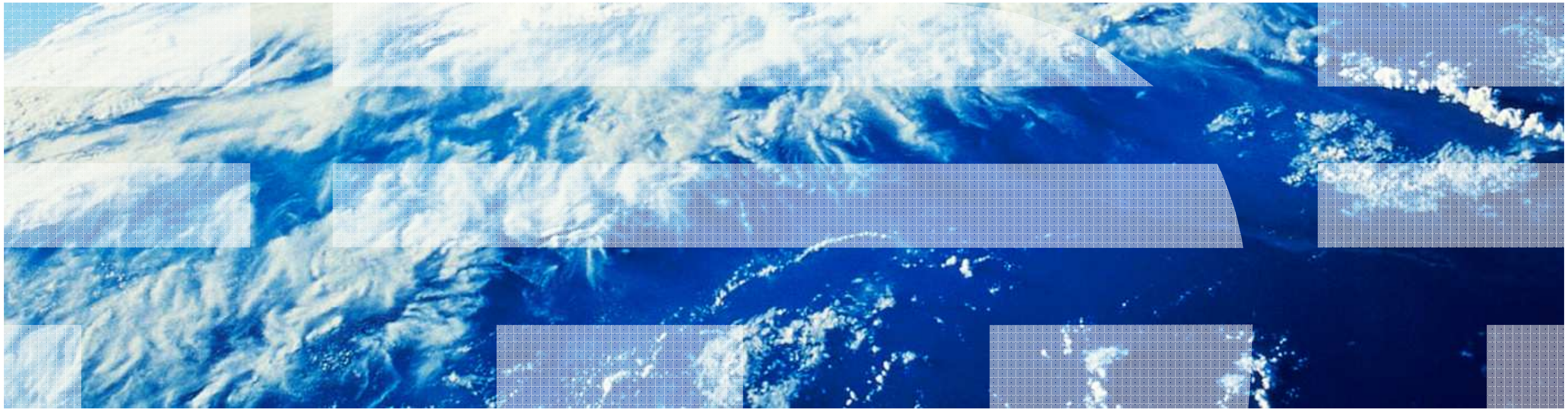
Marcos Didonet Del Fabro – Software Engineer – marcos.ddf@fr.ibm.com

29 March 2010 – MDE Diploma

# Model weaving
## Establishing links between model elements



© Copyright IBM France 2010

## Outline

**Model weaving : state of the art and concepts**

Practical work : schema mapping and traceability

Matching and transformation production

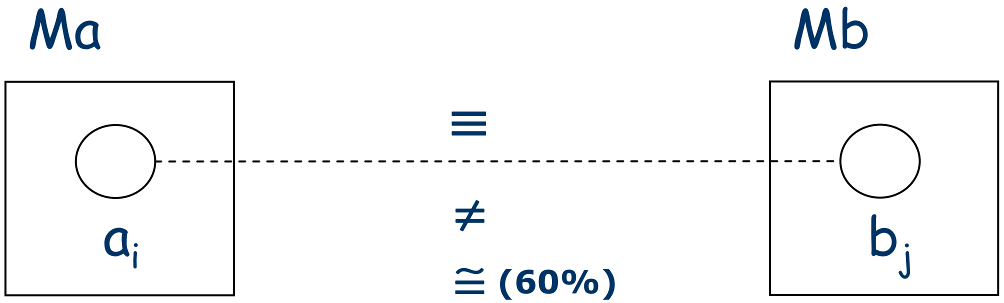
Practical work : matching and transformation production



© Copyright IBM France 2010

# Relationships between model elements

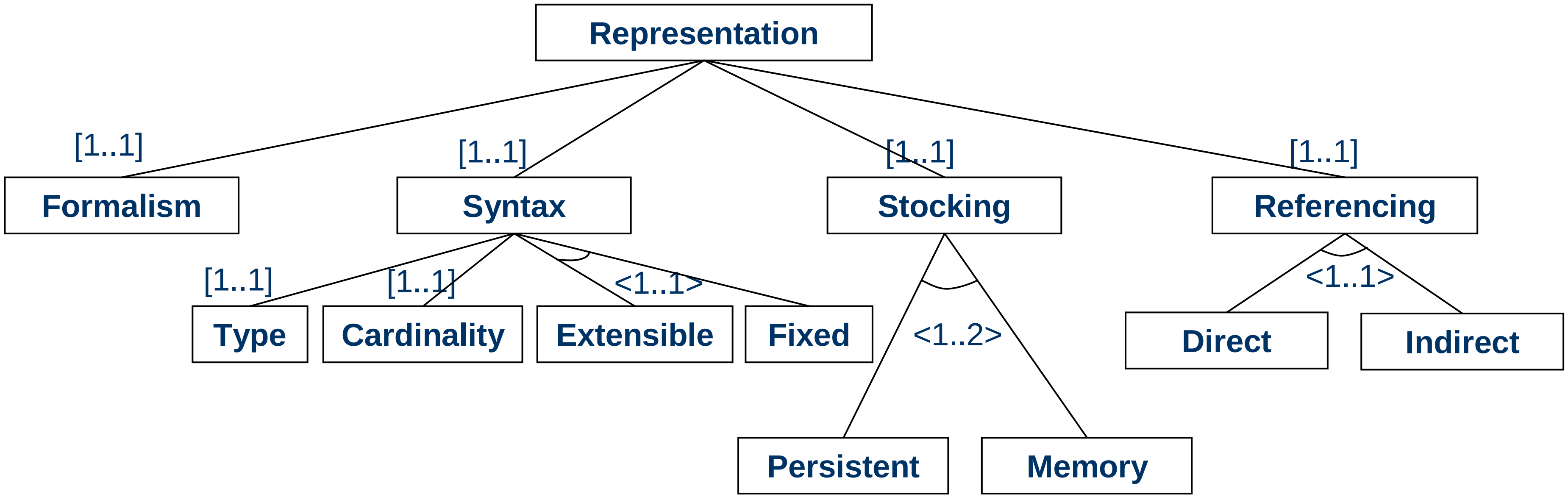
- Transformation are not always enough
  - Precise execution semantics




- If $a_i \equiv b_j$
  - How to express ?
  - How to compute?
  - How to generate Ma $\cap$ Mb or Ma $\cup$ Mb ?
- Three main aspects


© Copyright IBM France 2010

# First aspect: relationship representation

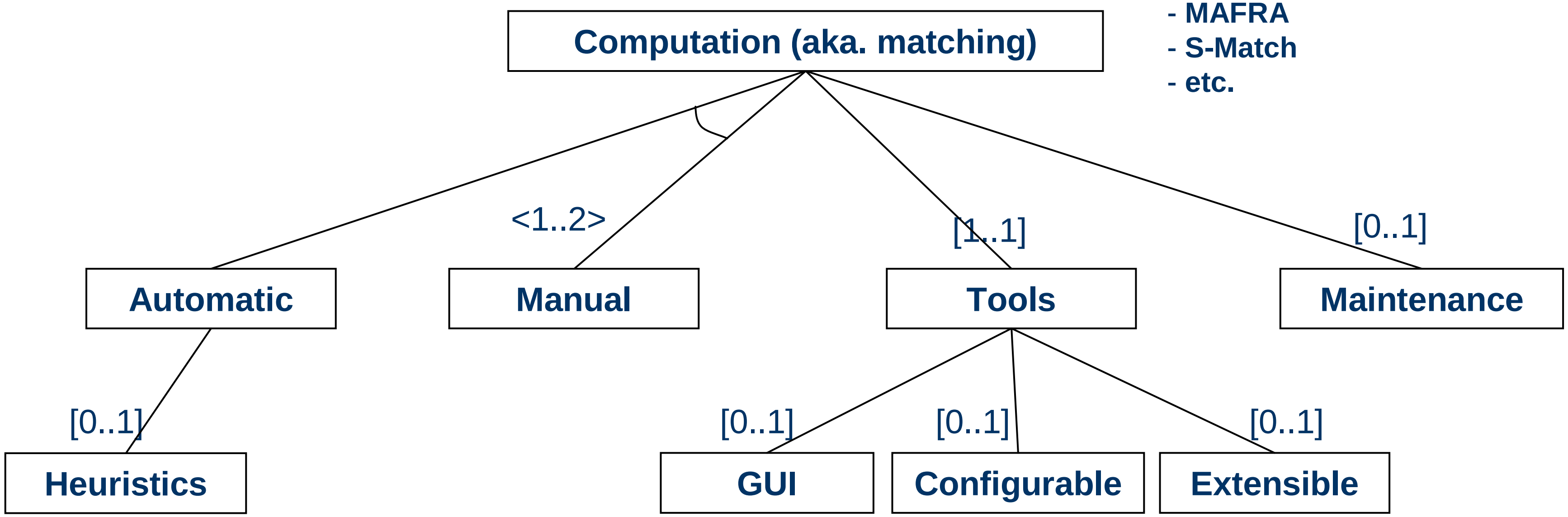



**Legend**
[] – unique cardinality
<> - shared cardinality
Formalism: feature diagrams


© Copyright IBM France 2010

# Second aspect: relationship computation

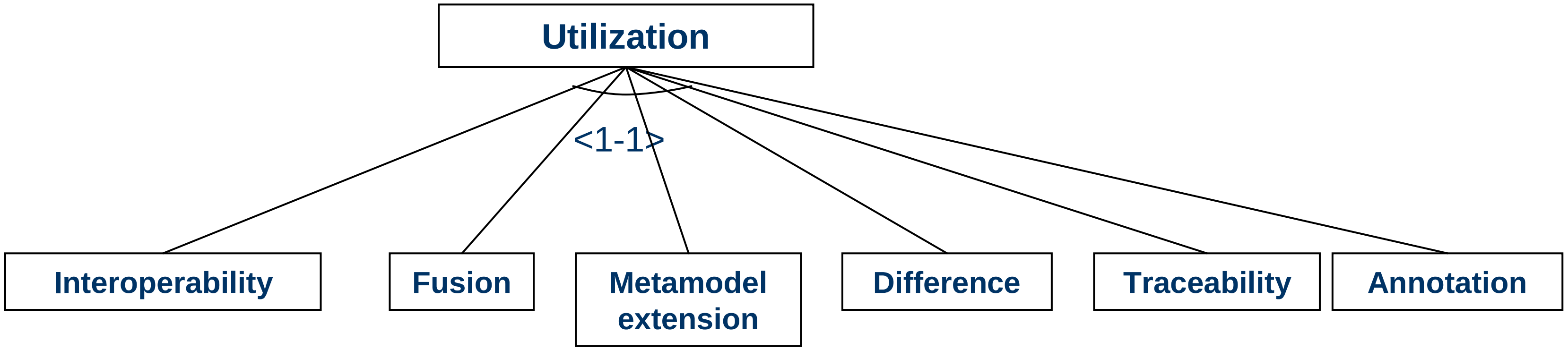
(around 138 approaches)
- COMA
- iMAP
- MAFRA
- S-Match
- etc.





© Copyright IBM France 2010

# Third aspect: relationship utilization




Strongly linked to how relationships are produced



© Copyright IBM France 2010

# Summary and requirements

- Representation
  - Different formats
  - Model management [Bernstein et al. 2000]
- Computation
  - Interoperability difficult
- Utilization
  - Transformation production
  - Traceability
  - Requirements
  - etc.


