Implementando la semántica de QVT mediante el mecanismo de transformación de modelos

Carina Moldes
Silvana Mossi

Directora de Tesis: Dra. Claudia Pons

Agenda

- Objetivos - Aportes
- Desarrollo Dirigido por modelos
- Transformaciones MDA
- QVT: el estándar de OMG para transformaciones
- Generación de modelos: manual y automática
- ATL
- Semántica de QVT
- Demo

Objetivos - Aportes

- *Investigar* herramientas de MDD que aplican para definir la semántica de un lenguaje.
- *Implementar* la transformación.
- *Aplicabilidad* de un lenguaje de transformación para implementar la semántica de otro lenguaje de transformación.

Desarrollo de Software Dirigido por Modelos

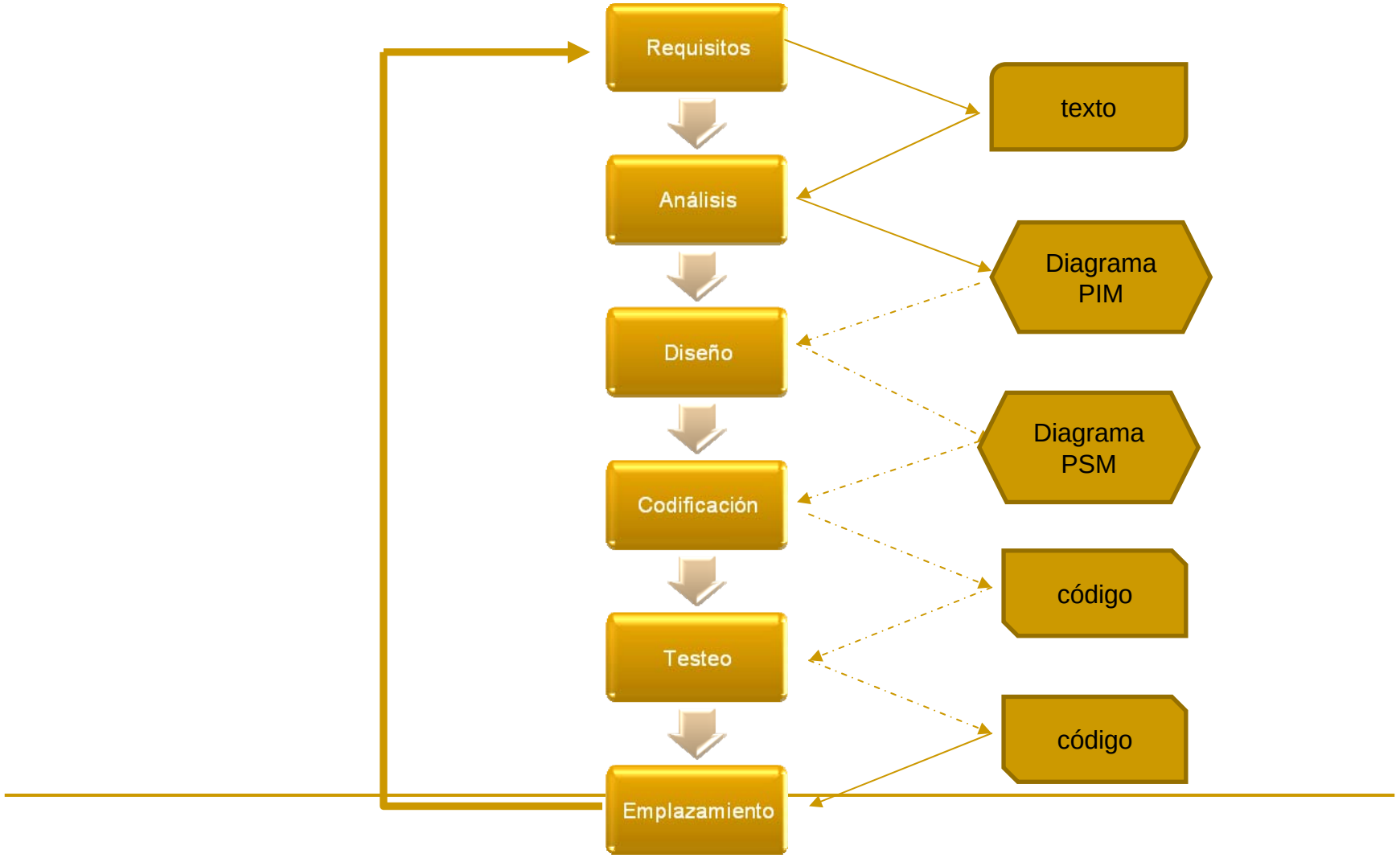
MDD – Ciclo de vida – MDA

Concepto

- Nueva ***área*** dentro del campo de la Ingeniería de Software.
- Nuevo ***paradigma*** de desarrollo de software.

- Hace uso de un proceso ***dirigido*** por modelos.
- La ***transformación*** entre modelos constituye el motor del MDD.

El ciclo de vida dirigido por modelos



Arquitectura Dirigida por Modelos

- **Concepto**
  - Interoperabilidad
  - Portabilidad
  - Separar diseño del sistema tanto de la arquitectura como de las tecnologías.

- **Funcionamiento**
  - Modelar en CIM.
  - Modelar el PIM.
  - Genera un PSM.
  - Traducir a código.

Transformaciones MDA

Elección de un lenguaje – Criterios de análisis

Elección de un lenguaje

- Variedad de tipos.
- Declarativos, operacionales.
- Composición de transformaciones.
- Basados en QVT.
- Plug-in de Eclipse.

Criterios de análisis

- **Definición**:
  - Tipo: gráfico o textual, m2m o m2t
  - Estilo: declarativo u operacional.
  - Pre y post condiciones
  - MOF/QVT compatible

Criterios de análisis (cont.)

- **Reglas de Transformación:**
  - Dominio: lenguajes, dirección.
  - Cuerpo de la transformación: declaración de variables, separación sintáctica, estructuras intermedias.
  - Aplicación de las reglas: orden (determinístico, no determinístico), aplicación condiciional, iteración de reglas.
  - Organización de las reglas: modularización, reuso.

Criterios de análisis (cont.)

