Vienna University of Technology

Introduction to Model Versioning

SFM-12: MDE

June 22, 2012

Petra Brosch, Gerti Kappel,
Philip Langer, Martina Seidl,
Konrad Wieland, and Manuel Wimmer

Business Informatics Group

Institute of Software Technology and Interactive Systems
Vienna University of Technology

Favoritenstraße 9-11/188-3, 1040 Vienna, Austria
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896
office@big.tuwien.ac.at, www.big.tuwien.ac.at

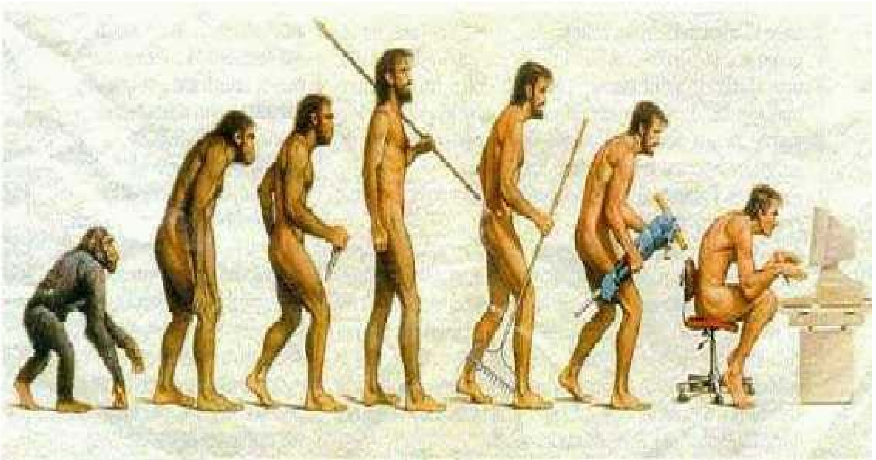
Outline

- **Context of Model Versioning**
- Foundations of Model Versioning
- Conflict Categorization
- Adaptable Model Versioning
- Future Challenges

Evolution

Definition

- Evolution in the broader sense taken from http://www.merriam-webster.com/dictionary/evolution
  - A **process** of **change** in a certain direction
  - A process of continuous change from a **lower**, simpler, or worse to a **higher**, more complex, or better state



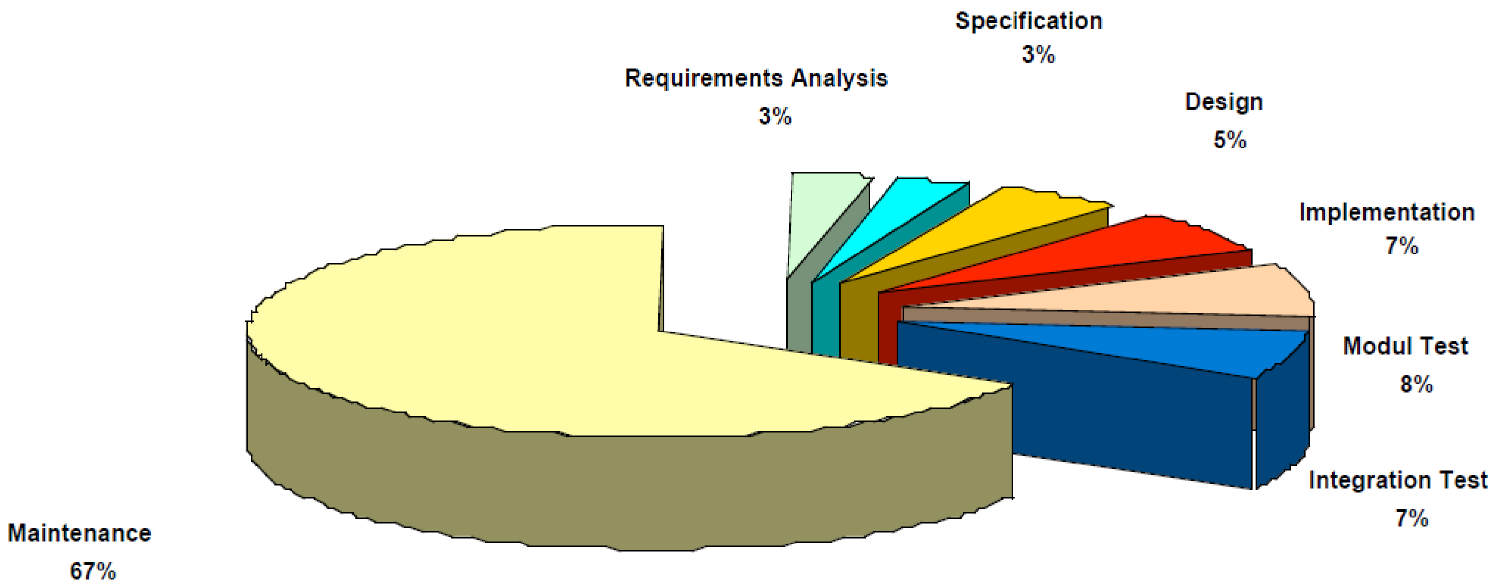
- Software Evolution
  - The process of **changing a software system** from its **creation** until its **shutdown**.

Software Evolution aka Software Aging

> *Programs, like people, get old. We can't prevent aging, but we can understand its causes, take steps to limits its effects, temporarily reverse some of the damage it has caused, and prepare for the day when the software is no longer viable. A sign that the Software Engineering profession has matured will be that we lose our preoccupation with the first release and focus on the long term health of our products. Researchers and practitioners must change their perception of the problems of software development. Only then will Software Engineering deserve to be called Engineering.*

David Lorge Parnas: *Software aging*. In Proc. of the International Conference on Software Engineering (ICSE'94), pages 279–287. IEEE Computer Society Press, 1994.

Software Evolution aka Software Maintenance



Cost Distribution in the Software-Life-Cycle

Source: Principles of Software Engineering and Design, Zelkovits, Shaw, Gannon 1979

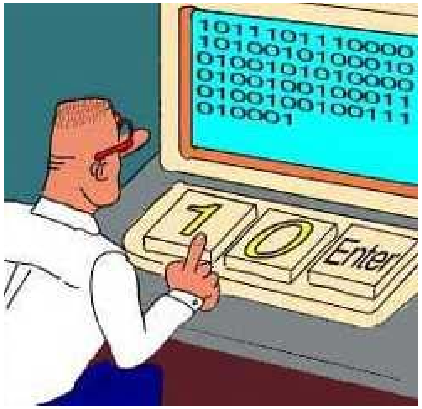
Evolution

Is Software resistant to change? Of course, not!

- Reasons for evolution
  - Technology Switch
  - Restructuring
  - Bug Fixing
  - Functionality Extension
  - Optimization
  - …

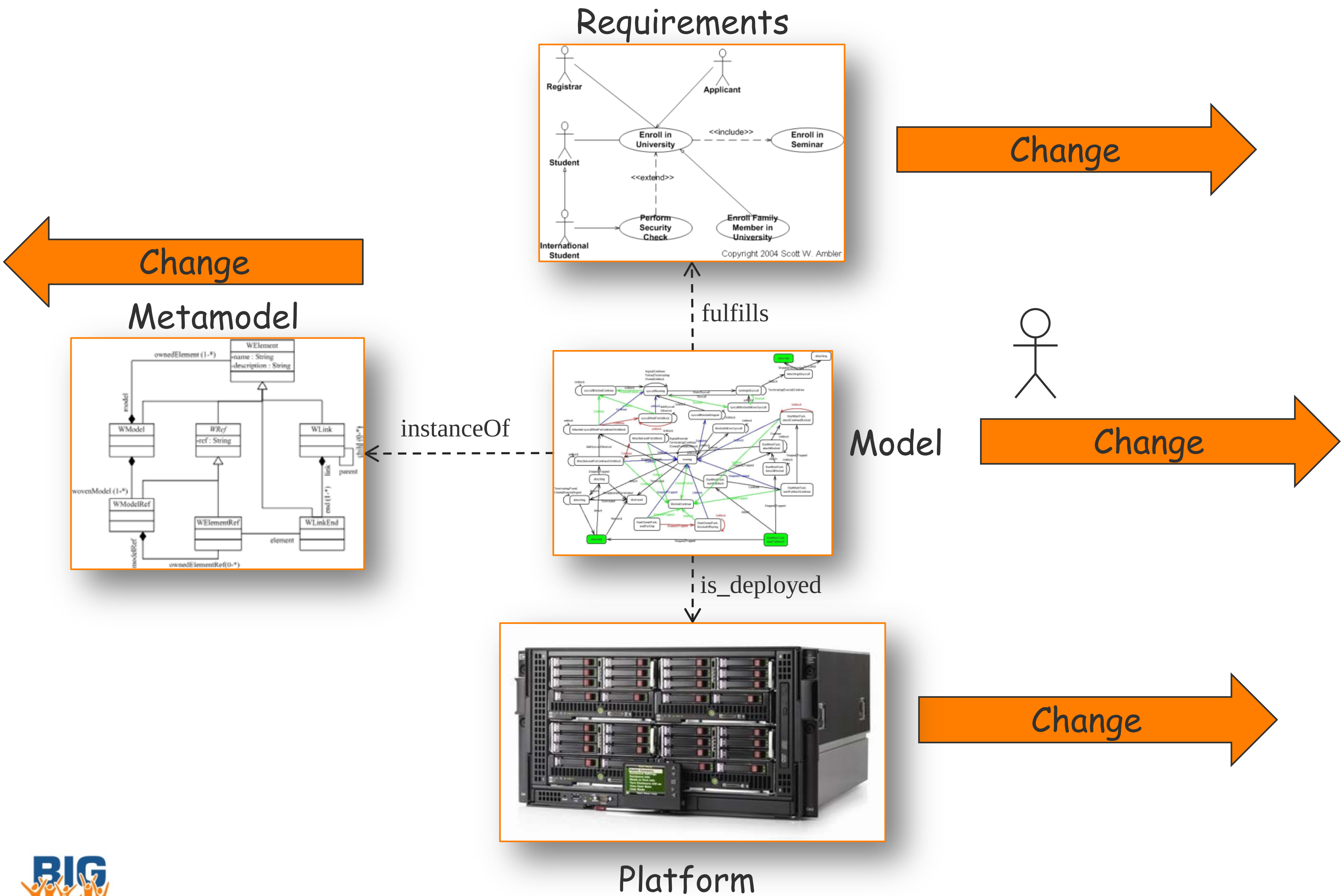
- Bad news: A computer program that is used will be modified, thus its **entropy increases.**

- Silver Bullet?: Model-driven Software Engineering
  - Platform independent models
  - Generate code automatically
  - Are models resistant to change?



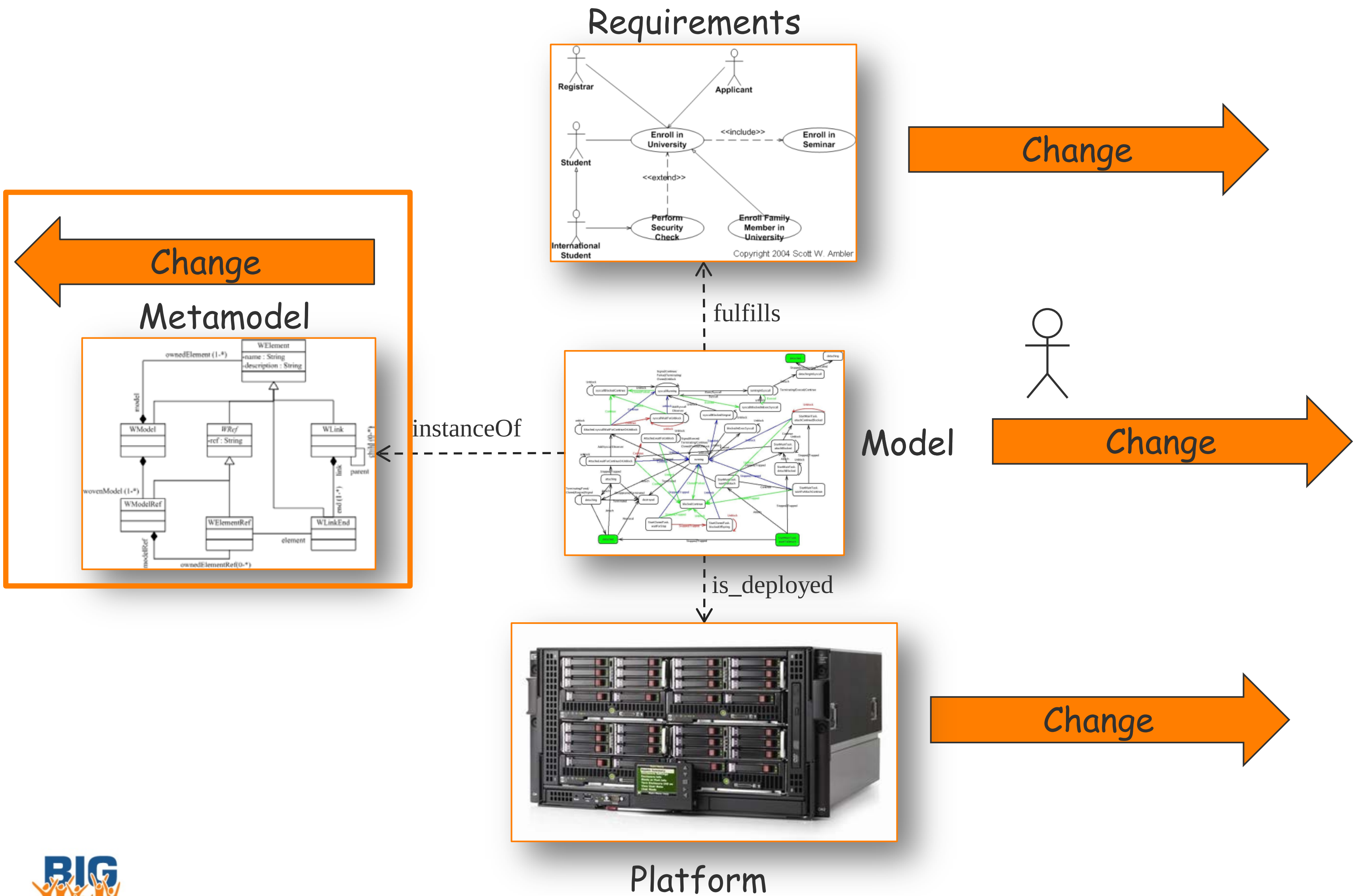
Everything changes…

Evolution in Model-driven Software Engineering



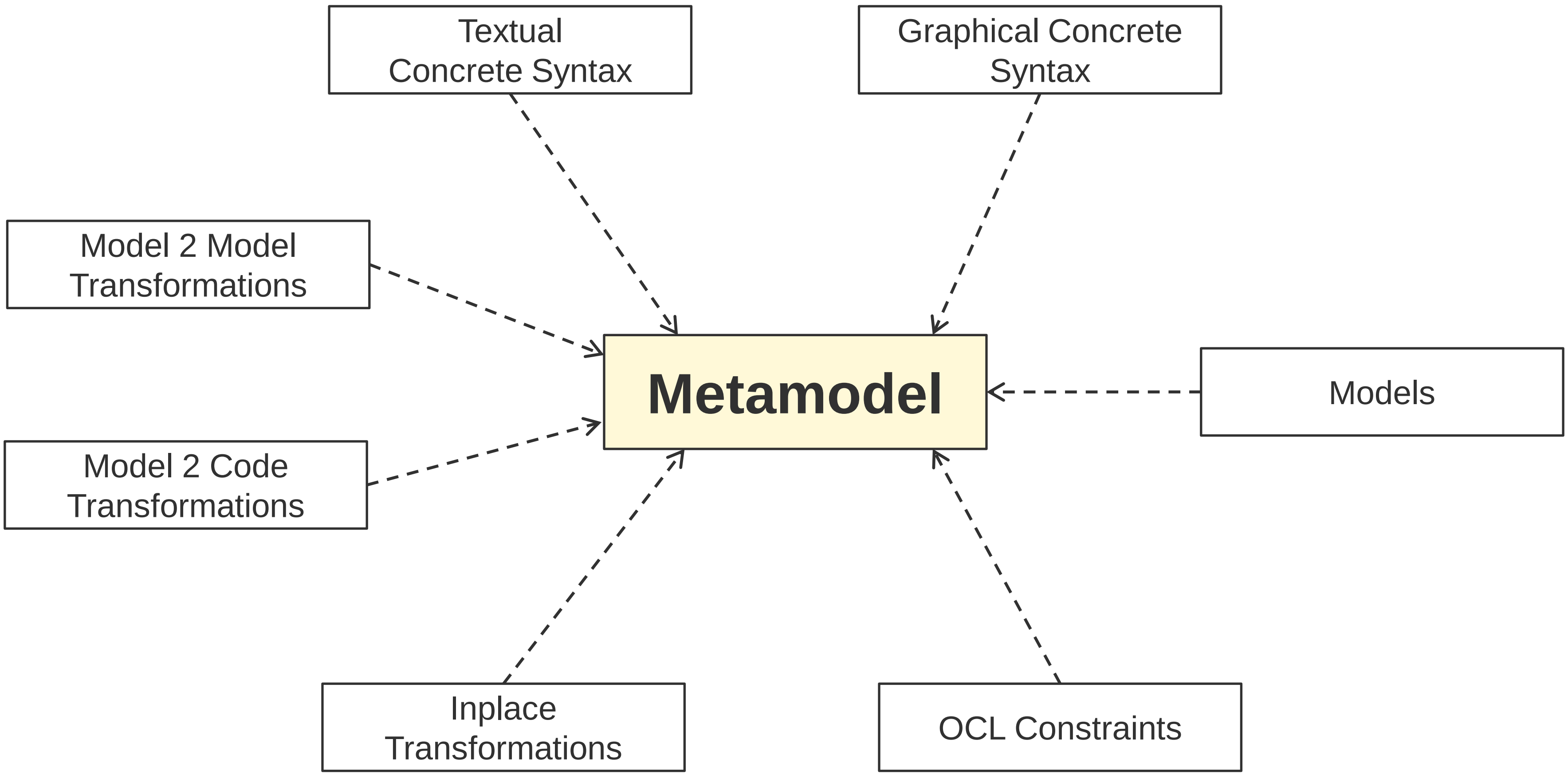
Evolution in Model-driven Software Engineering

Modeling Languages evolve too!



Modeling Language Evolution

Metamodels are the central artefacts!

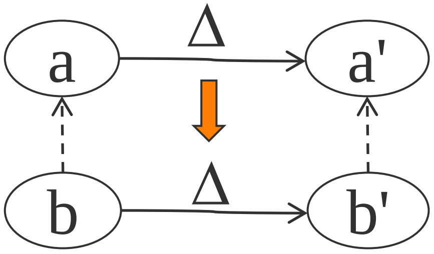


Co-Evolution

Consequence of Metamodel Evolution

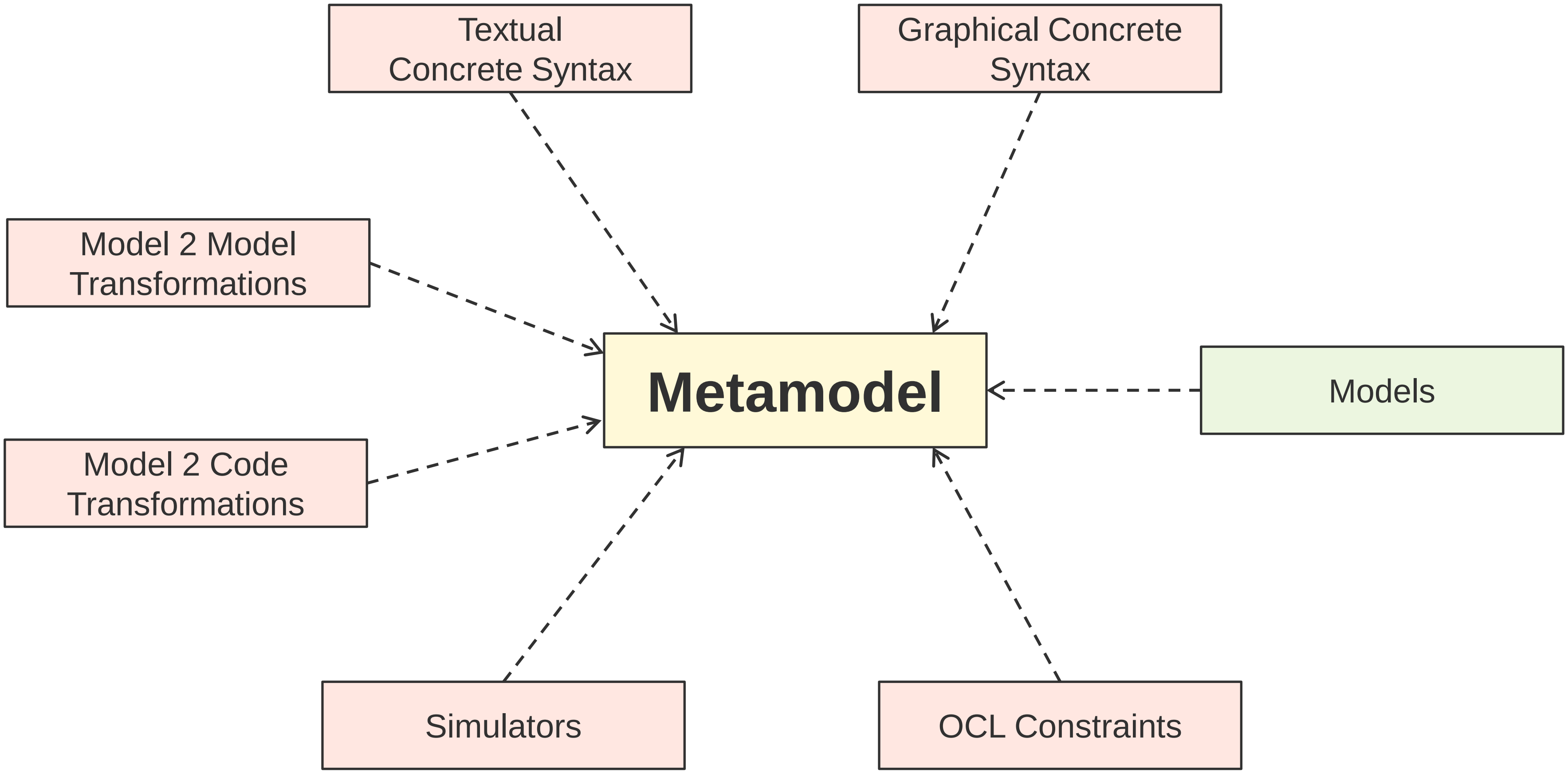
- Term “Co-Evolution” borrowed from Biology
  - Biological co-evolution is the **change** of a biological entity **triggered** by the **change** of a **related** entity
    - **One-to-one Relationships**: *Predator/Prey, Host/Symbiont, Host/Parasite, …*
    - **Diffuse Relationships**: An entity evolves in response to a number of other entities, each of which is also evolving in response to a set of entities

- 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’ | r(a’,b’)
    - **Challenge: Relationship Reconciliation**
  - Current research focus is on one-to-one relationships



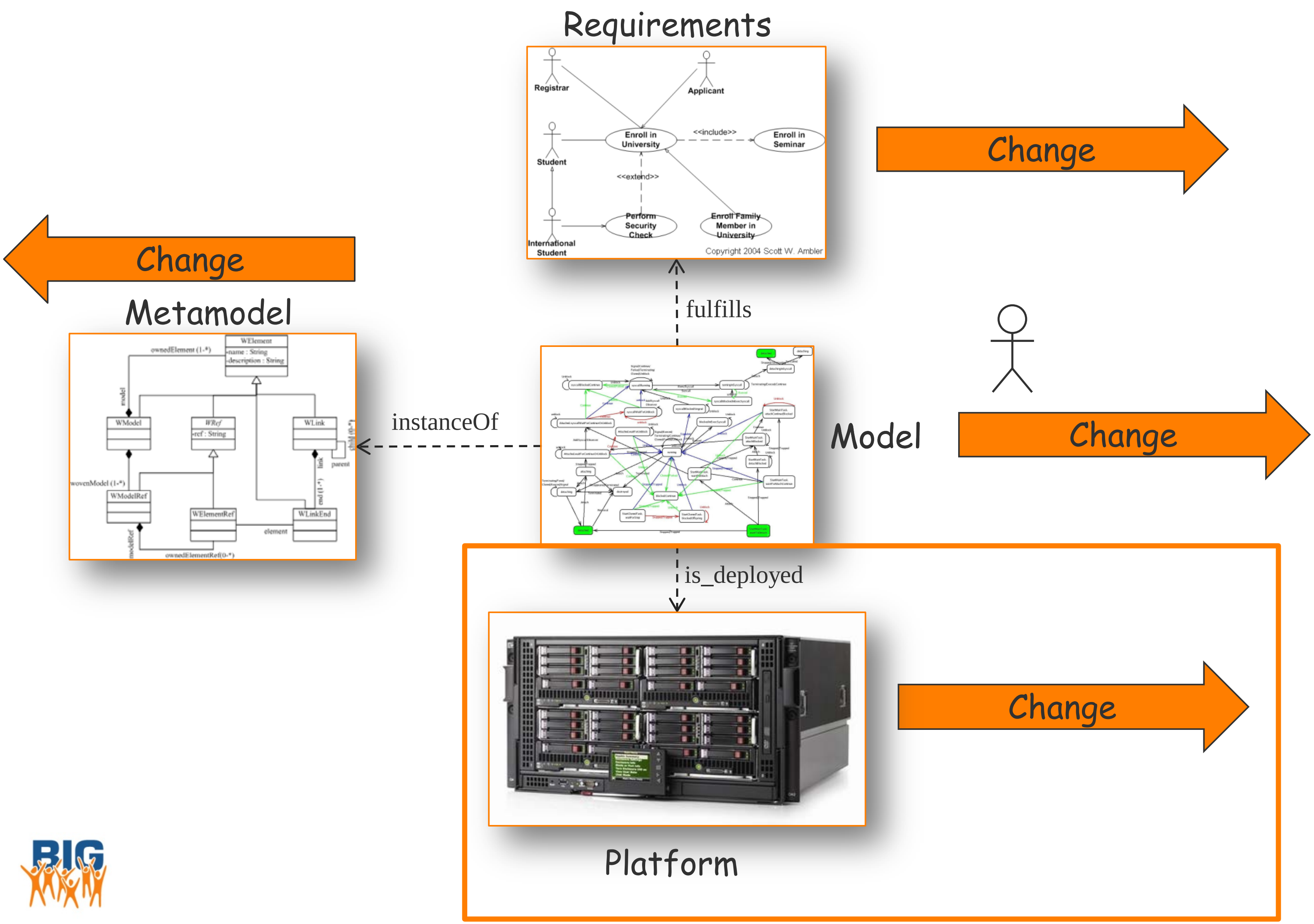
Metamodel Evolution

Mainly models, i.e., instances of metamodels, are currently co-evolved!



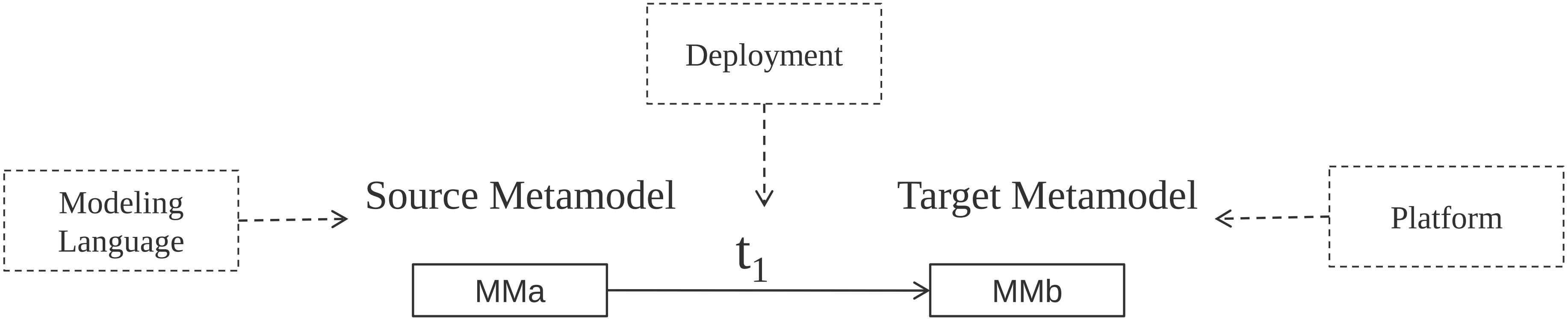
Evolution in Model-driven Software Engineering

Platforms (Software and Hardware) evolve rapidly!



Platforms evolve rapidly!

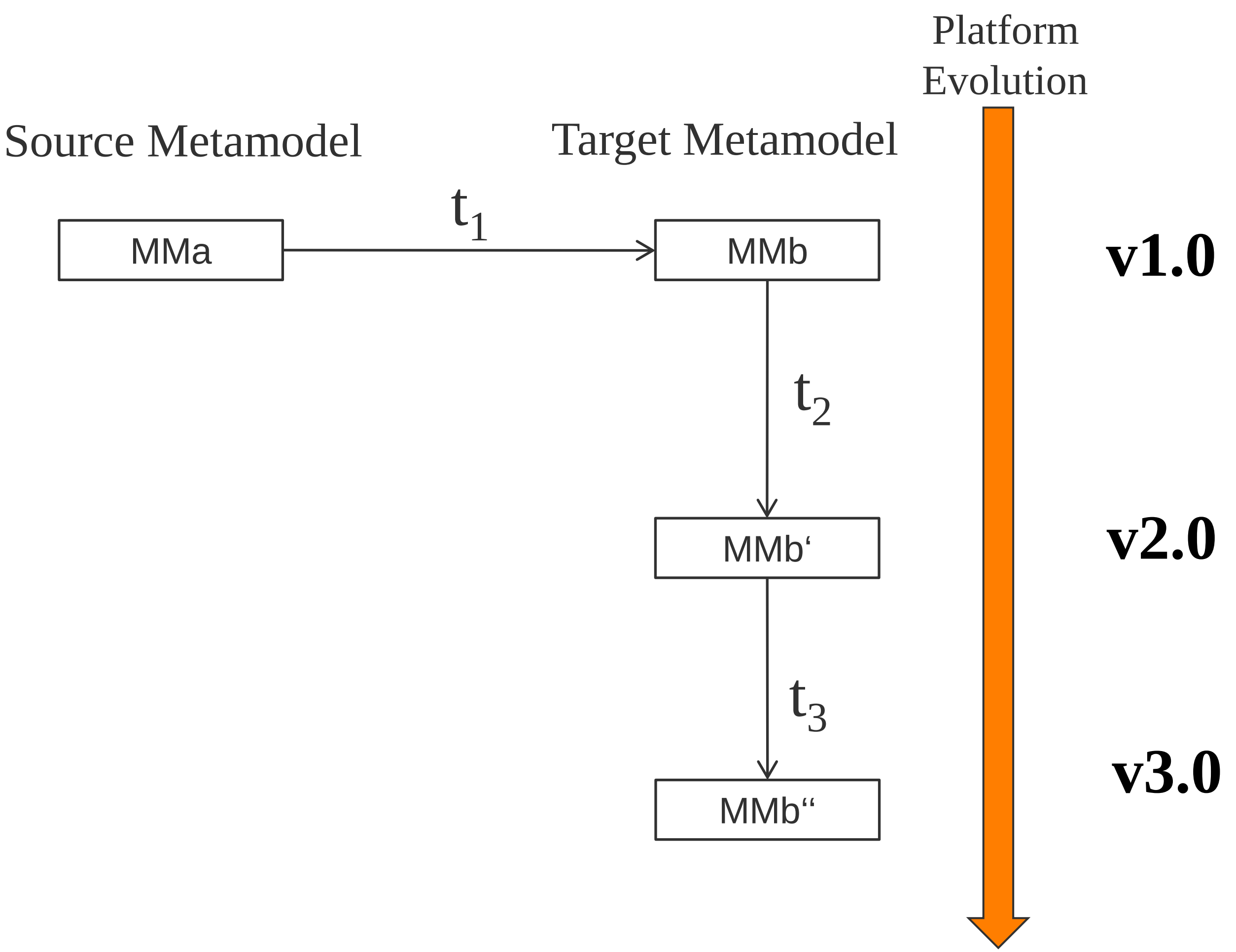
Plaforms are represented by metamodels



t_1 … Forward Transformation

Metamodel/Transformation (Co-)Evolution

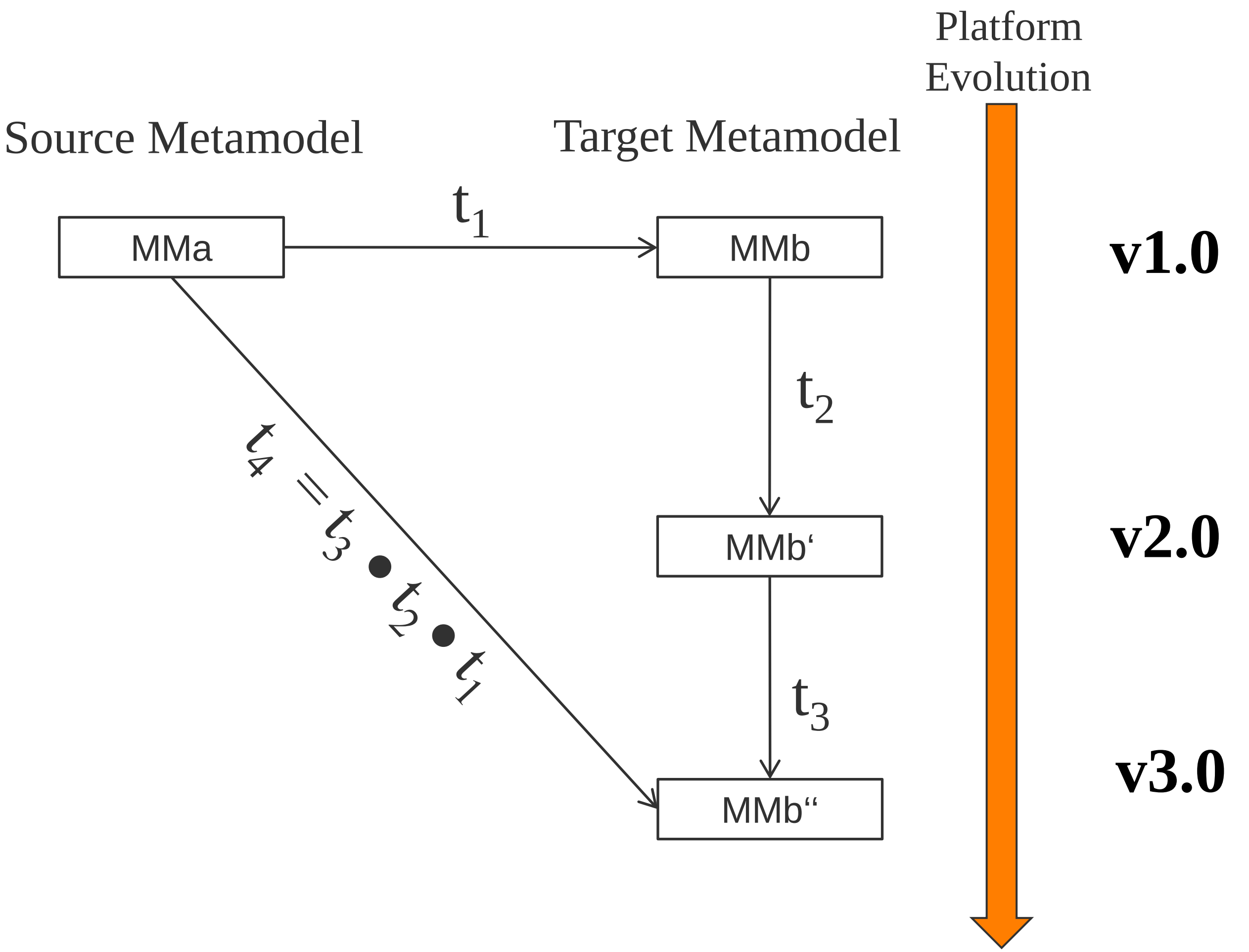
Platform evolution is reflected by target metamodel evolution



t_1 … Forward Transformation
$t_{2,}$ t_3 … Migration Transformation

Metamodel/Transformation (Co-)Evolution

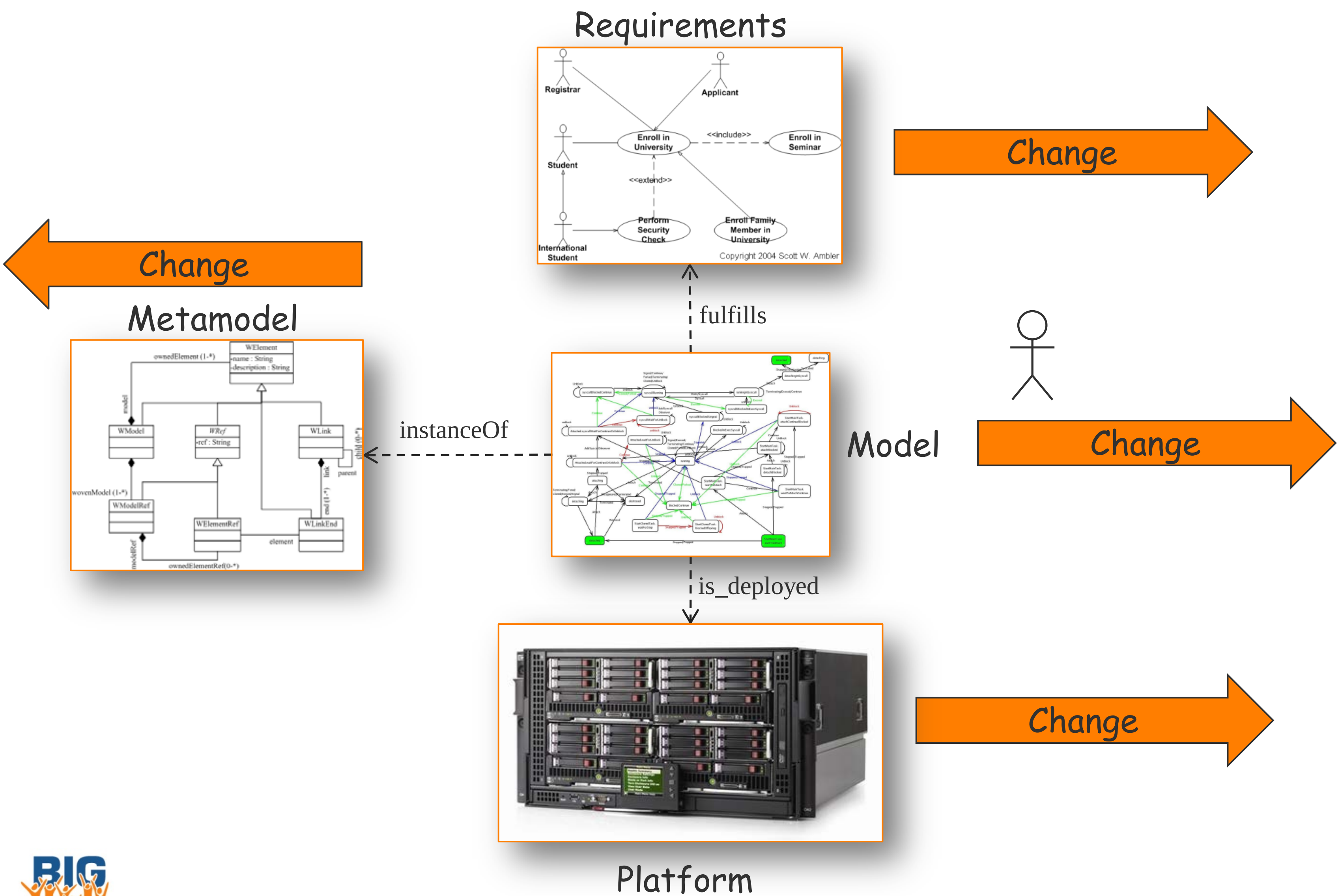
Transformation chains for tackling platform evolution



t_1 … Forward Transformation
t_2, t_3 … Migration Transformation

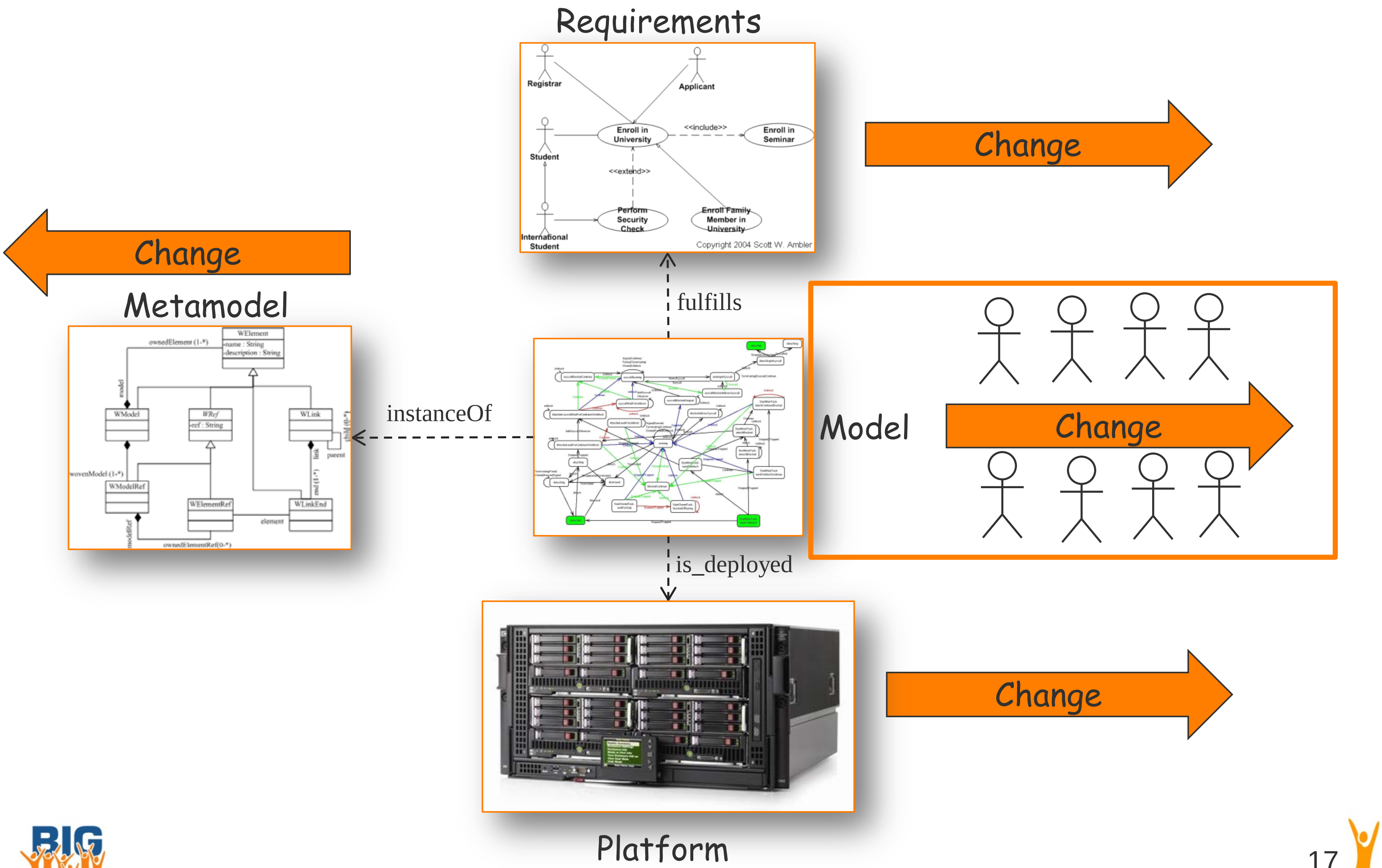
Evolution in Model-driven Software Engineering

Is modeling a one woman-show?



Evolution in Model-driven Software Engineering

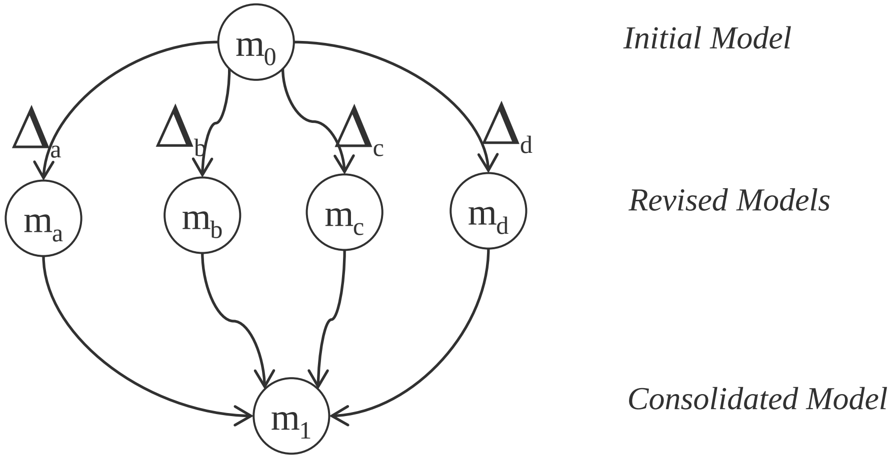
No, support for team-based development is needed!



Support for Team-based Development of Models is Needed!

- Recall some definitions of Software Engineering (SE)
  - SE is defined as the **multi-person construction** of multi-version software
    – *David Lorge Parnas, 1975*
  - SE deals with the building of software systems that are so large or so complex that they are **built by teams** of engineers
    – *Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli , 2002*

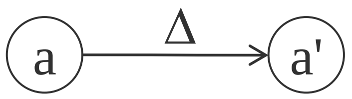
- Assume we have four modelers (a, b, c, d)



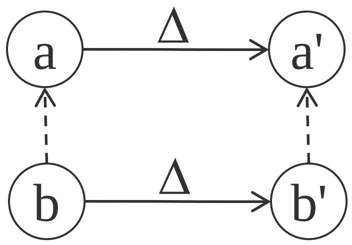
$m1 := m_a \oplus m_b \oplus m_c \oplus m_d$
pre := parallelIndependent(Δ_d, Δ_c, Δ_b, Δ_a)

Evolution Scenarios at a Glance

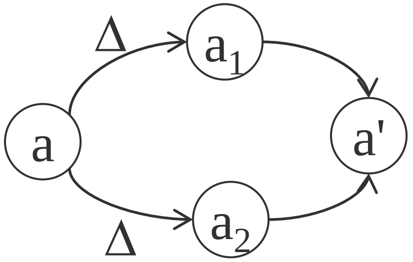
- Evolution
  - a → a’
  - Challenge: Realization & Understanding



- Co-Evolution
  - Consistency Relationship: c(a, b)
  - a → a’
  - b → b’ | c(a’, b’)
  - Challenge: Consistency Reconciliation



- Parallel Evolution
  - a → a1, a → a2
  - a1 ⊕ a2 → a’
  - Challenge: Conflict Detection & Resolution



Outline

- Context of Model Versioning
- **Foundations of Model Versioning**
- Conflict Categorization
- Adaptable Model Versioning
- Future Challenges

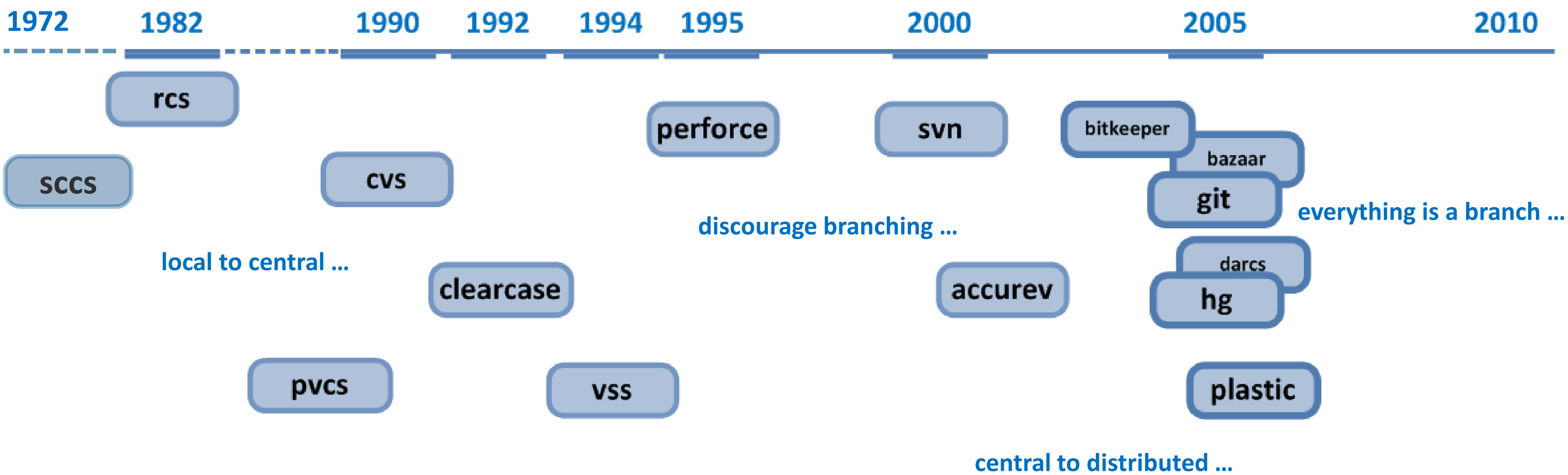
Versioning

- Software Configuration Management (SCM)
  - Originated from aerospace industries in the 1950s
    - Initiated by issues coming from inadequately documented engineering changes
  - Managing and controlling
    - Corrections
    - Extensions
    - Adaptations
  - Traceability throughout the system lifecycle
  - Complex software systems pose similar challenges as other systems
    → Configuration Management for software in late 1970s
    → Software Configuration Management was born (with SCCS)

- Versioning
  - "Maintaining a historical archive of a set of artifacts as they undergo a series of changes"
  - Fundamental building block of SCM

Estublier, J., Leblang, D., van der Hoek, A., Conradi, R., Clemm, G., Tichy, W., & Wiborg-Weber, D. (2005). Impact of Software Engineering Research on the Practice of Software Configuration Management. *ACM Transactions on Software Engineering and Methodology*, 14(4), 383-430.

