27. Megamodels in One Technical Space

Prof. Dr. U. Aßmann

Technische Universität Dresden

Institut für Software- und Multimediatechnik

http://st.inf.tu-dresden.de/teaching/most

Version 19-1.1, 16.12.19

1) Model-Driven Architecture (MDA)
2) MDA Toolkits
3) Traceability in Model Transformations
4) Direct Model Mappings between Requirements and Tests
5) RoSIMA – a Very Simple MDA

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

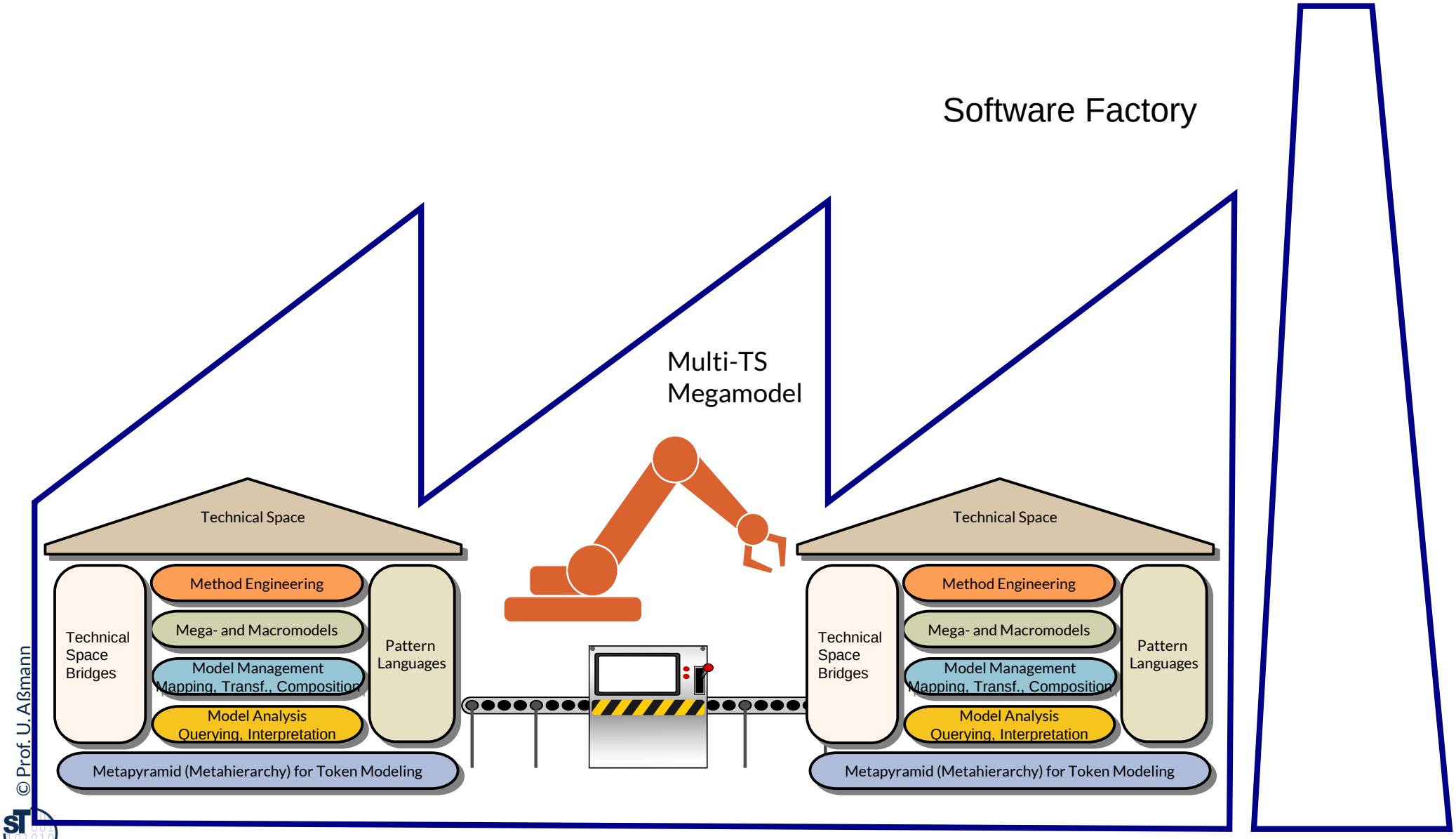
Literature

- [CH06] Krzysztof Czarnecki, Simon Helsen. Feature-based survey of model transformation approaches. IBM Systems Journal 2006. DOI:10.1147/sj.453.0621
- [Hedin09] Görel Hedin. Tutorial: Generating Language Tools with JastAdd
  - http://fileadmin.cs.lth.se/sde/people/gorel/misc/gttse-draft-oct-2009-tutorial.pdf
- Birgit Grammel. Automatic Generation of Trace Links in Model-driven Software Development. PhD thesis, Technische Universität Dresden, Fakultät Informatik, February 2014.
  - http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-155839
- Frédéric Jouault and Ivan Kurtev. On the Architectural Alignment of ATL and QVT. In: Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 06). ACM Press, Dijon, France, chapter Model transformation (MT 2006), pages 1188—1195.
  - http://atlanmod.emn.fr/bibliography/SAC06a
- Tutorial über ATL “Families2Persones”
  - http://www.eclipse.org/m2m/atl/doc/ATLUseCase_Families2Persons.ppt
- ATL Zoo von Beispielen: http://www.eclipse.org/m2m/atl/atlTransformations
- Kevin Lano. Catalogue of Model Transformations: http://www.dcs.kcl.ac.uk/staff/kcl/tcat.pdf
- Implementation in ATL
  - http://www.eclipse.org/m2m/atl/atlTransformations/EquivalenceAttributesAssociations/EquivalenceAttributesAssociations.pdf

Literature on MDA

- https://www.omg.org/mda/products_success.htm
  - https://www.omg.org/mda/mda_files/SuccesStory_DC_TSS_MDO_English.pdf
  - https://www.omg.org/mda/mda_files/SuccessStory_DBB_4pages.pdf
- Alan Brown. An introduction to Model Driven Architecture. Part I: MDA and today's systems
  - http://www.ibm.com/developerworks/rational/library/3100.html
- Petrasch, R., Meimberg, O.: Model Driven Architecture - eine praxisorientierte Einführung in die MDA. Dpunkt-Verlag. 2006
  - Teaser chapter https://www.researchgate.net/publication/220693090_Model_Driven_Architecture_-_eine_praxisorientierte_Einfuhrung_in_die_MDA

Q12: A Software Factory's Heart: the M[illegible]Megamodel



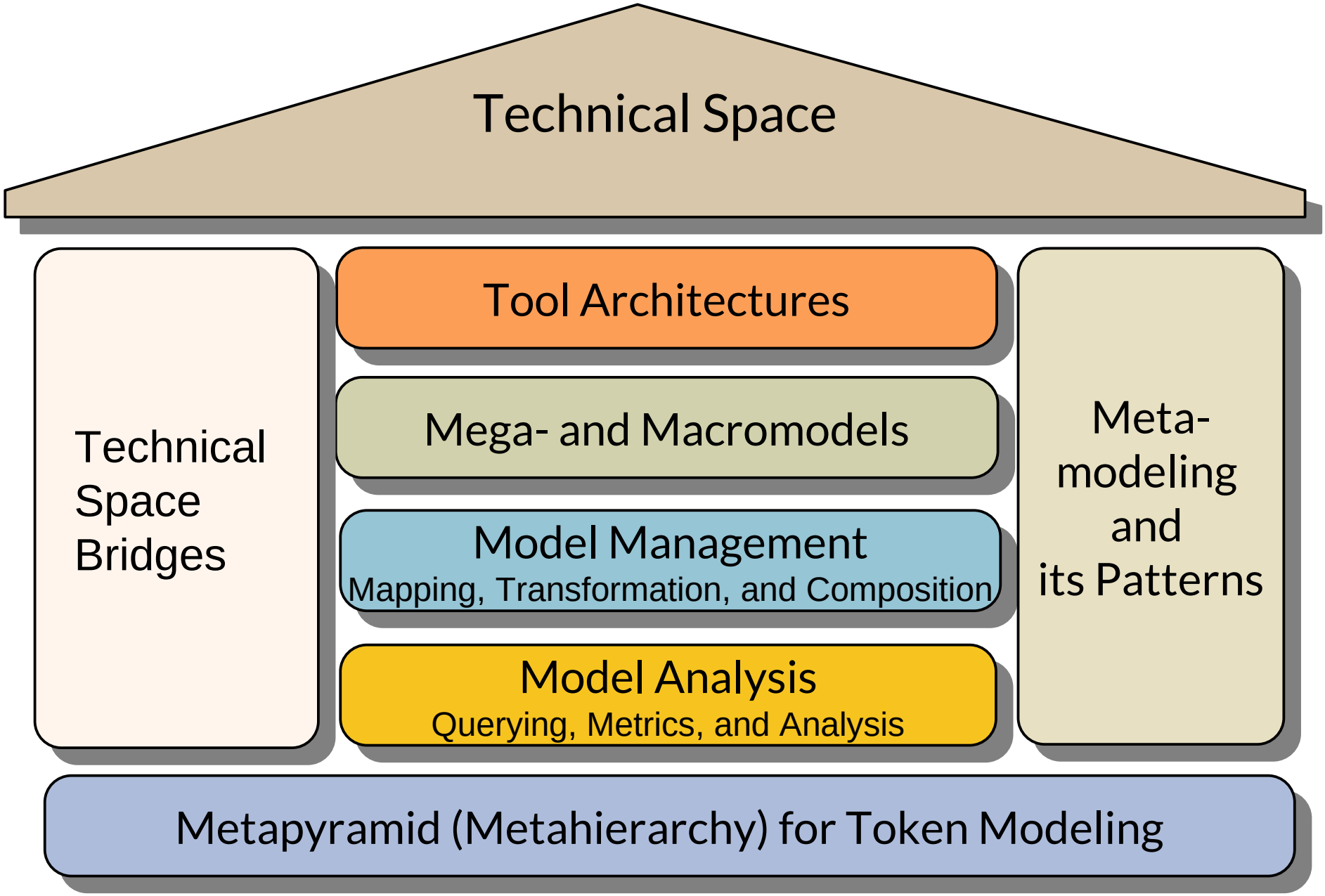
Software Factories

A **software factory** schema essentially defines a recipe for building members of a software product family.

