

Practice of Model Transformation and Model Weaving in the Eclipse Modeling Project with ATL and AMW



Part 2

**Atlas Model Weaver:
An adaptive plug-in to capture
relationships between model elements**

**Marcos Didonet Del Fabro (1)
Juan M. Vara (2)**



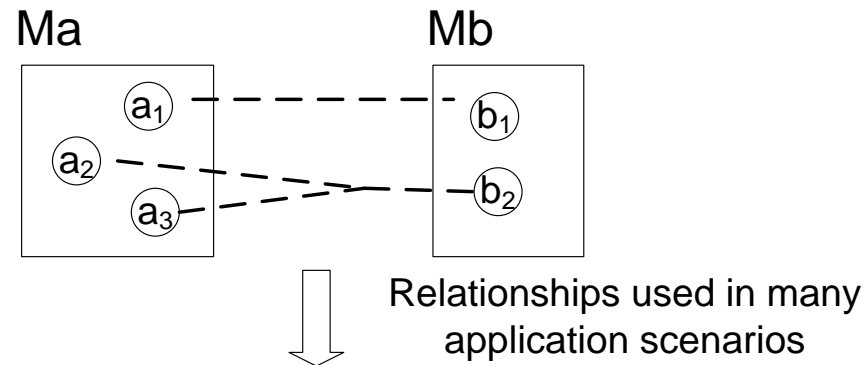
(1) ATLAS Group (INRIA & LINA), University of Nantes (France)
(2) Kybele Research Group, University Rey Juan Carlos of Madrid (Spain)

Outline

- Overview of model weaving
- Use cases (demo)
 - Simple AMW
 - Transformation production with AMW
 - Model matching
 - Model Annotation

Relationships between model elements

- It is often necessary to establish relationships between elements of different models

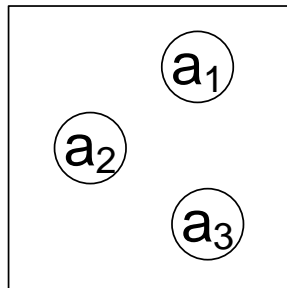


Traceability, transformation production, merging, aspect oriented modeling (AOM), etc.

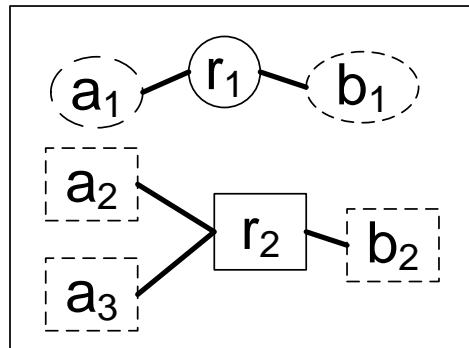
Model weaving

- ATLAS solution to capture relationships between model elements
- Relationships are reified in a **weaving model**
 - The model elements represent the relationships and the related elements
 - As any kind of model, the weaving model can be saved, stored, transformed, modified
 - Different kinds of links
 - Equality, concatenation, equivalence, etc.

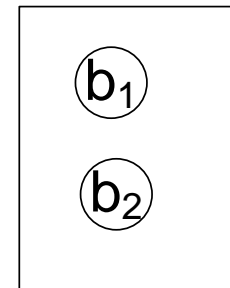
Ma



Weaving model

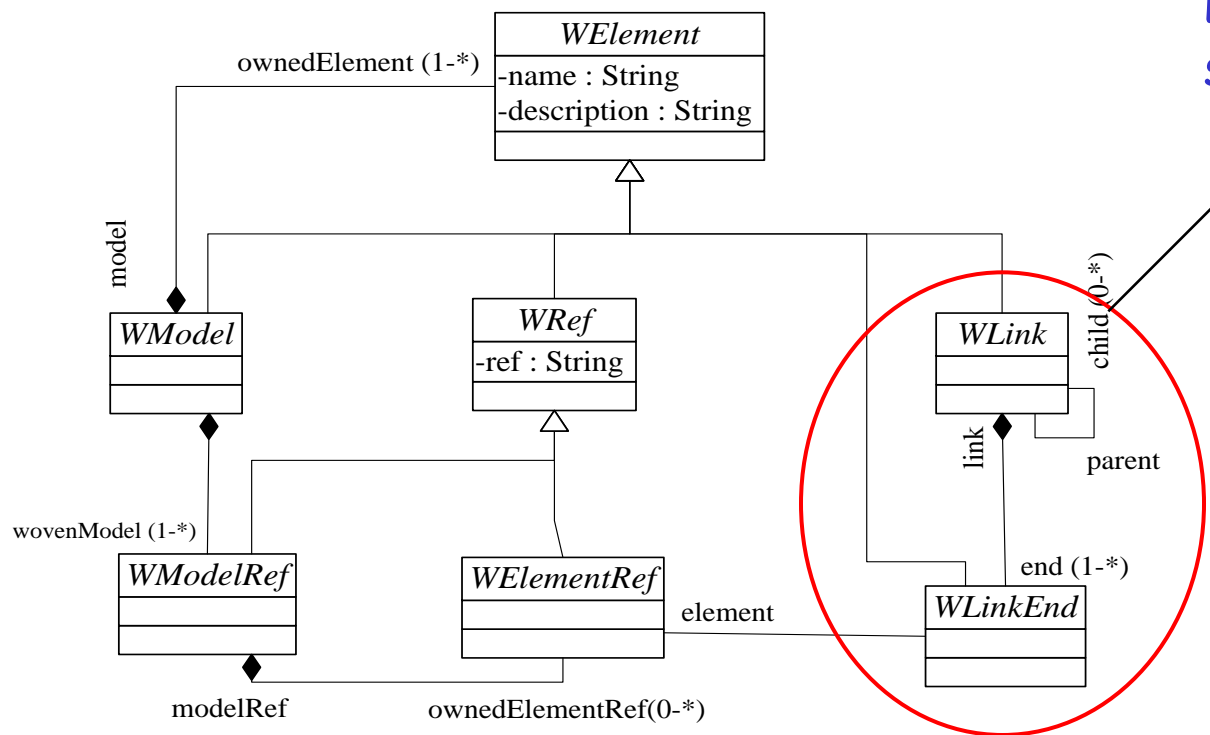


Mb



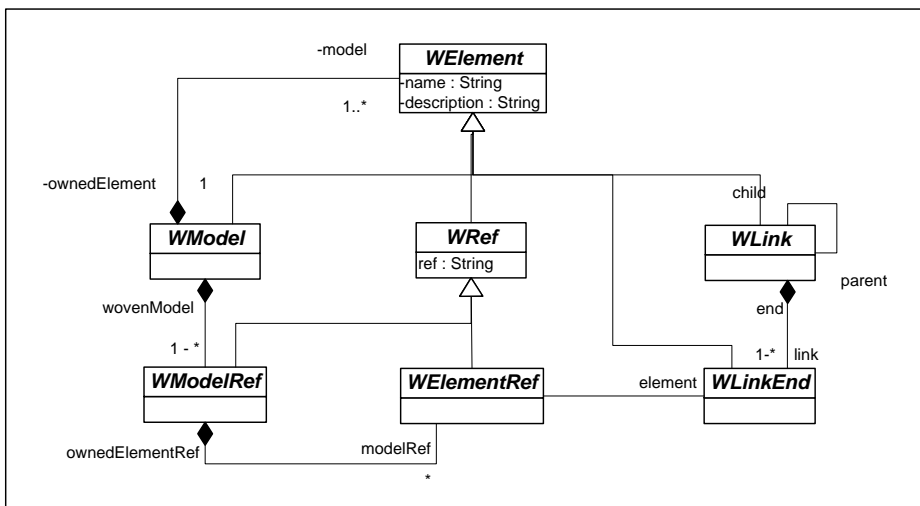
Core weaving metamodel

- Supports basic link management

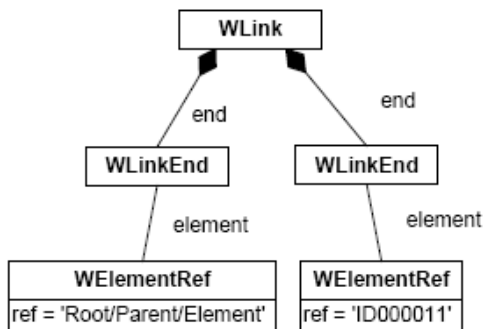
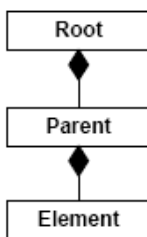


Linking semantics
(one link has N endpoints)

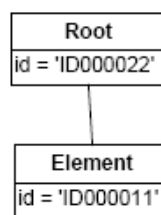
Core weaving metamodel



LeftMM



RightMM



```

package mwcore {
    abstract class WElement{
        attribute name : String;
        attribute description : String;
        reference model : WModel oppositeOf ownedElement;
    }

    abstract class WModel extends WElement{
        reference ownedElement[*] ordered container : WElement oppositeOf model;
        reference wovenModel[1-*] container : WModelRef;
    }

    abstract class WRef extends WElement{
        attribute ref : String;
    }

    abstract class WModelRef extends WRef{
        reference ownedElementRef[0-*] container : WElementRef oppositeOf modelRef;
    }

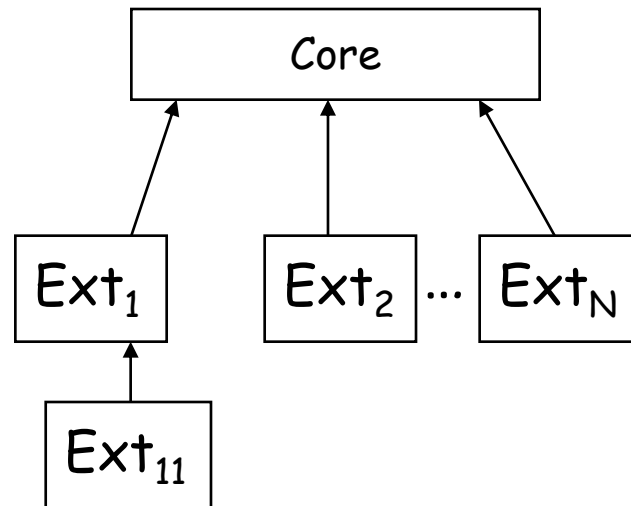
    abstract class WElementRef extends WRef{
        reference modelRef : WModelRef oppositeOf ownedElementRef;
    }

    abstract class WLink extends WElement{
        reference child[0-*] ordered container : WLink oppositeOf parent;
        reference parent : WLink oppositeOf child;
        reference end[1-*] container : WLinkEnd oppositeOf link;
    }

    abstract class WLinkEnd extends WElement{
        reference link : WLink oppositeOf end;
        reference element : WElementRef;
    }
}
    
```

