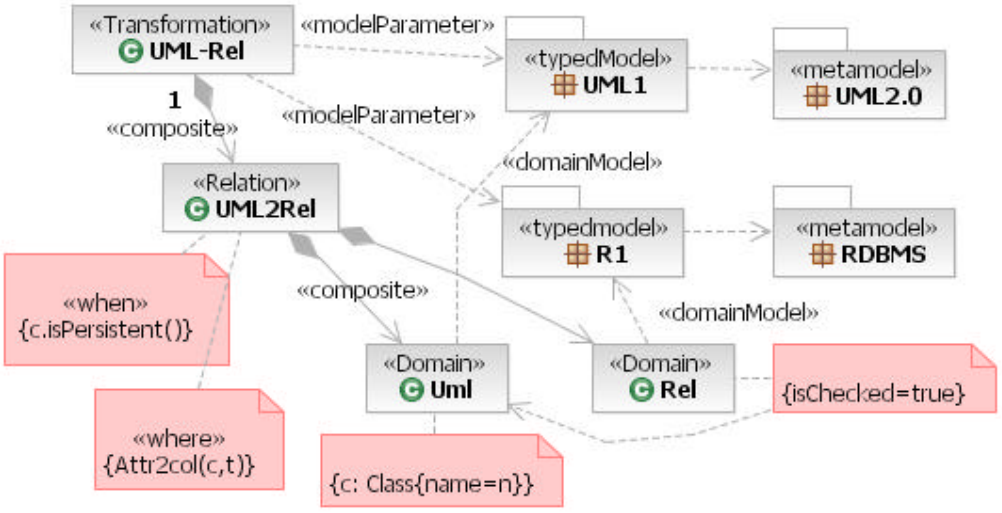


Figura 9. Metamodelado del Ejemplo

## 5. CONCLUSIONES Y TRABAJO FUTURO

La transformación entre modelos constituye el motor del MDD; de esta manera los modelos, dentro del proceso de desarrollo de software, pasan de ser entidades meramente contemplativas a ser entidades productivas. Las transformaciones entre modelos requieren de lenguajes específicos para su definición. Estos lenguajes deben tener base formal, por ejemplo tener un metamodelo que los sustente, y permitir un tratamiento automatizado.

En este trabajo hemos propuesto un lenguaje puramente declarativo para expresar transformaciones entre modelos que inspira su metamodelo inicial en el lenguaje QVT de OMG. Nuestra propuesta está basada en especificaciones ya existentes en OMG: por un lado extendemos, mediante la definición de estereotipos, la Infraestructura 2.0, especificación más abstracta que UML e independiente de él; por otro lado utilizamos al lenguaje OCL para expresar

patrones de transformación, ya que luego de analizar estos conceptos concluimos que no es necesario crear nuevos elementos para modelarlos. El lenguaje propuesto pretende ser el mínimo para poder expresar relaciones y *querys* de transformación entre modelos.

Como líneas de trabajo futuro, en lo inmediato implementaremos este lenguaje integrándolo a la herramienta Case [18] que viene desarrollando nuestro grupo de investigación. Esta herramienta orientada al modelado formal, incluye la especificación de refinamientos de modelos que puede verse como un caso particular de transformación de modelos. Sobre este tópico hemos publicado, entre otros, los artículos: [15, 19, 20] que aportan una base formal a la herramienta. Otros trabajos relacionados como: [16, 17], nos han servido de inspiración.

Nos proponemos además, incorporar al lenguaje elementos que faciliten la *traceability* entre modelos, el chequeo de consistencia (cuando la relación de transformación es forzada), el testing integrado a la transformación y la composición (encadenamiento) de transformaciones.

## REFERENCIAS


[1] MDA Guide, v1.0.1, omg/03-06-01, June 2003. http://www.omg.org.

[2] OMG (Object Management Group) http://www.omg.org

[3] Meta Object Facility (MOF) 2.0. OMG Adopted Specification. October, 2003. http://www.omg.org.

