

MODEL MANAGEMENT

Dániel Varró

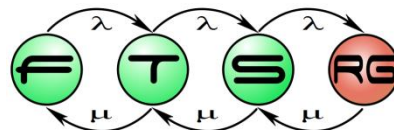
Ákos Horváth

Mostly Contributed by

M. Brambilla, J. Cabot and M. Wimmer

Model Driven Systems Development

Lecture 11





MORGAN & CLAYPOOL PUBLISHERS

Chapter #10

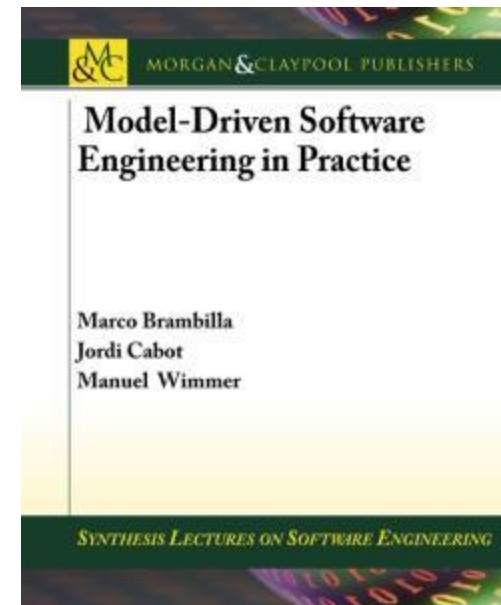
MANAGING MODELS

Teaching material for the book

Model-Driven Software Engineering in Practice

by Marco Brambilla, Jordi Cabot, Manuel Wimmer.

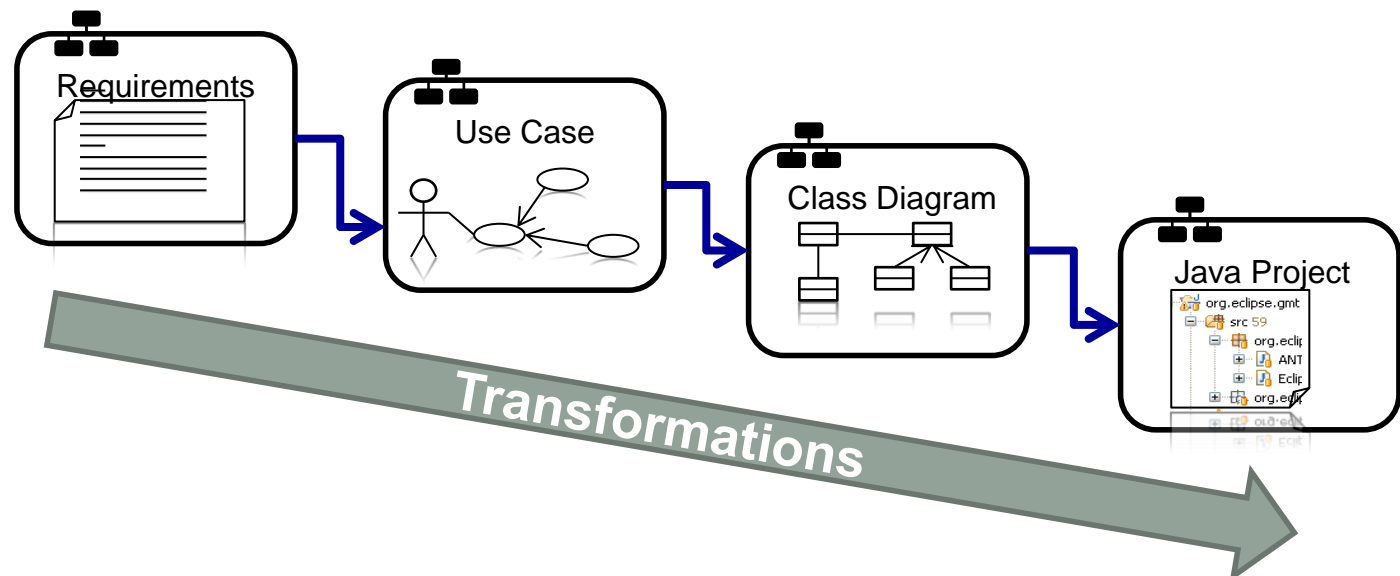
Morgan & Claypool, USA, 2012.



Motivation

Why Model managing?

- In MDE *everything is a model* but as important as that, *no model is an island*
- All modeling artefacts in a MDE project are interrelated. These relationships must be properly managed during the project lifecycle



Content

- Model interchange
- Model Persistence
- Model Comparison
- Model Versioning
- Model Co-Evolution
- Global Model Management
- Model Quality
- Collaborative modeling



MODEL INTERCHANGE



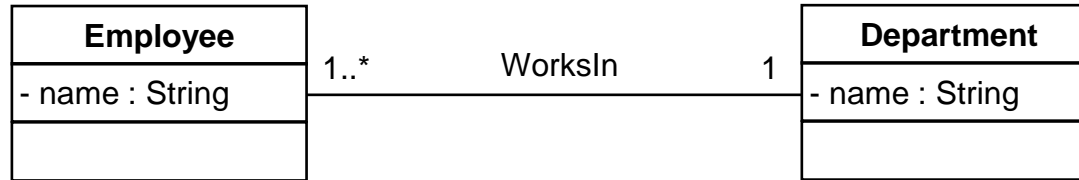
Model Once Open Everywhere

- There's a clear need to be able to exchange models among different modeling tools
 - In a perfect world, you'd be able to choose ToolA for specifying model, ToolB to check its quality, ToolC to execute it....
- We are still far away from this goal
- Solution attempt: XMI (XML Metadata Interchange), a standard adopted by OMG for serializing and exchanging UML and MOF models
- But each tools seems to understand the standard in a different manner



XML example

(simplified and partial versions of the actual XML files)



```
<packagedElement xmi : type="uml : Clas s " xmi : id=" c001 "
name="Employee">
<ownedAt t r i b u t e xmi : id=" a001 " name="name"/>
</packagedElement>
<packagedElement xmi : type="uml : Pr i m i t i v e T y p e " xmi : id="
t001 " name="St r i n g "/>
<packagedElement xmi : type="uml : Clas s " xmi : id=" c002 "
name="Department">
<ownedAt t r i b u t e xmi : id=" a002 " name="name" type=" t001 "/>
</packagedElement>
<packagedElement xmi : type="uml : As s o c i a t i o n " xmi : id="
as001 " name="WorksIn"
memberEnd=" e001 e002">
<ownedEnd xmi : id=" e001 " type=" c002 " a s s o c i a t i o n = "
as001"/>
<ownedEnd xmi : id=" e002 " name="" type=" c001 " a s s o c i a t i
o n = " as001">
<upperValue xmi : type="uml : Li t e r a l U n l i m i t e d N a t u r a l " xmi :
id="un001" value=""/>
</ownedEnd>
</packagedElement>
```

