# Soporte de Transformaciones de Refinamiento con QVT Relacional

Thomas Goldschmidt

Centro de Investigación de Tecnología de Información FZI

Haid-und-Neu-Str. 10-14

76131 Karlsruhe

Alemania

goldschmidt@fzi.de

Guido Wachsmuth

Humboldt-Universität zu Berlin

Unter den Linden 6

10099 Berlin

Alemania

guwac@gk-metrik.de

Contenido

[Soporte de Transformaciones de Refinamiento con QVT Relacional 1](#_Toc392167184)

[Abstracto 2](#_Toc392167185)

[1. Introducción 2](#_Toc392167186)

[Las transformaciones de modelos. 2](#_Toc392167187)

[Refinamientos Modelo. 2](#_Toc392167188)

[Estructura del documento. 3](#_Toc392167189)

[2. Trabajos Relacionados 3](#_Toc392167190)

[3. Patrones genéricos para las reglas de copia 3](#_Toc392167191)

[Ejemplo: modelos de red de Petri 4](#_Toc392167192)

[Figura 3.1: Ejemplo meta-modelo para las redes de Petri 4](#_Toc392167193)

[Copia de instancias 4](#_Toc392167194)

[Listado 3.1 Transformaciones de identificación y copia de la Meta-clase Net 5](#_Toc392167195)

[Listado 3.2 Transformaciones de copia e identificación de las meta-clases Place y CPlace 6](#_Toc392167196)

[Copia de ranuras y enlaces 6](#_Toc392167197)

[Listado 3.3 Relación de copia para el atributo Place.tokens. 6](#_Toc392167198)

[Listado 3.4 Relaciones de copia para las referencias Place.source y Place.sink. 7](#_Toc392167199)

[4.Generación de transformaciones de copiado 8](#_Toc392167200)

[4.1 Estructura general 8](#_Toc392167201)

[Listado 4.1 Estructura general de la transformación de orden superior emf2copyQVT. 9](#_Toc392167202)

[4.2 Generación de relaciones de copia de Metaclases 10](#_Toc392167203)

[Listado 4.2 generación de una relación de copia de una meta-clase. 11](#_Toc392167204)

[Listado 4.3 Generación de dominios para las relaciones. 11](#_Toc392167205)

[Listado 4.4 Generación de negación de llamadas a identificadores de subclases. 12](#_Toc392167206)

[4.3. Generación de relaciones de identificación de meta-clases 12](#_Toc392167207)

[**Listado 4.5 Generación de relaciones de identificación de mata-clases** 13](#_Toc392167208)

[4.4. Generación de relaciones de copia para atributos y referencias 13](#_Toc392167209)

[Listado 4.6 Generación de relaciones copia de atributos y referencias. 14](#_Toc392167210)

[**Listado 4.7 Generación de plantillas de atributos.** 15](#_Toc392167211)

[5. Redacción de transformaciones de refinamiento 15](#_Toc392167212)

[Transformación manualmente extendida 15](#_Toc392167213)

[Listado 5.1 Uso de extensiones manuales para el refinamiento de la transformación de copia 17](#_Toc392167214)

[Tejido automático 17](#_Toc392167215)

[Figura 5.1 Tejido automático de reglas de excepción 17](#_Toc392167216)

[6. Observaciones finales 18](#_Toc392167217)

[Referencias 18](#_Toc392167218)

## Abstracto

Las transformaciones de modelos son un concepto central en la Ingeniería Dirigida por Modelos. Las transformaciones de modelos se definen en los lenguajes de transformación de modelos. Este documento aborda QVT Relacional, un lenguaje declarativo de transformación de modelos de alto nivel estandarizado por el Object Management Group. QVT Relacional carece de soporte para reglas de copia predeterminada. Por lo tanto, los desarrolladores de transformación necesitan definir reglas de copia explícitamente. En especial se trata de una tremenda tarea para las transformaciones de refinamiento que copian grandes partes de un modelo. En este trabajo, proponemos pautas genéricas para reglas de copia en Relaciones QVT. En base a estos patrones, proporcionamos una transformación de orden superior para generar reglas de copia para un determinado meta-modelo. Finalmente, se exploran varias formas de obtener una transformación de refinamiento a partir de una transformación de copia generada.

# 1. Introducción

## Las transformaciones de modelos.

En ingeniería dirigida por modelos, las transformaciones de modelos son un concepto central. Se utilizan para traducir los modelos de origen a los modelos destino, por ejemplo, modelos independientes de plataforma en otras más específicas de la plataforma. Además, el modelo de transformaciones se pueden instrumentar para traducir los modelos en el texto, por ejemplo, en una lenguaje ejecutable como Java [1]. Con QVT) [2], grupo OMG proporciona un estándar para las transformaciones de modelo a modelo. En realidad, QVT define tres lenguajes de transformación de modelos: QVT Relacional y QVT Core son lenguajes declarativos en dos niveles diferentes de abstracción. El QVT de Asignaciones de Operaciones de Mapeado es un lenguaje imperativo. En este trabajo, nos centramos en QVT Relacional, el lenguaje declarativo de alto nivel.