Weaving metamodel extensions

- Core is extended with different kinds of relationships
 - Merging
(merge, override, inherits)
 - Interoperability
(translate, concatenate)
 - Traceability
(A generates B)
 - AOM
(A executesAfter B)



Atlas Model Weaver plug-in (AMW)

- Adapts to any weaving metamodel extension
 - The user interface is automatic generated according to the metamodel extensions
 - Reflective API of EMF (Eclipse Modeling Framework)
- A set of extension points is defined to enable to customize the standard user interface
 - Extension points to the panels, to the model elements, and to execute model transformations in ATL (Atlas Transformation Language)
 - New interfaces can be easily developed
- Available as a GMT component

AMW user interface

Plugged panels

Adaptive interface

Identification mechanism

Property	Value
Description	
Model	
Model Ref	
Model Ref	<<mantisModel>> Model Ref mantisModel
Name	ref_summary
Ref	EAttribute27
Misc	
Child	

Use cases available

- Traceability
- Transformation production
 - Tool interoperability, data integration
- Matching
- Metamodel comparison
- Model alignment

→ From now on, we present different use cases

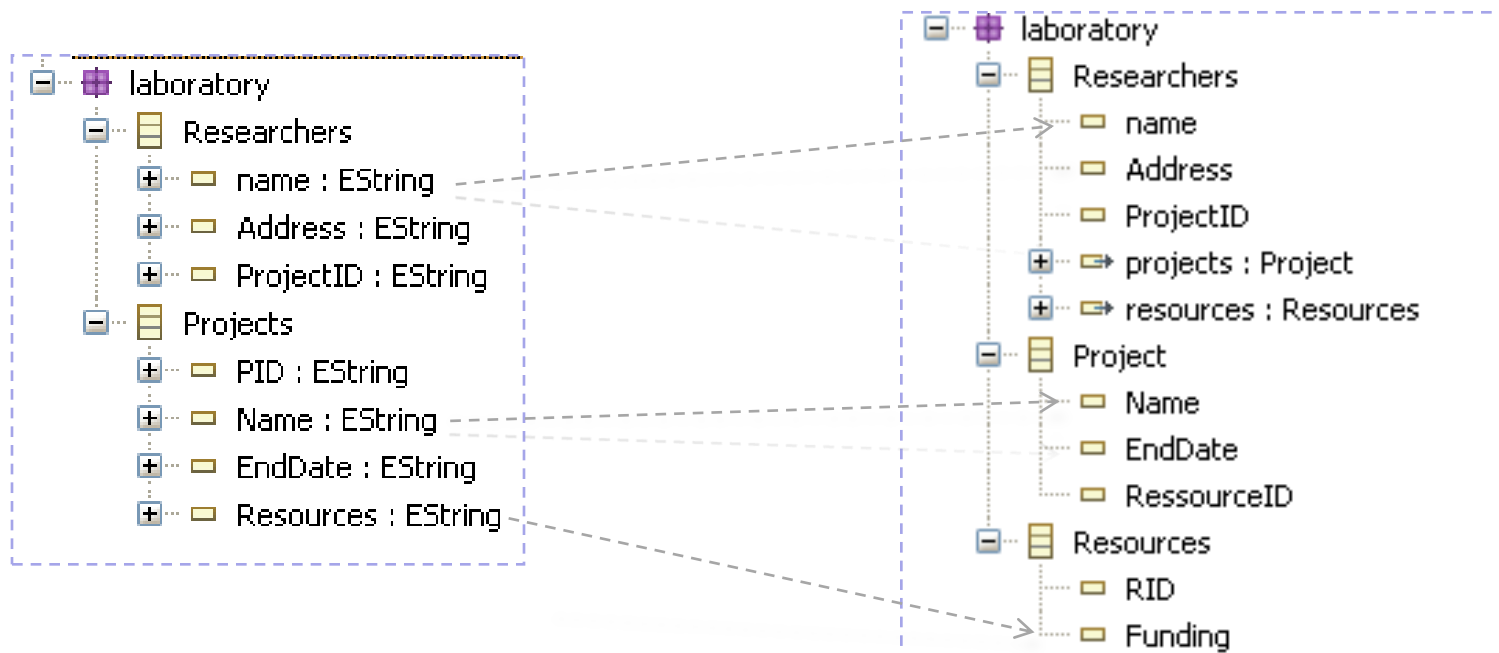
Outline

- Overview of model weaving
- Use cases (demo)
 - Simple AMW
 - Transformation production with AMW
 - Model matching
 - Model Annotation

Example & Demo

Relational database schema
to a XML schema

Relational database schema to a XML schema



The screenshot shows the Eclipse Modeling Project (EMF) interface with three views:

- leftModel:** Shows the EPackage 'laboratory' containing EClasses 'Researchers' and 'Projects'. 'Researchers' has attributes 'name', 'Address', and 'ProjectID'. 'Projects' has attributes 'PID', 'Name', 'EndDate', and 'Resources'.
- Weaving model:** Shows the generated XML schema for the 'laboratory' package. It includes a 'Model Researcher - Project' element with three 'ownedElement' children: 'Link Researcher Name', 'Link Researcher Address', and 'Link Project Name'. Each link element has left and right 'Link End' attributes. There is also a 'Link Project Fundings' element with left and right 'Link End' attributes.
- rightModel:** Shows the EPackage 'laboratory' with EClasses 'Researchers', 'Project', and 'Resources'. 'Researchers' has attributes 'name', 'Address', and 'ProjectID'. 'Project' has attributes 'Name', 'EndDate', and 'RessourceID'. 'Resources' has attributes 'RID' and 'Funding'.

Weaver Extension Base Metamodel

```

package mw_base_ext {

class Model extends WModel {
  -- @subsets wovenModel
  reference leftModel container : WModelRef;
  -- @subsets wovenModel
  reference rightModel container : WModelRef;
  --reference others[*] container subsets
  wovenModel : WModelRef;
}

-----

-- @wmodelRefType ModelRef
class ElementRef extends WElementRef {}
-- @welementRefType ElementRef
class ModelRef extends WModelRef {}
-- @welementRefType ElementRefXMI
class ModelRefXMI extends WModelRef {}
-- @wmodelRefType ModelRef
class ElementRefXMI extends WElementRef {}

-----

class Association extends WAssociation {}
class AssociationEnd extends WAssociationEnd {}

-----

class Link extends WLink {
  -- @subsets end
  reference left container : WLinkEnd;
  -- @subsets end
  reference right container : WLinkEnd;
}

class LinkEnd extends WLinkEnd {}

}

```

The screenshot displays the Eclipse IDE interface. The top toolbar shows icons for 'Weaving model', 'rightModel', and other actions. The main workspace is divided into two panes:

- Top Pane:** Shows a project named 'Model Researcher - Project'. It contains several 'ownedElement' links:
 - Link Researcher Name (with left and right end names)
 - Link Researcher Address (with left and right end addresses)
 - Link Project Name (with left and right end names)
 - Link Project Fundings (with left and right end resources and funding)
- Bottom Pane:** Shows a project named 'platform:/resource/AMW.Example/Metamodels/mw_base_ext.ecore'. It contains a package 'mwcore' with various classes and their relationships (e.g., WElement, WModel, WRef, WModelRef, WElementRef, WAssociation, WAssociationEnd, WLink, WLinkEnd). Below 'mwcore' is the 'mw_base_ext' package containing the classes defined in the code block on the left.

A context menu is open over the 'mw_base_ext' package, listing actions such as 'Run As', 'Debug As', 'Team', 'Compare With', 'Replace With', 'Undo', 'Redo', 'Cut', 'Copy', 'Paste', 'Delete', 'Save weaving metamodel in Ecore format', 'Copy name to container', 'Save weaving metamodel in KM3 format', and 'Show property view'. Dashed arrows point from the 'Save weaving metamodel in Ecore format' and 'Save weaving metamodel in KM3 format' options to the 'mw_base_ext' package in the bottom pane.