Jack Greenfield

https://www.researchgate.net/publication/213883069_Software_Factories_Assembling_Applications_with_Patterns_Frameworks_Models_and_Tools

Q10: The House of a Technical Space



27.1 Model-Driven Architecture (MDA) (Modellgetriebene Architektur)

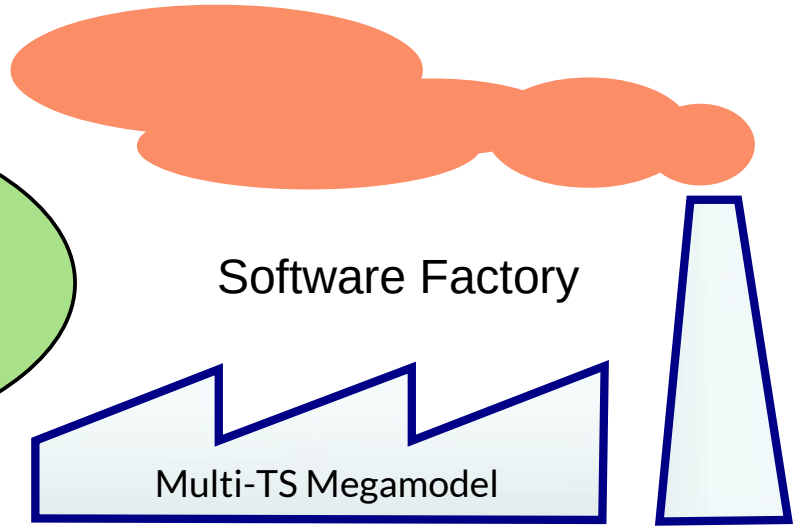
MDA is a trademark of OMG

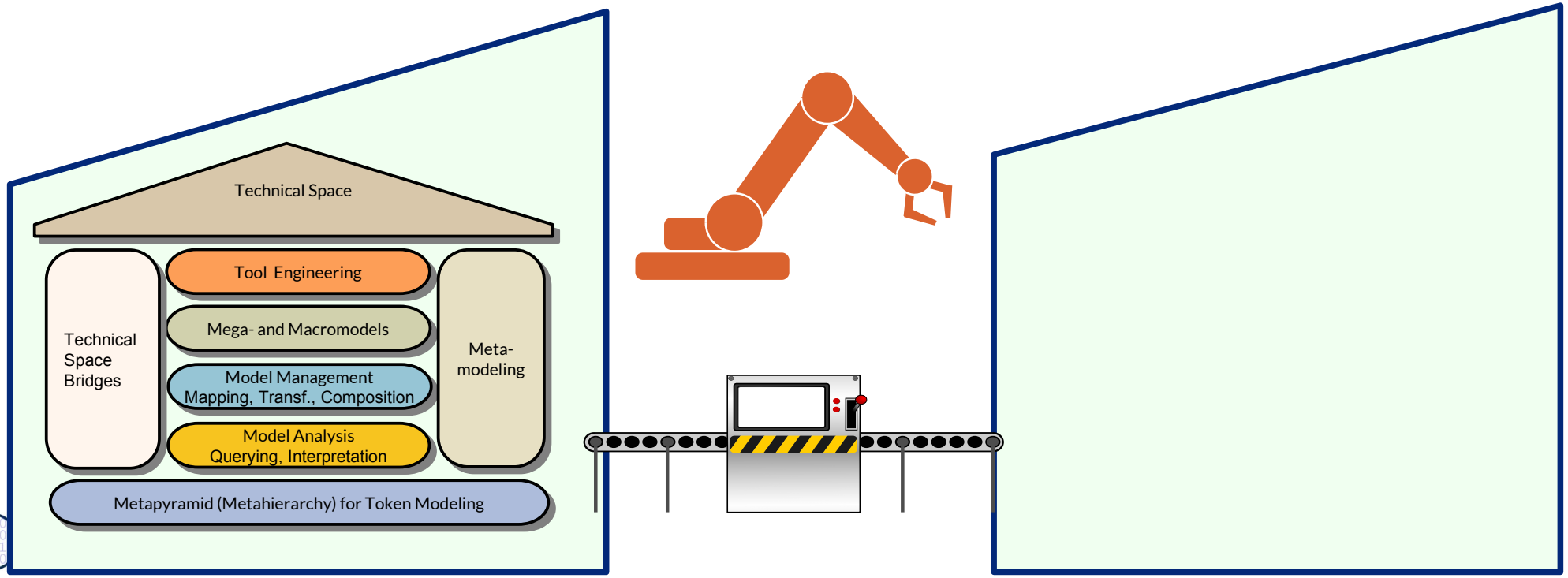
MDA is an industrial megamodel in the spirit of REDECT.

Its instances in software product are macromodels, connecting several ***model abstraction levels.***

Software Factories with Only 1 Technical Space

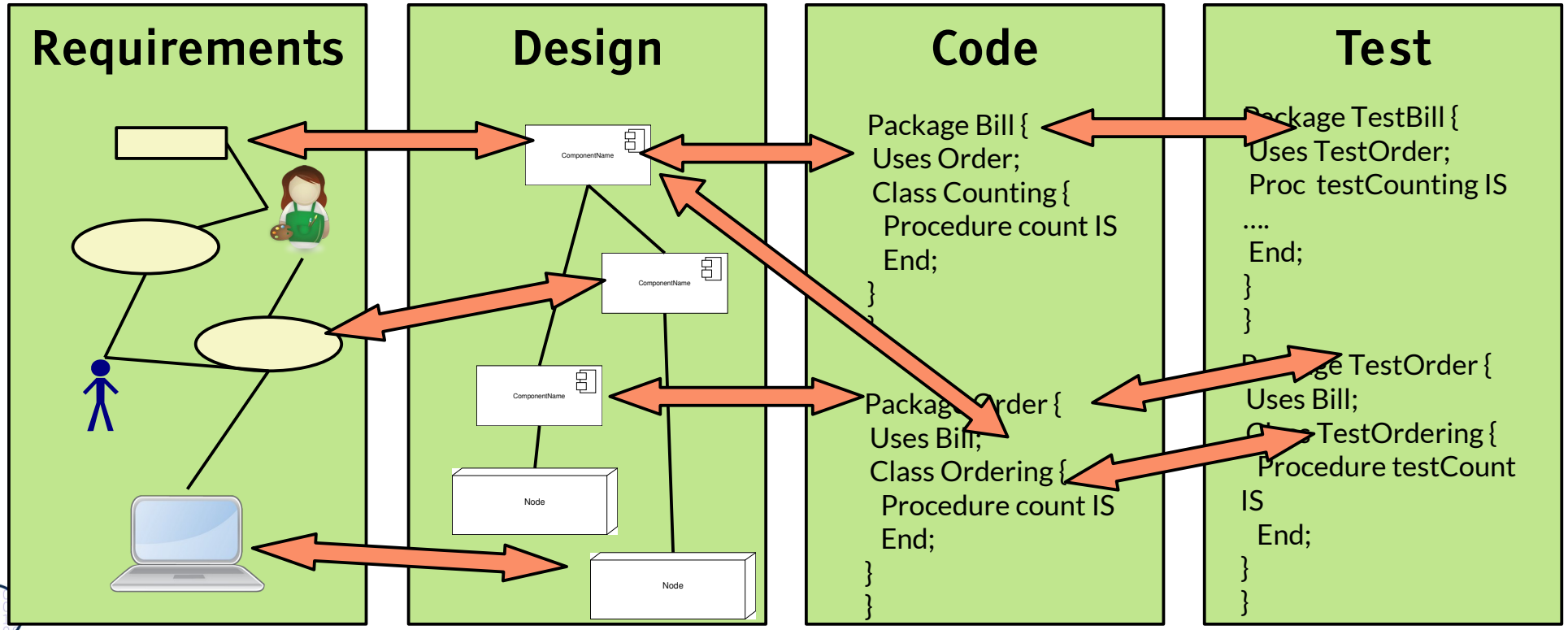
In this chapter:
1-TS Megamodels
MDA, RoSI-MA





Q12: The ReDeCT Problem and its Macromodel

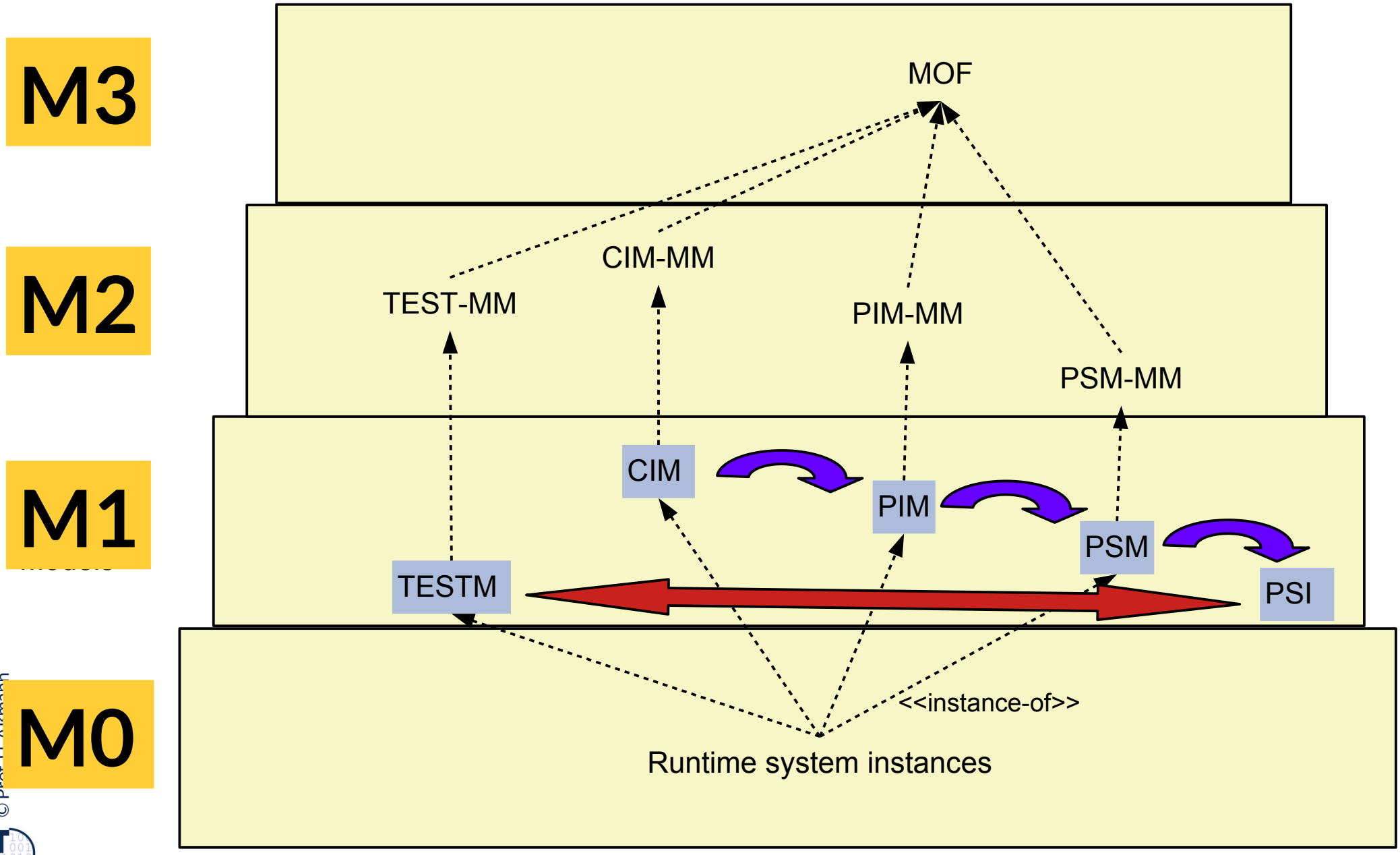
- The inter-model mappings between the Requirements, Design model, Code, Test cases are traceability links stemming for example from:
  - Lifted results of deep model analysis (reachability analysis)
  - Generated trace links from added trace link generators
- A **ReDeCT macromodel** has maintained intermodel mappings between all 4 models