Se pueden distinguir dos tipos de transformaciones de modelo a modelo

Transformaciones exógenas: traducen modelos expresados en un determinado idioma de origen a modelos expresados en un idioma de destino diferente. Para las transformaciones endógenas, los modelos de origen y destino se expresan en el mismo idioma. El modelo de destino puede ser derivado actualizando el modelo de entrada o mediante la creación de un modelo completamente nuevo.

## Refinamientos Modelo.

A menudo, el modelo de destino de una transformación es simplemente una refinación del modelo de origen, es decir, la transformación conserva gran parte de la fuente. Los refinamientos pueden ser endógenos, por ejemplo, una optimización, o exógenos, por ejemplo, la migración a una nueva versión. A pesar que las actualizaciones son particularmente útiles para describir refinamientos en una forma compacta, hay varias razones para preferir la creación de un nuevo modelo: En primer lugar, se preserva el modelo de origen. En segundo lugar, las trazas entre la fuente y el modelo de destino se convierten en explícitas. Por último, es necesario describir de forma explícita, las transformaciones exógenas ya que las actualizaciones están restringidos a transformaciones endógenas.

Una transformación que realiza un refinamiento tiene que copiar grandes partes de un modelo. Como que QVT Relacional no admite copias por defecto, tiene que especificarse explícitamente las transformaciones de copia en una definición de una transformación de refinamiento. En este trabajo, investigamos transformaciones de copia en QVT Relacional. En primer lugar, proponer pautas genéricas para las reglas de copia. En segundo lugar, ofrecemos una forma de generar la definición de una transformación de copia para un meta-modelo dado. La generación se especifica como una transformación de orden superior. Finalmente, se exploran varias formas de obtener una definición del refinamiento de una transformación de copia generada.

## Estructura del documento.

En la Sección 2, se discute el trabajo relacionado. En la sección 3, se examinan patrones genéricos en reglas de copia. En la Sección 4, se presenta una transformación de orden superior para la generación de transformaciones de copia. En la Sección 5, se discute la derivación de transformaciones de refinamiento a partir de transformaciones de copia generadas. El papel concluye en la Sección 6.

# 2. Trabajos Relacionados

En contraste con las relaciones QVT, las Asignaciones operacionales QVT proporcionan una operación de copia profunda que se puede utilizar dentro de las reglas de asignación imperativas. Esta operación crea una copia de una subestructura dada. Sin embargo, no es posible especificar excepciones para elementos que se producen dentro de la subestructura que se copia. Por lo tanto, este enfoque es sólo parcialmente útil para una transformación refinamiento.

El Lenguaje de Transformación Atlas (ATL) [3] es compatible con un modo especial que permite al programador de transformaciones especificar que una transformación se debe ejecutar como transformación refinamiento. Esto significa que todos los elementos se copian de forma predeterminada, mientras que aquellos elementos que coinciden con las reglas de transformación dentro de la actual transformación, no. Esos elementos sin embargo son tratados por reglas de transformación dadas

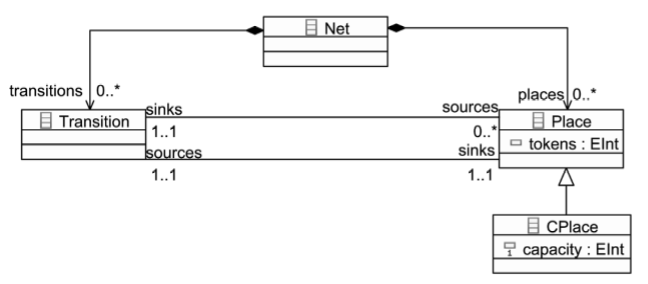
Otro enfoque es la transformación del modelo de Triple Graph Grammar (TGG) [5]

Dentro de este enfoque, el mecanismo de ejecución estándar se describe como transformaciones de actualización donde se aplican reglas de transformación, siempre y cuando no haya reglas a las que correspondan los patrones en el modelo. Así, TGGS se puede utilizar de forma natural para actualizaciones endógenas, y no necesitan un apoyo especial para las reglas de copia

# 3. Patrones genéricos para las reglas de copia

En esta sección, vamos a discutir las pautas genéricas para reglas de copia. Ilustramos estos patrones con reglas de copia para el modelo de redes de Petri.

## Ejemplo: modelos de red de Petri

****

### Figura 3.1: Ejemplo meta-modelo para las redes de Petri

La Figura 3.1 muestra el meta-modelo para modelos de red de Petri: Una red está formada por lugares y transiciones. Una transición tiene varios lugares como fuentes y como sumideros. De la misma manera, un lugar tiene varias transiciones como fuentes y como sumideros. Por otra parte, un lugar se marcada con un número opcional de fichas. Un CPlace está limitado, además, por una capacidad máxima.