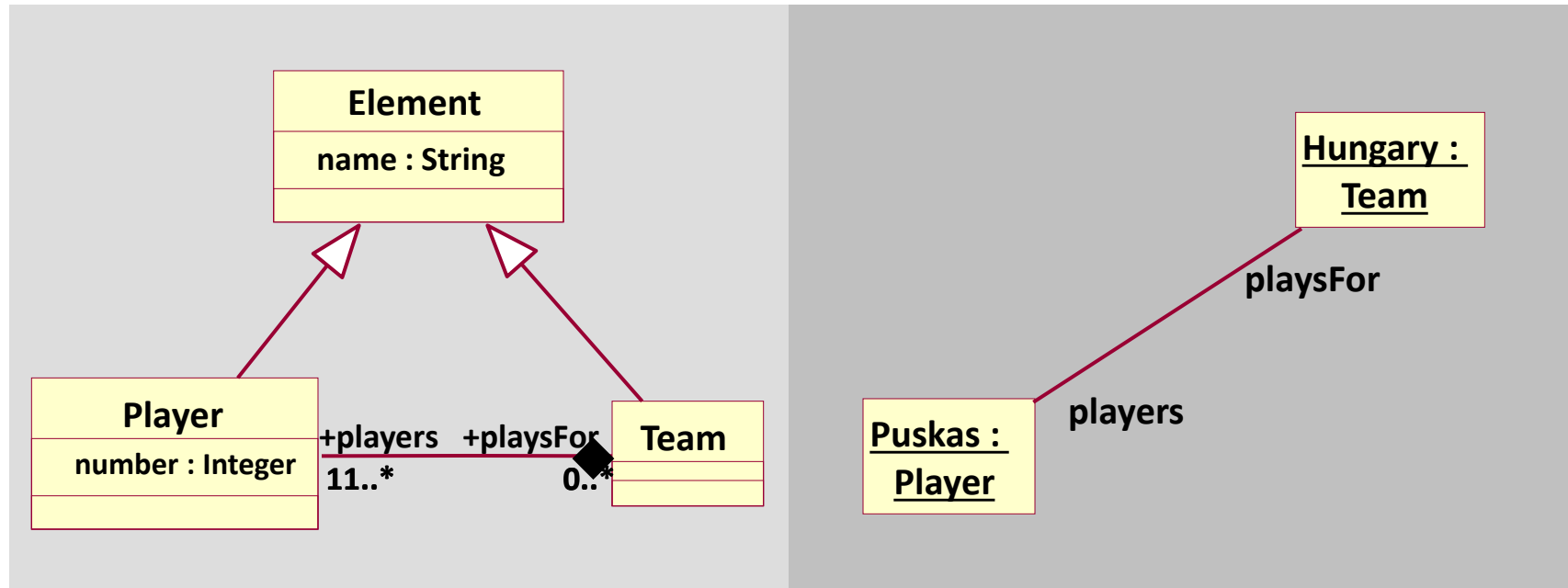
ECLIPSE

```
<UML: Clas s xmi . id = ' c001 '
name = 'Employee' visibility = ' public ' is S p e
c i f i c a t i o n = ' false ' isRoot = ' false '
i s L e a f = ' false ' isAbstract = ' false ' isAct i v e
= ' false ' >
<UML: C l a s s i f i e r . f e a t u r e >
<UML: At t r i b u t e xmi . id = ' a001 '
name = 'name' visibility = ' public ' is S p e c i f i
c a t i o n = ' false '
ownerScope = ' ins t a n c e ' c h a n g e a b i l i t y = '
c h a n g e a b l e ' t a r g e t S c o p e = ' ins t a n c e ' >
<UML: S t r u c t u r a l F e a t u r e . m u l t i p l i c i t y >
<UML: M u l t i p l i c i t y xmi . id = ' m001 ' >
<UML: M u l t i p l i c i t y . r a n g e >
<UML: M u l t i p l i c i t y R a n g e xmi . id = ' m r 0 0 1 '
lower = ' 1 ' upper = ' 1 ' />
</UML: M u l t i p l i c i t y . r a n g e >
</UML: M u l t i p l i c i t y >
</UML: S t r u c t u r a l F e a t u r e . m u l t i p l i c i t y >
</UML: Clas s >
```

ArgoUML



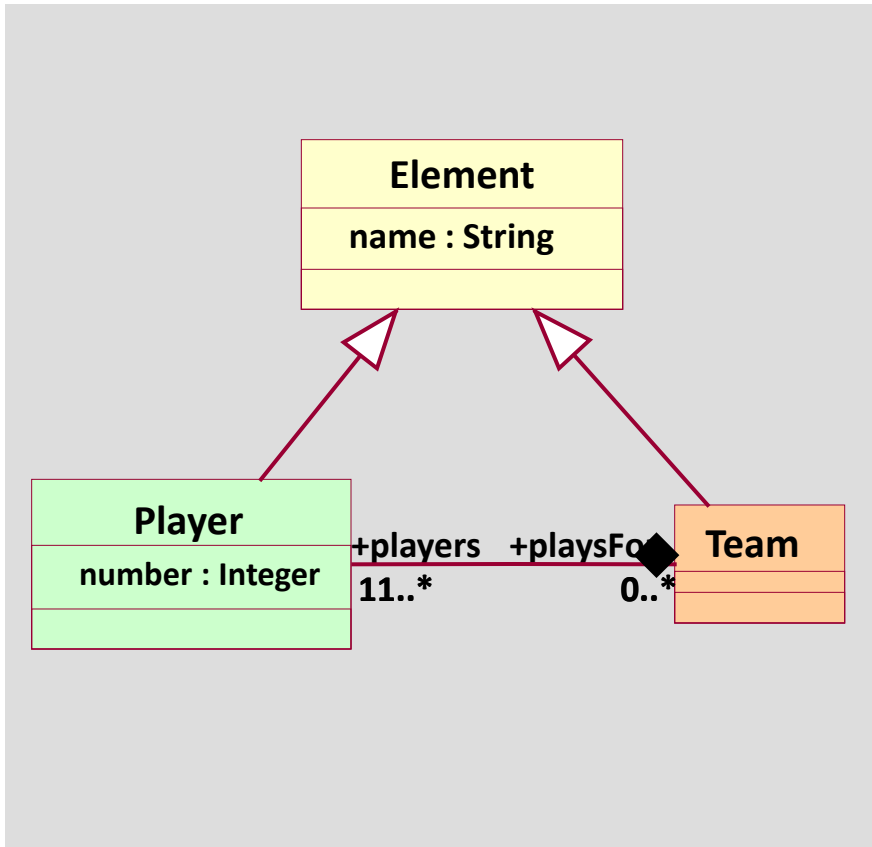
Example: metamodel and model



- Team metamodel

- Team model

Example: XMI 1.0 DTD



```
<!ELEMENT Team.players (Player)*>
```

```
<!ELEMENT Player.playsFor (Team)*>
```

```
<!ELEMENT Element.name
    (#PCDATA | XMI.reference)* >
```

```
<!ELEMENT Team (Element.name,
    XMI.extension*,
    Team.player) >
```