History of Versioning Systems



Based on http://codicesoftware.blogspot.com/2010/11/version-control-timeline.html

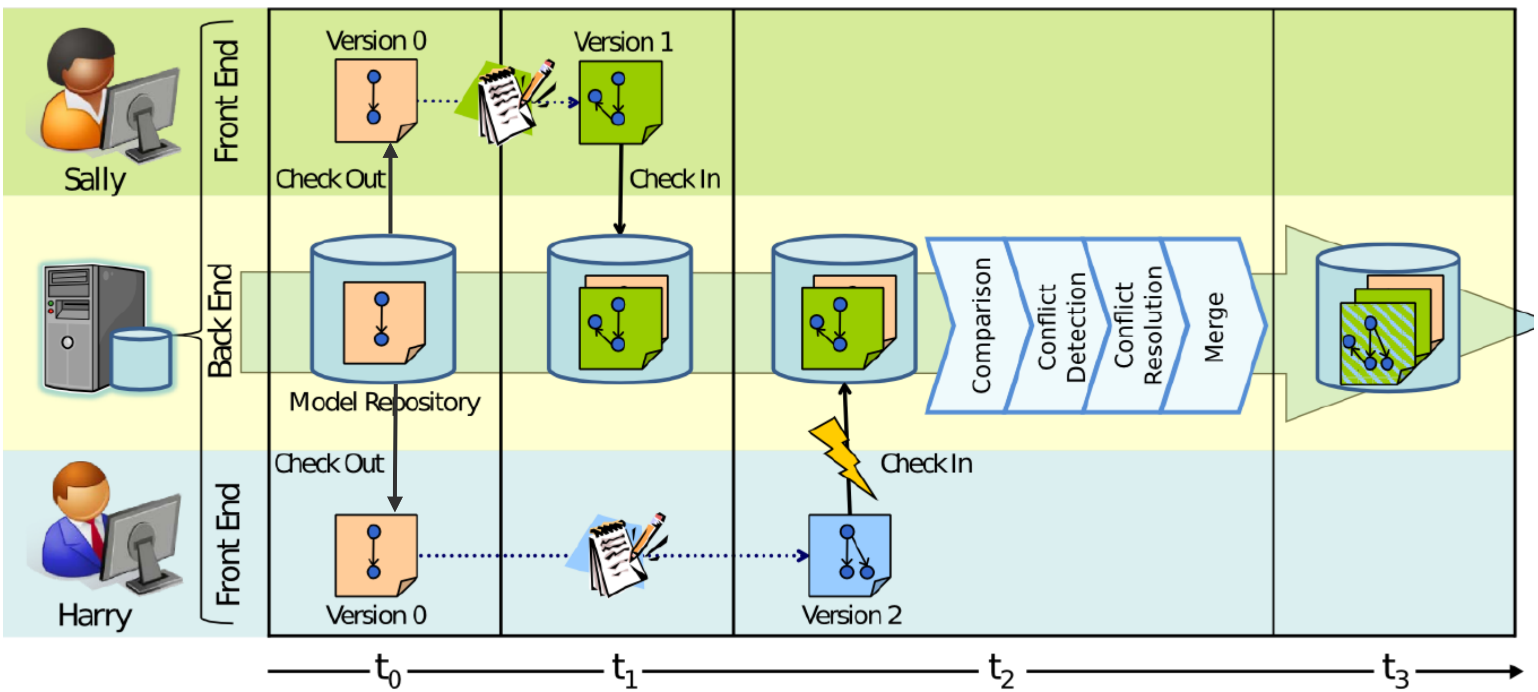
Versioning Paradigms

- Pessimistic Versioning
  - Central repository for storing shared artifacts
  - No concurrent write access allowed (file locking)

- Optimistic Versioning
  - Concurrent teamwork allowed
  - Version Merging necessary
    - Conflicts might occur when same unit of comparison is changed in different ways
      - → Update-update, Delete-update
    - Conflicts have to be resolved manually

Model Versioning

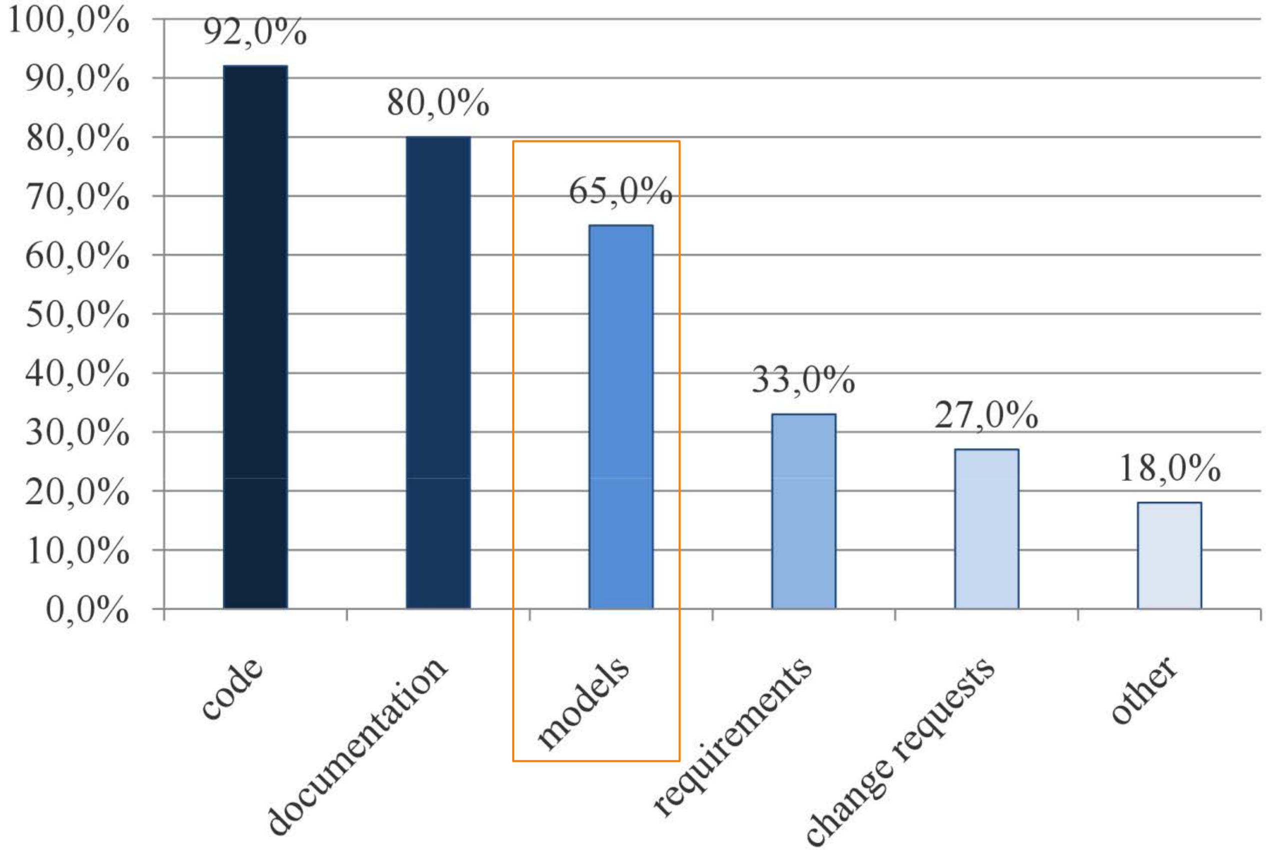
Overview on Optimistic Model Versioning



Model Versioning

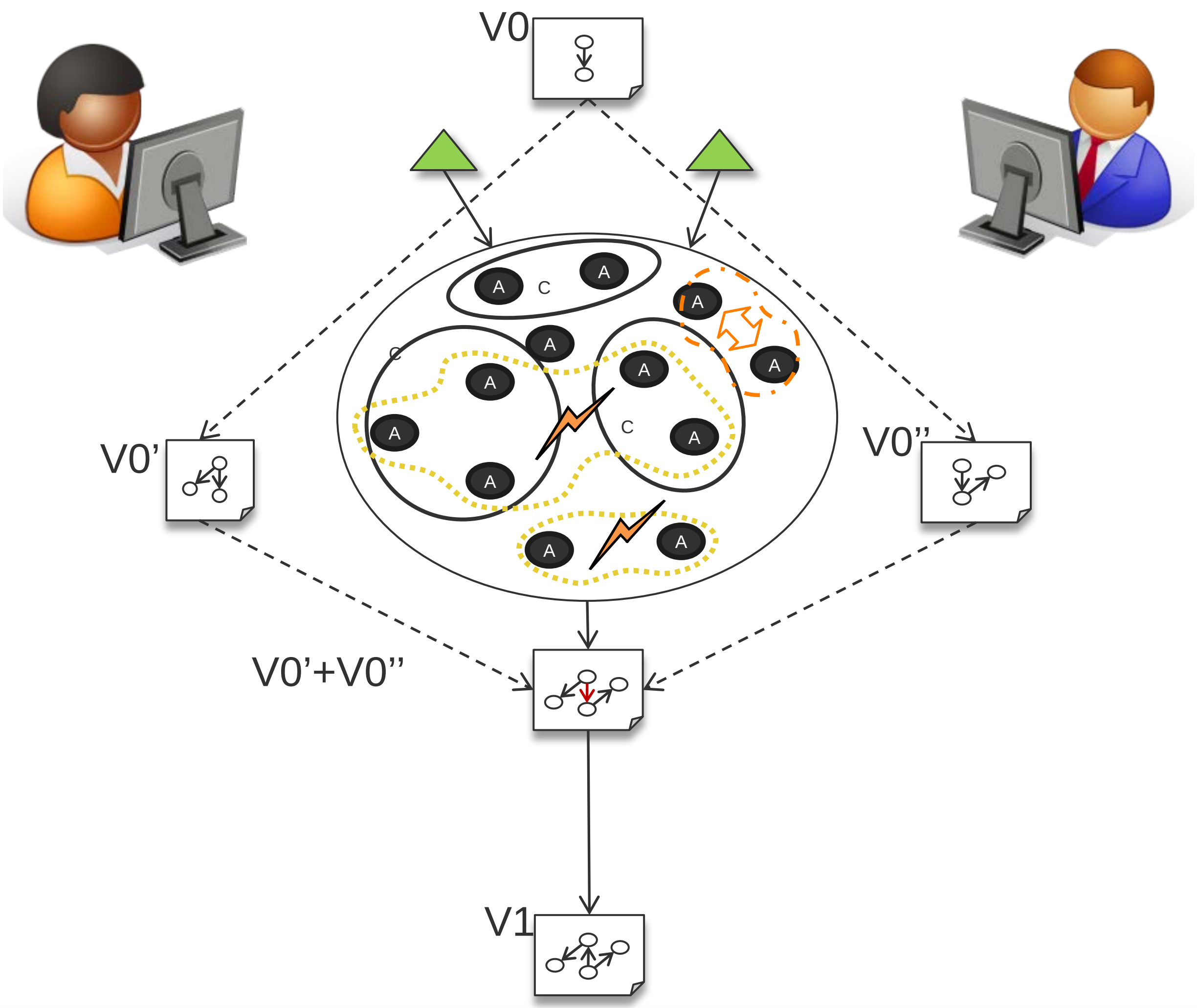
Is getting more and more important

- Empirical study on versioning habits in practice

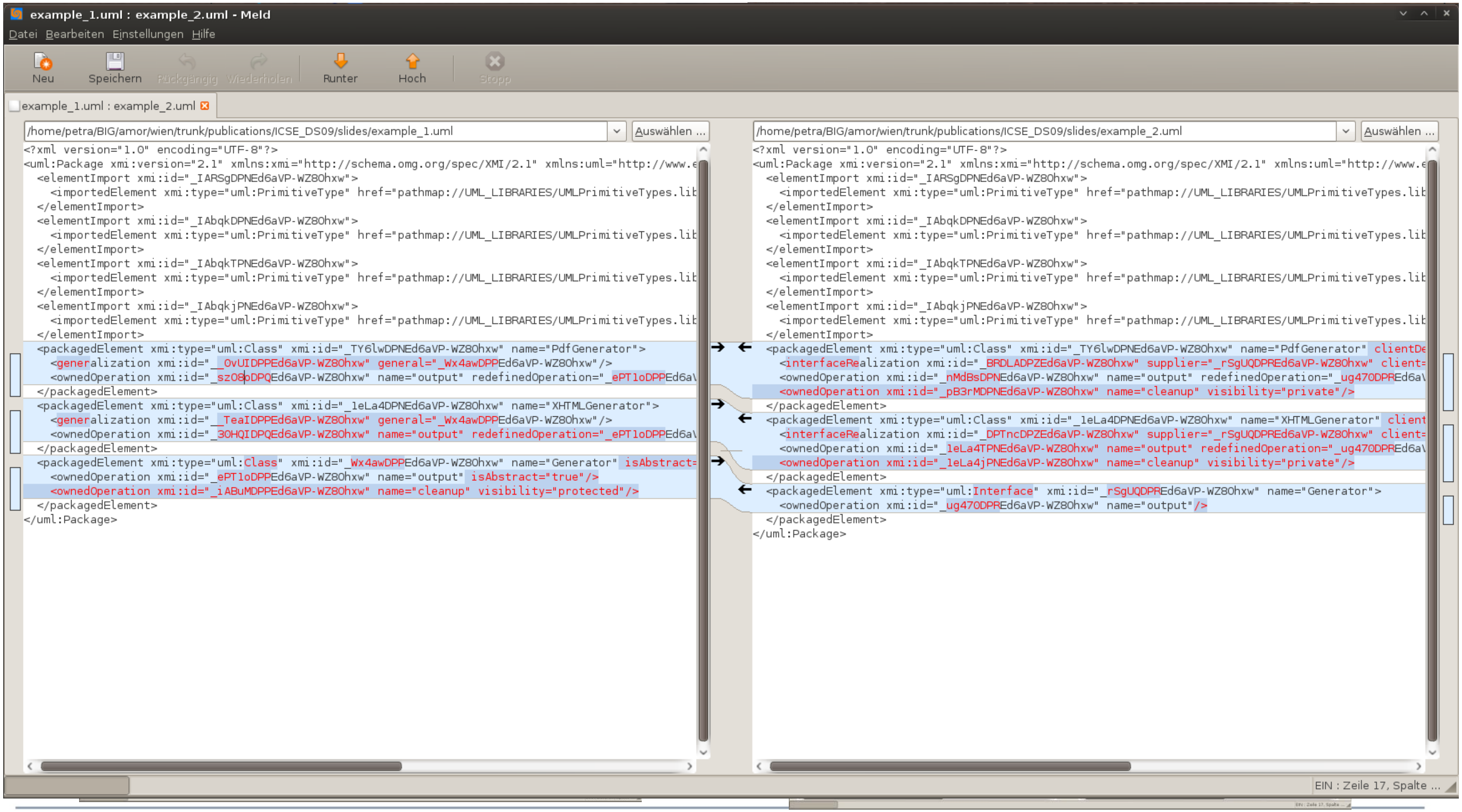


Wieland, K., Fitzpatrick, G., Kappel, G., Seidl, M., and Wimmer, M. (2012). Towards an Understanding of Requirements for Model Versioning Support. To appear in the *International Journal of People-Oriented Programming*, IGI Global.

The Model Versioning Process Revised



Why not SVN with Unix diff?



Design Dimensions of Versioning Systems

Artifact Representation		State-based	Operation-based
	Graph-based	**Graph-based** artifact representation **State-based** delta identification	**Graph-based** artifact representation **Operation-based** delta identification
	Text-based	**Text-based** artifact representation **State-based** delta identification	**Text-based** artifact representation **Operation-based** delta identification
		Delta Identification	

Text-based Merging

Example

Text-based Representation

Version 0

```
1: class Human {
2:        string[1..1] name
3: }
4: class Vehicle {
5:        integer[0..1] carNo
6: }
```

Grammar

```
Class:= "class" name=ID "{"
  (references+=Reference)*
  (attribute+=Attribute)*
"}";
Reference:= target=[Class] "[" lower=BOUND
            ".." upper=BOUND "]" name=ID;
Attribute:= type=ID "[" lower=BOUND
            ".." upper=CARD "]" name=ID;
terminal ID:= ('a'..'z'|'A'..'Z'|'_')+;
terminal BOUND:= (('0'..'9')+)|('*');
```

Version 1

```
1: class Person {
2:        string[1..1] name
3:        Vehicle[0..*] owns
4: }
5: class Vehicle {
6:        integer[1..1] carNo
7: }
```

Version 2

```
1: class Human {
2:        string[1..1] name
3: }
4: class Car {
5:        integer[0..1] regId
6: }
```

Text-based and State-based Merging

Version 0

```
class Human {
        string[1..1] name
}
class Vehicle {
        integer[0..1] carNo
}
```

Version 1

```
class Person {
        string[1..1] name
        Vehicle[0..*] owns
}
class Vehicle {
        integer[1..1] carNo
}
```

Version 2

```
class Human {
        string[1..1] name
}
class Car {
        integer[0..1] regId
}
```

Version 3

```
class Person {
        string[1..1] name
        Vehicle[0..*] owns
}
class Car {
        <<UP/UP>>
}
```

```
a: integer[1..1] carNo
b: integer[0..1] regId
c: integer[1..1] regId
```

Text-based and Operation-based Merging

Version 0

```
class Human {
        string[1..1] name
}
class Vehicle {
        integer[0..1] carNo
}
```

Version 1

```
class Person {
        string[1..1] name
        Vehicle[0..*] owns
}
class Vehicle {
        integer[1..1] carNo
}
```

Version 2

```
class Human {
        string[1..1] name
}
class Car {
        integer[0..1] regId
}
```

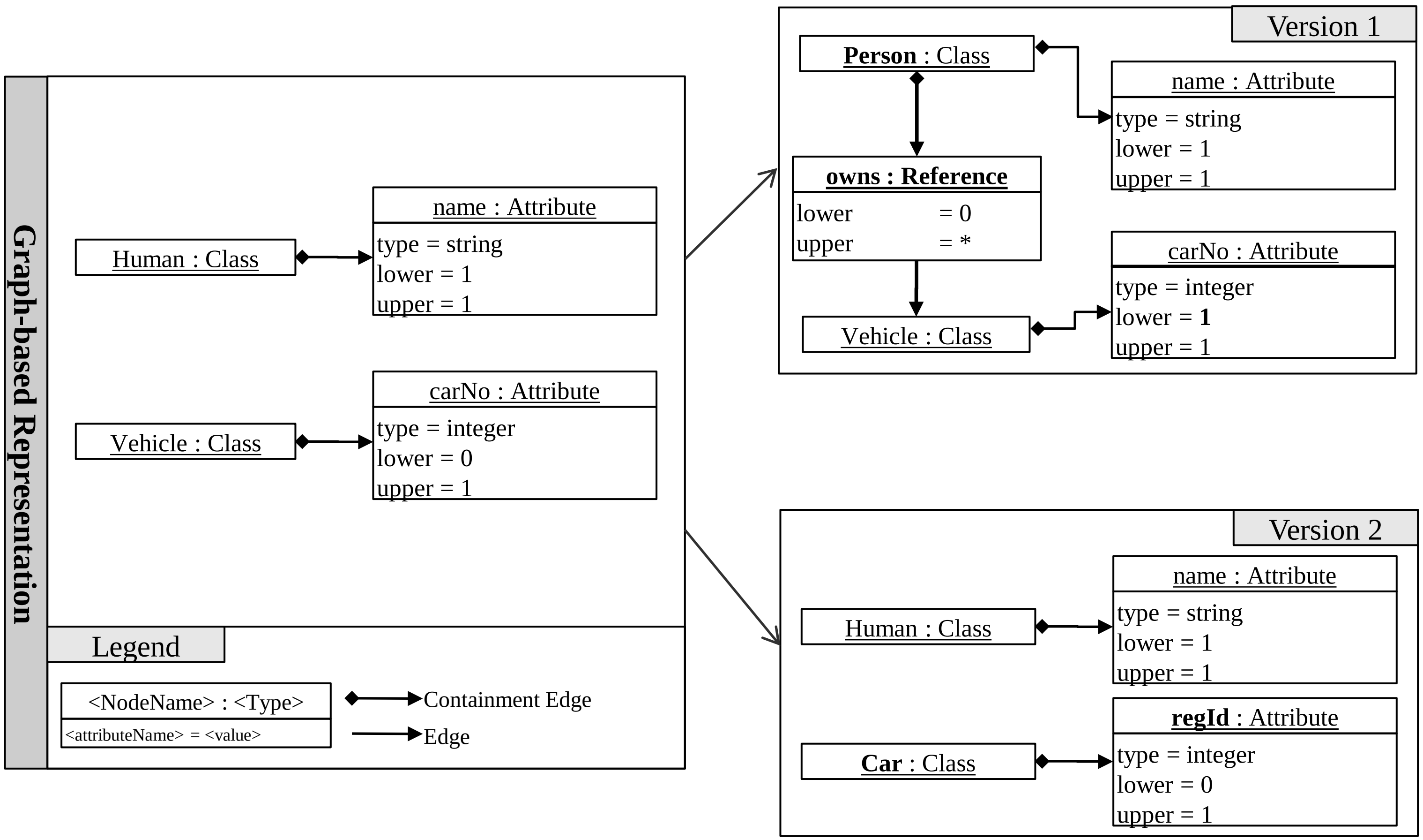
Version 3

```
class Person {
        string[1..1] name
        Car[0..*] owns
}
class Car {
        <<UP/UP>>   a: integer[1..1] carNo
                    b: integer[0..1] regId
                    c: integer[1..1] regId
}
```

Rename-Op:
change Class.name;
update Property.type
pre@Class.name with
post@Class.name;

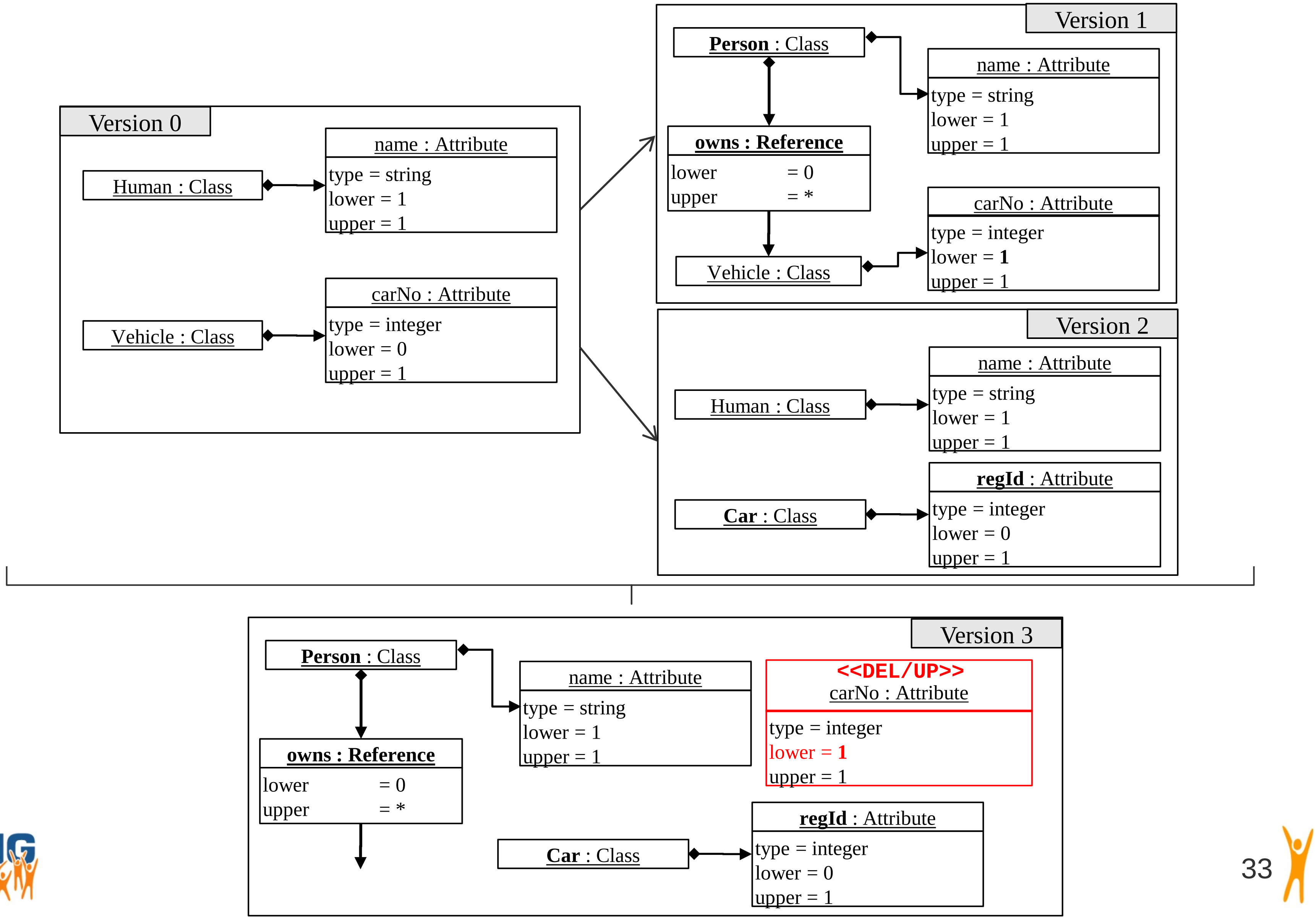
Graph-based Merging

Example



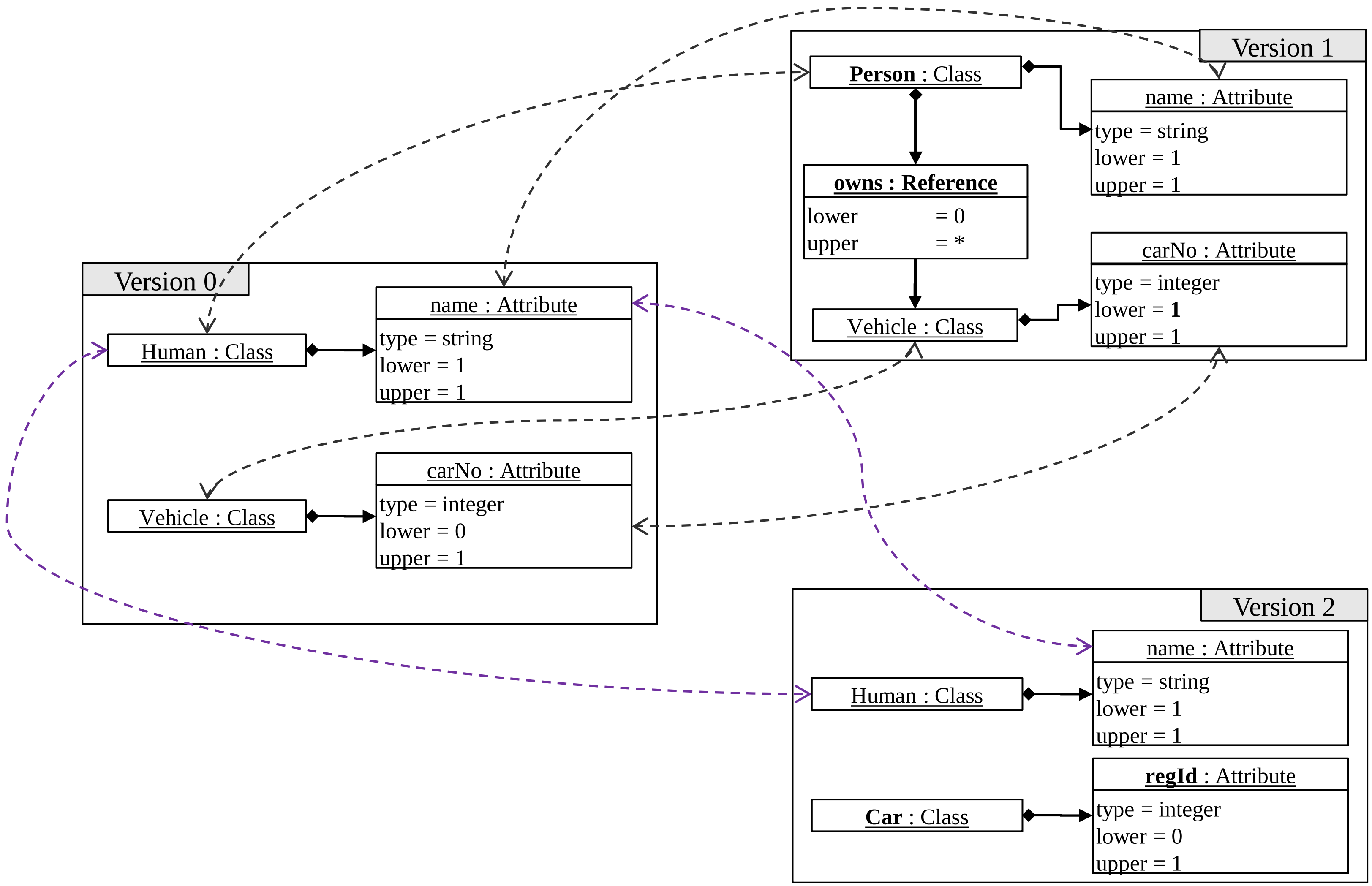
Graph-based and State-based Merging

Heuristic-based Matching



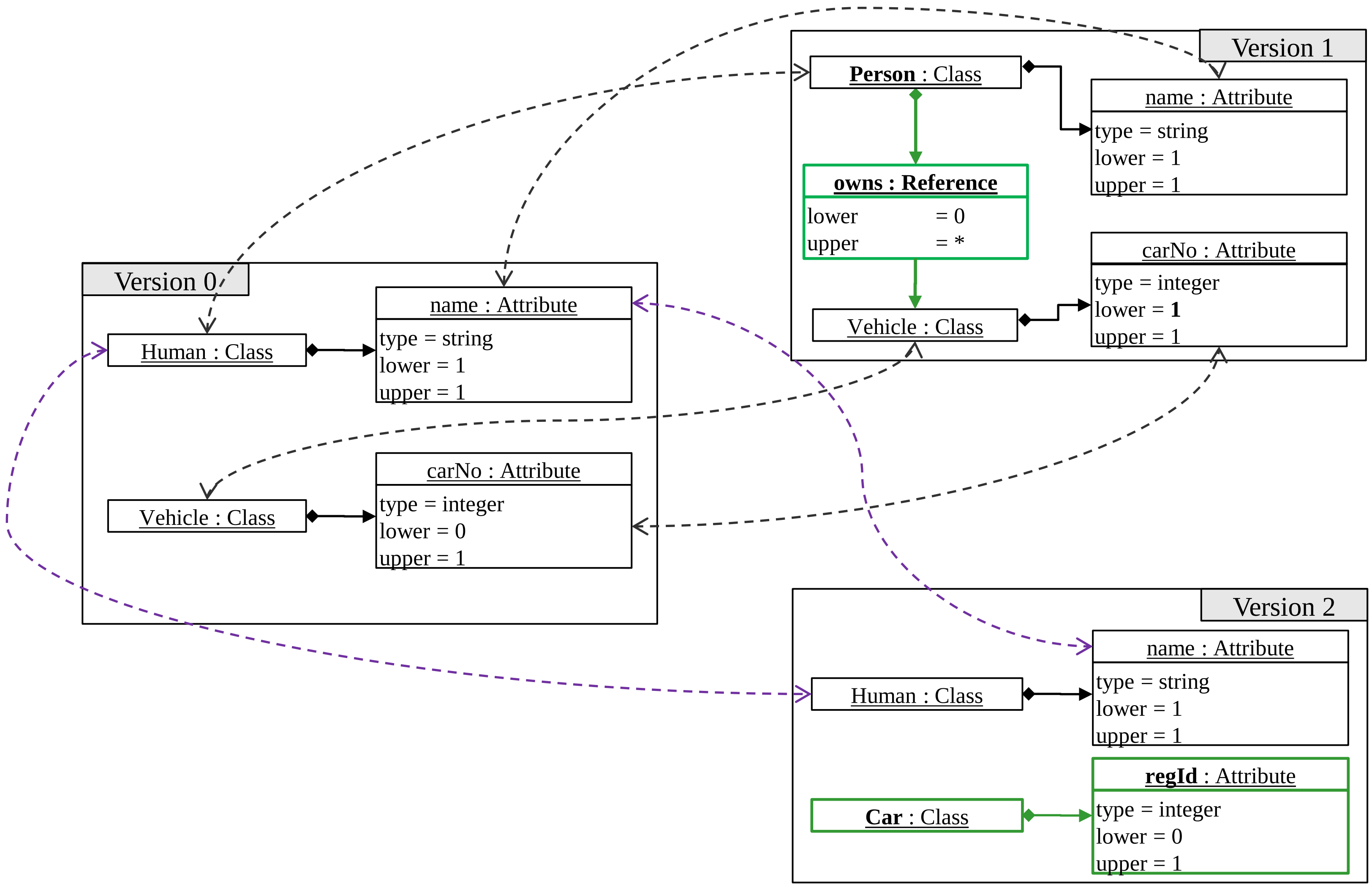
Graph-based and State-based Merging

Match Model based on Heuristic (Name + Structure Equivalence)



Graph-based and State-based Merging

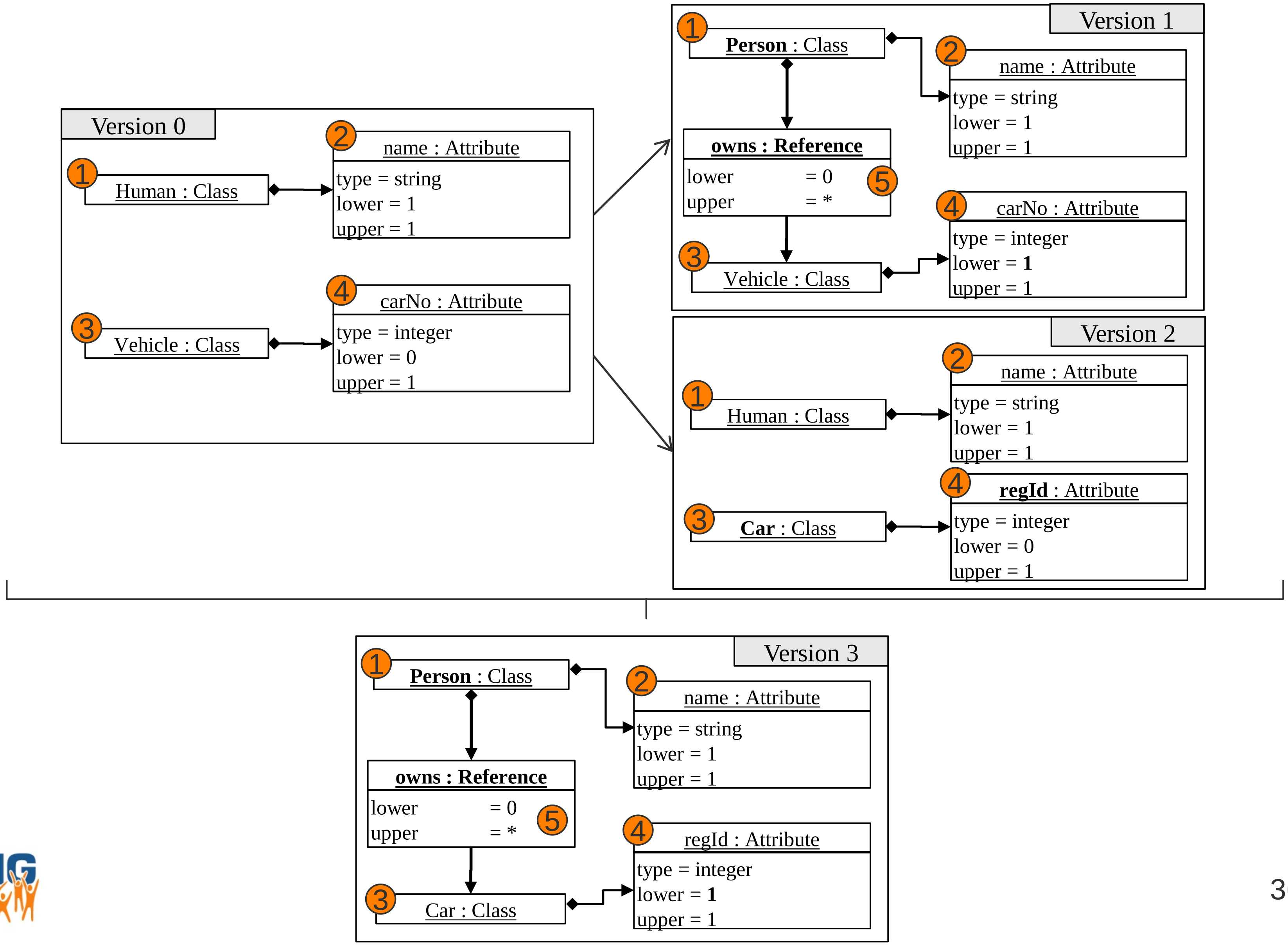
Match Model + Diff Model



remove(Vehicle:Class, carNo:Attribute)

Graph-based and State-based Merging

ID-based Matching



State-based Merge Algorithm (agnostic of representation & match and diff realization)

Design Decisions

- Matching `(hasMatch)`
  - By Equivalence
  - By ID
  - By Heuristics
- Comparison `(diff)`
  - Element granularity
  - Feature granularity
- Consolidation `(remove/add)`
  - Take element from either V1 or V2
  - Evolve original element (V0) → towards operation-based merging

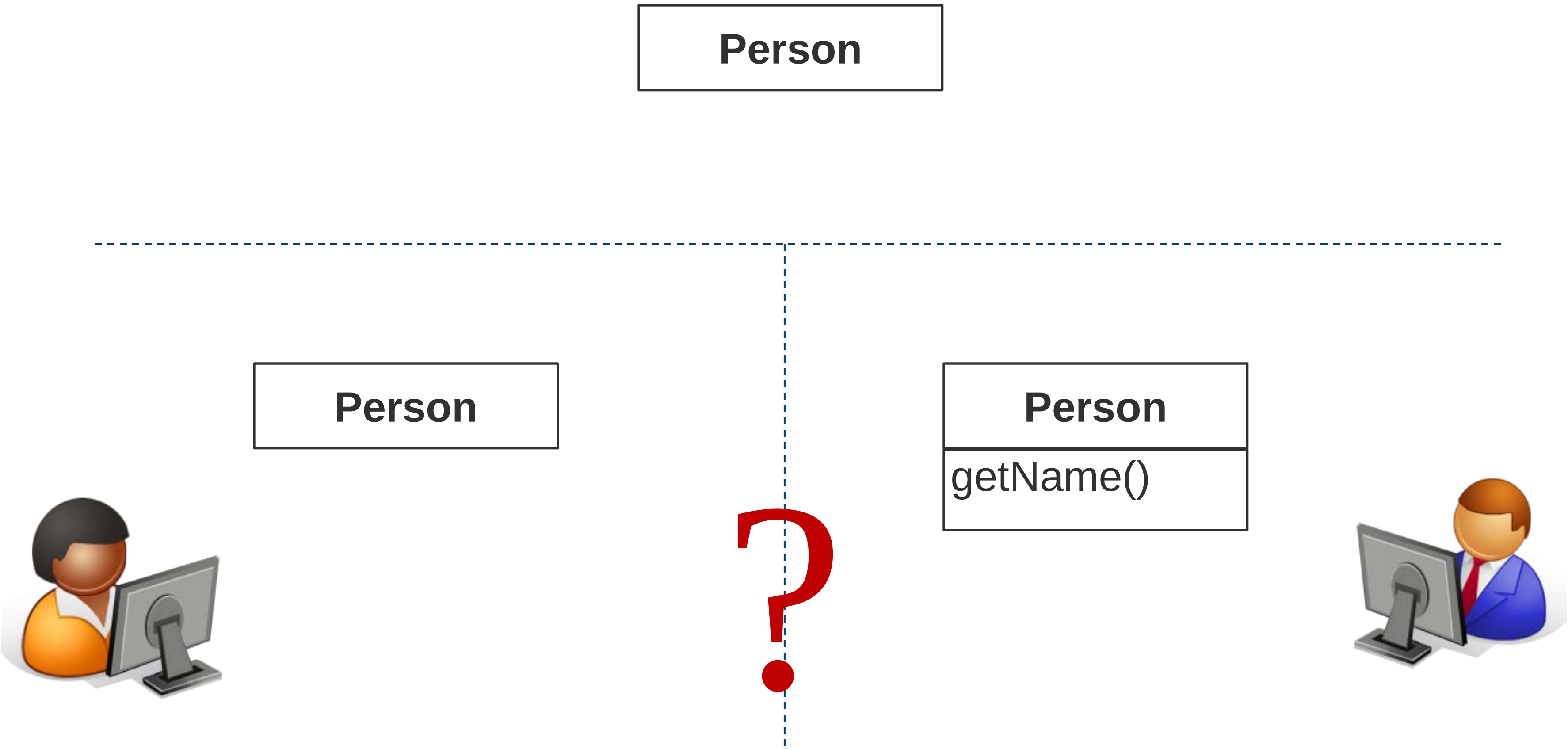
```
for each n0 ∊ V0
  n1 := match(n0 in V1)
  n2 := match(n0 in V2)
  if hasMatch(n0 in V1) && hasMatch(n0 in V2)
    if diff(n0, n1) && not diff(n0, n2) -> use n1
    if not diff(n0, n1) && diff(n0, n2) -> use n2
    if diff(n0, n1) && diff(n0, n2)
         -> raise update/update conflict
    if not diff(n0, n1) && not diff(n0, n2) -> use n0
  end if
  if hasMatch(n0 in V1) && not hasMatch(n0 in V2)
    if diff(n0, n1) -> raise delete/update conflict
    if not diff(n0, n1) -> remove n0
  end if
  if not hasMatch(n0 in V1) && hasMatch(n0 in V2)
    if diff(n0, n2) -> raise delete/update conflict
    if not diff(n0, n2) -> remove n0
  end if
  if not hasMatch(n0 in V1) && not hasMatch(n0 in V2)
         -> remove n0
end for
for each m1 ∊ V1 | not hasMatch(m1, V0)
         -> add m1 to merged version
for each m2 ∊ V2 | not hasMatch(m2, V0)
         -> add m2 to merged version
```

Outline

- Context of Model Versioning
- Foundations of Model Versioning
- **Conflict Categorization**
- Adaptable Model Versioning
- Future Challenges

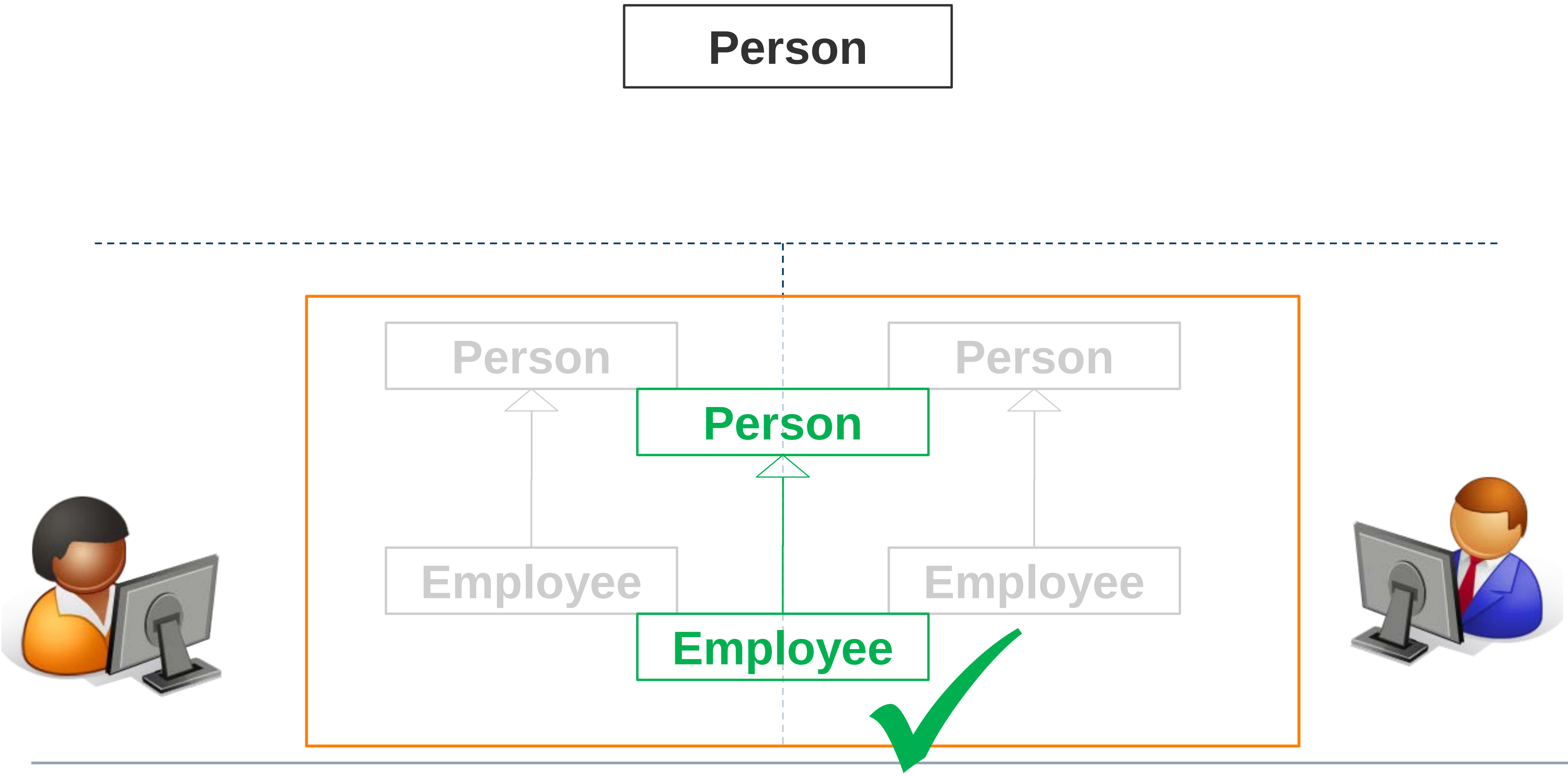
Conflict Examples

Contradicting Change



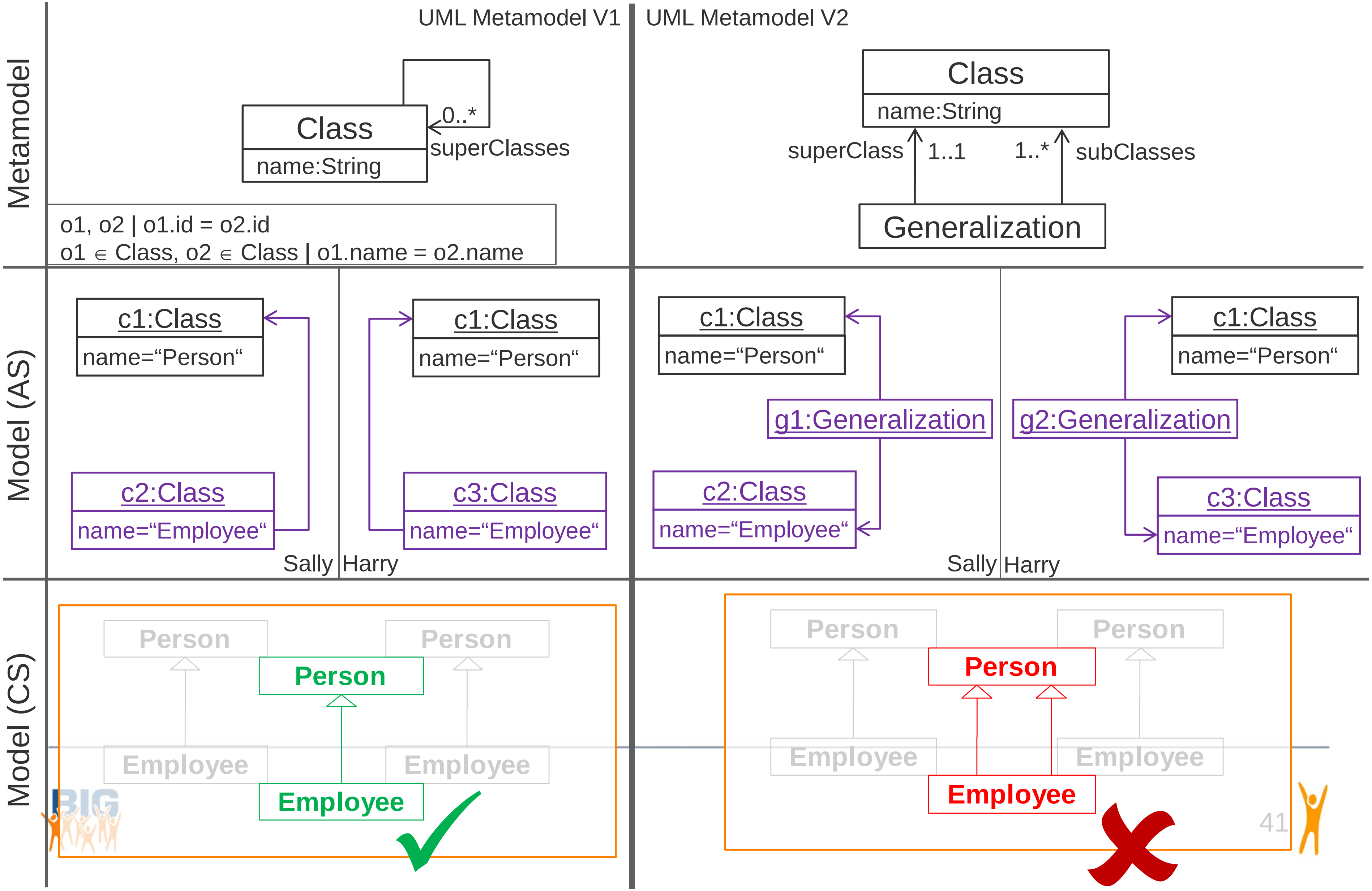
Conflict Examples

Equivalent Change



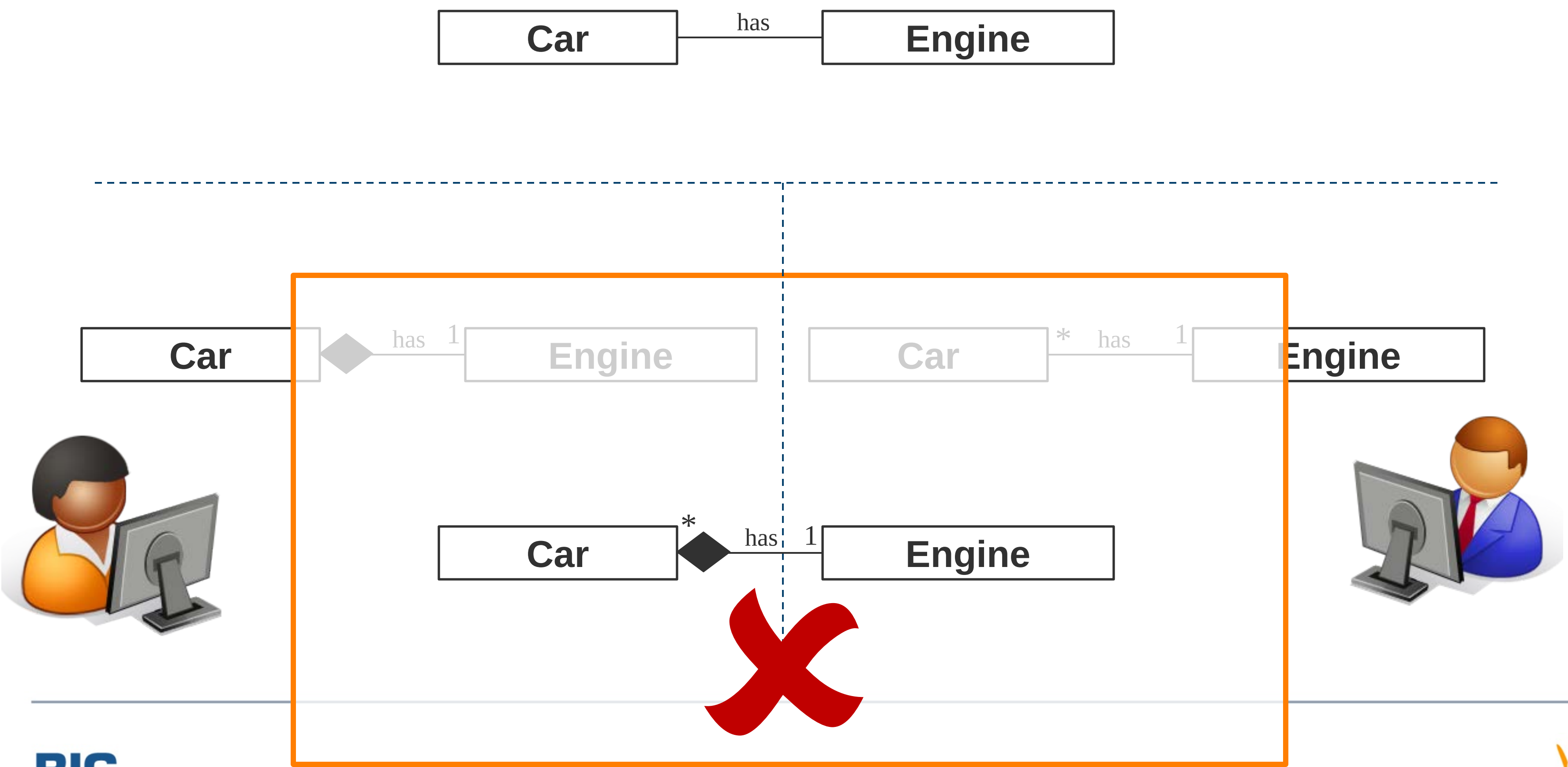
Conflict Examples

Equivalent Change



Conflict Examples

Syntactic Inconsistency

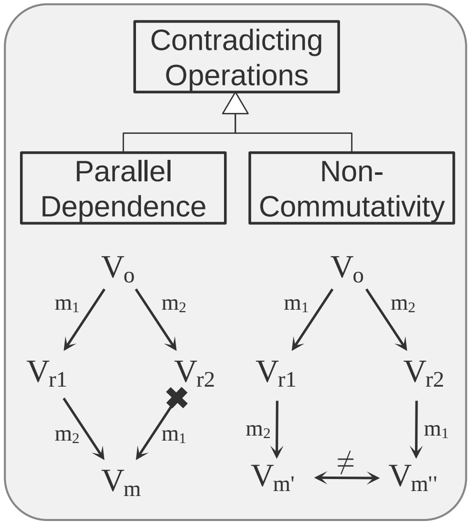


Conflict Categorization

- What is the reason for a conflict?
  - Contradicting operations
    - One operation cannot be applied after the other
    - The order in which the operations are applied affect the merge result