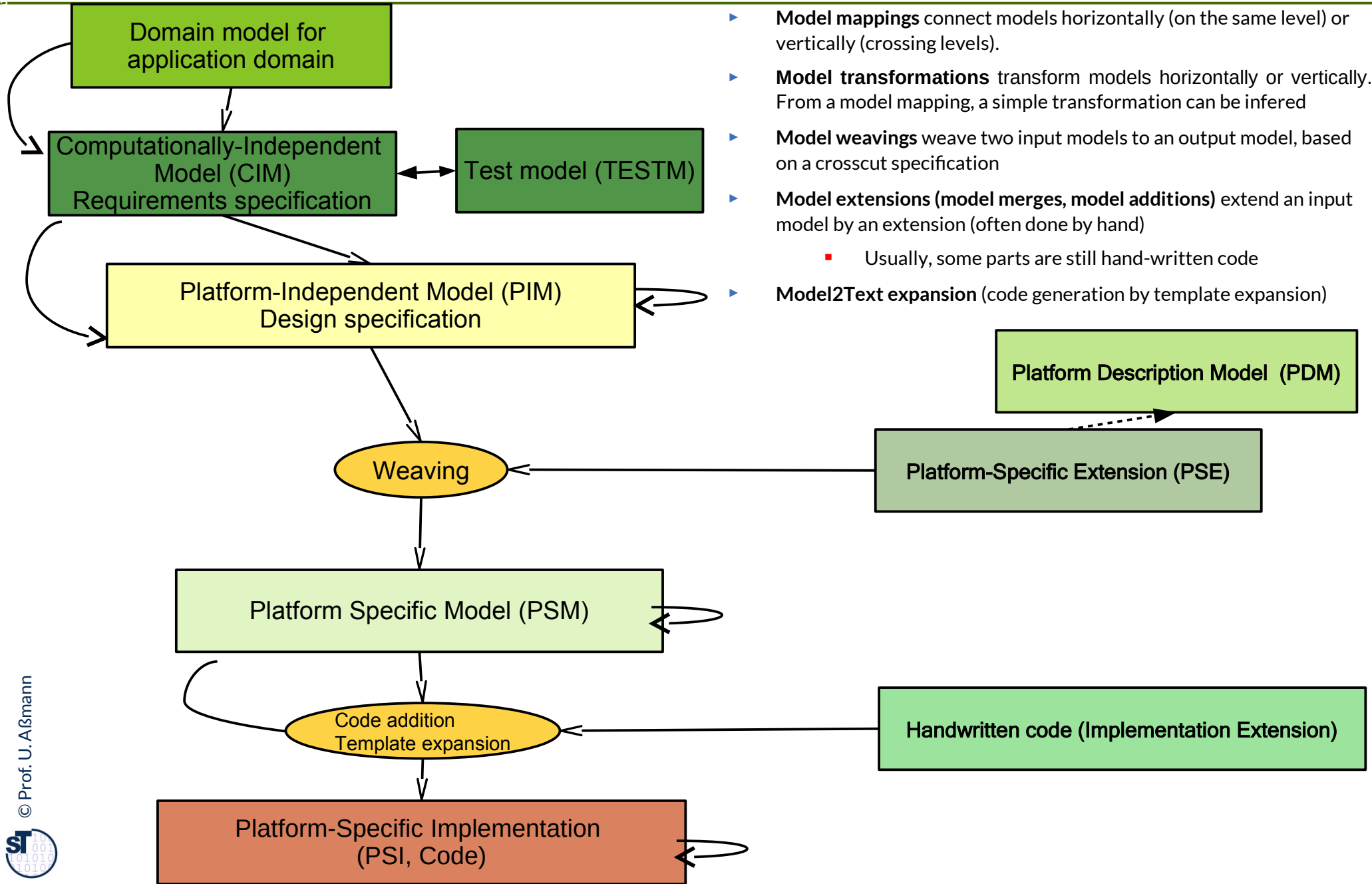
Model-Driven Software Development (MDSD) in 1 Technical Space

- MDSD in 1-TS falls into several main development methods with a macromodels:
  - Engineering with metamodels in ReDeCT-like megamodels (integrated software life-cycle management tools):
    - for integrated requirements, documentation, and testing along the life-cycle
    - Model-Driven Architecture (MDA) (MDA toolkits):
    - For platform-specific variation
  - Engineering with DSL (domain-specific modeling, DSM) (Meta-CASE toolkits)
    - For simplifying the specification of domain-specific software
- **Model mappings** correlate models defining *trace* relations between model elements
  - From them, model transformations can easily be derived
- **Model transformations**
  - **Horizontal model transformations** transform a model within a single language
  - **Vertical model transformations** transform a model from a higher-level language to a lower-lewel language (**lowering**)
  - **Broadband model transformations** transform a model from a higher-level set into a lower-level set of a broadband (wide-spectrum) language
- **Model weavings** extend models by other models

The MDA Embedded in the MOF Metapyramid



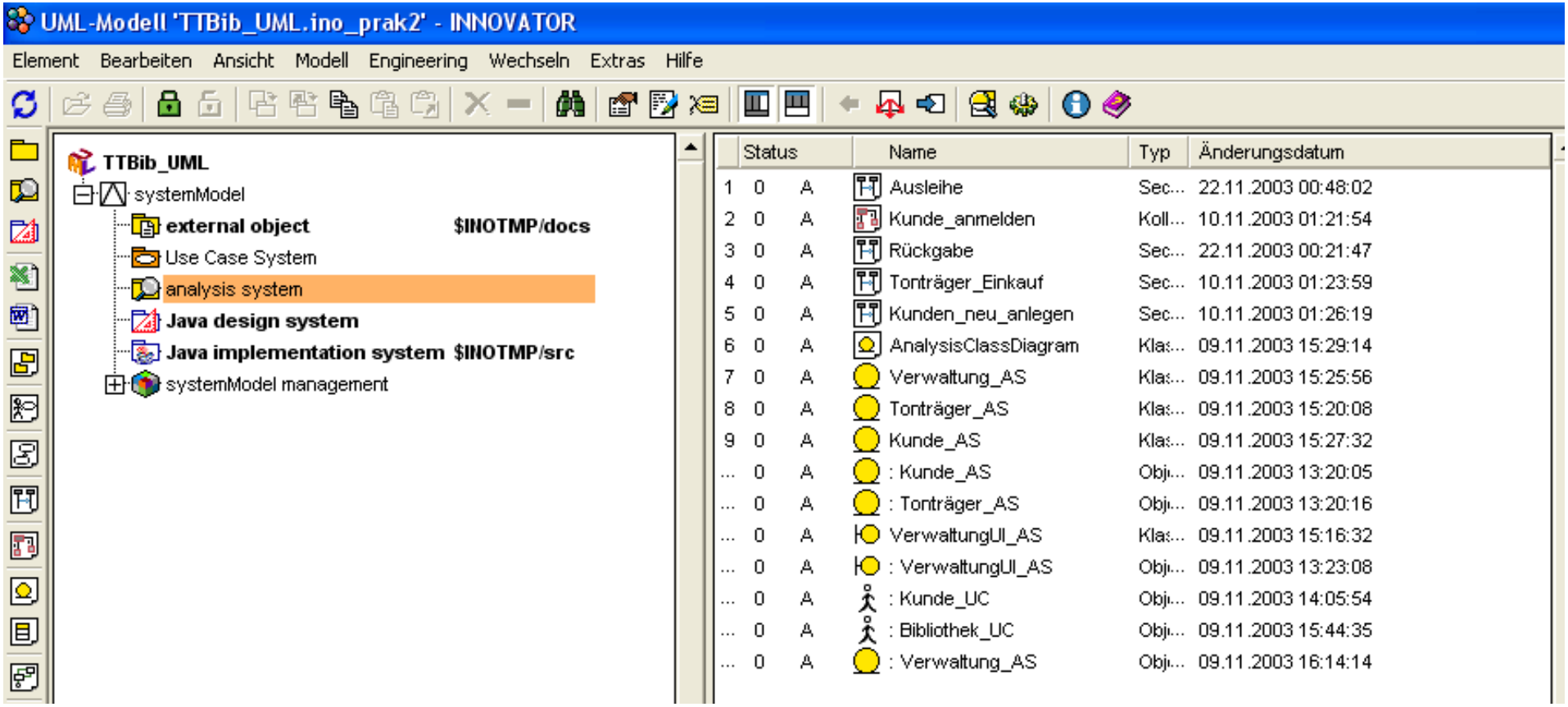
Model Mappings and Model Weavings in the MDA



- **Model mappings** connect models horizontally (on the same level) or vertically (crossing levels).
- **Model transformations** transform models horizontally or vertically. From a model mapping, a simple transformation can be infered
- **Model weavings** weave two input models to an output model, based on a crosscut specification
- **Model extensions (model merges, model additions)** extend an input model by an extension (often done by hand)
  - Usually, some parts are still hand-written code
- **Model2Text expansion** (code generation by template expansion)

