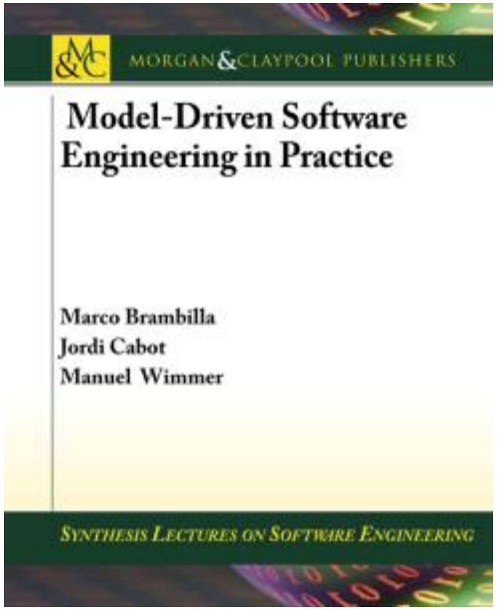
MORGAN & CLAYPOOL PUBLISHERS

Chapter #10

MANAGING MODELS

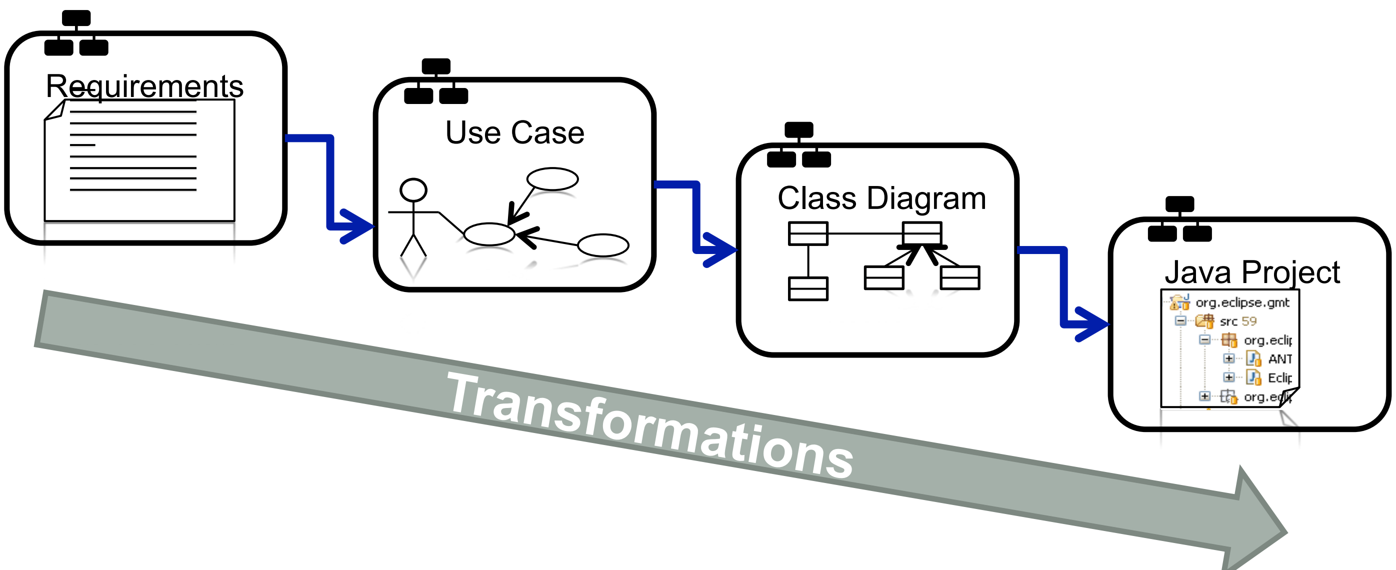
Teaching material for the book
Model-Driven Software Engineering in Practice
by Marco Brambilla, Jordi Cabot, Manuel Wimmer.
Morgan & Claypool, USA, 2012.



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….

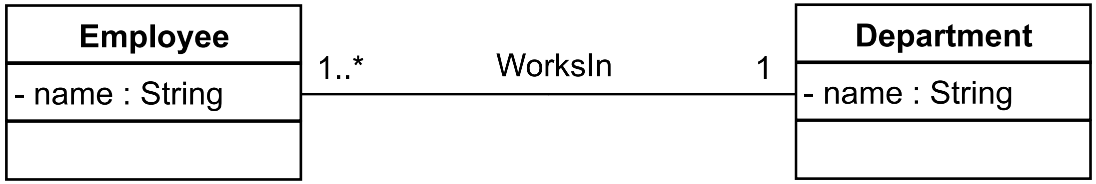
- We are still far away from this goal

- 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

XMI example

(Simplified and partial versions of the actual XMI files)




```
<packagedElement xmi:type="uml:Class" xmi:id="c001"
name="Employee">
<ownedAttribute xmi:id="a001" name="name"/>
</packagedElement>
<packagedElement xmi:type="uml:PrimitiveType" xmi:id=" t001"
name="String "/>
<packagedElement xmi:type="uml:Class" xmi:id="c002"
name="Department">
<ownedAttribute xmi:id="a002" name="name" type="t001"/>
</packagedElement>
<packagedElement xmi:type="uml:Association" xmi:id="as001"
name="WorksIn“ memberEnd="e001 e002">
<ownedEnd xmi:id="e001" type="c002" association="as001"/>
<ownedEnd xmi:id="e002" name="" type="c001" association=
"as001">
<upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="un001"
value=""/>
</ownedEnd>
</packagedElement>
```