    → **Operation-based conflicts**

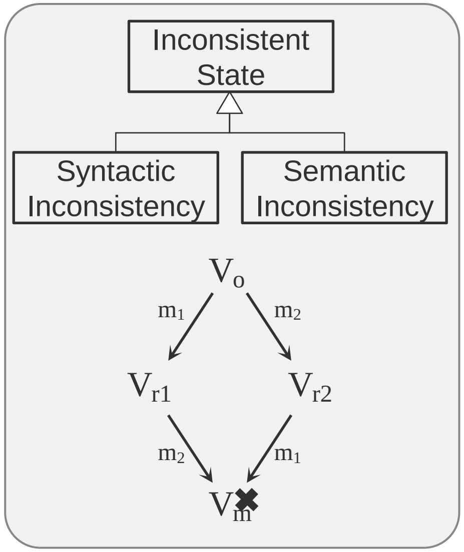
    → The set of applied operations and a specification of the characteristics of the operations is required, but no language specific knowledge



  - Inconsistent resulting state
    - Inconsistent regarding abstract syntax rules
    - Inconsistent regarding the semantics

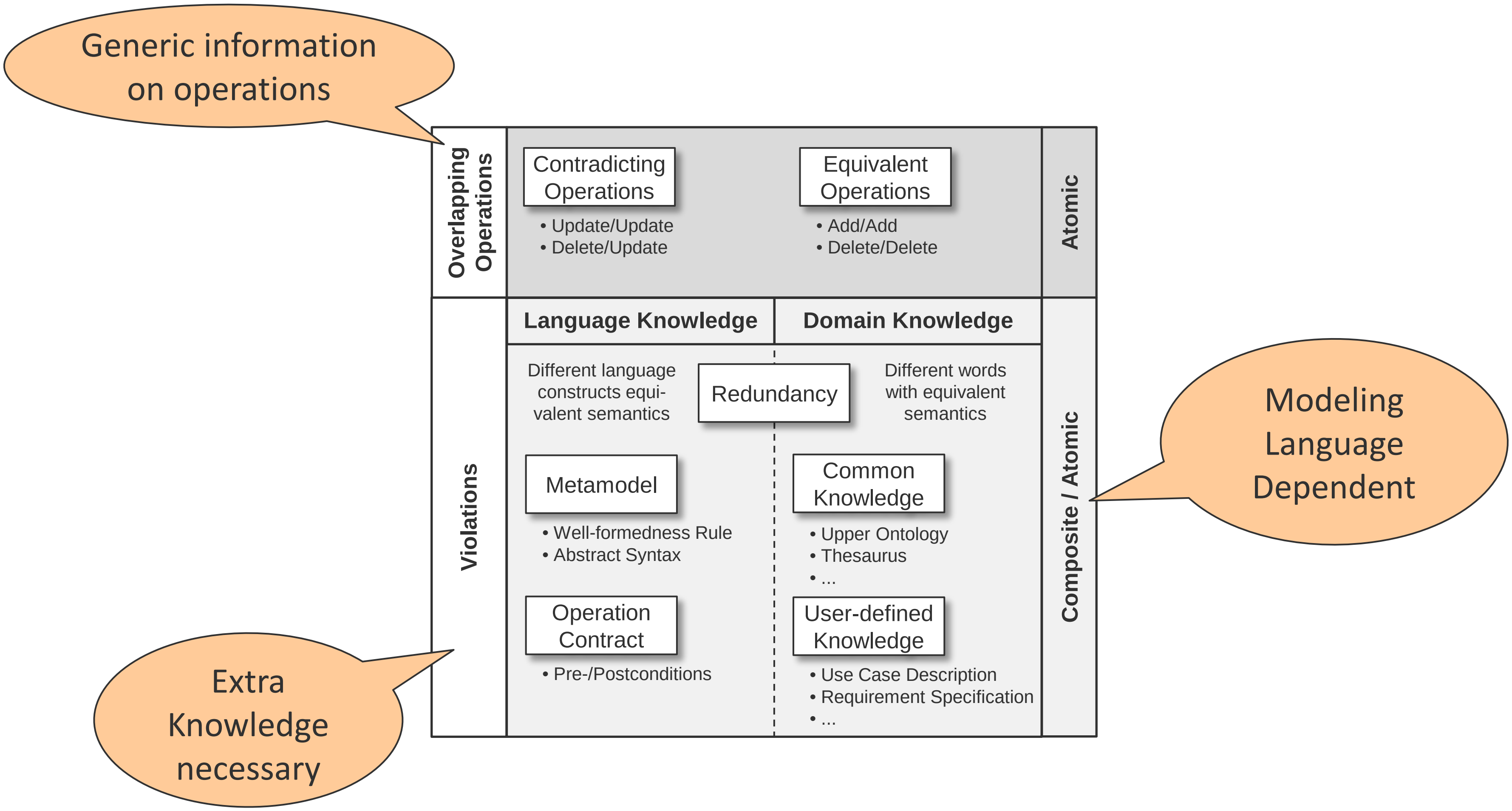
    → **State-based conflicts**

    → A merged state and consistency rules are required

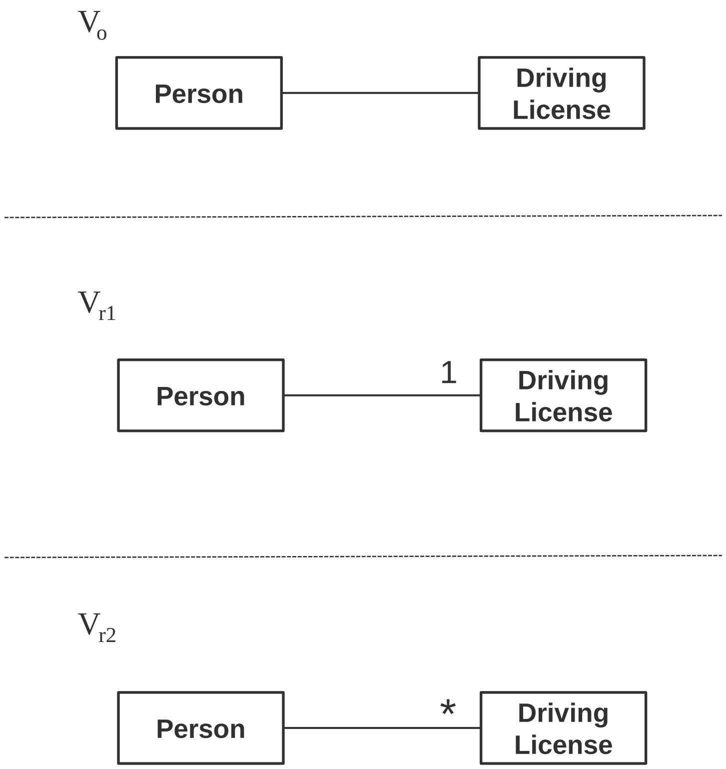


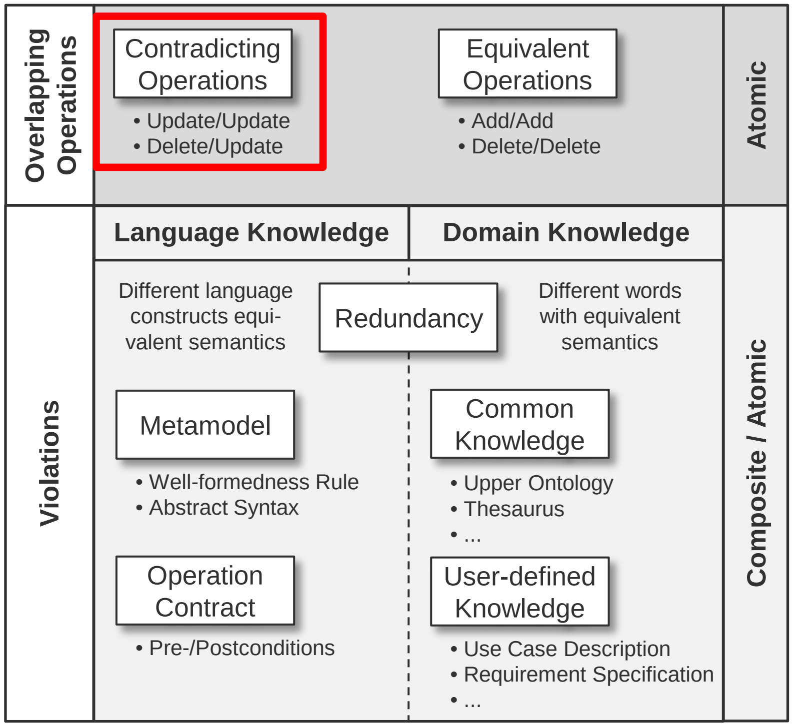
Conflict Categorization

Which information/knowledge is required for detection?

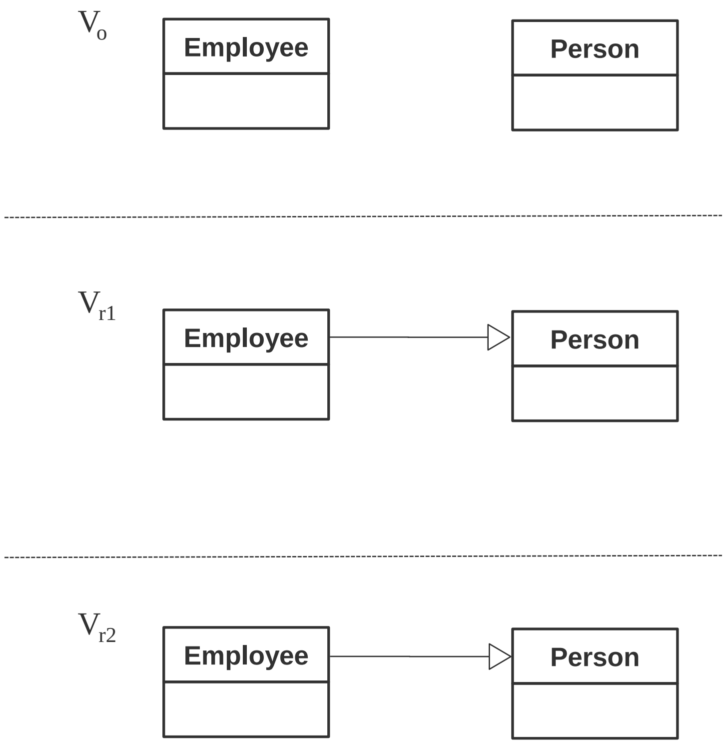


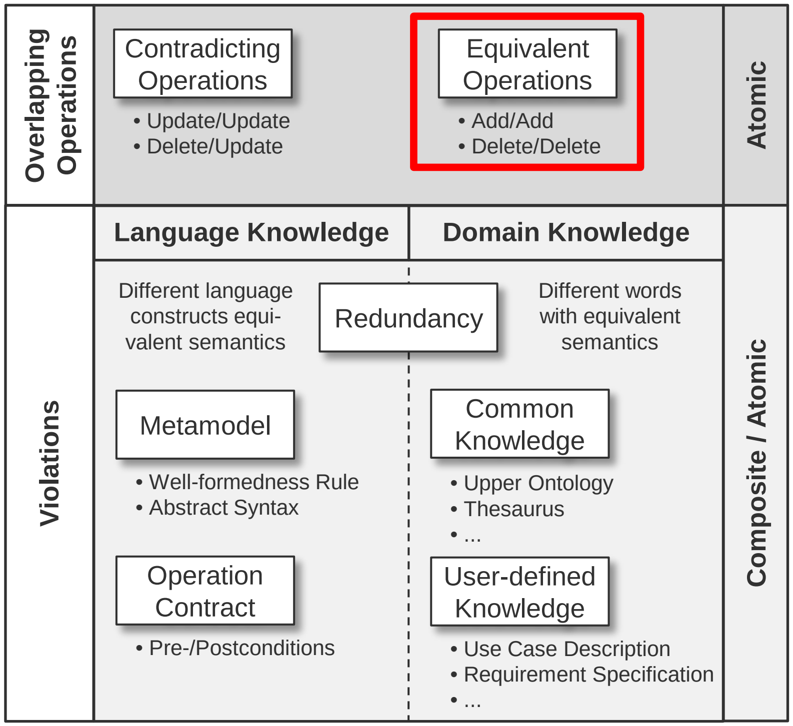
Contradicting Operations: Update/Update



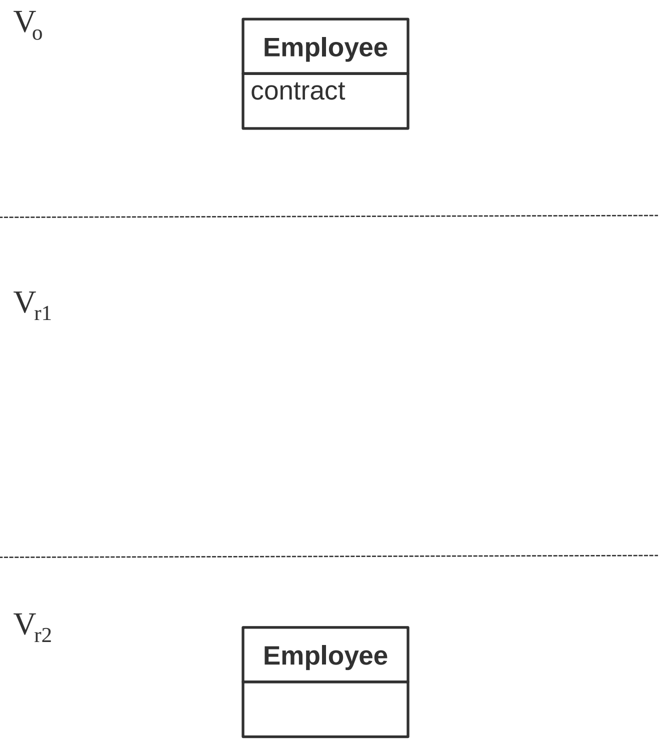


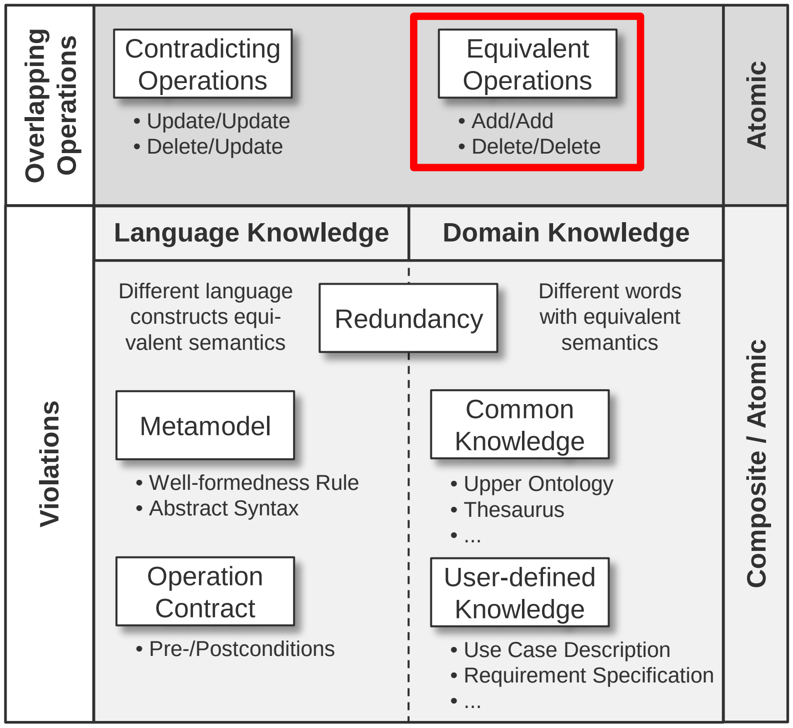
Equivalent Operations: Add/Add



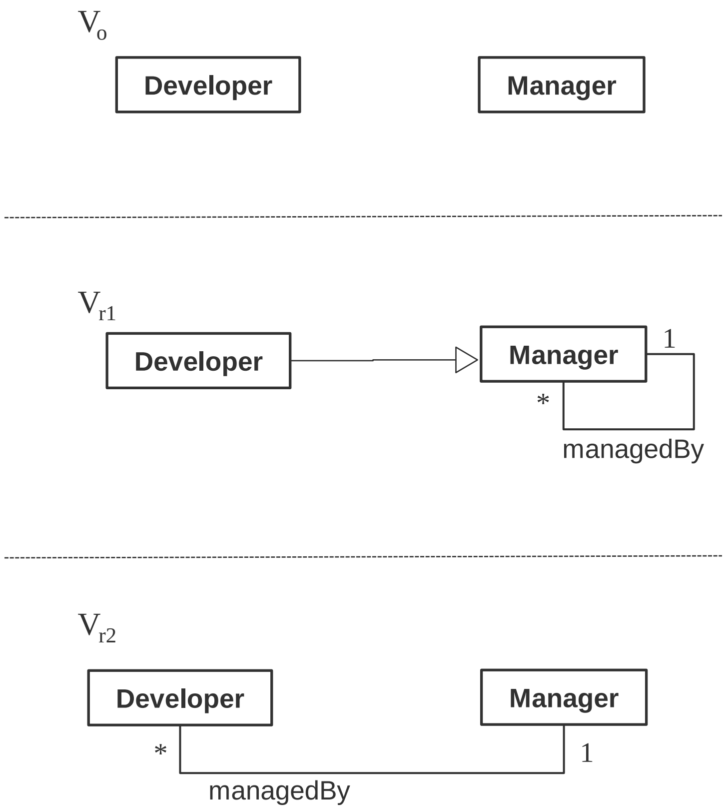


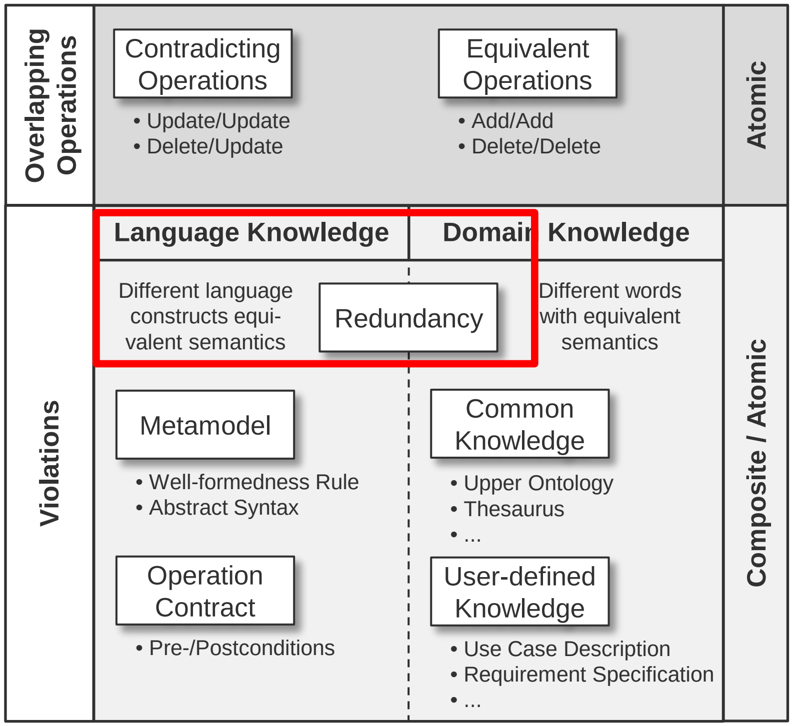
Equivalent Operations: Delete/Delete



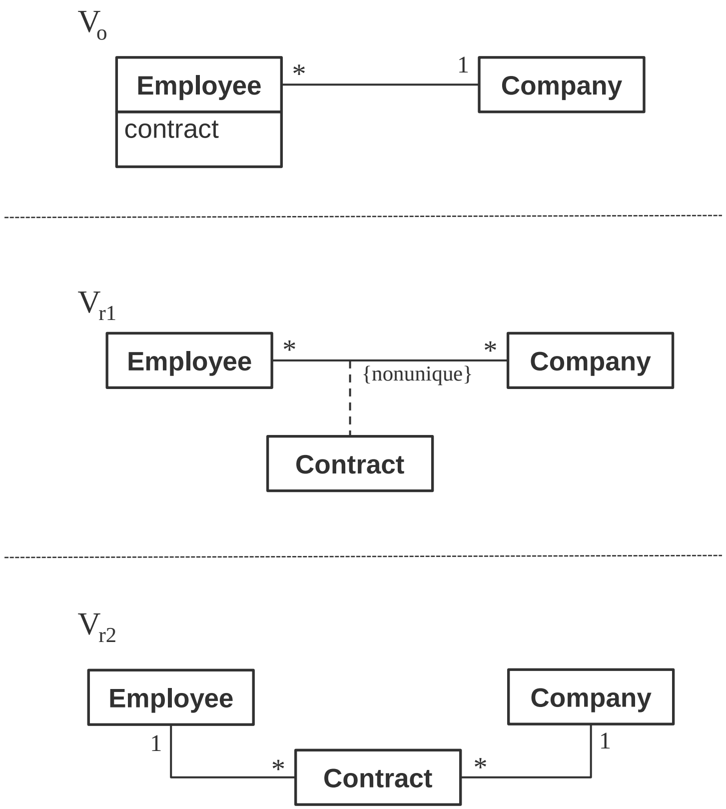


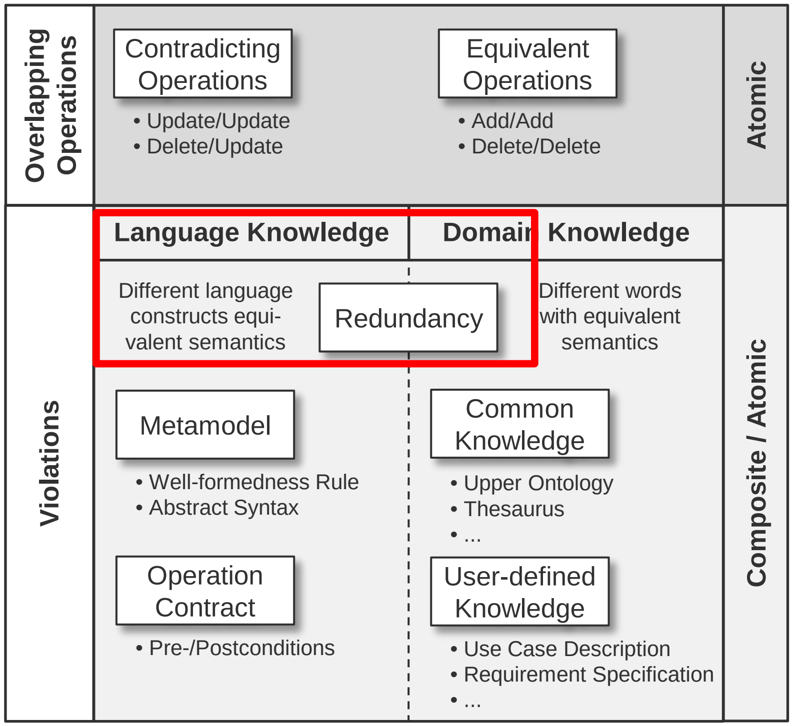
Redundancy: Language Knowledge



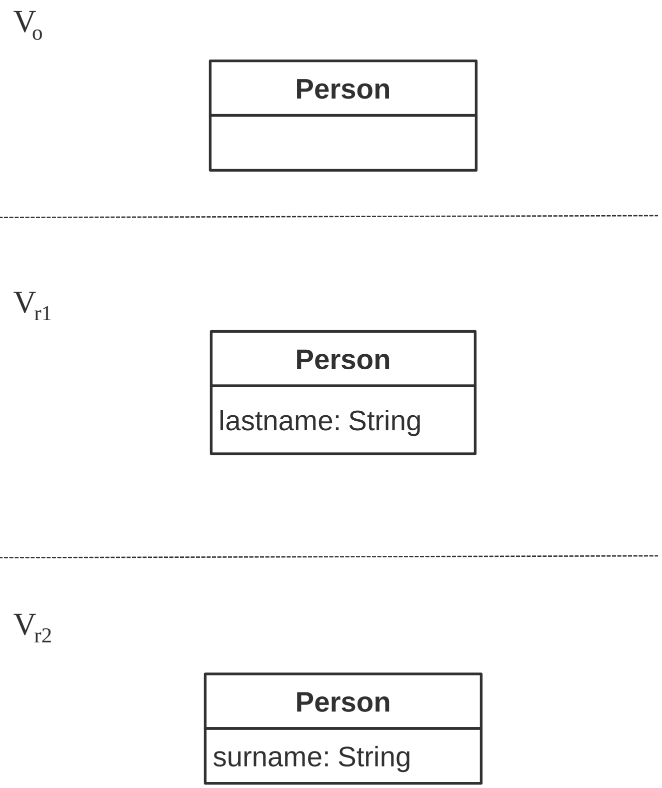


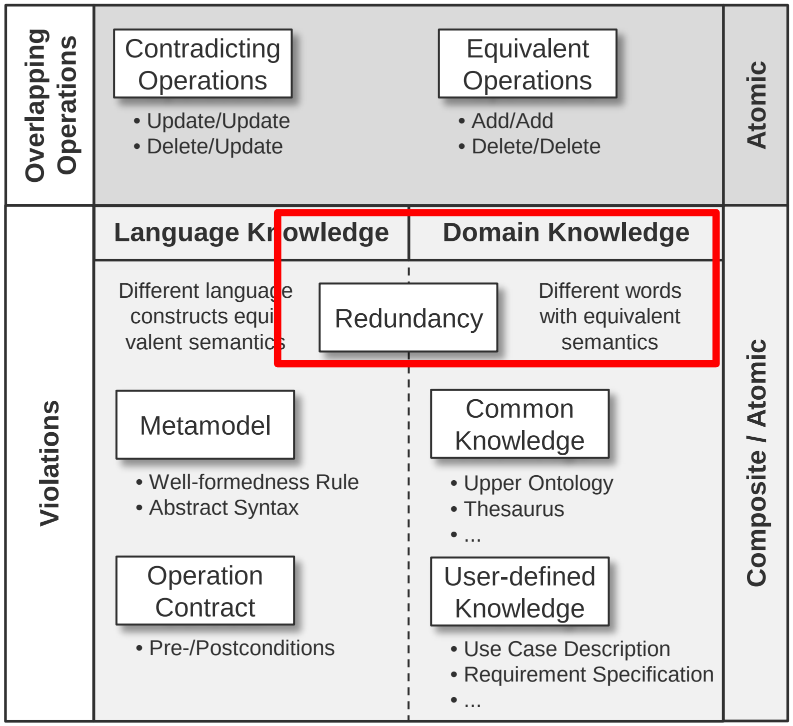
Redundancy: Language Knowledge



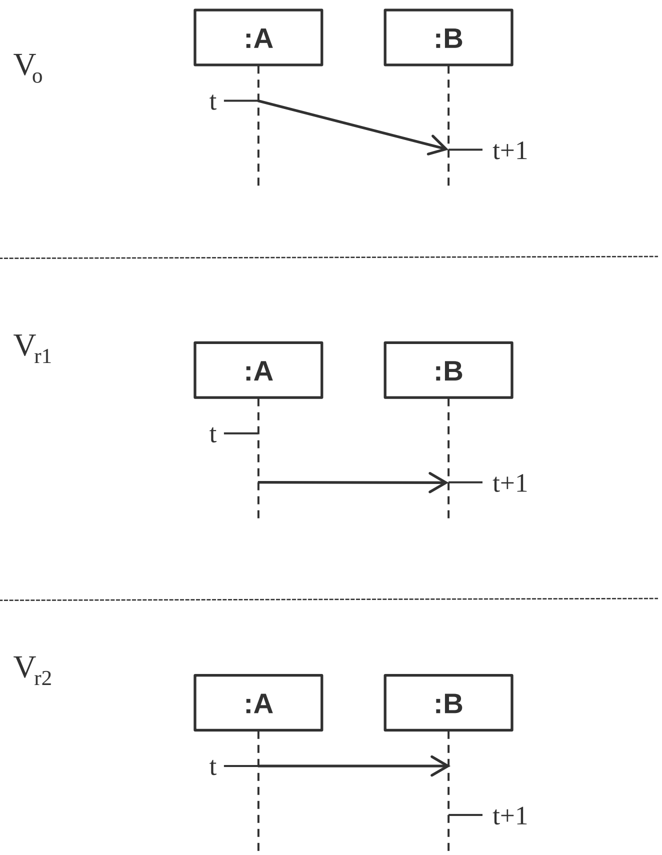


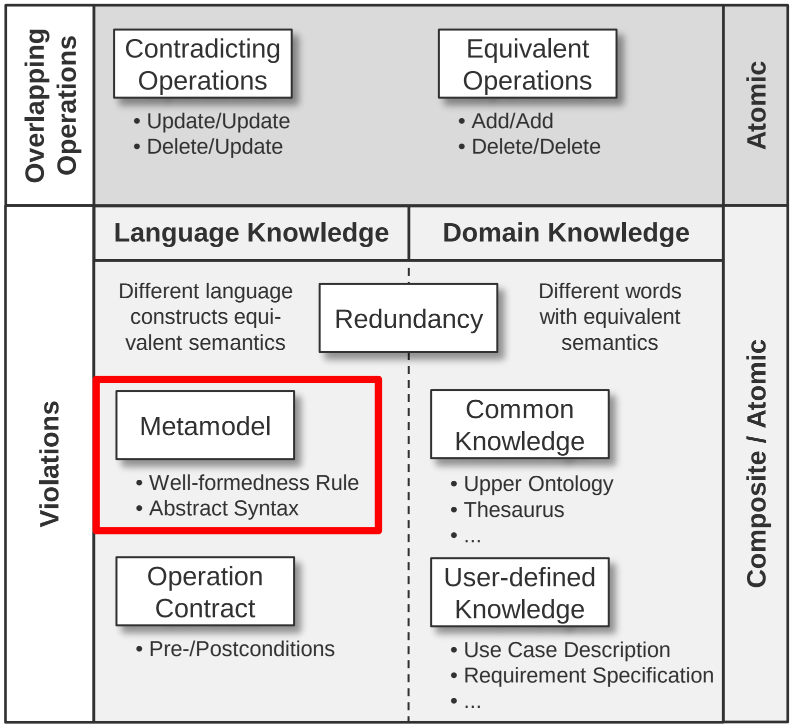
Redundancy: Domain Knowledge



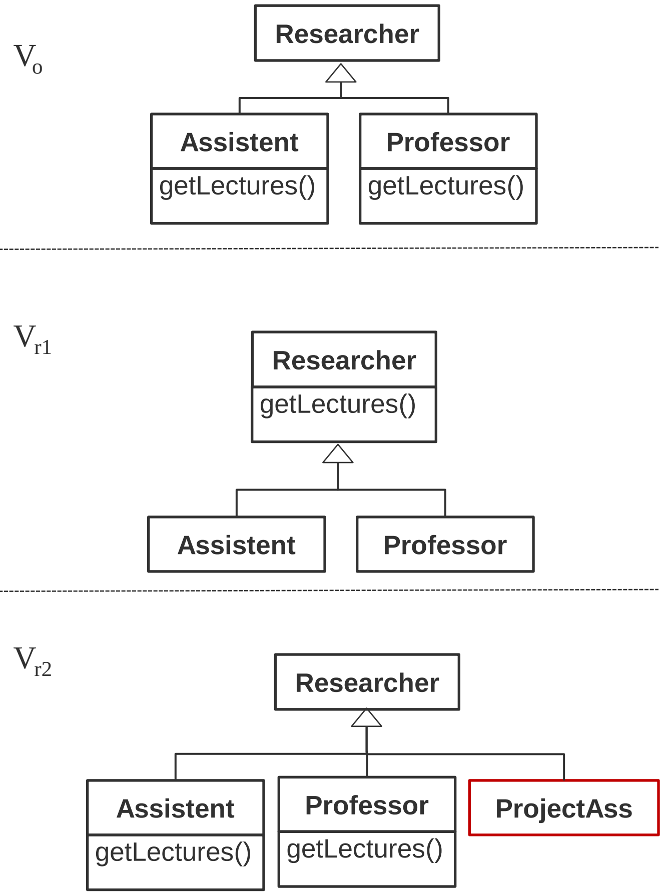


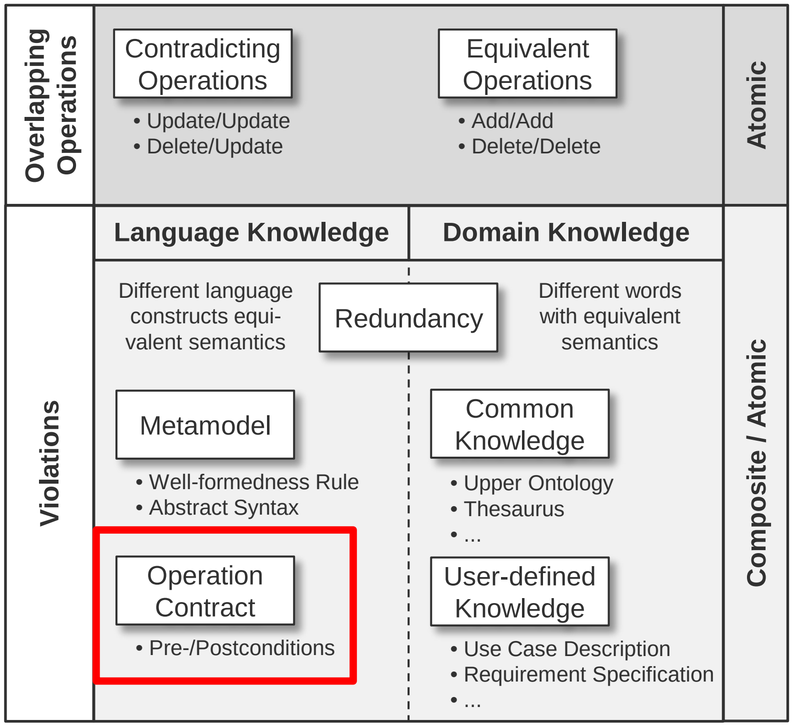
Metamodel Violation



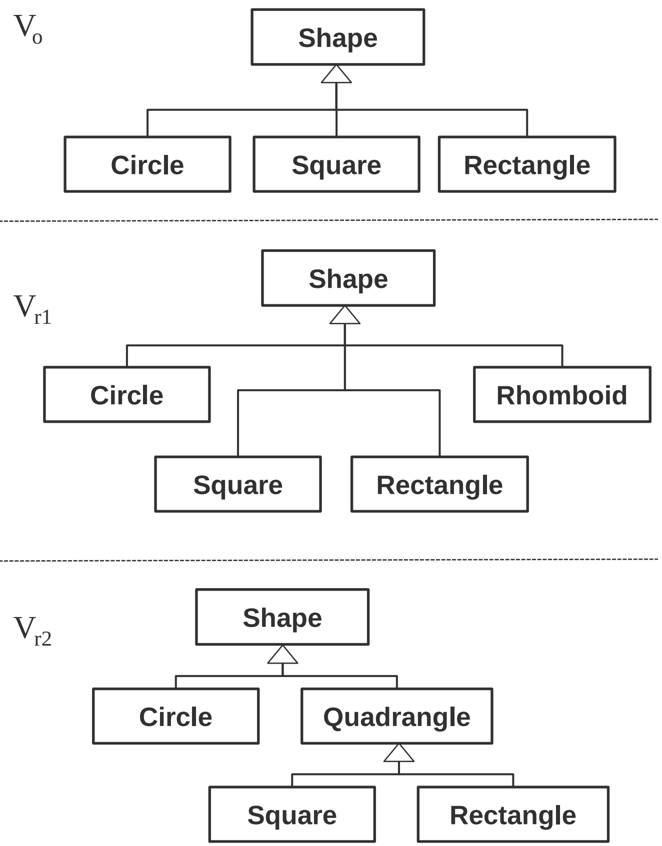


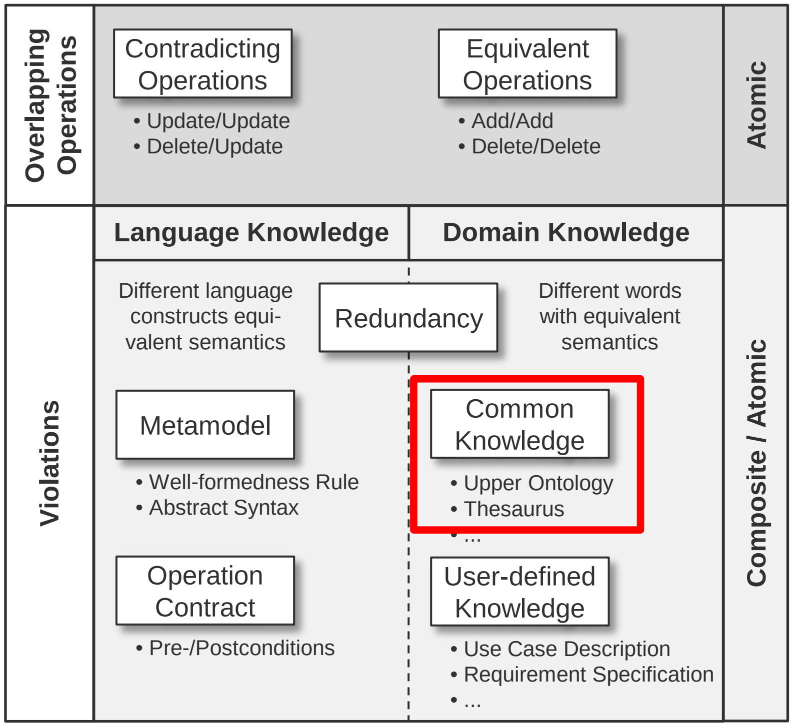
Operation Contract Violation



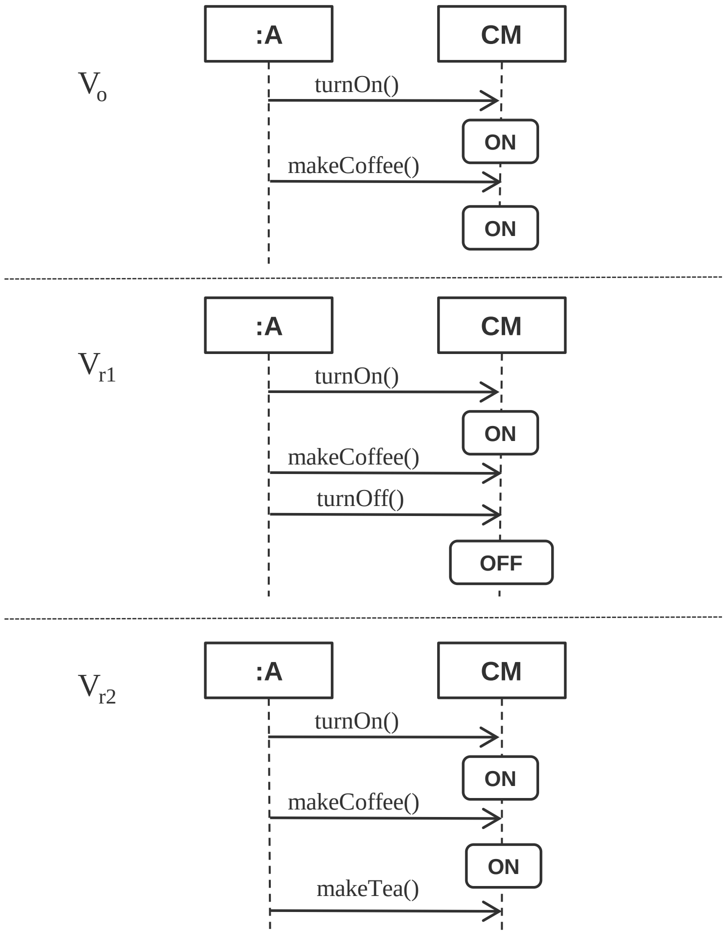


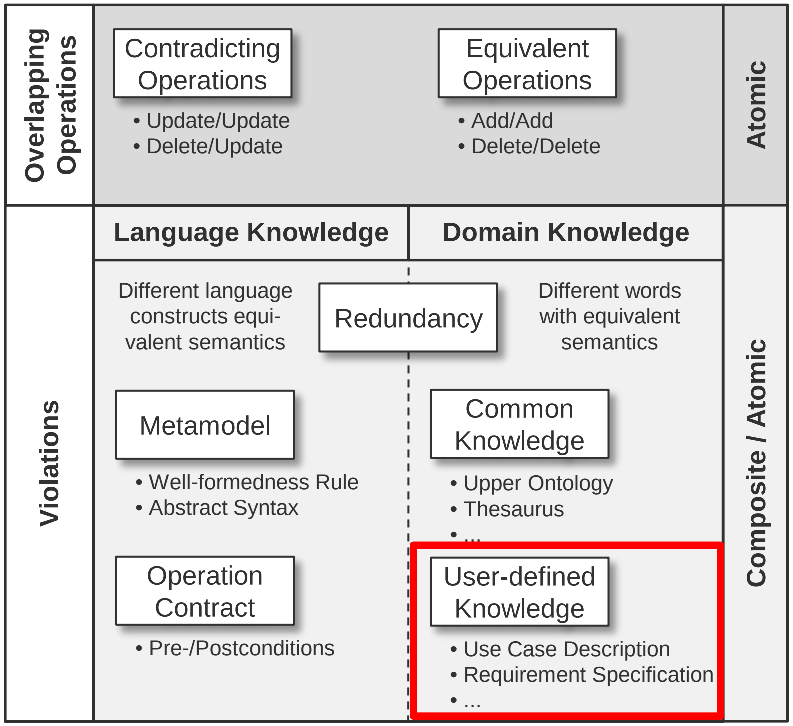
Common Knowledge Violation





User-defined Knowledge Violation





Outline

- Context of Model Versioning
- Foundations of Model Versioning
- Conflict Categorization
- **Adaptable Model Versioning**
- Future Challenges

AMOR: Adaptable Model Versioning

Overview

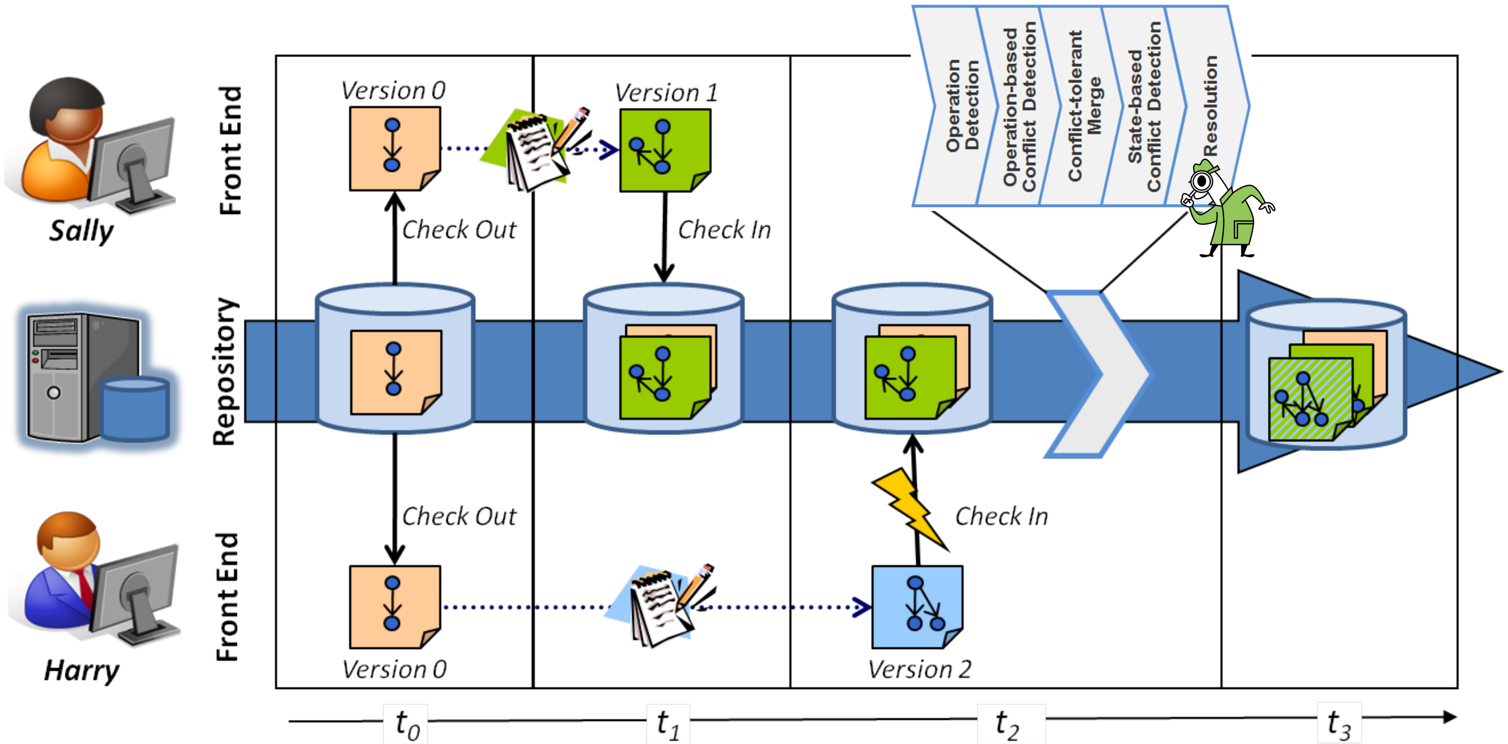
- FFG FIT-IT Semantic Systems Project
  - 2009 – 2011
- Collaborating parties
  - Vienna University of Technology
  - Johannes Kepler University, Linz
  - SparxSystems, the vendor of Enterprise Architect

www.modelversioning.org

- Three resulting dissertations
  - Petra Brosch, „Conflict Resolution in Model Versioning “, TU Wien, 2012.
  - Philip Langer, „Adaptable Model Versioning using Model Transformation By Demonstration“, TU Wien, 2011.
  - Konrad Wieland, „Conflict-tolerant Model Versioning“, TU Wien, 2011.