- **Trazabilidad**:
  - Mantiene conexiones entre los elementos del dominio y del codominio.
- **Composición:**
  - Se encadenan dos o mas transformaciones.
  - Se componen dos o mas transformaciones en una nueva transformación

QVT: el estándar de OMG para transformaciones

QVT: el estándar de OMG para transformaciones

- Uno de los principales objetivos del DSDM, es la transformación entre modelos. Por ello la OMG solicitó la creación de un estandar de transformación en el ámbito de MOF.

QVT: el estándar de OMG para transformaciones

Características:

- Permita definir transformaciones entre metamodelos definidos bajo MOF.
- Permita crear la vista de un modelo.
- Permita flexibilidad en su implementación.
- Sea declarativo.

QVT: el estándar de OMG para transformaciones

- QVT es un estándar que define la manera en la cual se llevan a cabo las transformaciones entre modelos cuyos metamodelos se basan en MOF. Este estándar consta de tres componentes : consultas, vistas y transformaciones.

QVT: el estándar de OMG para transformaciones

- Una consulta, es una expresión sobre un modelo. El resultado de la consulta, es una o mas instancias de los tipos definidos sobre el metamodelo fuente.

  Por ejemplo, si definimos una consulta sobre un modelo UML que retorna todas las clases del tipo persistente

  (select (c|c.kind = ”persistente”), el resultado podría ser una colección de las instancias tipo clase.

QVT: el estándar de OMG para transformaciones

- Como lenguaje de consulta, se emplea OCL. Con esta elección se obtienen los siguientes beneficios:
- La comunidad de usuarios ya está familiarizado con el lenguaje.
- No es necesario realizar un esfuerzo adicional para definir un nuevo lenguaje.
- Actualmente, OCL está implementado en muchas herramientas.

OCL(Object Constraint Language) es un lenguaje para escribir restricciones sobre objetos

QVT: el estándar de OMG para transformaciones

- Una vista, es un modelo derivado de otro modelo (modelo base). Una vista, no puede ser modificada independientemente del modelo del cual se deriva.

  Los cambios que se realicen en el modelo base, ocasionan cambios en la vista.

QVT: el estándar de OMG para transformaciones

- Transformaciones: Un proceso de transformación es la generación automática de un modelo destino a partir de un modelo fuente.

  Una transformación, se compone de un conjunto de reglas de transformación que juntas describen la forma como se transforma un modelo fuente en un modelo destino.

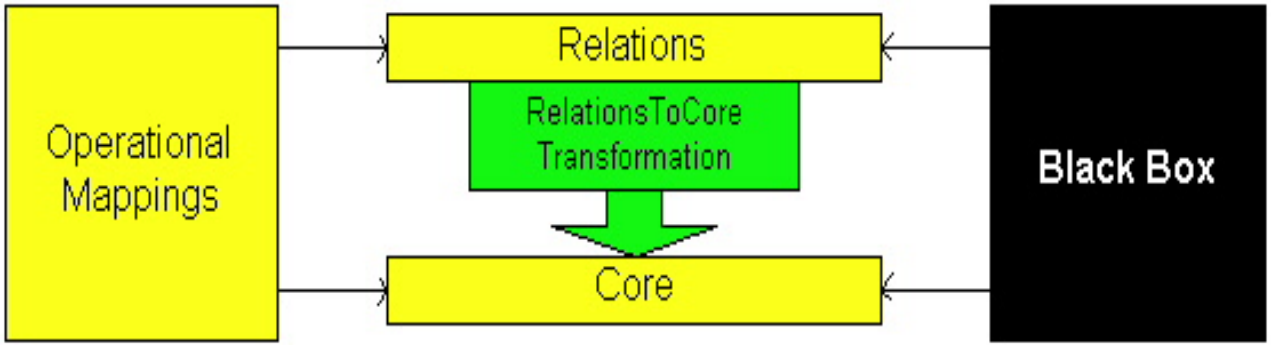
QVT: el estándar de OMG para transformaciones

- QVT comprende tres diferentes lenguajes M2M: dos lenguajes declarativos llamados Relations y Core, y un tercer lenguaje, de naturaleza imperativa, llamado Operational Mappings.

M2M: model to model

QVT: el estándar de OMG para transformaciones

- Relación entre metamodelos de QVT



QVT: el estándar de OMG para transformaciones

QVT declarativo. Dos Capas:

- Relations: es una especificación declarativa de las relaciones entre modelos MOF.

  relation PackageToSchema {

  checkonly domain uml p:Package{name=pn}

  enforce domain rdbms s:Schema {name=pn}}

- Core: lenguaje definido usando extensiones minimales de EMOF y OCL

QVT: el estándar de OMG para transformaciones

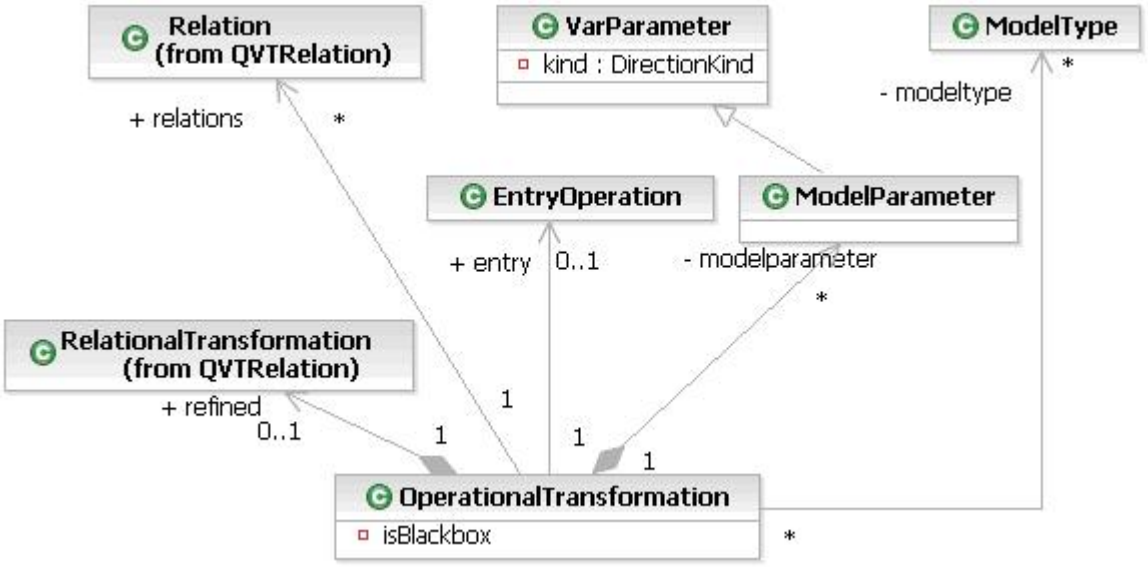
- QVT Operacional

  Este lenguaje puede ser usado en dos formas diferentes:

- Es posible especificar una transformación únicamente en el lenguaje *Operational Mappings*, este tipo de transformación se denomina *Transformación Operacional*
- Alternativamente, es posible trabajar en modo híbrido.