La copia de las redes de Petri revela varios aspectos:

(i) las redes, los lugares y las transiciones se deben copiar.

(ii) Para los lugares, tenemos que preservar su tipo (Place vs CPlace).

(iii) Para los lugares, tenemos que preservar el número de fichas de marcaje. Lo mismo para su capacidad máxima.

(iv) Los vínculos entre las redes, los lugares y las transiciones se deben copiar.

## Copia de instancias

Las dos primeras cuestiones se refieren a las copias de las instancias MetaClass. Para una meta-clase no abstracta, definimos una relación de copia para especificar copias de sus instancias. En el modelo fuente esta relación simplemente coincide con una instancia de la meta-clase. En el modelo de destino, la relación impone una instancia correspondiente de la misma meta-clase. La declaración de las relaciones superiores (top) aseguran que la relación se aplica a todas las instancias del meta-modelo en el modelo fuente.El Listado 3.1 muestra dicha relación de copia CopyNet para la meta-clase Net

*top relation copyNet {*

*checkonly domain orig netO: Net {};*

*enforce domain copy netC: Net {};*

*where { copiedNet(netO, netC); }*

*}*

*relation copiedNet {*

*checkonly domain orig netO: Net {};*

*checkonly domain copy netC: Net {};*

*}*

### Listado 3.1 Transformaciones de identificación y copia de la Meta-clase Net

En la cláusula where de esta relación, se llama a otra relación copiedNet con el net original y su copia como argumentos. Esta relación de identificación se muestra así también en el listado 3.1. **Las relaciones de identificación son relaciones no superiores (top) que simplemente responden a sus argumentos en el modelo fuente y de destino. Utilizamos este tipo de relaciones para indicar qué elementos han sido ya copiados. Una vez que se copia un elemento, llamamos a la relacion de identificación correspondiente desde la cláusula de where.**

*top relation copyPlace {*

*checkonly domain orig placeO: Place {};*

*enforce domain copy placeC: Place {}*

*when { not copiedCPlace(placeO, placeC); }*

*where { copiedPlace(placeO, placeC); }*

*}*

*top relation copyCPlace {*

*checkonly domain orig placeO: CPlace {};*

*enforce domain copy placeC: CPlace {};*

*where { copiedCPlace(placeO, placeC); }*

*}*

*relation copiedPlace {*

*checkonly domain orig placeO: Place {};*

*checkonly domain copy placeC: Place {};*

*}*

*relation copiedCPlace {*

*checkonly domain orig placeO: Place {};*

*checkonly domain copy placeC: Place {};*

*where { copiedPlace(placeO, placeC); }*

*}*

### Listado 3.2 Transformaciones de copia e identificación de las meta-clases Place y CPlace

El listado 3.2 muestra las relaciones de copia y marcado de Place y CPlace. Ilustra como **las relaciones de identificación de las subclases llaman a las relaciones de identificación de sus superclases**, es decir, copiedCPlace llama copiedPlace **desde su cláusula where. De esta manera podemos indicar que una instancia ya ha sido copiada por una regla más específica. La cláusula when de copyPlace lo asegura**. La relación sólo se ejecuta si nunca se llama copiedCPlace para un par dado de lugares en el modelo origen y destino. **Para poner en funcionamiento este patrón para cualquier jerarquía de clase dada, necesitamos relaciones de identificación para meta-clases abstractas también.**

## Copia de ranuras y enlaces

Los dos últimos números se refieren a las copias de las ranuras y los enlaces. Ambas se pueden realizar de una manera similar. **Para copiar ranuras que instancian un atributo definido por una meta-clase se define una relación superior (top) para ese atributo**. En el modelo origen la relación coincide con una instancia de la meta-clase así como el valor almacenado en la ranura correspondiente al atributo. En el modelo destino le corresponde otra instancia de la meta-clase. El valor original es copiado en la ranura correspondiente de la instancia de destino**. La cláusula *when* de la relación debe asegurar que la instancia de destino es una copia de la del origen llamando una relación de identificación**. El listado 3.3 muestra la relación de copia de Place.tokens. Desde las llamadas a las relaciones de identificación *copiedCPlace* esta relación copiará las ranuras de las fichas para las instancias de *CPlace* también.

*top relation copyPlace\_token {*

*tokenO: Integer;*

*checkonly domain orig placeO: Place { tokens = tokenO };*

*enforce domain copy placeC: Place { tokens = tokenO };*

*when { copiedPlace(placeO, placeC); }*

*}*

### Listado 3.3 Relación de copia para el atributo Place.tokens.

Para copiar enlaces instanciando una referencia entre meta-clases, el patrón es algo similar. Definimos una relación superior para las instancias correspondientes a las referencias en el modelo origen y destino. En ambos modelos la relación coincide con dos instancias conectadas por un enlace. **La cláusula when asegura que instancias del modelo destino son copias de las instancias del modelo origen**.