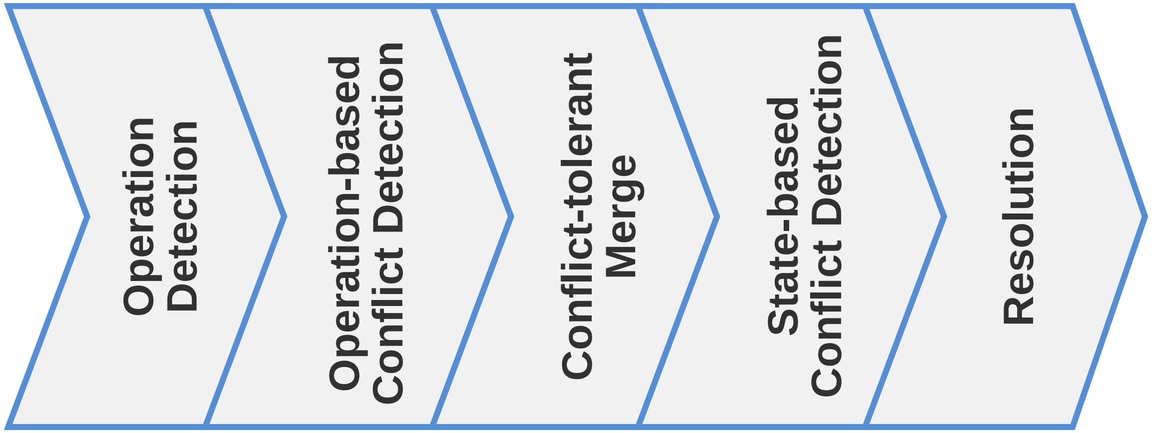
Adaptable Model Versioning

Overview



Adaptable Model Versioning

Versioning Process

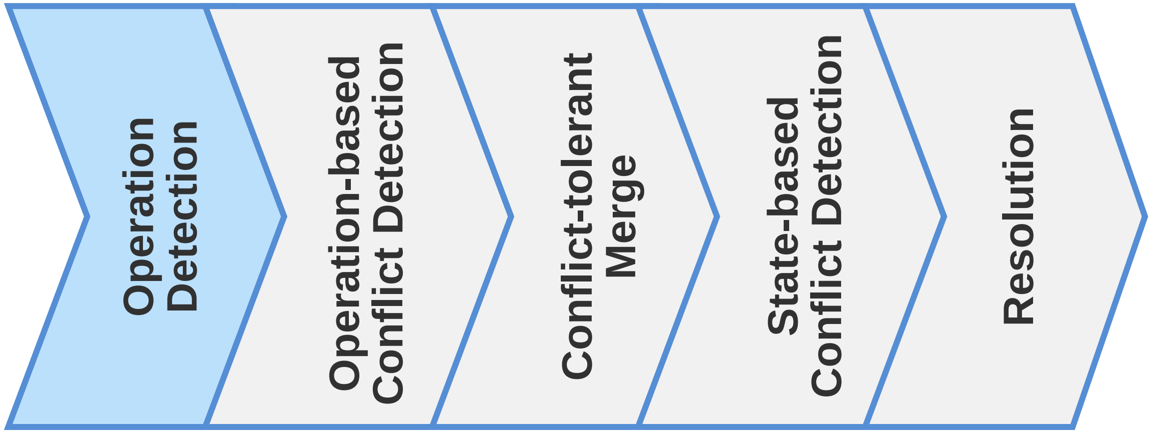


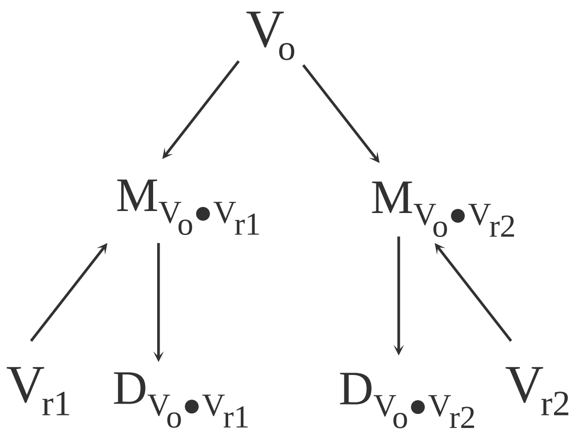
V_o

V_{r1} V_{r2}

Adaptable Model Versioning

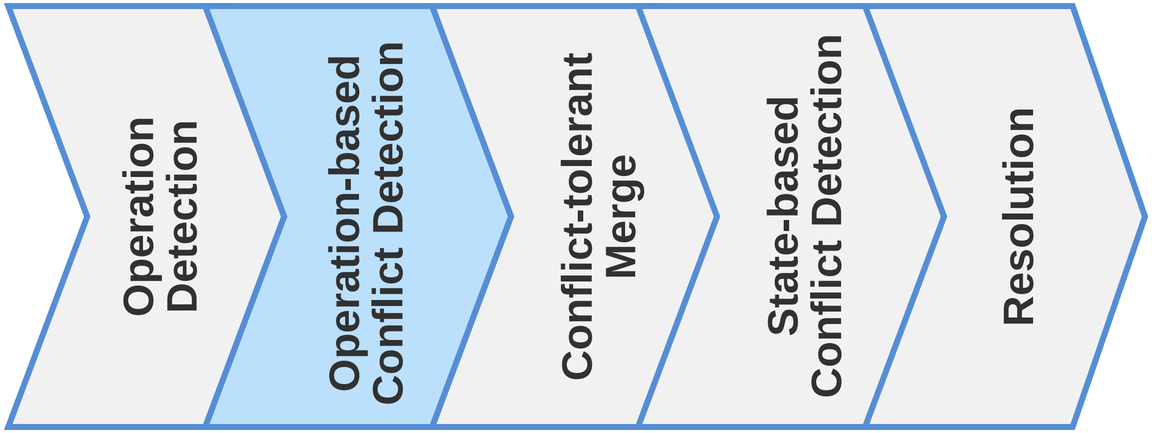
Versioning Process

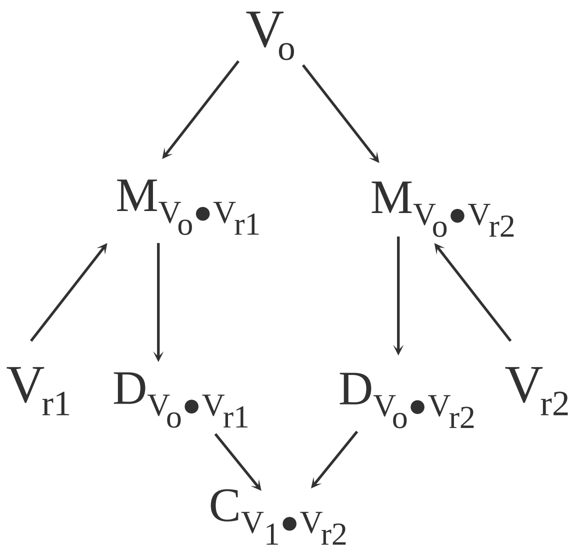




Adaptable Model Versioning

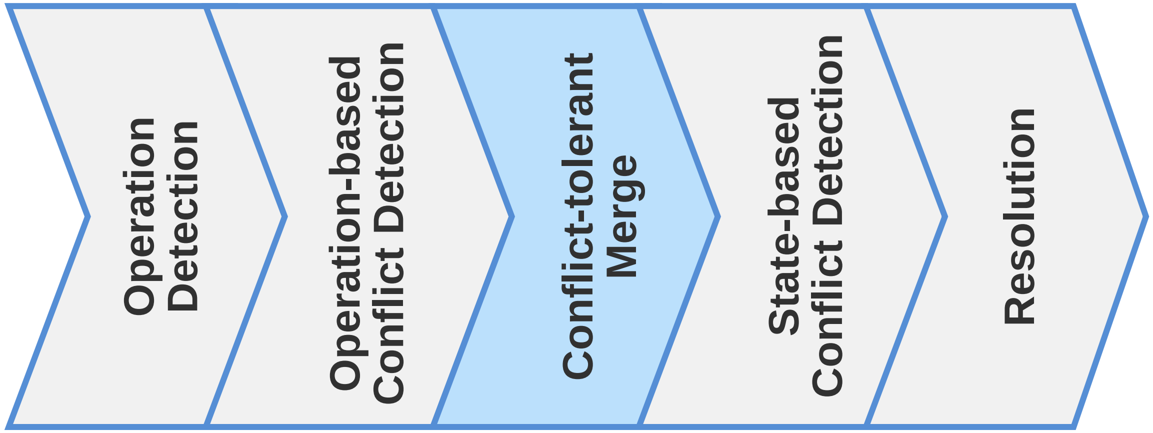
Versioning Process

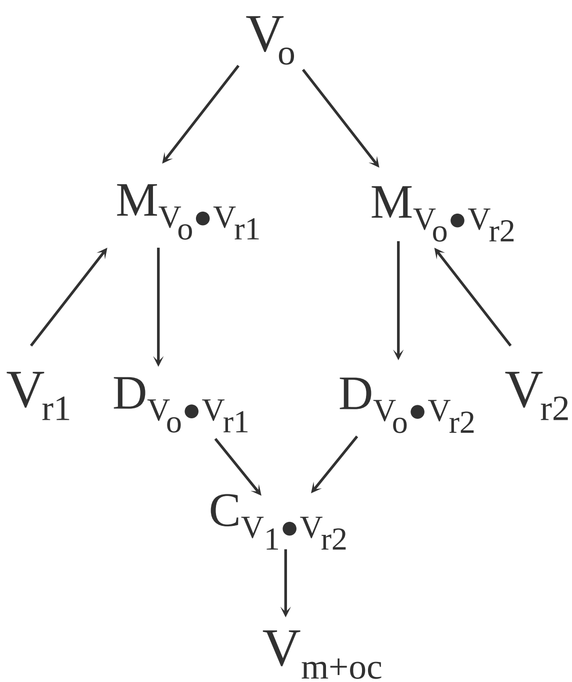




Adaptable Model Versioning

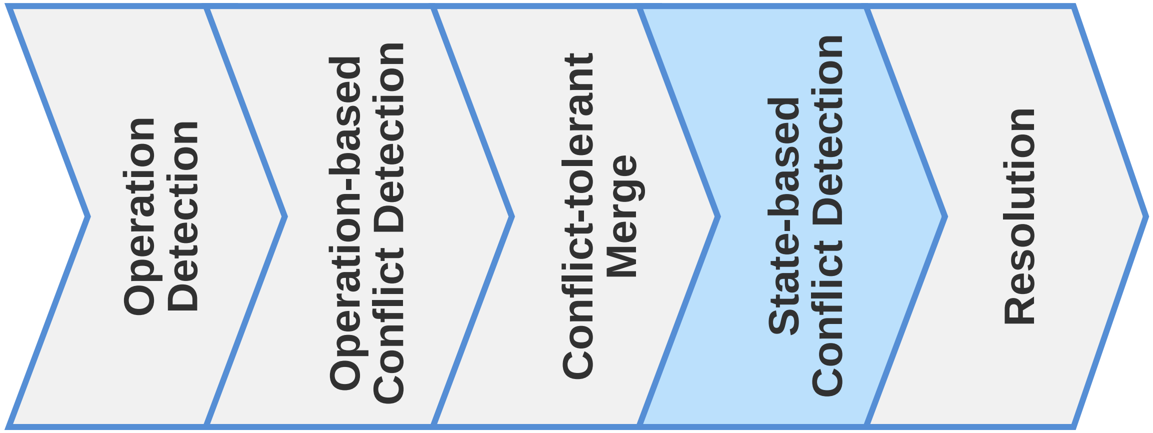
Versioning Process

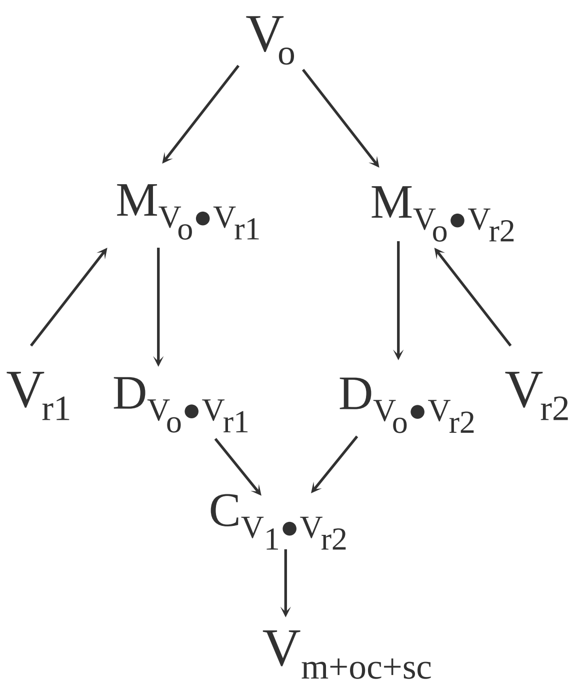




Adaptable Model Versioning

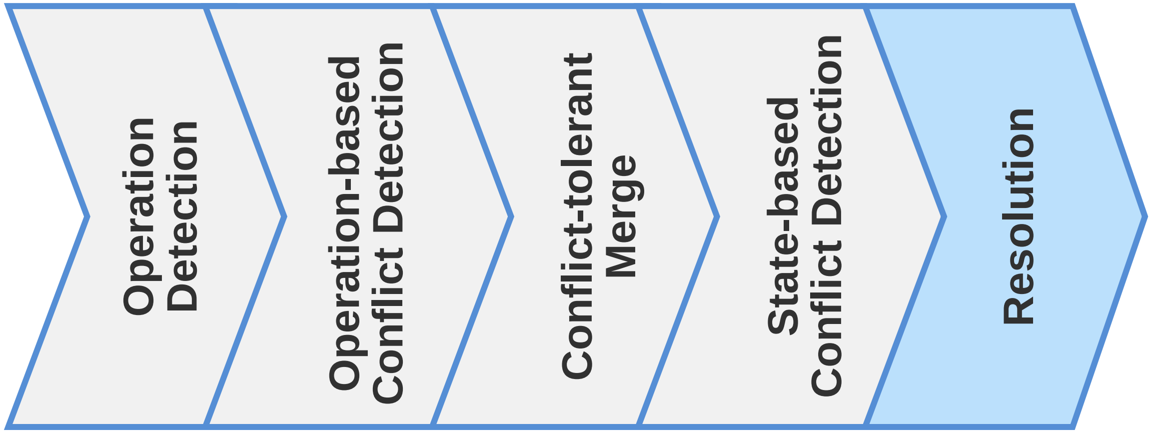
Versioning Process

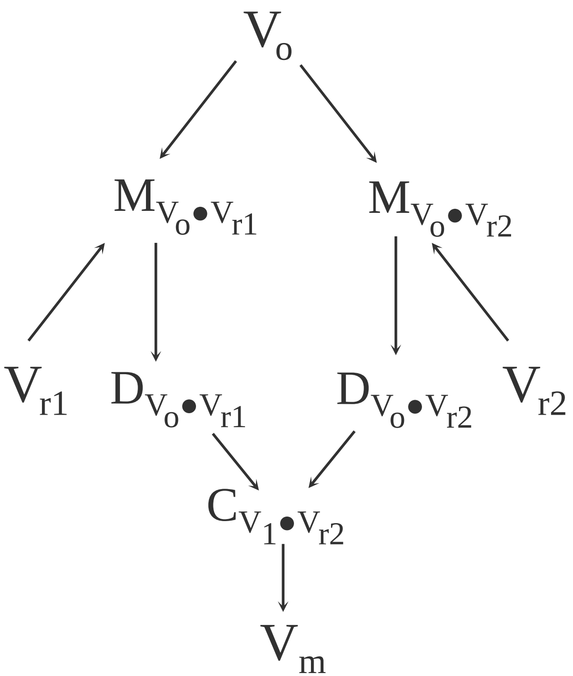




Adaptable Model Versioning

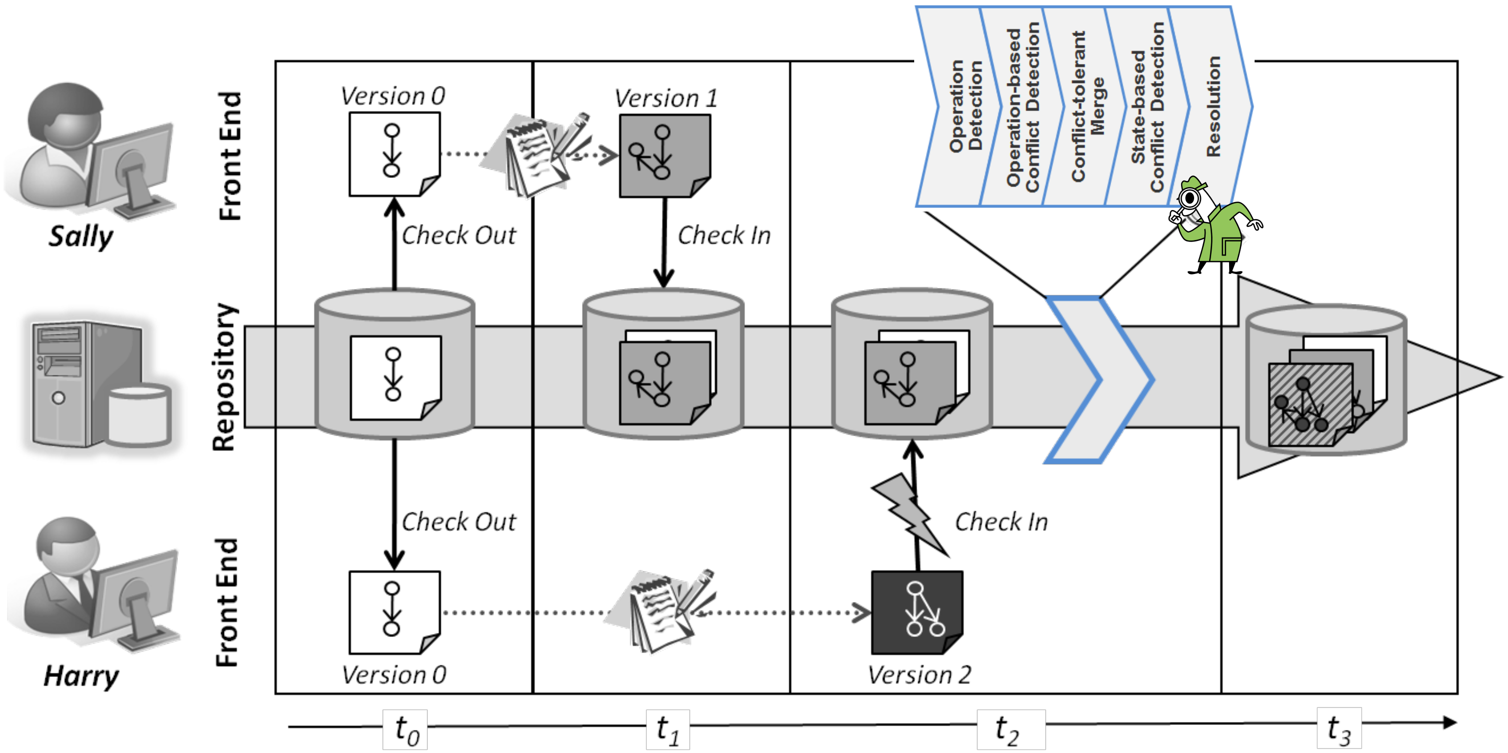
Versioning Process





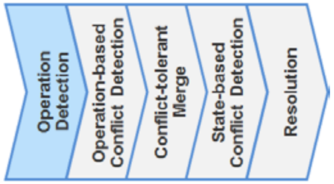
Adaptable Model Versioning

Versioning Process: Step by step



Operation Detection

Two approaches for obtaining applied operations



- Model Differencing
  - **Comparison** of **states** of an artifact
  - **Match function** to find correspondence of two elements in compared artifacts necessary
  - Differences are converted in atomic operations
  - Editor and language independent
  - **Composite operation** detection difficult, but possible in many cases

- Operation Recording
  - **Records** operation sequences
  - Recording of **atomic operations** (*add, update, delete*)
  - Recording of **composite operations** possible if available in the editor
  - Cleansing to remove obsolete operations for more efficient merging
  - Strong editor dependence

Operation Detection

Overview

- Two-phase process
  - **Phase 1**: Matching for finding correspondences between objects
  - **Phase 2**: Fine-grained comparison of corresponding objects
  - **Phase 3**: Detection of *composite* operation applications

Phase 1 $\quad V_o \longleftarrow M_{V_o \bullet V_{r1}} \longrightarrow V_{r1}$

Phase 2 $\quad D_{V_o \bullet V_{r1}}$

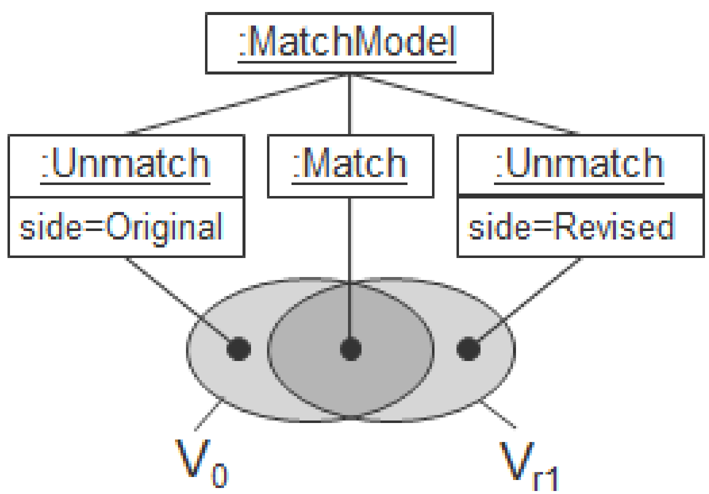
Phase 3 $\quad D^{cop}_{V_o \bullet V_{r1}}$

Operation Detection: Phase 1 – Matching

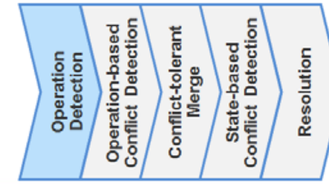
Goal

- Match function is needed
  1. Universally Unique ID (UUID)
  2. Heuristics to compute similarity measures based on features of objects

- Quality of operation detection depends on quality of the match

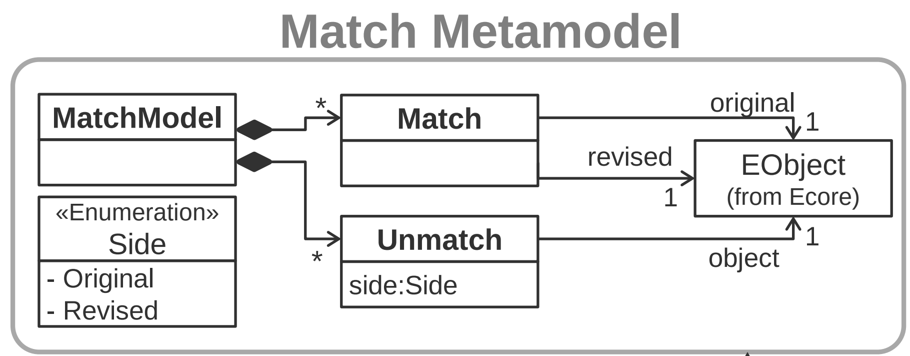


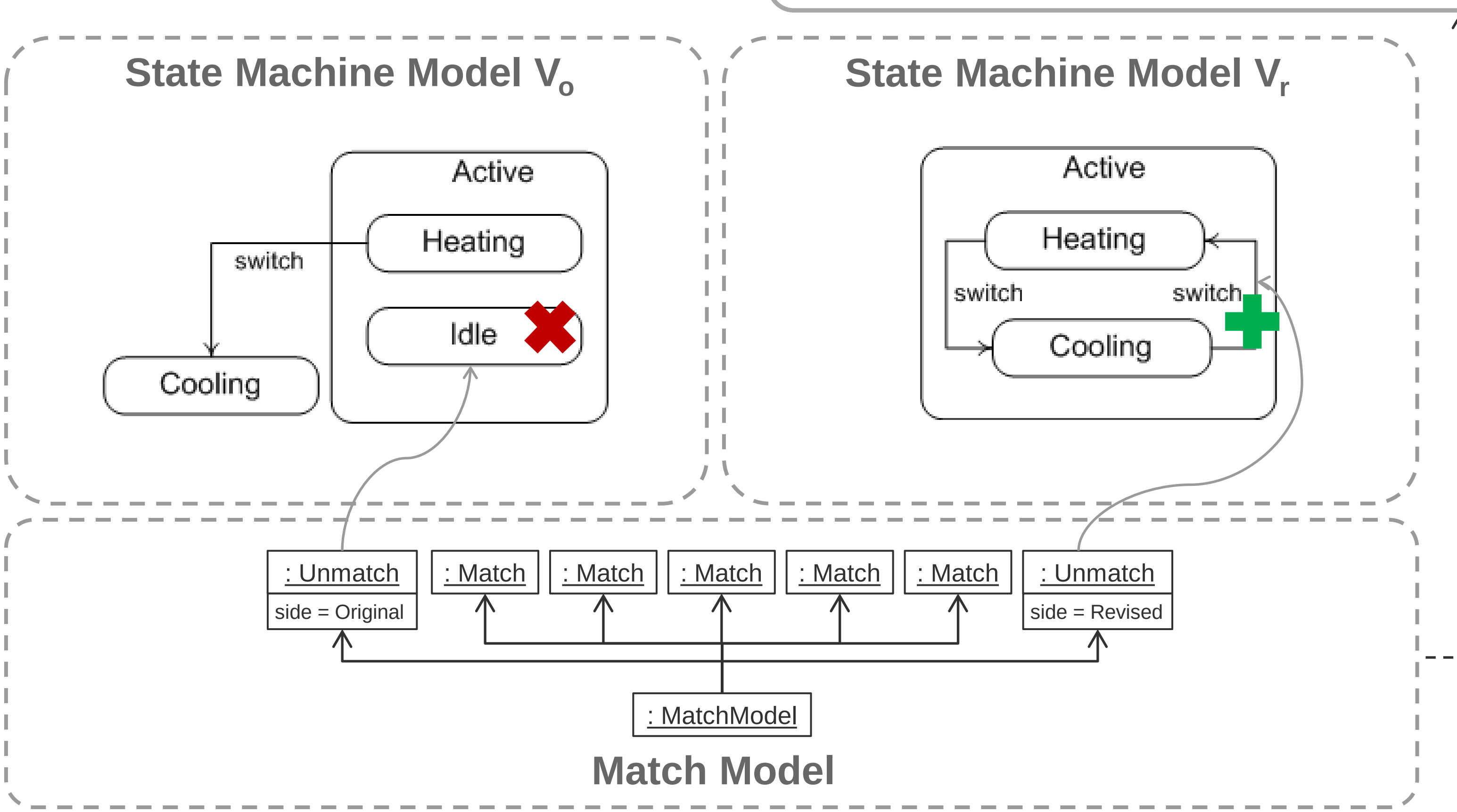
Operation Detection: Phase 1 – Matching



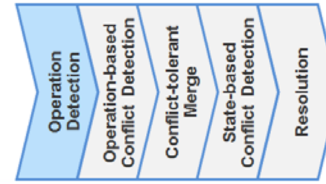
Metamodel and Example Model

- Correspondences are described by a match model (weaving model)
  - Links corresponding elements
  - Marks unmatched elements





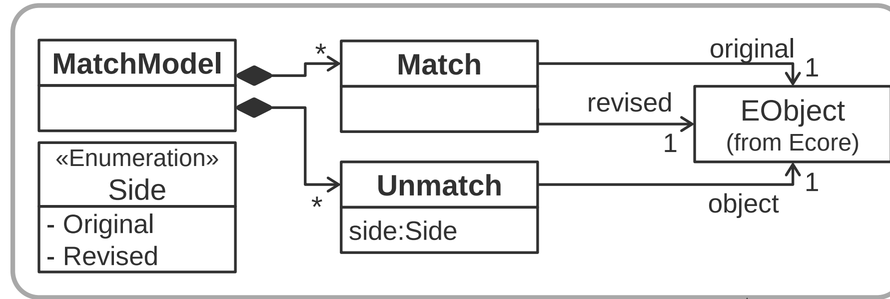
Operation Detection: Phase 1 – Matching

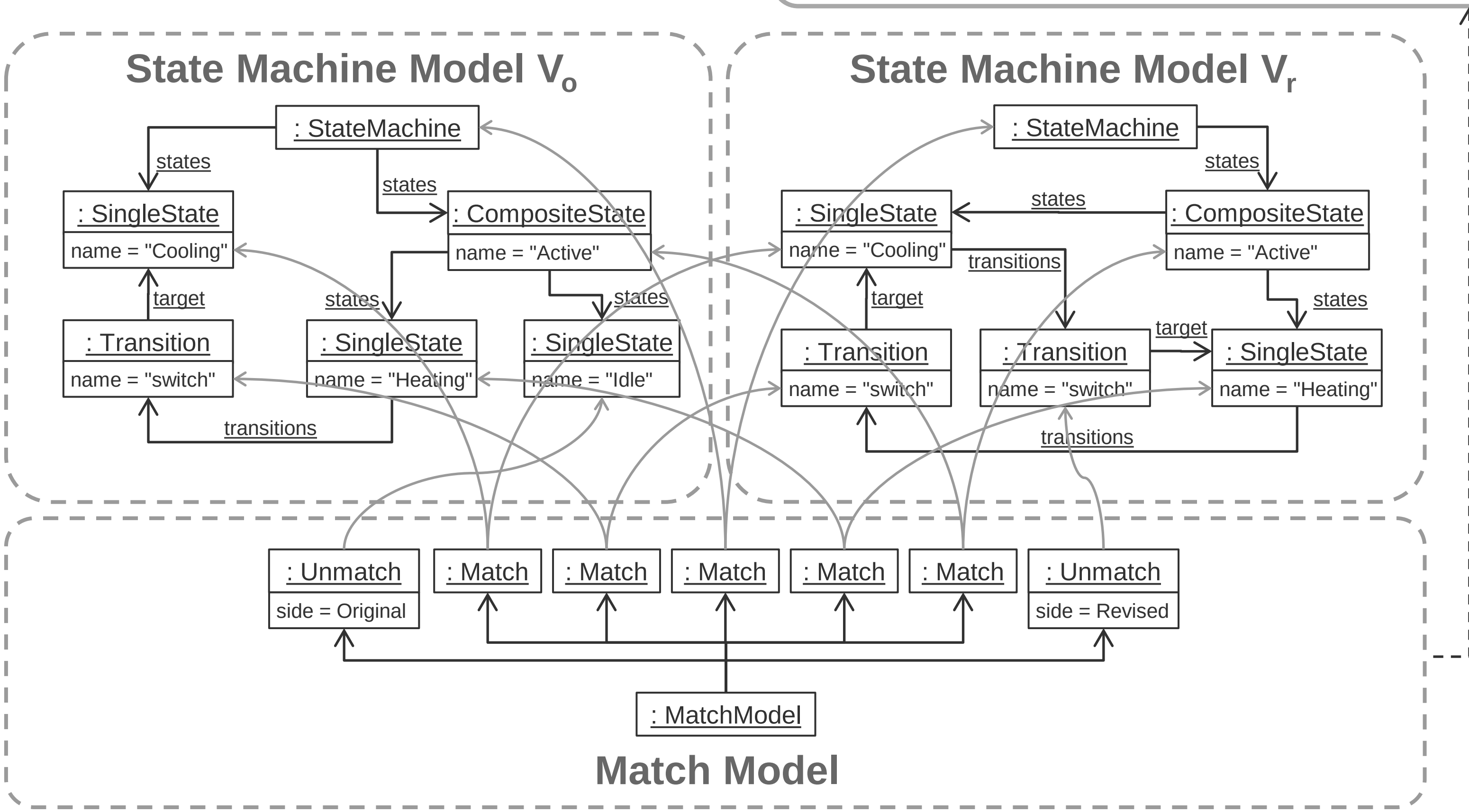


Metamodel and Example Model

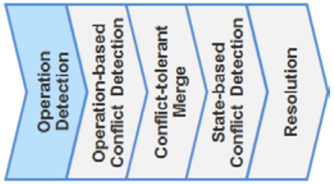
- Correspondences are described by a match model (weaving model)
  - Links corresponding elements
  - Marks unmatched elements

Match Metamodel

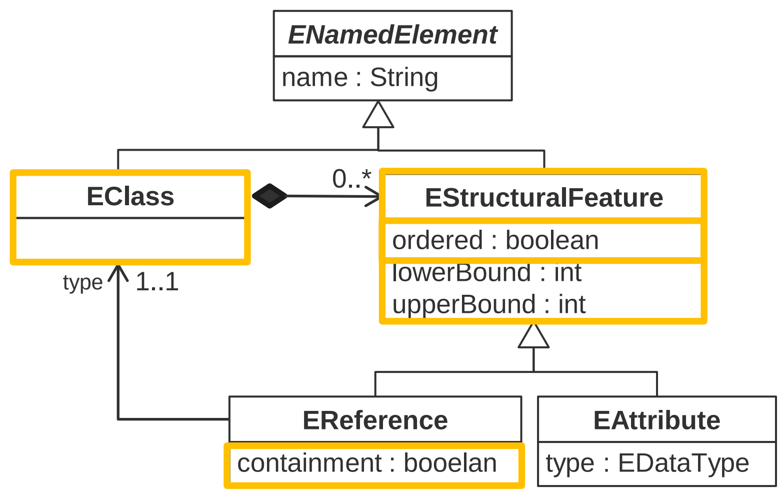




Operation Detection: Phase 2 – Comparison

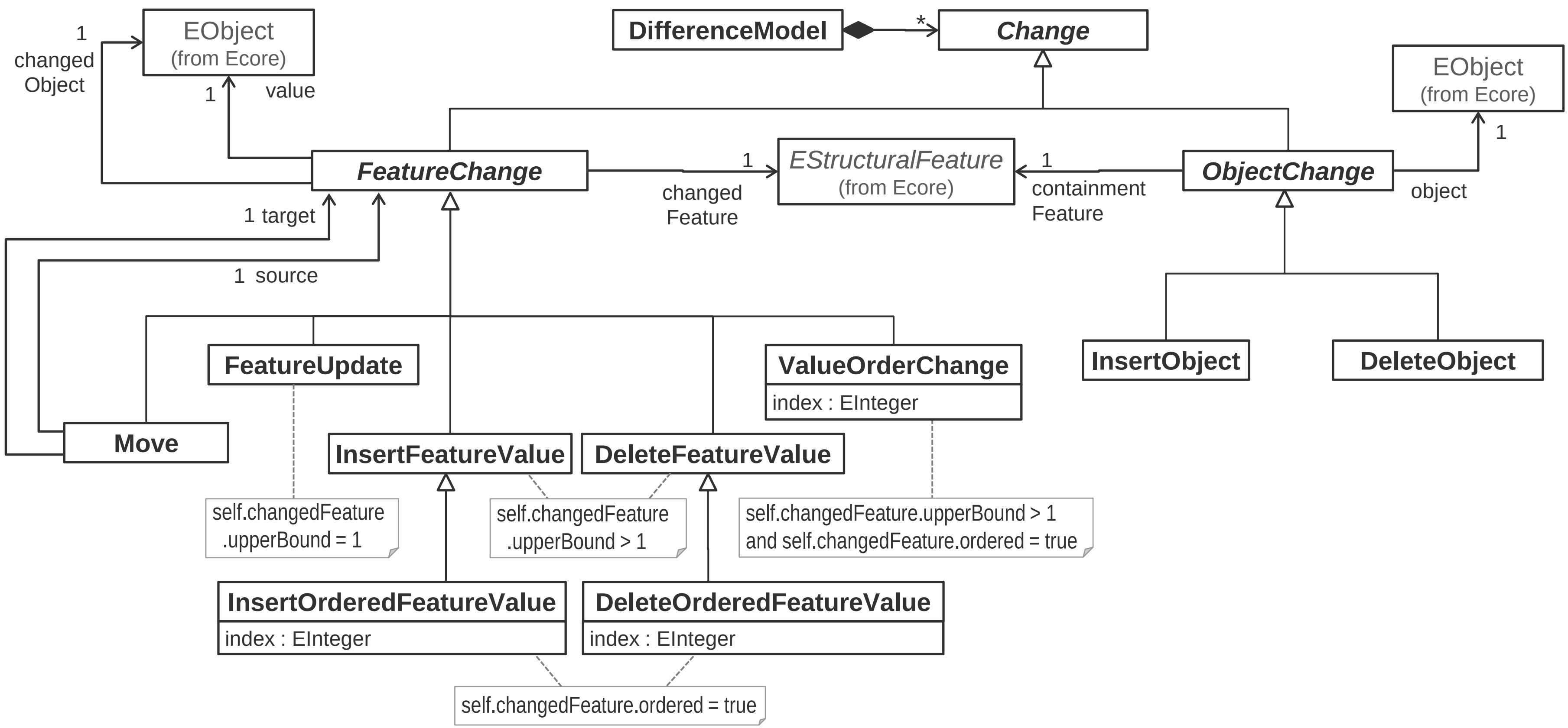


- Deleted and inserted objects are explicitly marked by match model
  - But what is with structural feature changes?
- Fine-grained operation types depend on the metamodeling features
  - E.g., Ecore offers ordered features, etc.
- Supported operations
  - Insert Object
  - Delete Object
  - Feature Update
  - Insert Feature Value
  - Delete Feature Value
  - *Insert Ordered Feature Value*
  - *Delete Ordered Feature Value*
  - *Feature Order Change*
  - Move



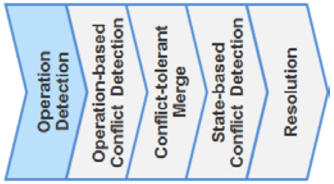
Operation Detection: Phase 2 – Comparison

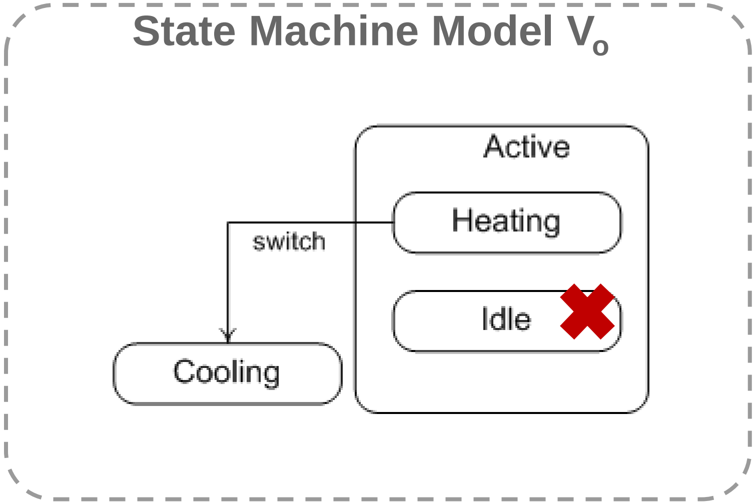
Difference Metamodel

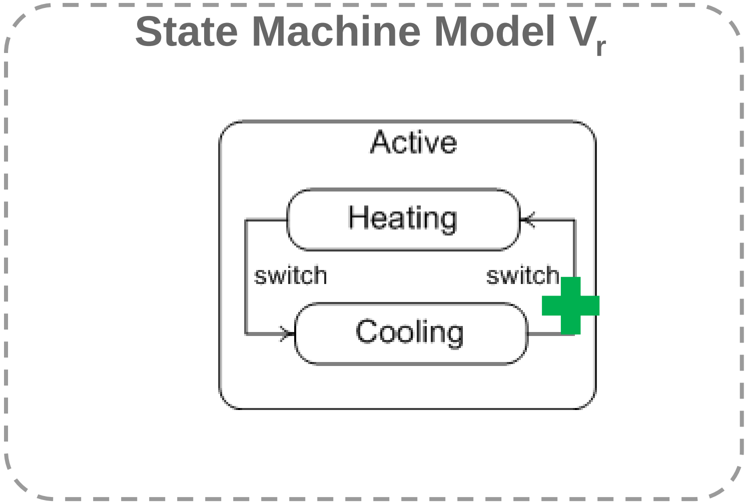


Operation Detection: Phase 2 – Comparison

Difference Model Example

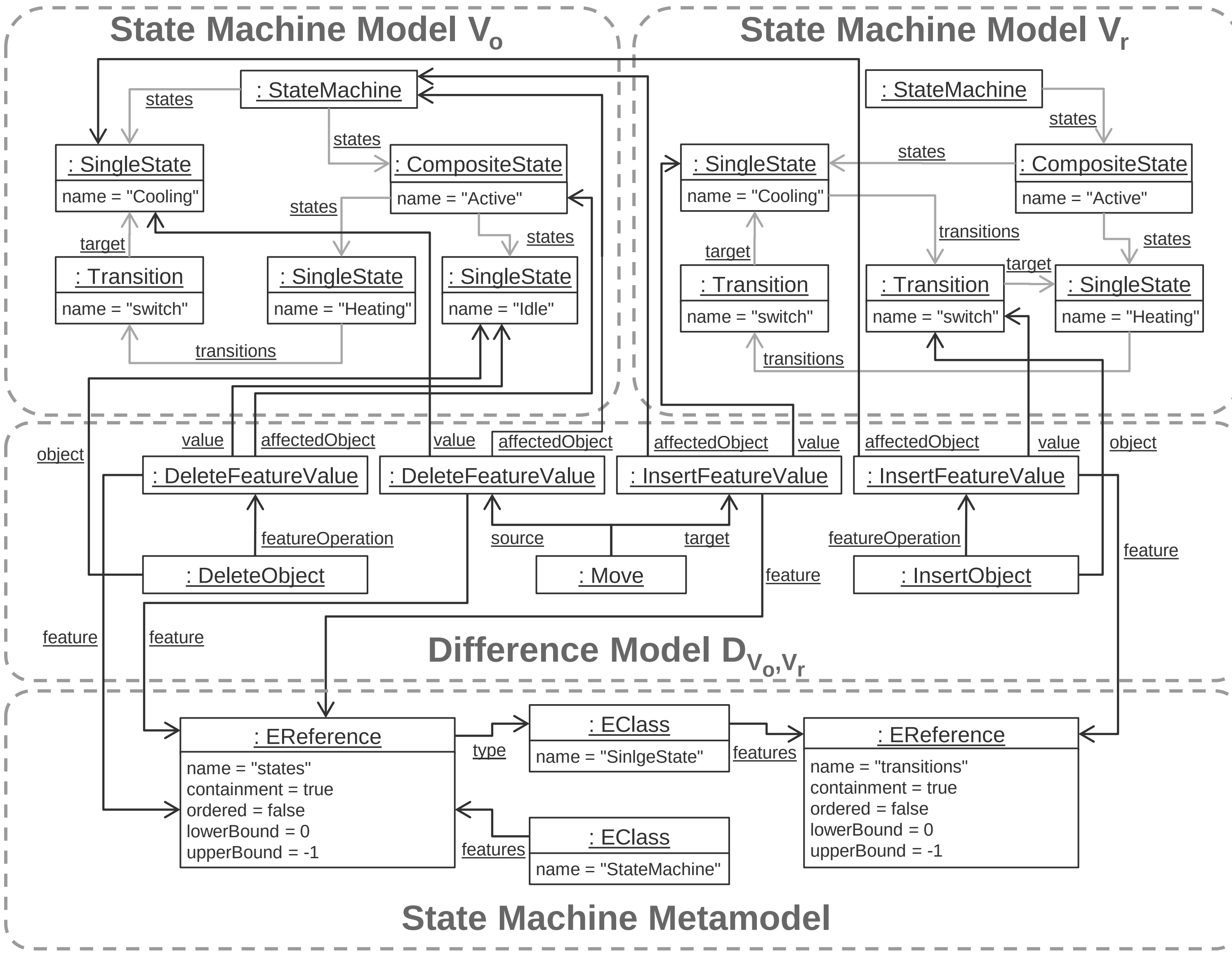






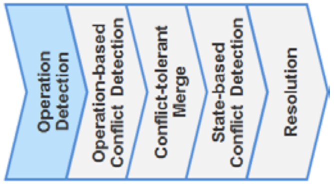
Operation Detection: Phase 2 – Comparison

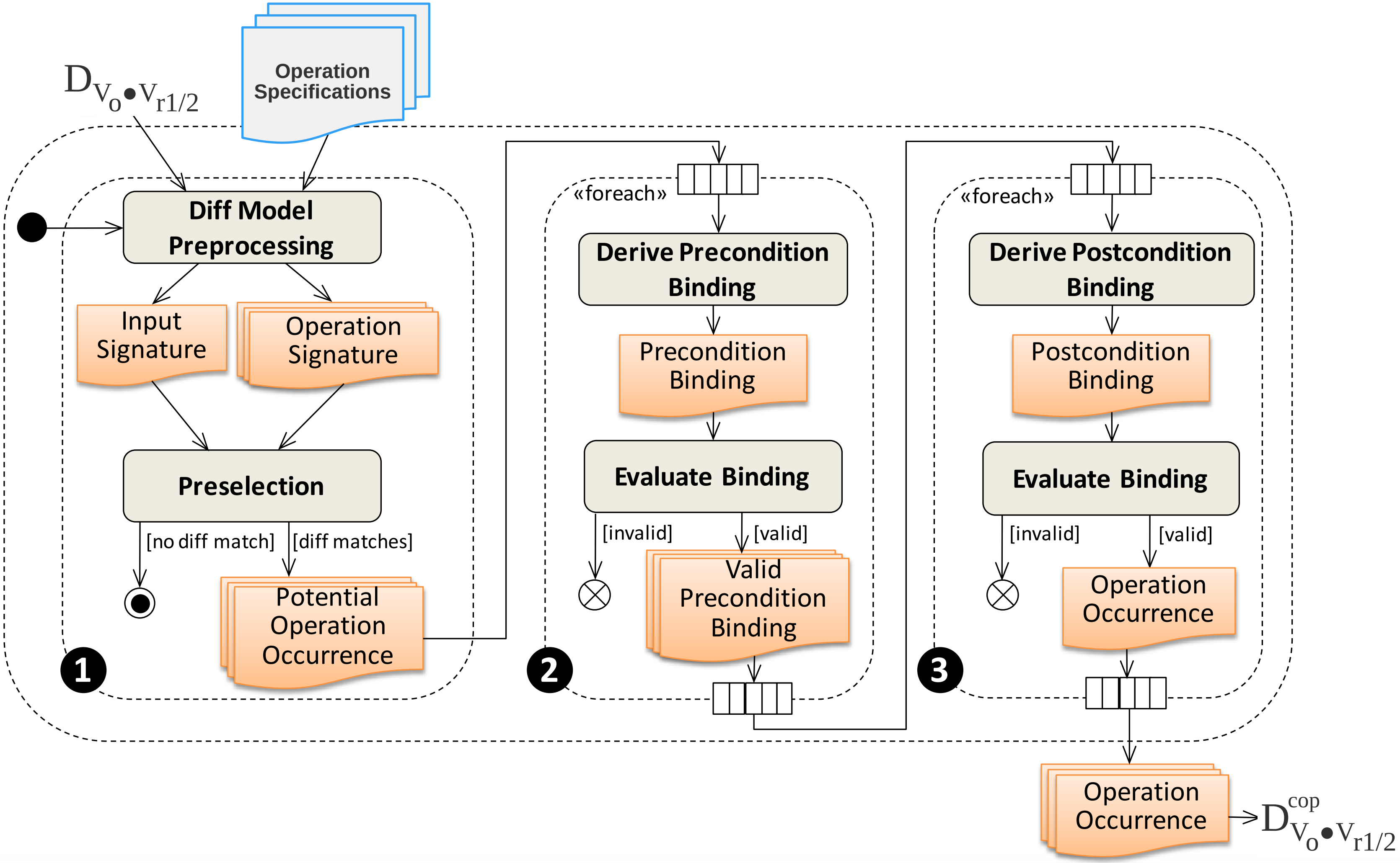
Difference Model Example



Operation Detection: Phase 3

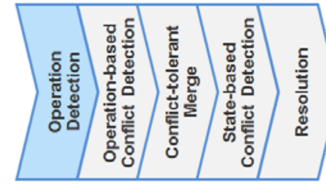
Composite operation detection process



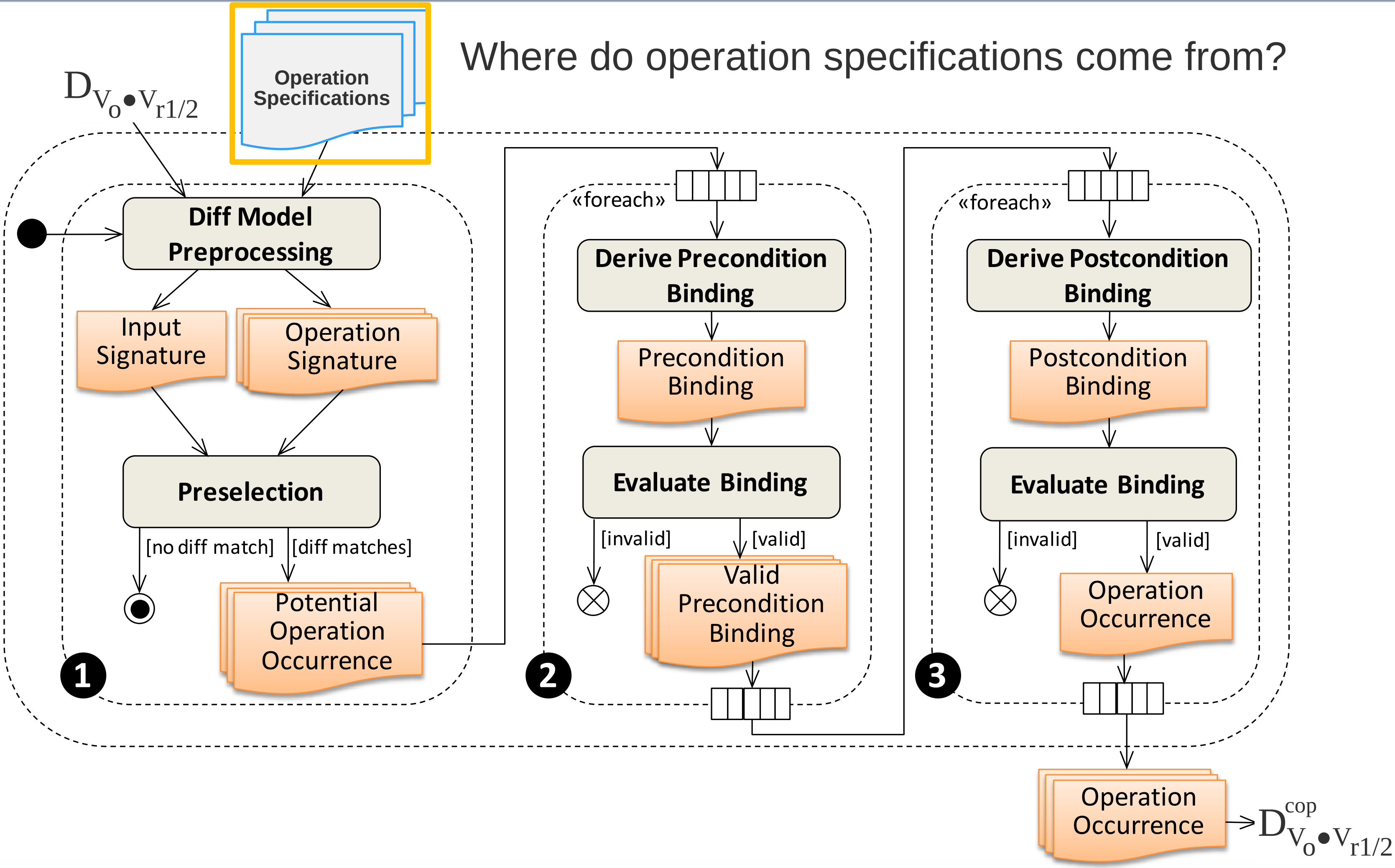


Operation Detection: Phase 3

Composite operation detection process



Where do operation specifications come from?