© Copyright IBM France 2010

# Representation of relationships

- Multiple technologies
  - Morphism
  - Value correspondences
  - Auxiliary model
  - Ontology bridges
- MDE solutions
  - QVT relations
  - TGG
  - Model link
  - Model weaving


© Copyright IBM France 2010

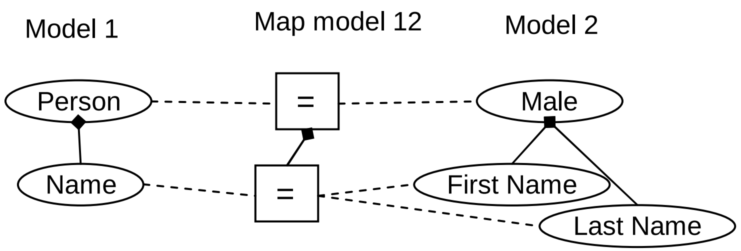
# Approaches for relationship representation (1/2)

- Morphism
  - A pair <L,R>, where
    - L is an identifier for the left element
    - R is an identifier for the right element
  - Bidirectional
  - Example : House <-> Home, Professor <-> Teacher
- Value correspondences
  - A function f : S → T.
    - A filter over the source elements from S.
  - Directed relationships
  - 1-to-1 function is the most common format
  - Example : People → Person, First + Last Name → Name
  - **Largely applied on DB community**


© Copyright IBM France 2010

# Approaches for relationship representation (2/2)

- Auxiliary model
  - A map *model* plus a pair of morphisms




- Ontology bridges
  - Mappings as first class entities for bridging ontologies
  - Identification using RDF IDs
  - Explicit mapping names
    - *AttributeBridge, ConceptBridge, RelationBridge*
    - *SubClassOf, InstanceOf*


© Copyright IBM France 2010

# Relationships on MDE : QVT Relations

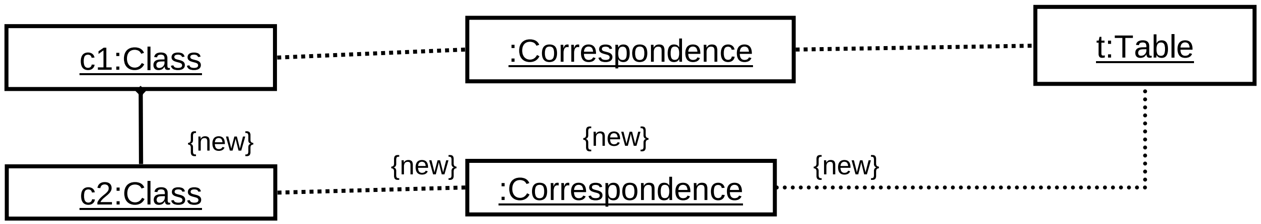
- QVT Relations : from QVT relations, core and mappings
  - “*A declarative specification of the relationships between MOF models.* “ from QVT spec.
- Not only simple correspondences
  - Support to pattern matching, (Bi) directionality, nested relations, traceability
  - Targeted for transformations

```
relation ClassToTable {
<checkonly/enforce>
domain uml c:Class {
  namespace = p:Package {},
  name=cn
}
domain rdbms t:Table {
  schema = s:Schema {},
  name=cn,
  column = cl:Column {
       name=cn+'_tid',
       type='NUMBER'},
       primaryKey = k:PrimaryKey {
       name=cn+'_PK',
       column=cl}
}
when {
  PackageToSchema(p, s);
}
where {
  AttributeToColumn(c, t);
}
}
```


© Copyright IBM France 2010

# Triple Graph Grammars (TGG)

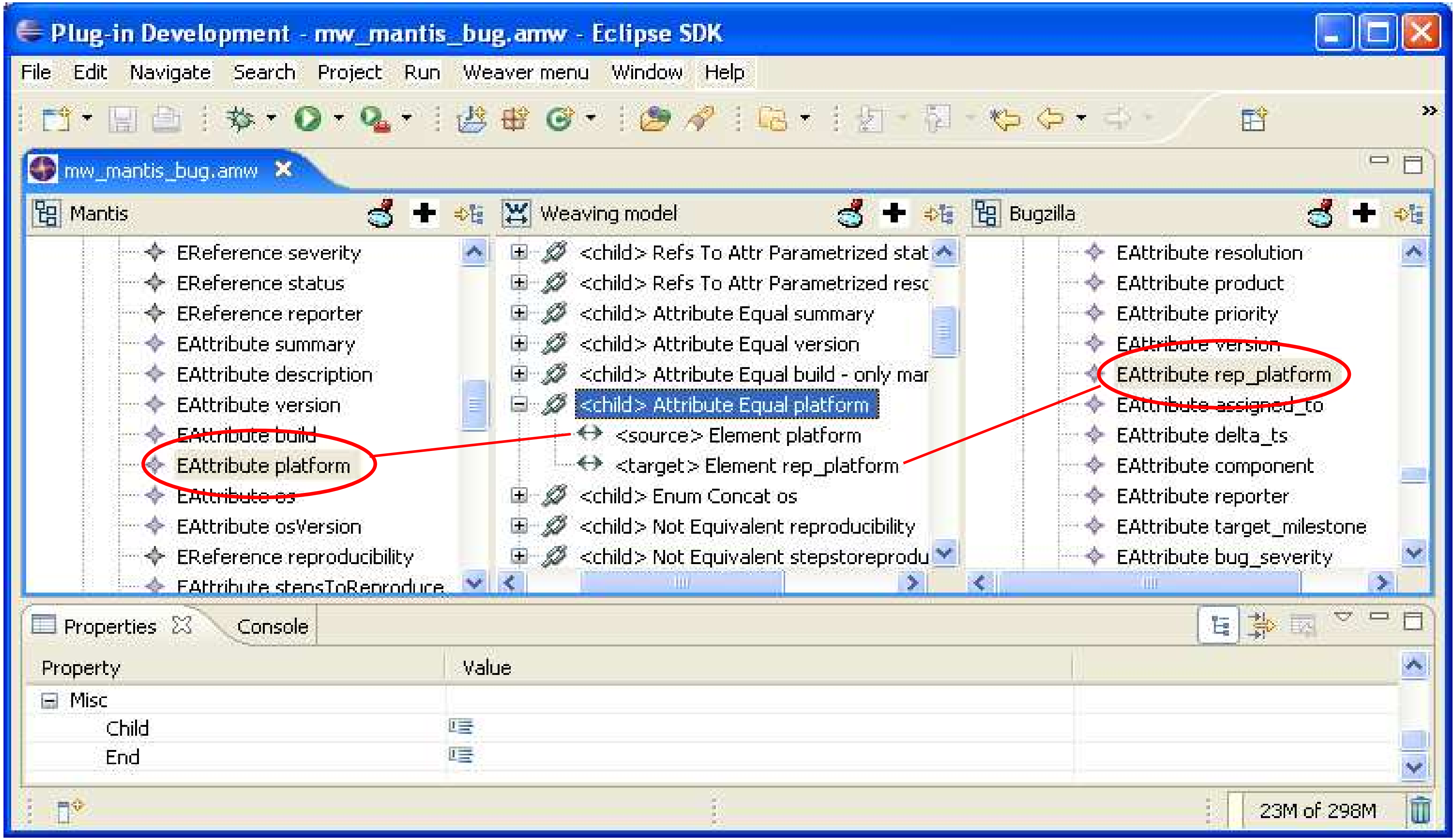
- TGG schema
  - a pair of graphs
  - a correspondence graph
- TGG rules
  - Instance-based approach
- Mix of LHS and RHS
  - Instantiate the three elements (left, link and right)
  - Transformation and weaving
    - Limited pattern and navigation expressions
- Rewriting rules (transformations) over the elements of 3 graphs




- Applications
  - (Bidirectional) transformations
  - Integration
  - Synchronization


© Copyright IBM France 2010

# Model weaving : an illustrative example

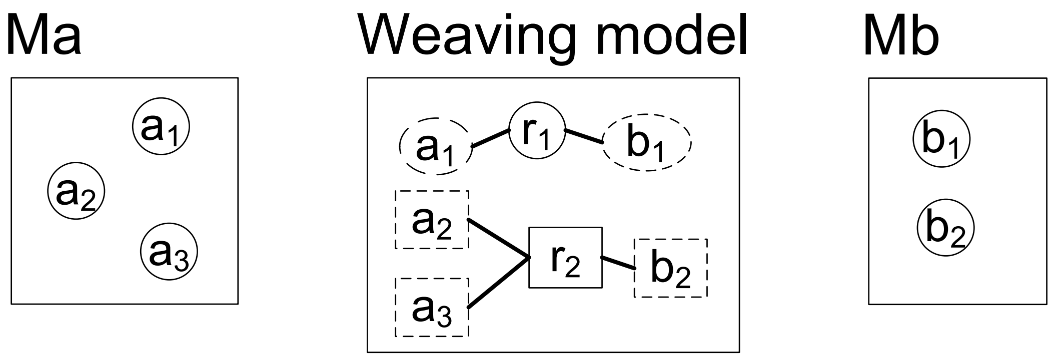





© Copyright IBM France 2010

# Model weaving

- 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.






© Copyright IBM France 2010

# Weaving model and metamodel

- **Weaving metamodel:** A weaving metamodel is a model $MM_W = (G_M, \omega_M, \mu_M)$, that defines link types, such that:
  - $G_M = (N_M, E_M, \Gamma_M)$,
  - $N_M = N_L \cup N_{LE} \cup N_O$, $N_L$ denotes the *link types*; $N_{LE}$ denotes the *link endpoint types* and $N_O$ denote other auxiliary nodes,
  - $\Gamma_M : E_M \rightarrow (N_L \times N_{LE}) \cup (N_O \times N_M)$, i.e., a *link type* refers to multiple *link endpoint types* and the auxiliary nodes refer to any kind of node.

- **Weaving model :** A weaving model is a model $M_W = (G_W, \omega_W, \mu_W)$, a graph $G_W = (N_W, E_W, \Gamma_W)$, such that its reference model is a weaving metamodel ($\omega_W = MM_W$).