ECLIPSE

```
<UML:Class xmi.id = 'c001'
name = 'Employee' visibility = 'public' isSpecification =
'false' isRoot = 'false'
isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
<UML:Classifier.feature>
<UML:Attribute xmi.id = 'a001'
name = 'name ' visibility = 'public' isSpecification =
'false‘ ownerScope = 'instance' changeability =
'changeable' targetScope = 'instance'>
<UML:StructuralFeature.multiplicity>
<UML:Multiplicity xmi.id = 'm001'>
<UML:Multiplicity.range>
<UML:MultiplicityRange xmi.id = 'mr001 '
lower = '1' upper = '1'/>
</UML:Multiplicity.range>
</UML:Multiplicity>
</UML:StructuralFeature.multiplicity>
</UML:Class>
```

ArgoUML

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 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: 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

Model Comparison tools

- EMF Compare
  - 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
  - Includes a DSL to enable the implementation of specialized higher-level changes
  - With it, high-level changes such as

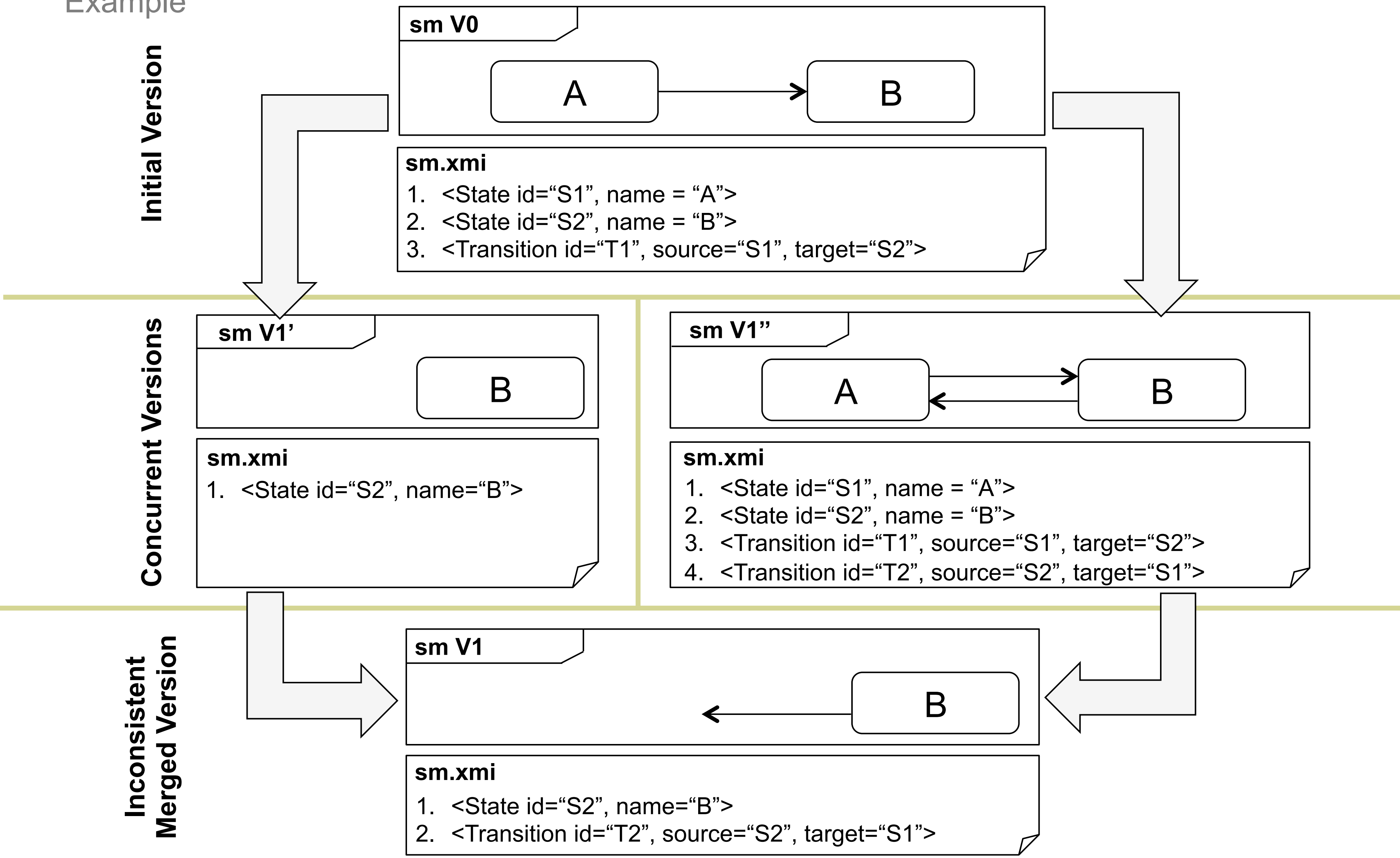
MODEL VERSIONING

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

Example



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

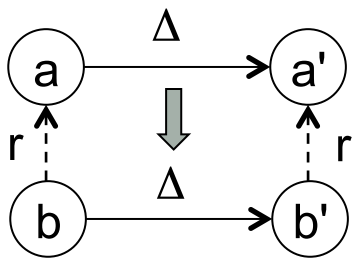
- Versioning of the graphical layout is still an open question (should moving a class two inches to the right count as a change?)

MODEL CO-EVOLUTION

Model Co-Evolution

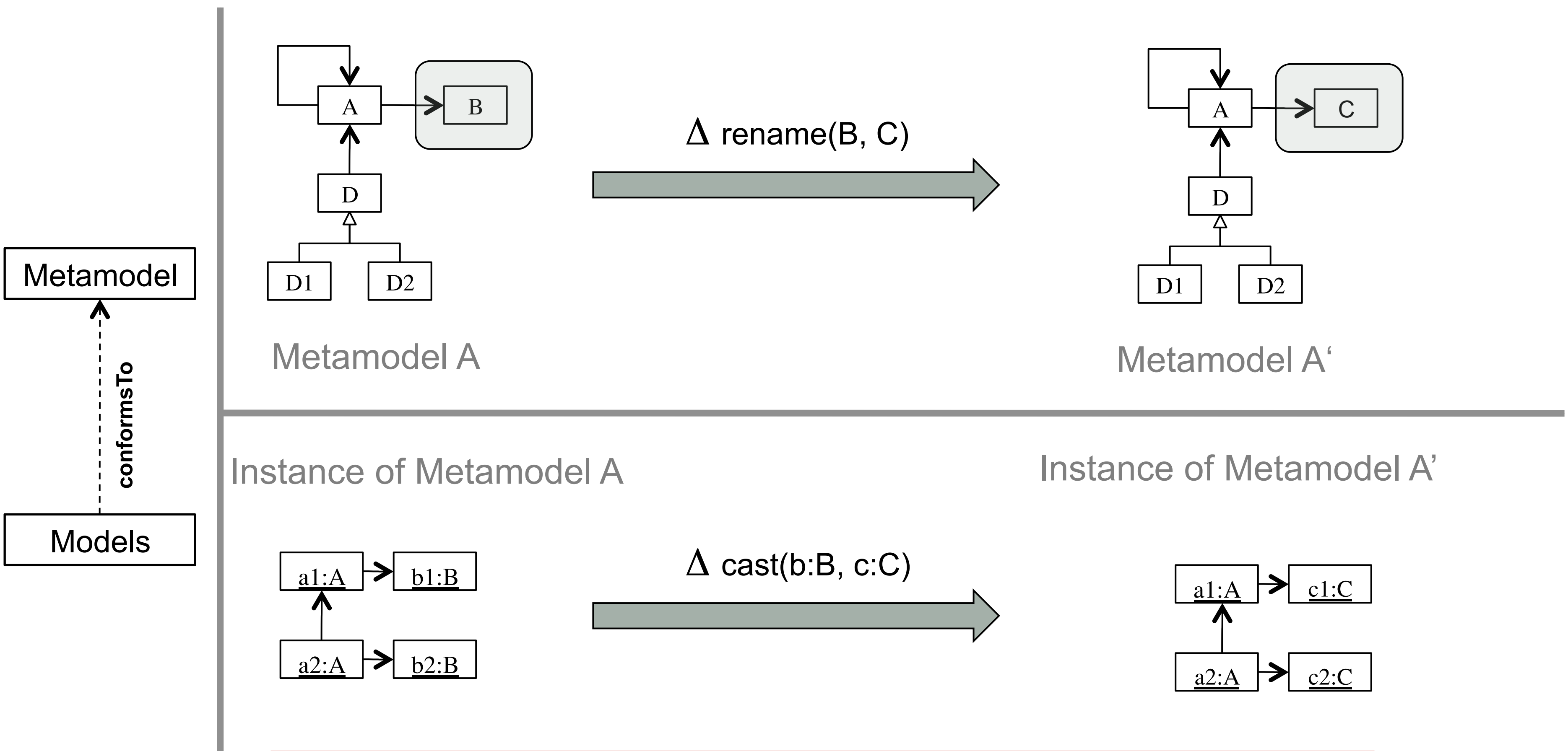
- Model versioning keeps track of the changes in a single modeling artefact but each change may affect many other related artefacts

- 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 → a'
    - b → b' | r(a',b')
    - **Challenge: Relationship Reconciliation**
  - Current research focus is on one-to-one relationships:
    - Model / Metamodel evolution
    - Metamodel / Transformation evolution
    - …



Model / Metamodel Co-Evolution

Example



Assumption: Renamed Class does not represent a new modeling concept!