Operation Specifications

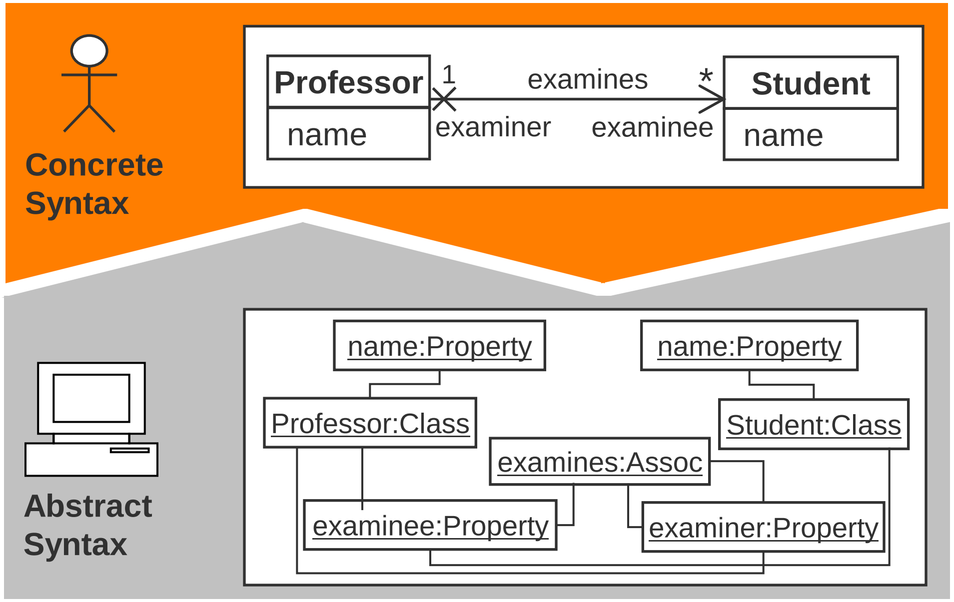
How to specify operation specifications

- Operation Specifications
  - Define composite operations
- Composite operations
  - Set of cohesive atomic operations
  - Applied in a transaction
  - To fulfill a common goal
  - → Common goal should be respected during merge
  - → Thus, they should be detected
- Operation specifications are language-specific
  - A plethora of modeling languages exists
  - Each has its own composite operations
  - Infeasable for language engineers to pre-specify all of them
  - → Allow modelers themselves to specify them
- Operation specifications are model transformations
  - In particular, endogeneous in-place transformations

Operation Specifications

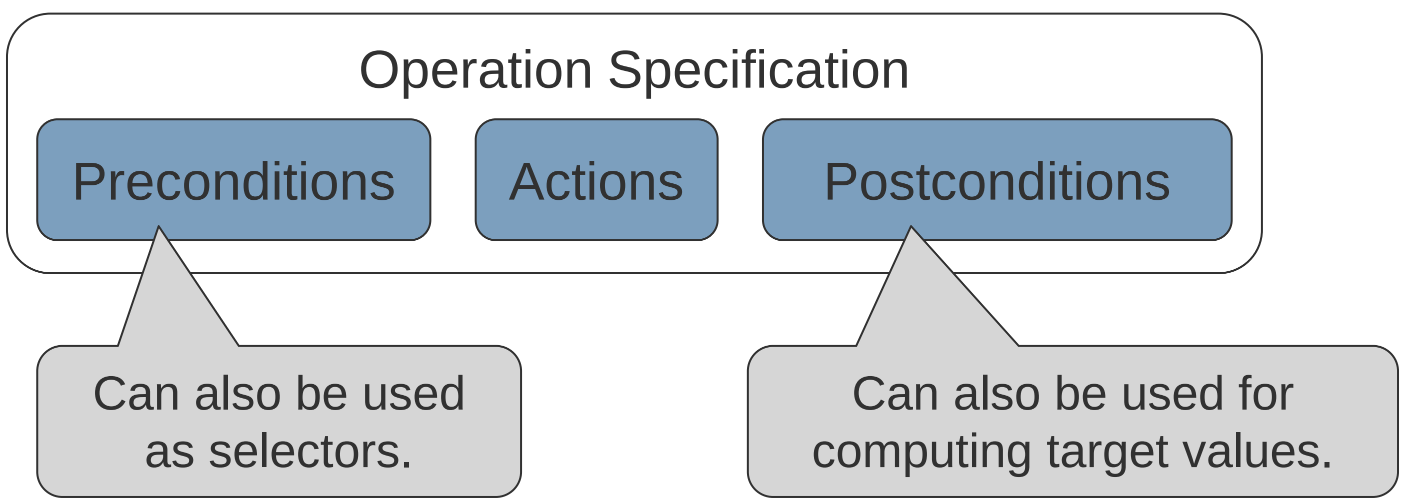
How to specify operation specifications

- Prerequisites for model transformation development
  - Experience in model transformation languages
  - Deep understanding of the involved metamodels
    - Common modelers are only aware of the concrete syntax
    - Model transformation languages are based on the abstract syntax

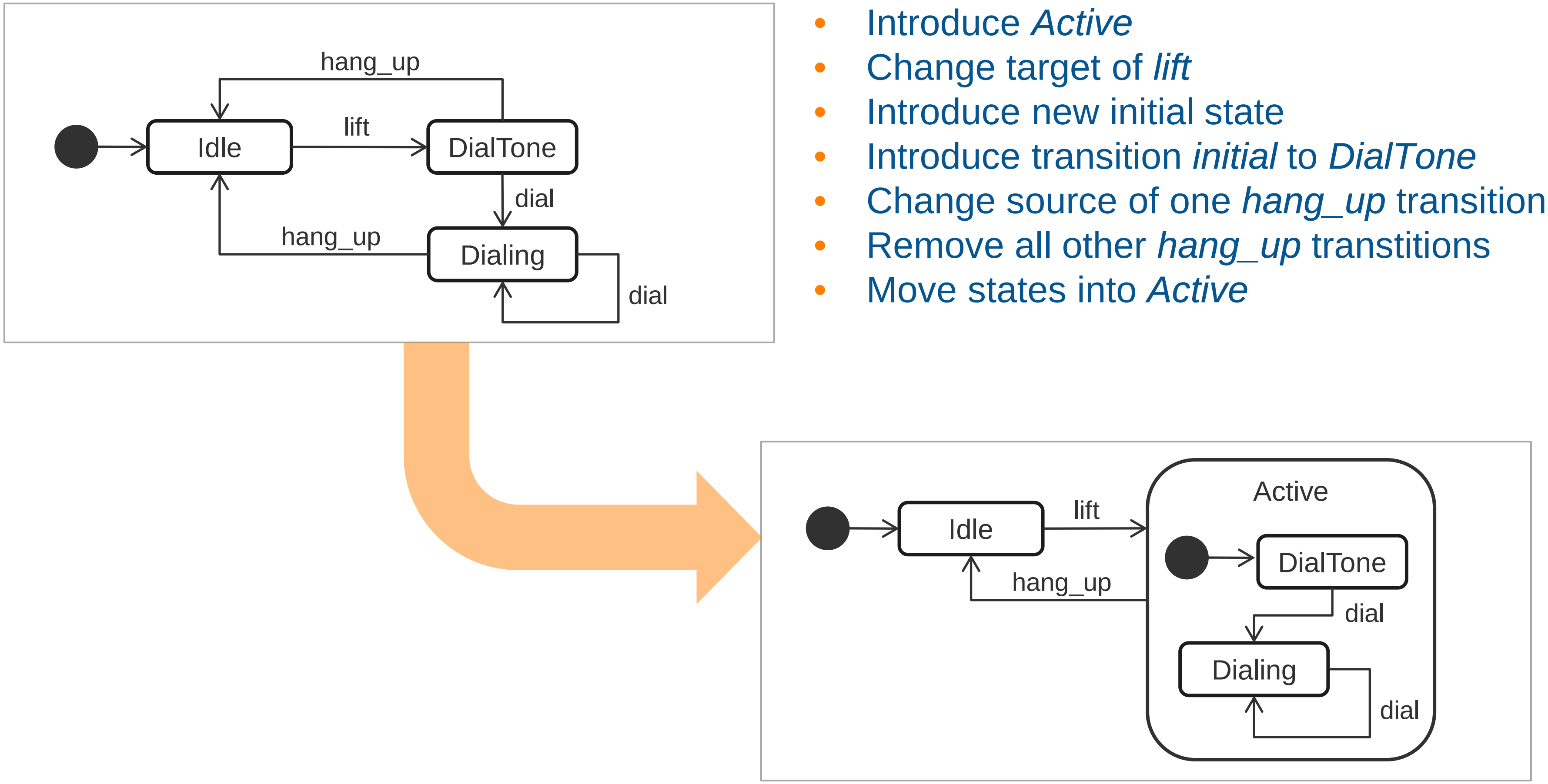


Model Transformation by Demonstration (MTBD)

- Goal of MTBD
  - Easing the specification of model transformations
  - Specification using the concrete syntax
  - → Derive the general transformation automatically from a *demonstration* of these changes applied to an example model



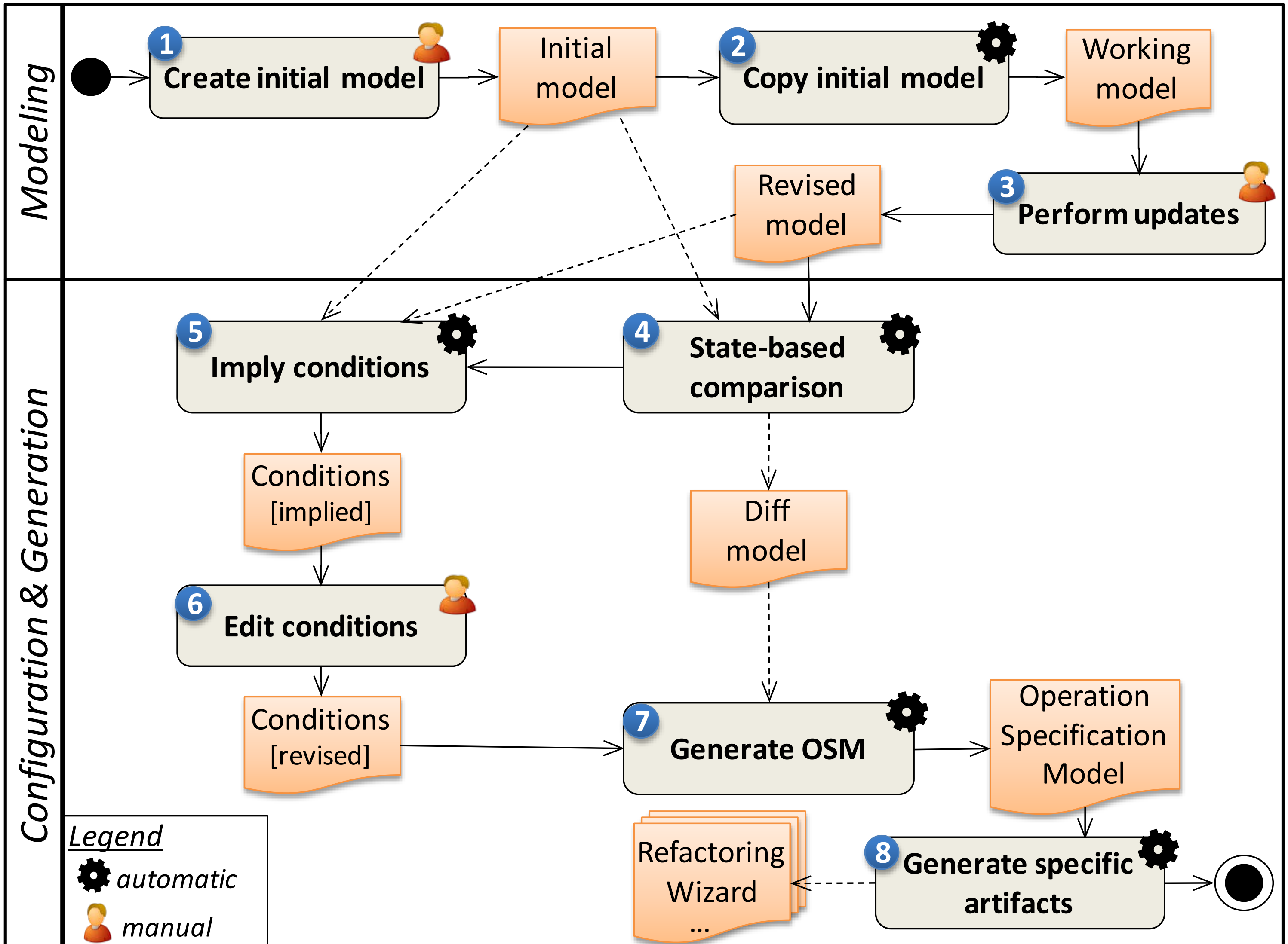
MTBD Example: Introduce Composite State



- Introduce *Active*
- Change target of *lift*
- Introduce new initial state
- Introduce transition *initial* to *DialTone*
- Change source of one *hang_up* transition
- Remove all other *hang_up* transtitions
- Move states into *Active*

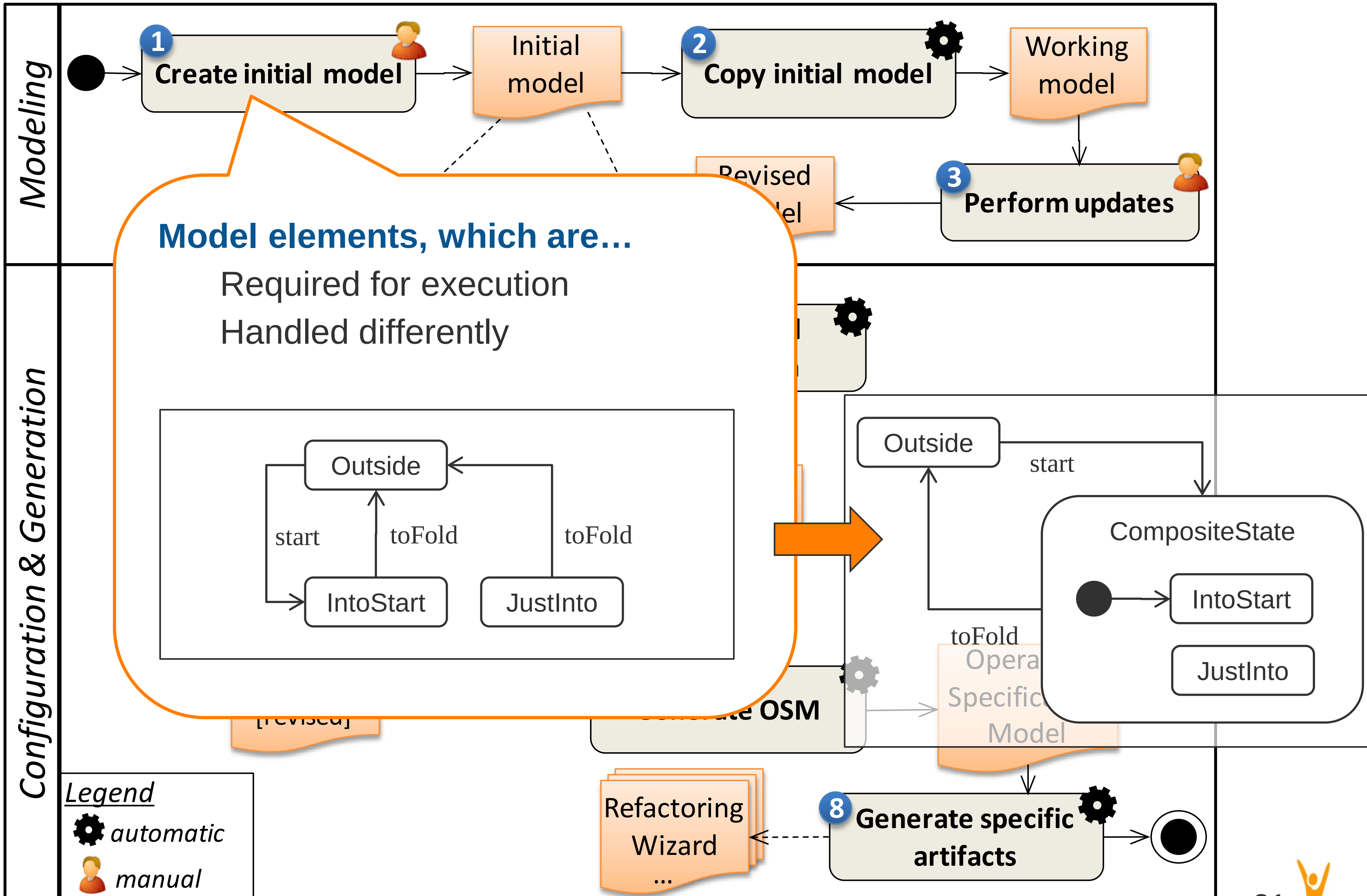
MTBD Process

Overview



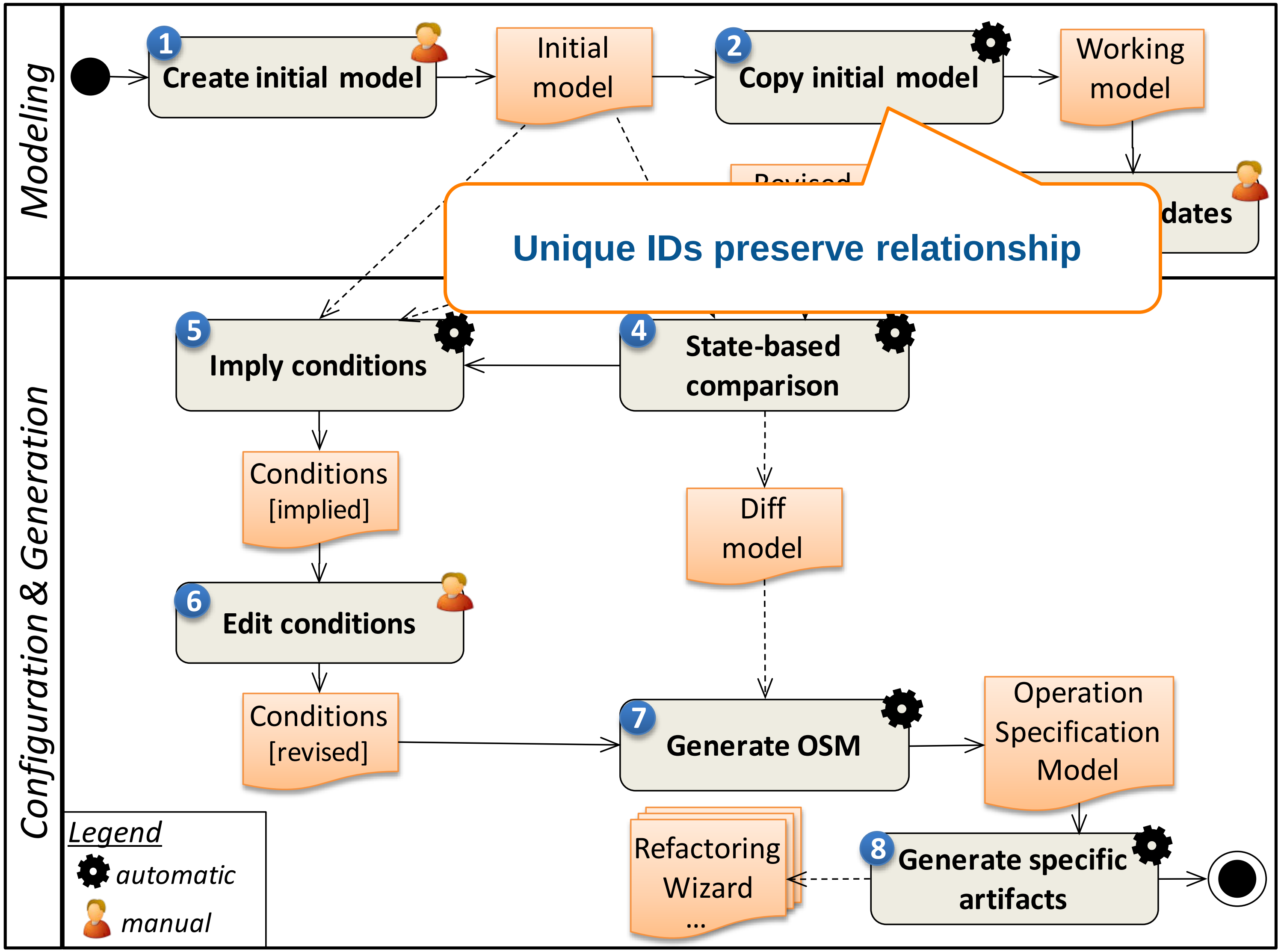
MTBD Process

Step 1: Create Initial Model



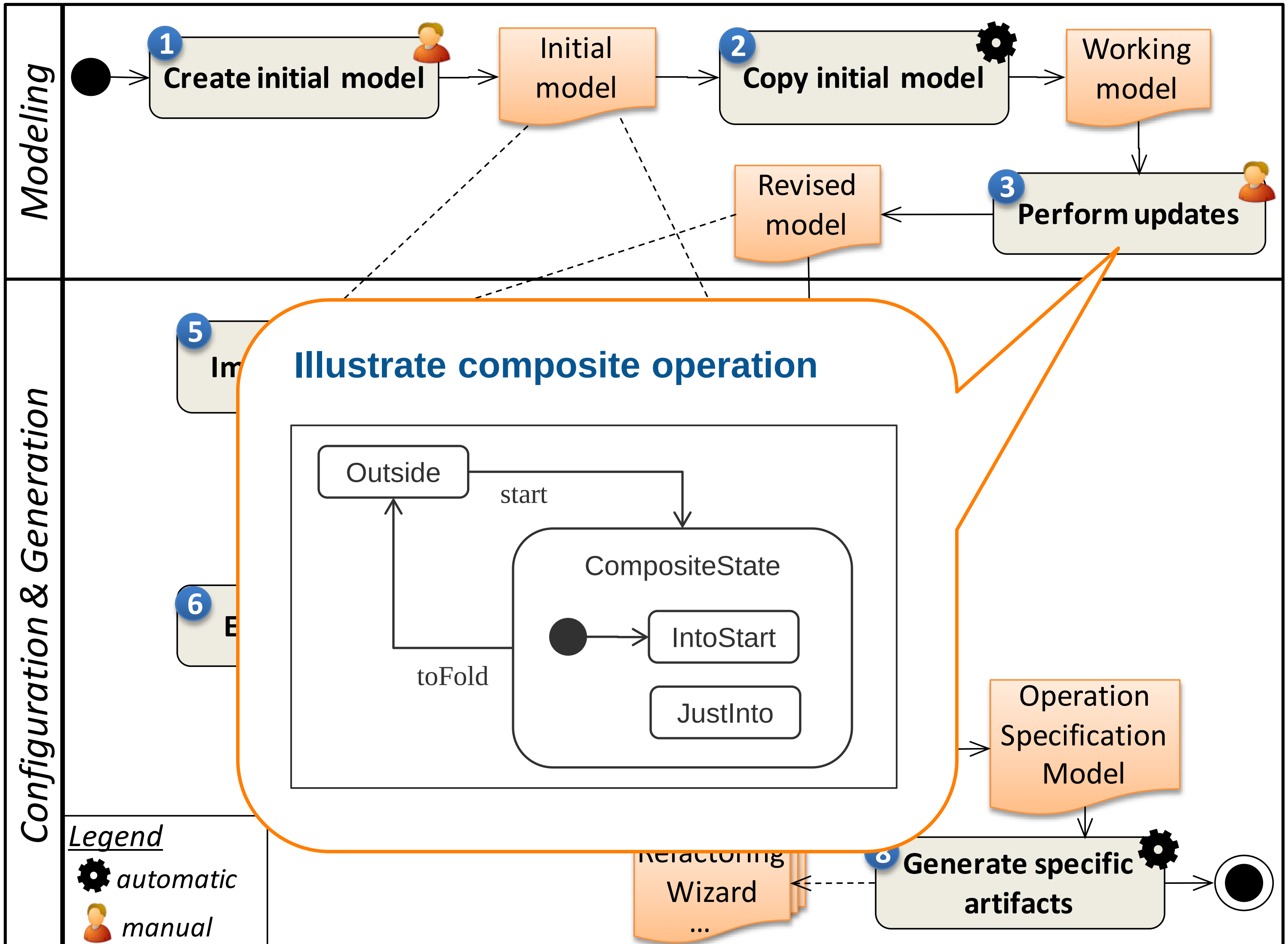
MTBD Process

Step 2: Copy Initial Model



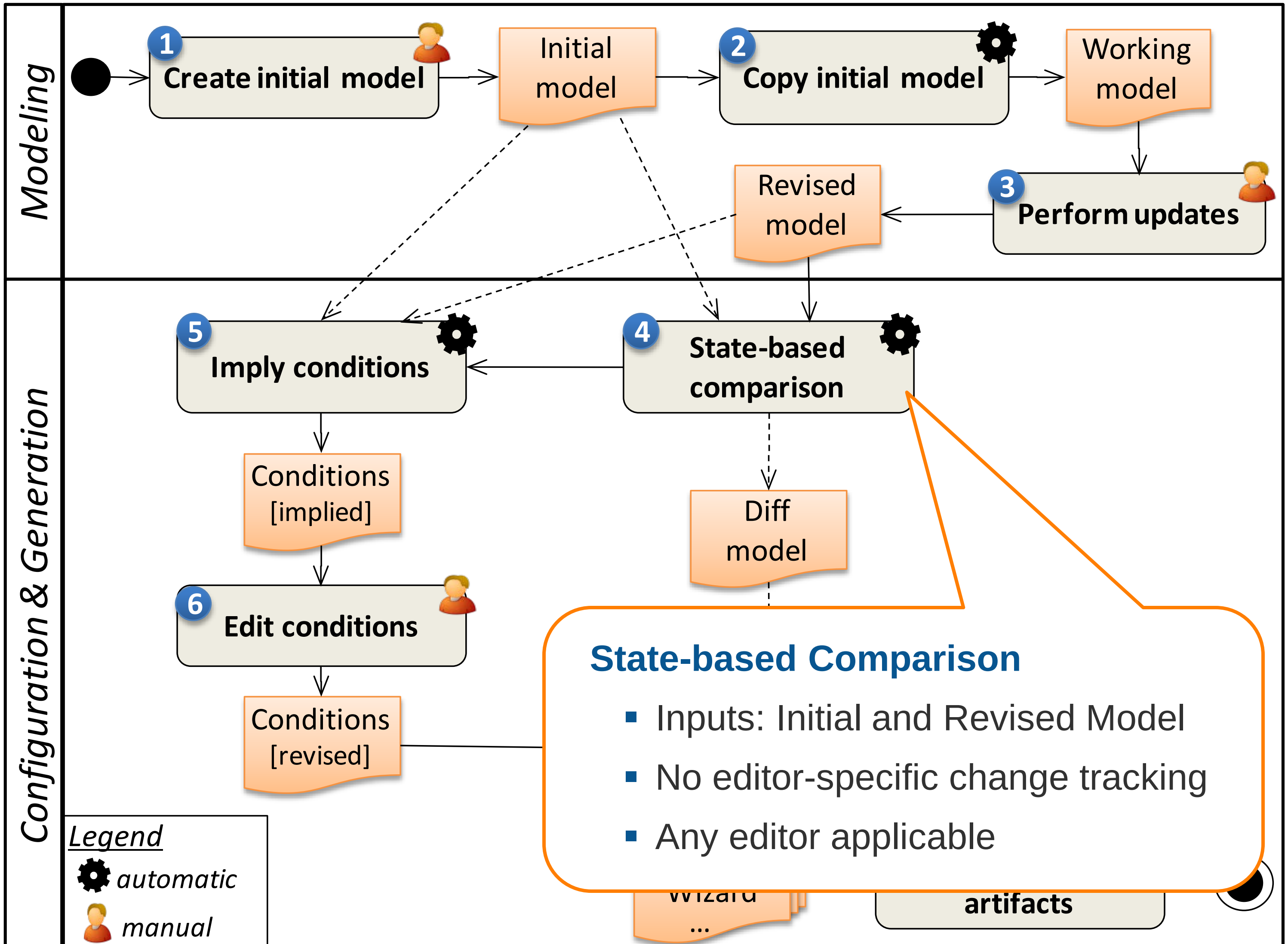
MTBD Process

Step 3: Perform updates



MTBD Process

Step 4: State-based Comparison

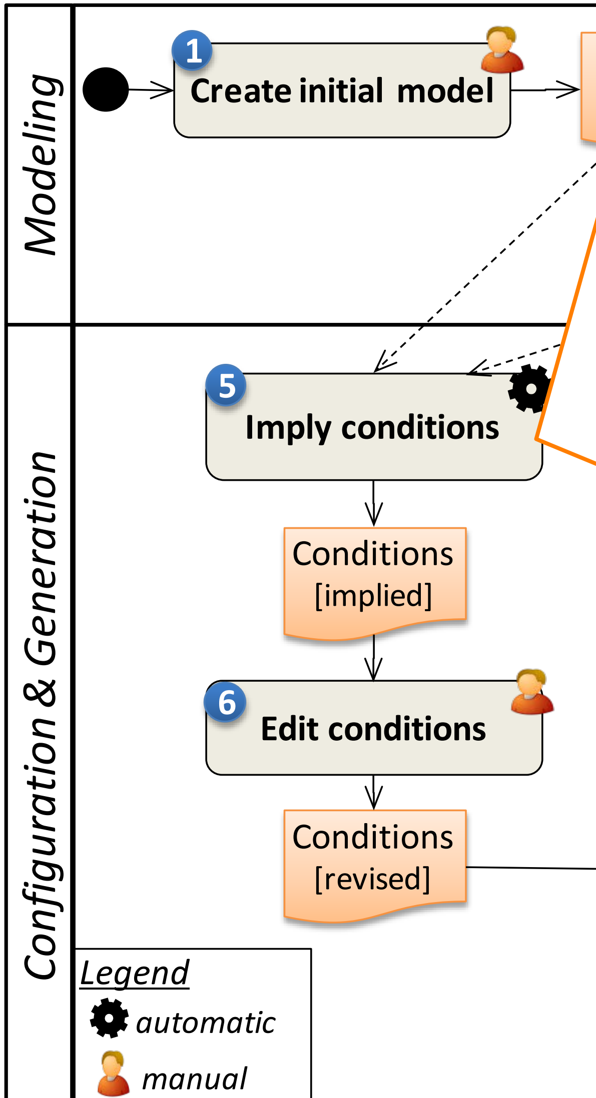


State-based Comparison

- Inputs: Initial and Revised Model
- No editor-specific change tracking
- Any editor applicable

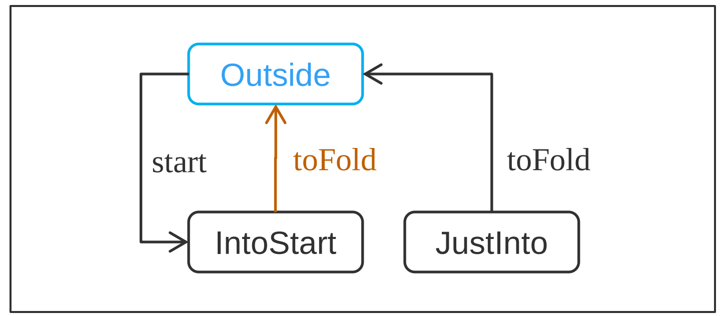
MTBD Process

Step 5: Imply Conditions



Generic Condition Generation

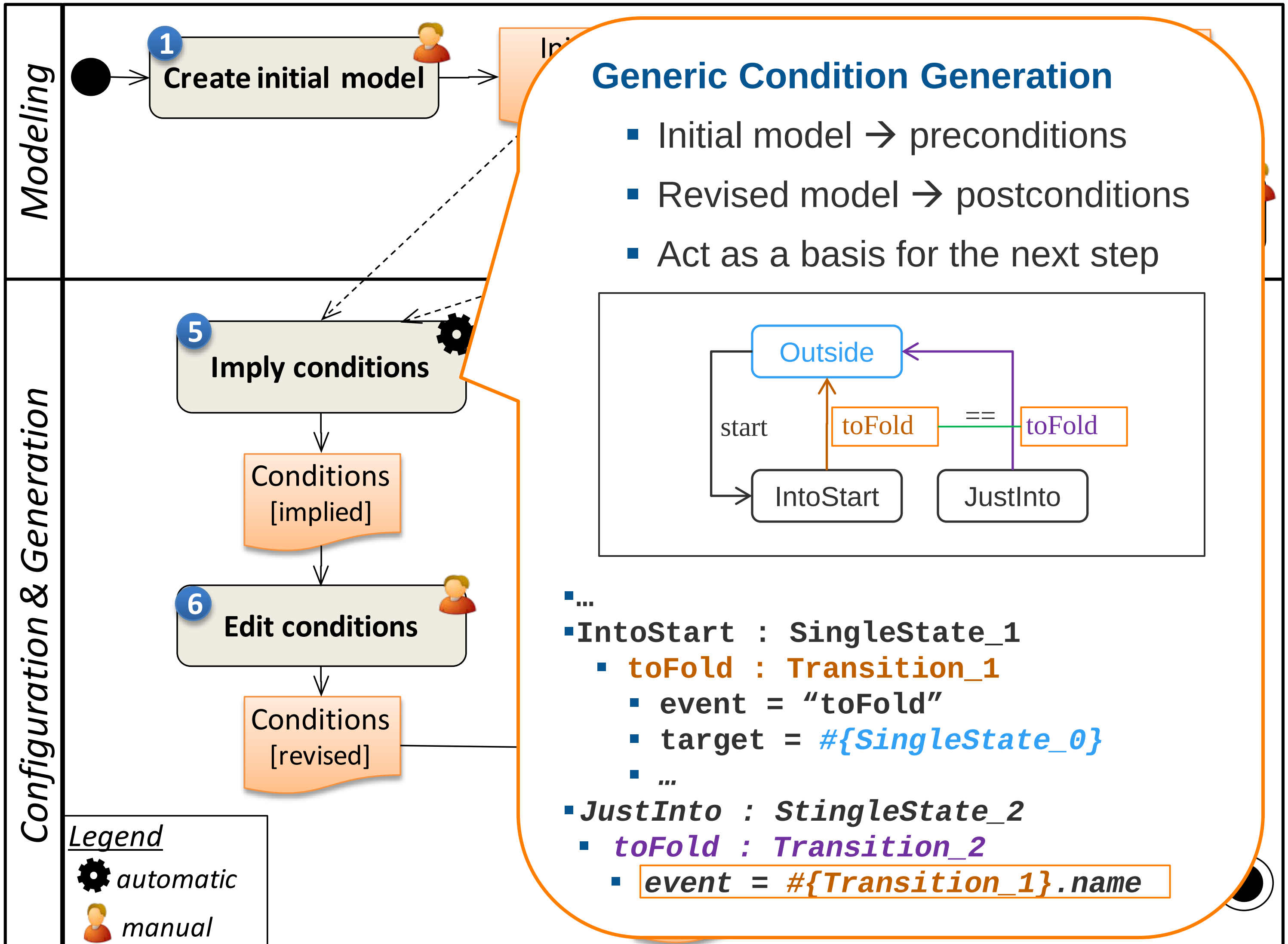
- Initial model → preconditions
- Revised model → postconditions
- Act as a basis for the next step



- `Outside : SingleState_0`
  - `name = “Outside”`
  - `incoming->includes(#{Transition_1})`
  - …
- `IntoStart : SingleState_1`
  - `toFold : Transition_1`
    - `event = “toFold”`
    - `target = #{SingleState_0}`
    - …

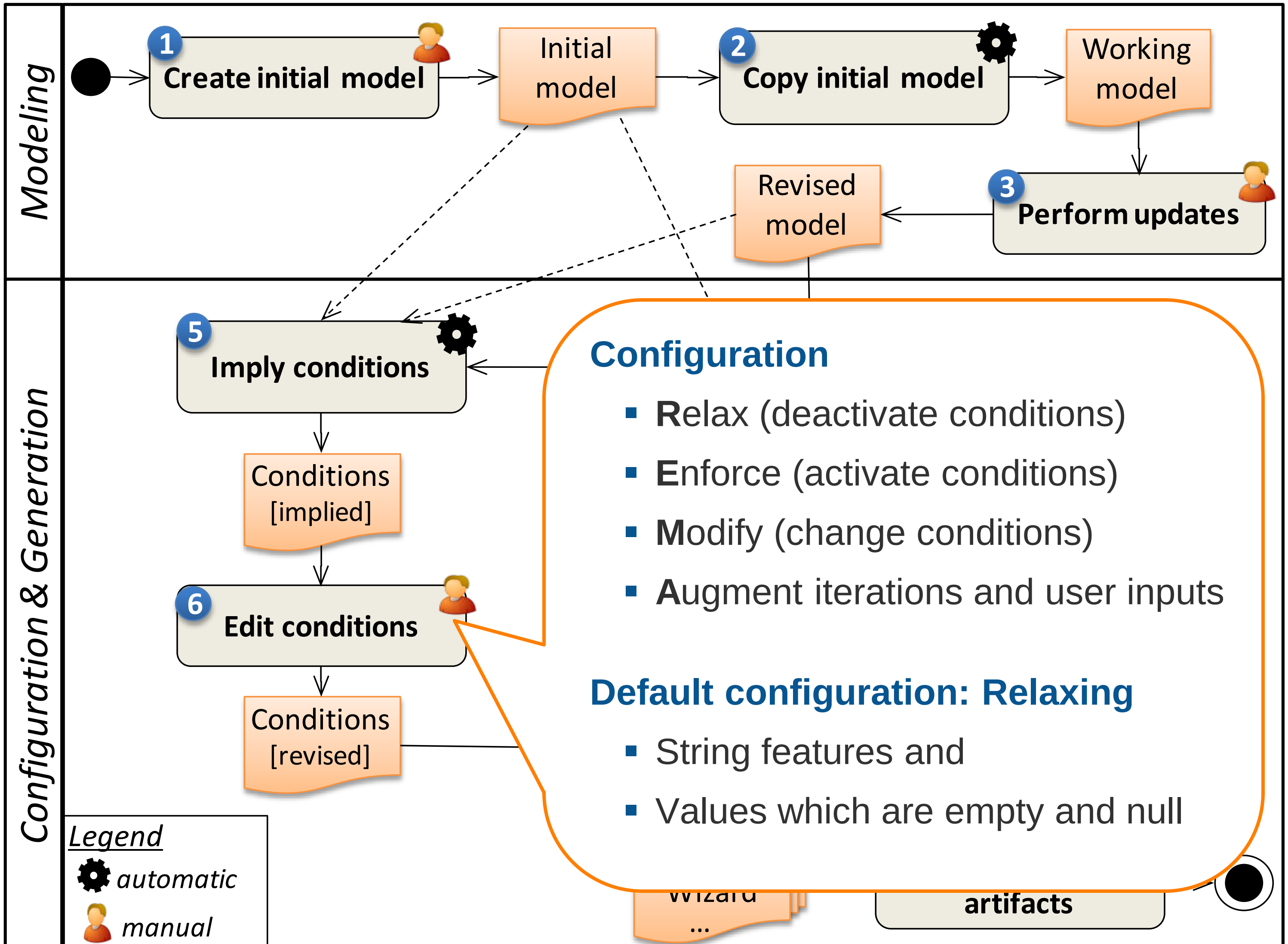
MTBD Process

Step 5: Imply Conditions



MTBD Process

Step 6: Edit Conditions



Configuration

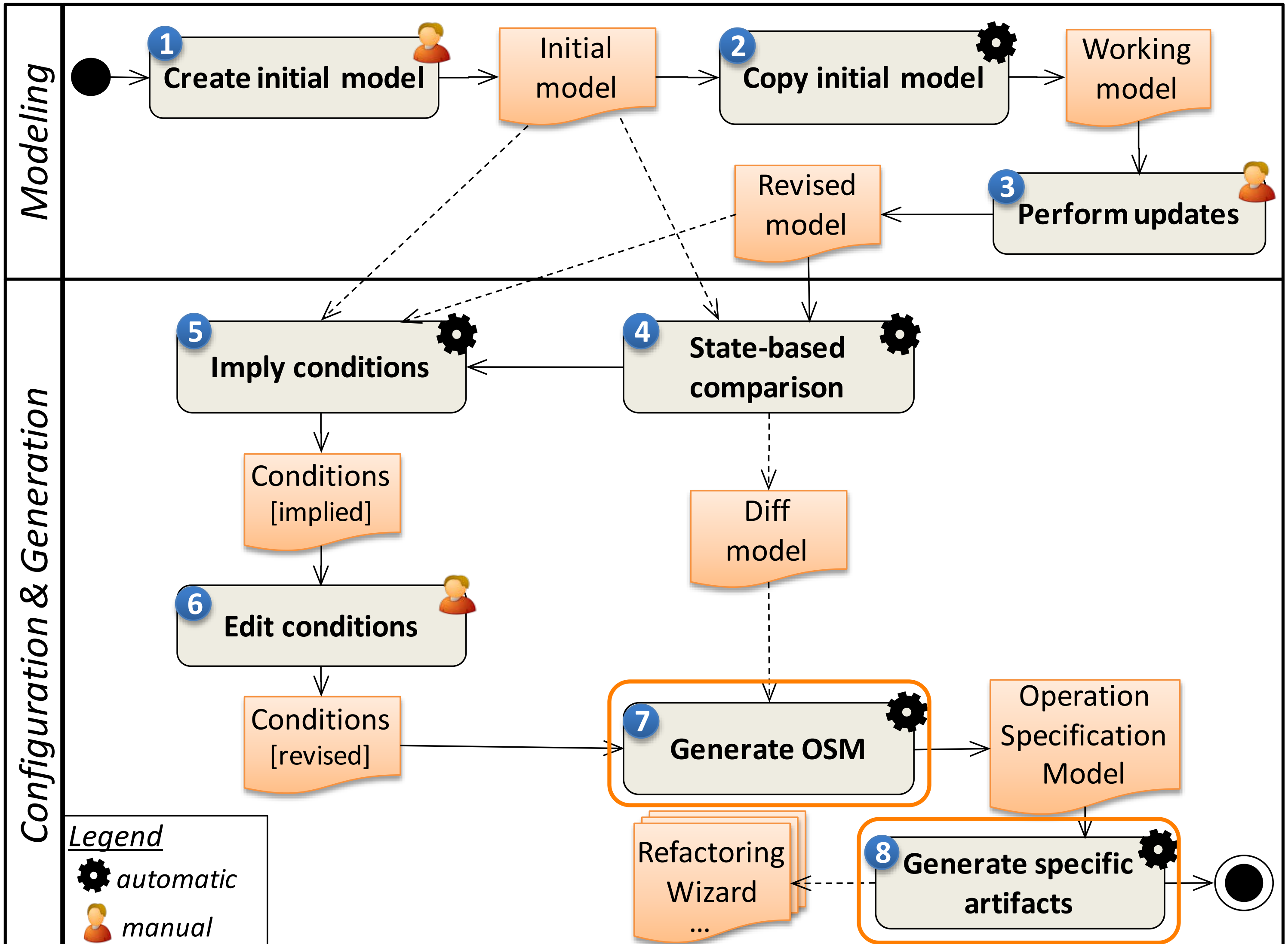
- **R**elax (deactivate conditions)
- **E**nforce (activate conditions)
- **M**odify (change conditions)
- **A**ugment iterations and user inputs

Default configuration: Relaxing

- String features and
- Values which are empty and null

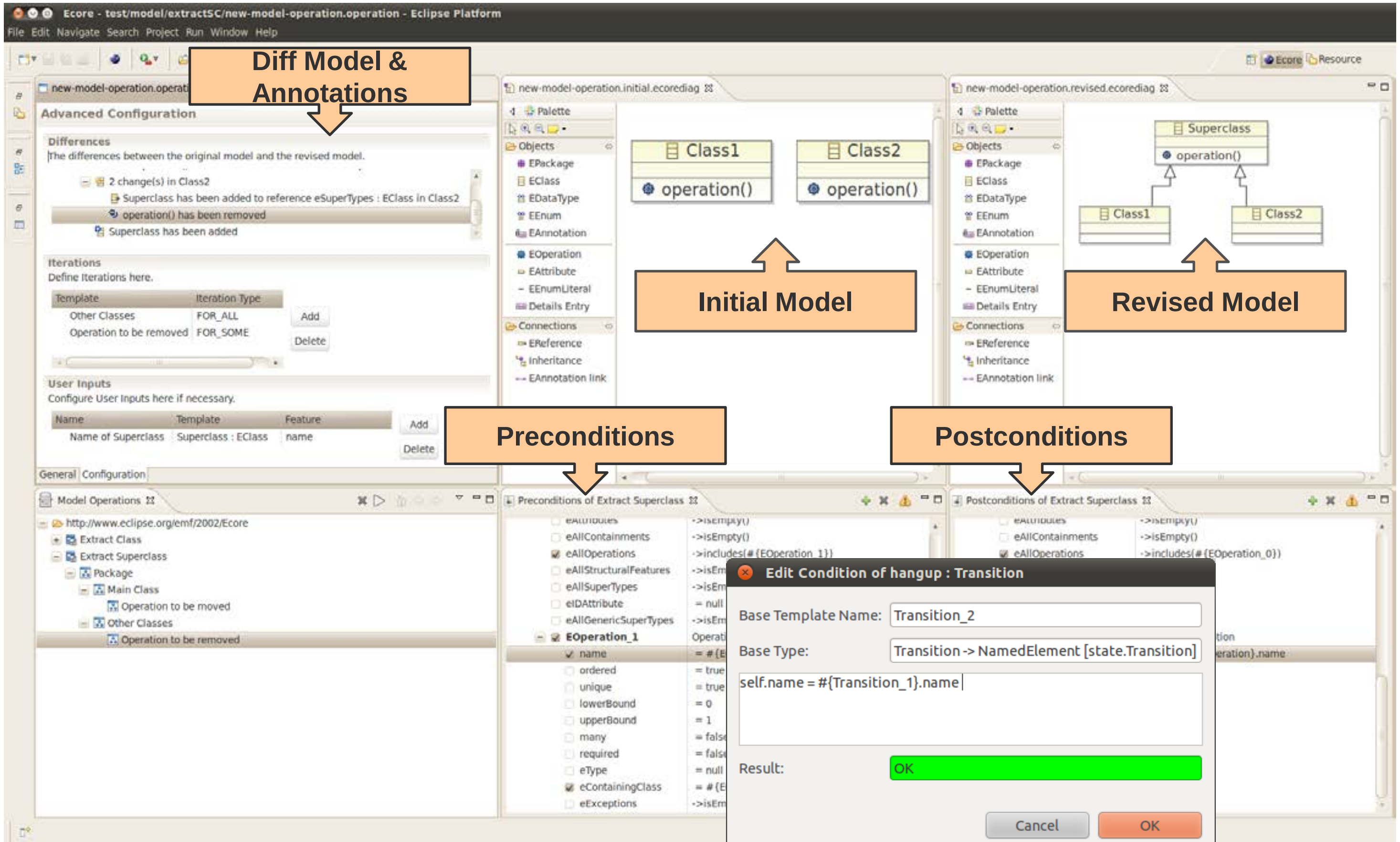
MTBD Process

Step 7 and Step 8



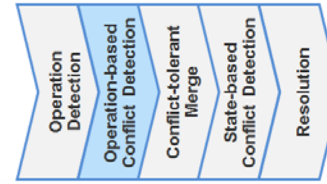
MTBD: Tooling

- Eclipse Modeling Operations (EMO)
  - http://www.modelversioning.org/index.php?option=com_content&view=article&id=68&Itemid=89



Operation-based Conflict Detection

Overview



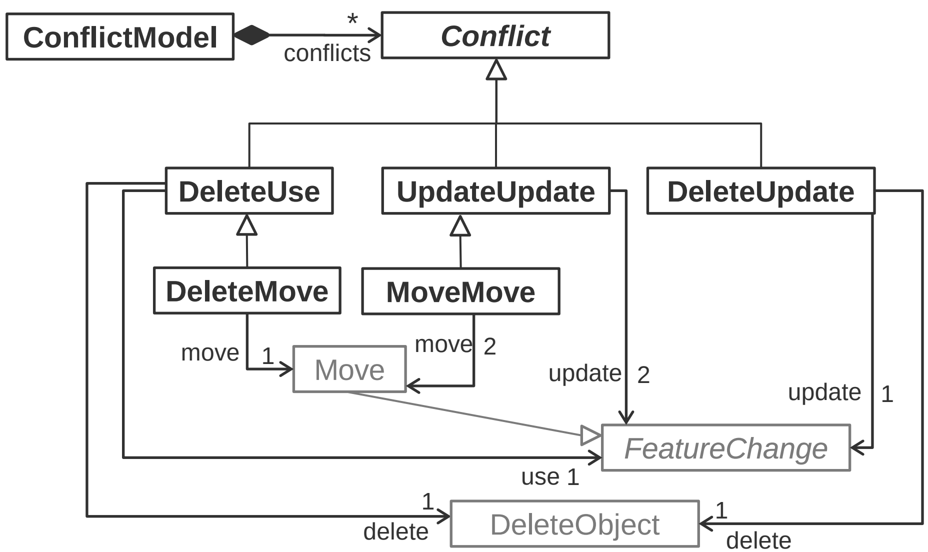
- Step 1: Atomic operation conflict detection
  - Does not regard composite operation
- Step 2: Composite operation conflict detection
  - Checks for violated preconditions of applied composite operations
- Input: Two difference models
- Output: Conflict model

$$\begin{tikzcd} D^{cop}_{V_o \bullet V_{r1}} \arrow[dr] & & D^{cop}_{V_o \bullet V_{r2}} \arrow[dl] \\ & C_{V_1 \bullet V_{r2}} & \end{tikzcd}$$

Atomic Operation-based Conflict Detection

Atomic Conflict Metamodel

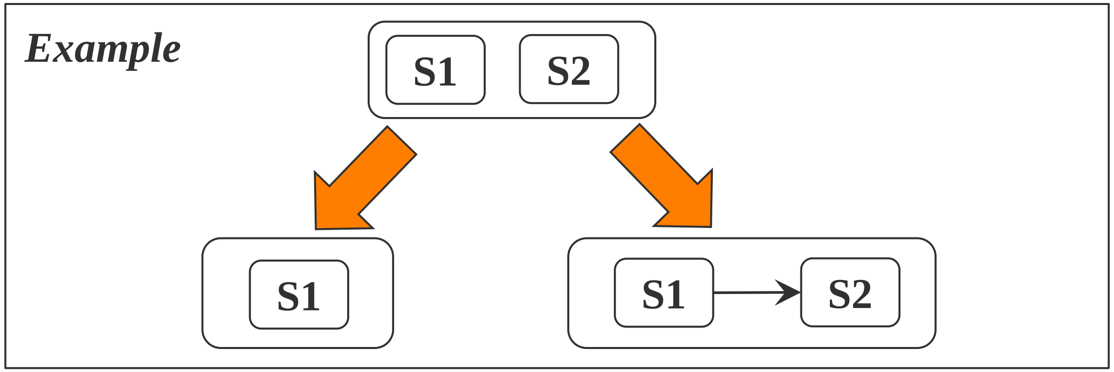
- Operation-based conflicts are described by a metamodel
  - Each conflict type is represented by a dedicated metaclass
- Conflict patterns for detecting operation-based conflicts
  - Each concrete conflict type has conflict patterns attached

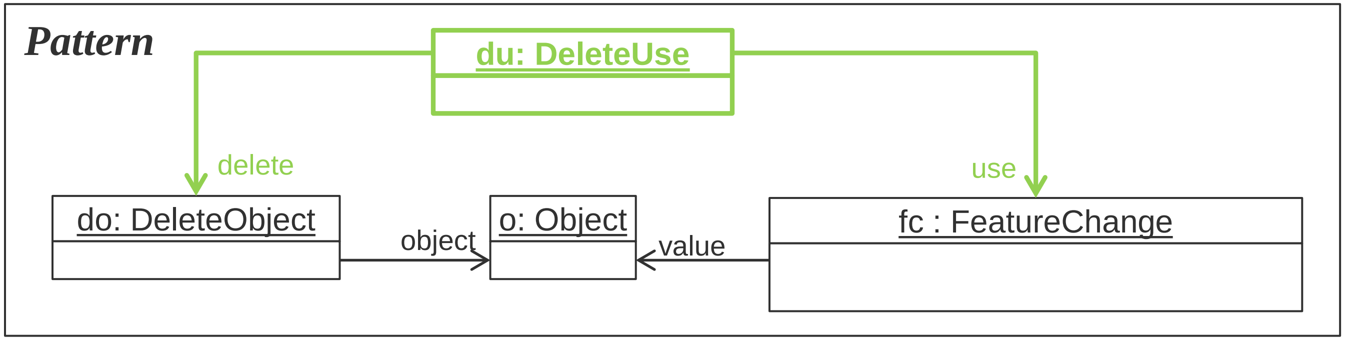


Atomic Operation-based Conflict Detection

Conflict Pattern 1: Delete/Use Conflict

- Delete/Use conflict occurs if
  - Modeler A deletes object o
  - Modeler B uses object o in a featureChange operation as value

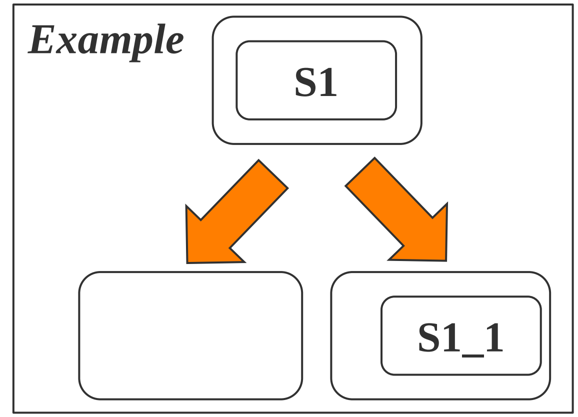


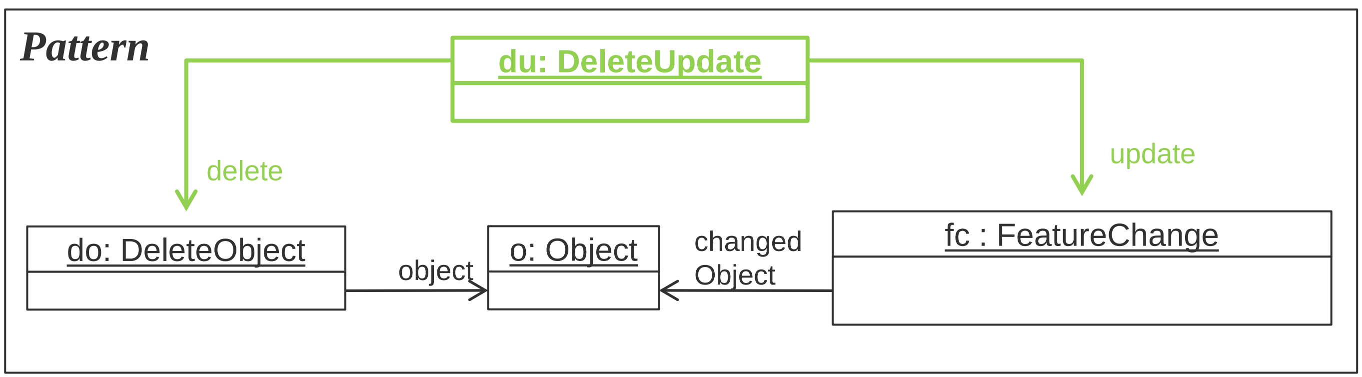


Atomic Operation-based Conflict Detection

Conflict Pattern 2: Delete/Update Conflict

- Delete/Update conflict occurs if
  - Modeler A deletes object o
  - Modeler B updates a feature f of object o

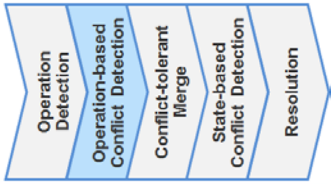




There are several more patterns!

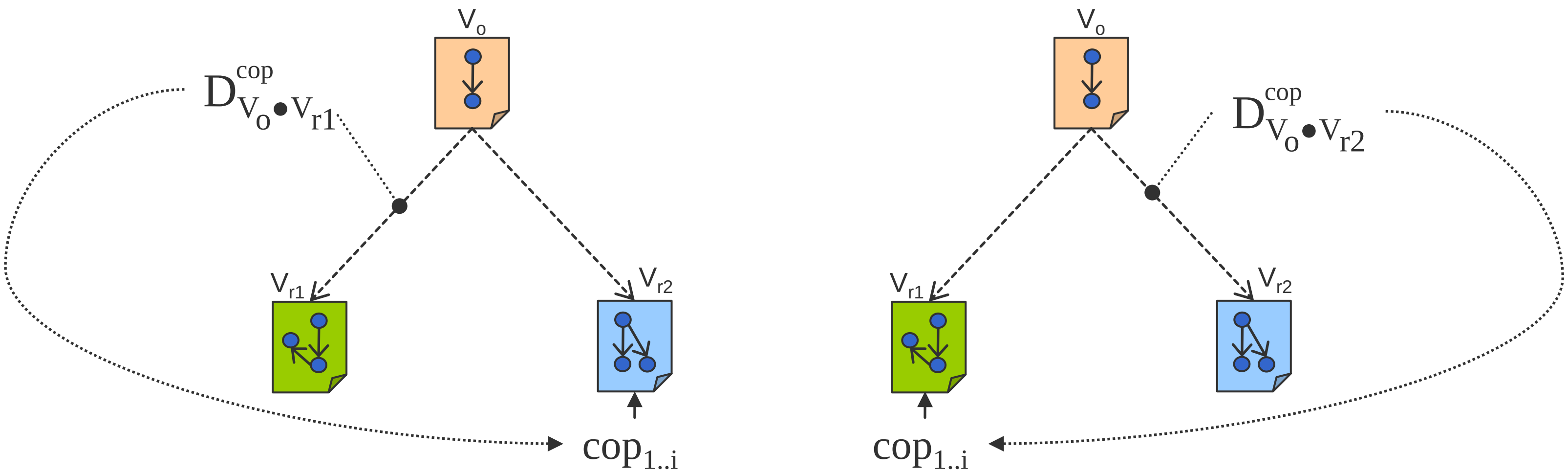
G. Taentzer, C. Ermel, P. Langer, M. Wimmer: *Conflict Detection for Model Versioning Based on Graph Modifications*; in: “Software and Systems Modeling (SoSym)", Springer, to appear in 2012.