QVT: el estándar de OMG para transformaciones

- Metamodelo de QVT: Transformación Operacional



QVT: el estándar de OMG para transformaciones

- Una transformación Operacional representa la definición de una transformación unidireccional:

```
transformation Uml2Rdbms(in uml:UML,out
rdbms:RDBMS) {
// el punto de entrada de la ejecución
main() {
uml.objectsOfType(Package)->map
packageToSchema();
}
..
}
```

QVT: el estándar de OMG para transformaciones

Elementos que conforman una transformación operacional:

- EntryOperation: Una operación actuando como punto de entrada para la ejecución de la transformación operacional.
- ModelParameter: Indica la signatura de una transformación operacional. Un parámetro de modelo indica un tipo de dirección (in/out/inout) y un tipo dado por un tipo de modelo.
- ModelType: Cada parámetro de modelo se corresponde a un tipo de modelo que esta definido por la transformación. Un tipo de modelo esta definido por el metamodelo.

Generación de modelos

Problemática – Generación manual y automática

Generación de modelos

- Problemática:

  ATL trabaja con modelos definidos en archivos xmi. Nuestro modelo es un archivo qvt.

- Dos Soluciones:

  Crear el modelo desde EMF.

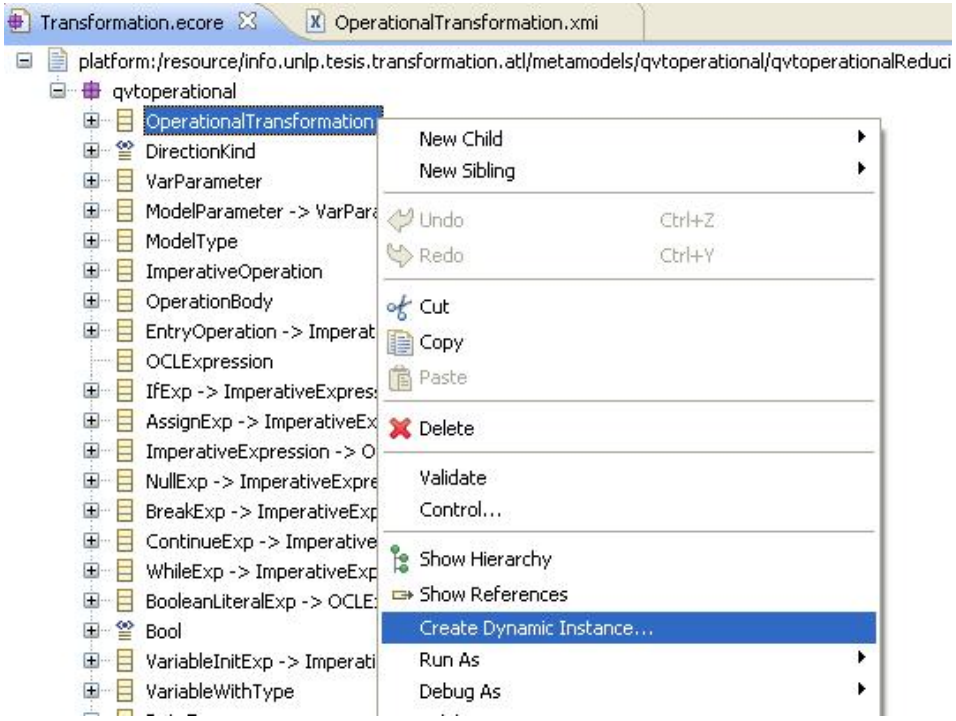
  Buscar otra alternativa que nos permita automatizar esta tarea.

Generación de modelo: Eclipse Modeling Framework

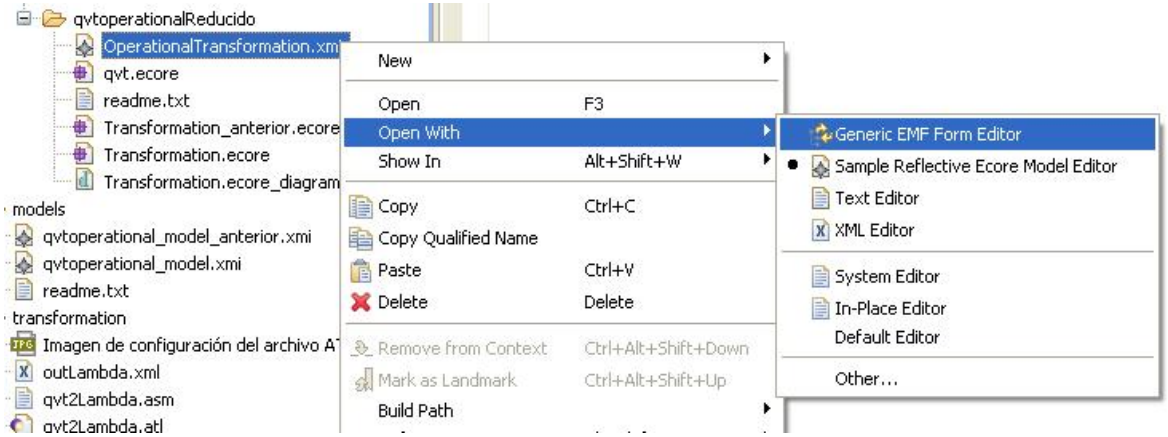
- EMF es un marco de trabajo de Eclipse que unifica Java, XML y UML, permitiendo a los desarrolladores construir rápidamente aplicaciones robustas basadas en modelos simples.
- El metamodelo usado para representar modelos en EMF se denomina Ecore.