PIM and PSM and Model Mapping in MID INNOVATOR

- Innovator can specify transformations between its models



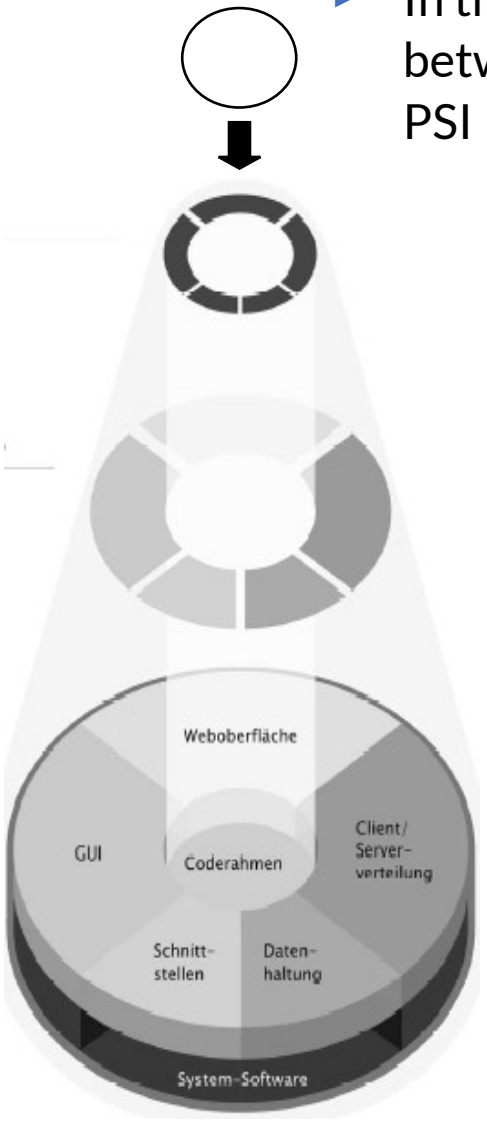
PIM and PSM Extend the CIM

Domain model and requirements model (**CIM**, Computation independent model)

Platform-independent Model (PIM)
Application architecture

Platform-specific Model (PSM)
Specific applicaiton parts
Communication

Platform-specific Implementation (PSI)
Handwriten additions
in programming language

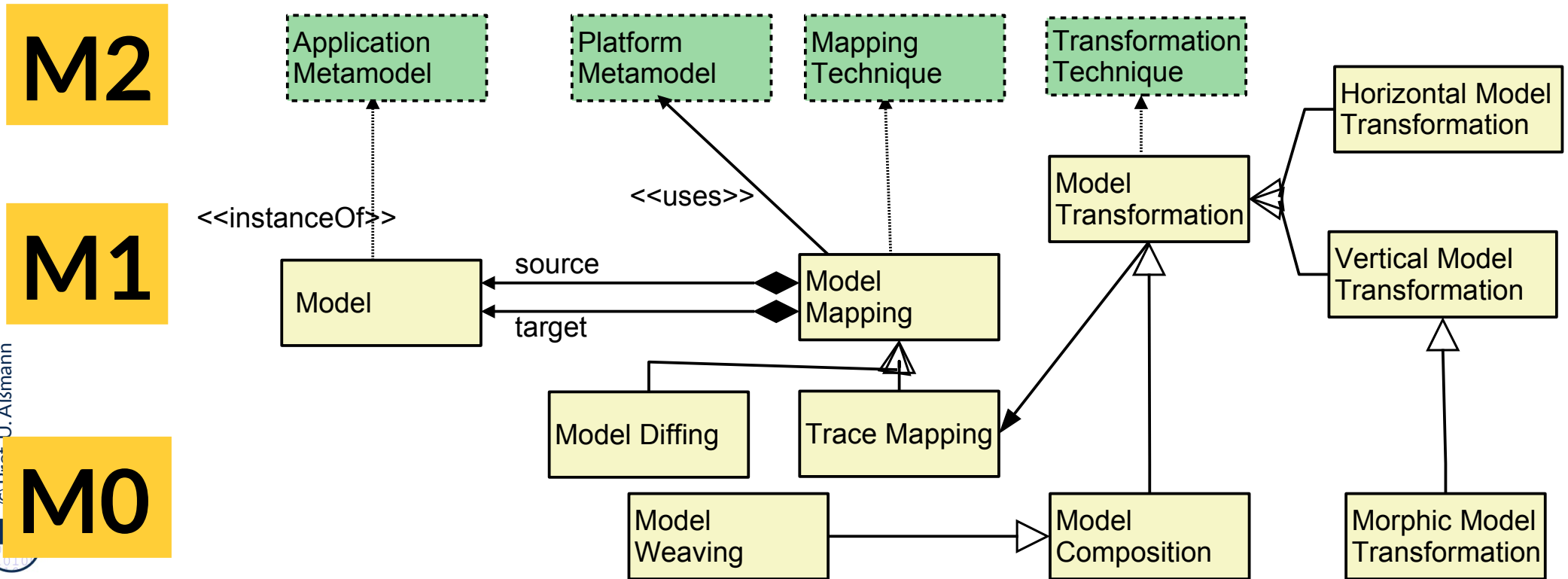


- In the MDA, there are **model mappings** between the models CIM – PIM – PSM - PSI

Quelle: Warum JANUS MDA und MDA JANUS ist; Whitepaper der Firma otris Software AG Dortmund; URL: www.otris.de
http://pi.informatik.uni-siegen.de/stt/15_3/15_3_weg_01.gif

What are Model Mappings?

- Model mappings are automatic or semi-automatic:
  - A model mapping can be generated from a model difference analysis
  - Some are step-wise refinement of the model by transformation (in MDA)
- A model mapping is ***horizontal***, if on the same abstraction level (CIM, PIM, PSM, PSI)
- It is ***vertical***, if abstraction level is crossed (e.g., PIM-2-PSM)
- A ***model transformation*** is a specific model mapping creating a "create trace mapping" with *create links*
- A ***morphic model transformation*** transforms 1 element of a PIM into 1 or n elements on PSM

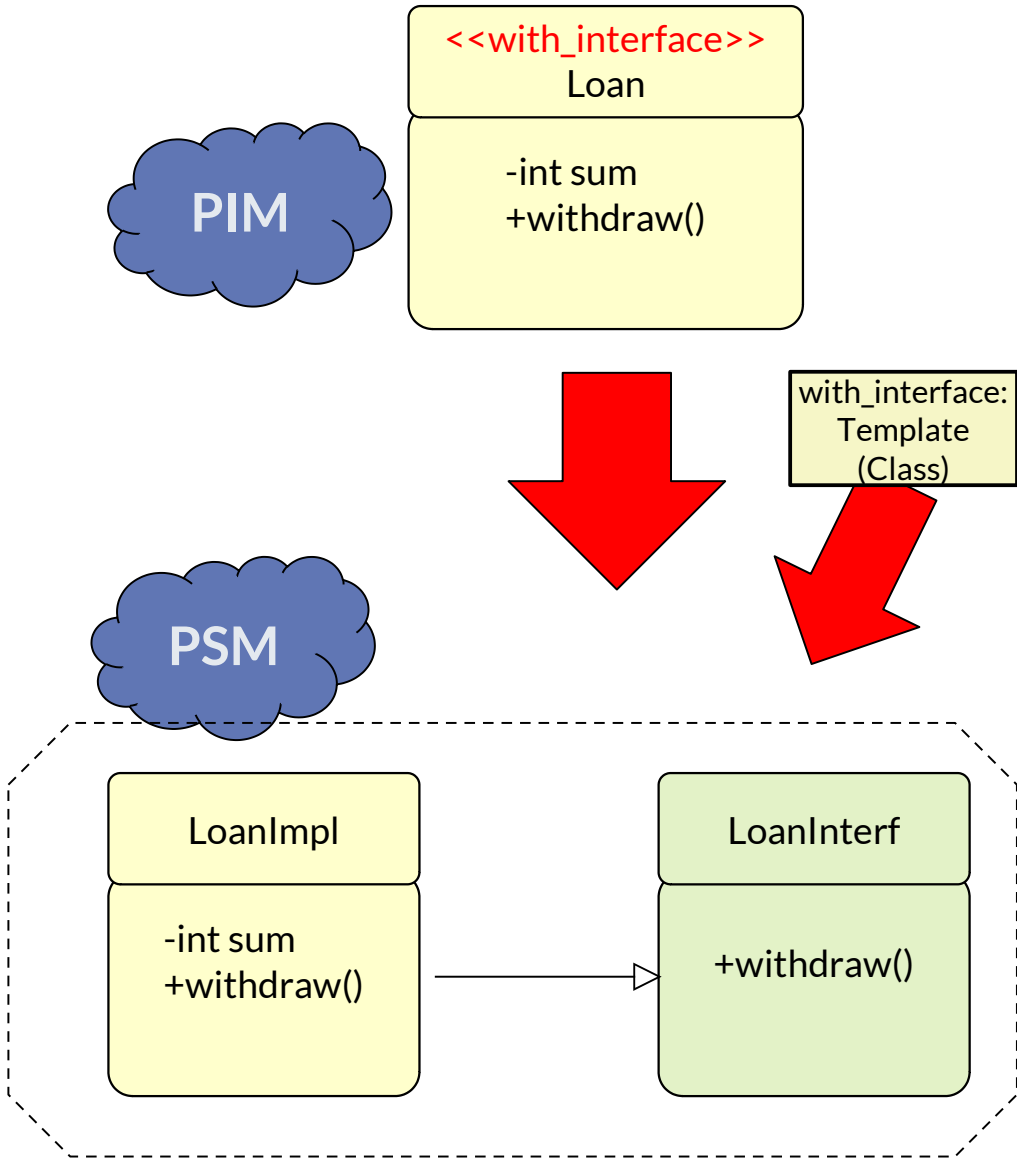


Model Management in Megamodels

- In the MDA megamodel, the mappings between models must be maintained with a model algebra:
  - Model difference analysis (Diff, comm of models)
    - Version management
    - Konfiguration management
  - Model composition
    - Lookup and query of model elements
    - Union, compose, weave, unweave of models
  - Model transformations
    - Transform models