El listado 3.4 muestra las relaciones de copia para Place.sources y Place.sinks

*top relation copyPlace\_sources {*

*checkonly domain orig placeO: Place {*

*sources = transO: Transition {}*

*};*

*enforce domain copy placeC: Place {*

*sources = transC: Transition {}*

*};*

*when {*

*copiedPlace(placeO, placeC);*

*copiedTransition(transO, transC);*

*}*

*}*

top relation copyPlace\_sinks {

checkonly domain orig placeO: Place {

sinks = transO: Transition {}

};

enforce domain copy placeC: Place {

sinks = transC: Transition {}

};

when {

copiedPlace(placeO, placeC);

copiedTransition(transO, transC);

}

}

### Listado 3.4 Relaciones de copia para las referencias Place.source y Place.sink.

**Para un par de referencias bidireccionales, sólo se necesita una relación de copia para una de las referencias**.

# 4.Generación de transformaciones de copiado

Los patrones presentados en la Sección 3 se pueden utilizar para especificar una transformación de copia en QVT Relacional para meta-modelos ECore arbitrarios. Obviamente estos patrones son tan genéricos que podemos generar una transformación de copia directa y automáticamente desde un meta-modelo genérico. En esta sección investigamos transformaciones de orden superior para éste propósito. Esta transformación de alto nivel está escrita en QVT Relacional ella misma y captura los patrones expuestos en las secciones anteriores. Se basa en el meta-meta-modelo Ecore, la librería estándar OCL (para la negación) y en el meta-modelo de las relaciones QVT. Después de ejecutar estas transformaciones de orden superior, un modelo completo de la transformación de copia se hace disponible. Esto puede también entonces ser usado directamente en su sintaxis abstracta, o una simple pretty-printer puede ser utilizada para imprimirlo en su sintaxis concreta de forma textual.

## 4.1 Estructura general

La estructura general de la transformación de orden superior se muestra en el listado 4.1. Básicamente la transformación funciona de forma análoga a los patrones de la sección 3. Para cada paquete del meta-modelo se genera una transformación de copia. (Package2Transformation ). El nombre de esta transformación se deriva del nombre del paquete. La transformación de copia declara dos dominios el origen y el destino, ambos escritos por el paquete del meta-modelo

transformation Ecore2copyQVT ( mm: ecore, oclstdlib:ecore, qvt: QVTRelation) {

top relation Package2Transformation {

n:String;

checkonly domain mm ePackage: ecore::EPackage { name = n };

enforce domain qvt t: QVTRelation::RelationalTransformation {

name = 'Copy' + n,

modelParameter = sourceMM: QVTBase::TypedModel {

name = 'source',

usedPackage = uPackage: ecore::EPackage{}

},

modelParameter = targetMM: QVTBase::TypedModel {

name = 'target',

usedPackage = uPackage: ecore::EPackage{}

}

};

when { ePackage.eContainer().oclIsUndefined(); }

where {

uPackage = ePackage;

MarkTypedModel(sourceMM, targetMM);

MarkTransformation(t);

}

}

relation MarkTypedModel { ... }

relation MarkTransformation { ... }

top relation Class2CopyRelation { ... }

top relation SubClass2MarkerCallInWhen { ... }

top relation Class2MarkerRelation { ... }

top relation Attribute2Relation { ... }

top relation Reference2Relation{ ... }

top relation MarkBooleanType { ... }

relation Class2Domain { ... }

relation Attribute2Template { ... }

relation Reference2Template { ... }

relation Class2MarkerCall { ... }

relation Class2MarkerCallInPattern { ... }

}

### Listado 4.1 Estructura general de la transformación de orden superior emf2copyQVT.

El resto de las relaciones de la transformación de orden superior generan las relaciones de transformación de copia:

(i) Para cada meta-clase no abstracta, se genera una relación de copia (Class2CopyRelation).

(ii) Para cada subclase, se añade una llamada negada a la relación de identificación correspondiente para la cláusula when de una relación de copia (SubClass2MarkerCallInWhen).

(ii) Para cada meta-clase, se genera una relación de identificación (Class2MarkerRelation ).

(iv) Para cada atributo, se genera una relación (Atribute2CopyRelation ).

(v) Para cada relación, se genera una relación de copia (Reference2CopyRelation ).

Vamos a discutir los detalles de la generación en el resto de esta sección.