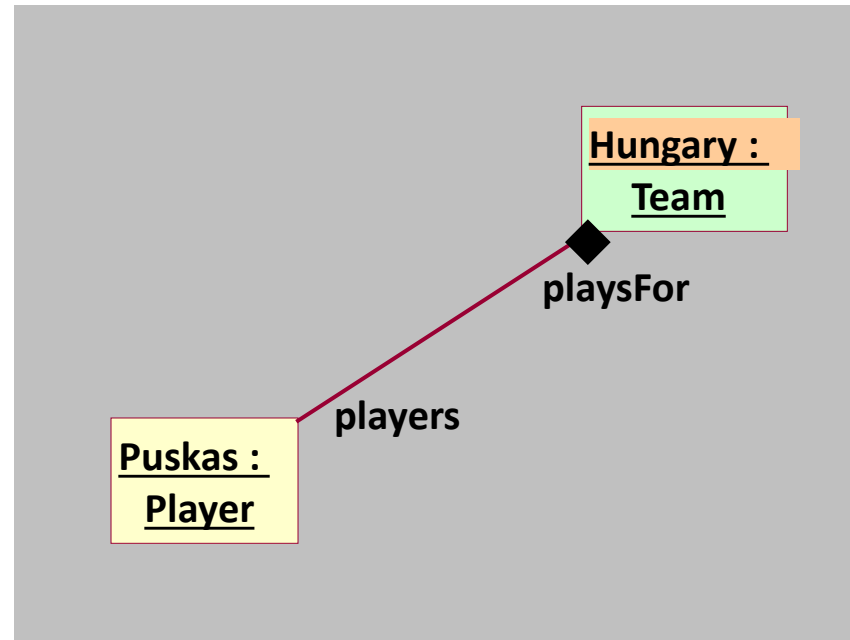
```
<!ATTLIST Team
    %XMI.element.att
    %XMI.link.att >
```

```
<!ELEMENT Player (Element.name,
    XMI.extension*,
    Team.playsFor) >
```

```
<!ATTLIST Player
    %XMI.element.att
    %XMI.link.att >
```

Example: XMI 1.0 document

```
<Team id='t1'>  
  <Element.name>  
    Hungary  
  </Element.name>  
  <Team.players>  
    <Player id='p1'>  
      <Element.name> Puskas  
    </Element.name>  
    <Player.number> 10  
    </Player.number>  
    <Player.playsFor  
      xmi.idref='t1' />  
    </Player>  
  </Team.players>  
</Team>
```



Example: XMI 1.1 document

```
<FB:Team id='t1' name='Hungary'>
```

```
<FB:Team.players>
```

```
<FB:Player id='p1'
```

```
  name='Puskas'
```

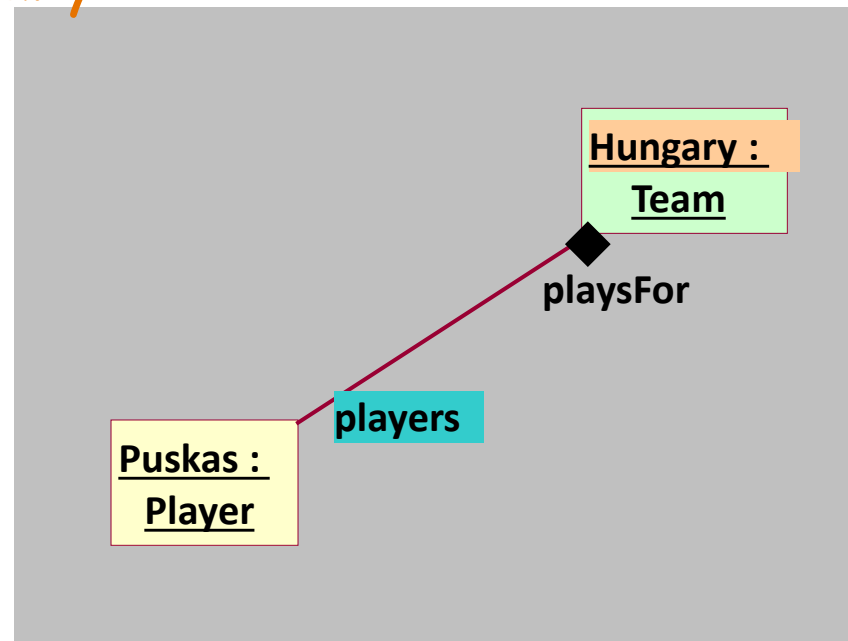
```
  number='10'
```

```
  playsFor='t1' />
```

```
</FB:Player>
```

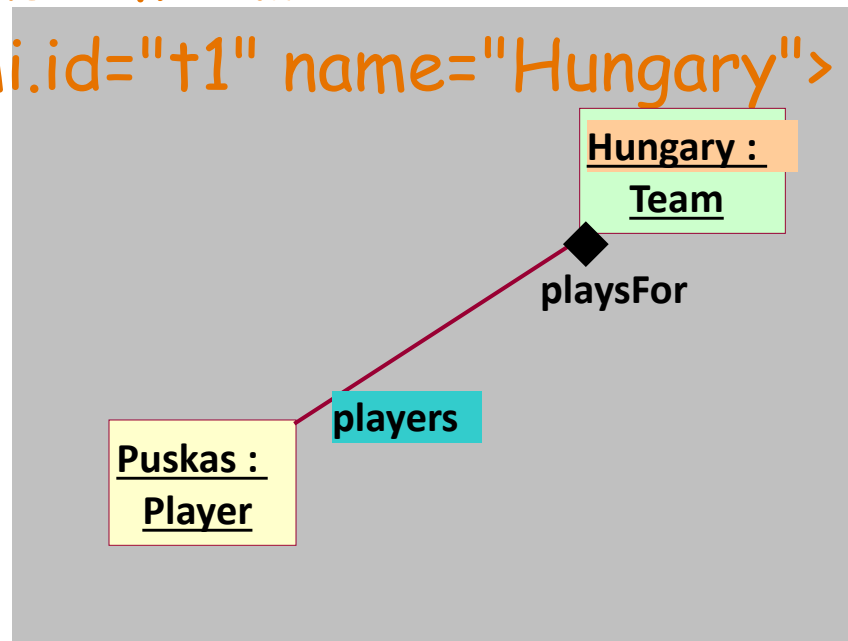
```
</FB:Team.players>
```

```
</FB:Team>
```



Example: XMI 2.0 document

```
<fb:Model xmlns:fb="...", xmlns:xmi="..."  
<teams xmi.type="Team" xmi.id="t1" name="Hungary">  
  <players xmi.id='p1'  
    name='Puskas'  
    number='10'  
    playsFor='t1' />  
</teams>  
</fb:Model>
```