- The related models are independent
  – 1-to-N models can be related



© Copyright IBM France 2010

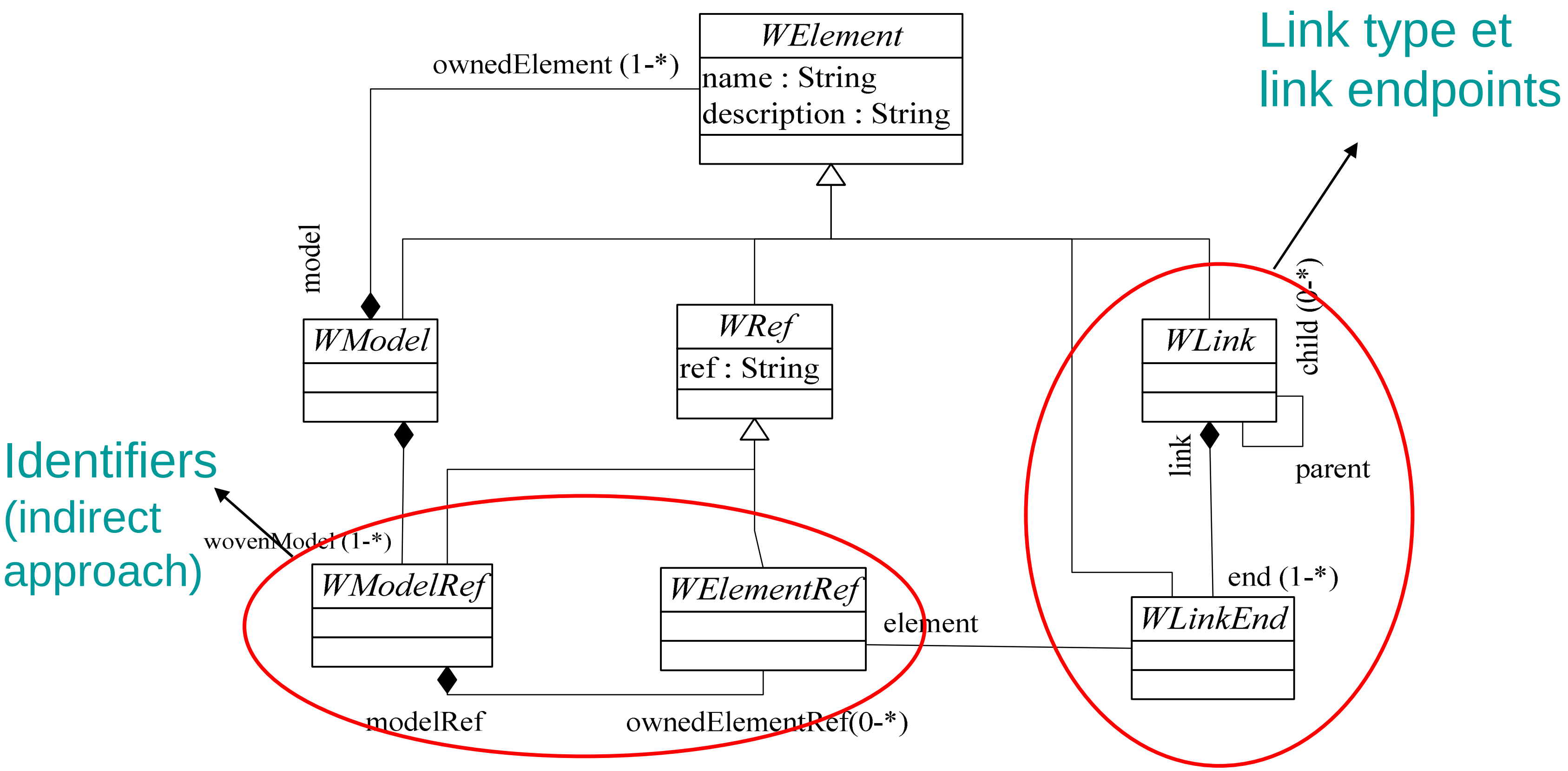
# Dereferencing function

- **Dereferencing function:** Given a weaving model $M_W = (G_W, MM_W, \mu_W)$, $G_W = (N_W, E_W, \Gamma_W)$ and a linked model $M = (G, \omega, \mu)$, $G = (N_G, E_G, \Gamma_G)$, a dereferencing function $\rho$ returns the elements of the linked model:
  - $\rho : N_{WLE} \rightarrow N_{G,}\ N_{WLE} \subset N_W$, such that $\mu_W (N_{WLE}) = N_{LE}$.

- This means the elements of the weaving models are "*pointers"* to the elements of the linked models, and they conform to the link end points.


© Copyright IBM France 2010

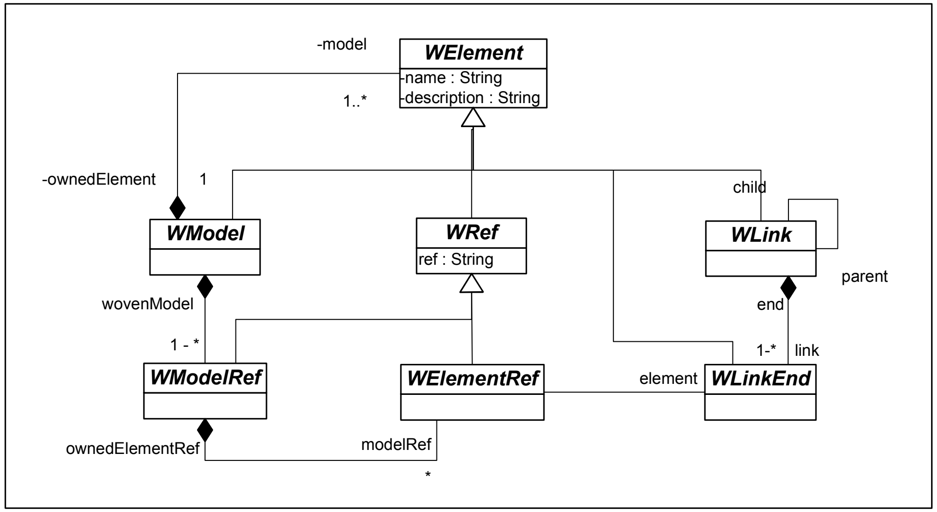
# Weaving metamodel (core)

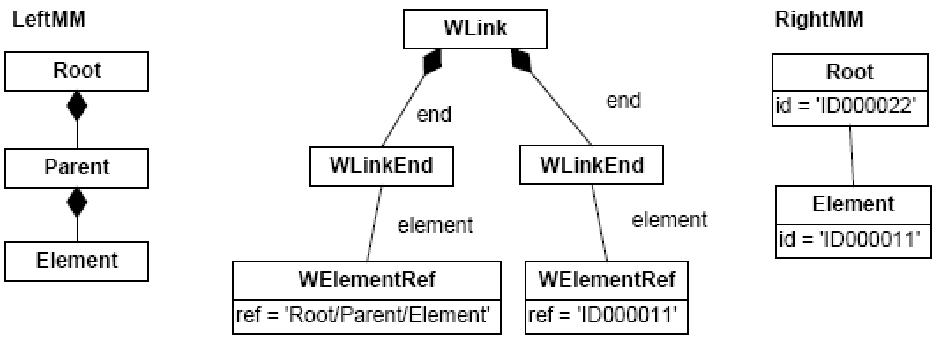





© Copyright IBM France 2010

# Core weaving metamodel



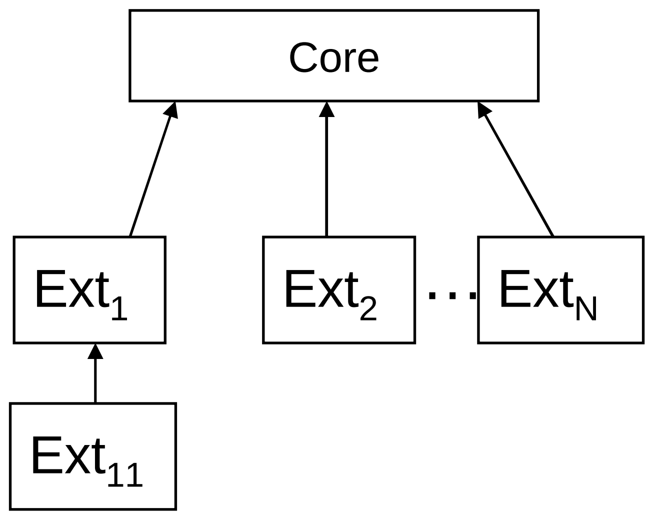

```
package mwcore {
        abstract class WElement{
                attribute name : String;
                attribute description : String;
                reference model : WModel oppositeOf ownedElement;
        }

        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;
        }
}
```



© Copyright IBM France 2010

# Weaving metamodel extensions

- The core metamodel must be extended for a given application domain
  – *Interoperability*
    •*Equality*, *SourceToTarget*.
  – *Data integration*
    •*Concatenation*, *Equality*, *IntToStr*.
  – *Traceability*
    •*Origin*, *Source*, *Evolution*, *Modified*, *Added*
  – *Composition*
    •*Override*, *Merge*, *Delete*.
  – *Ontology alignment*
    •*Equivalent*, *Equality*, *Resemblance*, *Proximity*.





© Copyright IBM France 2010

# Extension operation

```
class InheritanceLink extends WLink {
  reference parents[1-*] container : WLinkEnd;
  reference child container : WLinkEnd;
}
```

**$MM_R$ = Extend ($MM_W$, $MM_E$, $M_{WD}$)**

**Input:**

$MM_W$ : the metamodel to be extended

$MM_E$ : the metamodel extension

$M_{WD}$ : a weaving model between the elements of MMw and MMe

**Output:**

$MM_R$ : an extended MMw

/* add all elements and edges from $MM_E$ into $MM_W$, if they do not already exist*/

**for each** mme $\in MM_E$ **and not** mme $\in MM_W$

$MM_W \leftarrow MM_W \cup$ mme

/* addLink gets the elements represented by $M_{WD}$ and create a link between them*/

$MM_W \leftarrow MM_W$ **addLink** ($M_{WD}$)

**return** MMw



© Copyright IBM France 2010

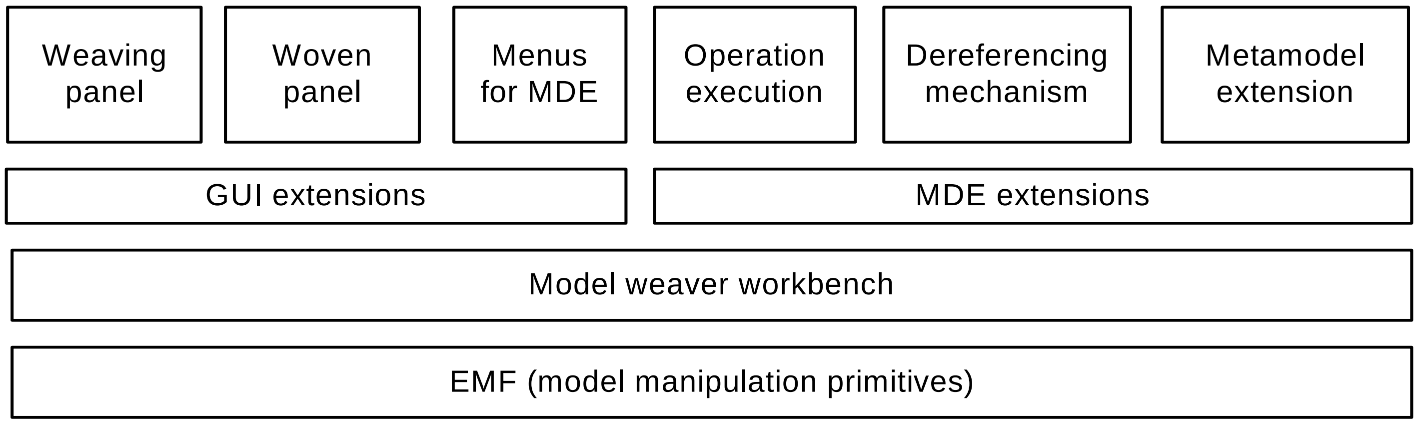
# A simple weaving metamodel extension

```
package mw_base_ext {
      class Model extends WModel {
         -- @subsets wovenModel
         reference leftModel container : WModelRef;
         -- @subsets wovenModel
         reference rightModel container : WModelRef;
      }
      class ElementRef extends WElementRef {
      }
      class ModelRef extends WModelRef {
      }
      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 {
      }
}
```