## 4.2 Generación de relaciones de copia de Metaclases

Los listados 4.2 y 4.3 muestran las relaciones para la generación de relaciones de copia desde meta-clases. Para cada meta-clase no abstracta en un paquete del meta-modelo se crea una relación de copia (Class2CopyRelation ). El patrón para crear la relación parece bastante complicada debido al complicado anidamiento de la sintaxis abstracta. Durante la creación se genera una cláusula where que contiene una RelationCallExp para llamar correspondiente relación de identificación. Los contenidos actuales de esta llamada se generan en la relación Class2MarkerCall

*top relation Class2CopyRelation {*

*sourceMM, targetMM : QVTBase::TypedModel;*

*checkonly domain mm eClass: ecore::EClass {*

*ePackage = ePackage: ecore::EPackage {},*

*name = n : String{},*

*abstract = false*

*};*

*enforce domain qvt rel: Relation {*

*name = 'Copy' + n,*

*isTopLevel = true,*

*variable = sourceVar: ocl::ecore::Variable {},*

*variable = targetVar: ocl::ecore::Variable {},*

*\_domain = sourceDom: QVTRelation::RelationDomain {*

*isCheckable = true*

*},*

*\_domain = targetDom: QVTRelation::RelationDomain {*

*isEnforceable = true*

*},*

*\_transformation = transfo : RelationalTransformation {},*

*\_where = wherePattern: QVTBase::Pattern {*

*predicate = pred: QVTBase::Predicate {*

*conditionExpression = markerCall:RelationCallExp{}*

*}*

*}*

*};*

*when {*

*Package2Transformation(rootPackage(ePackage), transfo)*

*or MarkTransformation(transfo);*

*MarkTypedModel(sourceMM, targetMM);*

*}*

*where {*

*Class2Domain(sourceMM,eClass,'source',sourceVar,sourceDom);*

*Class2Domain(targetMM,eClass,'target',targetVar,targetDom);*

*Class2MarkerCall(eClass,sourceVar,targetVar,markerCall);*

*}*

*}*

El contenido actual de esta llamada se genera en la relación Class2MarkerCall.

### Listado 4.2 generación de una relación de copia de una meta-clase.

Los dominios utilizados como fuente y destino en la relación Class2CopyRelation (ver listado 4.3) llaman a la relación Class2Domain en ss cláusulas where para construir el DomainPattern que es utilizado para encontrar coincidencias en la fuente y construir el dominio destino de las relaciones de copia.

Las ObjectTemplateExpressions se utilizan para encontrar correspondencias de la clase de los elementos del modelo que van a ser copiados.

relation Class2Domain {

checkonly domain qvt mm: QVTBase::TypedModel {};

checkonly domain mm eClass: ecore::EClass {

name = className: String {}

};

primitive domain prefix: String;

enforce domain qvt var: ocl::ecore::Variable {

name = prefix + className, eType = eClass

};

enforce domain qvt dom: QVTRelation::RelationDomain {

typedModel = mm,

name = prefix + className,

rootVariable = var,

pattern = p: QVTRelation::DomainPattern {

templateExpression = expr: ObjectTemplateExp {

referredClass = eClass,

eType = eClass,

bindsTo = var: ocl::ecore::Variable {}

},

bindsTo = var: ocl::ecore::Variable {}

}

};

}

### Listado 4.3 Generación de dominios para las relaciones.

Las cláusulas when de las relaciones de copia se generan en una relación separada (SubClass2MarkerCallInWhen) Para cada subclase directa, se genera una relación negada CallExp llamando a la relación de identificación

top relation SubClass2MarkerCallInWhen {

checkonly domain mm subClass: ecore::EClass {

eSuperTypes = superClass: ecore::EClass {}

};

checkonly domain oclstdlib notOp: EOperation { ... };

enforce domain qvt rel: Relation {

\_when = whenPattern: QVTBase::Pattern {

predicate = pred: QVTBase::Predicate {

conditionExpression = notCall: OperationCallExp {

source = markerCall: RelationCallExp {},

referredOperation = notOp,

eType = booleanType

}

}

}

};

when {

Class2CopyRelation(superClass, rel);

}

where {

Class2MarkerCall(

subClass,rel.variable->at(1)

, rel.variable-> at(2),markerCall);

}

### Listado 4.4 Generación de negación de llamadas a identificadores de subclases.

## 4.3. Generación de relaciones de identificación de meta-clases

El listado 4.4 muestra las relaciones para la generación de las relaciones de identificación. Se generan relaciones de identificación para cada meta-clase, incluyendo las abstractas (Class2MarkerRelation).Para cada superclase directa, se añade una llamada a la relación de identificación de esta superclase en la cláusula wherre. De nuevo, esto se consigue mediante una relación separada(Class2MarkerCallInPattern cf) similar a las llamadas negadas en las cláusulas when.

top relation Class2MarkerRelation {

sourceMM, targetMM: QVTBase::TypedModel;

checkonly domain mm eClass: ecore::EClass {

ePackage = ePackage: ecore::EPackage {},

name = n : String{}

};

enforce domain qvt rel: Relation {

name = 'Mark' + n,

isTopLevel = false,

variable = sourceVar: ocl::ecore::Variable {},

variable = targetVar: ocl::ecore::Variable {},

\_domain = sourceDom: QVTRelation::RelationDomain {

isCheckable = true

},

\_domain = targetDom: QVTRelation::RelationDomain {

isCheckable = true

},

\_transformation = transfo: QVTRelation::RelationalTransformation {},

\_where = wherePattern: QVTBase::Pattern {}

};

when {

Package2Transformation(rootPackage(ePackage), transfo)

Or MarkTransformation(transfo);