Model Once Open Everywhere

Recent advances

- Model Interchange Working Group³ (MIWG) to enable the assessment of model interchange capability of modeling tools by comparing the vendor XMI exports for a test suite
- New The new Diagram Definition standard will allow to exchange not only the modeling content but also the graphical layout of the models



MODEL PERSISTENCE



Model Persistence

- Typically models are serialized in plain files, following the previous XMI format or any other proprietary XML format
- Doesn't work well with large models. Scalability issues
 - Loading the whole model in memory may not be an option
 - Random access strategies plus lazy loading (i.e. loading on demand) are needed



Model Persistence

Alternatives

- CDO (Connected Data Objects) Model Repository
 - Run-time persistence framework optimized for scalable query and transactional support for large object graphs.
 - Back-ends: object, NoSQL, and relational databases.
 - For relational databases, CDO relies on Teneo6, a Model-Relational mapping and runtime database persistence
- Pure NoSQL solutions: Morsa and MongoEMF. Both use MongoDB as backend.
- Newer alternatives aim at using the Cloud as model storage solution



MODEL COMPARISON



Model Comparison

- Comparing two models is a key operation in many model-management operations like model versioning
- Goal of model comparison is to identify the set of differences between two models
- These differences are usually represented as a model themselves, called a *difference model*



Model Comparison: Model matching

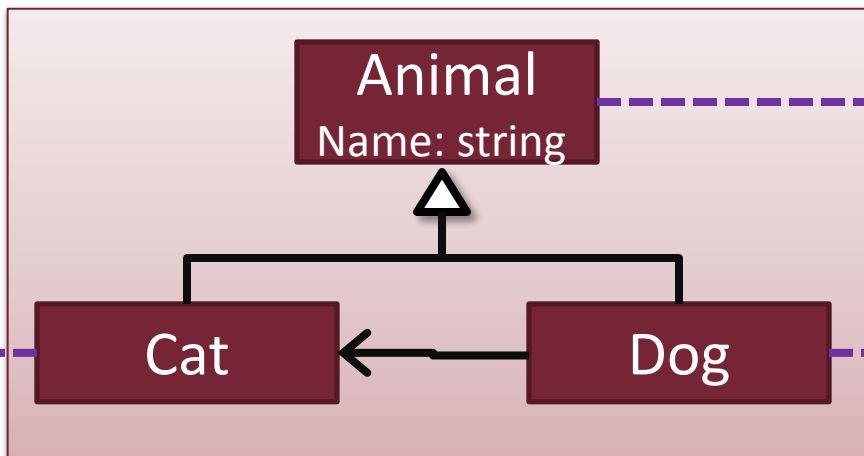
Phase 1 of a model comparison process

- Identify the common elements in the two models
- How do we establish which elements have the same identity?
 - Static identity: explicit id's annotating the elements
 - Signature identity: Identity based on the model element features (i.e. name, contained elements,...)
- Identity can be a probabilistic function (similarity matching)
- Works better if users redefine the concept of matching for specific DSLs (so that their specific semantic can be taken into account)

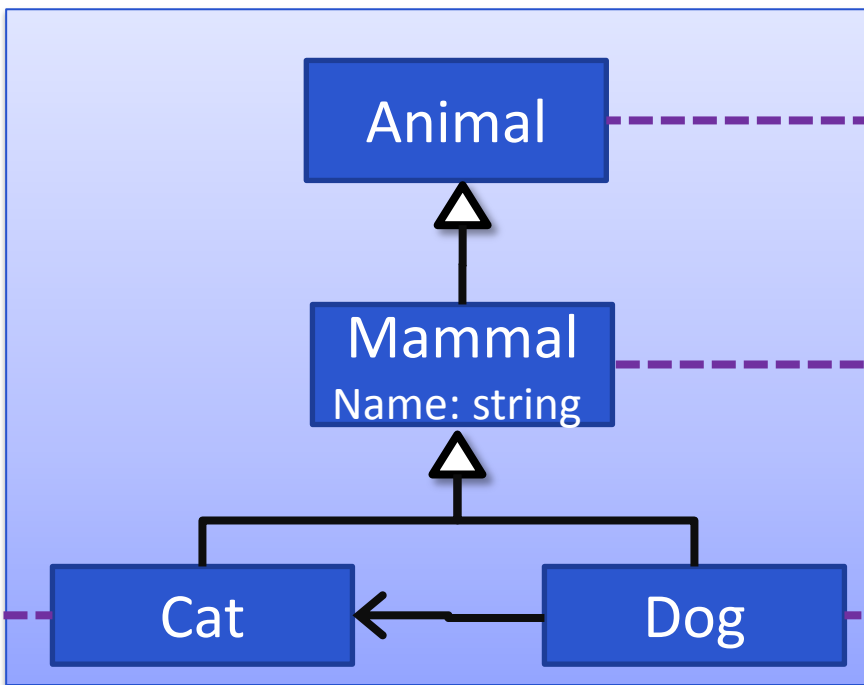
Model comparison =
Graph similarity problem



Example: Model Comparison



What is the best matching?



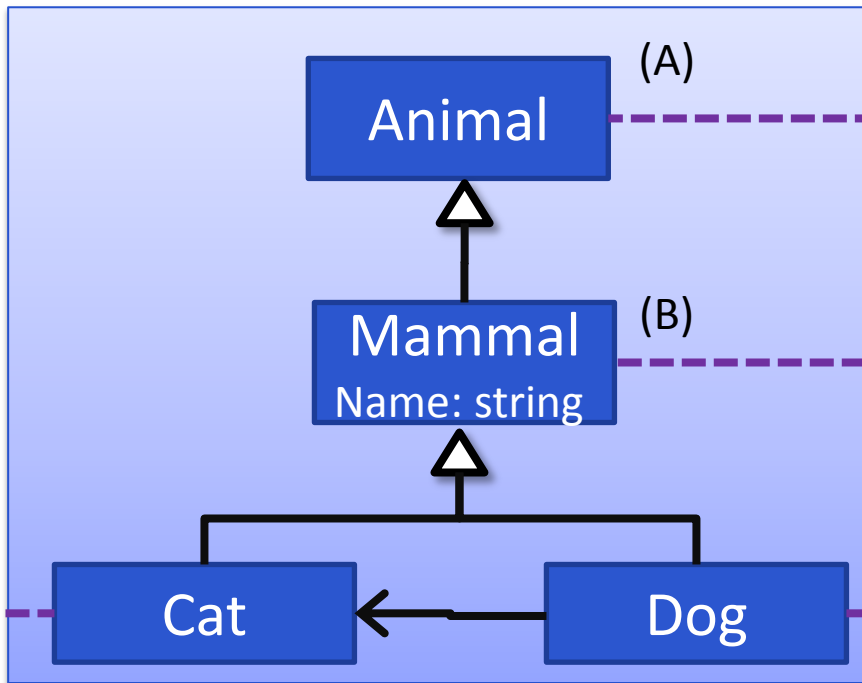
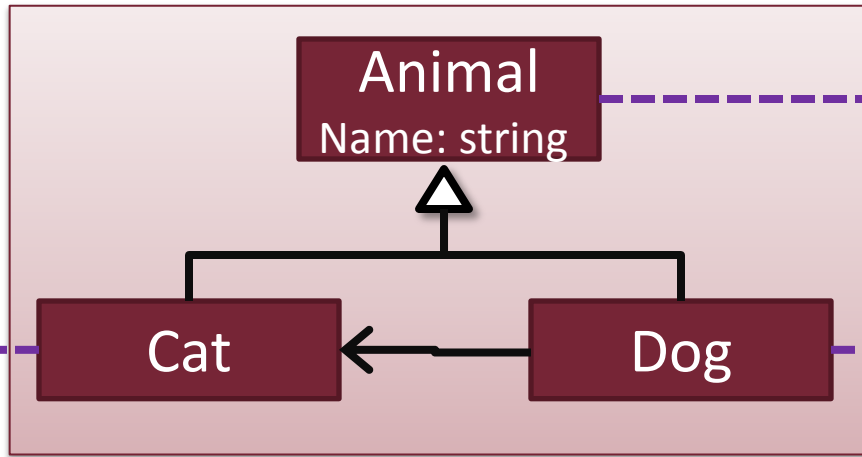
Model Comparison: Model differencing