P. Langer: *Adaptable Model Versioning based on Model Transformation By Demonstration*; Reviewer: G. Kappel, J. Gray; E188 Institut für Softwaretechnik und Interaktive Systeme, 2011.



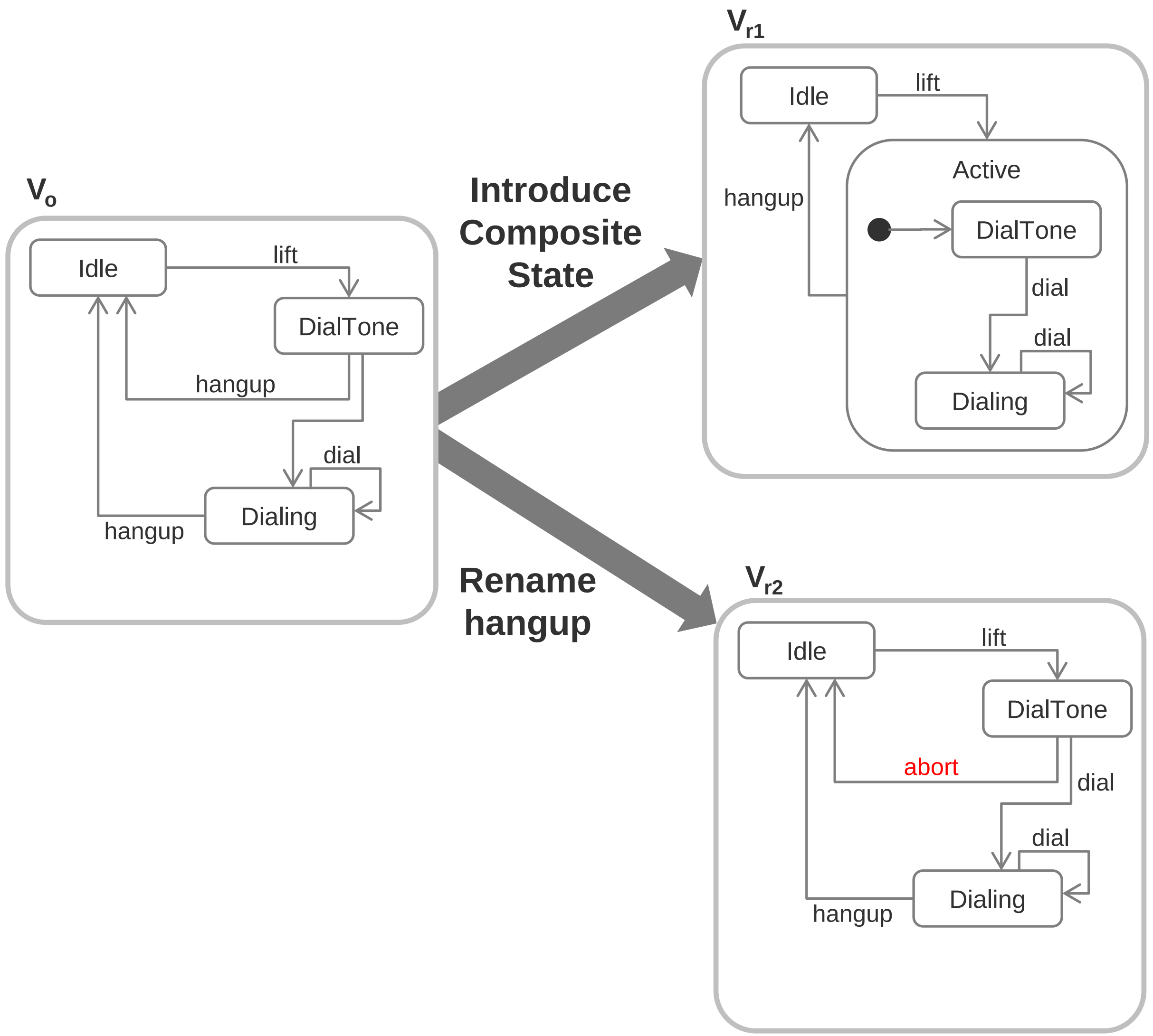
Composite Operation Conflict Detection

- For each composite operation application,
  - check whether it is ***applicable*** at the ***opposite side***



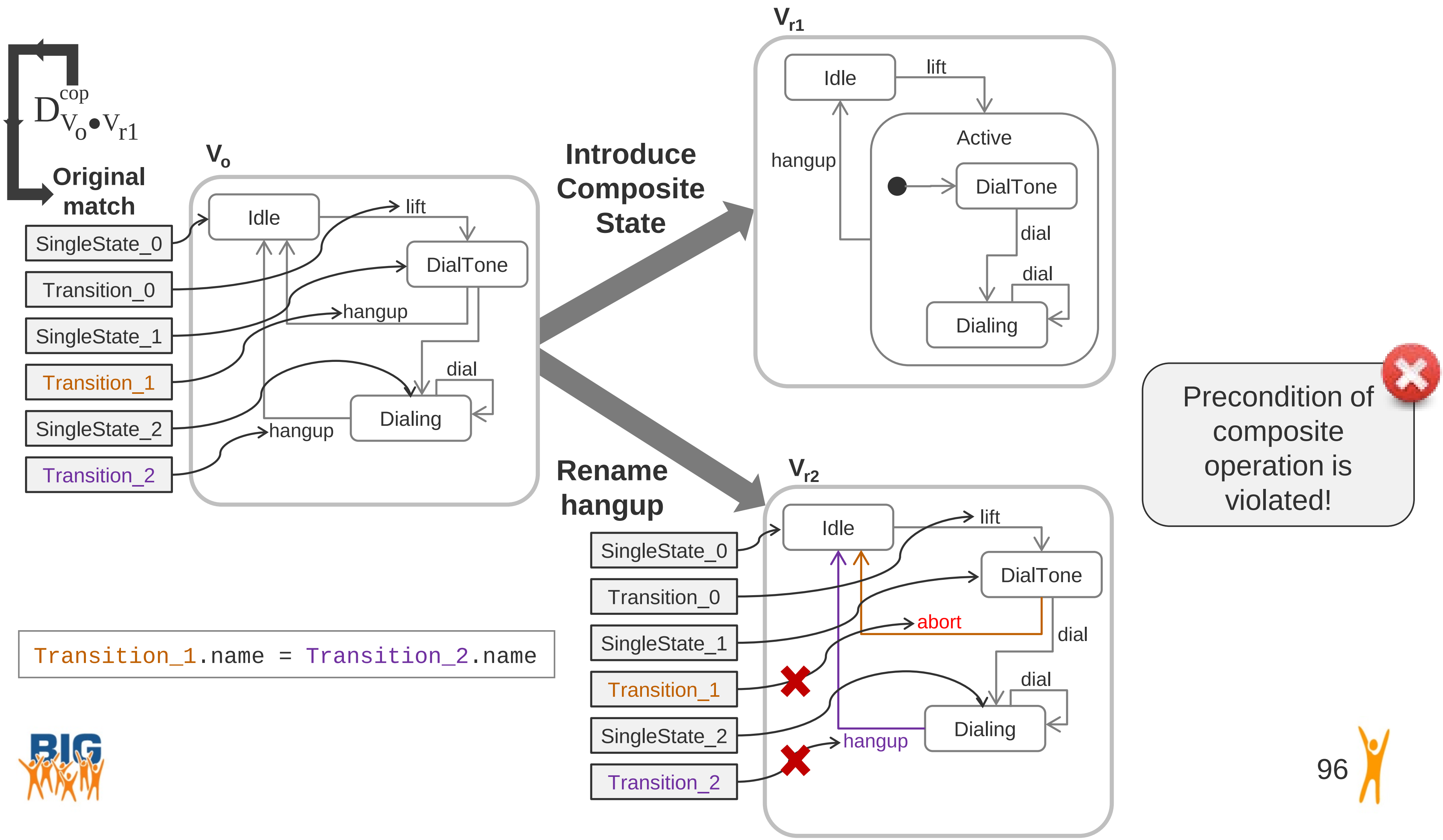
Composite Operation Conflict Detection

- Example



Composite Operation Conflict Detection

- Example



`Transition_1.name = Transition_2.name`

Conflict-tolerant Merge

How to support developers when merging different versions?

- Motivation
  - Only one modeler is responsible for conflict resolution
    → Final version does not reflect all intentions of the modelers
  - Conflicts are considered harmful
    → They should be subject of discussion
  - Merging conflicts is rejecting one of the conflicting operations
    → However, a merged model is helpful for deciding how to resolve conflicts
- Goal
  - Conflict-tolerant model versioning system supporting a *collaborative conflict resolution process*
- Challenge
  - Merge is not possible in case of conflicts!
  - How can a merge be accomplished in case of conflicts?
  - How can the conflicts be visualized for any modeling language?

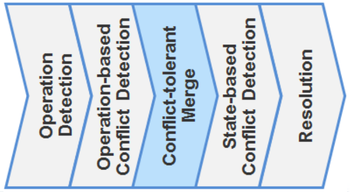
Conflict-tolerant Merge

Solution

- General Approach
  1. Conflict-tolerant merge rules
     - To avoid conflict resolution during check-in
     - To find a merge result irrespectively of conflicts
  2. Conflict annotations
     - To avoid information loss
     - To enable distributing the responsibility of conflict resolution
     - To provide a basis for discussion
  3. Conflict resolution model
     - To control the lifecycle of a conflict
     - To manage conflicts and resolutions

Conflict-tolerant Merge Rules

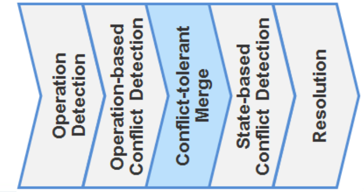
Prioritize one change or omit both changes



- To enable a merge irrespectively of conflicts

- Omit both conflicting changes in case of …
  - Update-update: Use original value
  - Move-move: Use original container

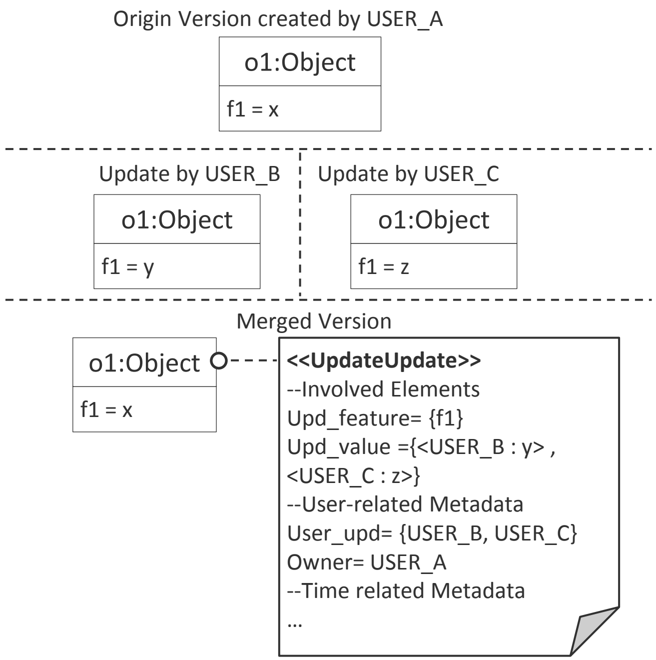
- Omit deletions out of two conflicting changes
  - Delete-update: Prioritize Update
  - Delete-use: Prioritize Use
  - Delete-move: Prioritize Move

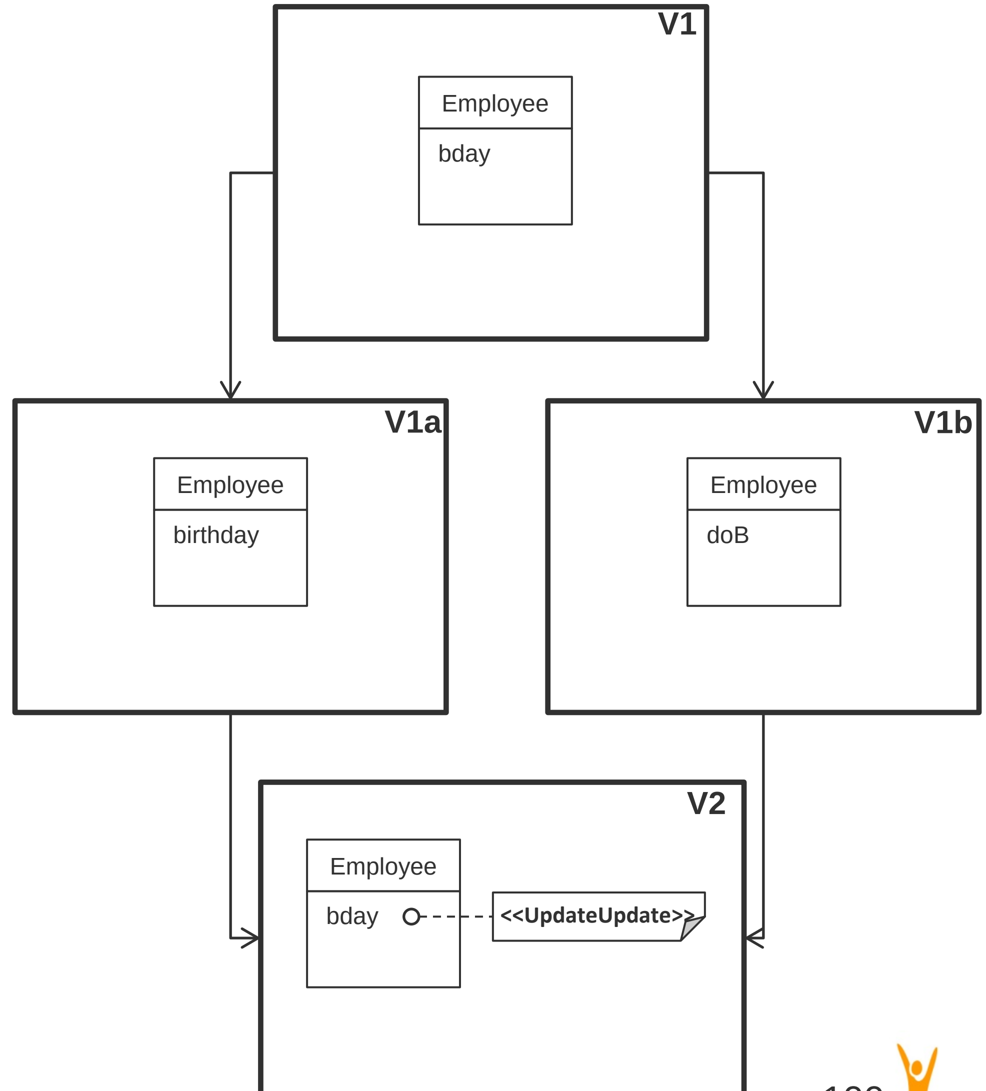


Conflict-tolerant Merge Rules

Example: Update-update Conflict

- Update-update Conflict

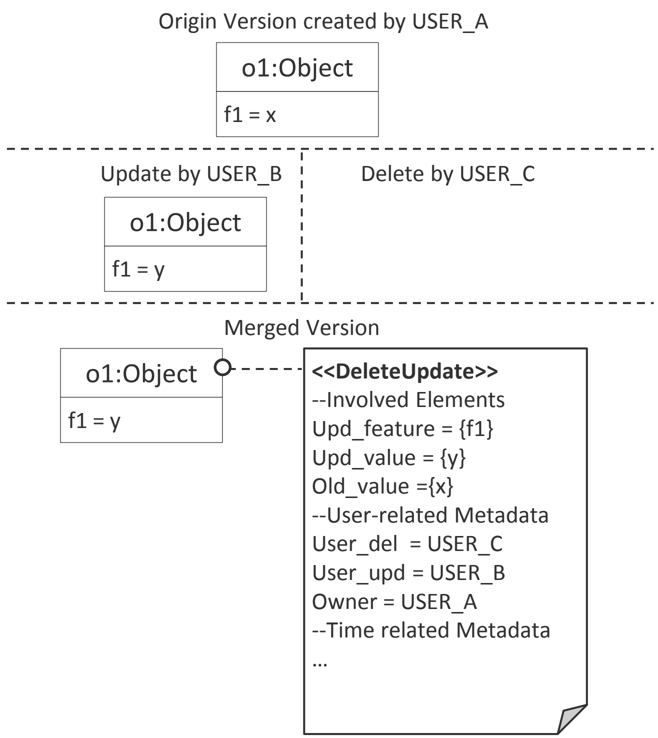


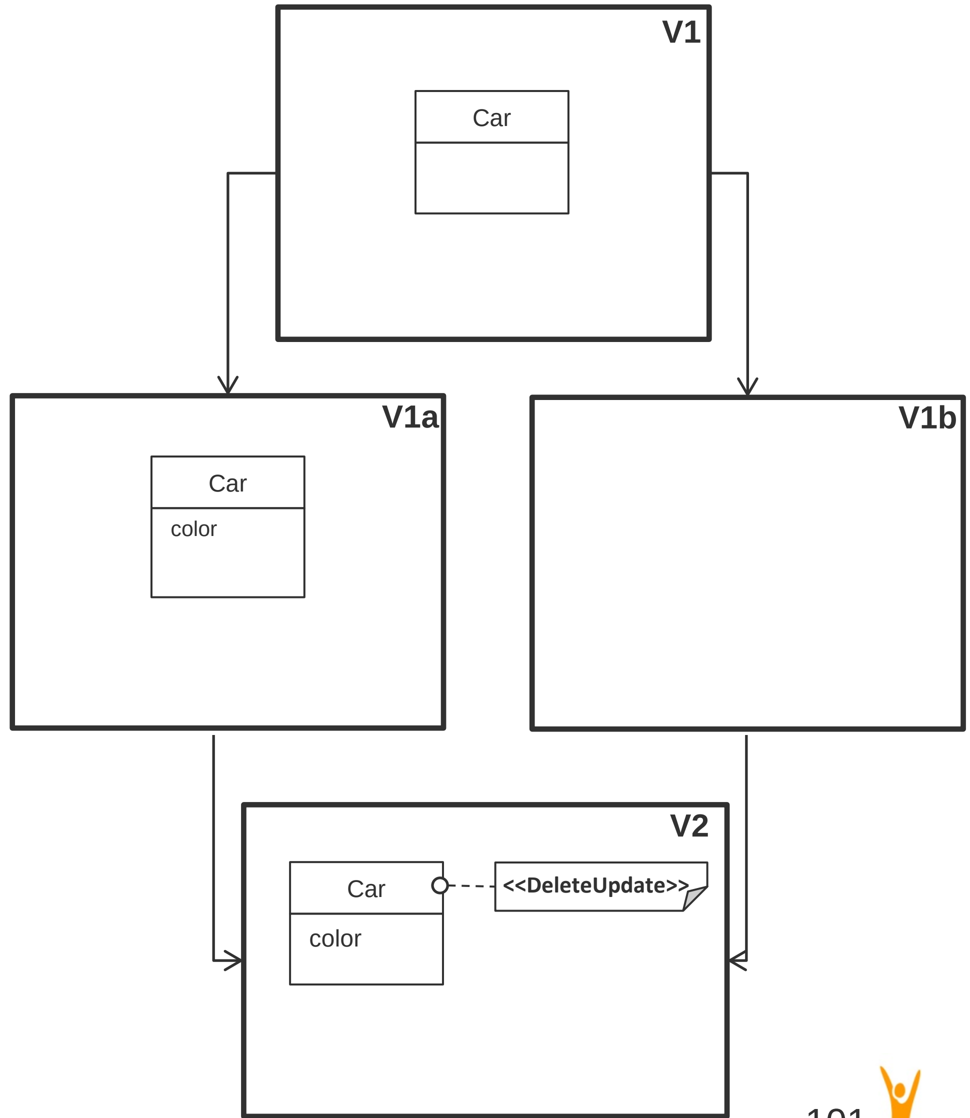


Conflict-tolerant Merge Rules

Example: Delete-update Conflict

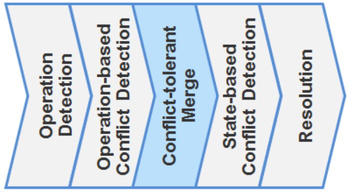
- Delete-update Conflict





Conflict Annotations

Using Annotations in the Merged Model for Marking Conflicts



- Annotate conflicts in the merged model
  - For visualizing conflicts
  - For providing a basis for resolution

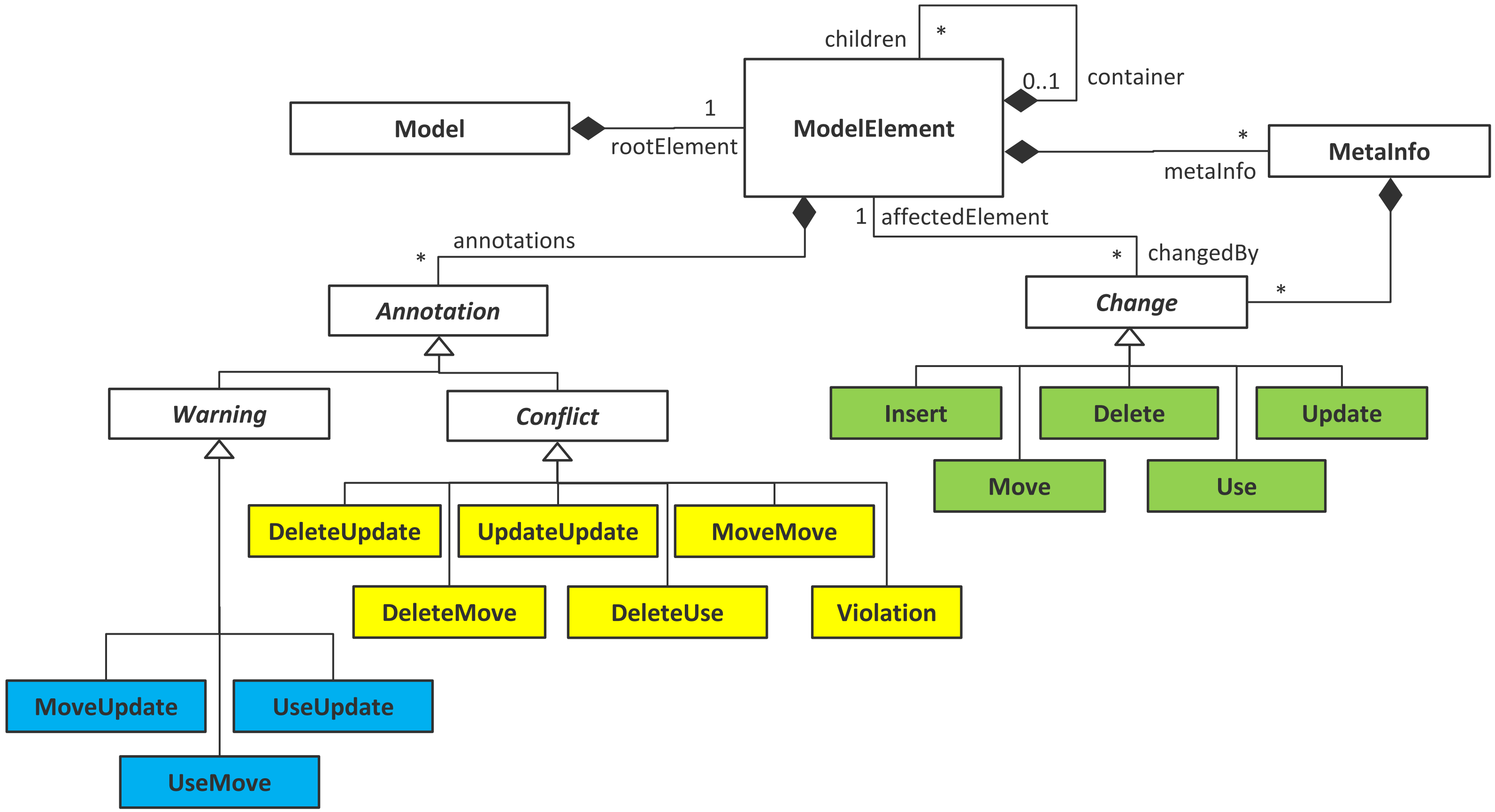
  → On top of the concrete syntax of the merged model

- Annotations can be stored across revisions
  - Allows to tolerate conflicts for a while
  - Enables distributing them among team members

- Lightweight annotations using profiles (cf. UML Profiles)
  - Model elements are annotated by stereotypes in concrete syntax
  - Tagged values comprise additional conflict information

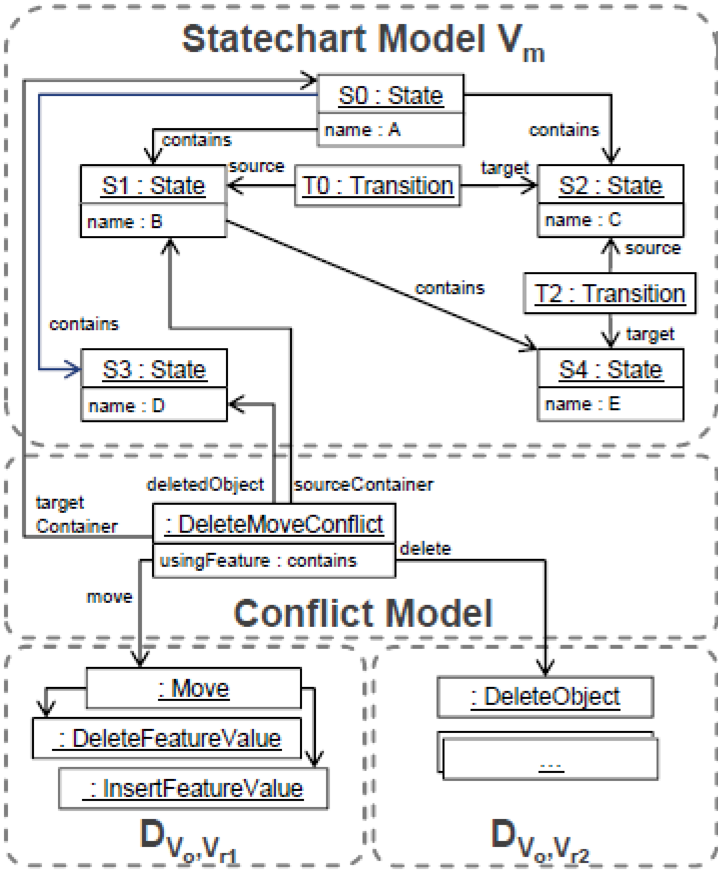
Conflict Annotations

Profile for Merge Conflicts



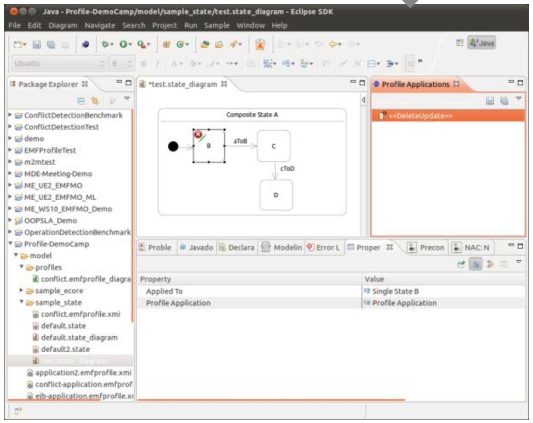
Conflict Annotations

Example



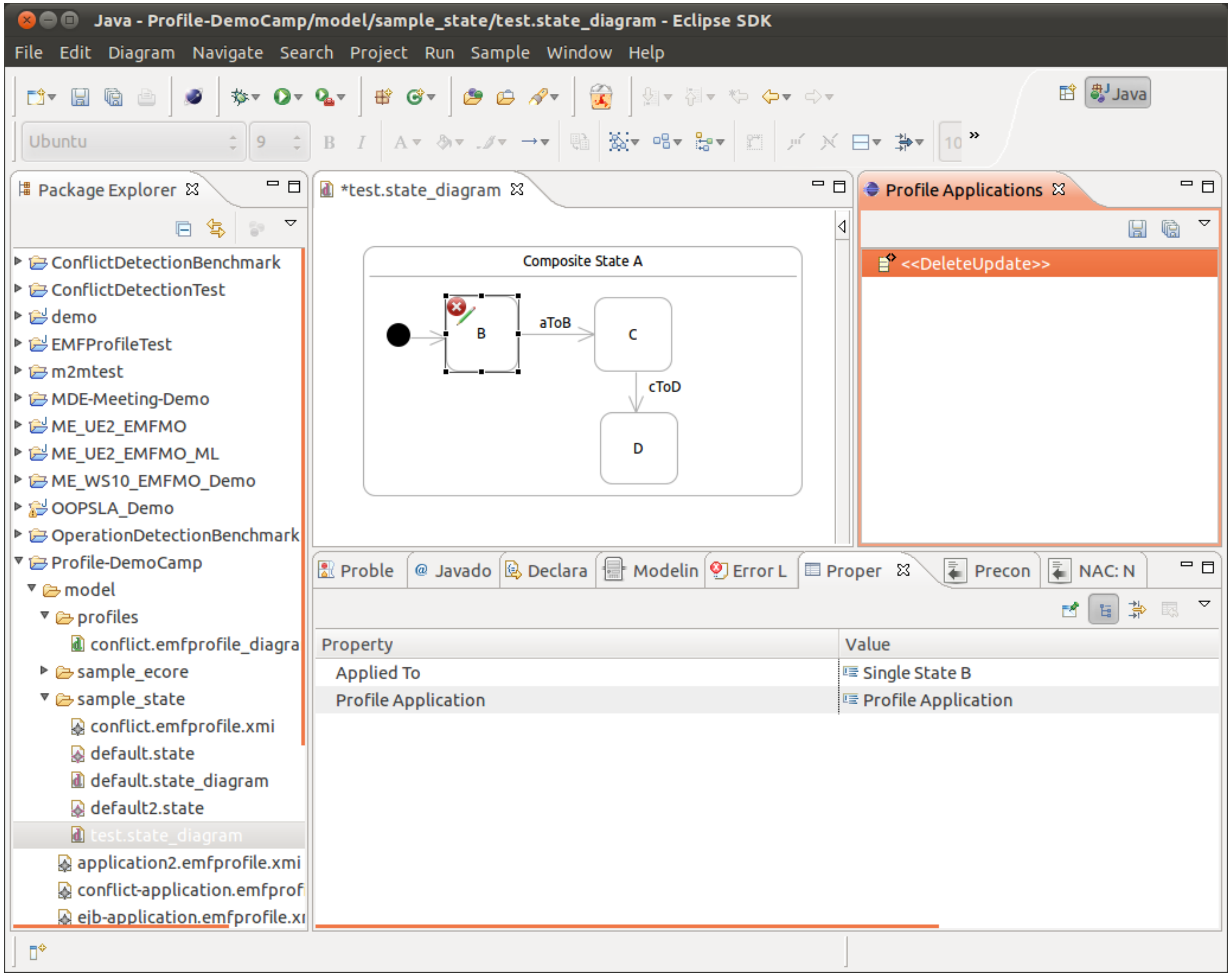
Apply conflict-tolerant merge rules

Annotate conflicts (i.e., apply conflict profile)



Conflict Annotations

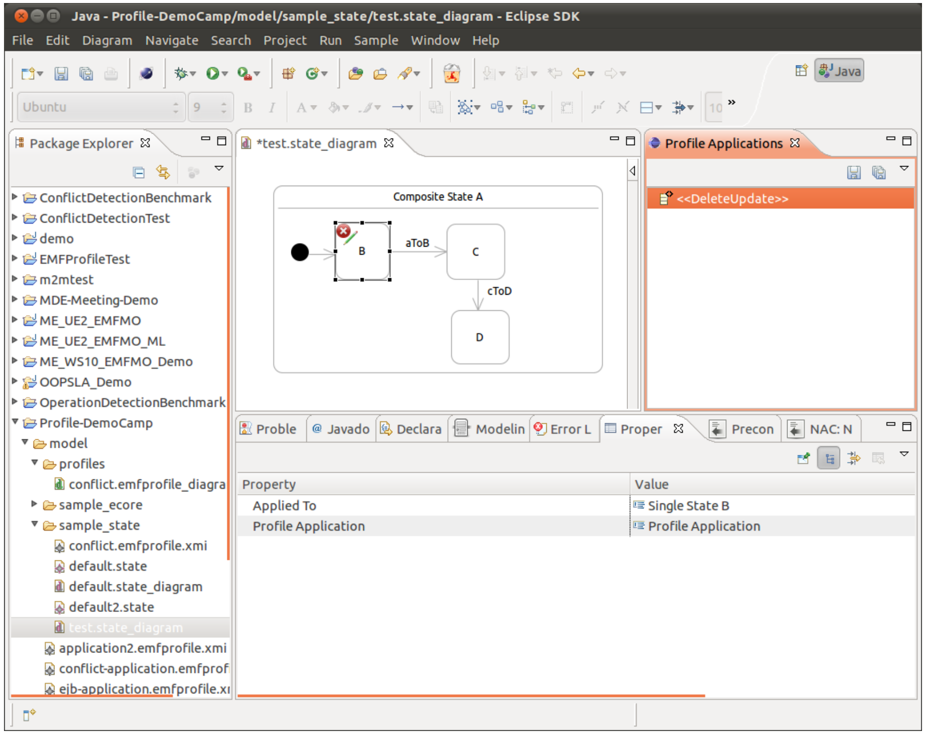
Applied Conflict Profile



Conflict Annotations

Applied Conflict Profile

- **How can profiles be applied to non-UML models?**
  - **→EMF Profiles**



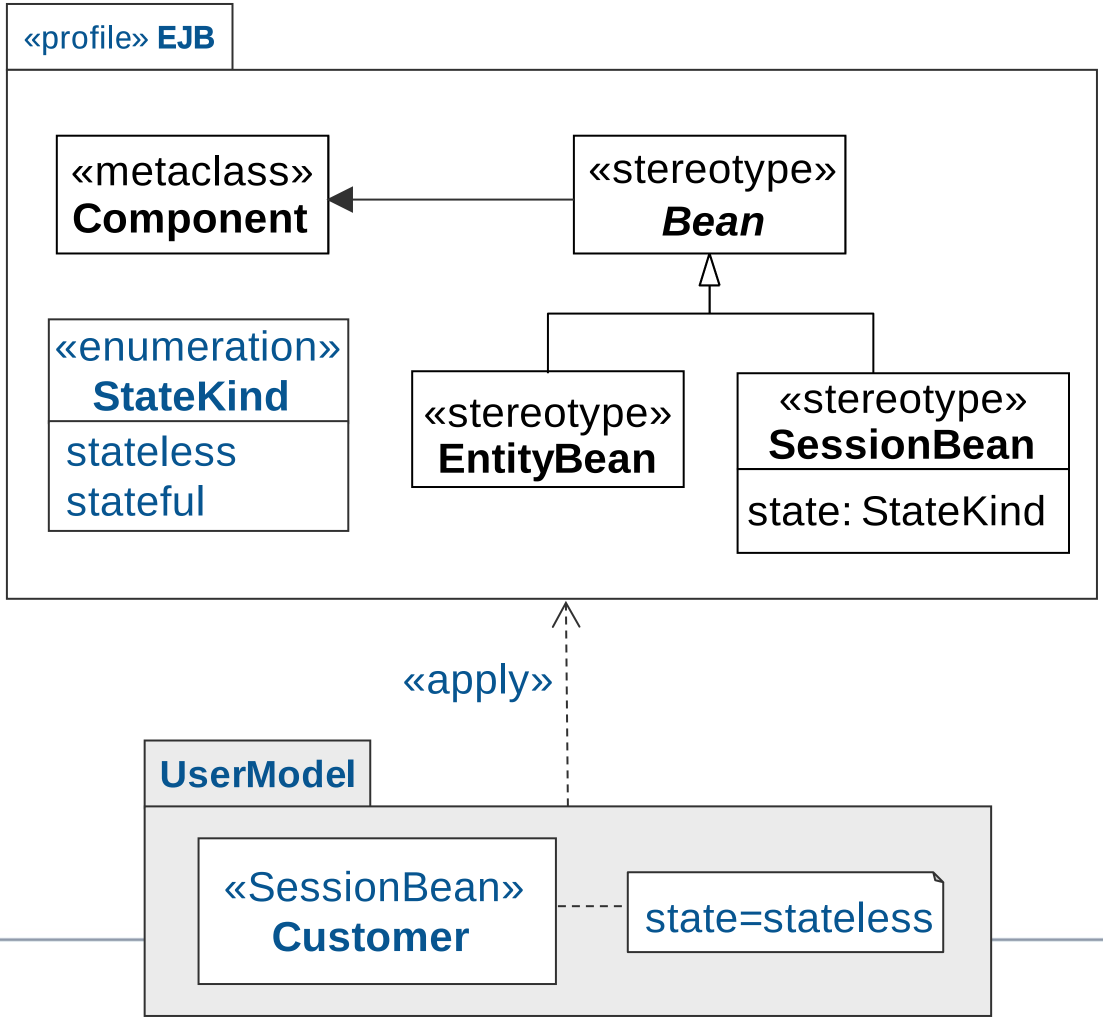
Overall Goal of EMF Profiles

"Adopt the notion of *UML Profiles* to *DSMLs* residing in *EMF*"

- UML Profiles: A Short Reminder
  - Lightweight Language Extension Mechanism of the UML
  - A Profile consists of Stereotypes
  - Stereotypes extend base classes
    - And may introduce "tagged values"
  - Profiles are applied to UML models
    - By applying stereotypes to instances of their base classes
    - Specifying concrete values for their defined "tagged values"

UML Profiles: A Short Introduction

Profile specification



Profile application

Motivation

- Overall goal
  - "Adopt the notion of *UML Profiles* to *DSMLs* residing in *EMF*"

- Is combination of profiles with DSMLs a contradiction?
  - Many debates on Pros and Cons of adopting UML Profiles or DSMLs

UML Profiles vs DSMLs

UML Profiles vs DSMLs

Reuse UML's language concepts and existing UML editors!

Create a lean language that is straight to the point!

UML Profiles vs DSMLs

You'll have to create the whole infrastructure by yourself!

You'll end up with an overloaded and imprecise language!

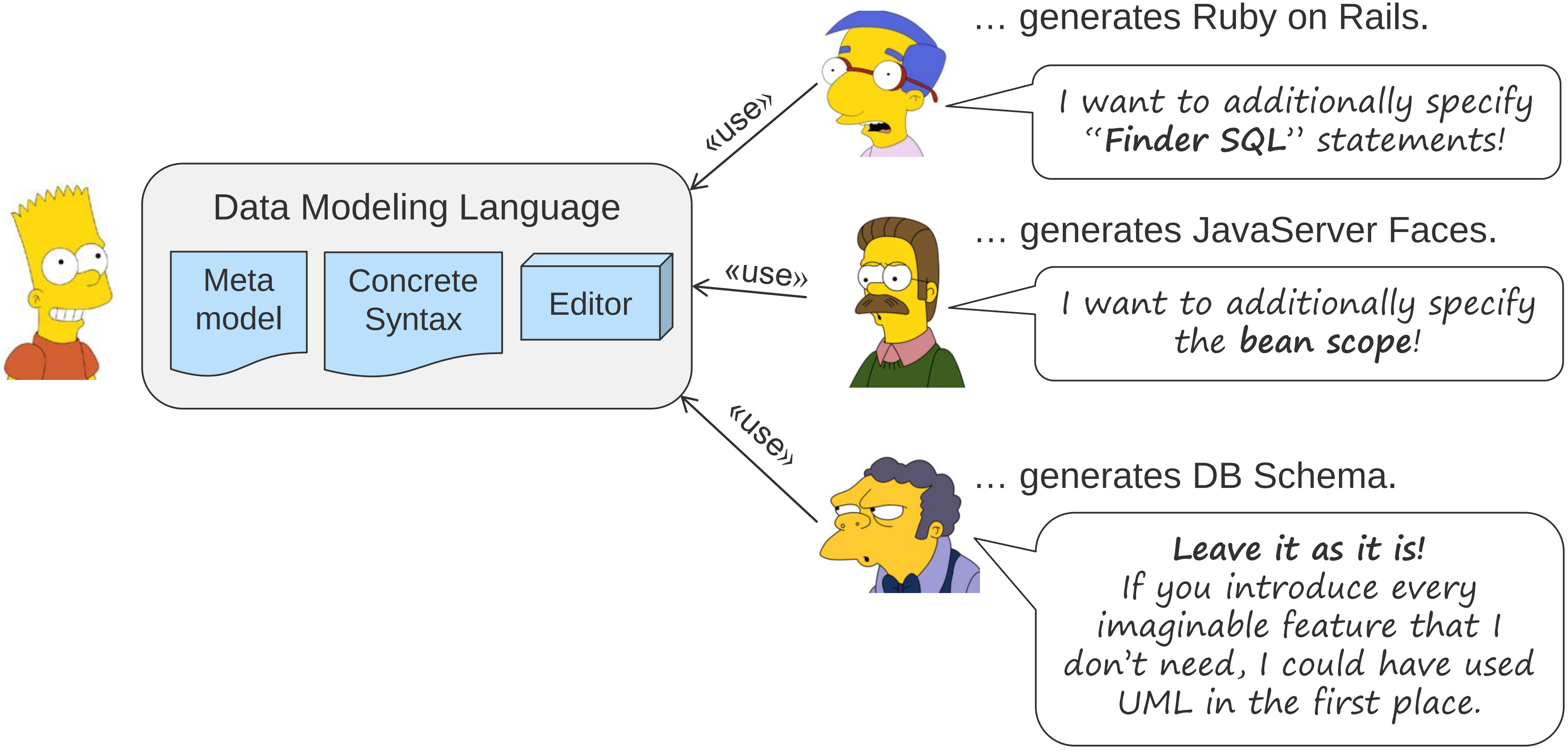
UML Profiles vs DSMLs

These debates concern adopting either UML Profiles or DSMLs **for creating new languages**.

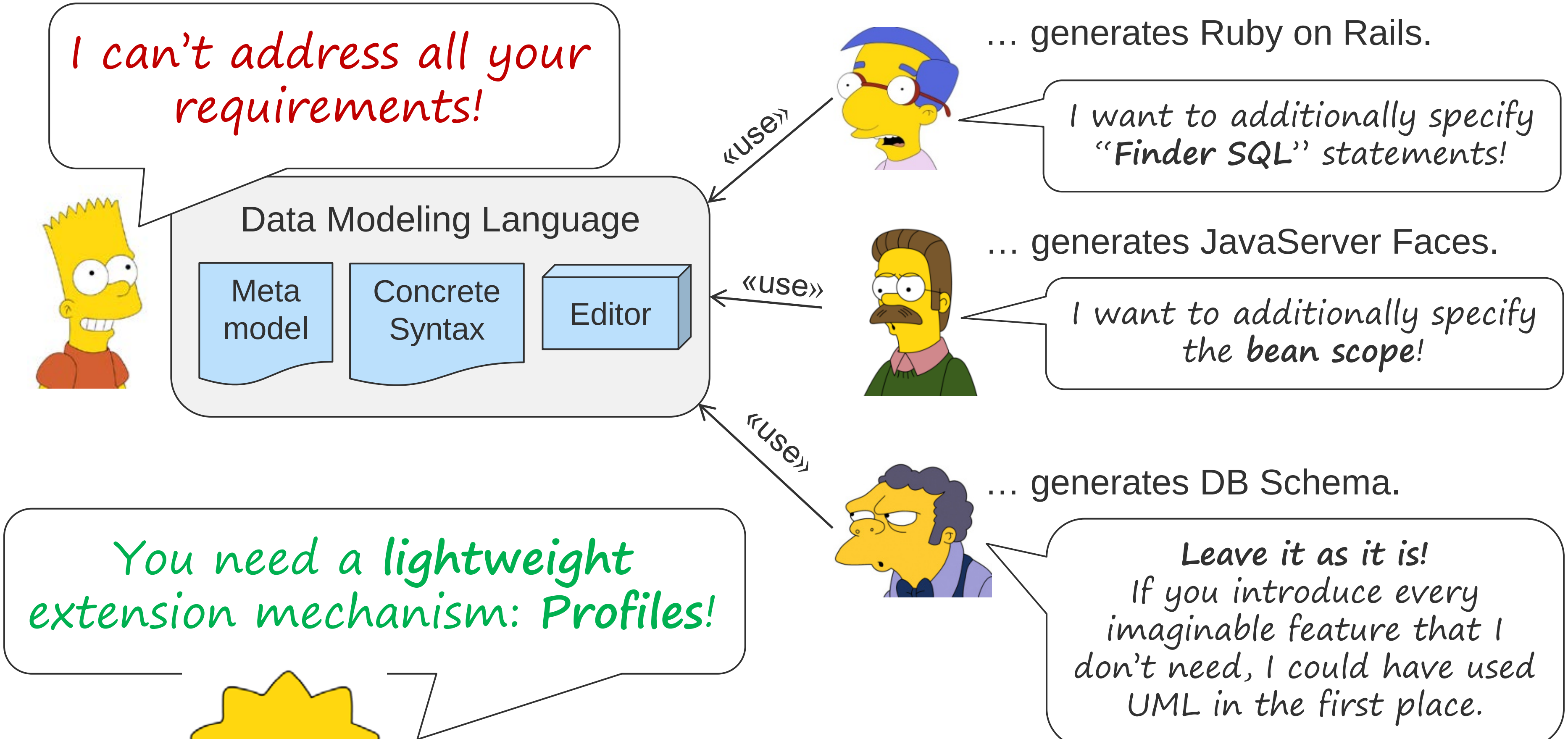
What about **extending existing languages**?

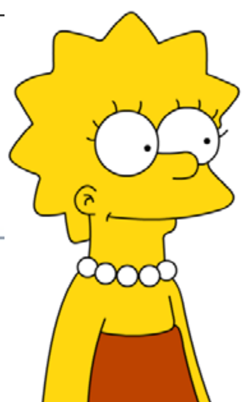
It's your language…
just extend your metamodel.

Motivating Scenario



Motivating Scenario





Other scenarios

- No influence on the metamodel
- "Concern-specific" annotations

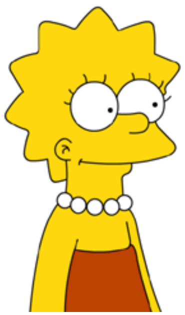
Benefits of Profiles for DSMLs

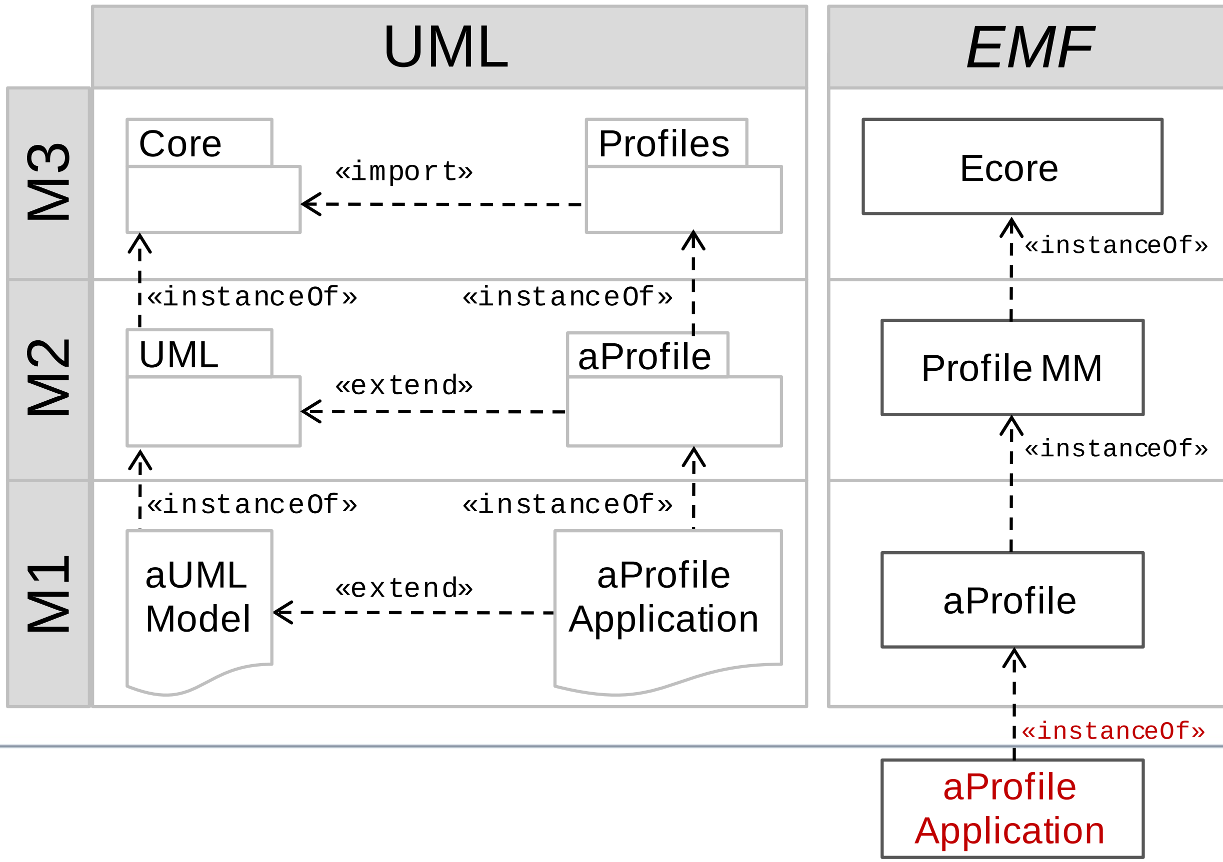
- Lightweight language extension
  - Introduce additional features
  - No need for changing the metamodel and the modeling infrastructure
- Dynamic model extension
  - Existing models can be extended; even by multiple profiles and stereotypes
  - Profile applications are separated from the models (→ no model pollution)
- Preventing metamodel pollution
  - Metamodels represent only information coming from the modeling domain
  - Concern-specific information is defined in a profile
- Model-based representation
  - Profile applications are well-defined (→ they can be validated)
  - Profile applications are just additional models (→ reuse existing frameworks)

The Challenge

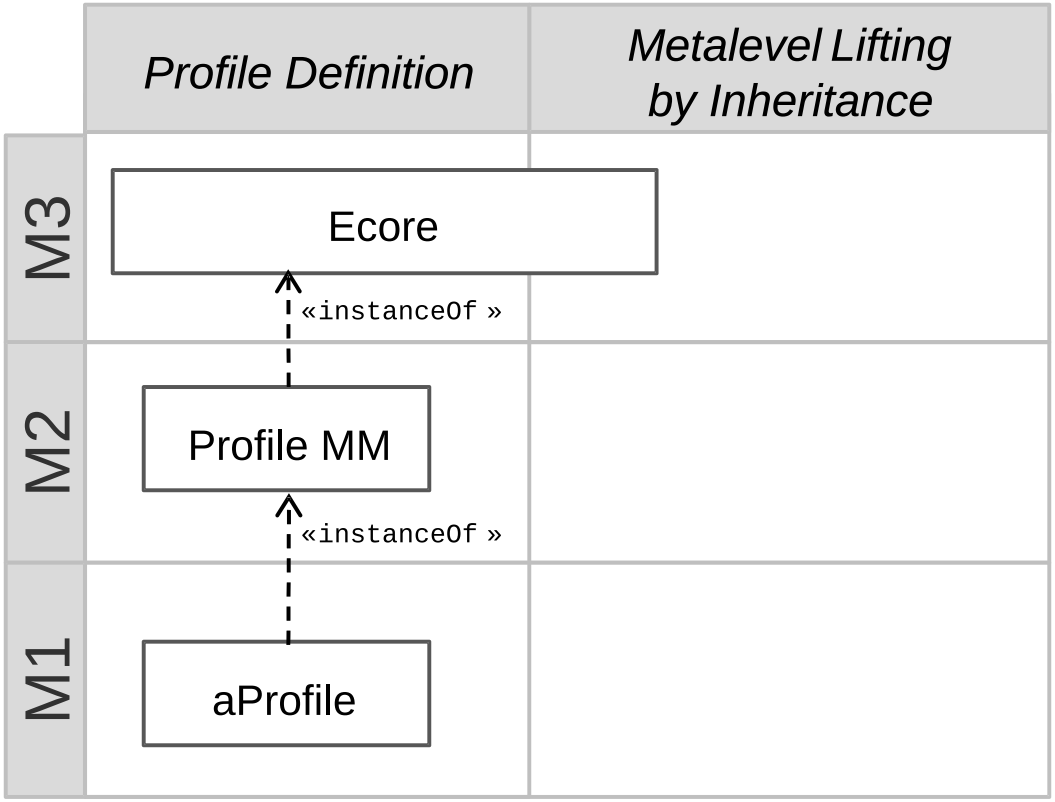
Okay, so let's use profiles!