MarkTypedModel(sourceMM, targetMM);

}

where {

Class2Domain(sourceMM, eClass, 'source', sourceVar, sourceDom);

Class2Domain(targetMM, eClass, 'target', targetVar, targetDom);

eClass.eSuperTypes -> forAll(

st | Class2MarkerCallInPattern(

st, sourceVar, targetVar, wherePattern)

);

}

}

### **Listado 4.5 Generación de relaciones de identificación de mata-clases**

## 4.4. Generación de relaciones de copia para atributos y referencias

Finalmente el Listado 4.5 muestra una relación para la generación de relaciones de copia de attributos. Esta relación genera una relación de copia para cada atributo en el meta-modelo. Una relación análoga (cf Reference2CopyRelation) se utiliza para crear copias de las referencias. De nuevo omitimos los patrones mencionados en otros listados.

top relation Attribute2Relation {

sourceMM, targetMM: QVTBase::TypedModel;

checkonly domain mm attribute: ecore::EAttribute {

name = attrName: String {},

eType = attrType: ecore::EDataType {},

eContainingClass = eClass: ecore::EClass {

ePackage = ePackage: ecore::EPackage {},

name = className: String {}

}

};

checkonly domain oclstdlib notOp: EOperation { ... };

enforce domain qvt rel: Relation {

name = 'CopyAttribute\_' + className + '\_' + attrName,

isTopLevel = true,

variable = attrVar: ocl::ecore::Variable {

name = 'local\_' + attrName + 'Value',

eType = attrType

},

variable = sourceVar: ocl::ecore::Variable {},

variable = targetVar: ocl::ecore::Variable {},

variable = attrSourceVar: ocl::ecore::Variable {},

variable = attrTargetVar: ocl::ecore::Variable {},

\_domain = sourceDom: QVTRelation::RelationDomain {

isCheckable = true

},

\_domain = targetDom: QVTRelation::RelationDomain {

isEnforceable = true

},

\_transformation = transfo: QVTRelation::RelationalTransformation {},

\_when = whenPattern: QVTBase::Pattern {

predicate = pred: QVTBase::Predicate {

conditionExpression = markerCall: RelationCallExp {}

}

}

};

when {

Package2Transformation(rootPackage(ePackage), transfo)

Or MarkTransformation(transfo);

MarkTypedModel(sourceMM, targetMM);

}

where {

Class2Domain(sourceMM, eClass, 'source', sourceVar, sourceDom);

Attribute2Template(

attribute, 'source',

attrVar,

attrSourceVar,

sourceDom.pattern.templateExpression

);

Class2Domain(targetMM, eClass, 'target', targetVar, targetDom);

Attribute2Template(

attribute,

'target',

attrVar,

attrTargetVar,

targetDom.pattern.templateExpression

);

Class2MarkerCall(eClass, sourceVar, targetVar, markerCall);

}

}

### Listado 4.6 Generación de relaciones copia de atributos y referencias.

Para hacer coincidir los valores de los atributos que se se copian, Attribute2Relation llama a la relación Attribute2Template. Por esta se genera una variable que es luego cotejada con los atributos de origen y destino en un PropertyTemplateItem de la relación de copia. Ver el Listado 4.7 para los detalles de esta parte de la relación de copia.

relation Attribute2Template {

attributeOwningClass : ecore::EClass;

checkonly domain mm attribute: ecore::EAttribute {

name = attrName: String {},

eType = attrType: ecore::EDataType {}

};

primitive domain prefix: String;

checkonly domain qvt attrValVar: ocl::ecore::Variable {};

enforce domain qvt attrVar: ocl::ecore::Variable {

name = prefix + '\_' + attrName, eType = attrType

};

enforce domain qvt expr: QVTTemplate::ObjectTemplateExp {

part = attrTemplate: QVTTemplate::PropertyTemplateItem {

referredProperty = attribute,

value = featureExp: ocl::ecore::VariableExp {

referredVariable = attrValVar,

eType = attrType

}

},

eType = attributeOwningClass

};

where {

attributeOwningClass = attribute.eContainingClass;

}

}

### **Listado 4.7 Generación de plantillas de atributos.**

# 5. Redacción de transformaciones de refinamiento

Hay dos maneras de especificar una relación de transformación de refinamiento basadas en una transformación de copia generada. La primera escribiendo una nueva transformación manualmente que sobrescriba reglas específicas de la transformación genérica de copia. Se llama a las reglas de la transformación de refinamiento en vez de las sobrescritas desde las transformaciones de copia. Esto nos permite manejar las partes del modelo que deben ser tratadas, más que simplemente copiarlas. En segundo lugar, se puede definir una transformación que sólo contenga las reglas de excepción y utilizar entonces una transformación de orden superior para tejer estas reglas en una transformación de copia generada.