\\ECLIPSE\plugins\org.eclipse.weaver.extension.base_2.0.0

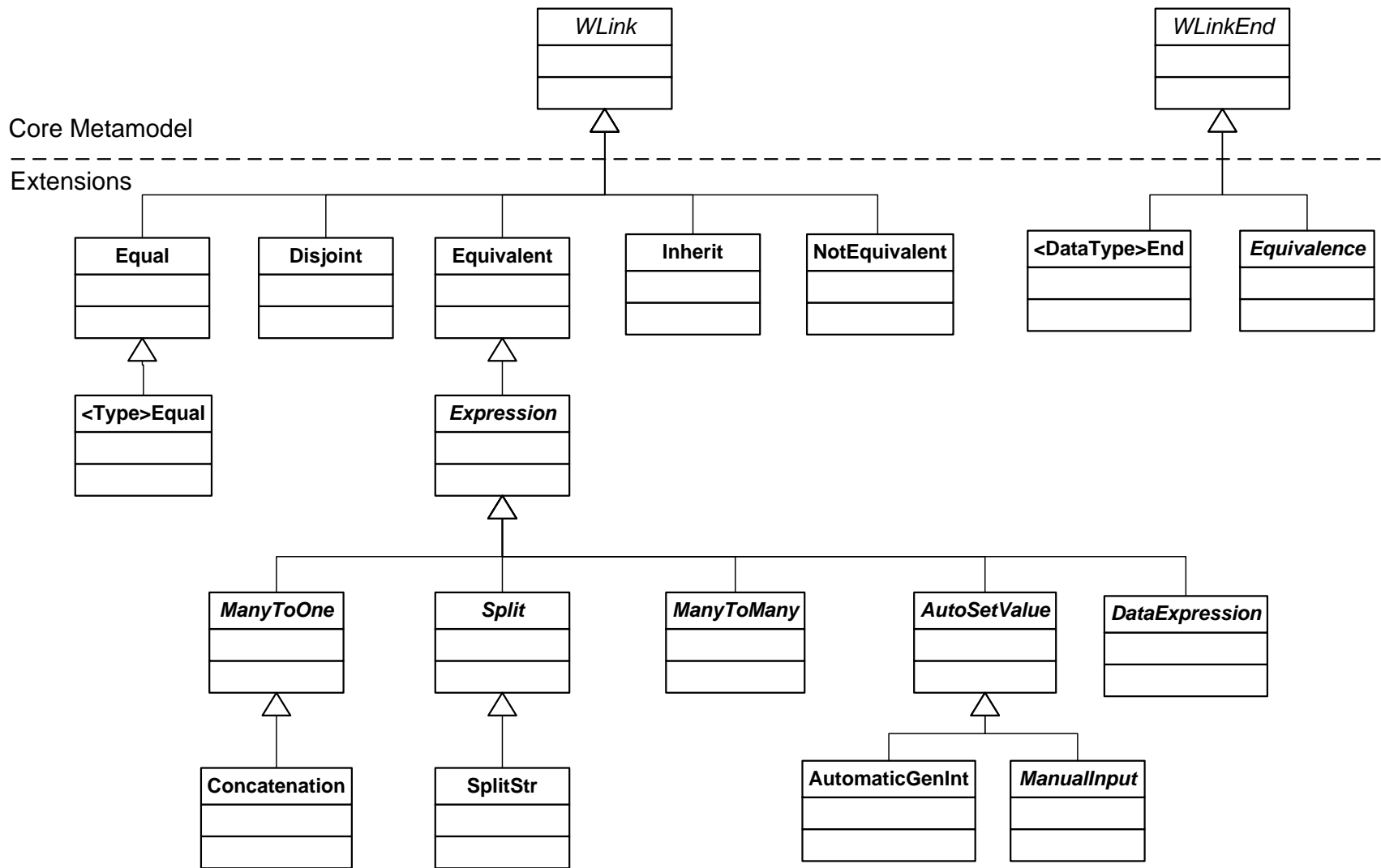
Outline

- Overview of model weaving
- Use cases (demo)
 - Simple AMW
 - Transformation production with AMW
 - Model matching
 - Model Annotation

Generic method

- Typical situation
 - Weaving model between 2 metamodels (source and target)
 - Transformation between source and target terminal models
- Based on 3 observations
 - Transformations have frequently-used expressions (e.g., equality, concatenation)
 - Metamodel has link types and link endpoints
 - Transformation languages are similar
- Pattern for generating transformations
 - TransfGen : weaving model-> transformation model

TransfGen: input metamodel extension



TransfGen: output metamodel

Transformation metamodel (extract)

```

class Module {
  reference rules [1-*] container : Rule;
}
class Rule {
  attribute name : DataType;
  reference input container: InputElement;
  reference output[*] container:
    OutputElement;
}
class InputElement {
  reference element : ReferredElement;
  reference condition [0-1] : Expression;
}
class OutputElement {
  reference element : ReferredElement;
  reference bindings [*] : Binding;
}
class Binding {
  reference target : ReferredElement;
  reference source : Expression;
}

```

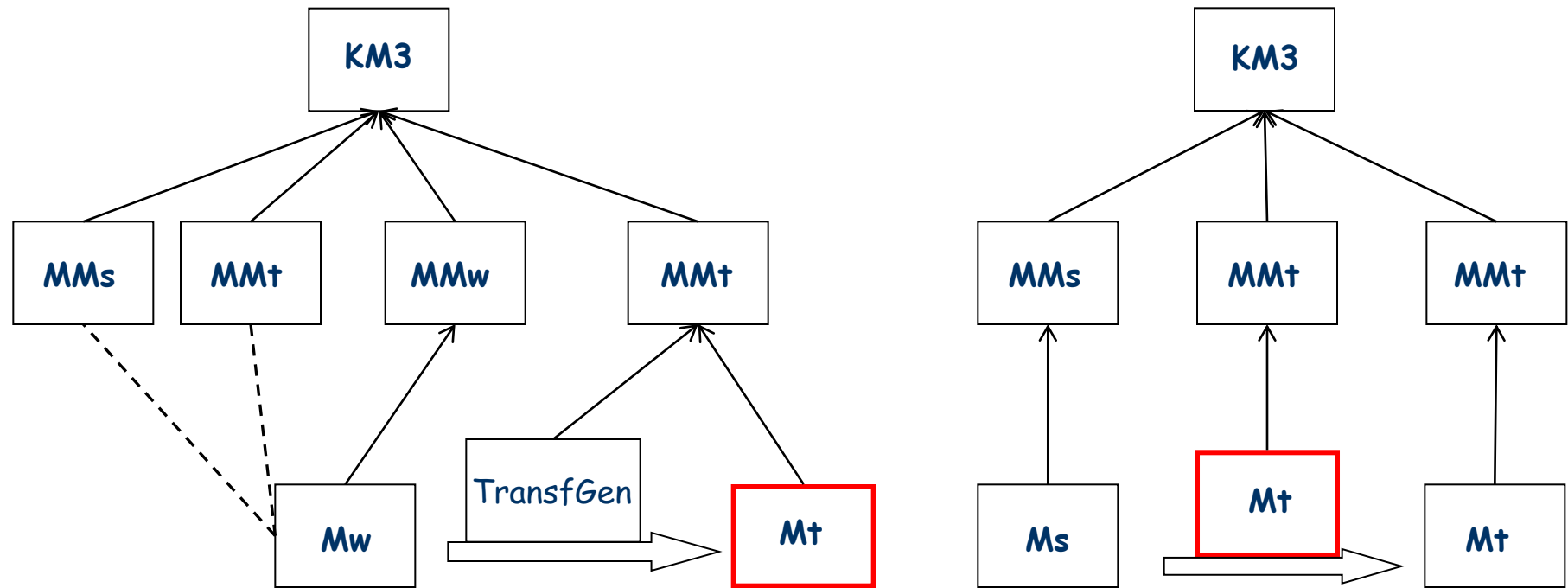
Transformation model

```

rule <name> {
  from
    input (condition)
  to
    output1 (
      target1 <- source1
      target2 <- source2
      targetN <- sourceN
    ),
    outputN ...
}
rule <name2> ...

```

General view



- Two major points
 - Transformations are models
 - Different transformation metamodels (e.g., ATL or XSLT)

Demo

Graphical development of an
ATL transformation

Families2Persons

Outline

- Overview of model weaving
- Use cases (demo)
 - Simple AMW
 - Transformation production with AMW
 - **Model matching**
 - Model Annotation

Matching heuristics

- String similarity
 - Date \leftrightarrow BirthDate
- Dictionaries
 - Car \leftrightarrow Vehicle
- Structural relations
 - Class.name \leftrightarrow Table.name
 - Class \leftrightarrow Table
- Different problems
 - How to express this heuristics ?
 - How to support different extensions ?

Matching transformations

- **Definition.** A **matching transformation** is a domain-specific transformation that takes two or more models as input, and that transform them into a new weaving model
 - $T_{\text{MATCHING}} \rightarrow \text{model} \times \dots \times \text{model} \rightarrow \text{weaving model}$

rule CreateLink {

from

aLeft : MMA!Class, aRight : MMb!Attribute (guard)

to

aLink : MMw!Equivalentent (

left <- getID(aLeft),

right <- getID(aRight),

similarity <- aLeft.calcSim(aRight)

)