© Copyright IBM France 2010

# AtlanMod Model Weaver (AMW): a tool for editing weaving models

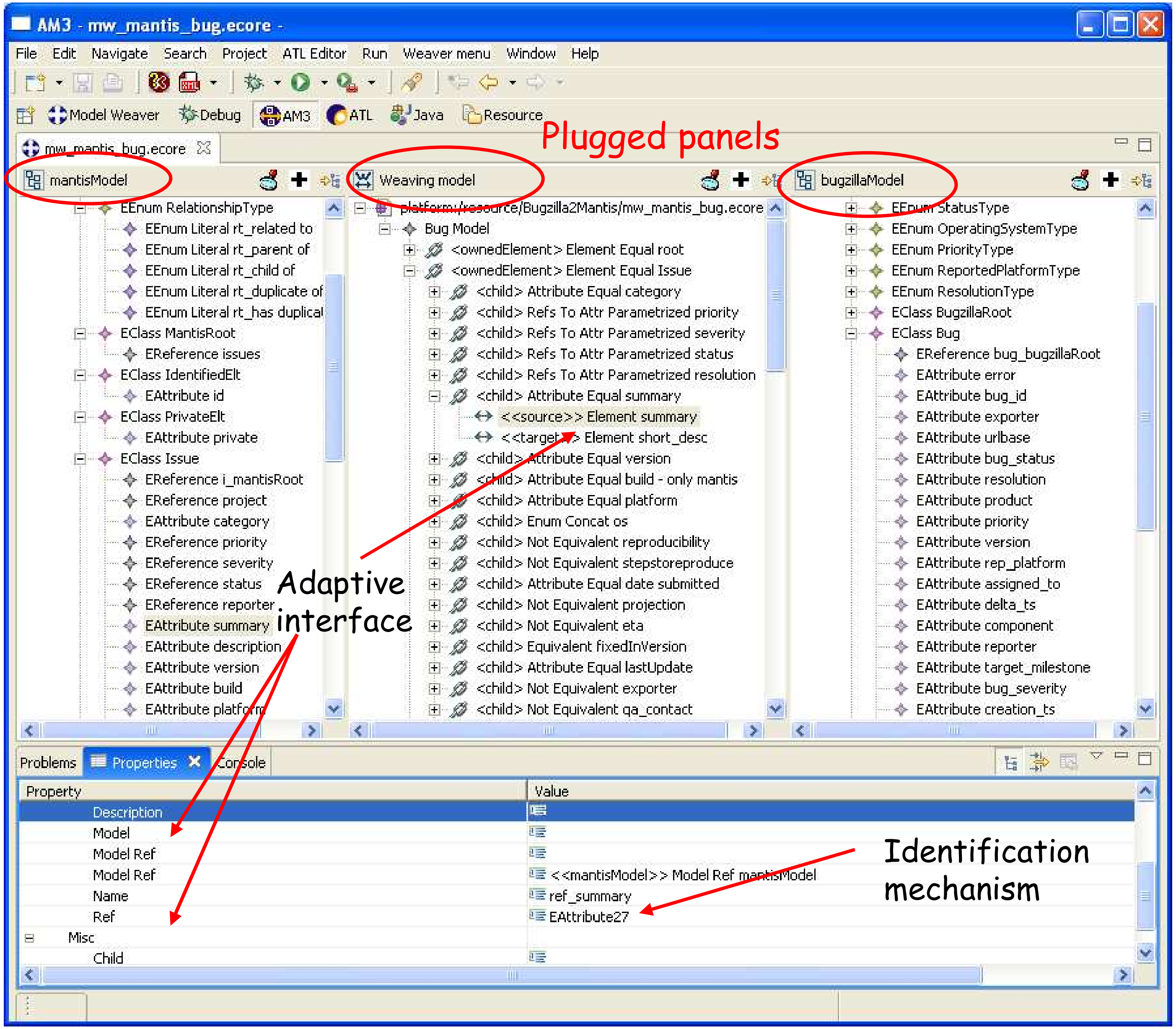
- Adapts to any weaving metamodel extension
  - The user interface is automatic generated according to the metamodel extensions
    - Reflective API of EMF (Eclipse Modeling Framework)
- A set of extension points is defined to enable to customize the standard user interface
  - Extension points to the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





© Copyright IBM France 2010

# AMW user interface






© Copyright IBM France 2010

# Summary

- Relationship between model elements
  - Several solutions, specialized for different aspects
- Model weaving
  - Generic representation
    - Based on the core metamodel and extensions
    - ATLAS Model Weaver tool


© Copyright IBM France 2010

## Traceability

- Data provenance
  *"the problem of discovering the origin of data after it was transformed from a source schema into a target schema"*
- Requirements traceability
  *"keeps track of all the steps of a development process: analysis, design, programming, testing. Some possible kinds of links are developed_by, allocated_to, performed, based_on, modify. The key processes are the identification of the possible kinds of links and the development of new traceability reference models" .*

  – Static requirements traceability
    - Requirements to code (several stages)

  – Event-based traceability
    - Subscribes to a service (observer pattern)

  – Reference models
    - Models used just for referring traceable models

Traceability survey [Galvao I, Goknil A. Survey of Traceability Approaches in Model-Driven Engineering. EDOC 2007]


© Copyright IBM France 2010

# Traceability

- Traceability of model transformations
  "*Similar to data provenance scenarios, it is often necessary to store the execution trace of model transformations. The execution trace of a transformation indicates, for a set of generated elements, which transformation rules are executed, and which input elements are used.*"

  – Loosely coupled
    - Batch execution of model transformation produces a weaving model

  – Embedded traceability → annotation
    - Merges a model and its trace information