## Transformación manualmente extendida

De acuerdo con la especificación QVT Relacional [2], es posible definir extensión de transformaciones. En estas extensiones es posible definir reglas que sobrescriban reglas de la transformación extendida de forma condicional. Sin embargo la semántica exacta del mecanismo de sobre-escritura no está definida en el estándar. Por ejemplo, no se menciona si esta sobre-escritura condicional significa que para los elementos donde el patrón sobre-escrito no coincide se usa la regla sobre-escrita, o si el elemento no coincide del todo. Además, los motores de Relaciones QVT como mediniQVT [4] actualmente tampoco admiten esta función. No obstante se propone la extensión como una forma natural de especificar transformaciones de refinamiento con ayuda de una transformación de copia generada. El desarrollador de transformaciones tiene que asegurarse que sólo las reglas completas deben ser facilitadas por la extensión. Completas en este caso significa que para cada regla de transformación de refinamiento, se debe generar una regla complementaria que empareje todos los elementos que no sean emparejados por la regla de refinamiento. Sólo entonces se puede asegurar que todos los elementos del modelo, no tratados por la regla de refinamiento. Una forma fácil de asegurarlo es llamar las reglas de refinamiento así como la regla de copia correspondiente utilizando un or exclusivo (XOR) en la sentencia where de una tercera regla que tiene el mismo origen y destino que la regla de copiado. Otro problema con la extensión es que el usuario debe asegurarse que las reglas de identificación correspondientes se llaman desde las reglas de refinamiento. Si no todas las reglas de copiado dependientes no se ejecutarán. Un ejemplo de esta extensión manual se puede ver en el listado 5.1. Tenga en cuenta que la llamada a la relación original de copia (copyPN::copyNet) simboliza que la regla de transformación original debería ser llamada aquí. Como el estándar no dice como hacer esto, también podría ser necesario copiar manualmente esta regla como una regla no superior para que la transformación de refinamiento pueda llamarla en la cláusula where de refinedCopyNetCall.

transformation refinePN extends copyPN (orig: petri, copy: petri) {

top relation refinedCopyNetCall overrides copyPN::copyNet{

checkonly domain orig netO: Net {};

enforce domain copy netC: Net {};

where {

if (refinedCopyNet(netO, netC)) then

true

else

copyPN::copyNet(netO, netC)

endif;

}

}

relation refinedCopyNet {

toks, newToks : Integer;

checkonly domain orig netO: Net {

p = refP : Place{token = toks}

};

enforce domain copy netC: Net {

p = refP : Place{token = newToks}

};

when {toks = 0;}

where {

newToks = toks+1;

copyPN::copiedNet(netO, netC);

}

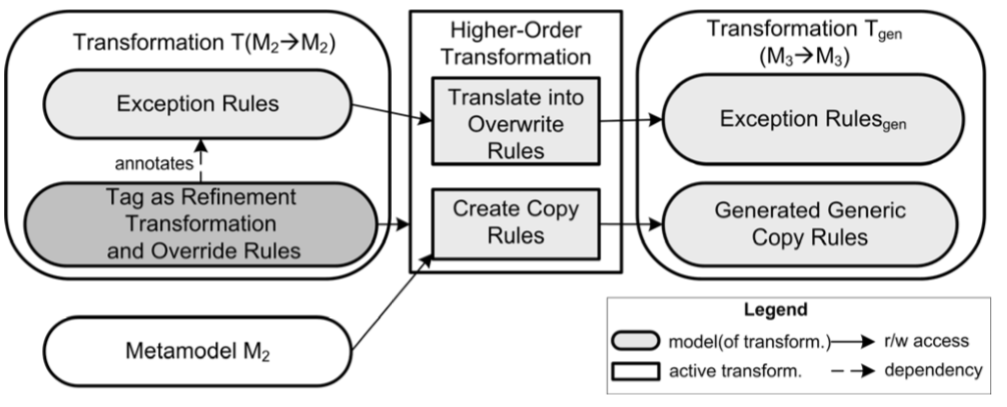
}

}

### Listado 5.1 Uso de extensiones manuales para el refinamiento de la transformación de copia

## Tejido automático

La segunda posibilidad para crear la transformación de refinamiento deseada se basa en la técnica de tejido de transformaciones. Como se expone en la figura 5.1 las reglas que re-ensamblan los refinamientos deseados se especifican como transformaciones QVT Relacional estándar. Como paso siguiente esta transformación es anotada como una transformación de refinamiento. Esta anotación puede por ejemplo ser realizado poniendo comentarios en la sintaxis concreta textual de la transformación, o como un modelo decorator no intrusivo en el modelo de la transformación.