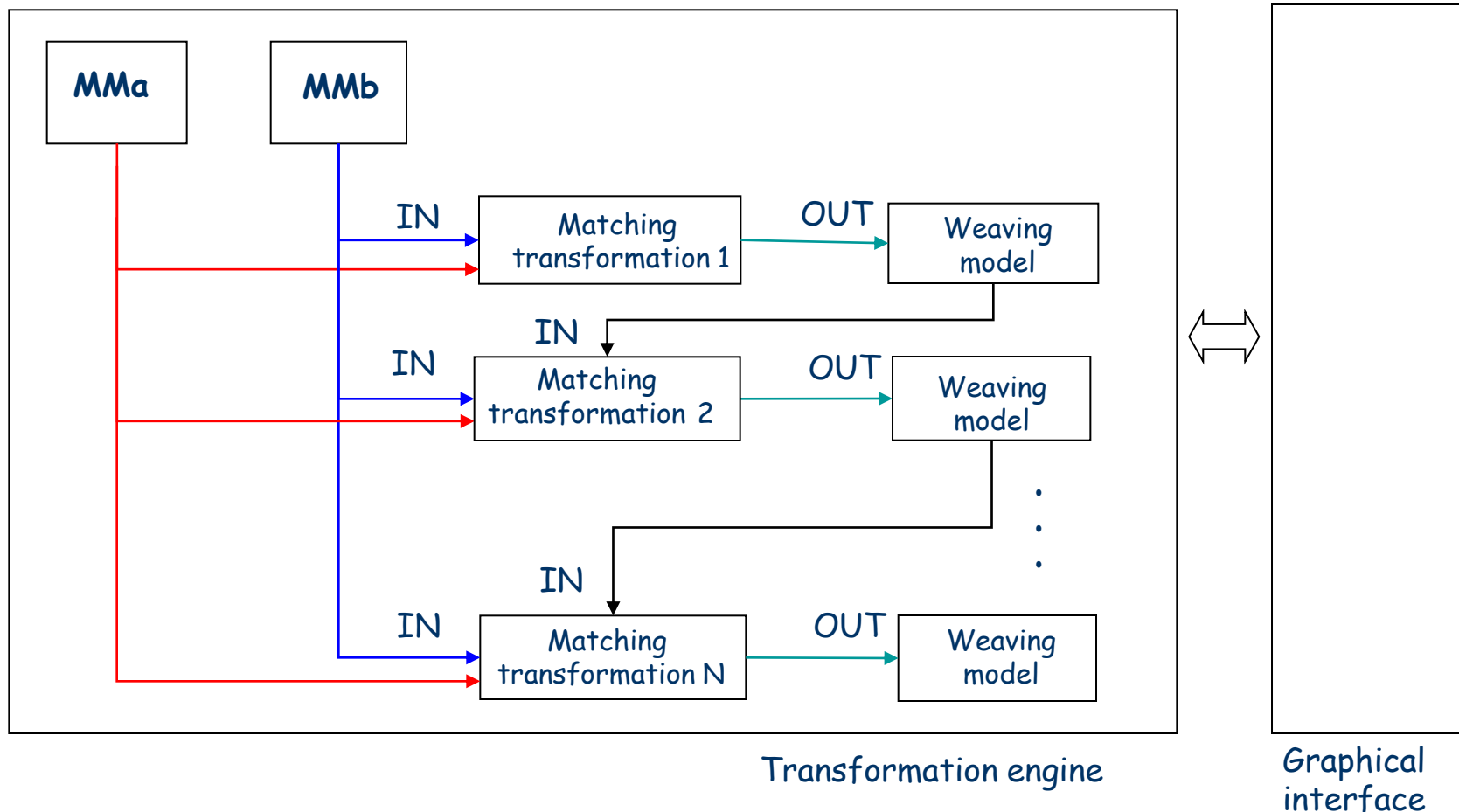
}

Execution condition

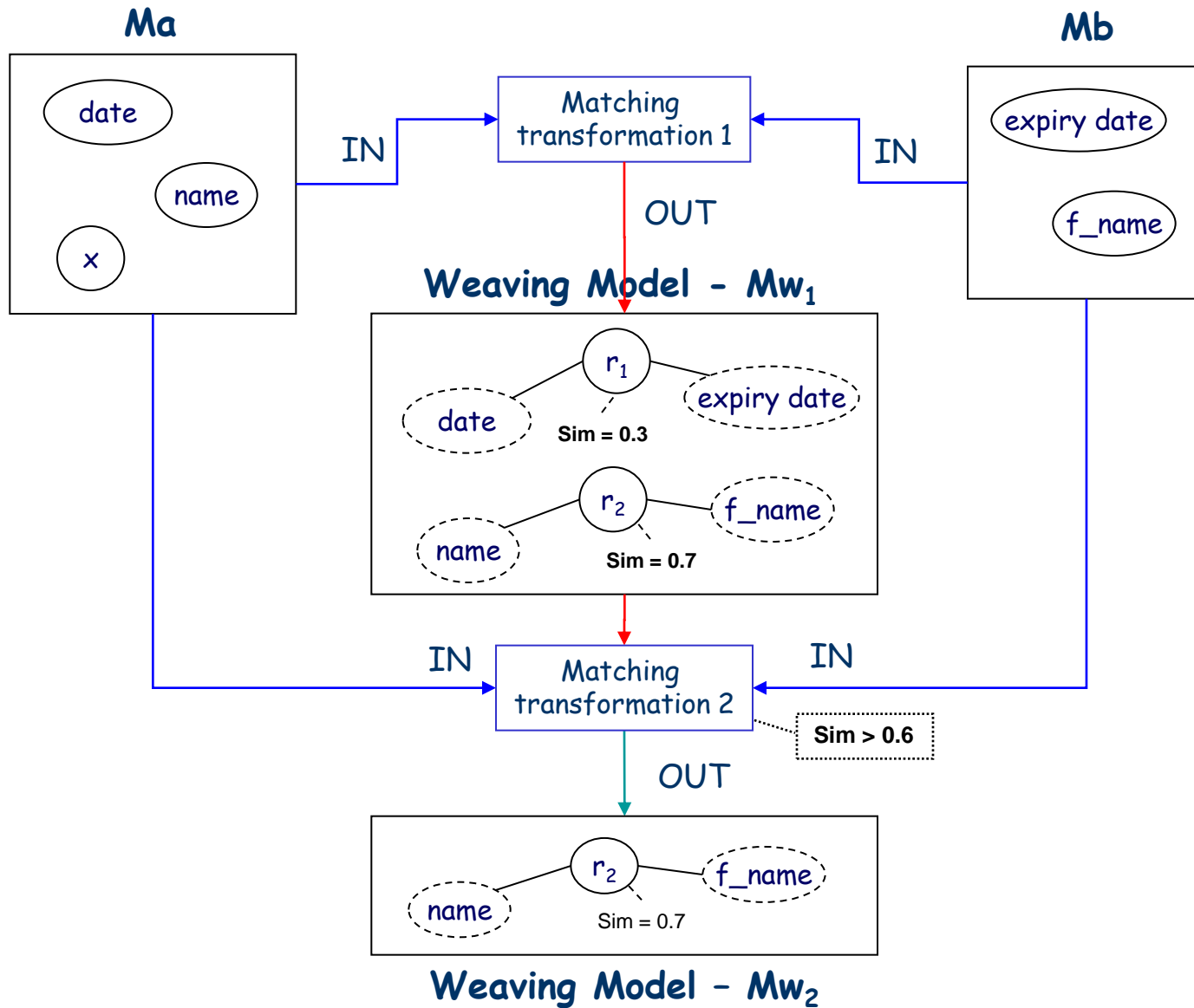
Similarity computation [0-1]