But wait, that's not so easy with EMF!

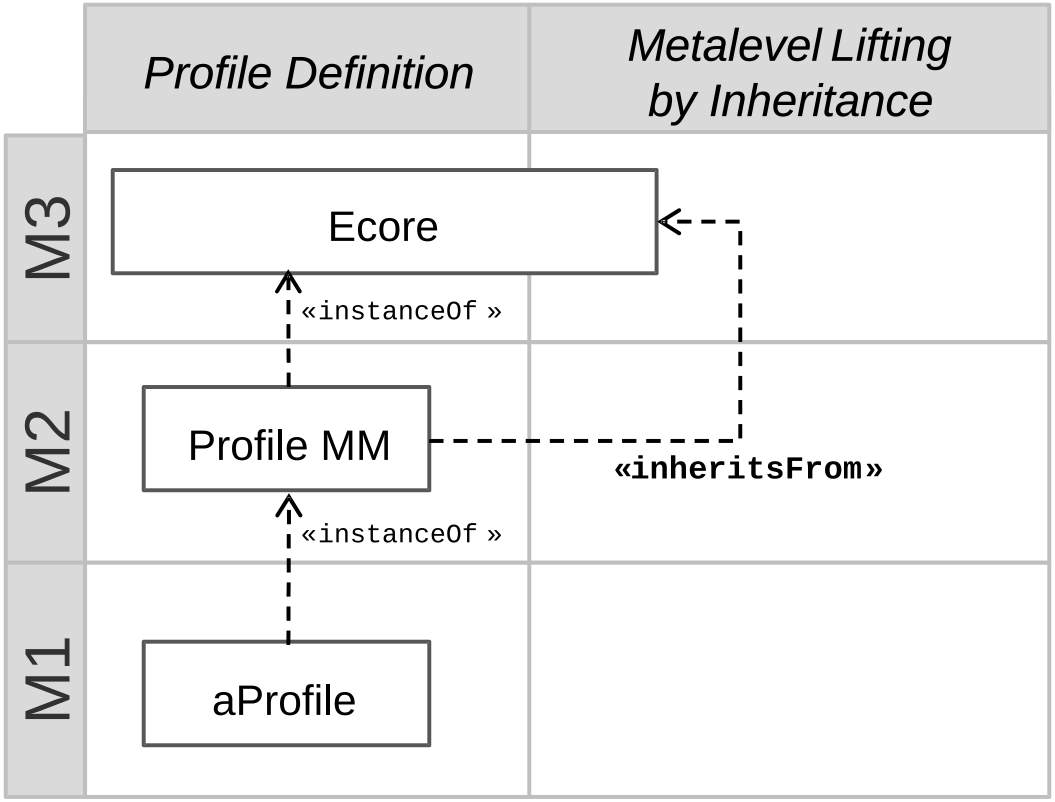




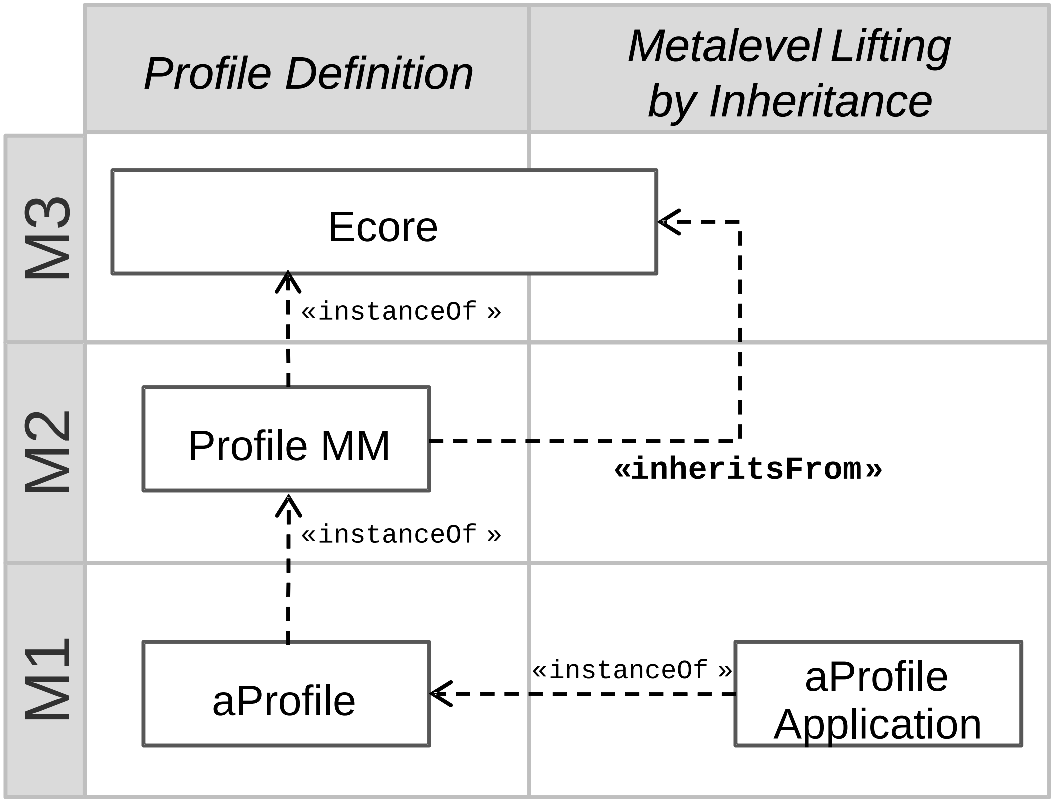
Solution: Metalevel Lifting by Inheritance



Solution: Metalevel Lifting by Inheritance

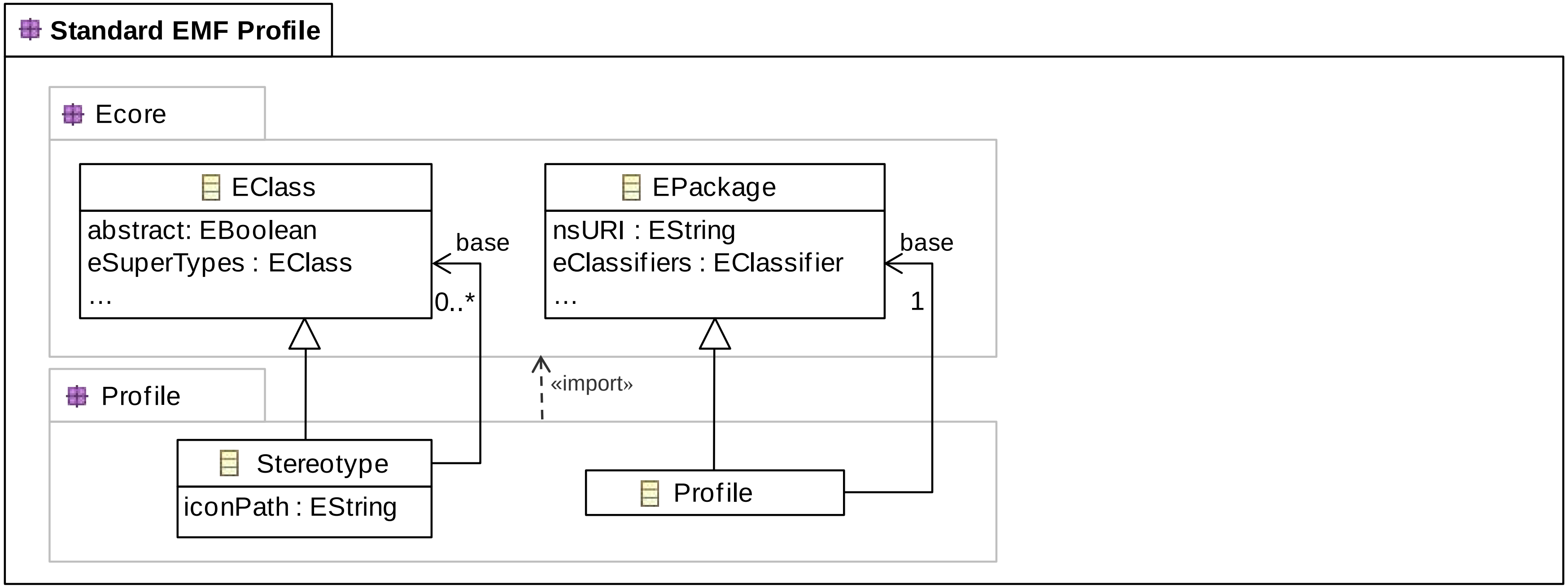


Solution*: Metalevel Lifting by Inheritance

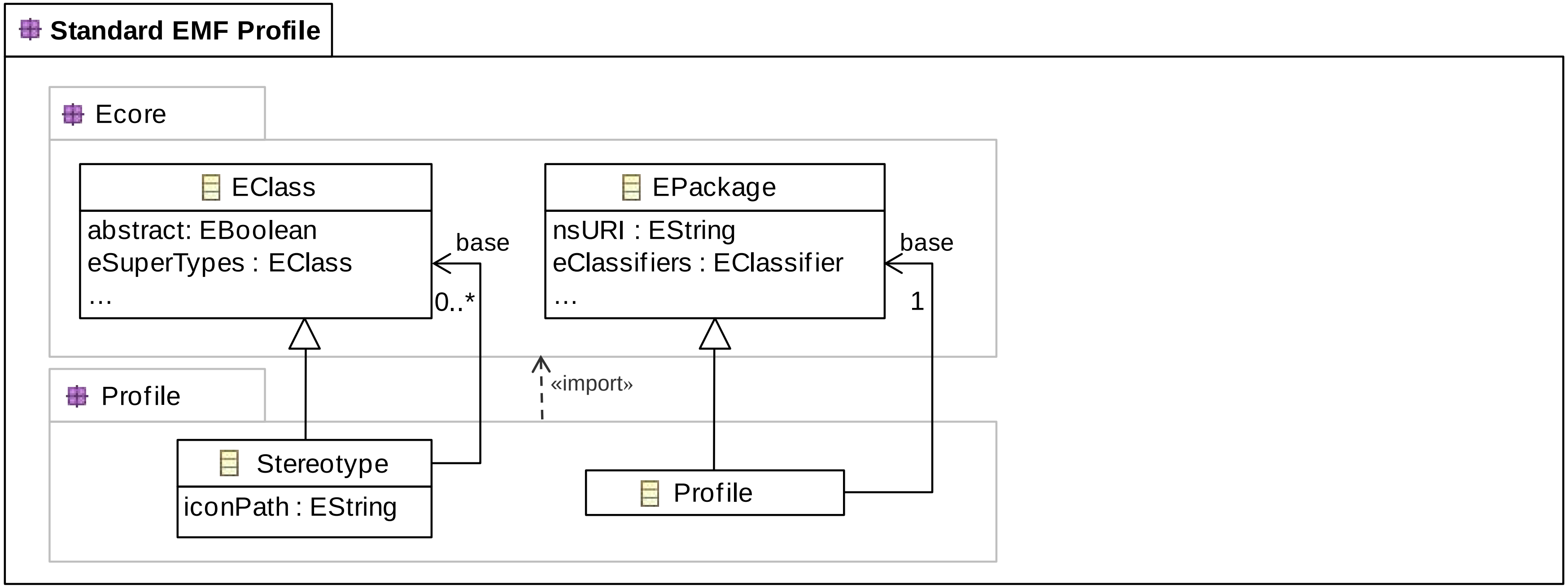


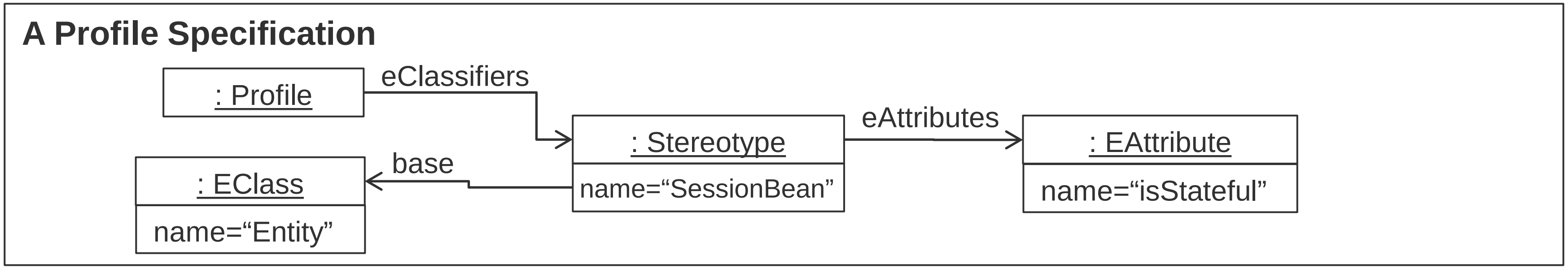
* There is an alternative solution as well (i.e., transformation to a model at M2).

EMF Profile Metamodel

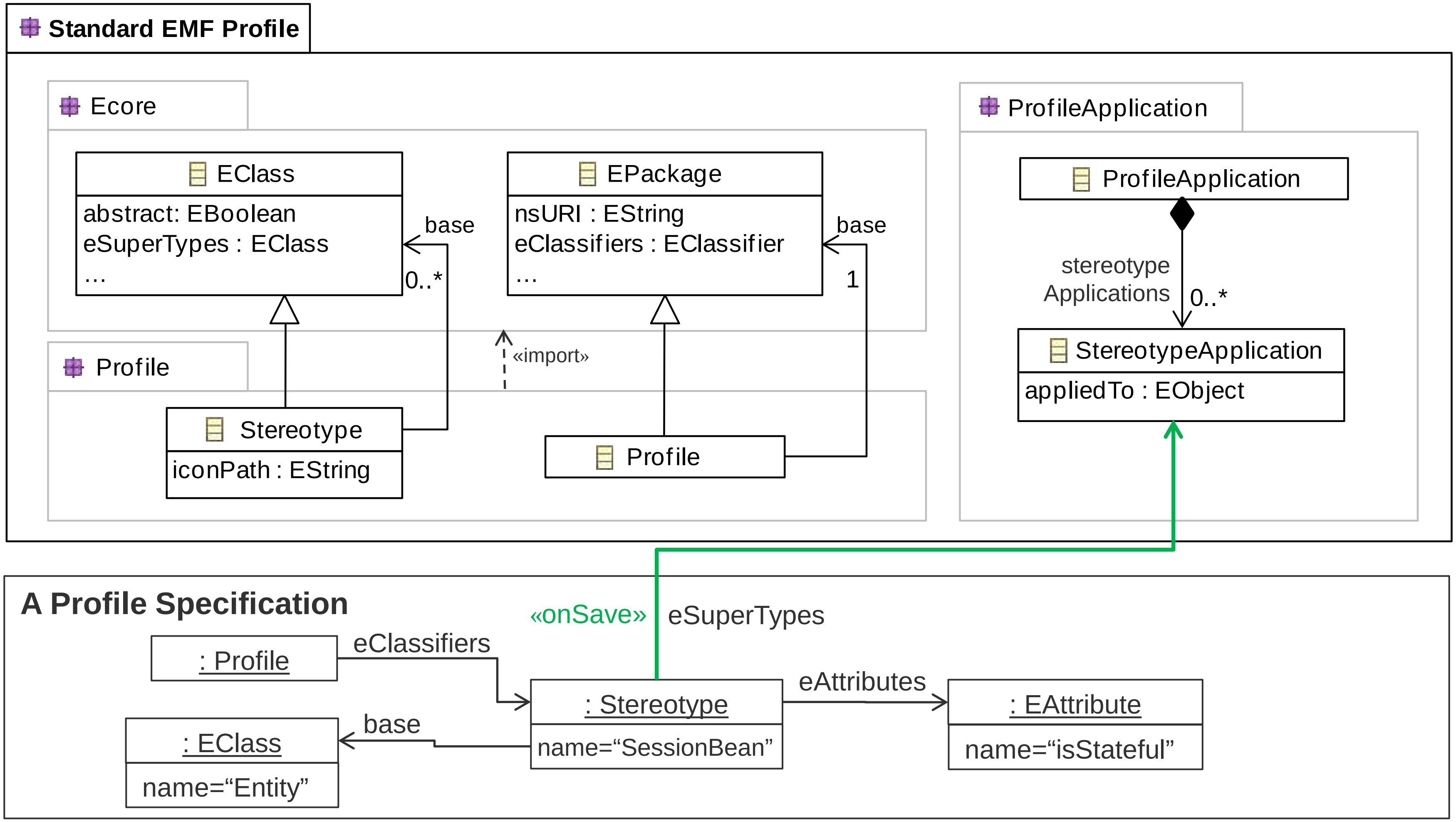


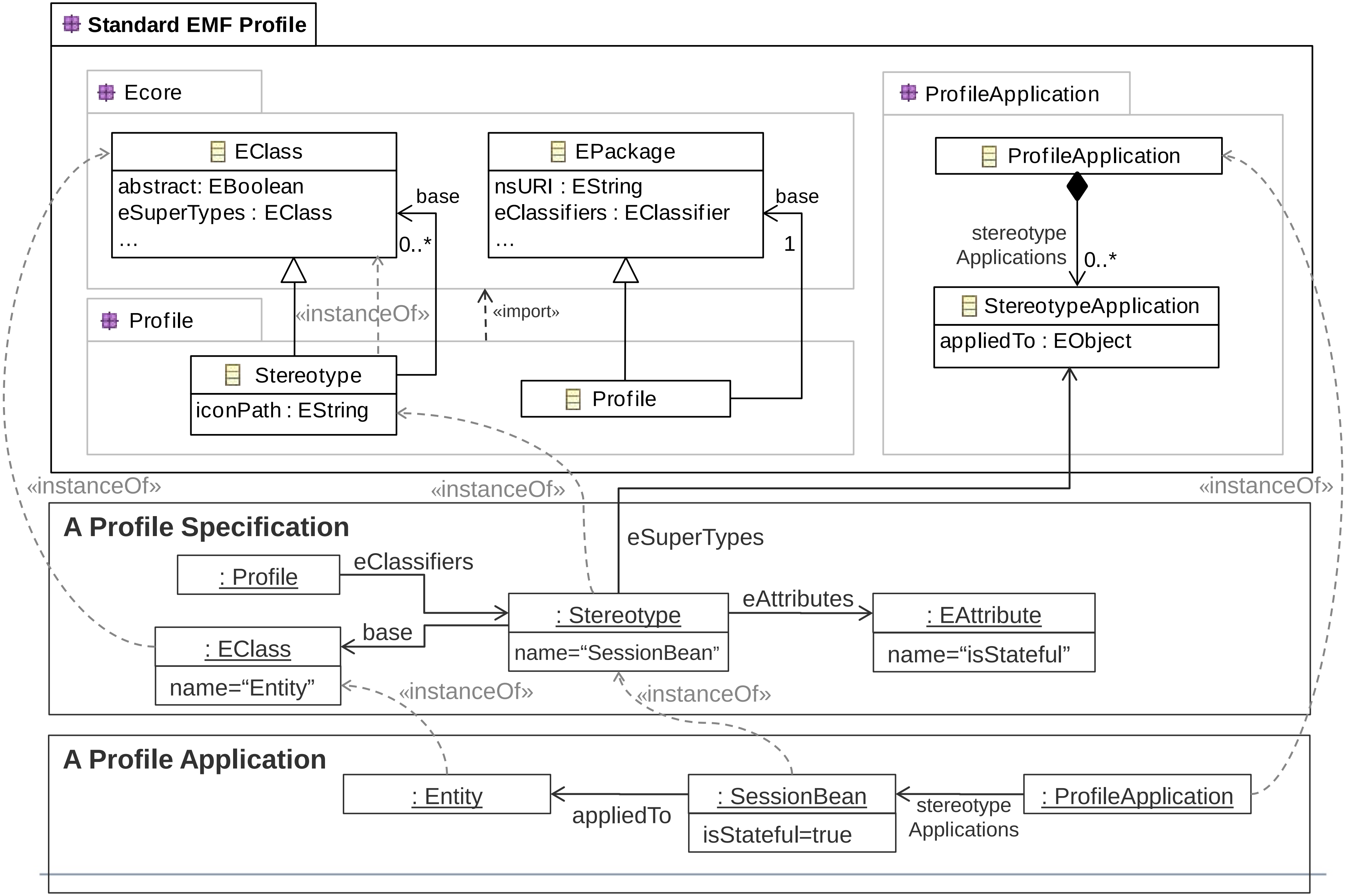
EMF Profile Metamodel and its Instantiation

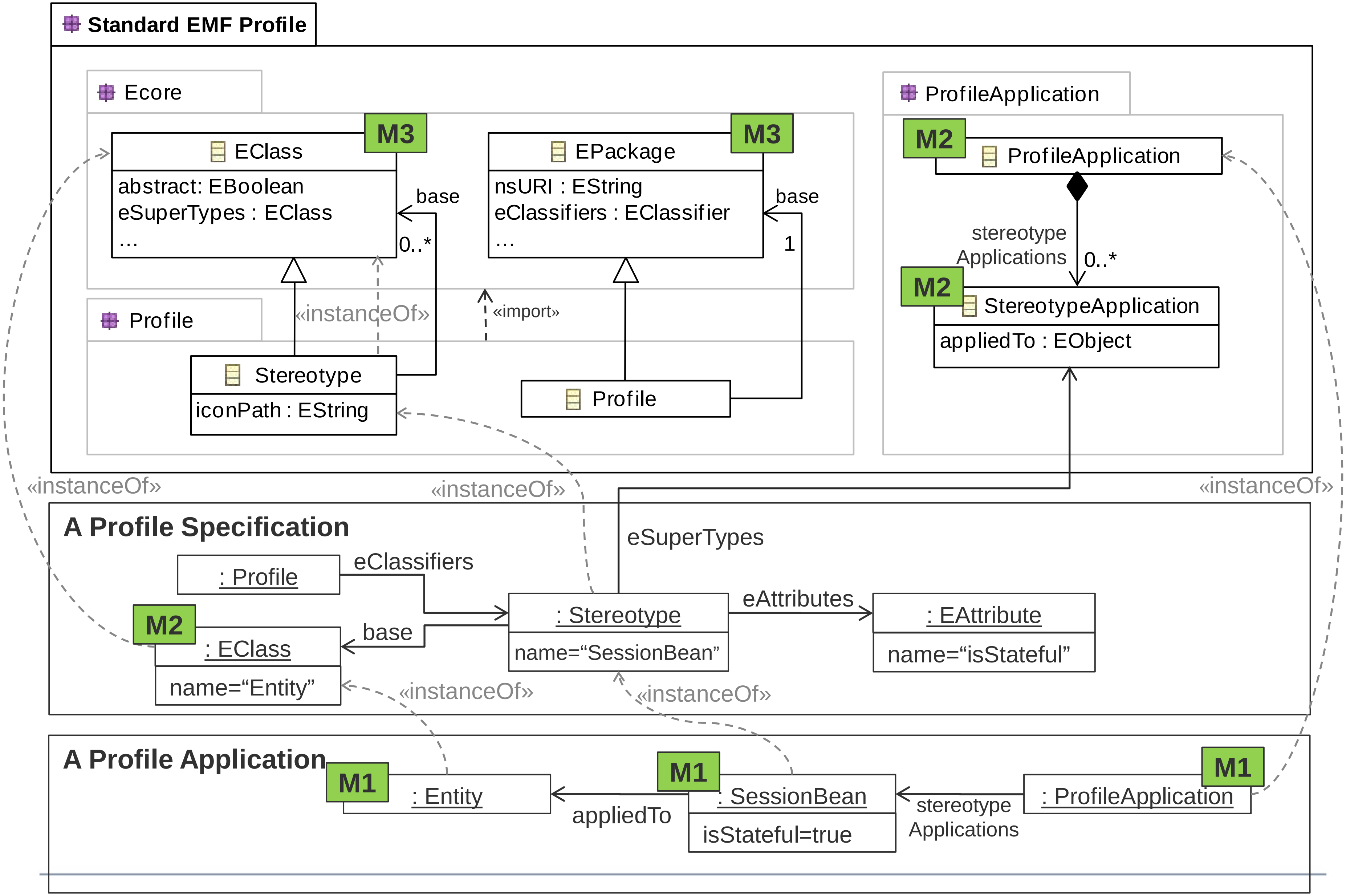


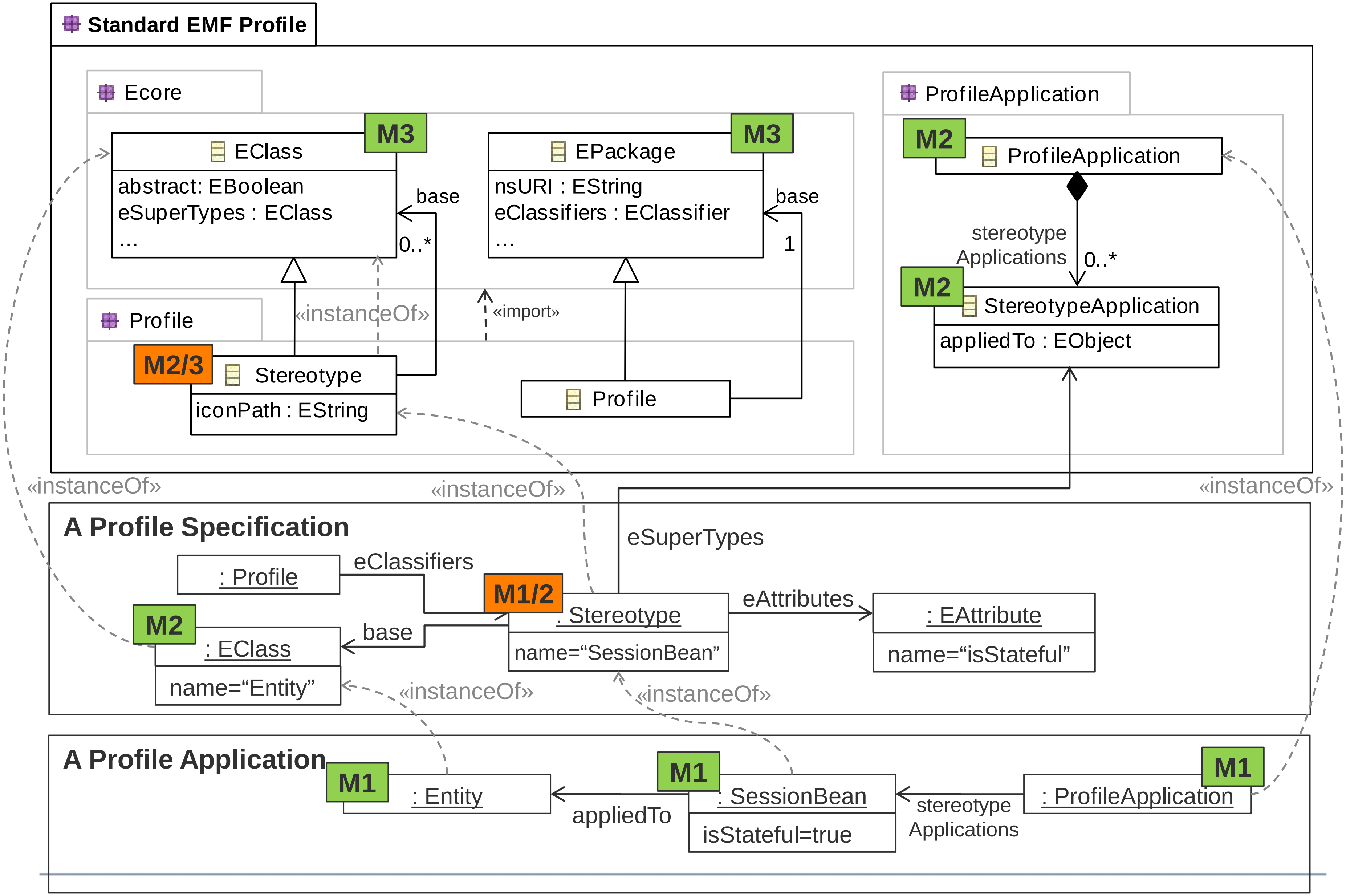


EMF Profile Metamodel and its Instantiation



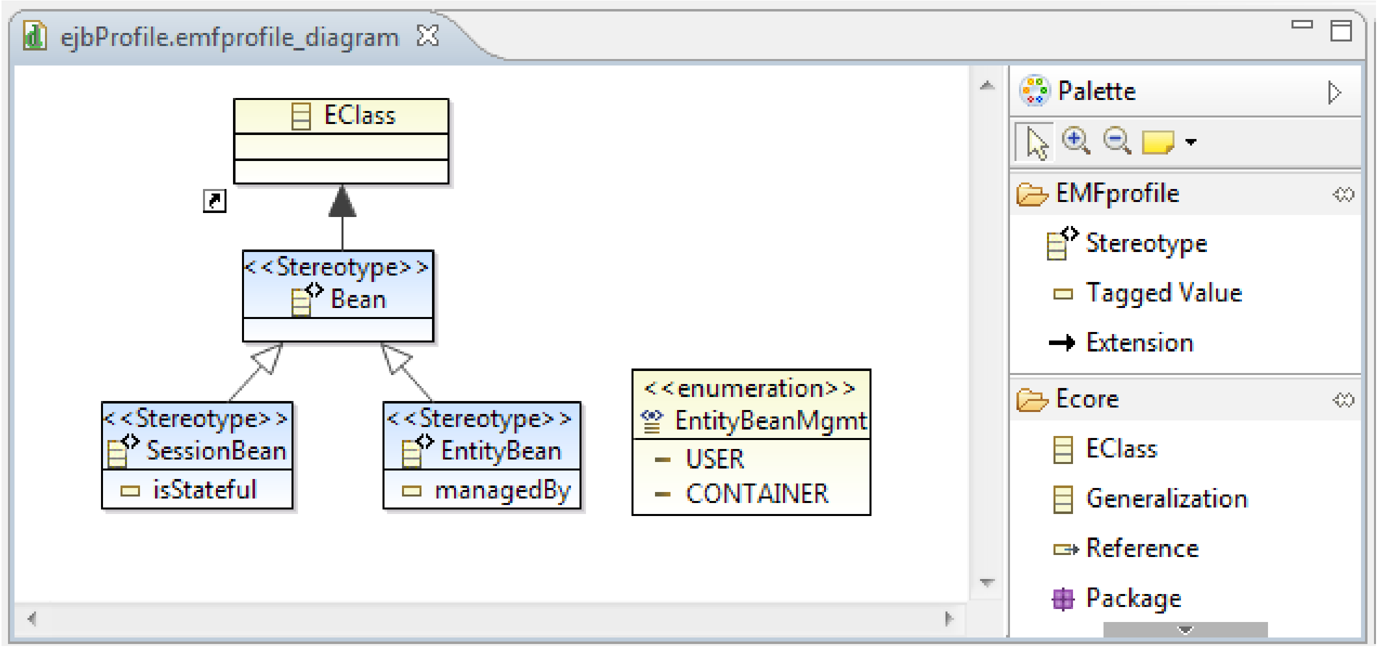

Standard EMF Profile
Ecore
EClass
abstract: EBoolean
eSuperTypes : EClass
…
base
0..*
EPackage
nsURI : EString
eClassifiers : EClassifier
…
base
1
ProfileApplication
ProfileApplication
stereotype Applications
0..*
StereotypeApplication
appliedTo : EObject
Profile
«instanceOf»
«import»
Stereotype
iconPath : EString
Profile
«instanceOf»
«instanceOf»
«instanceOf»
A Profile Specification
eSuperTypes
: Profile
eClassifiers
: Stereotype
name="SessionBean"
eAttributes
: EAttribute
name="isStateful"
: EClass
name="Entity"
base
«instanceOf»
«instanceOf»
A Profile Application
: Entity
appliedTo
: SessionBean
isStateful=true
stereotype Applications
: ProfileApplication


Standard EMF Profile
Ecore
M3
EClass
abstract: EBoolean
eSuperTypes : EClass
…
base
0..*
M3
EPackage
nsURI : EString
eClassifiers : EClassifier
…
base
1
Profile
«instanceOf»
«import»
Stereotype
iconPath : EString
Profile
ProfileApplication
M2
ProfileApplication
stereotype Applications
0..*
M2
StereotypeApplication
appliedTo : EObject
«instanceOf»
«instanceOf»
«instanceOf»
A Profile Specification
eSuperTypes
: Profile
eClassifiers
: Stereotype
name="SessionBean"
eAttributes
: EAttribute
name="isStateful"
M2
: EClass
name="Entity"
base
«instanceOf»
«instanceOf»
A Profile Application
M1
: Entity
appliedTo
M1
: SessionBean
isStateful=true
stereotype Applications
: ProfileApplication
M1

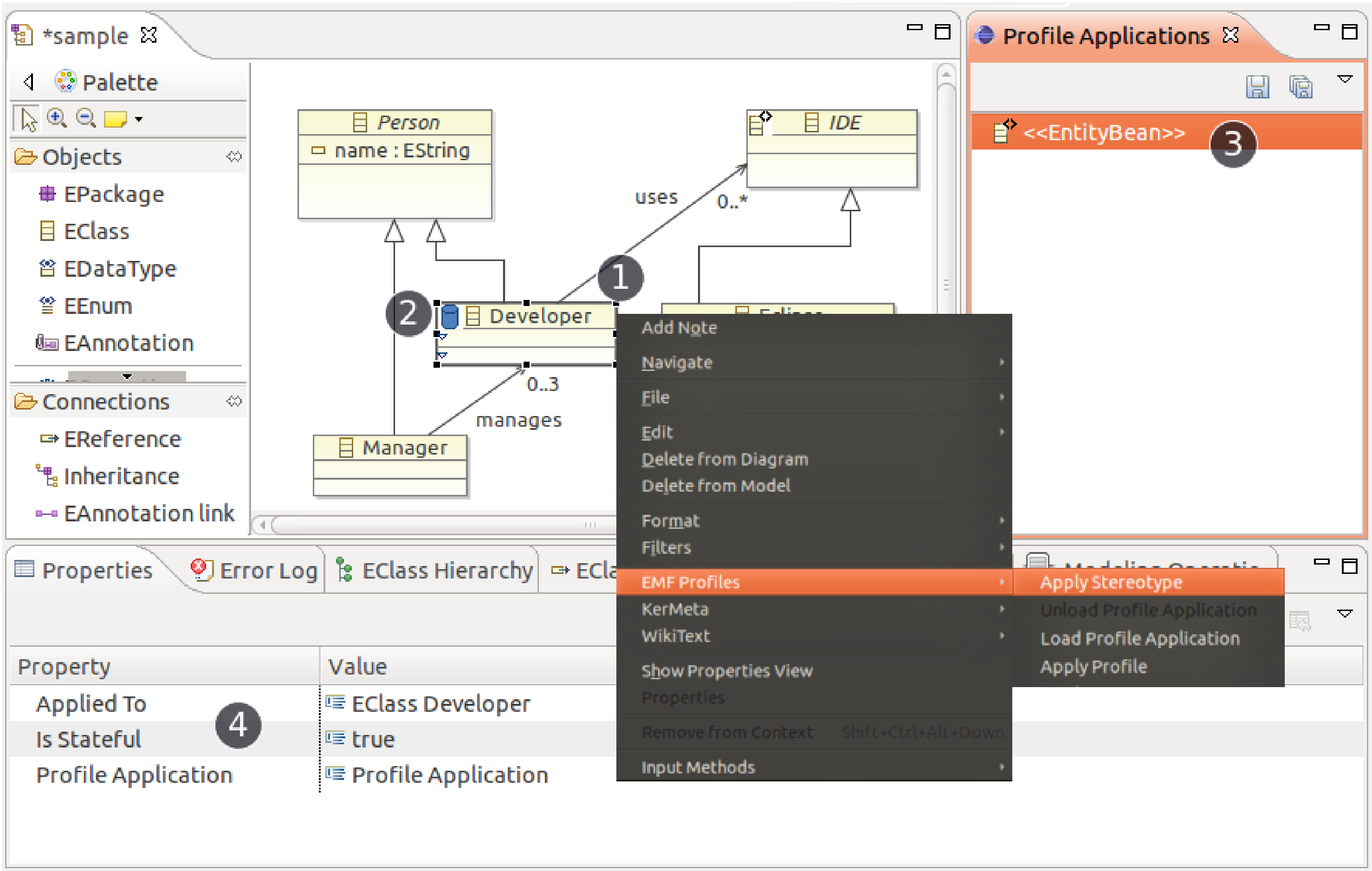


Standard EMF Profile
Ecore
M3
EClass
abstract: EBoolean
eSuperTypes : EClass
…
base
0..*
M3
EPackage
nsURI : EString
eClassifiers : EClassifier
…
base
1
ProfileApplication
M2
ProfileApplication
stereotype Applications
0..*
M2
StereotypeApplication
appliedTo : EObject
Profile
M2/3
Stereotype
iconPath : EString
«instanceOf»
«import»
Profile
«instanceOf»
«instanceOf»
«instanceOf»
A Profile Specification
eSuperTypes
: Profile
eClassifiers
M1/2
: Stereotype
name="SessionBean"
eAttributes
: EAttribute
name="isStateful"
M2
: EClass
name="Entity"
base
«instanceOf»
«instanceOf»
A Profile Application
M1
: Entity
appliedTo
M1
: SessionBean
isStateful=true
stereotype Applications
M1
: ProfileApplication

Example: EJB Profile Specification



Example: EJB Profile Application

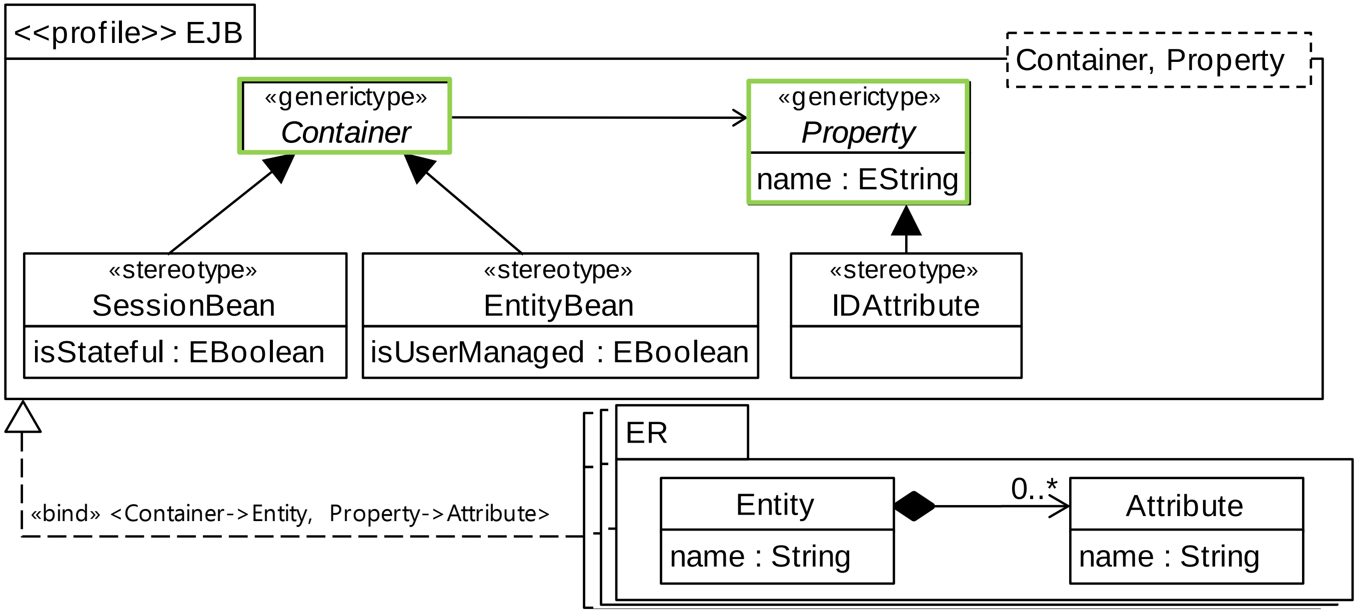


From UML Profiles to EMF Profiles **and Beyond**

- UML Profiles can be specified for UML only
  - → No need for further reuse of profiles for other languages
- EMF Profiles can be specified for <u>every</u> Ecore-based language
  - → Reuse a profile for more than one language
- Generic Profiles
  - Reuse a profile for several "*user-selected*" DSMLs
- Meta Profiles
  - Reuse a profile for *all* DSMLs

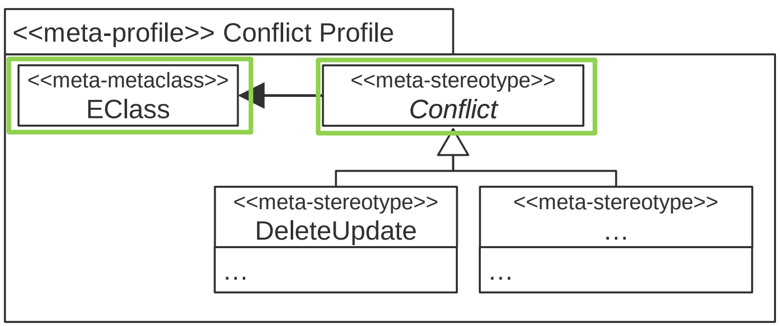
Generic Profiles

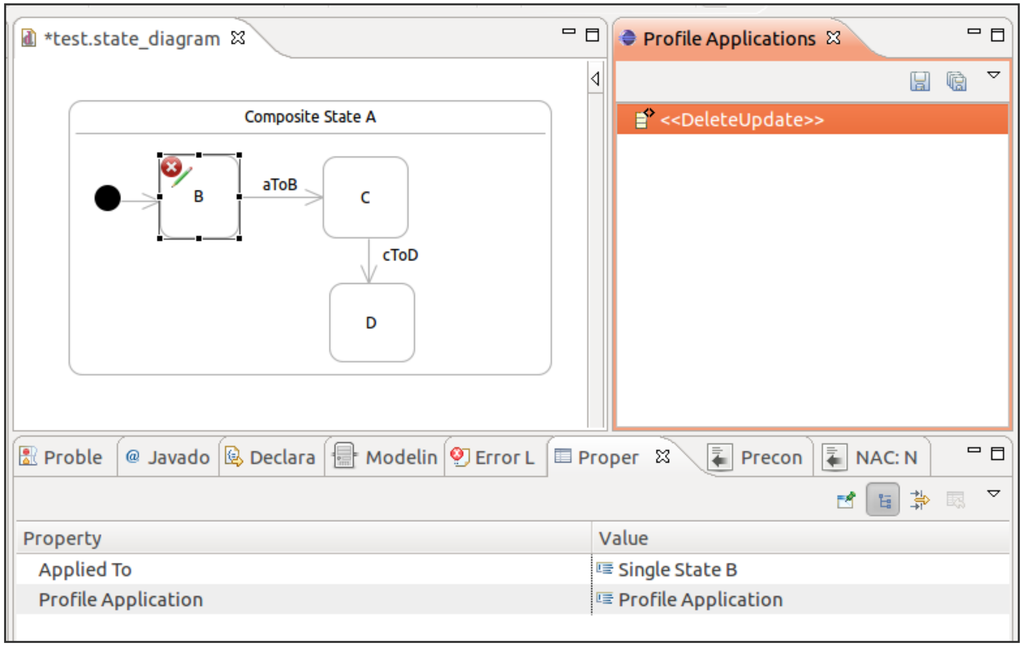
- Reuse a profile for several “user-selected” DSMLs
  - Extend generic types instead of concrete types
  - Bind generic types to concrete types to apply a profile
  - Use OCL to further restrict valid bindings



Meta Profiles

- Reuse a profile for *all* DSMLs (at once)
  - Each “Meta Stereotype” may be applied to every base type
  - Useful for general annotations
    - E.g., Conflict Profile





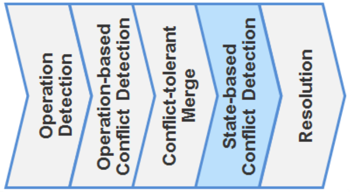
Implementation

- Based on the Eclipse Modeling Framework
- Supports extending every Ecore-based DSML
- Uses the Decoration Service to show icons in GMF editors

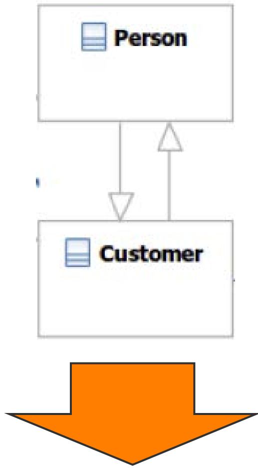
- Open Source (EPL 1.0)
  - http://code.google.com/a/eclipselabs.org/p/emf-profiles/
- Try EMF Profiles!
  - Eclipse Update Site
    http://www.modelversioning.org/emf-profiles-updatesite/
- Contact us and get involved!
  - http://groups.google.com/group/emf-profiles

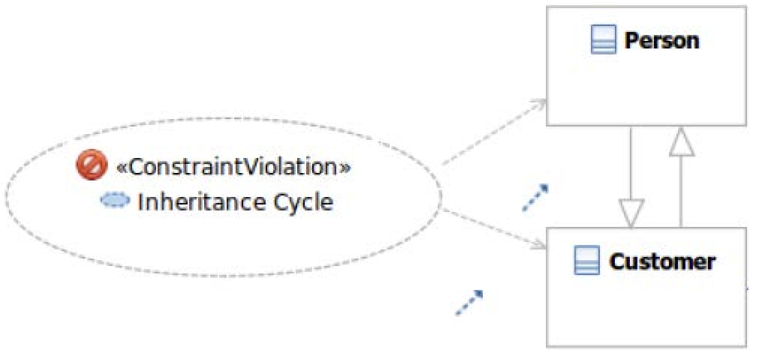
State-based Conflict Detection

Validate of the Result from Conflict-tolerant Merging



- General well-formedness rules are covered by conflict patterns already
  - Unique container
  - No cyclic containments
- But, language-specific validation rules need to be checked
  - Lower/upper bound of multi-valued features
  - Additional OCL Constraints
- Therefore, after conflict-tolerant merging, the resulting state is validated
  - If a constraint is invalid
  - → New state-based conflict annotation
- Reuse EMF validation framework
  - Annotate context elements of reported errors/warnings





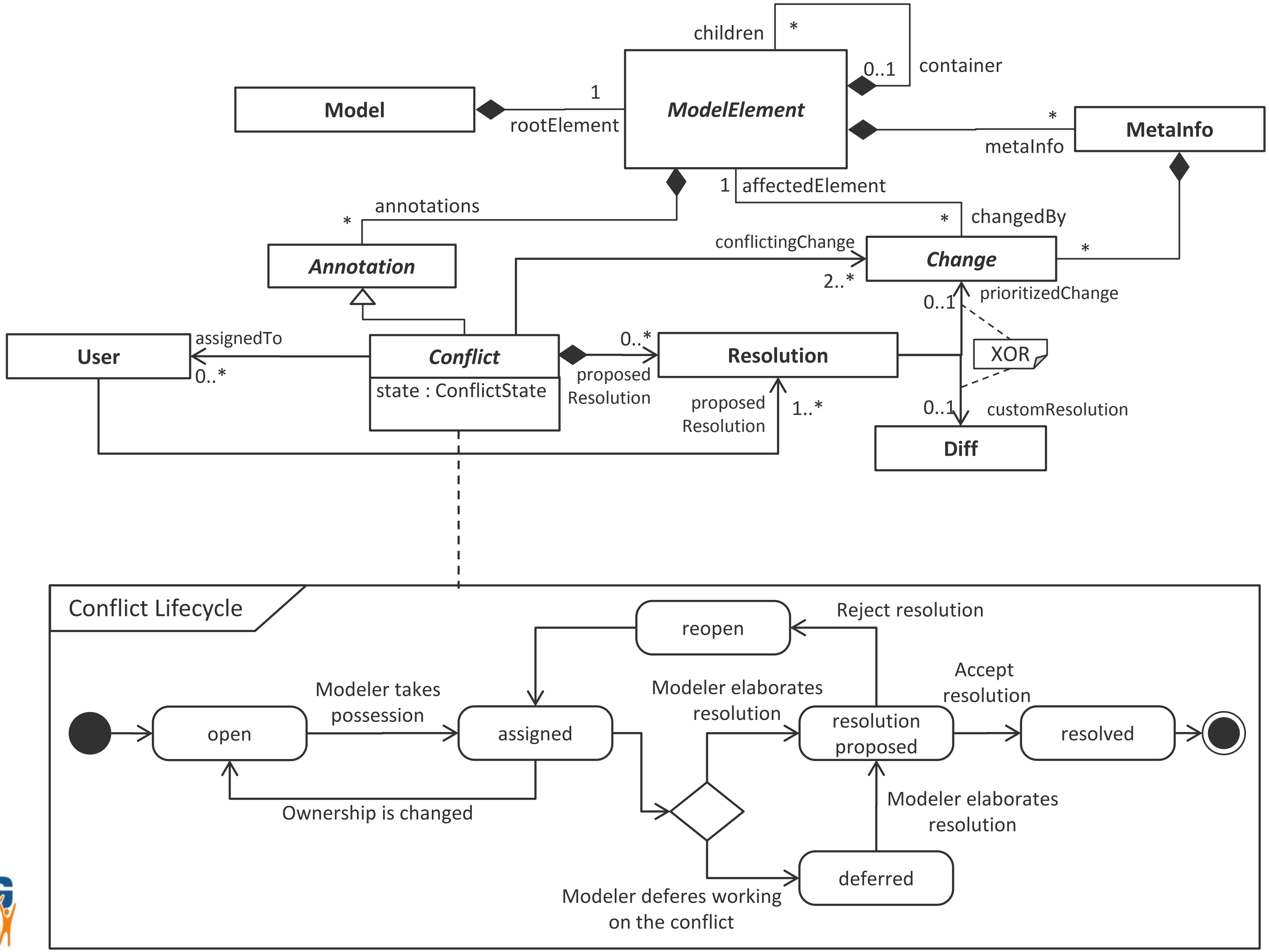
Resolution

- Dedicated conflict resolution view
  - Based on the conflict-tolerant merge result and the conflicts annotations
  - Allows to…
    - … resolve conflicts manually (e.g., use this change or that change)
    - … resolve conflicts by predefined resolution rules
  - Resolutions can be attached to a conflict annotation

- Lifecycle of a conflict
  - Conflicts have a state (e.g., open or resolved)
  - Cf. next slide

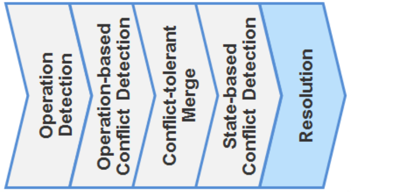
Resolution

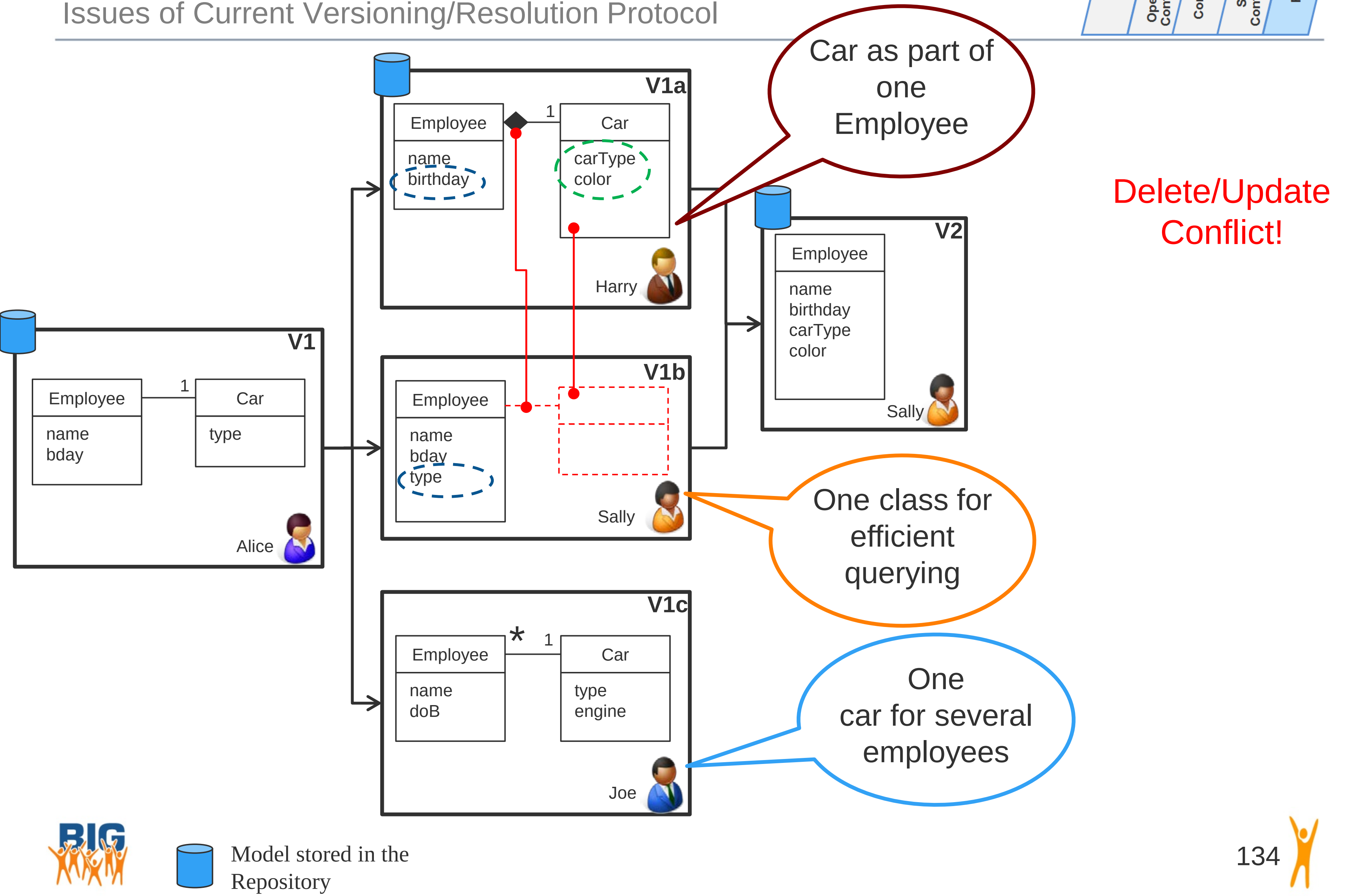
Conflict Resolution Model: The Lifecycle of Conflicts



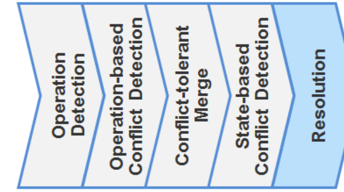
Example: Conflict-tolerant Merging Improves the Result

Issues of Current Versioning/Resolution Protocol





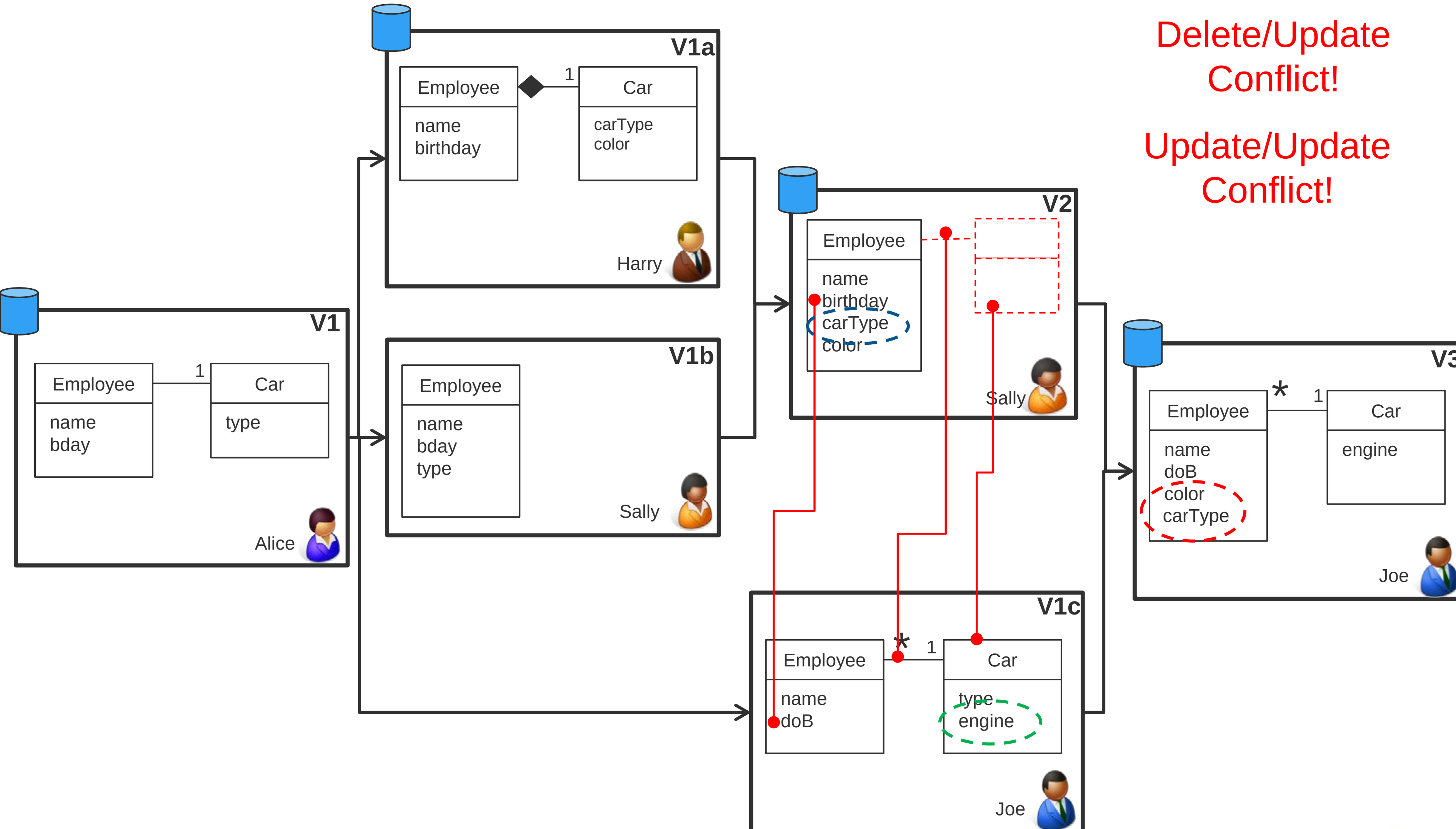
Example: Conflict-tolerant Merging Improves the Result



Issues of Current Versioning/Resolution Protocol

Delete/Update Conflict!