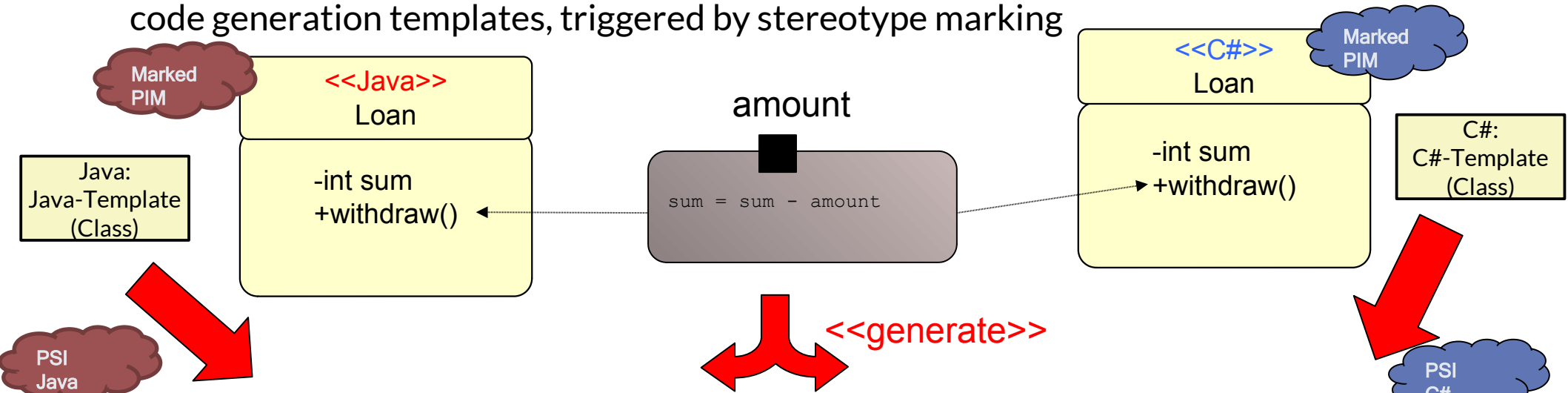
Morphic Mappings on Marked PIMs

- **Morphic mappings (1:1 or 1:n)** are defined by ***marked PIMs***:
  - Stereotypes introduce an exclusively-owns relationship from 1 element of the PIM to n elements in the PSM
  - Supported by many MDA tools, such as AndroMDA
- The stereotype creates a mapping between a PIM class and a set of PSM classes
  - The stereotype tells the MDA system how to transform the PIM class to the PSM (stereotype triggers template extension)
  - The stereotypes partition the PSM: The border of a partition is demarcated by the PIM stereotype tag
- Example: automatic creation of interfaces for implementation classes



Example of a Marked PIM and the Induced Model Transformations

- Tags (stereotypes) may denote different class implementations in a PSM or PSI
- Here: mapping of a class and activity diagram to different languages, using different code generation templates, triggered by stereotype marking



PSI Java

```
// Java implementation as a decorator
class Loan extends Account {
  // decorator backlink
  Account upper;

  private int sum;
  public void withdraw(
   int amount) {
   sum -= amount;
}
```

PSI C#

```
// C# implementation: a partial class
partial class Loan : Account {
  private int sum;
  public void withdraw(
   int amount) {
   sum -= amount;
}
```

Cartridges are Transformation Libraries for Marked PIMs

- A **Cartridge** defines both the model mapping and the model transformation
  - For vertical and horizontal transformations
  - Manual marking of the PIM
  - Selective transformation of the marked PIM classes
  - Automatic transformation using the mapping and transformations from the cartridge
  - No manual specifications of mappings and transformations necessary

60.1.2 Cartridges in RAGs and JastAdd

RAG Modules Compose Extensions into CIM or PIM

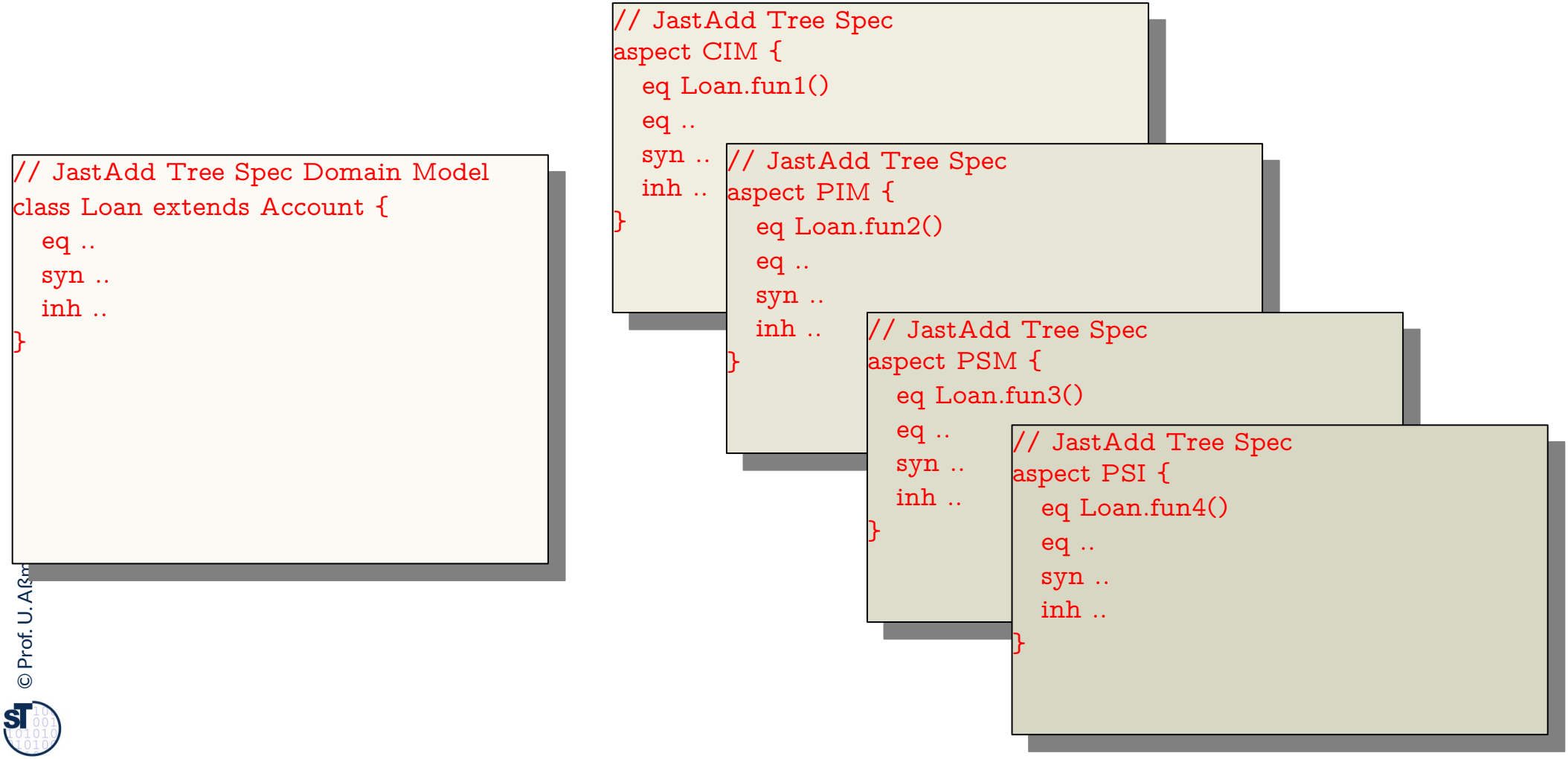
- The basic module can be DM, DM+CIM, DM+CIM+PIM
- Extensions are PSE, PSI
- Due to the declarativeness of attributions, modules can be unified by term (tree unification)
  - Names of the classes serve as unificator

```
// JastAdd Tree Spec
// Domain Model
class Loan extends Account {
   eq ..
   syn ..
   inh ..
}
class Saving extends Account {
  eq ..
  syn ..
  inh ..
}
```

```
// JastAdd Tree Spec for
// Requirements Model (cartridge for CIM)
aspect CIM {
 class  extends Account {
   eq Loan.fun1() = ..
   syn Savings.fun2 () = ..
   inh ..
 }
}
```

Ex.: JastAdd Aspects are Cartridges

- A JastAdd Aspect, like a cartridge, extends a set of Tree Nodes and their attributions with new attributions [Hedin09]
  - *Intertype declarations* distribute a class definition over several files of MDA
  - (Declarative) aspect files are composed by class unification



MDA by Composition of RAG Aspects

- RAG modules, e.g., JastAdd aspects, compose class extensions “around” class names
  - Model weaving is done by class composition
- Model Refinement (in MDA) is done by modular composition (aspect composition)
  - Model synchronisation is done by re-composition
- Model mappings achieved by common class names
  - Tracing is easy (common classes for extensions)