Cumulative matching

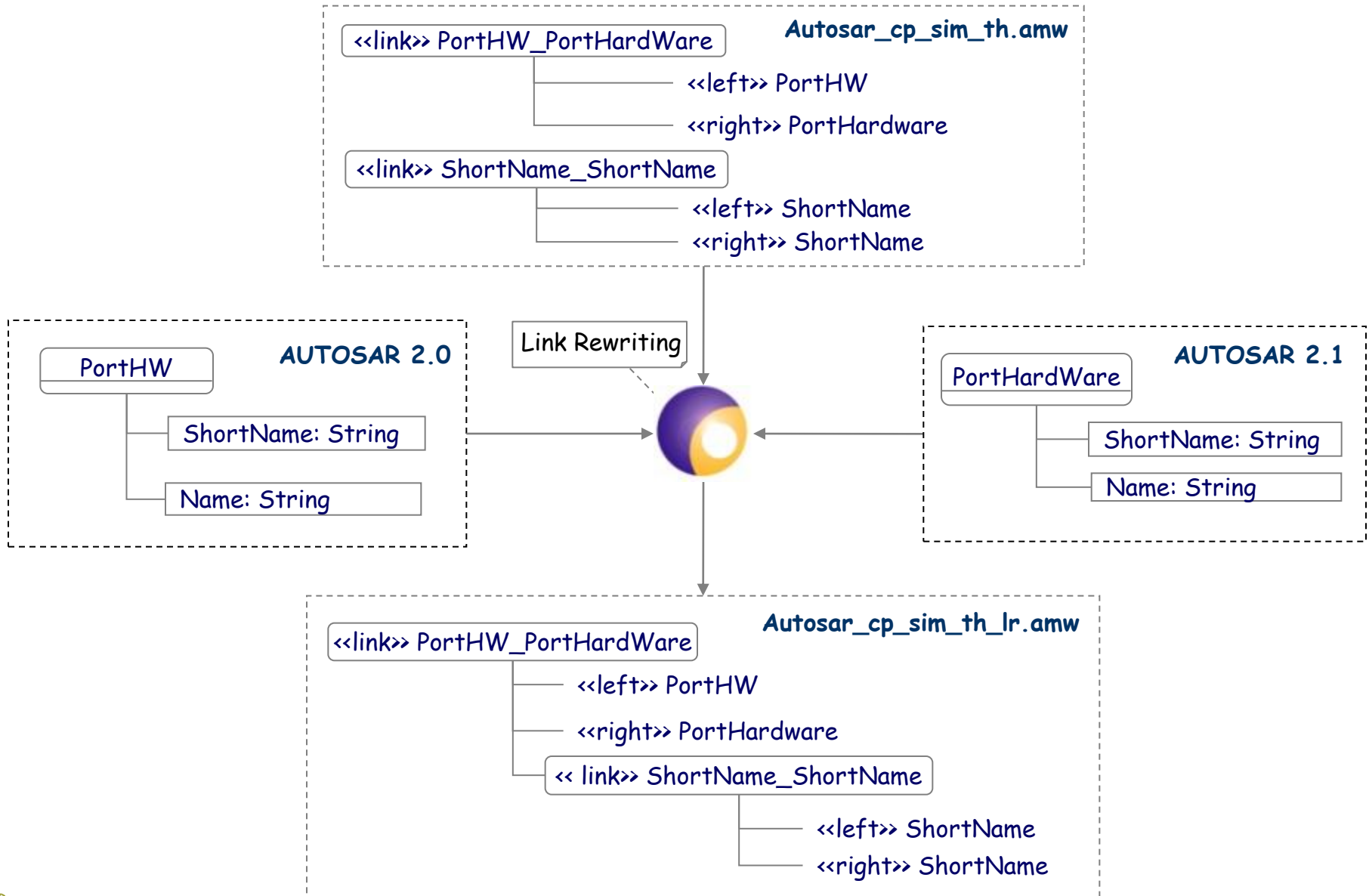
- Transformations executed sequentially



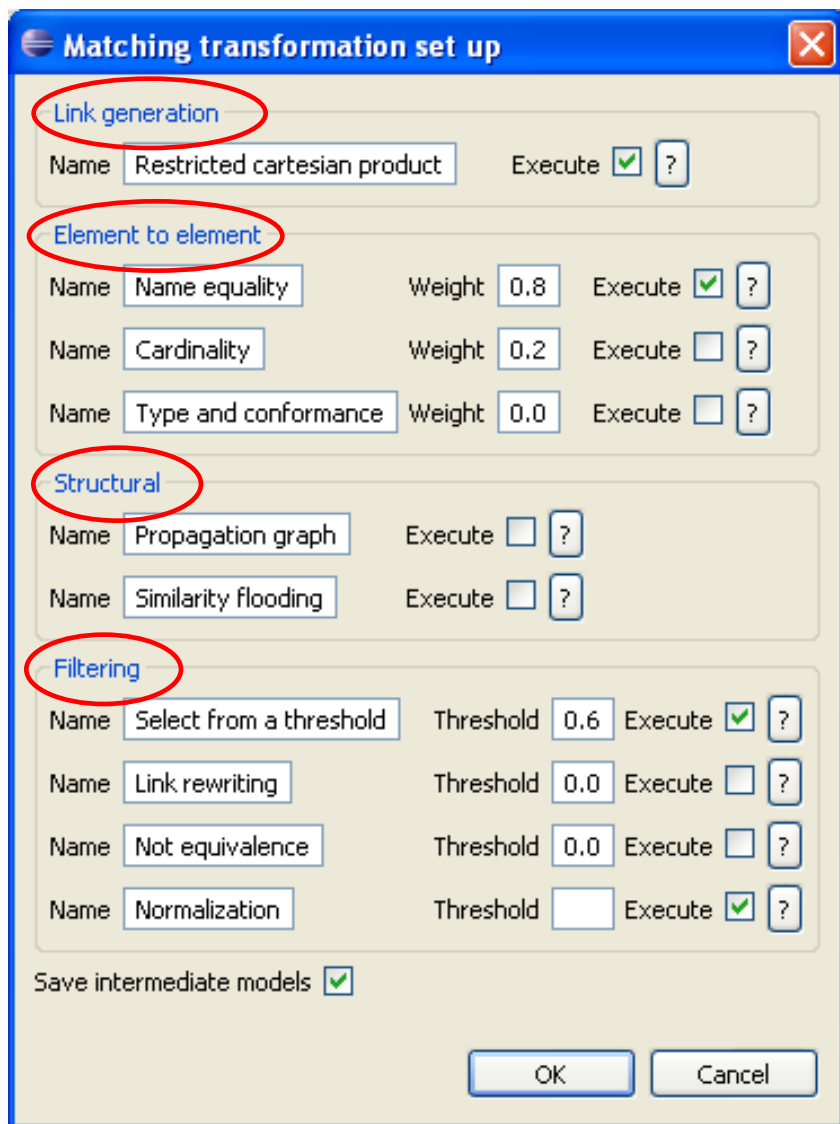
Cumulative matching: similarity + threshold



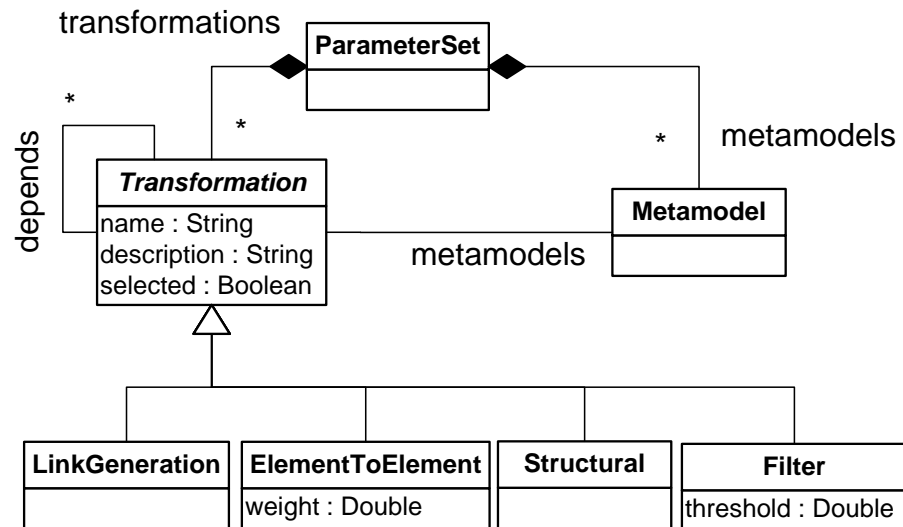
Link Rewriting Output



Configuration model



Configuration metamodel



Example

Using ATL transformations to generate
AMW weaving models
(Cumulative Matching)

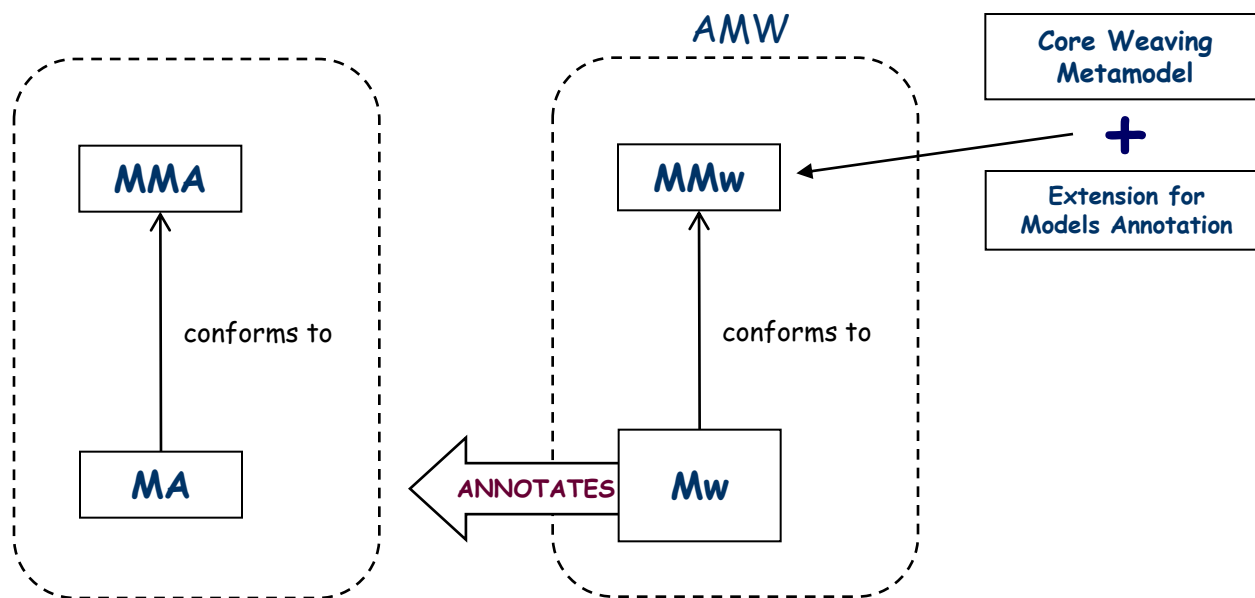
Versioning Scade Metamodel

Outline

- Overview of model weaving
- Use cases (demo)
 - Simple AMW
 - Transformation production with AMW
 - Model matching
 - **Model Annotation**

Model Annotation with AMW

- Collect extra information not defined in the metamodel
- Avoid polluting the model
- In a real MDE context \rightarrow data = model



Annotation Metamodel

```

package mmw_annotation {
  class AnnotationModel extends WModel {
    -- @subsets ownedElement
    reference contents[*] ordered container : Annotation;
    -- @subsets wovenModel
    reference referencedModel container : AnnotatedModel;
  }
  -- @welementRefType AnnotatedModelElementRef
  class AnnotatedModel extends WModelRef {}

  -- @wmodelRefType AnnotatedModel
  class AnnotatedModelElementRef extends WElementRef {}

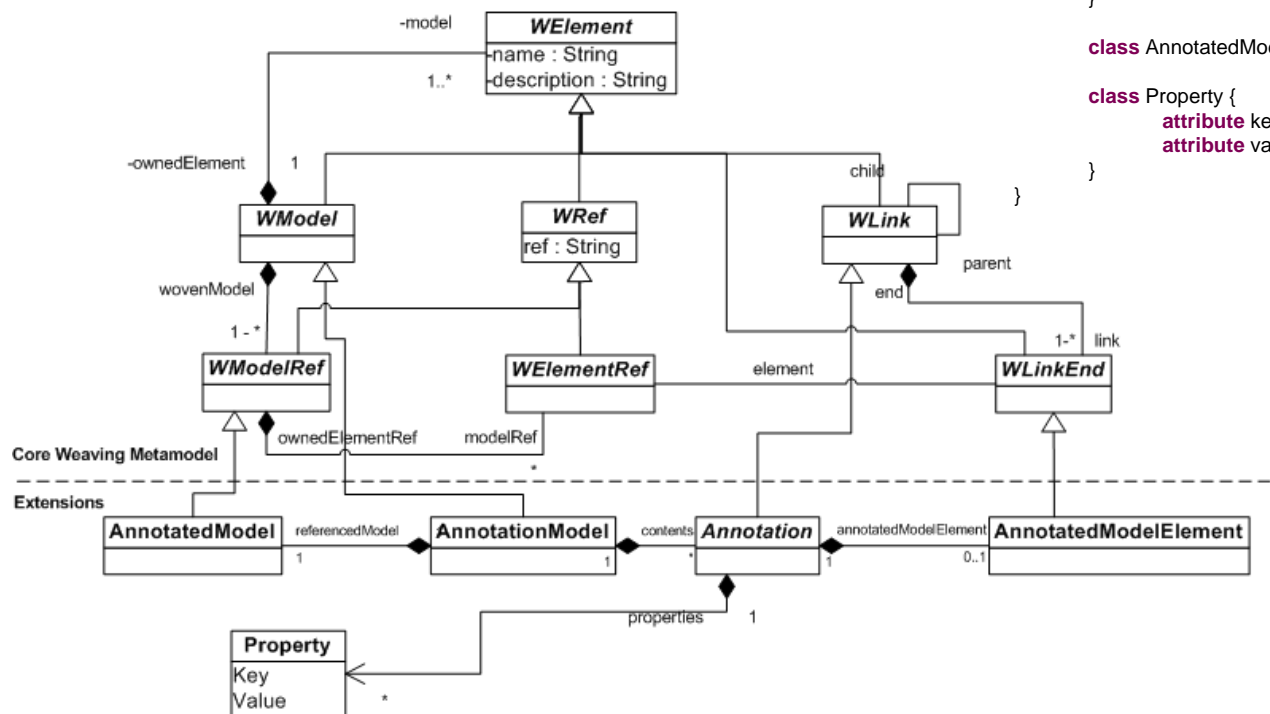
  class Annotation extends WLink {
    reference properties[*] ordered container : Property;
    -- @subsets end
    reference annotatedModelElement container : AnnotatedModelElement;
  }
}
    
```

```

class AnnotatedModelElement extends WLinkEnd {}
    
```

```

class Property {
  attribute key : String;
  attribute value : String;
}
    
```