Phase 2 of a model comparison process

- Matched elements are searched for differences
- A difference corresponds to an atomic add / delete / update / move operation executed on one of the elements
- These differences are collected and stored in the difference model



Example: Model Difference



- What is the difference?
- Matching (A)
 - Del Gen: Cat \rightarrow Animal
 - Del Gen: Dog \rightarrow Animal
 - Add Cls: Mammal
 - Add Gen: Mammal \rightarrow Animal
 - Add Gen: Cat \rightarrow Mammal
 - Add Gen: Dog \rightarrow Mammal
 - Move Att:
Name: Animal \rightarrow Mammal
- Matching (B)
 - Rename: Animal \rightarrow Mammal
 - Add Cls: Animal
 - Add Gen: Mammal \rightarrow Animal

Model Comparison tools

- EMF compare:
 - Most popular one
 - Generic comparison facilities for any kind of EMF model
 - Differences can be exported as a model patch
- SiDiff:
 - Mainly similarity-based matching
 - Adaptable to any graph-like model
- Epsilon Comparison Language:
 - Includes a DSL to enable the implementation of specialized higher-level changes
 - With it, high-level changes such as refactorings may be also detected



MODEL VERSIONING

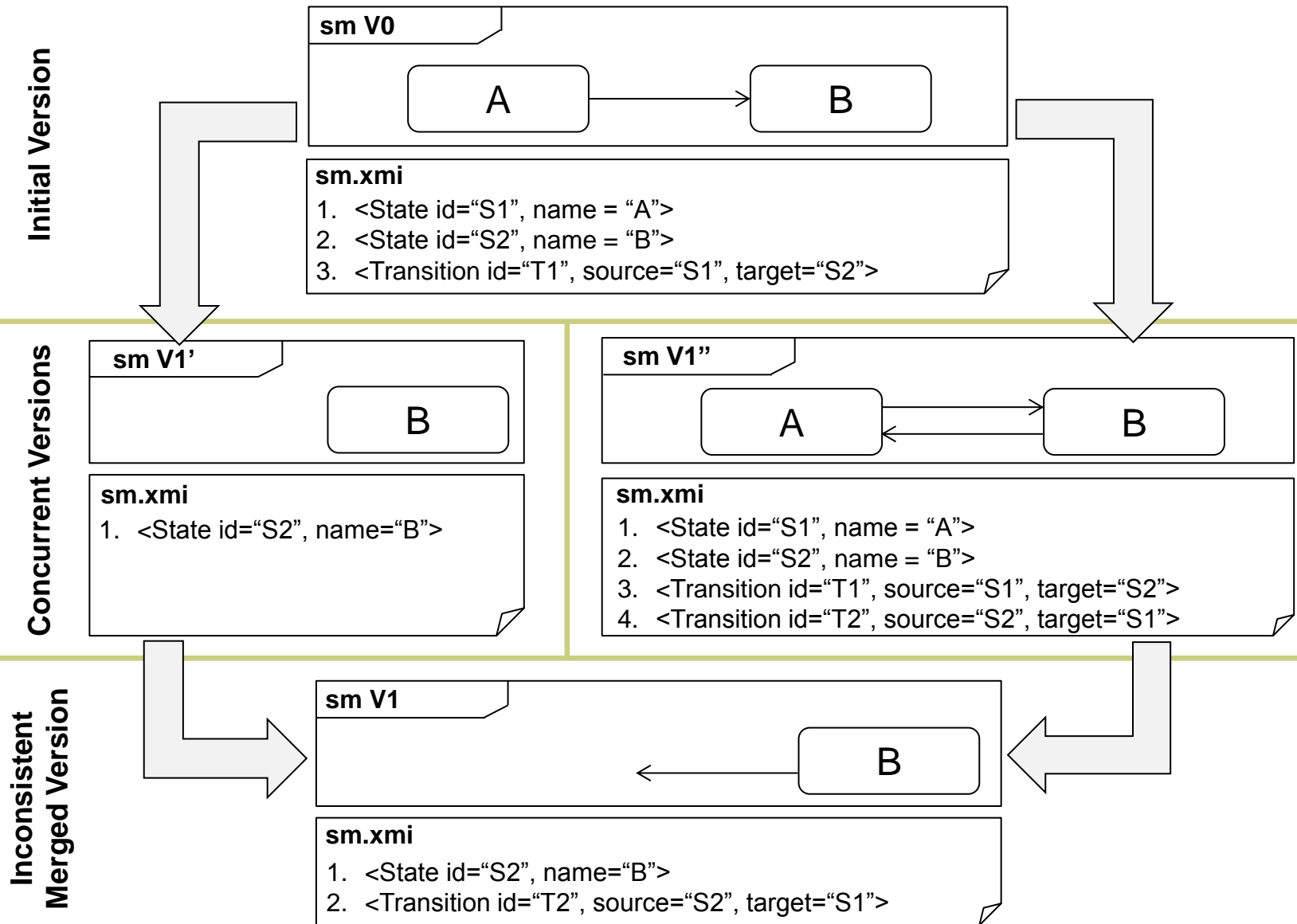


Model Versioning

- Programmers can't live without version control systems like SVN or GIT. Designers need the same for models
- VCSs help detect, manage and resolve conflicts arising when merging models
- Current VCSs are text-based. Using them to merge models may result in inconsistent results due to the graph-based semantics of models.