60.2 Industrial MDA Toolkits

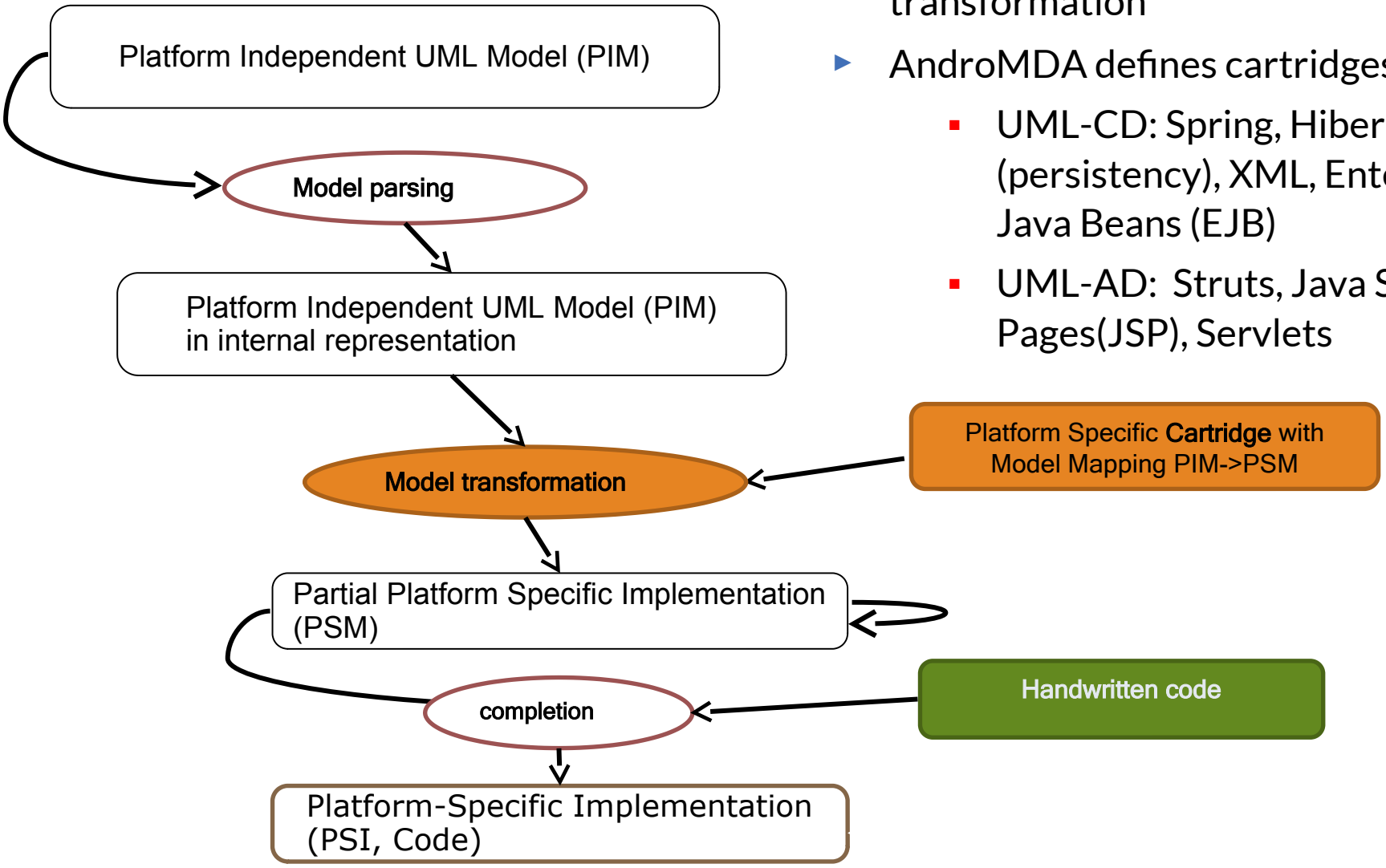
Important Features of MDA Toolkits

- **Model-to-Model Mapping** bzw. **Model-to-Model Transformation** (e.g., PIM to PSM) with cartridges
- **User definition of model transformation cartridges** with query and transformation languages
  - e.g., with QVT, ATL, Graph writing or XML Rewriting
- **Forward- und Reverse-Engineering**
  - Code generation (Model-to-Code Transformation, PSM to PSI)
    - Mapping to a programming language (e.g., with JMI)
- **Roundtrip-Engineering** between models and code
- **Model-driven Testing:** generation of test cases ad test data based on models

[Petrasch, R., Meimberg, O.: Model Driven Architecture - eine praxisorientierte Einführung in die MDA; dpunkt-verlag 2006]

27.2.1 AndroMDA, a Leading MDA Toolkit

- AndroMDA defines model mappings in platform-specific cartridges.



- A cartridge contains a mapping from UML to e.g., Java, C# or C++ and a model transformation
- AndroMDA defines cartridges for
  - UML-CD: Spring, Hibernate (persistency), XML, Enterprise Java Beans (EJB)
  - UML-AD: Struts, Java Server Pages(JSP), Servlets

27.2.2 MDA Toolkit ArcStyler

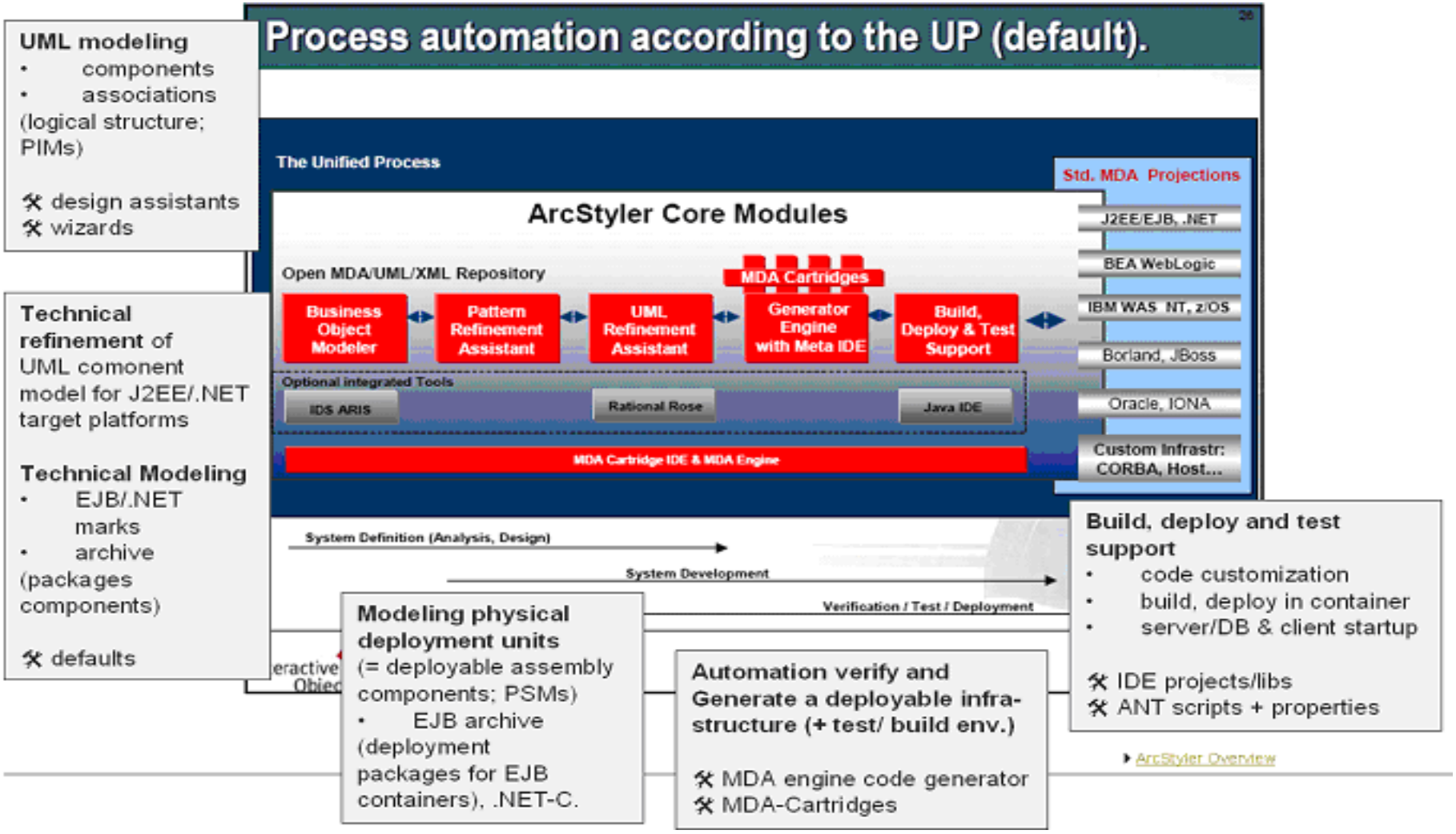
ArcStyler is a toolkit working with several UML-editors such as MagicDraw or Rational Rose

- Cartridges for model mappings and transformations
- **Object Modeler** for requirements modeling; based on CRC-Cards
- **Pattern Refinement Assistent** transforms the domain model interactively into a PIM UML-model (with MagicDraw or Rational Rose)
  - With annotation of design decisions
- **Refinement of the PIM**
  - Horizontal refinement on PIM level
  - Vertical transformation to PSM or PSI (code generation)
- **Code completion (Codevervollständigung)** and optimization for an application platform
- **Component generation** for user interface
- Generation for build tools
- Generation for database persistency

http://www.software-kompetenz.de/servlet/is/27460/?print=true

Versteegen, G.: Wege aus der Plattformabhängigkeit - Hoffnungsträger Model Driven Architecture; Computerwoche 29(2002) Nr. 5 vom 1. Febr. 2002

Process Engineering with ArcStyler

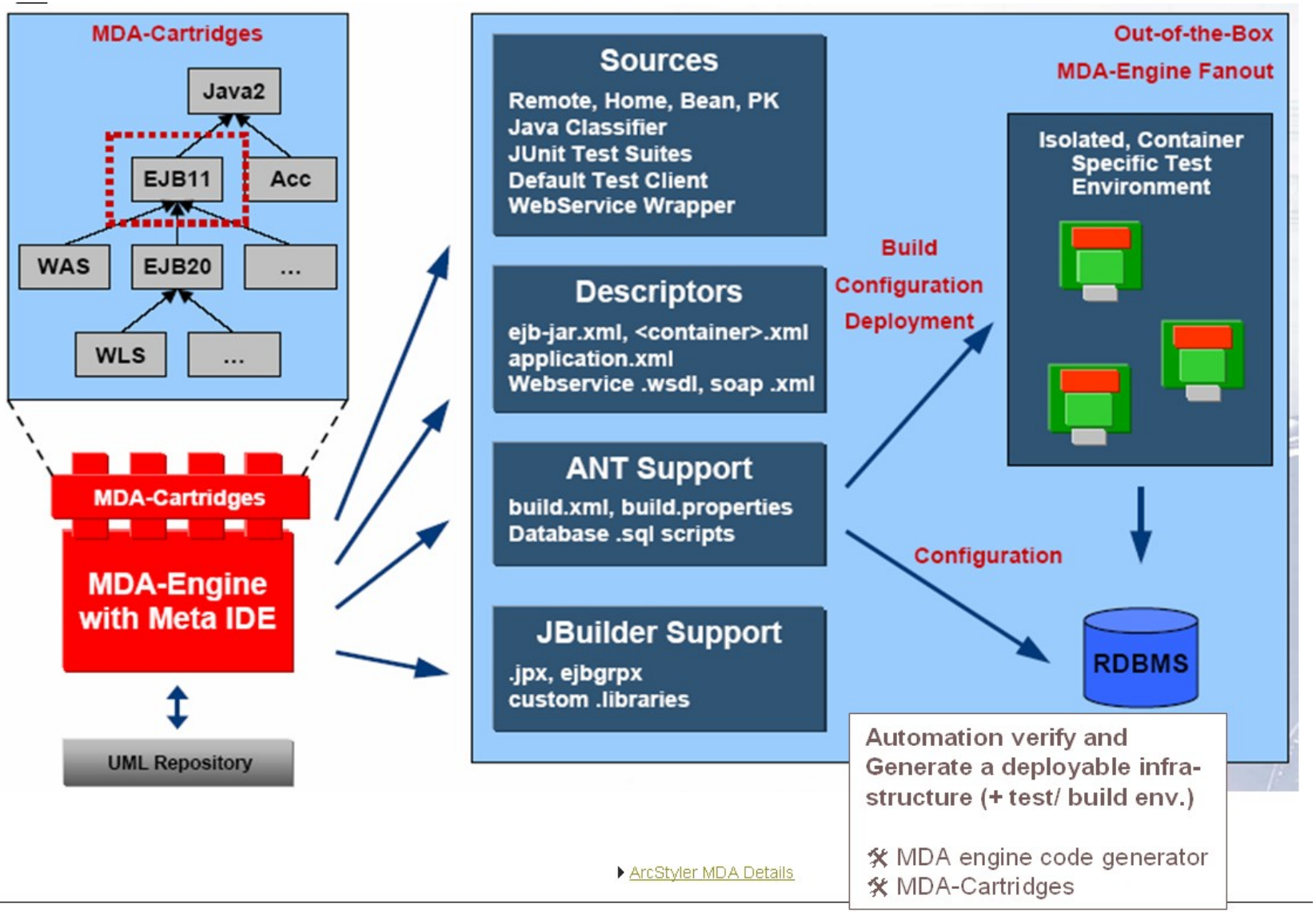


https://www.omg.org/mda/mda_files/P2A_Tutorial.pdf

http://www.interactive-objects.com/products/arcstyler/supportdocumentation.html

http://arcstyler.software.informer.com/

Cartridges and Generated Artifacts



Quelle: Butze, D.: Entwicklung eines Praktikums für die werkzeugestützte Softwareentwicklung nach der Model-Driven-Architecture; Großer Beleg an der Fakultät Informatik der TU Dresden 2004