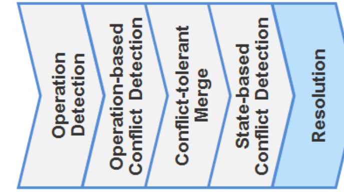
Update/Update Conflict!

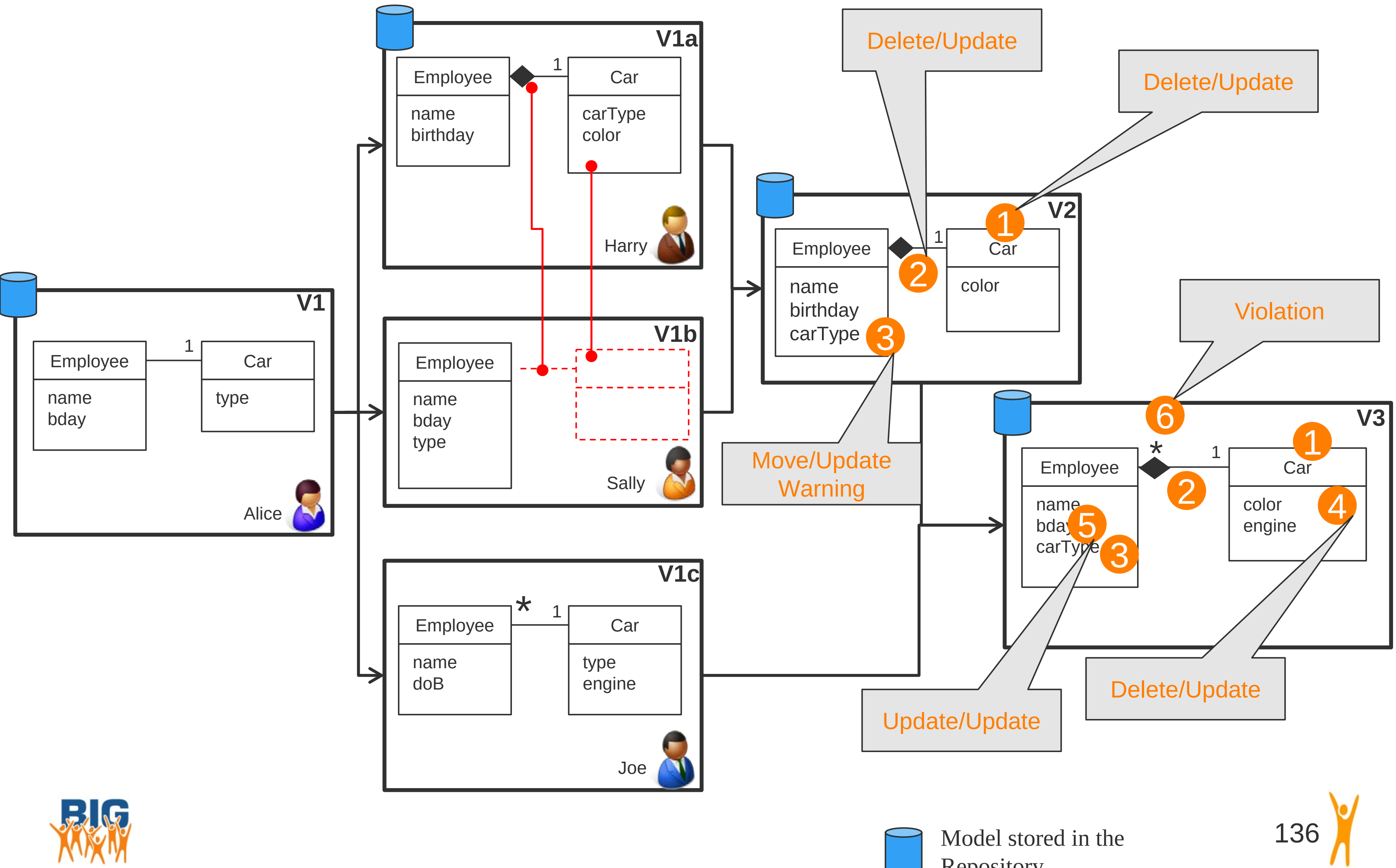


Model stored in the Repository

Example: Conflict-tolerant Merging Improves the Result

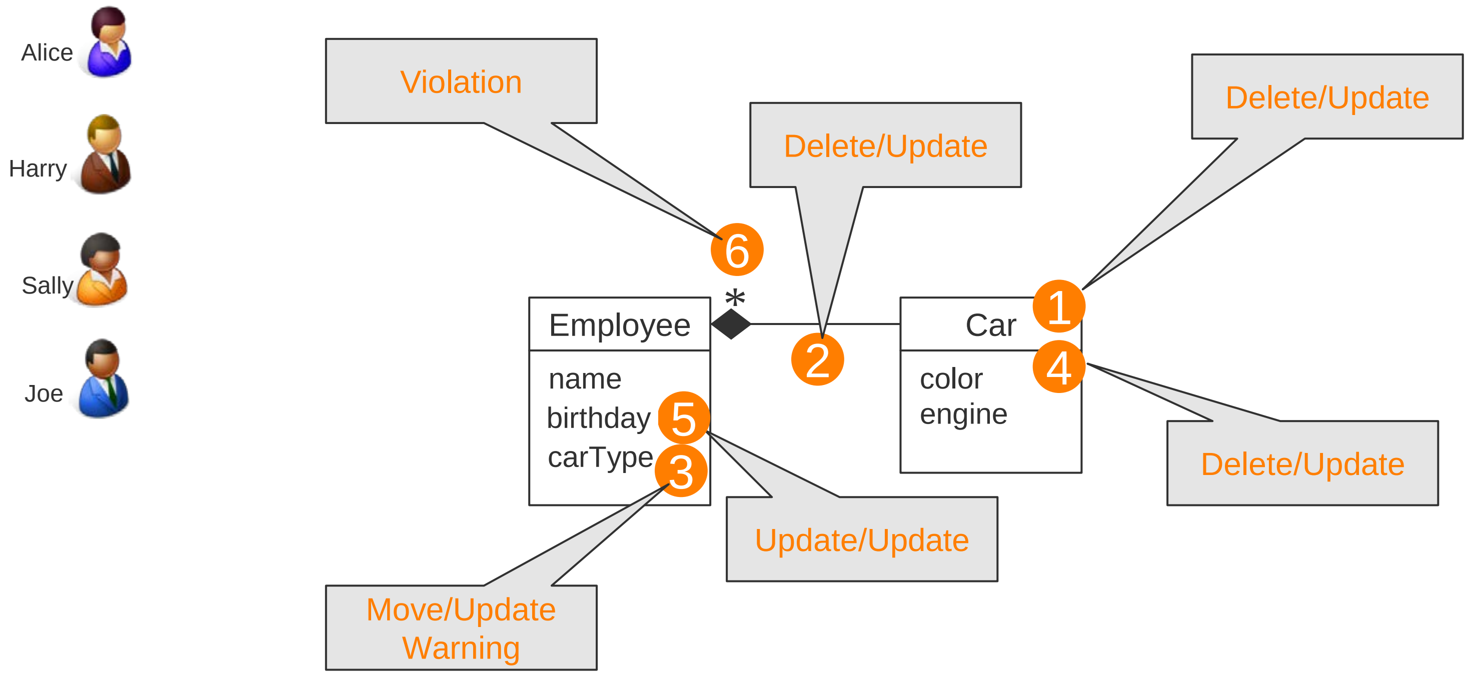
Result of Conflict-tolerant Merge



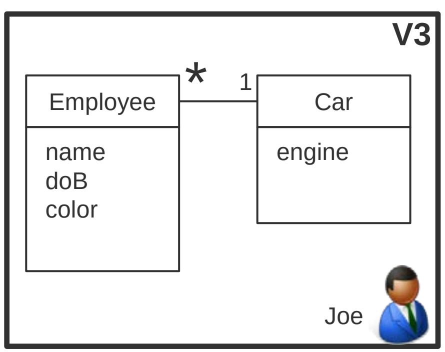


Example: Conflict-tolerant Merging Improves the Result

Collaborative Resolution based on Conflict-tolerant Merge



Result of Standard Versioning Process

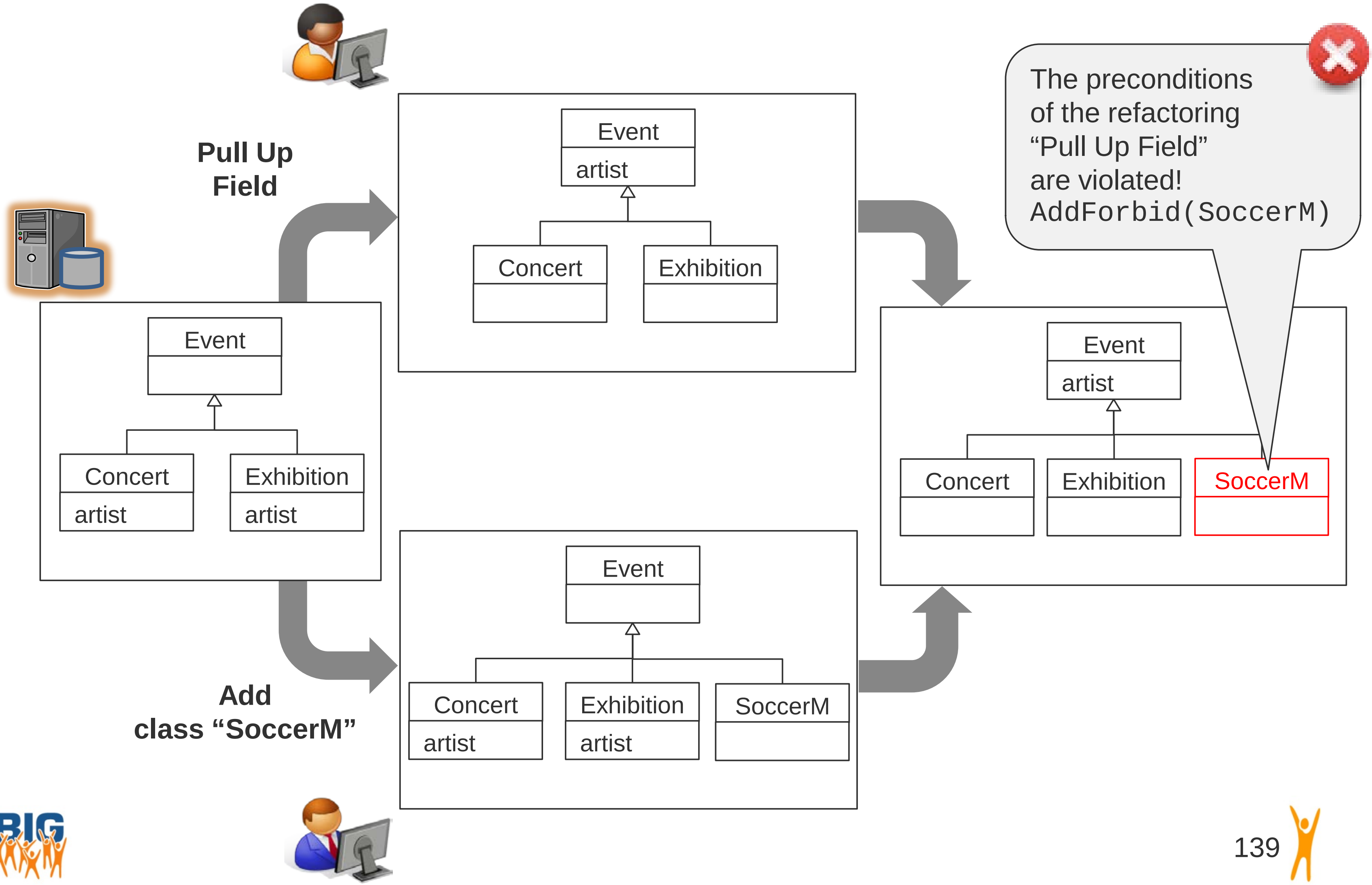


Semi-Automated Resolution

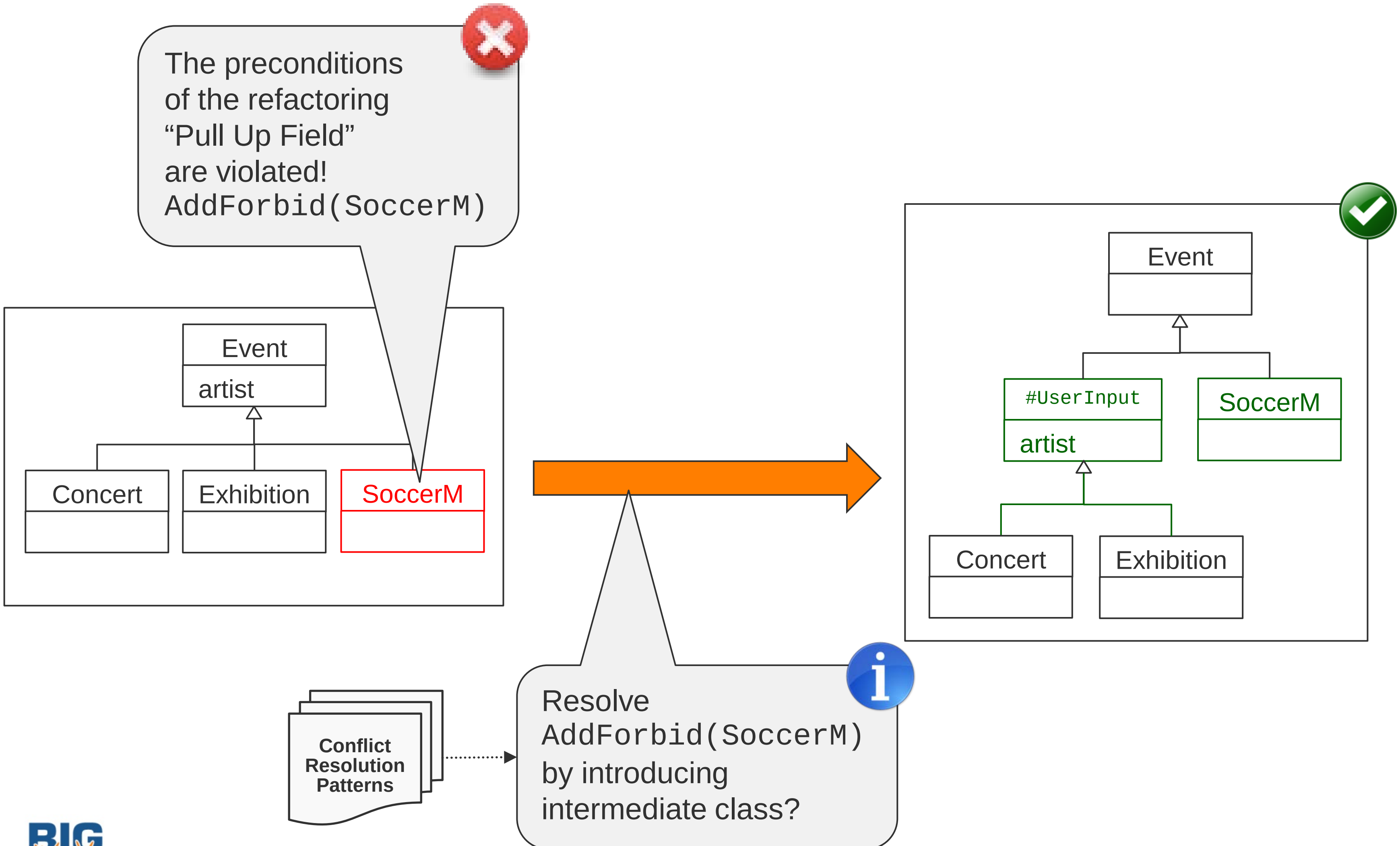
- Conflict Resolution Recommender
  - For increasing efficiency and avoiding errors

- Conflict Resolution Patterns
  - Preconditions defined for
    - the merged model
    - the applied changes
    - the annotated conflicts
  - Actions
    - Operations to be applied to resolve the conflict
  - Formally defined using graph transformation systems

Example: Resolution Recommender



Example: Resolution Recommender



Outline

- Context of Model Versioning
- Foundations of Model Versioning
- Conflict Categorization
- Adaptable Model Versioning
- **Future Challenges**

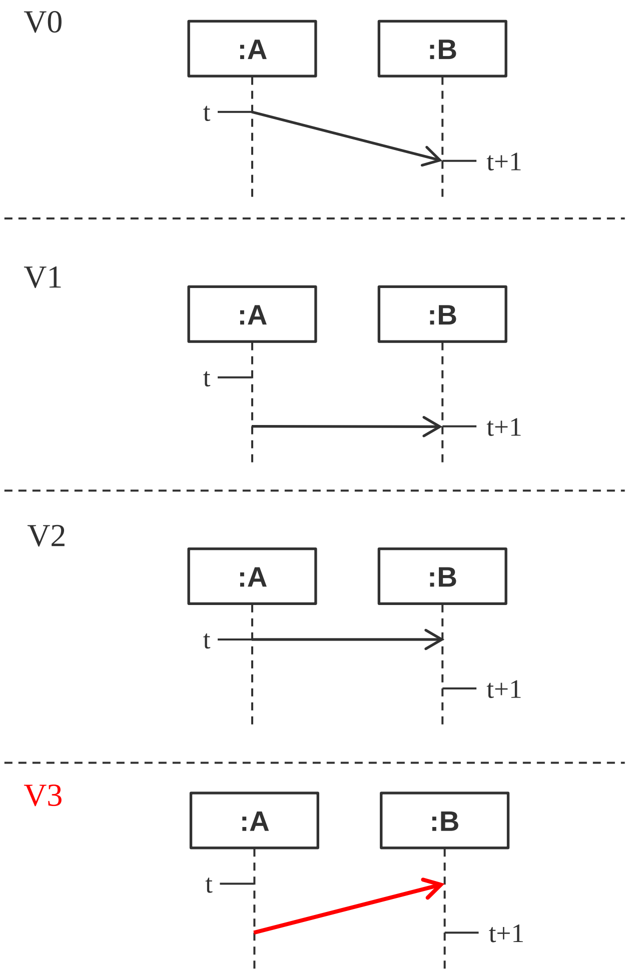
Future Challenges

Consistency-aware Versioning

- Consistency rules, e.g.
  - Metamodel
  - OCL Constraints
  - Requirement documents
  - Additional models

- Merged model violates consistency rules even if both versions are consistent

→ Trace back to the relevant changes causing the violation



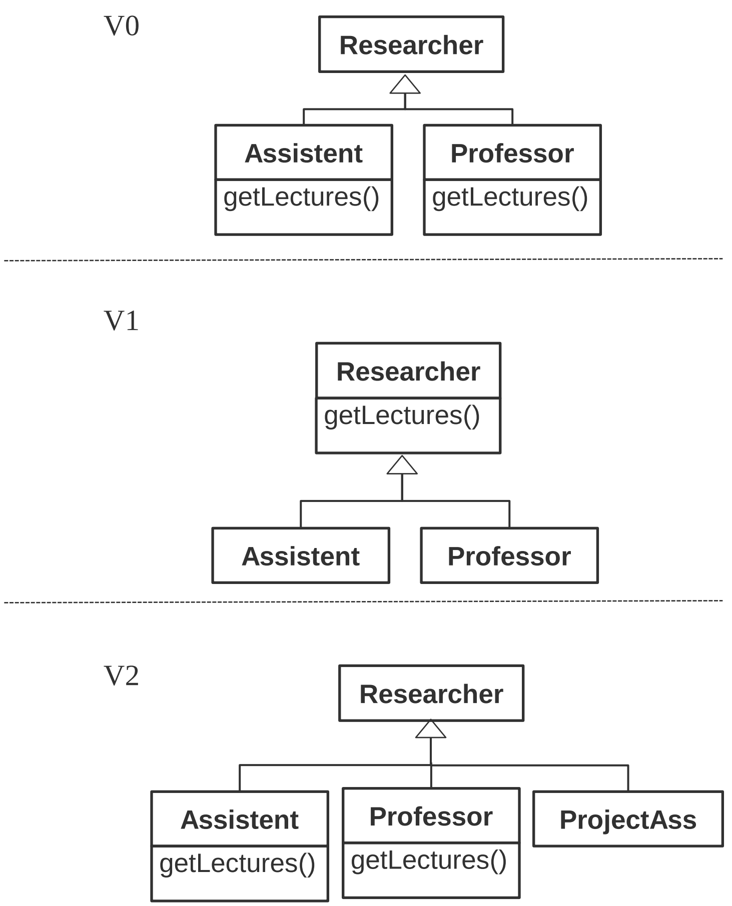
Lopez-Herrejon, R.E., & Egyed, A. (2010). *Detecting Inconsistencies in Multi-View Models with Variability*. Proc. of the European Conference on Modelling Foundations and Applications (ECMFA), Springer, 217-232.

Future Challenges

Intention-aware Versioning

- Merged version should incorporate all intentions of each modeler

→ Treat composite operations such as *model refactorings* as first-class entities



→ Going beyond composite operations: E.g., a set of operations has been applied in order to fix a bug… is the bug still fixed after merging?

→ Other ways to capture and respect the intention?

Dig, D., Manzoor, K., Johnson, R., & Nguyen, T.N. (2007). *Refactoring-aware Configuration Management for Object-oriented Programs*. Proc. of the 29th International Conference on Software Engineering (ICSE), IEEE Computer Society, 427-436.

Future Challenges

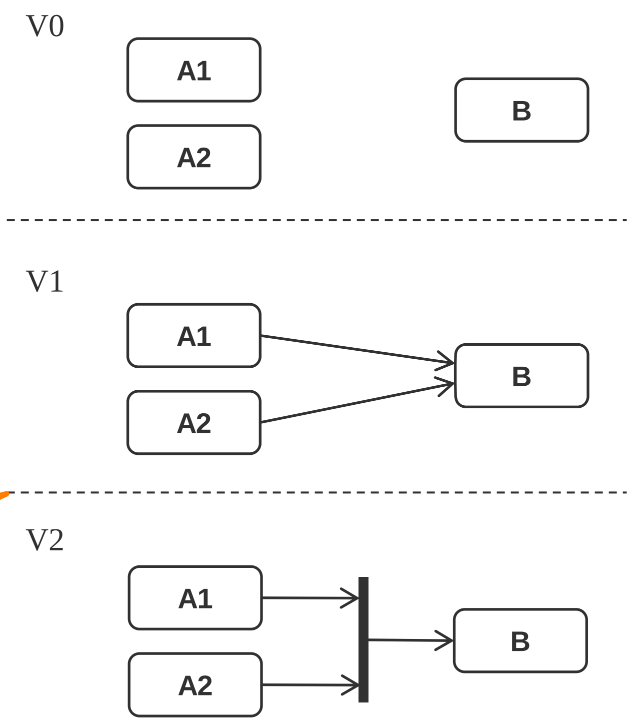
Semantics-aware Versioning

- Model differencing only on syntactic level

→ Formal definition of the modeling language's semantics

→ Mapping between modeling language and semantic domain

→ Including intra-model dependencies



Different syntax, equal semantics

Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., & Zave, P. (2007). *Matching and Merging of Statecharts Specifications*. Proc. of the 29th International Conference on Software Engineering (ICSE), IEEE, 54-64.

Maoz, S., Ringert, J.O., & Rumpe, B. (2010). *A Manifesto for Semantic Model Differencing*. Proc. of the International Workshop on Models and Evolution (ME2010) @ MoDELS'10.

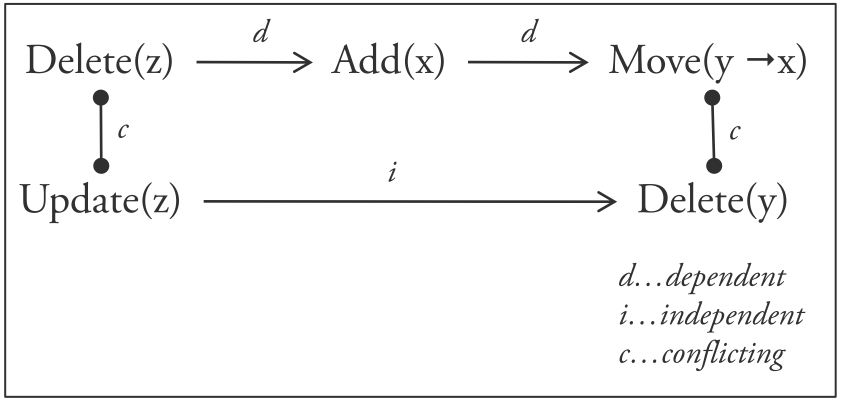
Future Challenges

Conflict Dependencies

- Dependencies between a sequence of changes
- Leading to dependencies between conflicts

→ Efficient detection between more complex changes like refactorings or violations

→ Optimal order for resolving conflicts



Koegel M, Herrmannsdoerfer M, Wesendonk O., & Helming J. (2010). *Operation-based Conflict Detection on Models*. Proc. of the International Workshop on Model Comparison in Practice (IWMCP) @ TOOLS'10.

Küster, J., Gerth, C., & Engels, G. (2009). *Dependent and Conflicting Change Operations of Process Models*. Proceeding of the Conference on Model Driven Architecture - Foundations and Applications, vol. 5562 of LNCS, Springer, 158-173.

Future Challenges

- Co-evolution of diagram layout information with the model
- Existing approaches neglect the nature of 2D diagram layout

→ Which concurrently performed layout change is in fact a contradicting change of the **layout** (more fuzziness required than for models)?

→ Respect the preservation of **mental map** when merging diagram changes!

Ohst, D., Welle, M. & Kelter, U. (2003). *Differences Between Versions of UML Diagrams*. Proc. of the 9th European Software Engineering Conference, ACM, 227-236.

Mehra, A., Grundy J.C., & Hosking, J.G. (2005). *A Generic Approach to Supporting Diagram Differencing and Merging for Collaborative Design*. Proc. of the 20th International Conference on Automated Software Engineering, ACM, 204-213.

Future Challenges

Avoiding Conflicts

- Pessimistic versioning and Synchronous modeling not always desirable

→ Avoid conflicts through awareness
    → Notification when modelers work on the same model fragment(!)

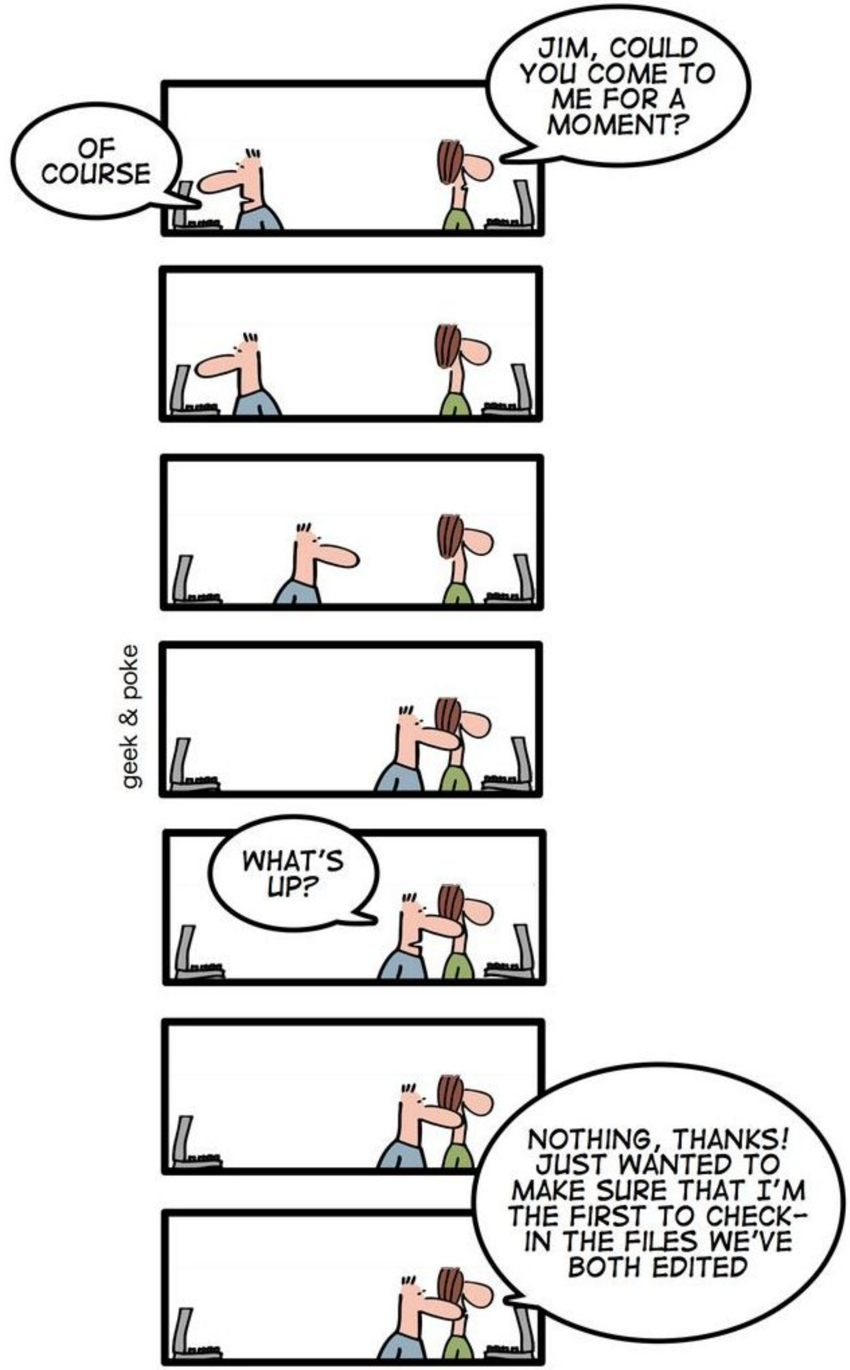
→ Model partitioning
    → How to separate a model?
    → Find appropriate mechanisms for separation of concerns techniques

Future Challenges

Model/Code Versioning

- Roundtrip Engineering
  - Synchronizing models and code
  - Problem of inconsistencies when changing models and code in parallel
  - Especially when changes are not on the same granularity level
- Using models for versioning large code repositories
  - In case several conflicts between different versions occur

Estublier, J., Leveque, T., & Vega, G. (2010). *Evolution Control in MDE Projects: Controlling Model and Code Co-evolution*. Proc. of the Conference on Fundamentals of Software Engineering (FASE), Springer, 431-438.


JIM, COULD YOU COME TO ME FOR A MOMENT?
OF COURSE
WHAT'S UP?
NOTHING, THANKS! JUST WANTED TO MAKE SURE THAT I'M THE FIRST TO CHECK-IN THE FILES WE'VE BOTH EDITED
geek & poke

The AMOR Team

- Gerti Kappel

- Martina Seidl
- Manuel Wimmer


- Petra Brosch
- Philip Langer
- Konrad Wieland



These Slides are Based on the Following Papers (1/2)

- P. Brosch:
  Conflict Resolution in Model Versioning; Reviewer: G. Kappel, A. Pierantonio; E188 Institut für Softwaretechnik und Interaktive Systeme, to appear 2012.
- P. Brosch, P. Langer, M. Seidl, K. Wieland, M. Wimmer, G. Kappel:
  The Past, Present, and Future of Model Versioning; in: "Emerging Technologies for the Evolution and Maintenance of Software Models", IGI Global, 2011, ISBN: 9781613504383, 410 - 443.
- P. Brosch, G. Kappel, M. Seidl, K. Wieland, M. Wimmer, H. Kargl, P. Langer: *Adaptable Model Versioning in Action*. Proc. of the Modellierung, GI, 221-236.
- P. Brosch, H. Kargl, P. Langer, M. Seidl, K. Wieland, M. Wimmer, G. Kappel:
  Conflicts as First-Class Entities: A UML Profile for Model Versioning; in: "Models in Software Engineering - Workshops and Symposia at MODELS 2010, Reports and Revised Selected Papers", Lecture Notes in Computer Science Volume 6627, Springer, 2011, ISBN: 978-3-642-21209-3, 184 - 193
- P. Brosch, P. Langer, M. Seidl, K. Wieland, M. Wimmer, G. Kappel:
  Concurrent Modeling in Early Phases of the Software Development Life Cycle. Proc. of the 16th Collaboration Researchers' International Working Group Conference on Collaboration and Technology (CRIWG), Springer, 129-144.
- G. Kappel, P. Langer, W. Retschitzegger, W. Schwinger, M. Wimmer:
  Model Transformation By-Example: A Survey of the First Wave; in: "Conceptual Modelling and Its Theoretical Foundations", Springer LNCS, Berlin / Heidelberg, 2012, (invited), ISBN: 978-3-642-28278-2, 197 - 215.
- P. Brosch, P. Langer, M. Seidl, K. Wieland, M. Wimmer, G. Kappel, W. Retschitzegger, W. Schwinger:
  An Example Is Worth a Thousand Words: Composite Operation Modeling By-Example; 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS'09), Denver, USA; in: "Proc. of the 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS'09)", Springer, LNCS 5795 (2009), ISBN: 978-3-642-04424-3; 271 - 285.

These Slides are Based on the Following Papers (2/2)

- P. Langer:
  Adaptable Model Versioning based on Model Transformation By Demonstration; Reviewer: G. Kappel, J. Gray; E188 Institut für Softwaretechnik und Interaktive Systeme, 2011.
- P. Langer, M. Wimmer, G. Kappel:
  Model-to-Model Transformations By Demonstration; in: "Proc. of the 3rd International Conference on Model Transformation (ICMT 2010)", Springer, LNCS 6142, 2010, ISBN: 978-3-642-13687-0, 153 - 167.
- P. Langer, K. Wieland, M. Wimmer, J. Cabot:
  From UML Profiles to EMF Profiles and Beyond; 49th International Conference on Objects, Models, Components, Patterns (TOOLS) 2011, Zürich; in: "Proc. of the 49th International Conference on Objects, Models, Components, Patterns (TOOLS) 2011", Springer, LNCS 6705 (2011), 52 - 67.
- B. Meyers, M. Wimmer, A. Cicchetti, J. Sprinkle:
  A generic in-place transformation-based approach to structured model co-evolution; in: "Proc. of the 4th International Workshop on Multi-Paradigm Modeling (MPM'10) @ MoDELS'10", 2010, 13 pages.
- G. Taentzer, C. Ermel, P. Langer, M. Wimmer:
  Conflict Detection for Model Versioning Based on Graph Modifications; in: “Software and Systems Modeling (SoSym)", Springer, to appear in 2012.
- K. Wieland:
  Conflict-tolerant Model Versioning; Reviewer: G. Kappel, G. Fitzpatrich; E188 Institut für Softwaretechnik und Interaktive Systeme, 2011.
- M. Wimmer, A. Kusel, J. Schönböck, W. Retschitzegger, W. Schwinger, G. Kappel:
  On using Inplace Transformations for Model Co-evolution; in: "Proc. of the 2nd International Workshop on Model Transformation with ATL (MtATL 2010)", INRIA & Ecole des Mines de Nantes, 2010, 14 pages.

References and Further Reading (1/4)

- Alanen, M., & Porres, I. (2003). *Difference and Union of Models*. Proc. of the Conference on the Unified Modeling Language (UML), volume 2863 of LNCS, Springer, 2-17.
- Altmanninger K., Seidl M., & Wimmer, M. (2009). A Survey on Model Versioning Approaches. *International Journal of Web Information Systems*, 5(3), 271-304.
- Apiwattanapong, T., Orso, A., & Harrold, M.J. (2007). JDiff: A differencing technique and tool for object-oriented programs. *Automated Software Engineering*, 14(1), 3-36.
- Balzer, R. (1989). *Tolerating Inconsistency*. Proc. of the 5th Int. Software Process Workshop (ISPW), IEEE Computer Society, 41-42.
- Barrett, S., Chalin, P., & Butler, G. (2008). *Model Merging Falls Short of Software Engineering Needs*. Proc. of the 2nd Workshop on Model-Driven Software Evolution @ MoDELS'08.
- Chikofsky, E., & Cross, J.H. (1990). Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, 7, 13-17.
- Cicchetti, A., Ruscio, D.D., & Pierantonio, A. (2008). *Managing Model Conflicts in Distributed Development*. Proc. of the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS), volume 5301 of LNCS, Springer, 311-325.
- Conradi, R., & Westfechtel, B. (1998). Version Models for Software Configuration Management. *ACM Computing Surveys*, 30(2), 232-282.
- De Lucia A., Fasano F., Oliveto R., & Tortora G. (2006). *ADAMS: ADvanced Artefact Management System*. Proc. of the 10th European Conference on Software Maintenance and Reengineering, IEEE, 349-350.
- Dewan, P., & Hegde, R. (2007). *Semi-synchronous conflict detection and resolution in asynchronous software development*. Proc. of the 10th European Conference on Computer-Supported Cooperative Work, Springer, 159–178.
- Dewan, P., & Riedl, J. (1993). Toward Computer-Supported Concurrent Software Engineering. *IEEE Computer*, 26(1), 17–27.
- Dig, D., Manzoor, K., Johnson, R., & Nguyen, T.N. (2007). *Refactoring-aware Configuration Management for Object-oriented Programs*. Proc. of the 29th International Conference on Software Engineering (ICSE), IEEE Computer Society, 427-436.
- Dig, D., Manzoor, K., Johnson, R., & Nguyen, T.N. (2008). Effective Software Merging in the Presence of Object-Oriented Refactorings. *IEEE Transactions on Software Engineering*, 34(2), 321-335.

References and Further Reading (2/4)

- Edwards, W.K. (1997). *Flexible Conflict Detection and Management in Collaborative Applications*. Proc. of the ACM Symposium on User Interface Software and Technology, 139-148.
- Estublier, J., Leblang, D., van der Hoek, A., Conradi, R., Clemm, G., Tichy, W., & Wiborg-Weber, D. (2005). Impact of Software Engineering Research on the Practice of Software Configuration Management. *ACM Transactions on Software Engineering and Methodology*, 14(4), 383-430.
- Estublier, J., Leveque, T., & Vega, G. (2010). *Evolution Control in MDE Projects: Controlling Model and Code Co-evolution*. Proc. of the Conference on Fundamentals of Software Engineering (FASE), Springer, 431-438.
- Finkelstein, A., Gabbay, D.M., Hunter, A., Kramer, J., & Nuseibeh, B. (1994). Inconsistency Handling in Multiperspective Specifications. *IEEE Trans. on Software Engineering*, 20(8), 569-578.
- Ghezzi, C., Jazayeri, M., & Mandrioli, D. (2002). *Fundamentals of Software Engineering*. Prentice Hall PTR, 2002.
- Harel, D., & Rumpe, B. (2004). Meaningful Modeling: What's the Semantics of "Semantics"?. *Computer*, 37(10), IEEE, 64-72.
- Herrmannsdoerfer, M., Benz, S., and Juergens, E. (2009). COPE - Automating Coupled Evolution of Metamodels and Models. In Proc. of the 23rd European Conference on Object-Oriented Programming (ECOOP'09), pages 52-76. Springer-Verlag.
- Hunt, J.W., & McIllroy, M.D. (1976). *An Algorithm for Differential File Comparison*. Technical Report 41, AT&T Bell Laboratories Inc.
- Khuller, S., & Raghavachari, B. (1999). Graph and network algorithms. *ACM Computing Surveys*, 28(1), 43-45.
- Kim, M., & Notkin, D. (2006*). Program Element Matching for Multi-version Program Analyses*. Proc. of the 2006 International Workshop on Mining Software Repositories (MSR '06), ACM.
- Koegel M, Herrmannsdoerfer M, Wesendonk O., & Helming J. (2010). *Operation-based Conflict Detection on Models*. Proc. of the International Workshop on Model Comparison in Practice (IWMCP) @ TOOLS'10*.*
- Kolovos, D.S., Ruscio, D.D., Pierantonio, A. & Paige R.F. (2009). *Different Models for Model Matching: An Analysis of Approaches to Support Model Differencing*. Proc. of the International Workshop on Comparison and Versioning of Software Models, IEEE.
- Küster, J., Gerth, C., & Engels, G. (2009). *Dependent and Conflicting Change Operations of Process Models*. Proceeding of the Conference on Model Driven Architecture - Foundations and Applications (MDAFA), volume 5562 of LNCS, Springer, 158-173.
- Lin, Y., Gray, J. & Jouault F. (2007). DSMDiff: A Differentiation Tool for Domain-specific Models. *European Journal on Information Systems*, 6, 349-361.

References and Further Reading (3/4)

- Lippe, E., & Florijn, G. (1991). *Implementation Techniques for Integral Version Management*. Proc. of the European Conference on Object Oriented Programming (ECOOP), 342-359.
- Lippe, E., & van Oosterom, N. (1992). *Operation-based Merging*. Proc. of the 5th ACM SIGSOFT Symposium on Software Development Environments (SDE), ACM, 78-87.
- Lopez-Herrejon, R.E., & Egyed, A. (2010). *Detecting Inconsistencies in Multi-View Models with Variability*. Proc. of the European Conference on Modelling Foundations and Applications (ECMFA), Springer, 217-232.
- Lucia, A., Fasano, F., Scanniello, G. & Tortora, G. (2009). *Concurrent Fine-Grained Versioning of UML Models*. Proc. of the European Conference on Software Maintenance and Reengineering, IEEE, 89-98.
- Maoz, S., Ringert, J.O., & Rumpe, B. (2010). *A Manifesto for Semantic Model Differencing*. Proc. of the International Workshop on Models and Evolution (ME2010) @ MoDELS'10.
- Mehra, A., Grundy J.C., & Hosking, J.G. (2005). *A Generic Approach to Supporting Diagram Differencing and Merging for Collaborative Design*. Proc. of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE), ACM, 204-213.
- Mens, T. (2002). A State-of-the-Art Survey on Software Merging. *IEEE Transactions on Software Engineering*, 28(5), 449-462.
- Mens, T. (2008). Introduction and Roadmap: History and Challenges of Software Evolution. In *Software Evolution*, Springer, 1-11.
- Misue, K., Eades, P., Lai, W., & Sugiyama, K. (1995). Layout Adjustment and the Mental Map. *Journal of Visual Languages & Computing*, 6(2), 183-210.
- Murta, L., Corrêa, C., Prudêncio, J.G., & Werner, C. (2008). *Towards Odyssey-VCS 2: Improvements over a UML-based version control system*. Proc. of the International Workshop on Comparison and Versioning of Software Models (CVSM) @ ICSE'08, ACM, 25–30.
- Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., & Zave, P. (2007). *Matching and Merging of Statecharts Specifications*. Proc. of the 29th International Conference on Software Engineering (ICSE), IEEE, 54-64.
- Nuseibeh, B., Easterbrook, S.M., & Russo, A. (2001). Making Inconsistency Respectable in Software Development. *Journal of Systems and Software,* 58(2), 171-180.
- Oda, T. & Saeki, M. (2005). *Generative Technique of Version Control Systems for Software Diagrams*. Proc. of the 21st IEEE International Conference on Software Maintenance (ICSM), IEEE Computer Society, 515-524.
- Ohst, D., Welle, M. & Kelter, U. (2003). *Differences Between Versions of UML Diagrams*. Proc. of the 9th European Software Engineering Conference, ACM, 227-236.
- Oliveira H., Murta L. & Werner C. (2005). *Odyssey-VCS: A Flexible Version Control System for UML Model Elements*. Proc. of the 12th International Workshop on Software Configuration Management, ACM, 1-16.

References and Further Reading (4/4)

- Reiter, T., Altmanninger, K., Kotsis, G., Schwinger, W., & Bergmayr, A. (2007). *Models in Conflict - Detection of Semantic Conflicts in Model-based Development.* Proc. of the 3rd International Workshop on Model-Driven Enterprise Information Systems (MDEIS) @ ICEIS'07, 29-40.
- Schmidt, D.C. (2006). Guest Editor's Introduction: Model-Driven Engineering. IEEE Computer, 39(2), 25-31.
- Schneider, C., Zündorf, A. & Niere, J. (2004). *CoObRA - A Small Step for Development Tools to Collaborative Environments*. In Proc. of the Workshop on Directions in Software Engineering Environments.
- Schwanke, R.W., & Kaiser, G.E. (1988). *Living With Inconsistency in Large Systems*. Proc. of the Workshop on Software Version and Configuration Control, pp. 98-118.
- Sun, Y., White, J., and Gray, J. (2009). "Model Transformation by Demonstration," International Conference on Model Driven Engineering Languages and Systems (MoDELS), Springer-Verlag LNCS 5795, Denver, CO, October 2009, pp. 712-726.
- Tichy, W.F. (1988). *Tools for software configuration management*. Proc. of the International Workshop on Software Version and Configuration Control, J. F. H. Winkler, Ed., Teubner Verlag, 1–20.
- UML 2.3 Standard (2010), OMG Unified Modeling Language (OMG UML) Superstructure, Version 2.3, http://www.omg.org/spec/UML/2.3/
- Westfechtel, B. (1991). *Structure-Oriented Merging of Revisions of Software Documents*. Proc. of the Third International Workshop Software Configuration Management (SCM), 68-79.
- Westfechtel, B. (2010). *A Formal Approach to Three-Way Merging of EMF Models*. Proc. of the 1st International Workshop on Model Comparison in Practice (IWMCP), ACM, 31-41.