Generación manual de instancias de los metamodelos con herramienta EMF

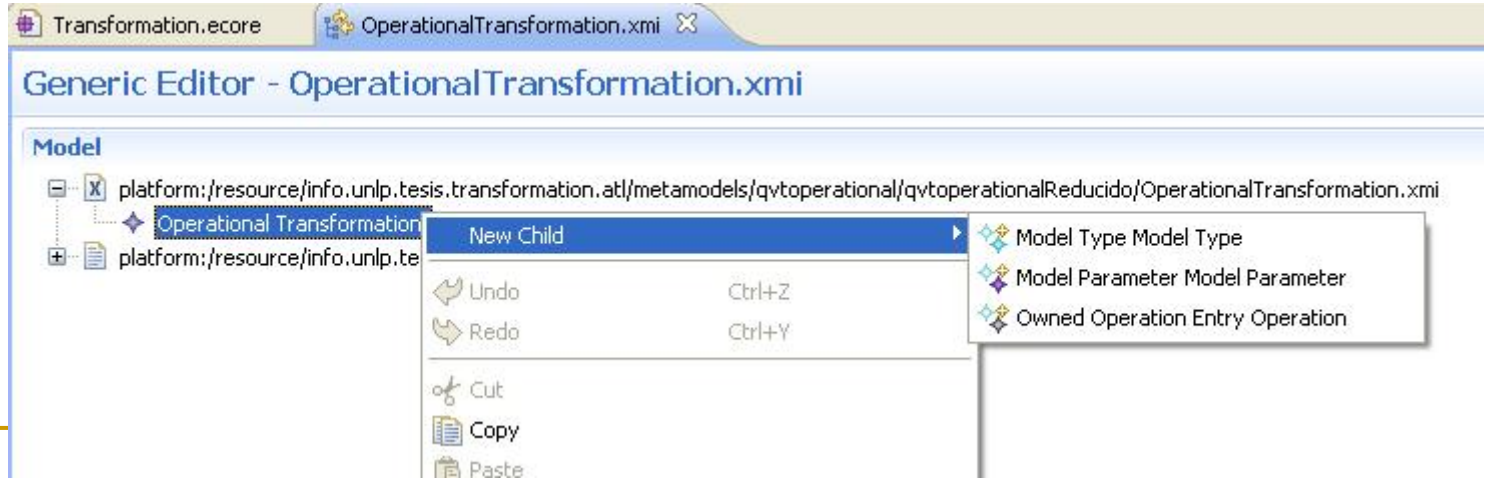
- Se crea la instancia a partir de la clase del metamodelo.



Generación manual de instancias de los metamodelos con herramienta EMF



Se editan las propiedades desde el Editor



Generación manual de instancias de los metamodelos con herramienta EMF

Ventajas:

- Solo se necesita del plugin EMF para obtener los modelos.
- Practicidad, por el hecho de trabajar con la misma herramienta.

Generación manual de instancias de los metamodelos con herramienta EMF

Desventajas:

- Esta metodología puede tornar muy compleja la creación del modelo si el mismo es relativamente grande.
- Una vez salvado el modelo, no es posible modificarlo, excepto que se edite el archivo que se generó lo cual no es una tarea sencilla de realizar.

Generación manual de instancias de los metamodelos con herramienta EMF

Conclusión:

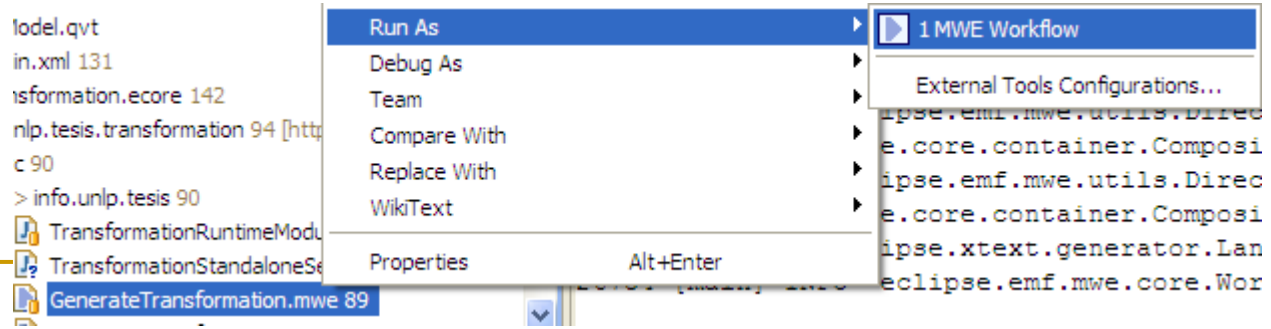
- Decidimos para nuestro trabajo buscar una forma de obtener los modelos en una manera más directa y automática, por lo cual se descartó el uso de EMF para la generación de las instancias de los modelos.

Generación automática de las instancias de los metamodelos mediante herramienta XTEXT

- Xtext permite crear editores de texto para lenguajes basados en EMF.
- Para crear un editor Xtext es necesario definir una gramática EBNF . El resto del trabajo es automático y como resultado se genera el código necesario para crear un editor textual.