Demo

Annotating JAVA models

Outline

- Overview of model weaving
- Use cases (demo)
 - ATL Transformations production
 - Model matching
 - **Model Annotation**
 - Using annotation models to drive ATL transformations

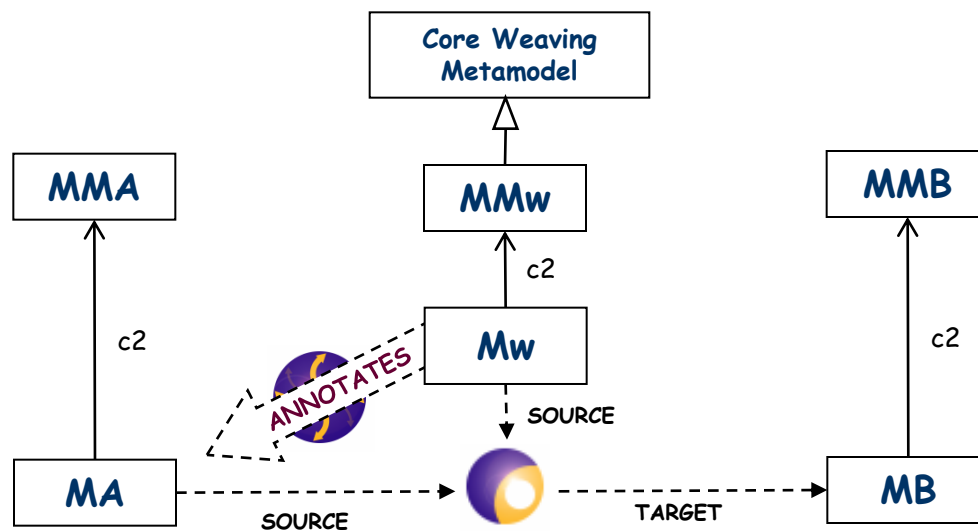
Example

Using Annotation models
to drive ATL
transformations

UseCase2Activity

Annotation models for ATL transformations

- Gap between given metamodels
- A generic ATL transformation is not enough
- An annotation model parameterize the transformation



Rules specifications

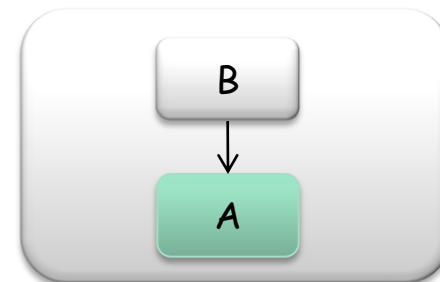
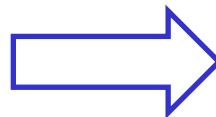
- UseCase2Node: a **Node** is created for each **UseCase**
 - The **name** of the **Node** is the **name** of the **UseCase**

```
rule UseCase2Node {  
  from  
    u : UseCase!UseCase  
  
  to  
    n : Activity!Node (  
      name <- u.name  
    )  
}
```

Rules specifications

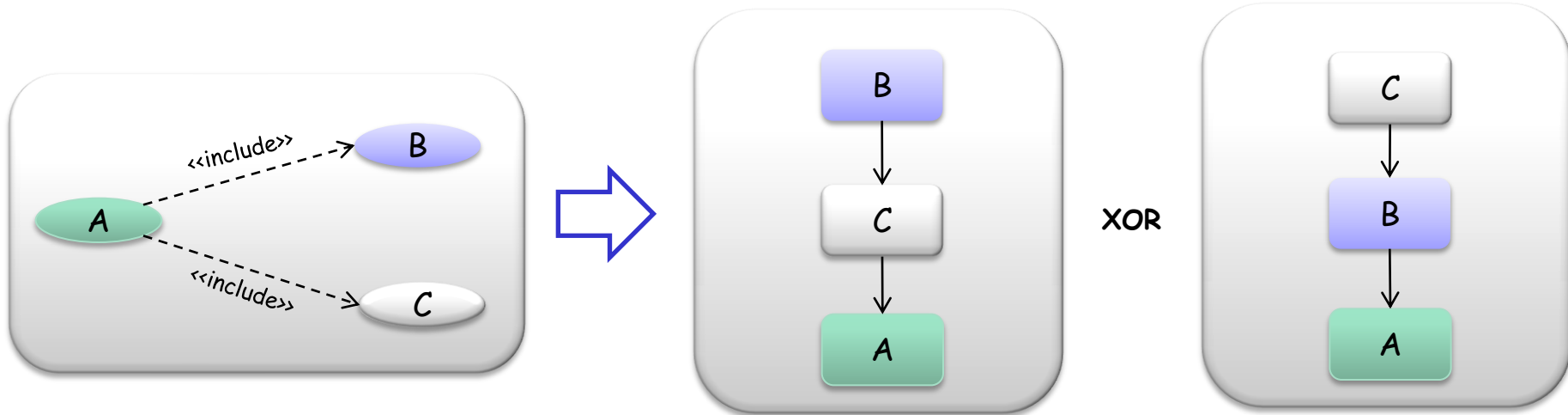
- Include2Edge: an **Edge** maps every **Include** association
 - The **name** of the **Edge** is the **name** of the **Include**
 - The **source** of the **Edge** is the Node that maps the **addition**
 - The **target** of the **Edge** is the Node that maps the **including case**

```
rule Include2Edge {
  from
    i : UseCase!Include
  to
    e : Activity!Edge (
      name <- i.name,
      source <- i.addition,
      target <- i.includingCase
    )
}
```



The need for annotation models

- And what if???



Configuring the ATL program

- The source model is still the UseCase diagram
- The target model is still the Activity diagram
- An extra source model is added: the annotation model

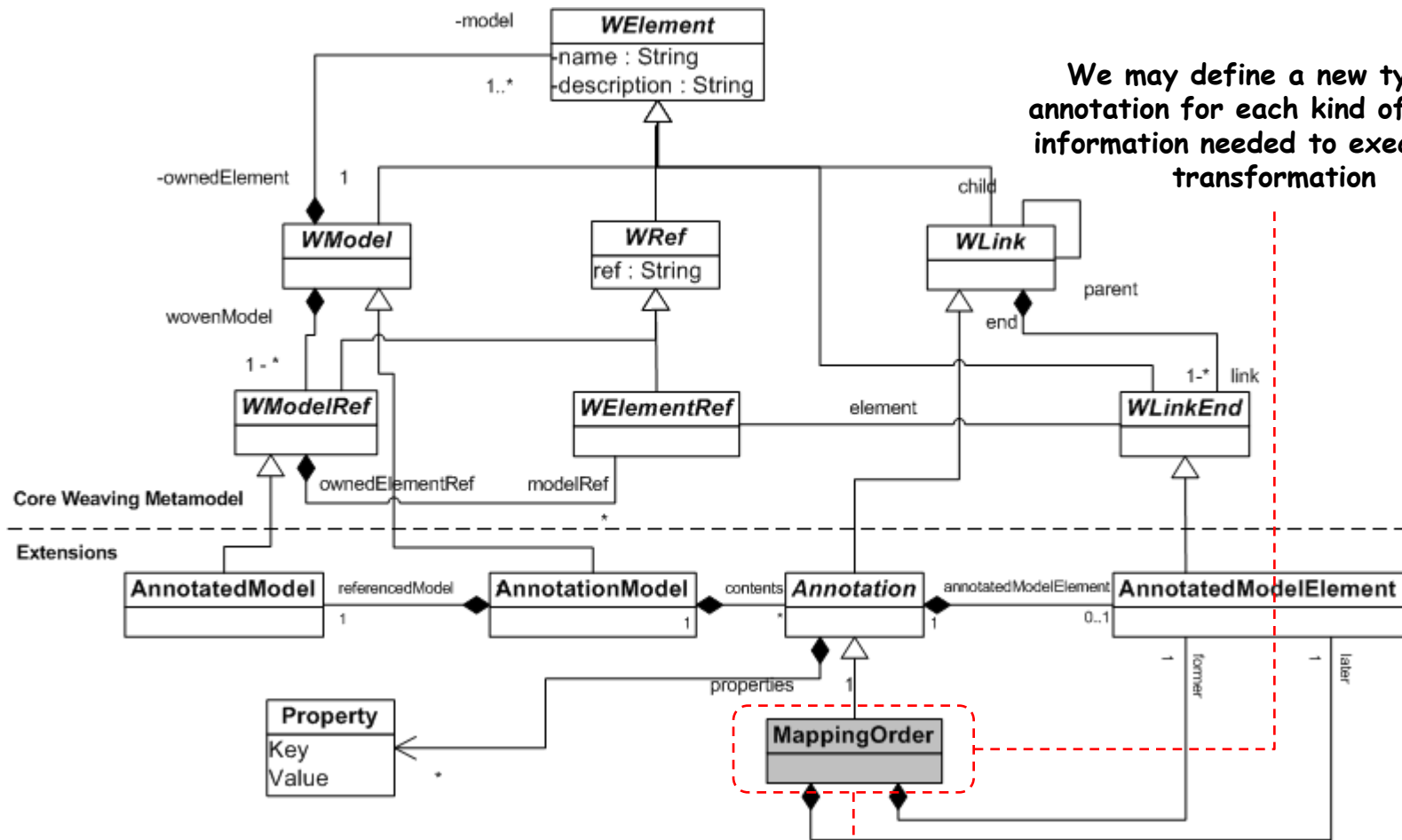
→ ATL declaration of the *parameterized* transformation:

module UseCase2Activity;

create OUT : Activity **from** amw : AMW(*) IN : UseCase;

(*) We have to create the weaving metamodel from the KM3 extension → AMW does it for free ☺