Some MDA Tools

	Integrated into	URL
AndroMDA	Eclipse	http://www.andromda.org/
XText, Xpand	Eclipse	http://www.eclipse.org/Xtext/
IBM Rational Suite Software Architect	Eclipse	
BITplan smart Generator	Eclipse	http://www.bitplan.com/
Epsilon	Eclipse	https://www.eclipse.org/epsilon/

[Petrasch, R., Meimberg, O.: Model Driven Architecture - eine praxisorientierte Einführung in die MDA; dpunkt-verlag 2006]

27.3 Traceability between Models

- Model transformations generate trace mappings

Why Traceability in a Macromodel?

System Comprehension:

- To improve orientation by navigating via trace links along model transformation chains

- **Change Impact Analysis**:
  - to analyze the impact of a model change on other models
  - to analyze the impact of a model change on existing *generated* or *transformed* output
  - To enable to do model synchronization (hot updating dependent parts)
- **Orphan Analysis:** finding orphaned elements in models

Validation and Verification:

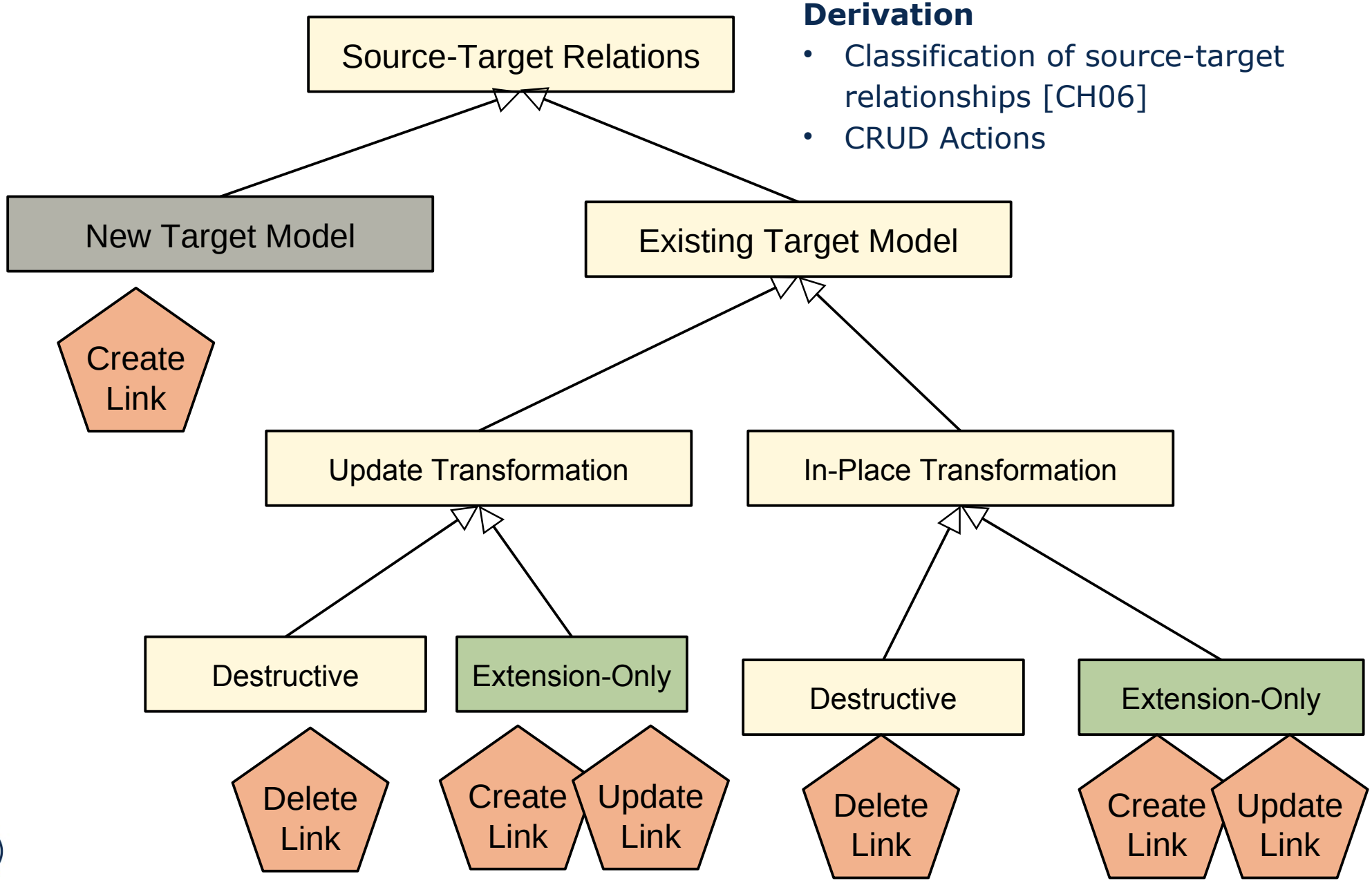
- **System Validation:** Connecting the requirements with the customer's goals and problems (see ZOPP method)
- **(Test) Coverage analysis**: to determine whether all requirements were covered by test cases in the development life cycle
- **Debugging**: To locate bugs when tracing code back to requirements
  - To locate bugs during the development of transformation programs

Traceability Metamodel: CRUD Types of Trace Links between Model Elements of Different Models

[Grammel]