Model / Metamodel Co-Evolution

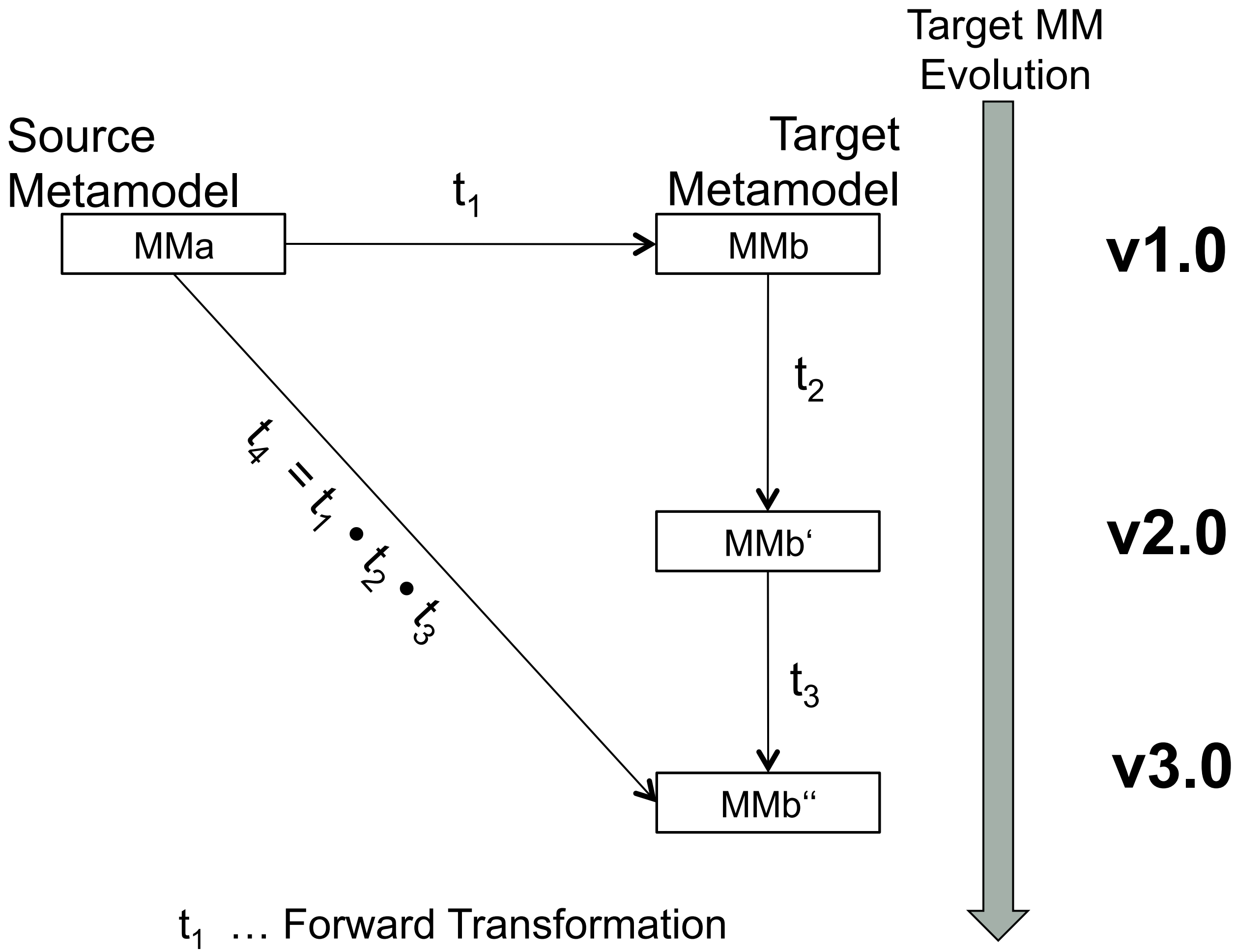
Process

- 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

Metamodel / Transformation Co-Evolution

Other Co-Evolution Scenarios



t_1 … Forward Transformation

t_2,t_3 … Migration Transformations