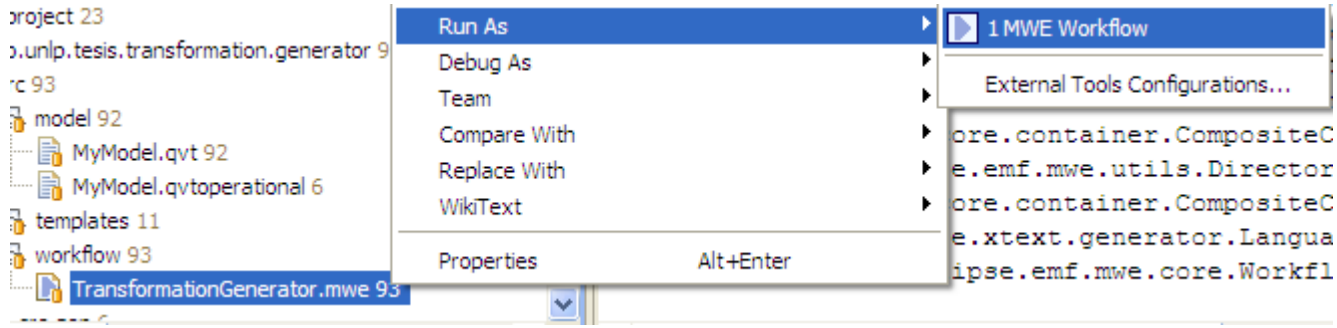
Xtext: Primer Etapa – Generación del editor

```
grammar info.unlp.tesis.Transformation with
org.eclipse.xtext.common.Terminals

generate qvtoperational "http://qvtoperational"

OperationalTransformation:
  (modelType+=ModelType)+
  'transformation' name=ID '(' (modelParameter+=ModelParameter) (','
(modelParameter+=ModelParameter))* ')' (';')?
  (ownedOperation=EntryOperation)?;

enum DirectionKind : in = 'in' | inout = 'inout' | out = 'out' ;

VarParameter:
  kind=DirectionKind|ModelParameter;

ModelParameter:
  kind=DirectionKind name=ID ':' modelType=[ModelType];

ModelType:
```



Xtext: Segunda Etapa – Generación del modelo en xmi

```
1modelType UML strict uses "http://www.eclipse.org/qvt/1.0.0/Operational/examples/simpleuml" ;
2modelType RDBMS strict uses "http://omg.qvt-samples.SimpleRdbms";
3transformation Uml2Rdbms (in uml:UML, out rdbms:RDBMS);
4main{
5var i : integer := 0;
6var cantidadPares : integer := 0;
7while ((i < 10)){
8    i := (i + 1);
9    if((((i/2)*2) = i)) then {cantidadPares := (cantidadPares + 1)} else {continue}
10}
11}
```



Generación automática de las instancias de los metamodelos mediante herramienta XTEXT

Ventajas

- La extracción de modelos se realiza en solo dos pasos, se escribe en un archivo el modelo, y se ejecutan los pasos necesarios para la generación.
- Es más intuitivo para trabajar , para el desarrollador es transparente el uso del formato xmi.
- En el caso de un error en el modelo, sólo se debe corregir el mismo, y ejecutar nuevamente el segundo workflow.

Generación automática de las instancias de los metamodelos mediante herramienta XTEXT

Desventajas:

- La principal desventaja se encuentra en el hecho de trabajar con una herramienta externa a ATL.
- Se debe definir previamente la sintaxis EBNF para el metamodelo.
- Es una herramienta que hoy en día se está desarrollando, por lo cual se actualiza constantemente.

Generación automática de las instancias de los metamodelos mediante herramienta XTEXT

Conclusión:

- Se optó por el uso de Xtext, dado que nos permitió trabajar de una manera más cómoda por la flexibilidad que presenta.

ATL (Atlas Transformation Language)

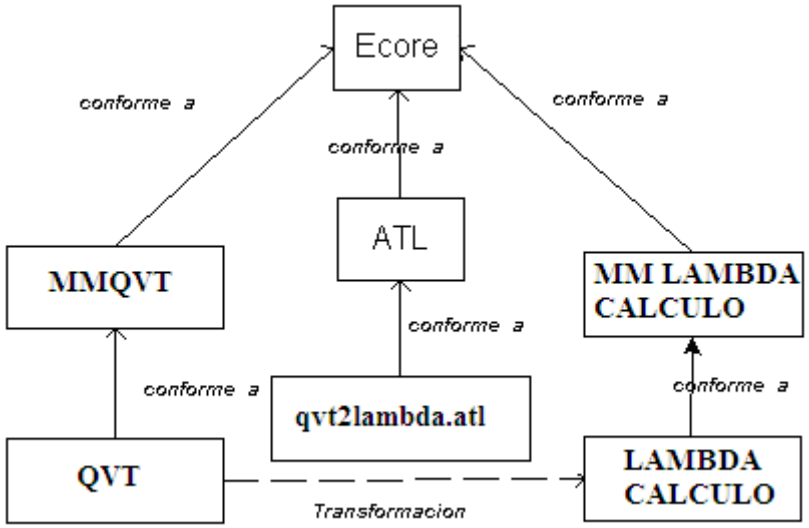
ATL

- ATL es un lenguaje de transformación de modelos y un conjunto herramientas que dan soporte a las transformaciones. Provee la manera de generar modelos destino a partir de modelos fuente.
- Los modelos fuente, destino y la definición de la transformación, responden a sus metamodelos respectivos y, a su vez, todos los metamodelos se ajustan a MOF.