Model Versioning



Model Versioning

Tools

- Dedicated model-based VCSs are needed
- Some first attempts:
 - **EMFStore**: Official Eclipse project for model repositories. Follows the same SVN interaction protocol at the model-level
 - **AMOR** (Adaptable model versioning): Several conflict detection and resolution strategies possible. Visual merge process by means of annotations of conflicts directly on the graphical view of the models
 - CDO includes branching support for models
 - Epsilon Merging Language is a rule-based language for merging (heterogeneous) models
- Versioning of the graphical layout is still an open question (should moving a class two inches to the right count as a change?)



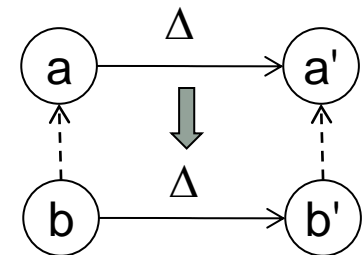
MODEL CO-EVOLUTION



Model Co-Evolution

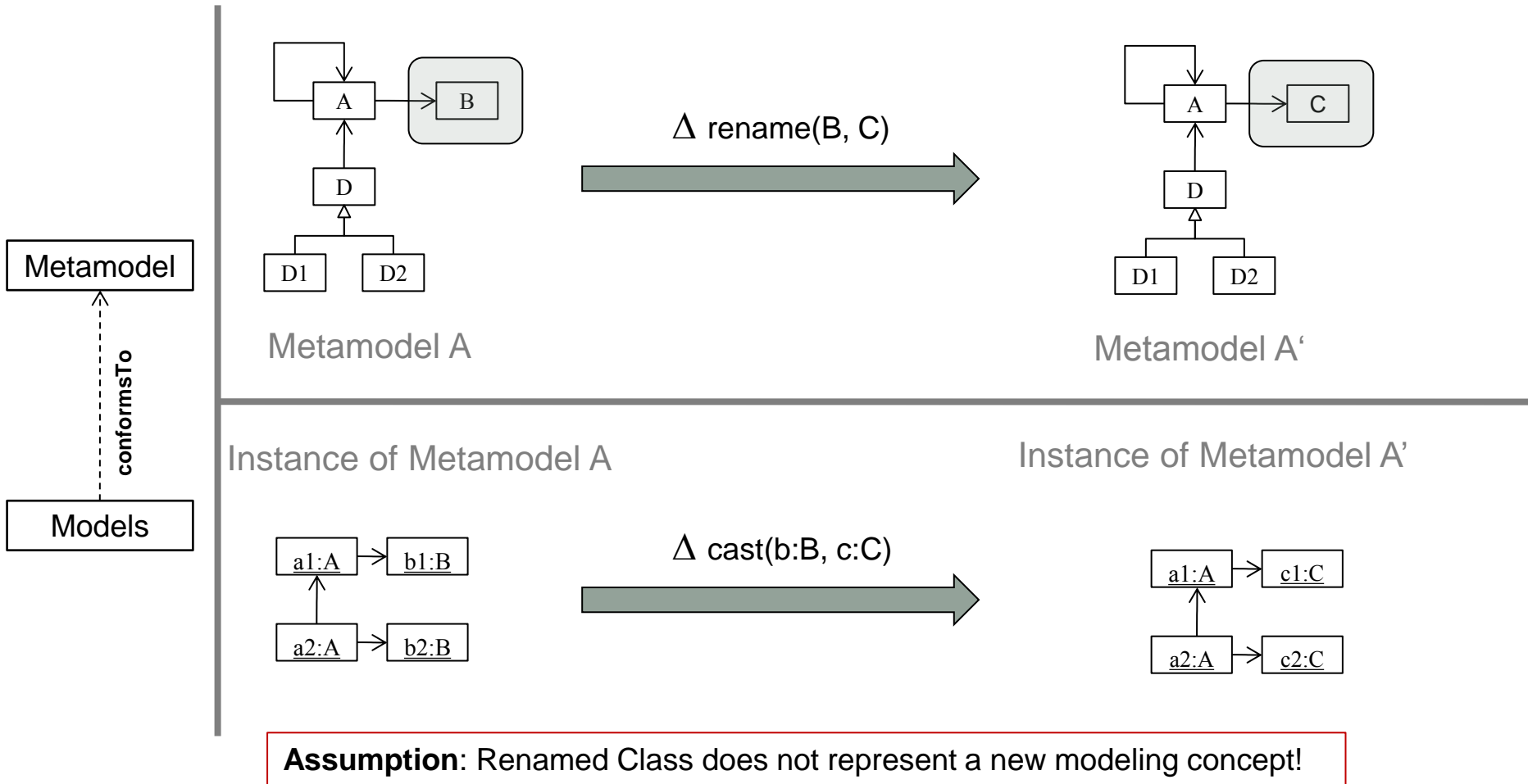
Tools

- Model versioning keeps track of the changes in a single modeling artefact but each change may affect many other related artefacts
- Co-Evolution in MDE
 - Co-evolution is the **change** of a model **triggered** by the **change** of a **related** model
 - Current View
 - Relationship: $r(a,b)$
 - $a \rightarrow a'$
 - $b \rightarrow b' \mid r(a',b')$
 - **Challenge: Relationship Reconciliation**
 - Current research focus is on one-to-one relationships:
 - Model / Metamodel evolution
 - Metamodel / Transformation evolution
 - ...