[4] MOF 2.0 Query/View/Transformations - OMG Adopted Specification. March 2005. http://www.omg.org.

[5] OMG. The Object Constraint Language Specification – Version 2.0, for UML 2.0, revised by the OMG, http://www.omg.org, April 2004.

[6] The Unified Modeling Language Superstructure version 2.0.,OMG Final Adopted Specification. April 2004. http://www.omg.org.

[7] The Unified Modeling Language Infrastructure version 2.0, OMG Final Adopted Specification. March 2005. http://www.omg.org

[8] Jouault F., Kurtev I. Transforming Models with ATL Workshop in Model Transformation in Practice at the MODELS 2005 Conference. Montego Bay, Jamaica, Oct 3, 2005

[9] Akehurst D., Howells W., McDonald-Maier K. Kent Model Transformation Language. Workshop in Model Transformation in Practice at the MODELS 2005 Conference. Montego Bay, Jamaica, Oct 3, 2005

[10] Lawley M., Steel J. Practical Declarative Model Transformation with TefKat. Workshop in Model Transformation in Practice at the MODELS 2005 Conference. Montego Bay, Jamaica, Oct 3, 2005

[11] Blankenhorn Kai & Jeckle Mario, "A UML Profile for GUI Layout", NODe 2004, LNCS 3263, pp.110-121, 2004 Springer-Verlag Berlin Heidelberg 2004.

[12] Fuentes,L.,Troya, J.M., Vallecillo, A.. "Using UML Profiles for Documenting Web-based Application Frameworks". Annals of Software Engineering, Vol. 13, pp.249-264, June 2002.

[13] Grassi,V., Mirandola, R., and Sabetta,A. "A UML Profile to Model Mobile Systems". Seventh International Conference on UML Modeling Languages and Applications, UML 2004. Lisboa, Portugal.

[14] Ziadi,T. Héloüet,L., and Jézéquel,J.M. "Towards a UML Profile for Software Product Lines", PFE 2003, LNCS 3014, pp. 129–139, 2004 Springer-Verlag Berlin Heidelberg 2004.

[15] Giandini, R., Pons, C., Pérez,G. Use Case Refinements in the Object Oriented Software Development Process. Proceedings of CLEI 2002, ISBN 9974-7704-1-6,Uruguay. 2002.

[16] Hnatkowska B., Huzar Z., Tuzinkiewicz L. On Understanding of Refinement Relationship. Workshop in Consistency Problems in UML-based SoftwareDevelopment III – Understanding and Usage of Dependency Relationships at the 7thInternational Conference on the UML. Lisbon, Portugal, October 11, 2004.

[17] Liu Z., Jifeng H., Li X., Chen Y. Consistency and Refinement of UML Models. .Workshop in Consistency Problems in UML-based Software Development III at the 7th International Conference on the UML. Lisbon, Portugal, Oct 11, 2004

[18] Pons C., Giandini R., Pérez G., Pesce P., Becker V., Longinotti J., Cengia J.,Correa N. and Labaronnie P. Precise Assistant for the Modeling Process in an Environment with Refinement Orientation. In "UML Modeling Languages and Applications: UML 2004 Satellite Activities, Revised Selected Papers". Lecture
Notes in Computer Science number 3297. Springer, Oct., 2004. The tool can be downloaded from http://sol.info.unlp.edu.ar/eclipse

[19] Pons, C., Kutsche, R. Traceability Across Refinement Steps in UML Modeling. 3rd Workshop in Software Model Engineering WiSME at the 7th UML Conference. October 11, 2004. Lisbon, Portugal .

[20] Pons, C., Pérez,G., Giandini, R., Kutsche, R. Understanding Refinement and Specialization in the UM L. and 2nd International Workshop on Managing SPEcialization/Generalization Hierarchies (MASPEGHI). In IEEE ASE 2003, Canada.