Extended Annotation Metamodel



```
class MappingOrder extends Annotation {
```

```
-- @subsets end
```

```
reference former container : AnnotatedModelElement;
```

```
-- @subsets end
```

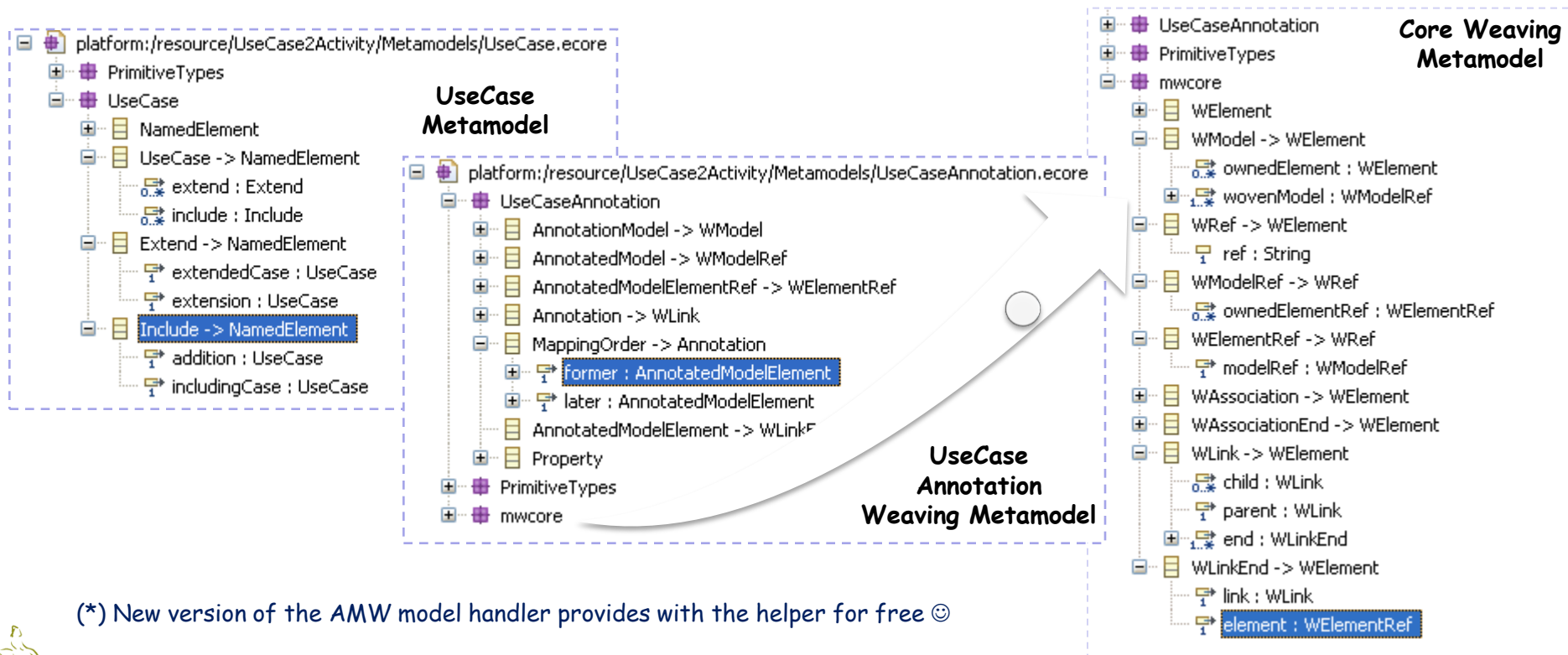
```
reference later container : AnnotatedModelElement;
```

```
}
```

Navigating the annotation model

- The *getReferredElement* takes as input a *WLinkEnd* of a weaving model, returning the referred element of a woven model (*)

helper context AMW!WLinkEnd **def**: getReferredElement() : UseCase!NamedElement = UseCase!NamedElement.allInstances()->asSequence()-> select(aux | aux.__xmlID__ = self.element.ref)->first();



(*) New version of the AMW model handler provides with the helper for free ©

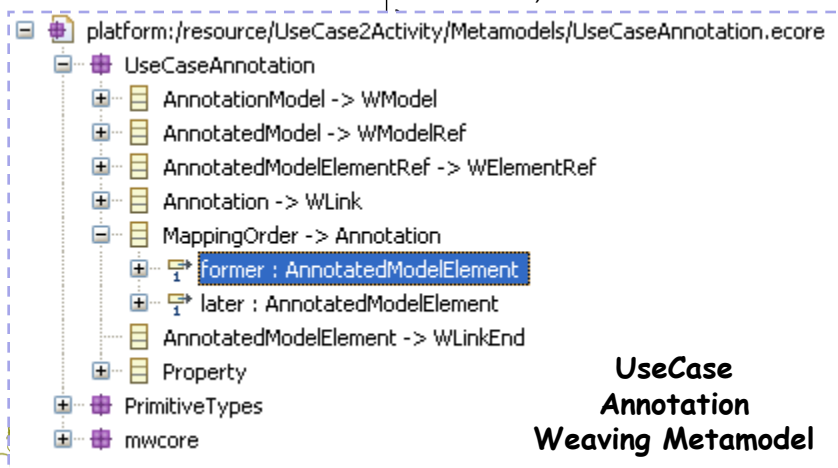
Using the annotation model

- A new rule uses the information that the annotation provides with

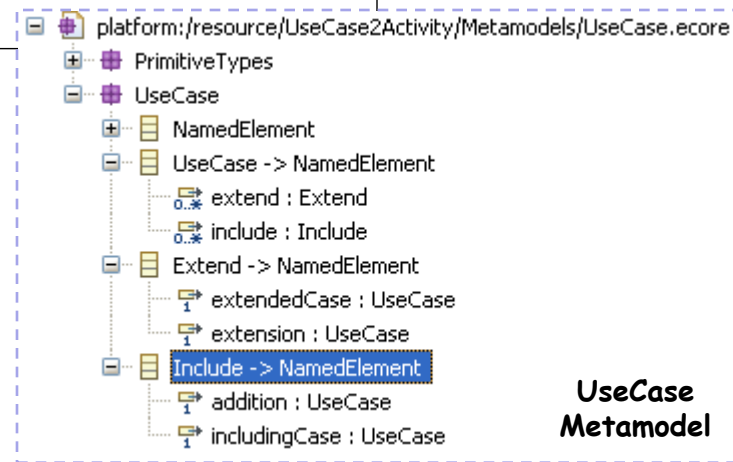
```

rule Annotation2Edges {
  from
    a : AMW!MappingOrder
  to
    e1 : Activity!Edge (
      name <- 'FROM ' + a.former.getReferredElement().addition.name + ' TO '
        + a.later.getReferredElement().addition.name,
      source <- a.former.getReferredElement().addition,
      target <- a.later.getReferredElement().addition
    ),
    e2 : Activity!Edge (
      name <- 'FROM ' + a.later.getReferredElement().addition.name + ' TO '
        + a.later.getReferredElement().includingCase.name,
      source <- a.later.getReferredElement().addition,
      target <- a.later.getReferredElement().includingCase
    )
}

```



**UseCase
Annotation
Weaving Metamodel**



**UseCase
Metamodel**

Demo

Using Annotation models
to drive ATL
transformations

UseCase2Activity

Example

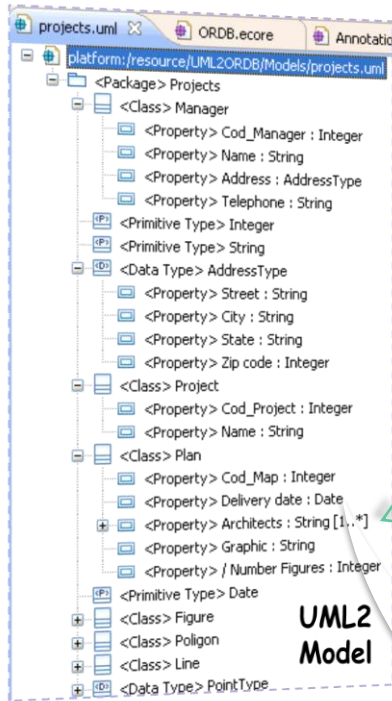
Using Annotation models
to drive ATL
transformations

UML2ORDB

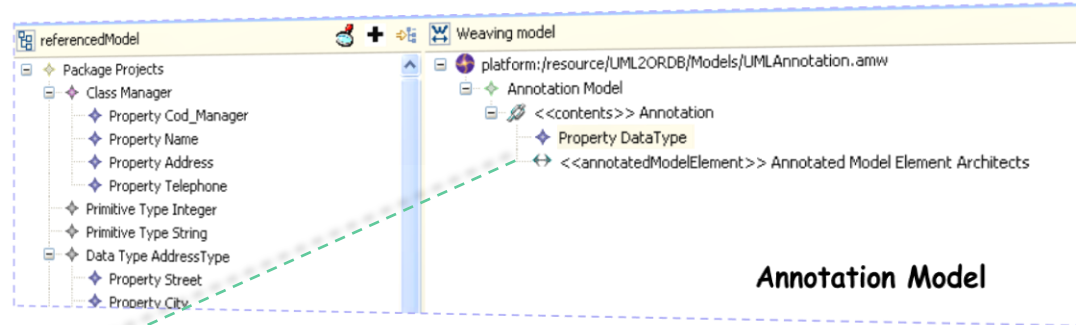
Annotation models for ORDB development

- Conceptual Data model to OR DB model is not immediate
- Some design decisions needed just before *run-time* 😊
- Simplistic How-to for one of such decisions

Annotation models for ORDB development



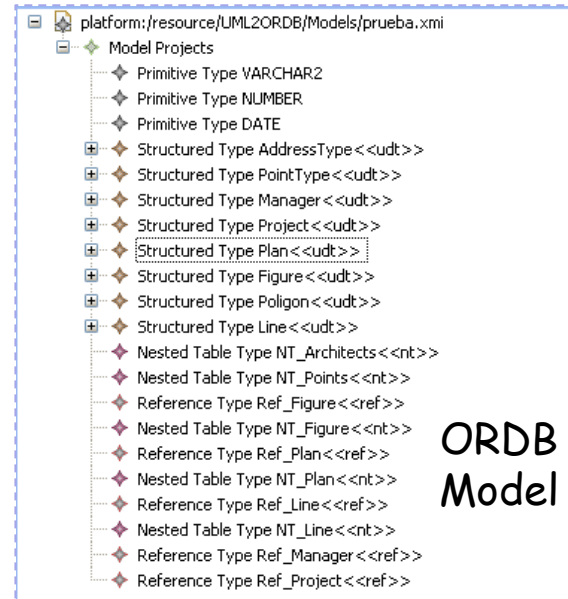
UML2 Model



Annotation Model

UML Multivalued properties should be mapped to NT or to VARRAY columns in Oracle??