Model / Meta-model Co-evolution

Example



Model / Meta-model Co-Evolution

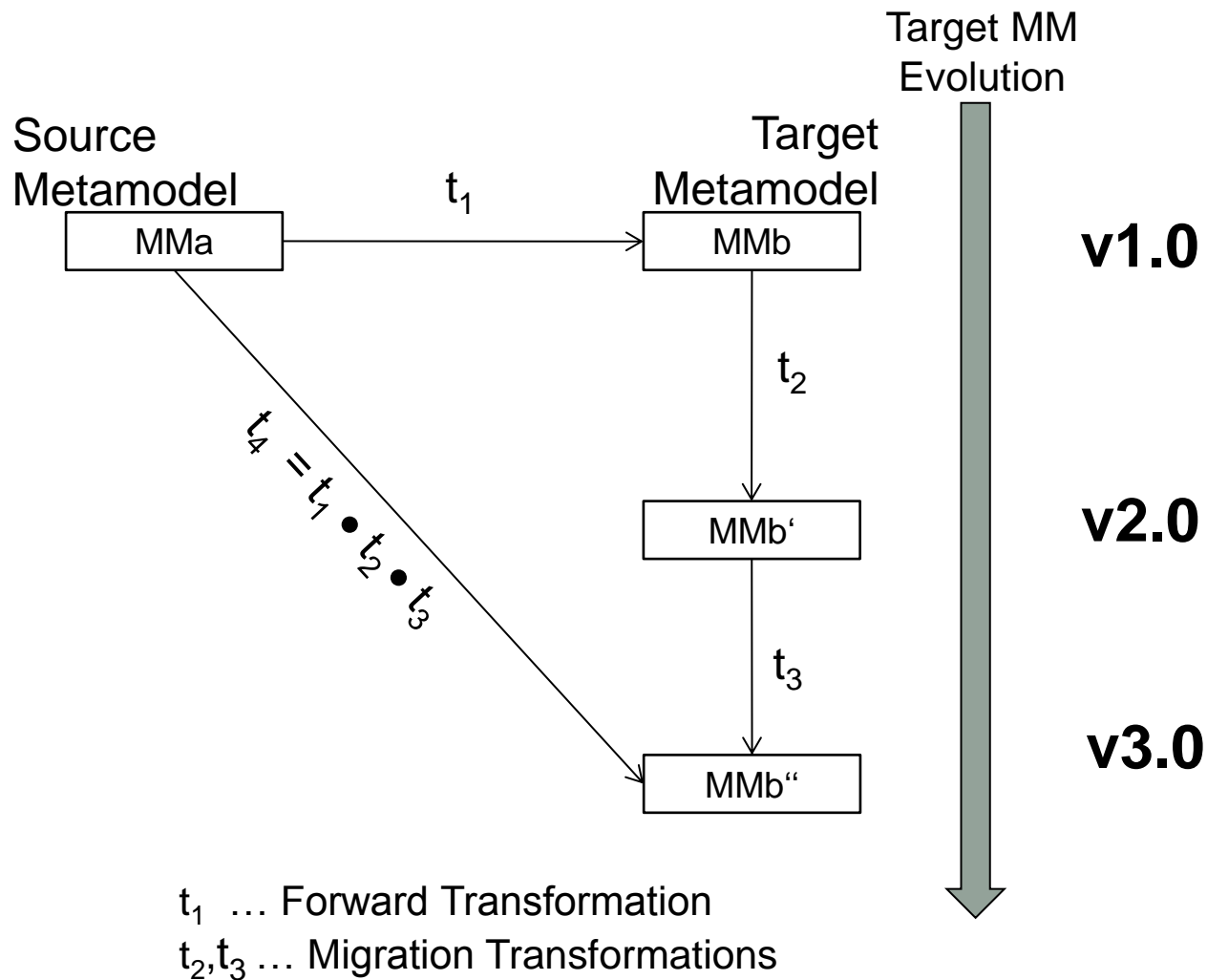
Process

- Classification of meta-model changes
 - Non-breaking operations: No need to migrate the models
 - Breaking and resolvable: Automatic migration of existing models is possible
 - Breaking and unresolvable: User intervention is necessary
- Tools like Edapt and Epsilon Flock can derive a migration transformation to adapt current models to the new metamodel structure when possible



Meta-model / Transformation co-evolution

Other co-evolution scenarios



GLOBAL MODEL MANAGEMENT

www.mdse-book.com



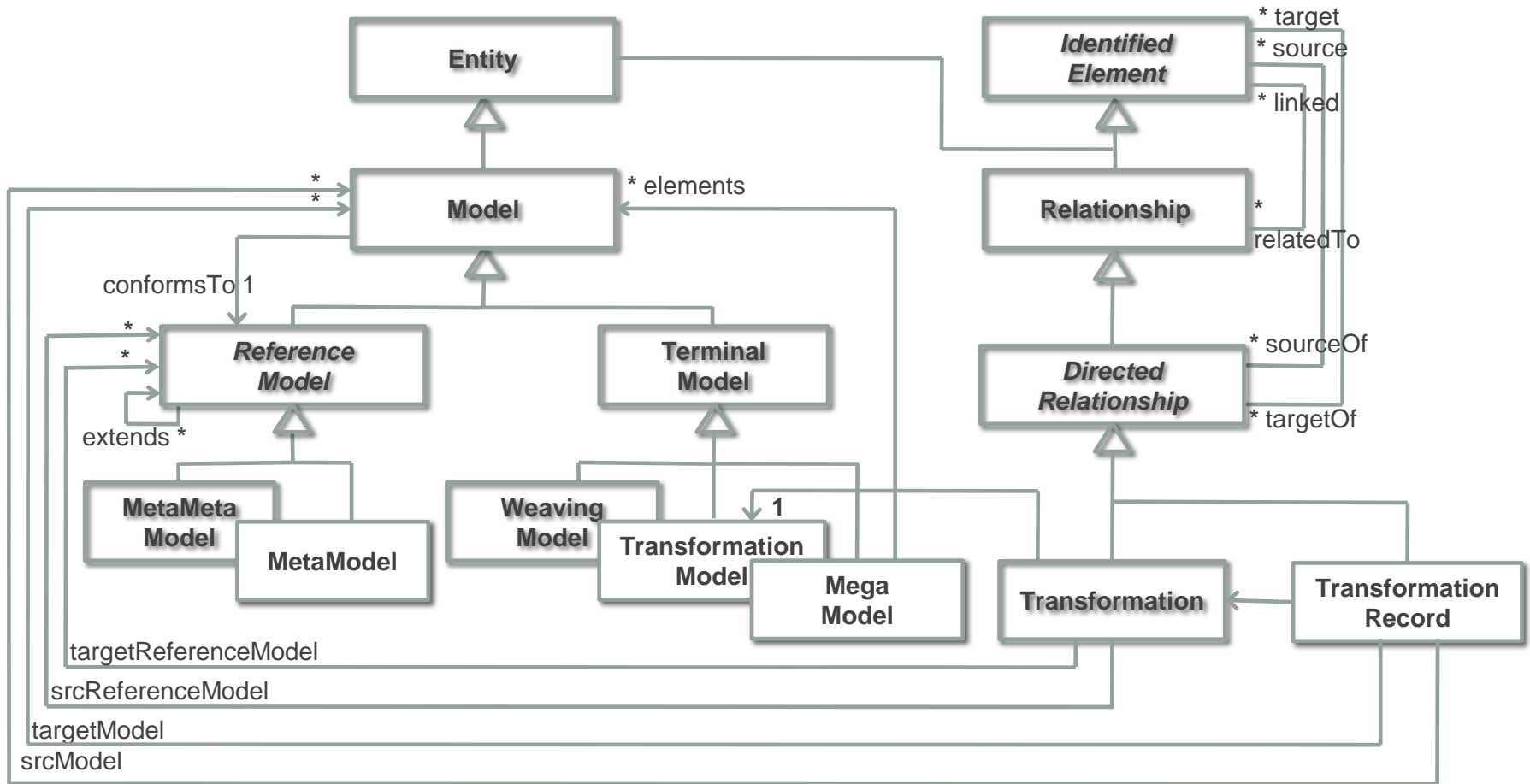
Global Model Management

- Model-based solution to the problem of managing all this *model ecosystem* appearing in any MDE project
- We represent with a model, the *megamodel*, all the models (and related artefacts like configuration files) and relationships in the ecosystem
- A megamodel can be viewed as a metadata repository for the project
- A megamodel is a model whose elements are in fact other models
- As a model, a megamodel can be directly manipulated using the same tools employed to manipulate “normal” models