ATL

ATL

- Transformación QVT2LambdaCalculo



ATL

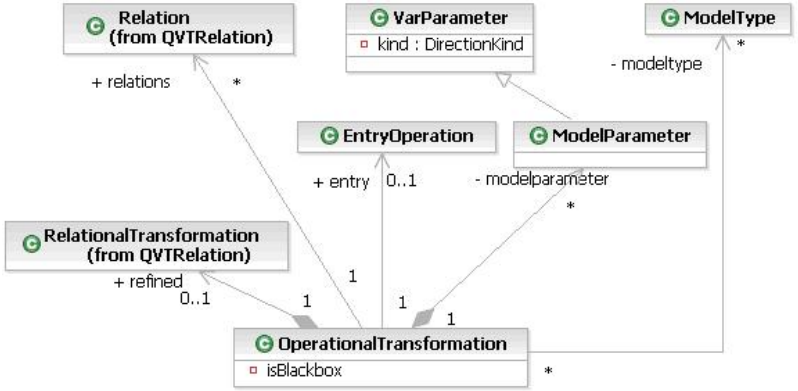
ATL module

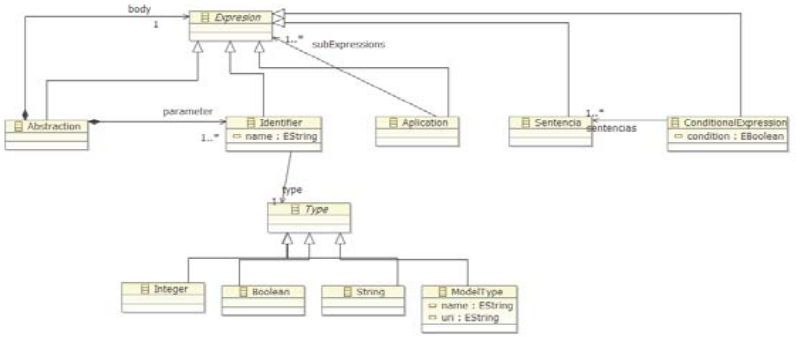
- Un modulo ATL corresponde a una transformación modelo a modelo.
- La sección cabecera que define algunos atributos que son relativos al modulo de transformación.
- Una sección opcional de librerías que permite importar algunas librerías existentes de ATL.
- Un conjunto de helpers que pueden ser vistos en ATL como un equivalente en Java a métodos.
- Un conjunto de reglas que define la manera que los modelos de salida son generados desde un modelo de origen.

ATL: semántica de if-then-else

```
module qvt2Lambda;
create OUT : Lambda from IN : QVT;

unique lazy rule IfExp2Abstraction{
      from ifQVT:QVT!IfExp, par:Lambda!ParSimple
    to ifLambda: Lambda!Abstraction(
      parameter <- par,
      body <- thisModule.IfExp2IfExp(ifQVT, par))

}
unique lazy rule IfExp2IfExp{
   from ifQVT:QVT!IfExp, par:Lambda!ParSimple
   to ifLambda: Lambda!IfExp(
      expCondition <- ifQVT.condition.getExpression(par),
      expThen<- ifQVT.thenExpression.getExpression(par),
      expElse<- ifQVT.elseExpression.getExpression(par))}
```

Integración XTEXT y ATL

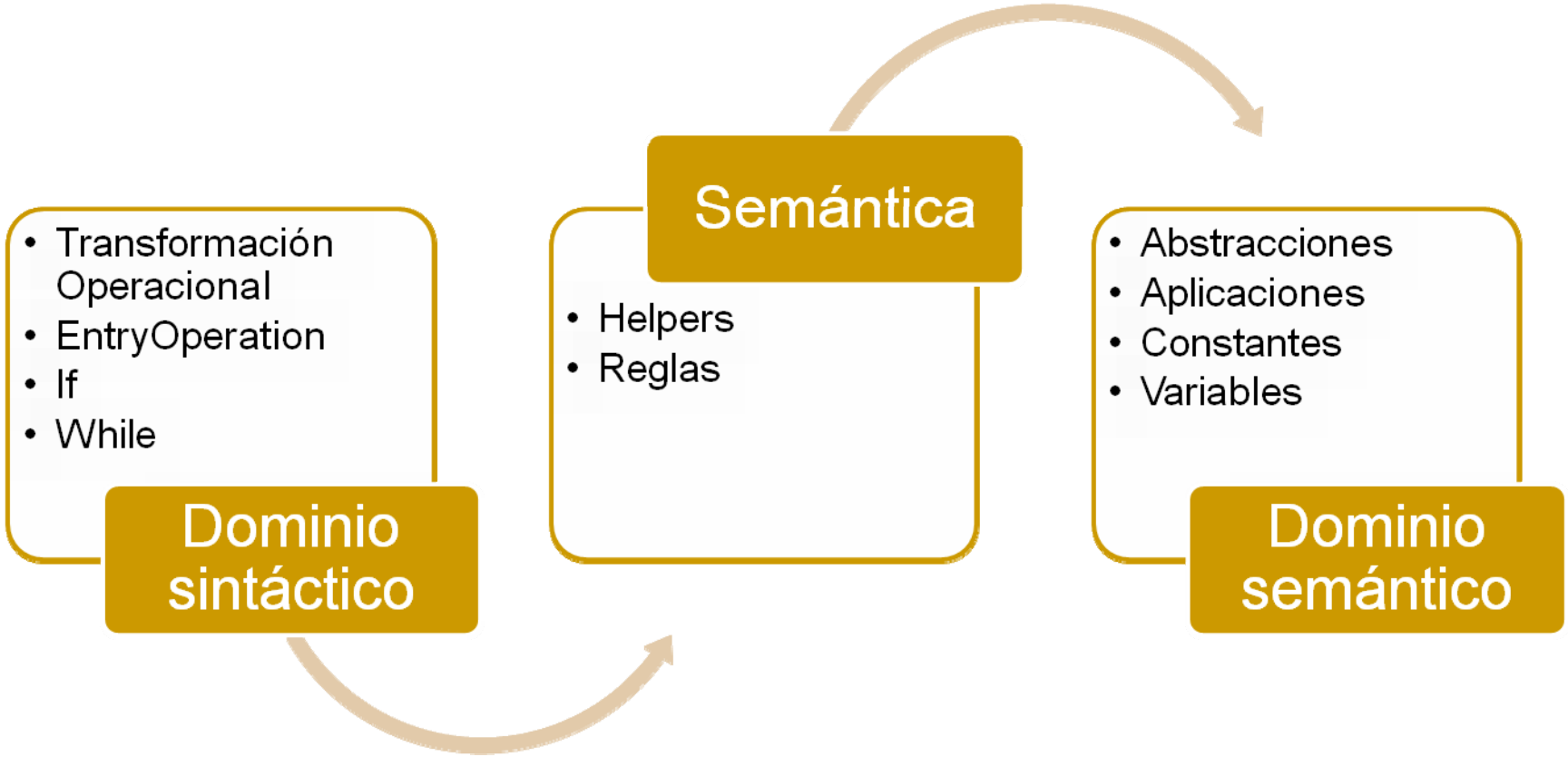




Semántica de QVT

Definición – Implementación – Metamodelo de QVT – Metamodelo de Lambda Cálculo

Definición


Transformación Operacional
EntryOperation
If
While
Dominio sintáctico
Semántica
Helpers
Reglas
Abstracciones
Aplicaciones
Constantes
Variables
Dominio semántico

Implementación

```
modelType UML strict uses "http://www
modelType RDBMS strict uses "http://or
transformation Uml2Rdbms (in uml:UML,
main(
var i : integer := 0;
var cantidadPares : integer := 0;
while ((i < 10))(
    i := (i + 1);
    if((((i/2)*2) = i)) then (cantida(
)
)
```