© Copyright IBM France 2010

# Outline

Model weaving : state of the art and concepts

**Practical work : schema mapping and traceability**

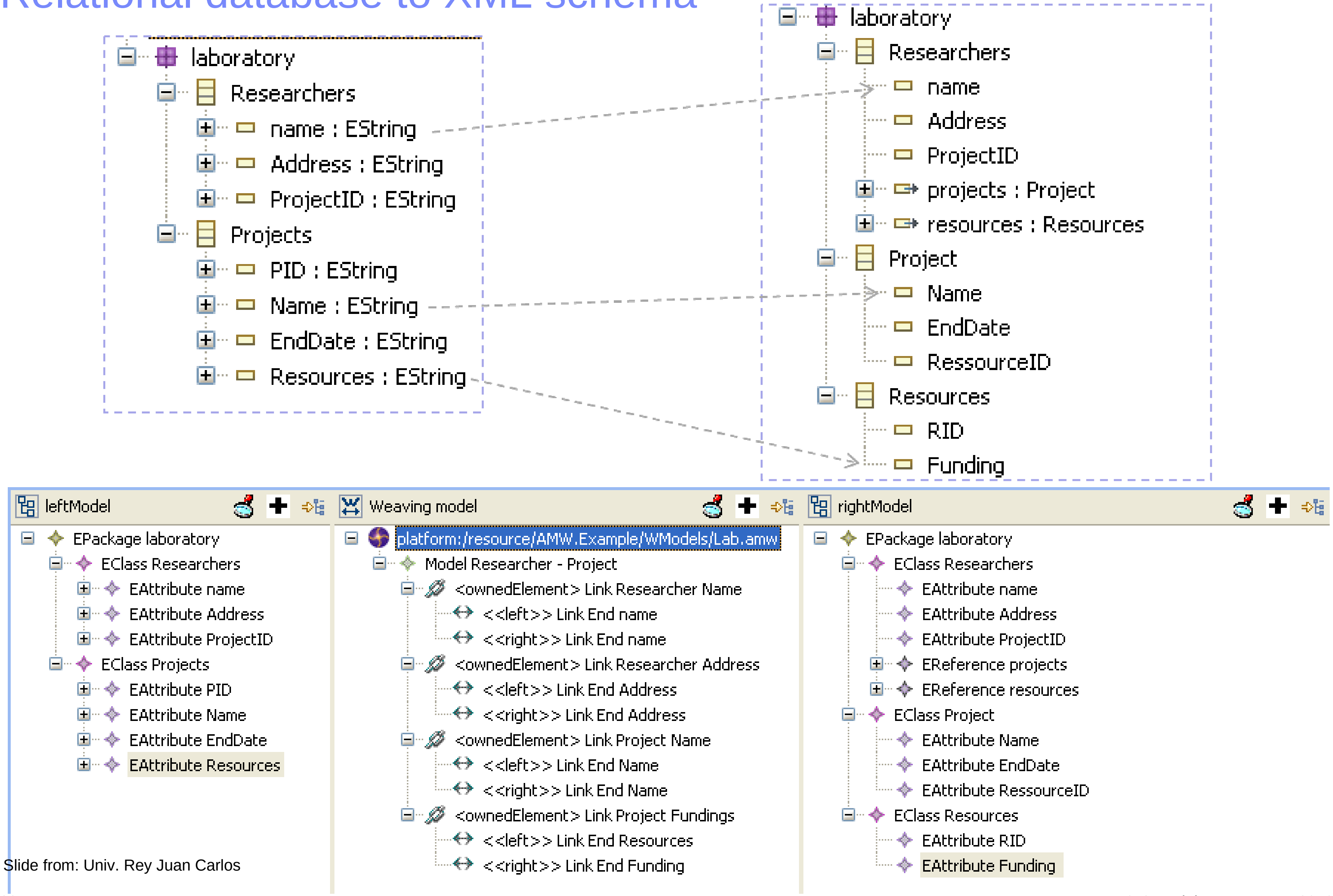
Matching and transformation production

Practical work : matching and transformation production



© Copyright IBM France 2010

# Relational database to XML schema



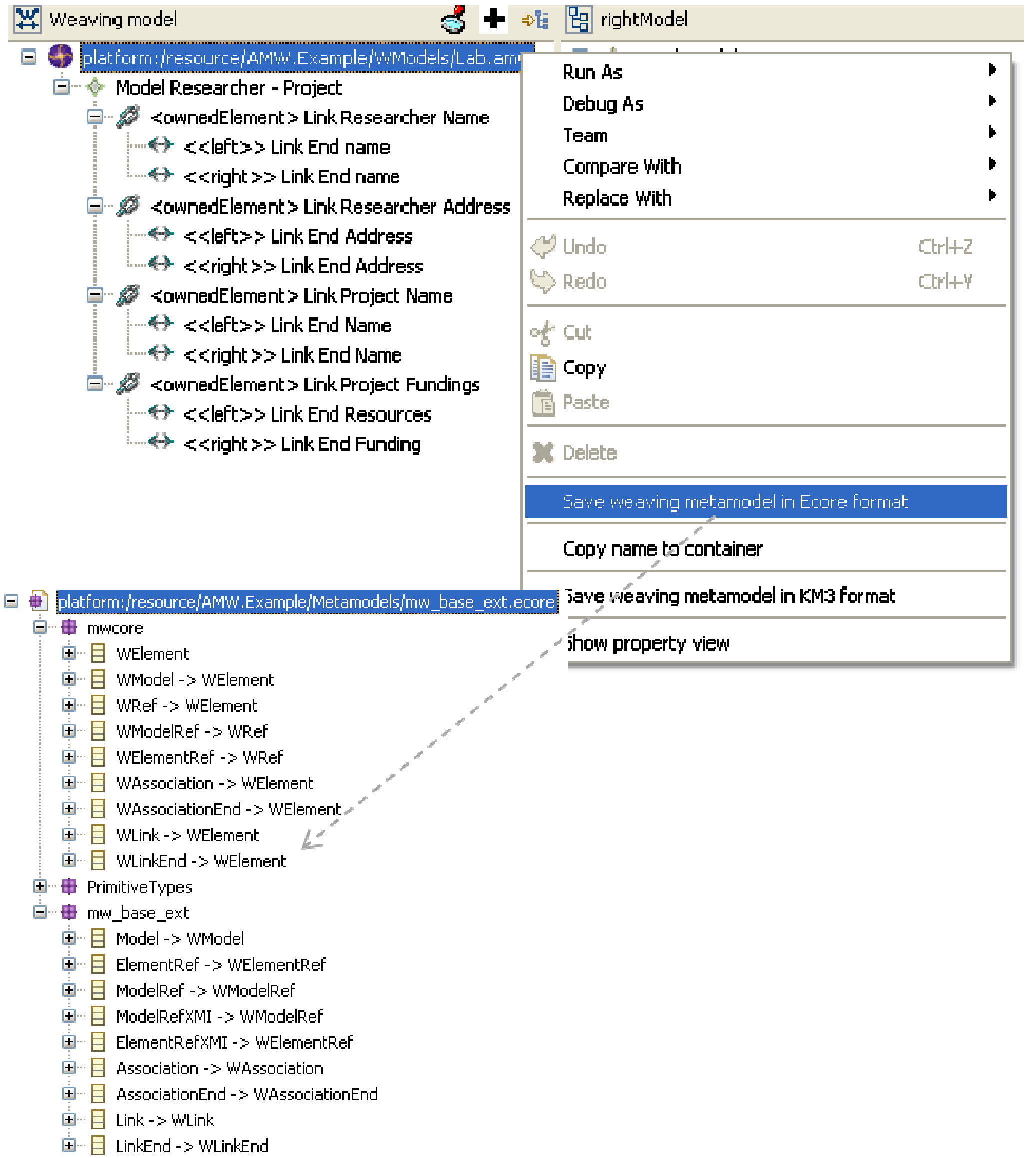

Slide from: Univ. Rey Juan Carlos



© Copyright IBM France 2010

# Weaving metamodel extension

```
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 {}

}
```



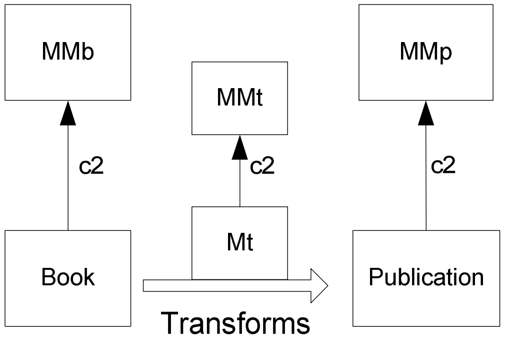

Slide from: Univ. Rey Juan Carlos



© Copyright IBM France 2010

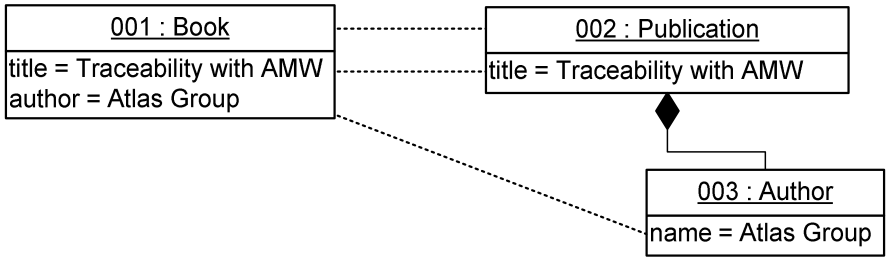
# Traceability of model transformations

- Original transformation setting




```
rule Book2Publication {
   from
      s : MMb!Book
   to
      t : MMp!Publication (
         title <- s.title + s.subtitle,
         pubYear <- s.year,
         authors <- author
      ),
      author : MMp!Author (
         name <- s.author
      )
}
```

- How to store traceability information?

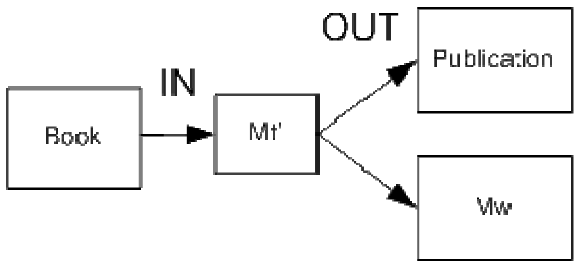




© Copyright IBM France 2010

# Traceability of model transformations

- Produce Mt' from Mt using a Higher Order Transformation
- Mt' produces an additional weaving model




- Weaving metamodel extension

```
class TraceLink  extends WLink{
    attribute ruleName : String;
    reference sourceElements[*] ordered container : WLinkEnd;
    reference targetElements[*] ordered container : WLinkEnd;
}
class TraceLinkEnd extends WLinkEnd {
}
class ElementRef extends WElementRef {
}
```


© Copyright IBM France 2010

# Outline

Model weaving : state of the art and concepts

Practical work : traceability and schema mapping

**Matching and transformation production**

Practical work : matching and transformation production


© Copyright IBM France 2010

# Matching

- Matching is the process of establishing relationships between elements belonging to different models

- Manual
  - User interface
- Automatic
  - Algorithms
- Semi-automatic
  - Utilization of heuristics


© Copyright IBM France 2010

# Matching heuristics

- String similarity
  – Date <-> BirthDate
- Dictionaries
  – Car <-> Vehicle
- Structural relations
  – Class.name <-> Table.name
    •Class <-> Table

- Different problems
  – How to express this heuristics ?
  – How to support different extensions ?


© Copyright IBM France 2010

# Matching tools

| | Input (internal and external representation) | Matching techniques | Mapping nature | Application scenario |
|---|---|---|---|---|
| **CUPID** | DB and XML schemas (rooted graphs) | Structural and linguistic | 1:1 correspondences | Generic matching tool |
| **GLUE** | Unified ontology (rooted graph) | Data instances, probability distribution | 1:1 mappings | Generic matching tool |
| **PROMPT** | Ontologies (general knowledge model) | Set of iterative operations | None: merges the ontologies | Ontology merging and alignment |
| **COMA / COMA++** | SQL, XML and OWL schemas (rooted directed graphs) | Library of heuristics | Equivalence 1:1 correspondences | Generic matching tool |
| **ONION** | Ontologies (directed graphs) | Interoperation operators | Articulation ontologies | Ontology integration |
| **MAFRA** | RDF schemas | Multi strategies (lexical and structural) | Semantic bridging ontology (SBO) | Alignment of distributed ontologies |
| **S-Match** | Ontologies (propositional formulas) | Propositional unsatisfiability problem | Logical relations | Generic framework |
| **API for ontology alignment** | RDF graphs | Provides an API | Simple 1:1 correspondences translated into XSLT, C-OWL,RDF | Generic ontology matching |
| **iMAP** | Database schemas (graphs) | Different machine learn searchers. Use of domain knowledge | 1:1 mappings and complex functions | Data integration |
| **Xu et al.** | Database schemas | Different matchers and domain ontologies | Complex mapping expressions | Data integration |


© Copyright IBM France 2010

# Matching transformations

- 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
  – $T_{MATCHING}$ → model x … x model → weaving model

```
rule CreateLink {
from
   aLeft : MMa!Class, aRight : MMb!Attribute   (guard)
to
  aLink : MMw!Equivalent (
     left <- getID(aLeft),
     right <- getID(aRight),
     similarity <- aLeft.calcSim(aRight)
)
}
```

Execution condition

Similarity computation [0-1]



© Copyright IBM France 2010

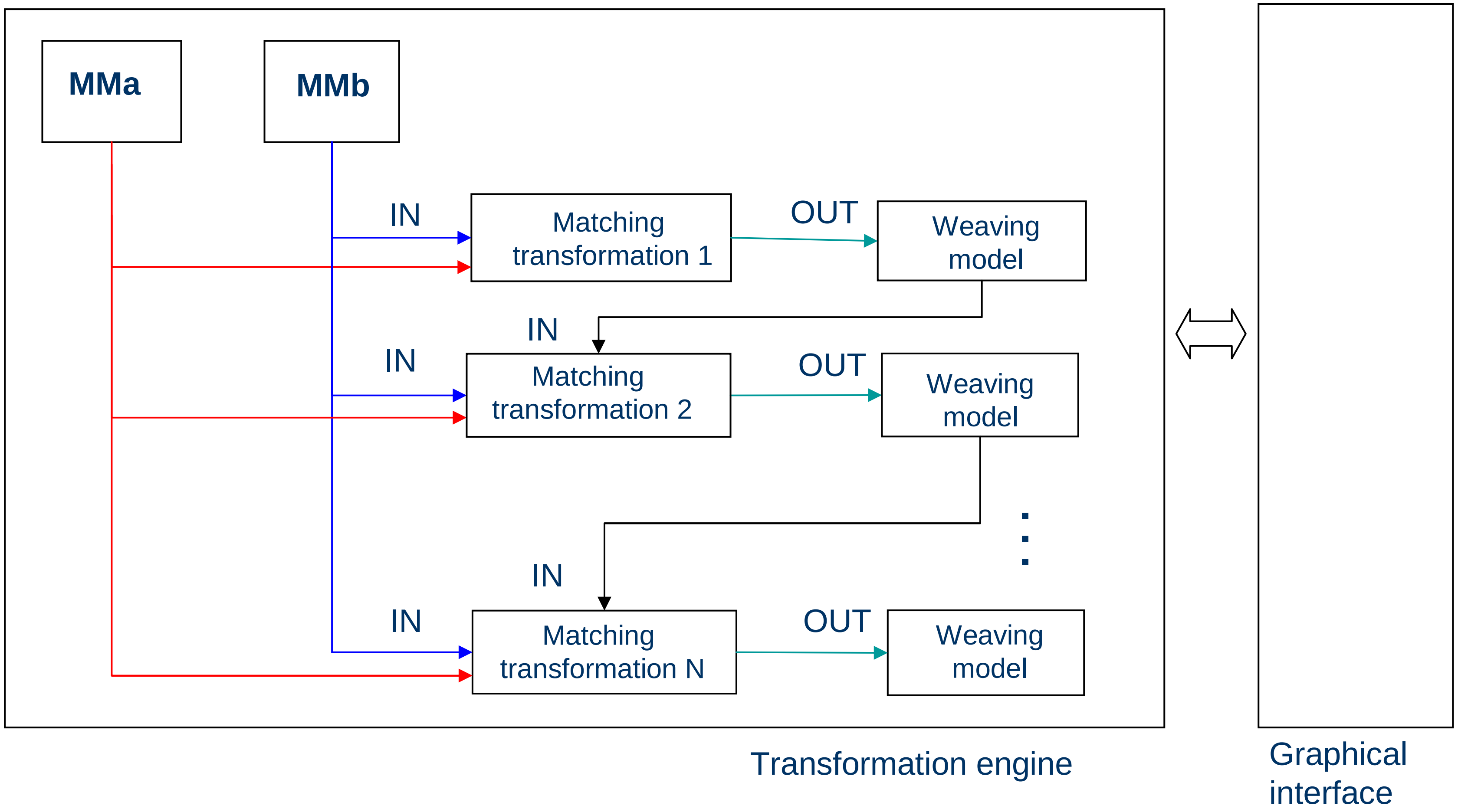
# Simple matching extension

```
class Element extends WLinkEnd {
}
class Equivalent extends WLink {
          attribute similarity : Double;
          reference source container : Element;
          reference target container : Element;
}
class <Type>Equal extends Equivalent {
}
class AttributeToRef extends Equivalent {
          reference targetAttribute container : Element
}
```



© Copyright IBM France 2010

# Cumulative matching





© Copyright IBM France 2010

# Cumulative matching

- Different kinds of matching transformations
- Element creation
- Similarity calculation and propagation
- Link rewriting
- Link selection