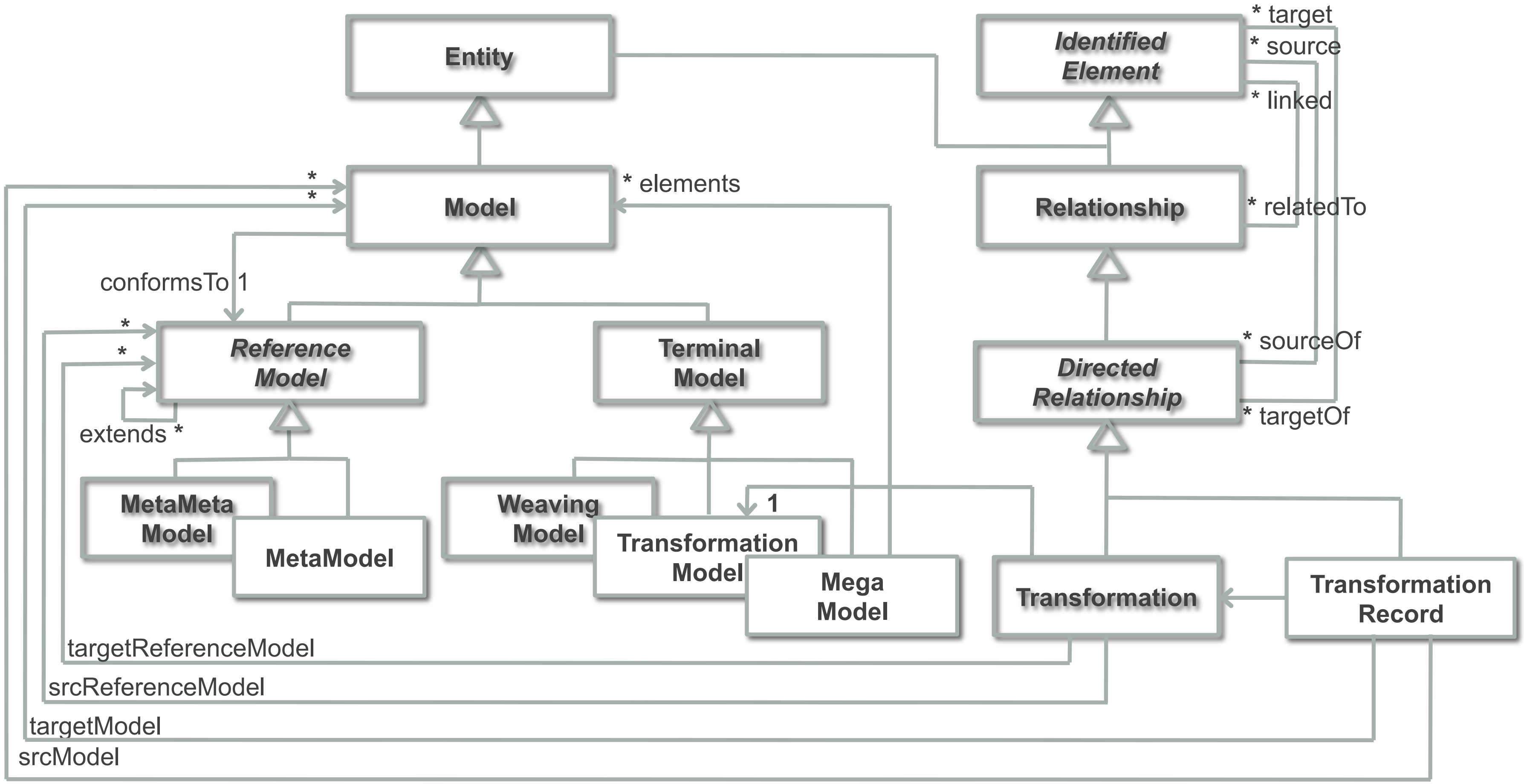
GLOBAL MODEL MANAGEMENT

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

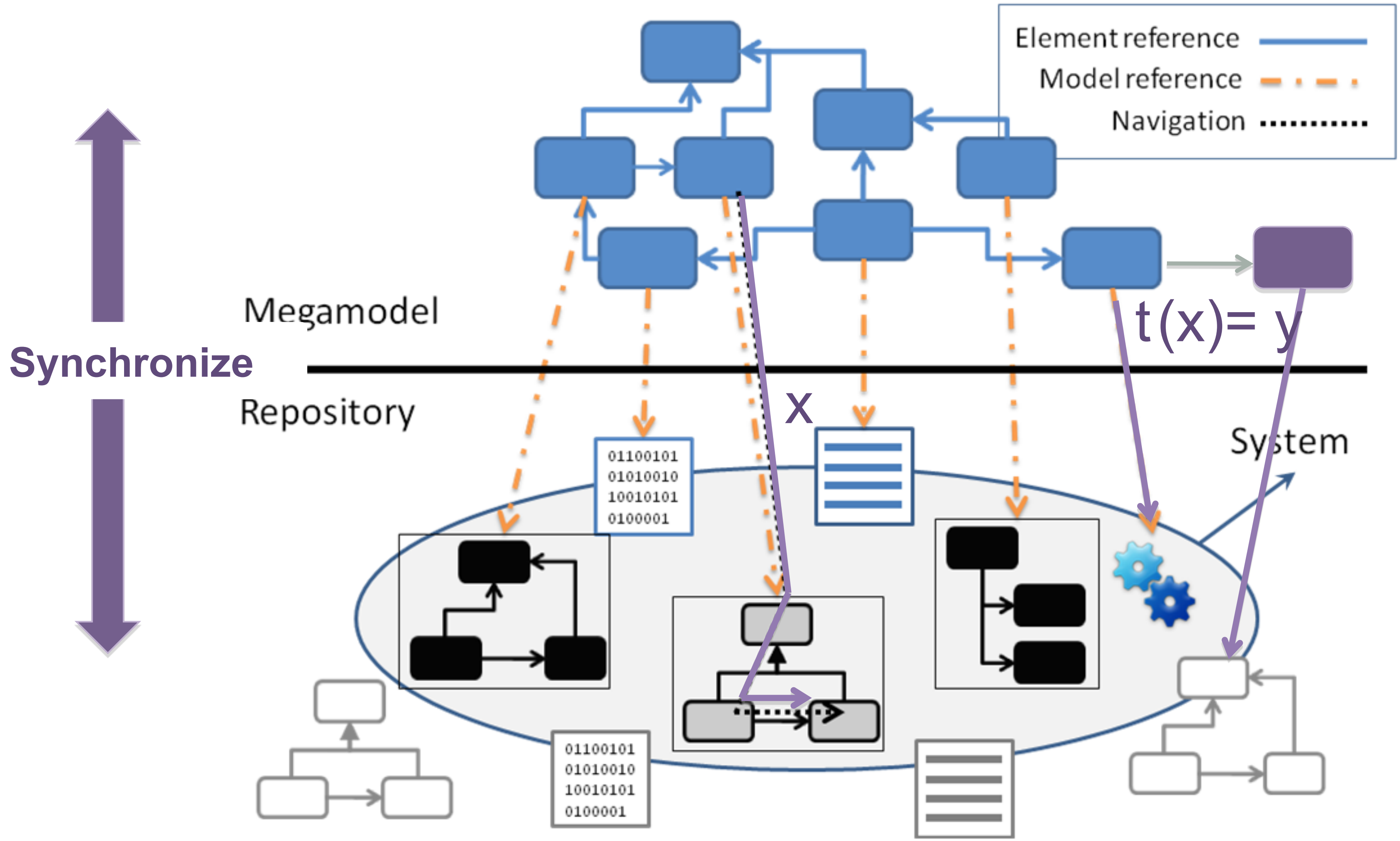
Global Model Management

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.indentifier = 'SimpsonFamily')
                         ->allContents()->collect(el | el.name))
```

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

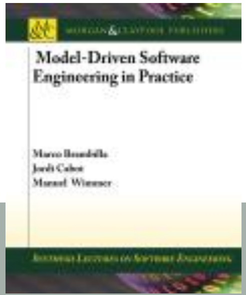
- Model to Model transformations (M2M)

```
let j2dNet : Transformation = Transformation::allInstances()
       ->any(t | t.identifier = 'j2dNet')
in