QVT

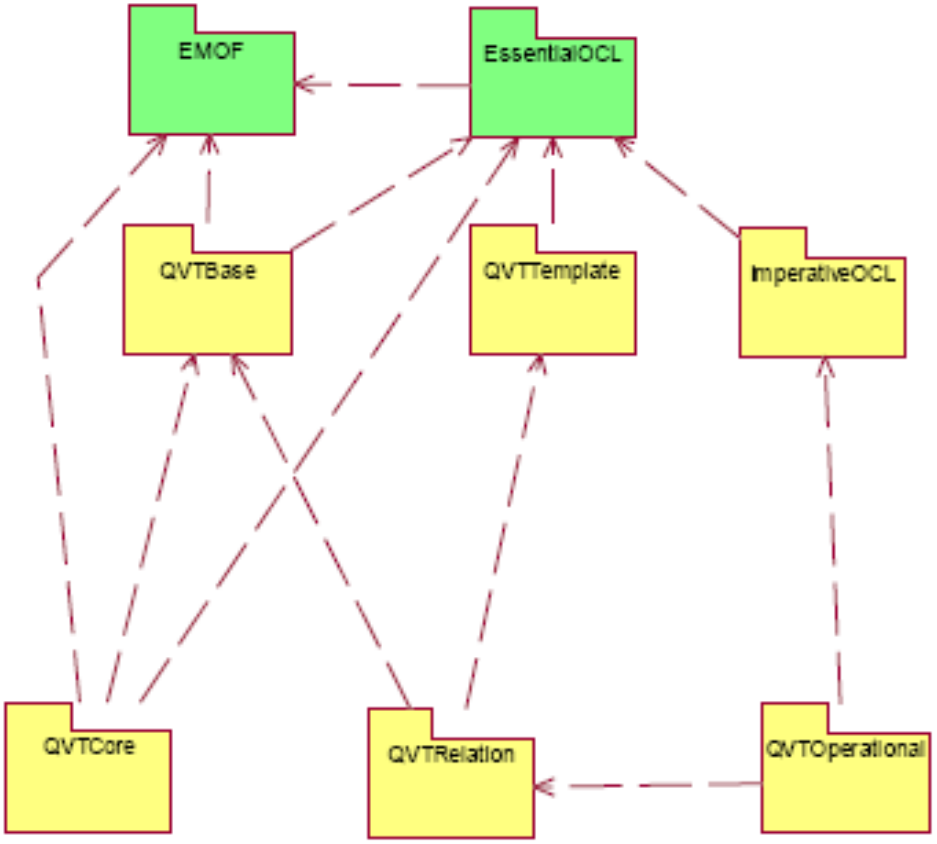
Implementación en ATL

```
-- @path QVT=/info.unlp.tesis.transf
-- @path Lambda=/info.unlp.tesis.tra

module qvt2Lambda;
create OUT : Lambda from IN : QVT;

...

-- Esta regla es de donde COMIENZA l
-- matchedRule: a partir de una Oper
rule operationalTransformation2Lambd
from ot : QVT!OperationalTransformat
using {
    x:String = 'x';
    y:String = 'y';
}
to otLambda : Lambda!Abstraction(
            parameter <- thisModule.
            body <- secondElement),
            secondElement :  Lambda!
                exp <- thisModule.Ent
            )
}

...
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi=
 <lambda:Abstraction>
  <parameter type="/1" name="uml"/>
  <body xsi:type="lambda:Snd" exp="/4"/>
 </lambda:Abstraction>
 <lambda:ModelType name="UML" uri="http://www.eclipse.org/qvt/1.0.0/Op
 <lambda:StringType value="x"/>
 <lambda:StringType value="y"/>
 <lambda:Abstraction>
  <body xsi:type="lambda:Abstraction">
   <body xsi:type="lambda:Abstraction">
    <body xsi:type="lambda:ParExpression" e1="/5" e2="/7"/>
   </body>
  </body>
 </lambda:Abstraction>
 <lambda:Variable name="i"/>
 <lambda:Variable name="cantidadPares"/>
 <lambda:ParExpression e1="/6" e2="/28"/>
 <lambda:Abstraction>
  <body xsi:type="lambda:IfExp" expThen="/26/@body" expElse="/9">
   <expCondition href="file:/E:/runtime-EclipseApplication/test%20plugin/src
  </body>
 </lambda:Abstraction>
```