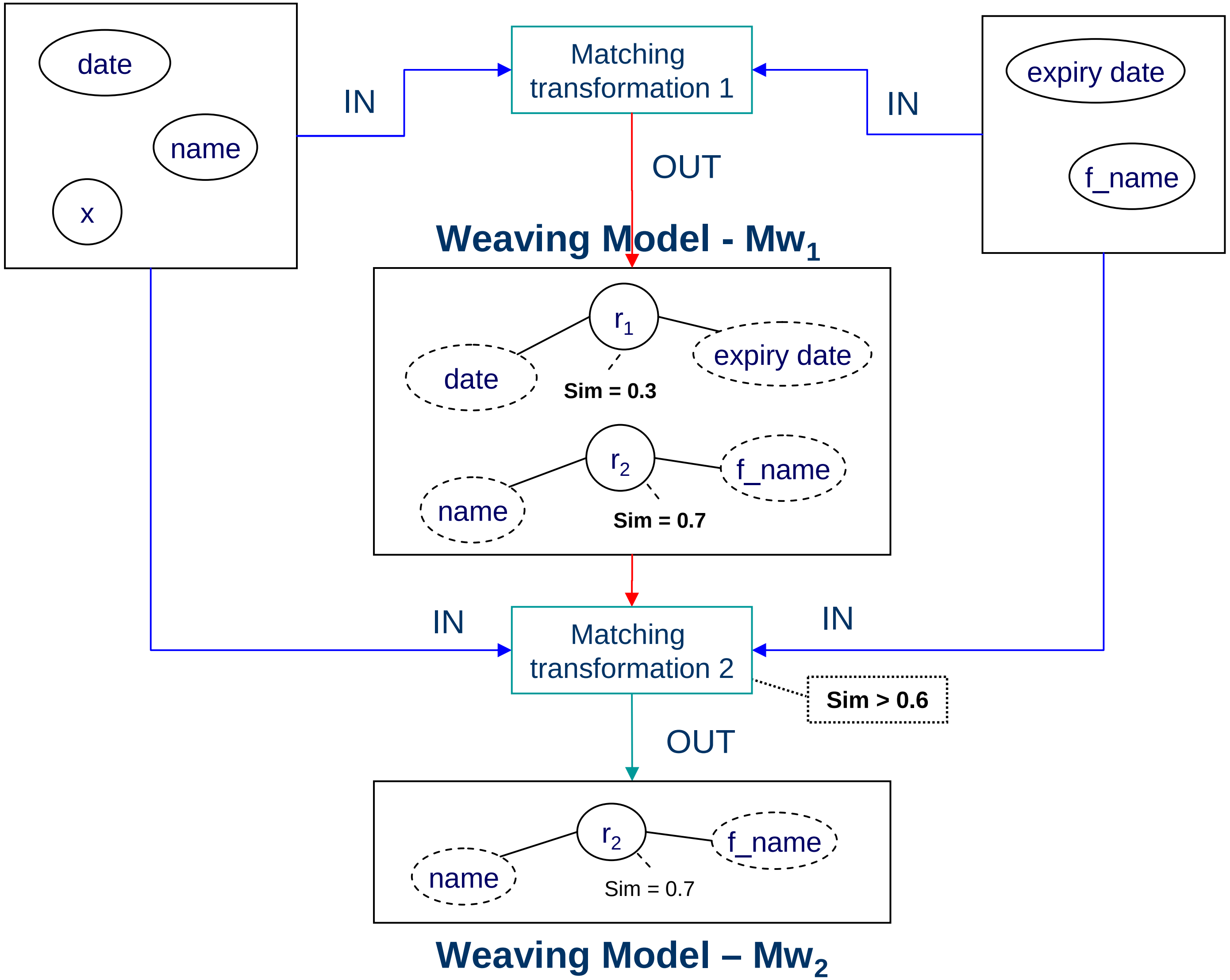

© Copyright IBM France 2010

# Matching rule for creating simple links

```
rule CPClass {
  from
    left : Ecore!EClass, right : Ecore!EClass
to
    AMW!ClassEqual
}
rule CPAttr {
  from
    left : Ecore!EAttribute, right : Ecore!EAttribute
  to
    AMW!AttributeEqual
}
```


© Copyright IBM France 2010

# Cumulative matching: similarity + link filtering




Slide from: Univ. Rey Juan Carlos



© Copyright IBM France 2010

## Calculating similarity

- Simple element-to-element similarities

```
rule AttributeSimilarity {
  from
    mmw : AMW!AttributeEqual
  to
    alink : AMW!AttributeEqual (
    similarity <- (mmw.similarity + mmw.left.similarityName(mmw.right)) * weight
)
```

- Structural similarity

```
rule UpdateStructuralSim {
from
  mmw : MMw!Equal mmw.source.isTypeOf(KM3!Attribute) and mmw.target.isTypeOf(SQLDDL!Column))
to
  alink : MMw!Equal (
  similarity <- ( mmw.similarity +  mmw.source.requiredSim( mmw.target )) * weight
  )
}
helper context KM3!Attribute def: requiredSim (column : SQLDDL!Column) : Real =
  if (self.lower = 0 and column.canBeNull) then
    1
  else
    0
  endif;
```


© Copyright IBM France 2010

# Similarity flooding (SF) : a generic structural algorithm

- Input
  - Two metamodels $M_a$ and $M_b$,
  - Model elements *a, a'* $\in M_a$ and *b, b'* $\in M_b$.
    - Elements *a* and *a'* are connected by a labeled edge *(a, "containment", a')*.
    - Elements *b* and *b'* are connected by a labeled edge *(b, "containment", b')*.
- Initial setup and execution
  - Link creation : Cartesian product of $M_a \times M_b$
  - Similarity assignment for every pair of elements.
- Iterative propagation
  - **General idea:** consider the pairs *(a, b)* and *(a', b'),* with similarities *x* and *y*, respectively. The algorithm propagates *x* to *(b, b')* and it updates the similarity value *y.*
  - **Propagation formula**
    - $y = y + (p * x)$.
  - **Calculation of *p***
    - number of edges connecting a given pair of elements
    - Ex.: if *(a, a')* has 10 neighbors, then $p = 1/10$.
  - **Propagation graph :** stores the propagation information.


© Copyright IBM France 2010

# Adaptation of SF for model weaving and matching transformations

- Choose one kind of structural information
  – Containment graph
  – Inheritance tree
  – Relation graph
  – Any other relations
- Define how to calculate **p**
- Create a propagation weaving model
- Write the propagation transformation


© Copyright IBM France 2010

# Propagation weaving metamodel extension

- Propagation extension : propagation from one link into another

```
package mmw_propagation {
  class PropagationElement extends WAssociation {
    reference incomingLink : Equivalent;
    reference outgoingLink : Equivalent;
    attribute propagation : Double;
  }
}
```

- Creation of propagation elements

```
rule CreatePropagationElement {
  from
    source_link : AMW!Equivalent,
    target_link : AMW!Equivalent ( <semantic guard> )
to
   out : AMW!PropagationElement (
      propagation <- 1 / <propagation_value>,
      outgoingLink <- source_link,
      incomingLink <- target_link
  )
}
```


© Copyright IBM France 2010

# Containment propagation model creation

```
from
   source_link : AMW!ClassEqual,
   target_link : AMW!AttributeEqual (
     target_link.getReferredLeft.owner = source_link.getReferredLeft
     and target_link.getReferredRight.owner = source_link.getReferredRight
)
to
  out : AMW!PropagationElement (
   outgoingLink <- source_link,
   incomingLink <- target_link
   propagation <-
       1 /
       ( source_link.getReferredLeft.getAttributeCount()->size() *
       source_link.getReferredRight.getAttributeCount()->size()
)
```


© Copyright IBM France 2010

# Propagation rule : valid for any kind of propagation model

```
rule PropagationClass {
  from
     mmw : AMW!Equivalent
  to
     alink : AMW!Equivalent()
do {
   thisModule.aTuple <-
     AMW!PropagationElement.allInstances()->
     select ( e | e.incomingLink = mmw)->
     iterate (e1; acc : TupleType(value : Real, count : Integer) =
        Tuple {value = 0, count = 0} |
        Tuple {
          value = acc.value + (e1.outgoingLink.similarity * e1.propagation),
          count = acc.count + 1
        }
    );
    alink.similarity <- mmw.similarity +
    thisModule.aTuple.value / thisModule.aTuple.count;
  }
}
```


© Copyright IBM France 2010

# Link filtering

- Called-rule for selecting better similarities

```
rule getMaxLink (aSource : MMa!ModelElement) {
using {
  newLink : MMw!Equivalent = null;
  maxSim : Real = 0;
}
do {
   for(e in MMw!Equivalent.allInstaces()->select(e.source=aSource)){
   if (e.similarity > maxSim) {
        maxSim <- e.similarity;
         newLink <- e;
   }
}
```


© Copyright IBM France 2010

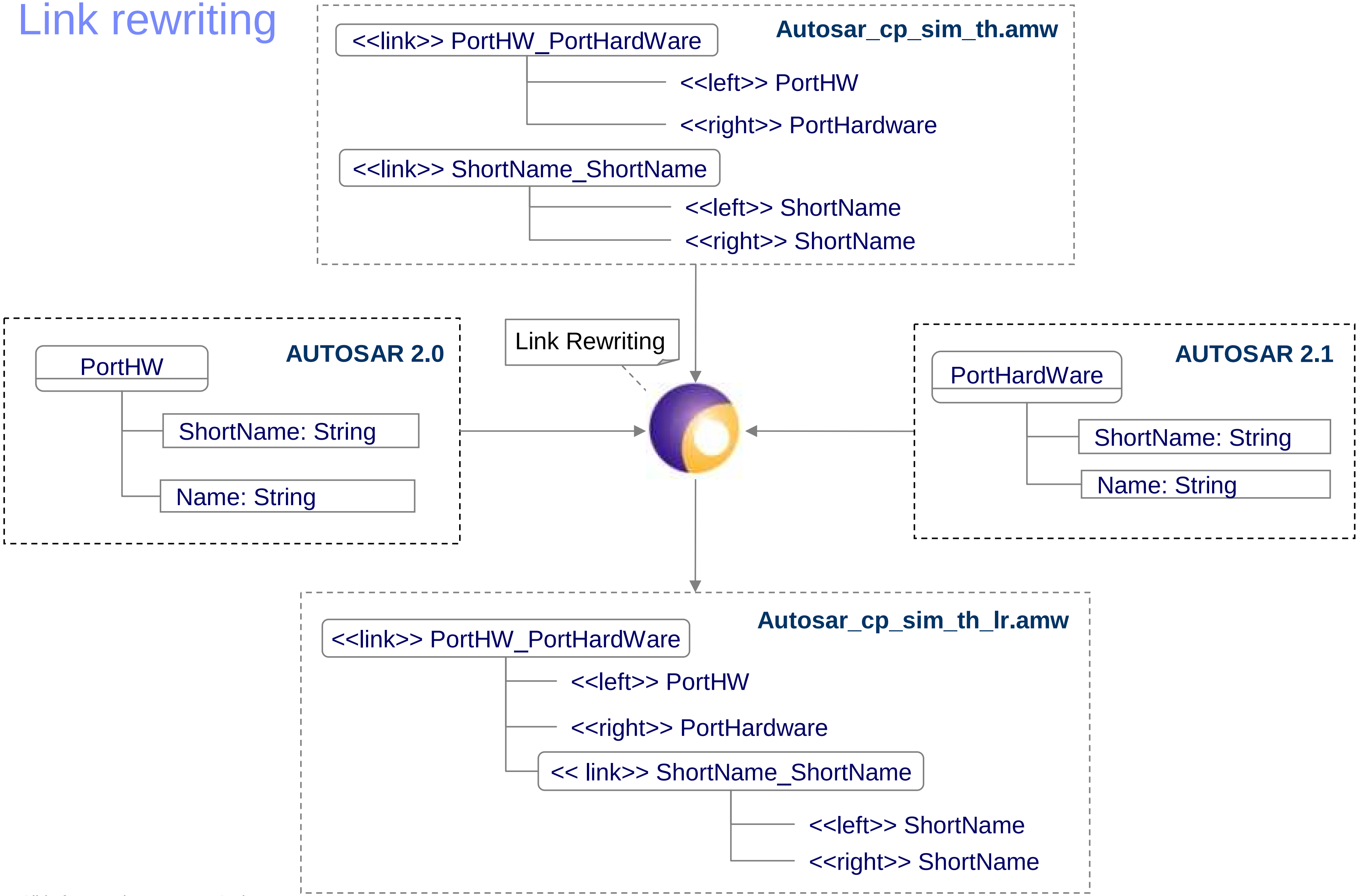
# Link rewriting

- The final similarity weaving model may not have the right connections
  – Nested relationships
  – Hierarchy
  – Others

```
rule NestedRewriting {
  from
    attr_link : MMw!AttributeEqual,
    class_link : MMw!ClassEqual (
        attr_link.source.owner = class_link.source and
        attr_link.target.owner = class_link.target
     )
  to
    link : MMw!AttributeEqual (
        parent <- class_link
    )
}
```