Model::allInstances()
       ->select(m | m.conformsTo.kind = 'Java'))
       ->collect (jModel | j2dNet.applyTo(jModel))
```

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

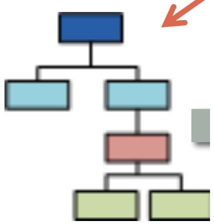

MODEL QUALITY

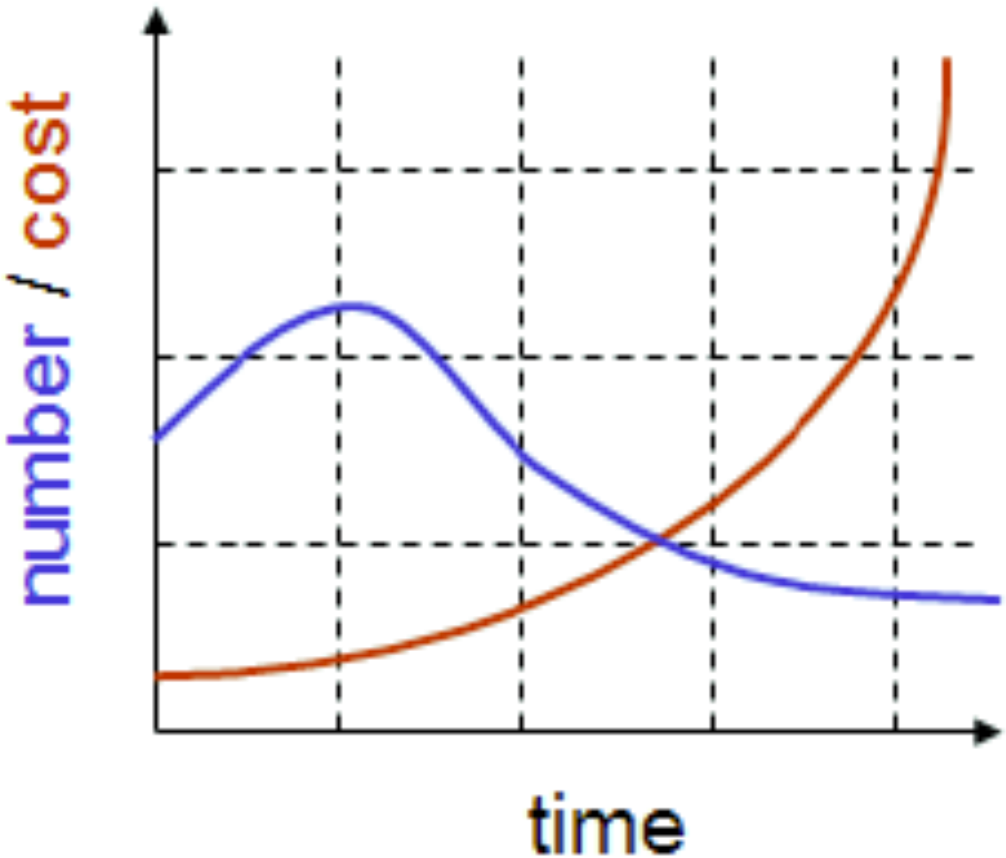


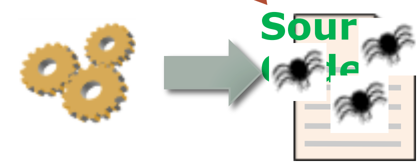
Motivation

MDE-l … rocess

Original
model







M … del
T … nsformation

E … rors in
t

Model Quality

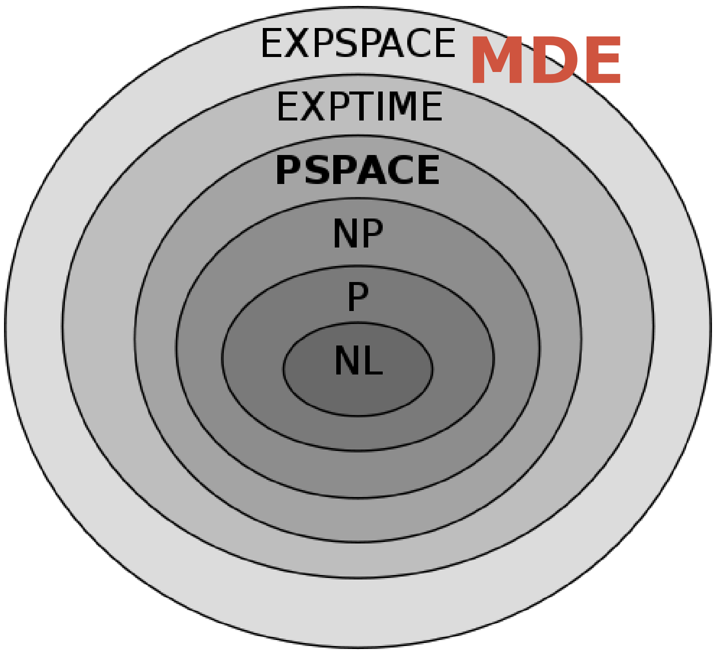
- Modeling tools only check for well-formedness
  - Is a model conforming to its metamodel, i.e., is a model a valid instance of its metamodel?

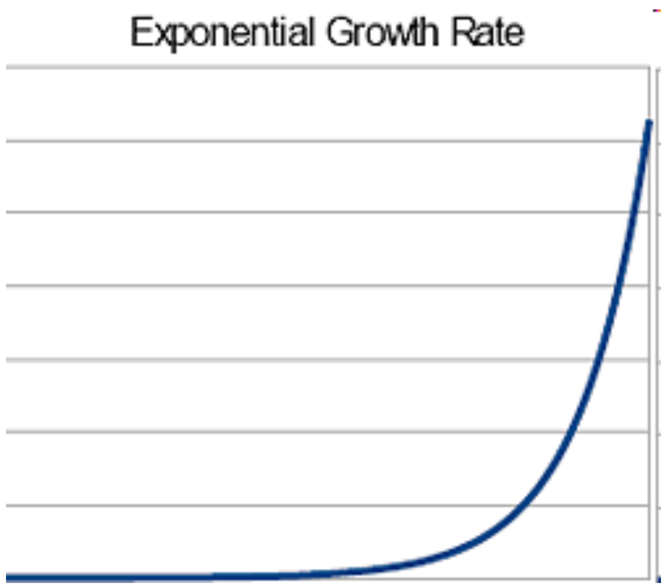
- But this is just the tip of iceberg when it comes to evaluating the quality of a model. There are many other properties to verify:
  - For static models: satisfiability, liveliness, redundancy, subsumption …
  - For dynamic models: absence of deadlocks, reachability,…

- Evaluation of these properties can be done through formal model verification or testing

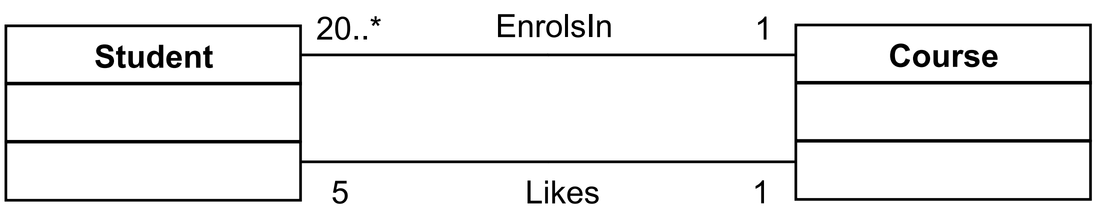
Example Property: Satisfiability

- A model is satisfiable if it is possible to create a valid instantiation of that model. A instantiation is valid if it satisfies all model constraints