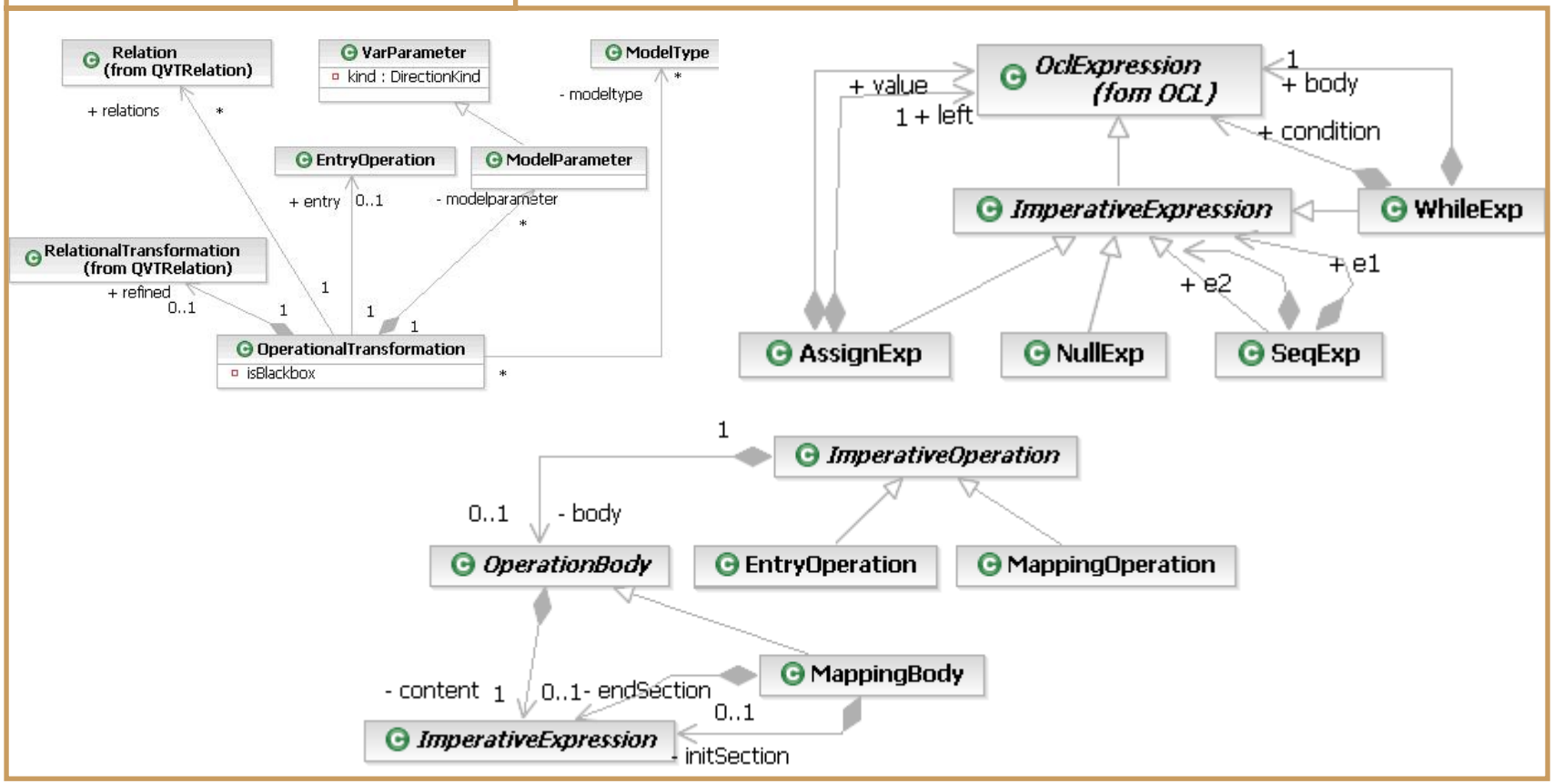
Lambda Cálculo

Implementación –Metamodelo de QVT

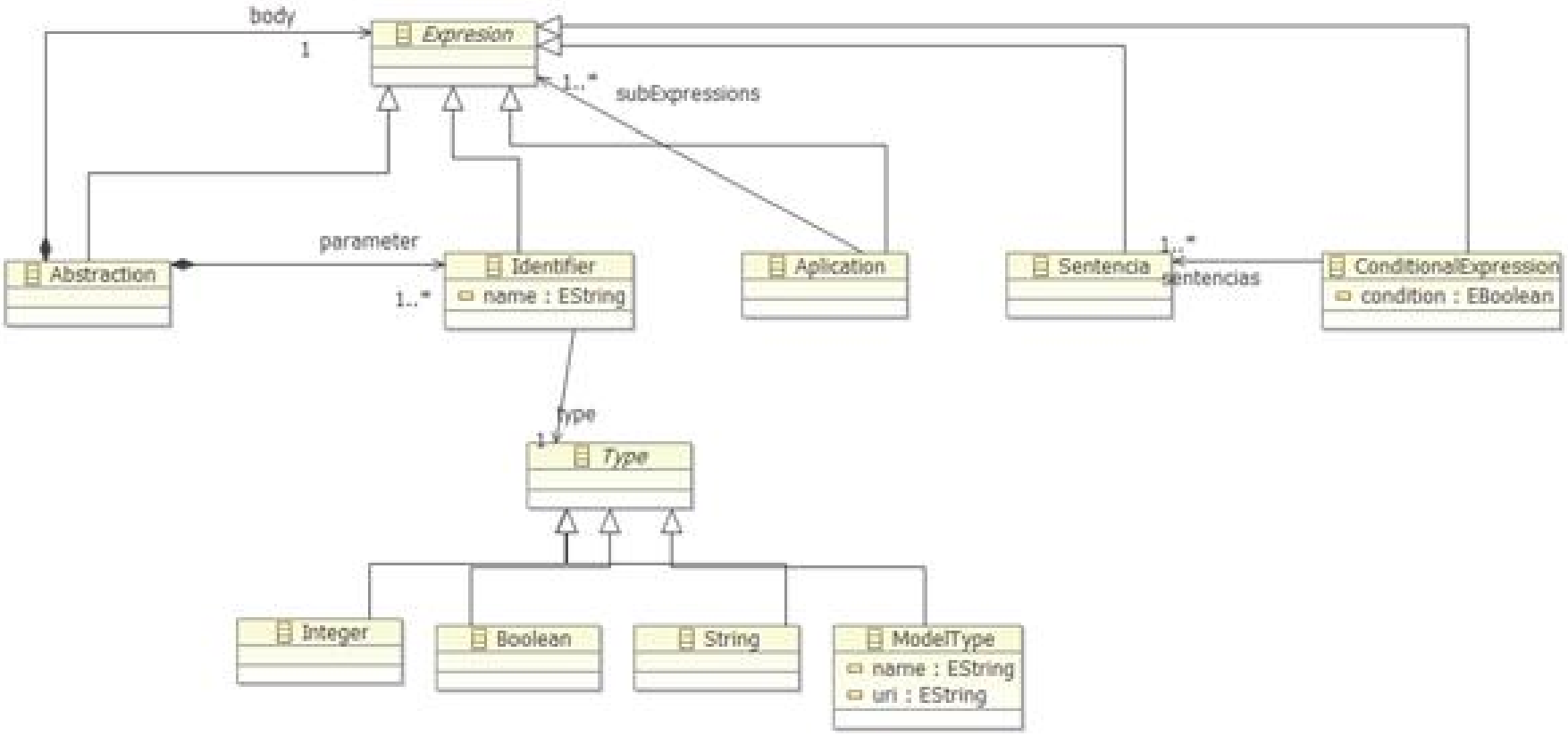


Implementación –Metamodelo de QVT

QVTOperacional



Implementación –Metamodelo de Lambda Cálculo



Demo

Herramienta para Eclipse