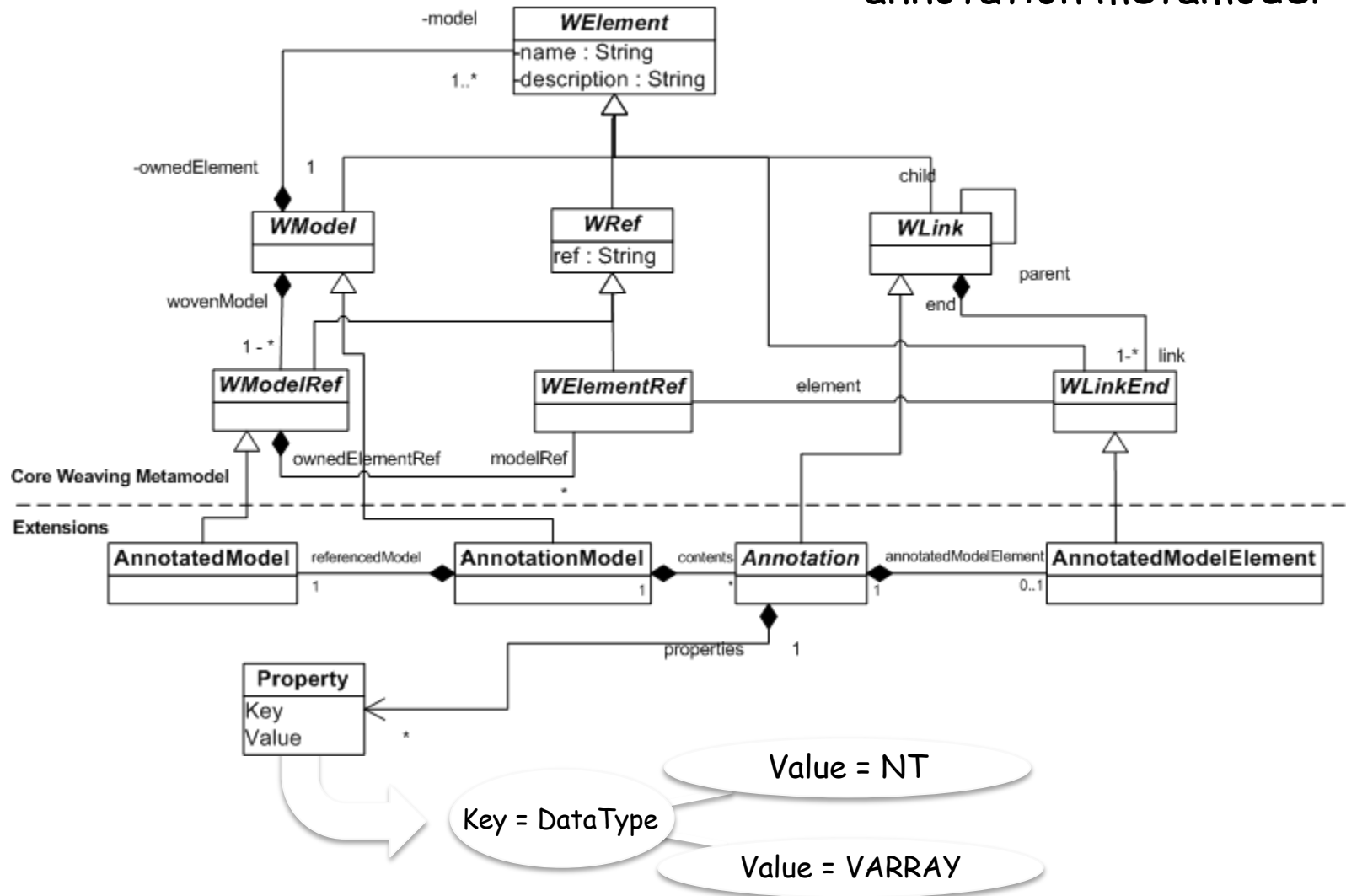
UML2ORDB



ORDB Model

Annotation Metamodel

We use the very same annotation metamodel

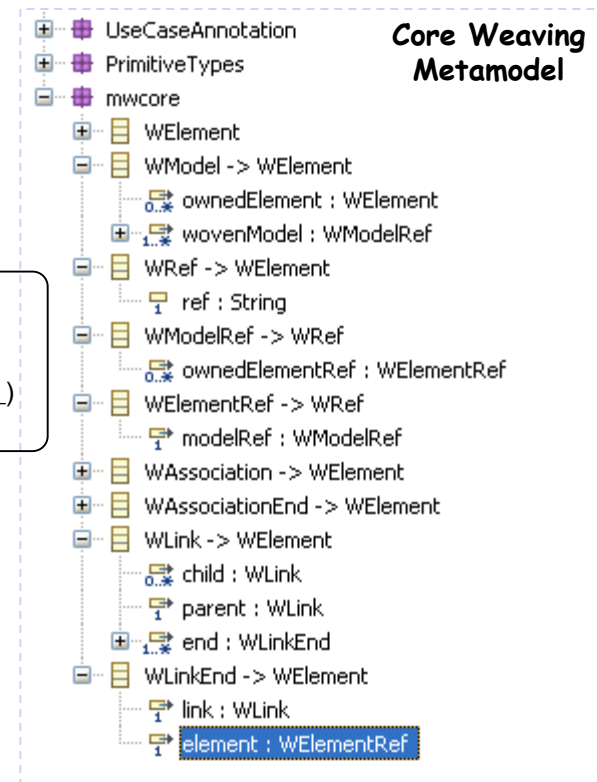


Using the Annotations

- Some helpers are coded to use the annotations

-- Given an object from the woven model, it returns the Link referring to that
 -- object in the weaving model

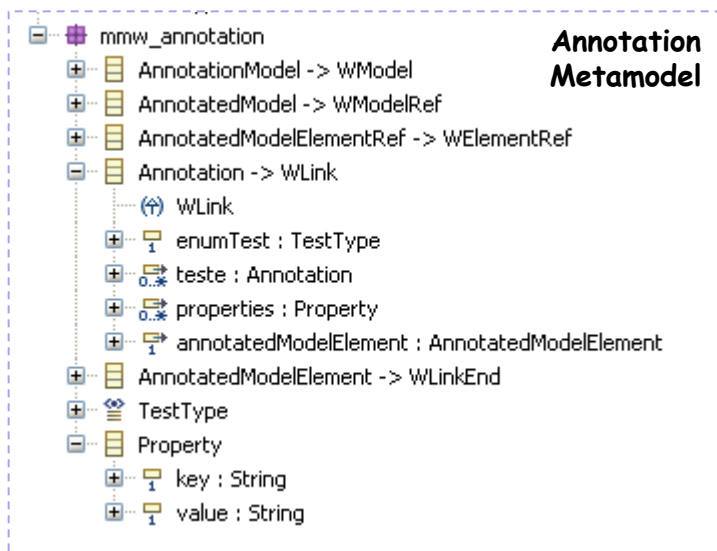
helper context UML!NamedElement **def:** getLink() : AMW!WLink =
 AMW!WLinkEnd.allInstances()->asSequence()->select(aux | aux.element.ref = self.__xmlID__)-
 ->first().refImmediateComposite();



Using the Annotations

- Some helpers are coded to use the annotations

```
-- Given an Annotation Link, it returns the value of its 'key' Property
helper context AMW!WLink def: getAnnotationValue(key: String) : String =
  self.properties->asSequence()->select(prop | prop.key = key)->first().value;
```



Using the Annotations

- Some helpers are coded to use the annotations

```
-- It says whether the 'self' multivalued property should be mapped to an NT or a VARRAY attribute
helper context UML!Property def: mapTo() : String =
  if self.getLink().oclIsUndefined() then
    'NT'
  else
    if self.getLink().getAnnotationValue('DataType') = 'NT' then
      'NT'
    else
      'VARRAY'
    endif
  endif;
```

```
-- Given an object from the woven model, it returns the Link referring to that
-- object in the weaving model (if existing)
helper context UML!NamedElement def: getLink() : AMW!WLink =
  AMW!WLinkEnd.allInstances()->asSequence()->select(aux | aux.element.ref = self.__xmlID__)->first().refImmediateComposite();
```

```
-- Given an Annotation Link, it returns the value of its 'key' Property
helper context AMW!WLink def: getAnnotationValue(key: String) : String =
  self.properties->asSequence()->select(prop | prop.key = key)->first().value;
```

Demo

Using Annotation models
to drive ATL
transformations

Summary

- We have presented here a the basics of the AMW tool as well as some Use Cases.
- We have focused on the use of weaving models as annotation models.
- It provides with a newly and complete model-driven way of *parameterizing* model transformations.
- This is not a recommendation, just some initial examples.
- For any further questions, see the documentation mentioned in the resource page (FAQ, Manual, Examples, etc.).

Links

- **AMW use cases**
 - <http://www.eclipse.org/gmt/amw/usecases/>
- **Atlas Group**
 - <http://www.sciences.univ-nantes.fr/lina/atlas/>
- **AMMA (Atlas Model Management Architecture)**
 - <http://www.sciences.univ-nantes.fr/lina/atl/>
- **AMW (Atlas Model Weaver)**
 - <http://www.eclipse.org/gmt/amw/>
- **ATL (Atlas Transformation Language)**
 - <http://www.eclipse.org/gmt/atl/>

End of part 2: Model Weaving with AMW

- Thanks
 - Questions?
 - Comments?