© Copyright IBM France 2010

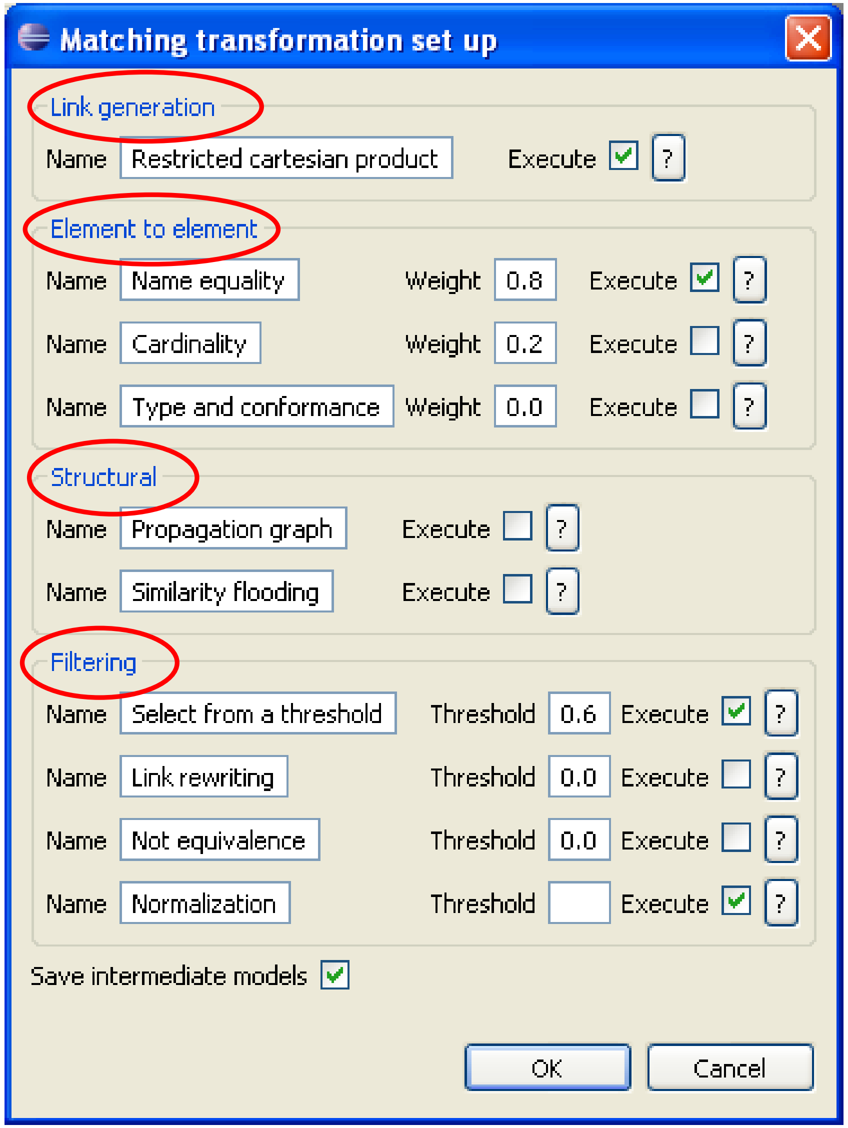
# Link rewriting



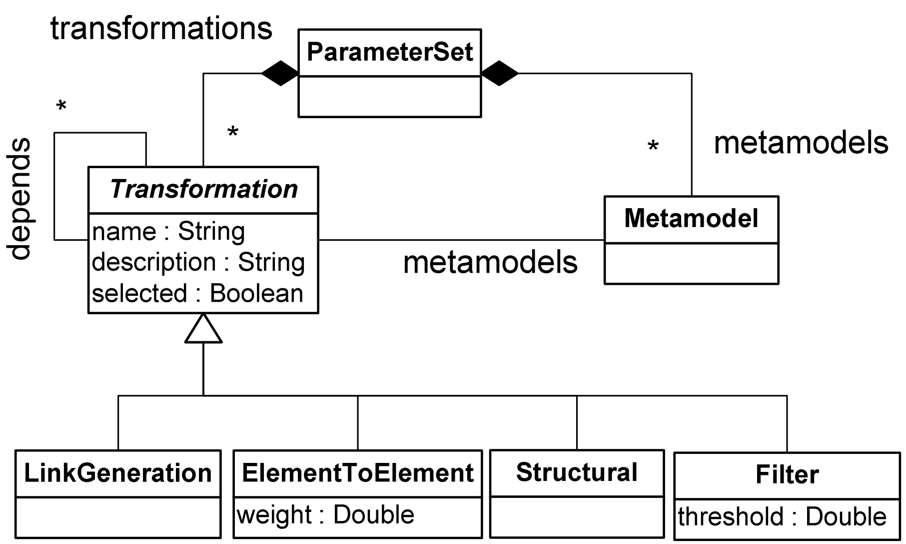

Slide from: Univ. Rey Juan Carlos


© Copyright IBM France 2010

# Configuration model  : setting up an execution chain




## Configuration metamodel




Finding the good combination of transformations/parameters is fundamental



© Copyright IBM France 2010

# Transformation production : how to use these weaving models?

- 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


© Copyright IBM France 2010

# Some solutions in DB community

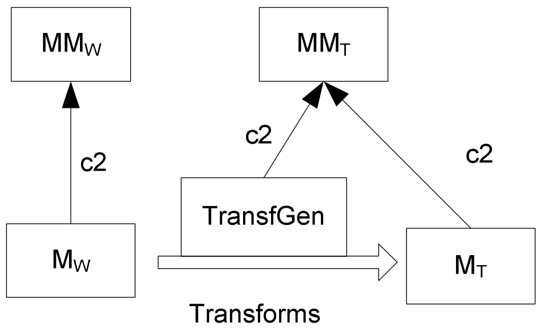
- Typically called *query discovery*
- Difficult when using complex mappings
- Specific to the corresponding application domain

| | **Input** | **Mapping nature** | **Transformations** |
|---|---|---|---|
| **Clio** | A pair of relational and XML nested schemas (internal nested format) | 1:1/n:m value correspondences | Produces logical operational mappings that are translated in SQL or XSLT |
| **Kedad et al.** | Two or more XML schemas | 1:1 value correspondences | XQuery |
| **An et al.** | Relational schemas plus a conceptual model | 1:1 value correspondences and the mappings between a schema and its conceptual model | Relational mappings |
| **SMART** | XML schemas and conceptual schemas | 1:1 value correspondences with inclusion labels | XML transformations |


© Copyright IBM France 2010

# Definitions

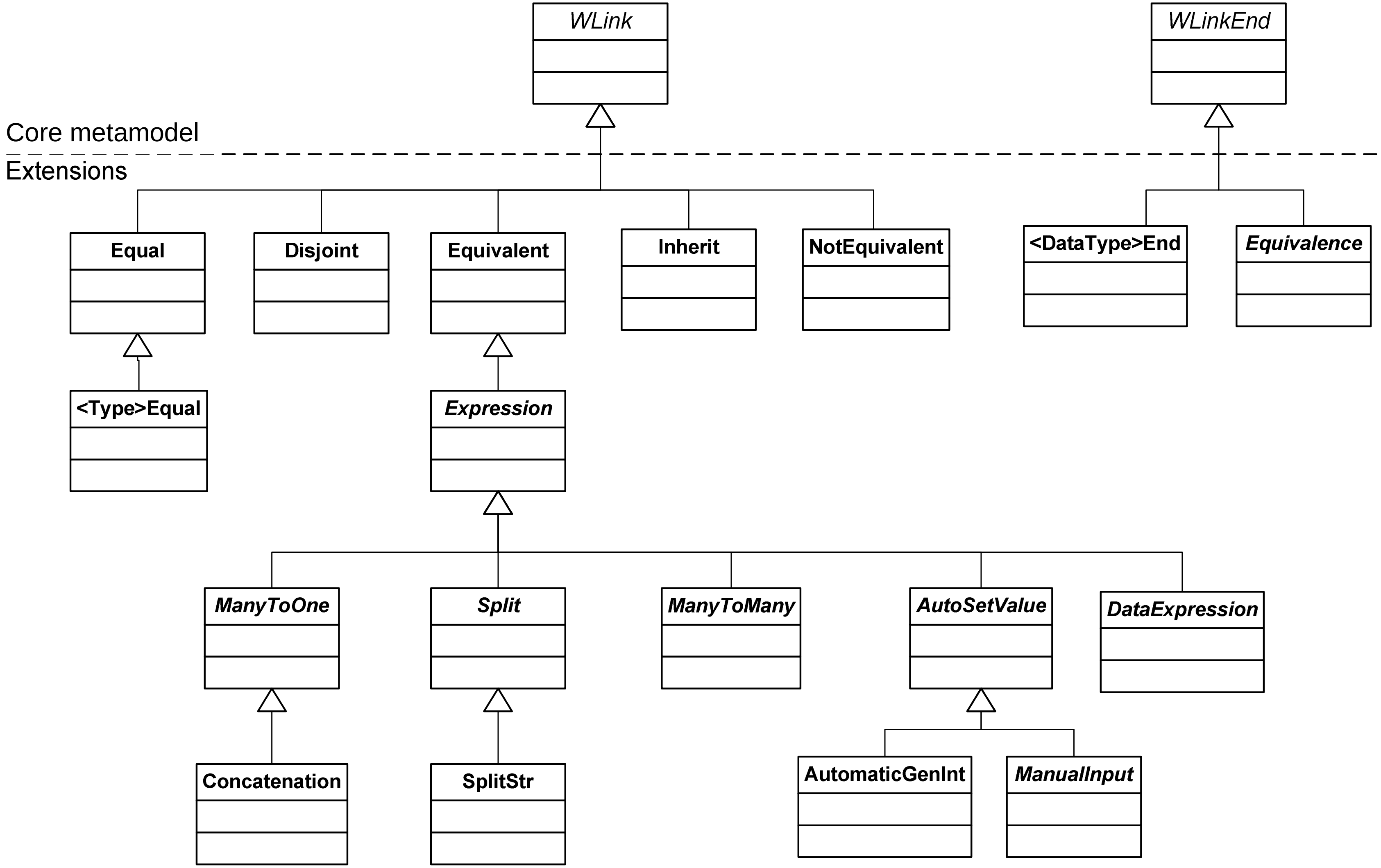
- **Higher order transformation.** A higher-order transformation is a transformation $T_{OUT} : MM_T = T_{HOT} (T_{IN} : MM_T)$, such that the input and/or the output models are transformation models. Higher-order transformations either take a transformation model as input, either produce a transformation model as output, or both.
- **TransfGen.** *TransfGen* is a higher-order transformation that takes a weaving model $M_w$ as input and that produces a transformation model $M_T$ as output. The weaving model conforms to a data interoperability metamodel extension $MM_w$.
  - $M_T : MM_W = TransfGen (M_w : MM_w)$.