- More difficult than it seems



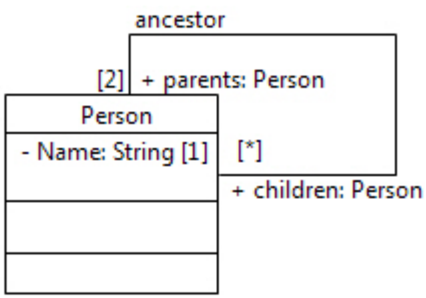


Example of Unsatisfiability (1)

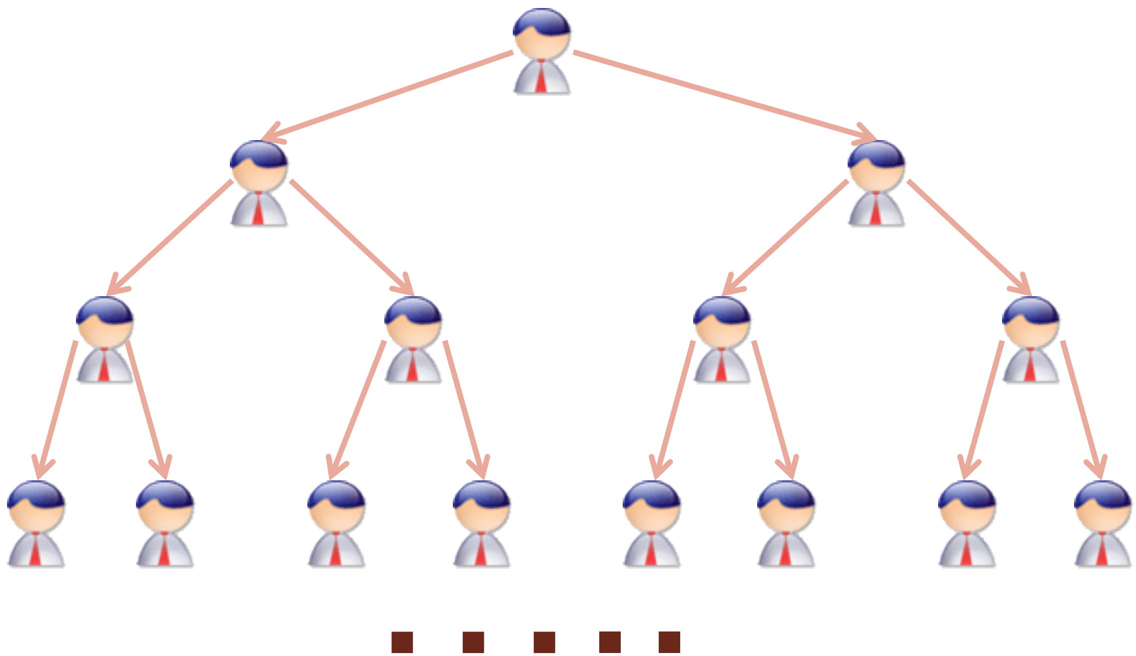


- Due to EnrolsIn |student|>=20*|course|
- Due to Likes |student|=5*|course|

Example of Unsatisfiability (2)



And no person is his/
her own ancestor



Strong Satisfiability

Typical Formal Verification Approach

EMF/UML model

1. Class diagram / metamodel
2. OCL constraints

→ Property?

Translate

Constraint Satisfaction Problem / SAT SMT / …

1. Variables – basic types + struct/list
2. Domains – finite
3. Constraints – Prolog
4. Property -> Additional Constraint

Solve → Solution?

Deduce

Ex: EMFtoCSP tool

Testing models

Derive tests from your models

- Same as we test code, models can also be tested
  - Tools like USE can create snapshots of a system and evaluate OCL constraints on them to test the OCL expressions
- Specially useful for dynamic models & operations like model transformations
  - E.g., we may want to check a transformation generates a valid output model every time a valid input model is provided
- Several black-box and white-box techniques for model testing have been proposed

COLLABORATIVE MODELING

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