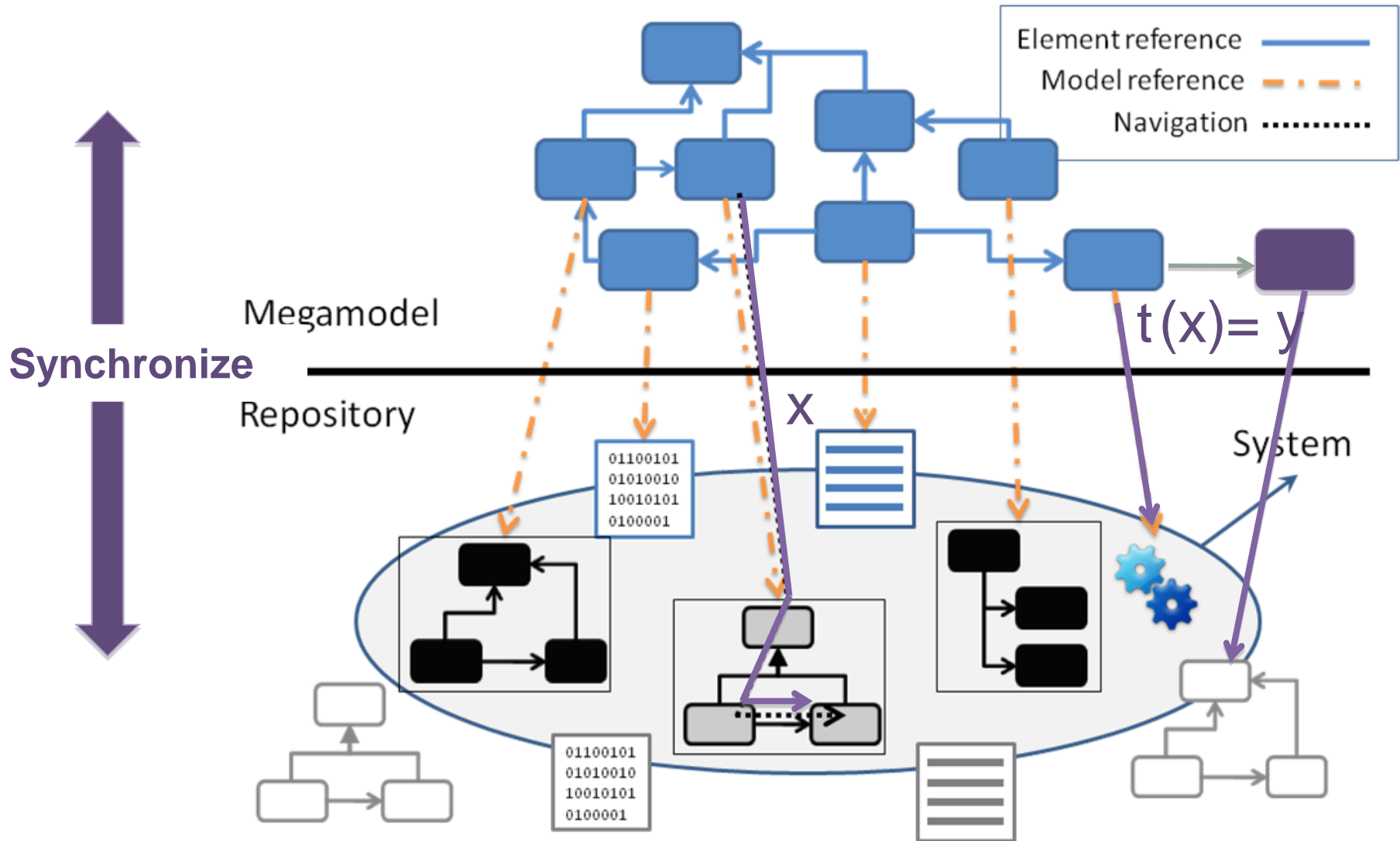
Global Model Management

The metamodel of a megamodel



Global Model Management

Using megamodels



Global Model Management

MoScript

- DSL to write model management scripts on megamodels
- It allows the automation of complex modelling tasks, involving several (batch) consecutive manipulations on a set of models.



Global Model Management

MoScript Examples

- Query operations

```
Model::allInstances()->any(m | m.identifier = 'SimpsonFamily')  
->allContents()->collect(el | el.name))
```

```
Collection {'Bart', 'Homer', 'Lisa', 'Maggie', 'Marge'}
```

- Model to Model transformations (M2M)

```
1 let j2dNet : Transformation = Transformation::allInstances()  
2   ->any(t | t.identifier = 'j2dNet')  
3 in  
4  
5 Model::allInstances()  
6   ->select(m | m.conformsTo.kind = 'Java'))  
7   ->collect (jModel | j2dNet.applyTo(jModel))
```

```
TransformationRecord::allInstances()->collect(tr | tr.run())
```



COLLABORATIVE MODELING

www.mdse-book.com



Collaborative modeling

- Modeling is by definition a team activity
- Offline synchronization of models can be handled using the model versioning tools seen before
- Online collaborative modeling (several users updating the same model at the same time) is more problematic
 - Based on a short transaction model where changes are immediately propagated to everybody
 - Very lightweight conflict management mechanisms (e.g. voluntary locking)
 - Conflict resolution by explicit consensus among all parties



Collaborative modeling

Tools

- EMFCollab
 - Master copy in a server. Slave copy in each client.
 - Commands to modify the models are serialized and distributed across the network
- SpacEclipse-CGMF
 - Integration of collaborative functionality in GMF-based editors
 - This functionality can be generated as part of the generation of the own GMF editor and workspace
- Dawn
 - Subproject of CDO
 - Aimed at providing collaborative access to GMF diagrams.





MORGAN & CLAYPOOL PUBLISHERS

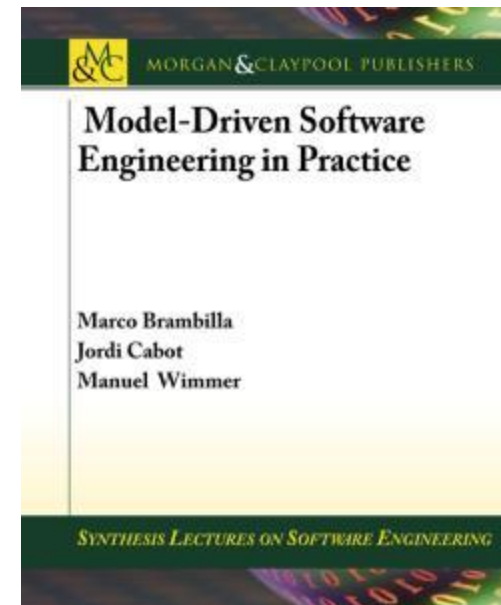
MODEL-DRIVEN SOFTWARE ENGINEERING IN PRACTICE

Marco Brambilla,
Jordi Cabot,
Manuel Wimmer.
Morgan & Claypool, USA, 2012.

www.mdse-book.com

www.morganclaypool.com

or buy it at: www.amazon.com



www.mdse-book.com