© Copyright IBM France 2010

# TransfGen: input metamodel extension





© Copyright IBM France 2010

# TransfGen: output metamodel

## Transformation metamodel
(an abstraction of ATL metamodel)

```
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> …
```



© Copyright IBM France 2010

# TransfGen operation template

```
Module TransfGen (C: ω_C)

inputModel:  C /* a correspondence model conforming to a correspondence metamodel ω_C*/
outputModel: T /* a transformation model conforming to ω_T */

rule newModule
  input WModel
  output Module
       rules ← ownedElement (ownedElement isA WLinkST)

rule newRule
  input  WLinkST (parent isA WModel)        /*classifiers (classes, references, attributes)*/
  output Rule
       input ← source
       output ← target

rule newInput
  input WLinkEnd (link.source = self)
  output InputElement
       element ← ρ (element.ref)
       condition ← /*depends on the WLinkST and WLinkEnd types*/

rule newOutput
  input WLinkEnd (link.target = self)
  output OutputElement
       element ← ρ (element.ref)
       bindings ← link.child /*get the sibling WLinkEnd*/

rule newExpression
  input WLinkST (parent isA WLinkST)
  output Binding
       source ← MapExp (ρ (source.element.ref) ) /*mapping expressions here,*/
       target  ←  ρ (target.element.ref)              /*according to the WLinkST type*/
```



© Copyright IBM France 2010

# Outline

Model weaving : state of the art and concepts

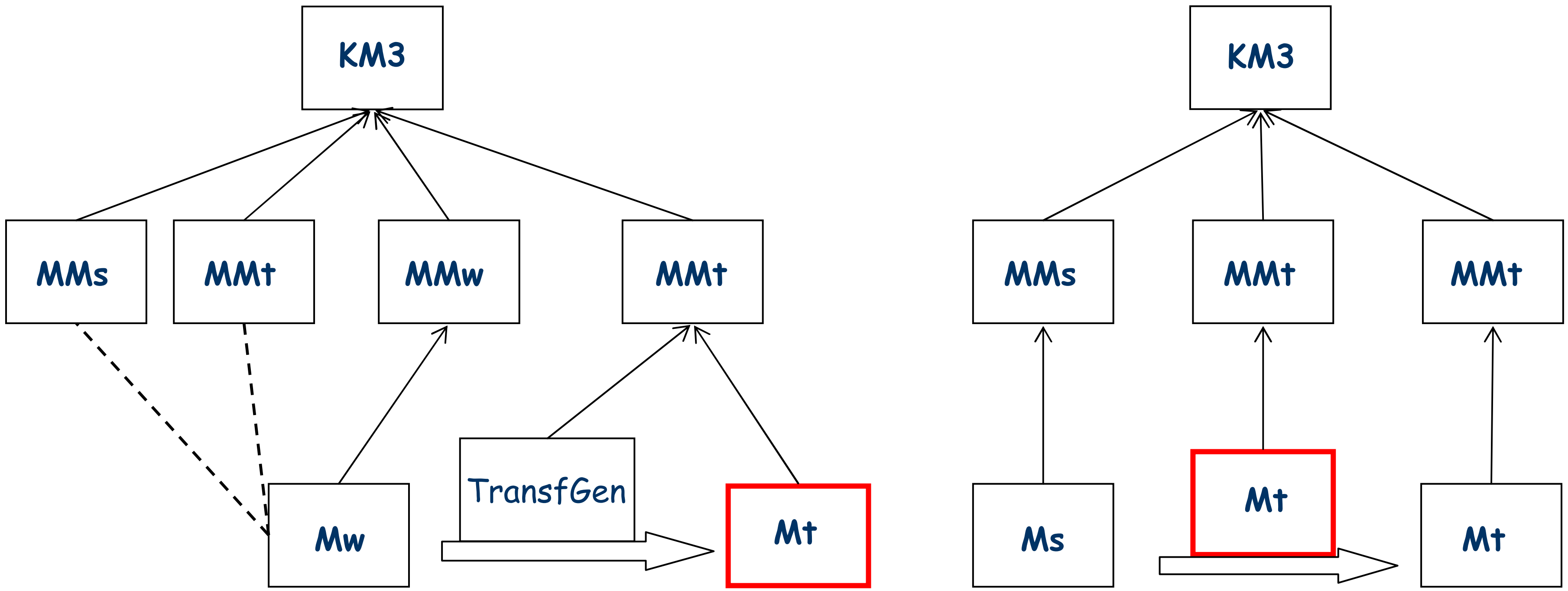
Practical work : schema mapping and traceability

Matching and transformation production

**Practical work : matching and transformation production**


© Copyright IBM France 2010

# General view

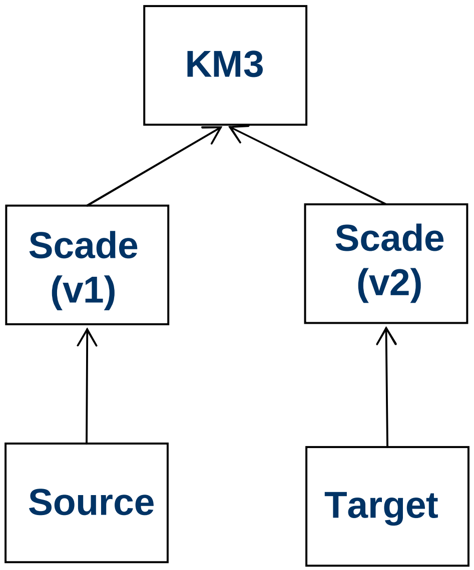



- Two major points
  - – Transformations are models
  - – Different transformation metamodels (e.g., ATL or XSLT)

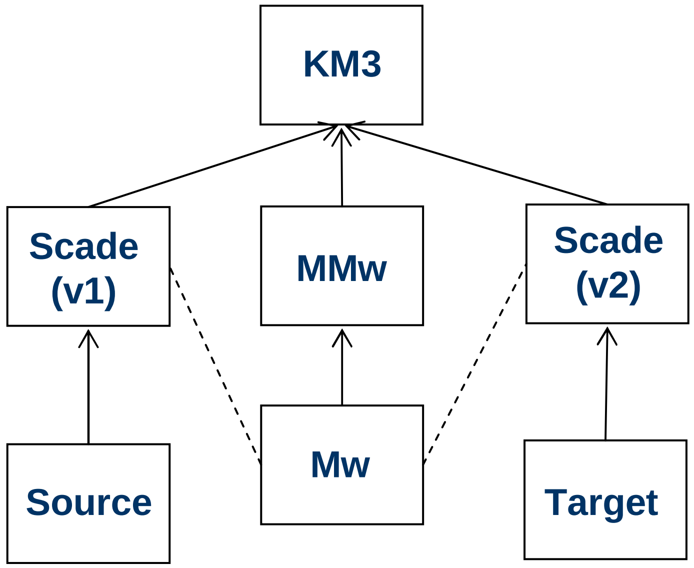

© Copyright IBM France 2010

# Case study : Comparison and migration – putting all together

- Two versions
  - Scade of Esterel Technologies (v1 and v2)
  - Autosar (v2.0 and v2.1)




- Creation of the weaving model

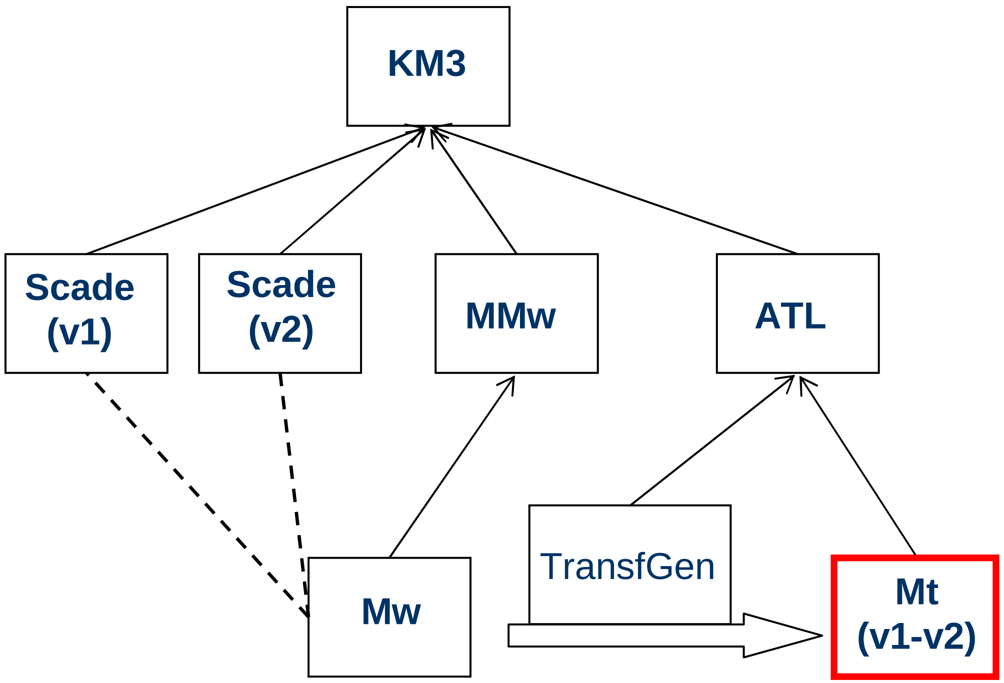




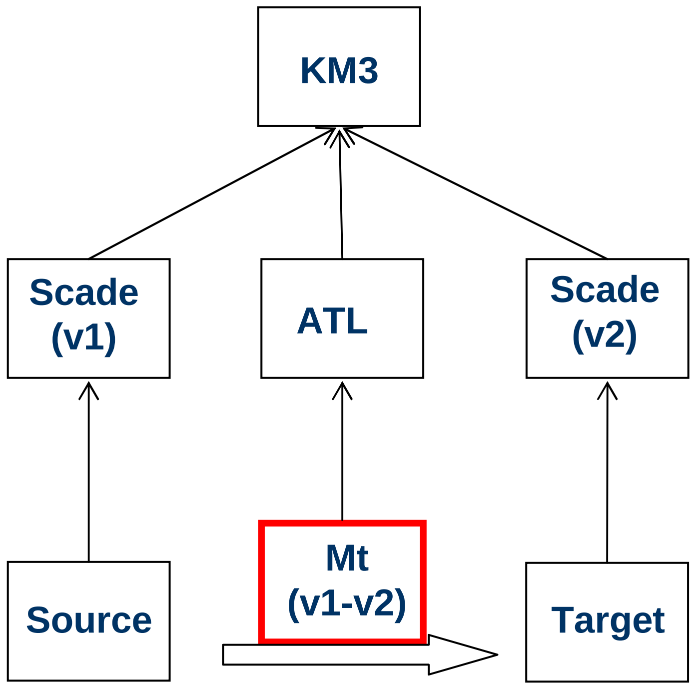
© Copyright IBM France 2010

# Comparison and migration (cont'd)

- Transformation generation




- Model migration





© Copyright IBM France 2010

# Scalability

- Scade

| | Elements | Classes | Attributes | References |
|---|---|---|---|---|
| **Version 1** | 449 | 106 | 105 | 231 |
| **Version 2** | 381 | 95 | 89 | 190 |

**Links:** 379 **Transformation:** 1030 **lines**

- Autosar

| | Elements | Classes | Attributes | References |
|---|---|---|---|---|
| **Version 2.0** | 4569 | 700 | 2262 | 1607 |
| **Version 2.1** | 6360 | 1020 | 3254 | 2086 |

**Links:** 3411 **Transformation:** 7990 **lines**

- Remarks
  - Identical executions
  - Optimized transformations
  - Graphical interface essential


© Copyright IBM France 2010

# Summary

- Matching
  - Several solutions
  - (semi) automatic creation of weaving models
  - Coupling of transformations and weaving models provide a generic framework
  - Necessary for real world model integration/migration scenarios
- Transformation production
  - Uses the result of a matching operation
  - Generates the final model transformation
  - Relies on higher-order transformations : difficult to write, but quite useful


© Copyright IBM France 2010

# General conclusions

- Relationships between model elements are ubiquitous
- Several solutions, different application domains, implementations, techniques
- Model weaving
  – Generic MDE solution
  – Simple core that is extended to a given application domain
- Others
  – TGG : transformation by example
  – Model link : simple Ecore2Ecore links
- Several use cases
  – Traceability
  – Model integration and comparison
  – Model merging
  – Annotation
  – Others


© Copyright IBM France 2010

# Q & A


© Copyright IBM France 2010