**

### Figura 5.1 Tejido automático de reglas de excepción

La transformación anotada es entonces transformada en una transformación de orden superior (cf HOT). Esta transformación traduce las reglas de refinamiento de una manera que se ajusten a las llamadas esperadas de las relaciones de identificación. Además se generan reglas de copia (como las definidas en la sección 4) para aquellos patrones que no son manejados por el refinamiento. Este paso se basa en el meta-modelo que se utiliza como parámetro de entrada y salida de la transformación de refinamiento. Como las reglas de refinamiento sólo deben coincidir con una parte de los elementos que serían emparejadas normalmente por sus reglas de copiado correspondientes, una construcción similar debe ser introducida tal y como se muestra en la figura 5.1. Usando la técnica de anotación y tejido el desarrollador no necesita preocuparse acerca de asegurar la consistencia de las reglas de refinamiento con las reglas de copiado generadas. Sin embargo, ya que el desarrollador no necesariamente ve la transformación generada que se ejecuta, la depuración se complica en este caso. Se requeriría soporte adicional para trazar la información de depuración de la transformación generada hacia la transformación original. Preferimos utilizar transformaciones modelo-a-modelo a transformaciones modelo-a-texto para las transformaciones de orden superior ya que las trazas se incluyen en la mayoría de estos motores de transformación. Estas trazas pueden ser utilizadas fácilmente durante la depuración para navegar fácilmente desde la generada, en realidad los elementos de la transformación original depurada a los elementos especificados manualmente. Una técnica similar de refinamiento se describe en [6], llamada superimposition. La transformación que superpone vagamente sobrescribe reglas de la transformación superpuesta con el mismo nombre. Se utiliza una transformación de orden superior para juntar ambas transformaciones en una tercera. Un método similar se ha utilizado aquí cuando se tejen las reglas explícitas de refinamiento con reglas de copiado generadas. Sin embargo en este caso no hay transformación superpuesta explícitamente, ya que las reglas de copia (que serían superpuestas por las reglas de refinamiento) también se generan durante el proceso de tejido y no existían antes.

# 6. Observaciones finales

Hemos presentado un método que permite a los desarrolladores de transformaciones eludir la falta de reglas de copiado por defecto en QVT Relacional. Hemos presentado patrones genéricos para copiar instancias, ranuras, y enlaces. Basándonos en estos patrones hemos facilitado una transformación de orden superior para generar transformaciones de copia desde un meta-modelo. La transformación de orden superior está escrita en QVT Relacional. Finalmente hemos propuesto dos posibilidades diferentes sobre cómo el método puede ser integrado en el desarrollo de diferentes transformaciones de refinamiento. El trabajo futuro se enfocará al soporte de transformaciones de refinamiento de transformaciones exógenas.

# Referencias

1. Czarnecki, K., Helsen, S.: Classification of model transformation approaches. 1 Czarnecki, K., Helsen, S.:. Clasificación de transformación de modelos se acerca.

OOPSLA (2003) OOPSLA (2003)

2. Object Management Group: Meta Object Facility (MOF) 2.0 Que- . 2 Object Management Group: Meta Object Facility (MOF) 2.0 Que-

ry/View/Transformation (QVT) http://www.omg.org/docs/formal/08-04-03.pdf. ry / Vista / Transformación (QVT) http://www.omg.org/docs/formal/08-04-03.pdf.

3. Mens, T, Gorp, PV: A taxonomy of model transformation. . 3 para hombre, T, Gorp, PV: Una taxonomía de transfor mación de modelos. GraMoT (2005). GraMoT (2005).

4. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. 4 Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.:. ATL: Una herramienta de transformación de modelos.

Science of Computer Programming, Special Issue on Second issue of experimen- Ciencias de Programación de Computadoras, número especial sobre el Segundo tema de la experimentación

tal software and toolkits (EST) 72(1-2) (2008) 31–39 software y juegos de herramientas (EST) 72 (1-2) (2008) 31-39 tal

5. ikv++: medini QVT. 5 ikv + +:. Medini QVT. http://www.ikv.de/ Last retrieved 2008-10-30. http://www.ikv.de/ Última recuperado 2008-10-30.

6. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. . 6 Schürr, A.: Especificación de Traductores Gráfico con Triple Gráfico Gramáticas. In G. En G.

Tinhofer, editor, WG'94 20th Int. Tinhofer, editor, WG'94 20 Int.. Workshop on Graph-Theoretic Concepts in Taller sobre Conceptos Gráfico-teóricas en

Computer Science, volume 903 of Lecture Notes in Computer Science (LNCS), Ciencias de la Computación, volumen 903 de Lecture Notes in Computer Science (LNCS),

pages 151–163,Heidelberg, 1994. Springer Verlag. páginas 151-163, Heidelberg, 1994. Springer Verlag.

7. Wagelaar, D.: Composition Techniques for Rule-Based Model Transformation 7 Wagelaar, D.:. Técnicas de Composición para basada en reglas modelo de transformación

Languages, 1 Idiomas, 1staInternational Conference on Model Transformation - Theory and Conferencia Internacional sobre el Modelo de Transformación - Teoría y

Practice of Model Transformations, volume 5063 of Lecture Notes in Computer Práctica del modelo Las transformaciones, volumen 5063 de Lecture Notes in Computer Science (LNCS), pages 152-167, Heidelberg, 2008. Springer Verlag. Science (LNCS), páginas 152-167, Heidelberg, 2008. Springer Verlag.