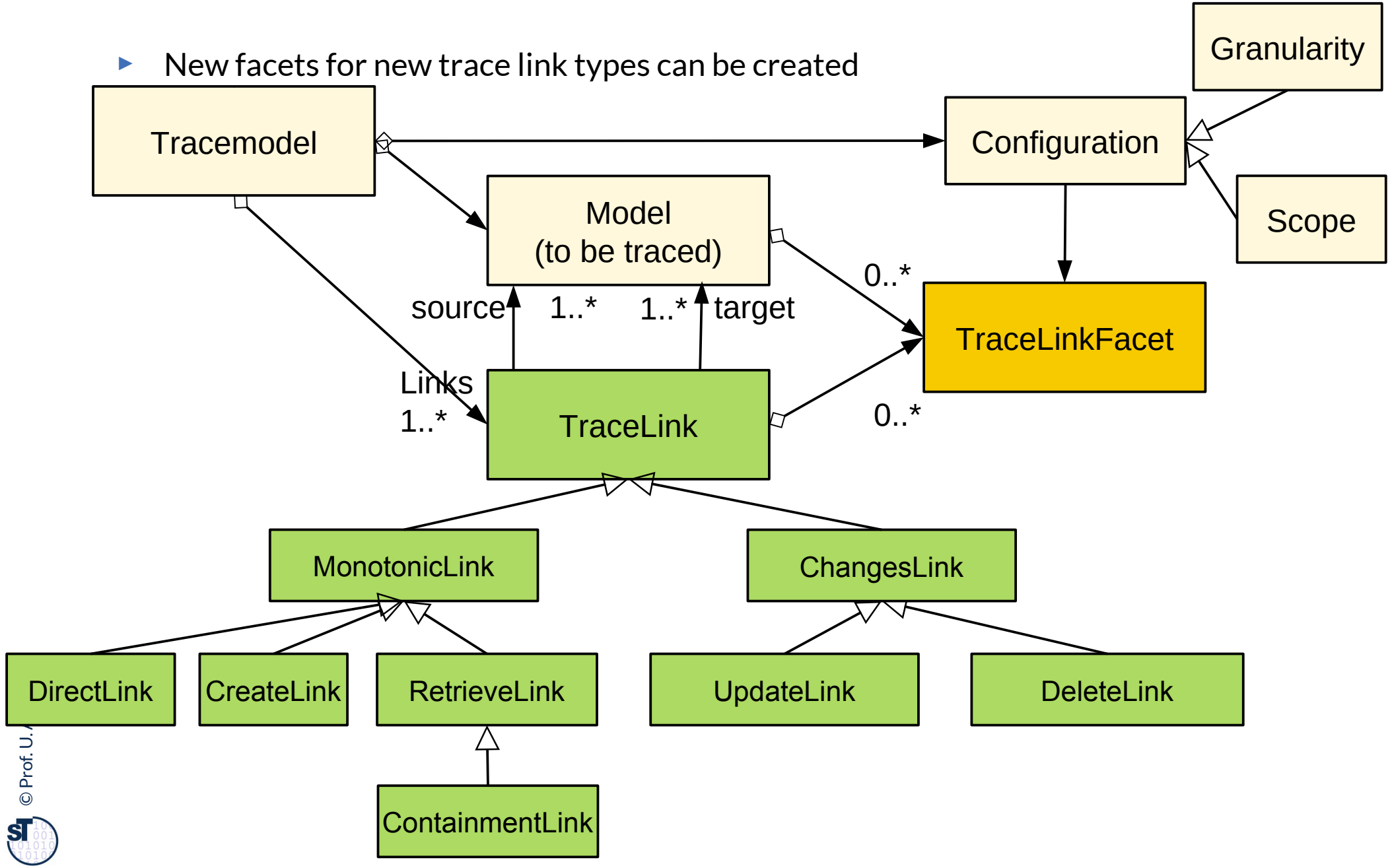
Derivation

- Classification of source-target relationships [CH06]
- CRUD Actions



Extensible Traceability Metamodel acc. to Grammel

- New facets for new trace link types can be created

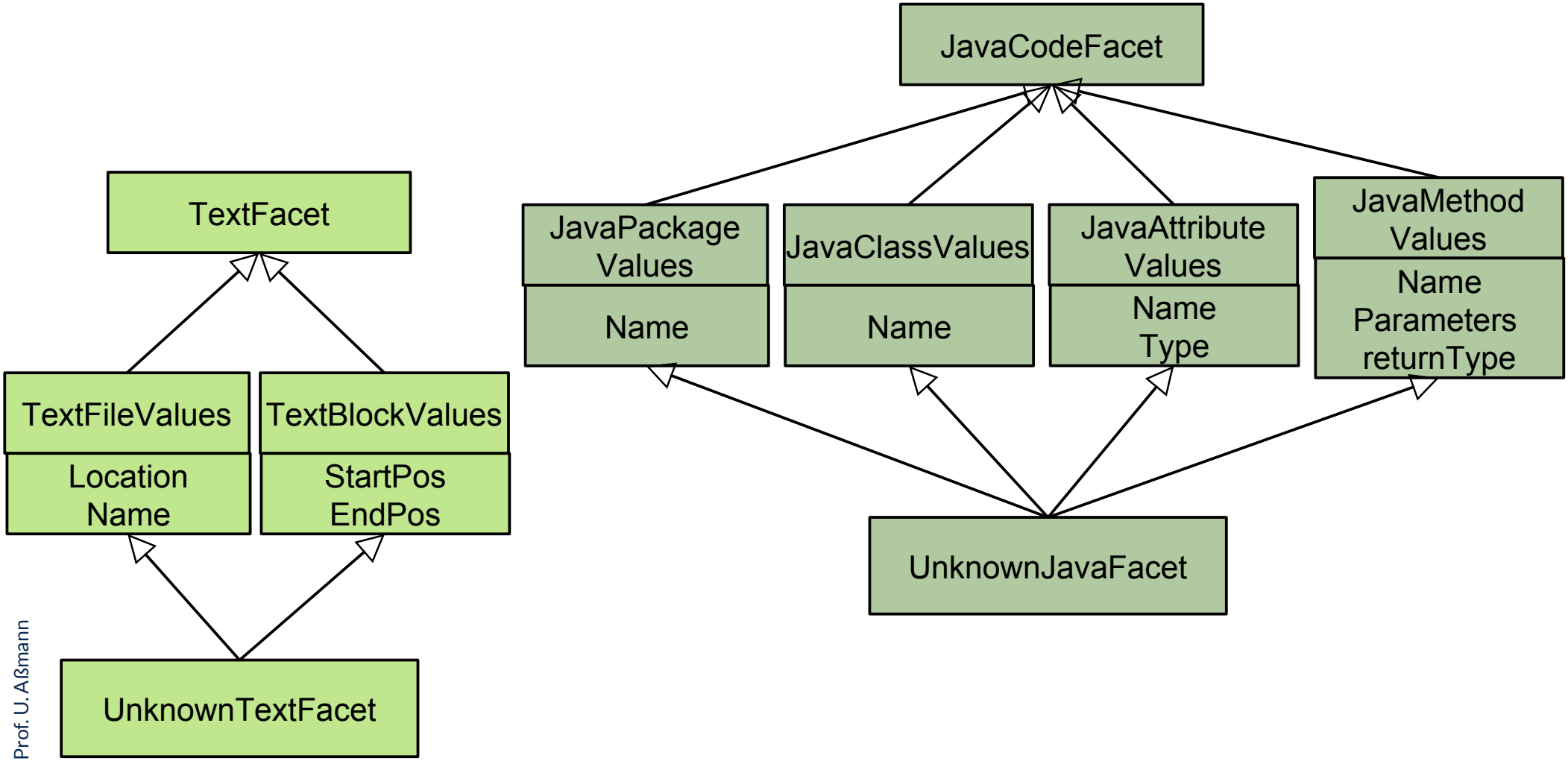


Traceability in Macromodels

- Piecemeal growth of macromodels in the software process:
  - Start with requirements, then add more stuff and models
- **Add links**
  - **Symmetric "Direct" links** are drawn between model element MA from model A and model element MB whenever MB is related to MA
    - Specified by hand
    - Found by a model difference, model analysis or a model query
  - **Create links** are drawn between model element MA from model A and model element MB whenever MB is generated or added because of MA
  - **Retrieve links** are drawn when MB is extracted (queried) from a model A and added to another model B
  - **Containment links** are drawn, when in a new model B the model element MA is contained in another model element MB'
  - **Delete links** are drawn if In model B the model element MB should be deleted
  - **Update links** are drawn if MA has changed and MB should be changed too

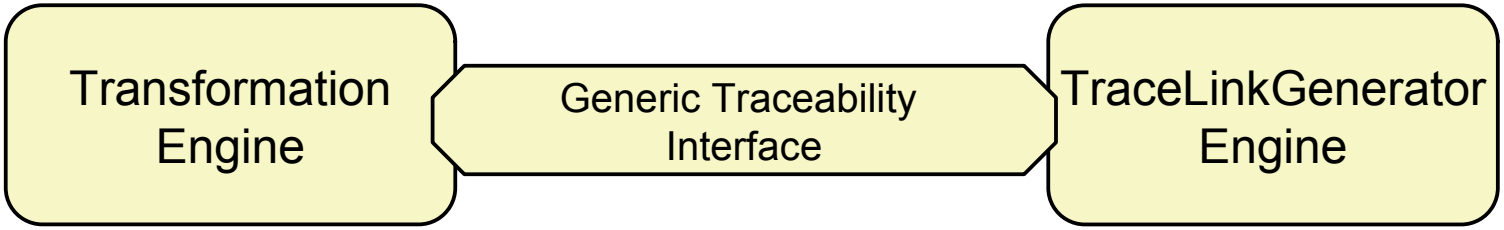
Examples for TraceLinkFacet

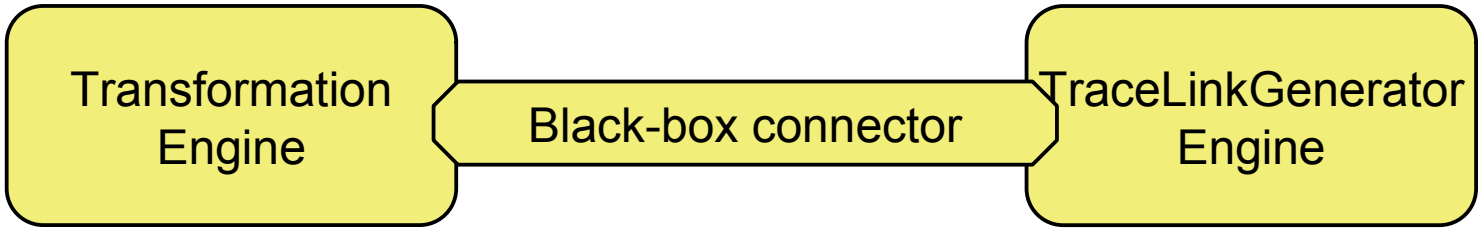
- Facets factorize inheritance hierarchies; new facets extend inheritance hierarchies



Adding a Trace Link Generator to Tools

- TraceLinkGenerators can be connected to transformation engines and cartriges in three ways, following a generic traceability interface:





Transformation engine must know and call the generator

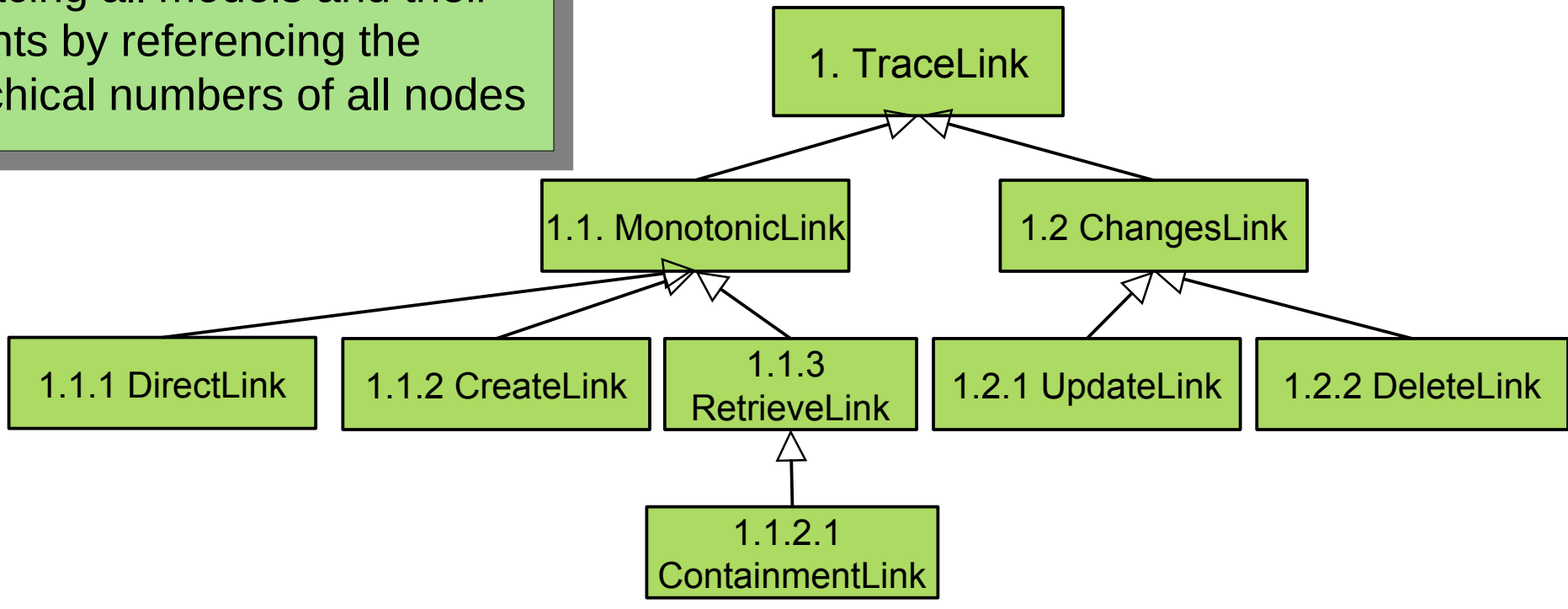
Transformation engine need not know but is extended Invasively or woven By AOP

Traceability in Macromodels with Models from Link-Treeware

- In link-tree models, a skeleton tree exists, in which every model element has a unique *tree node number (hierarchical number)*
- Trace links can be added with tree node number and stored externally of the model *in the macromodel*

In link-treeware, macromodels maintain *tracelink models* linking and tracing all models and their elements by referencing the hierarchical numbers of all nodes

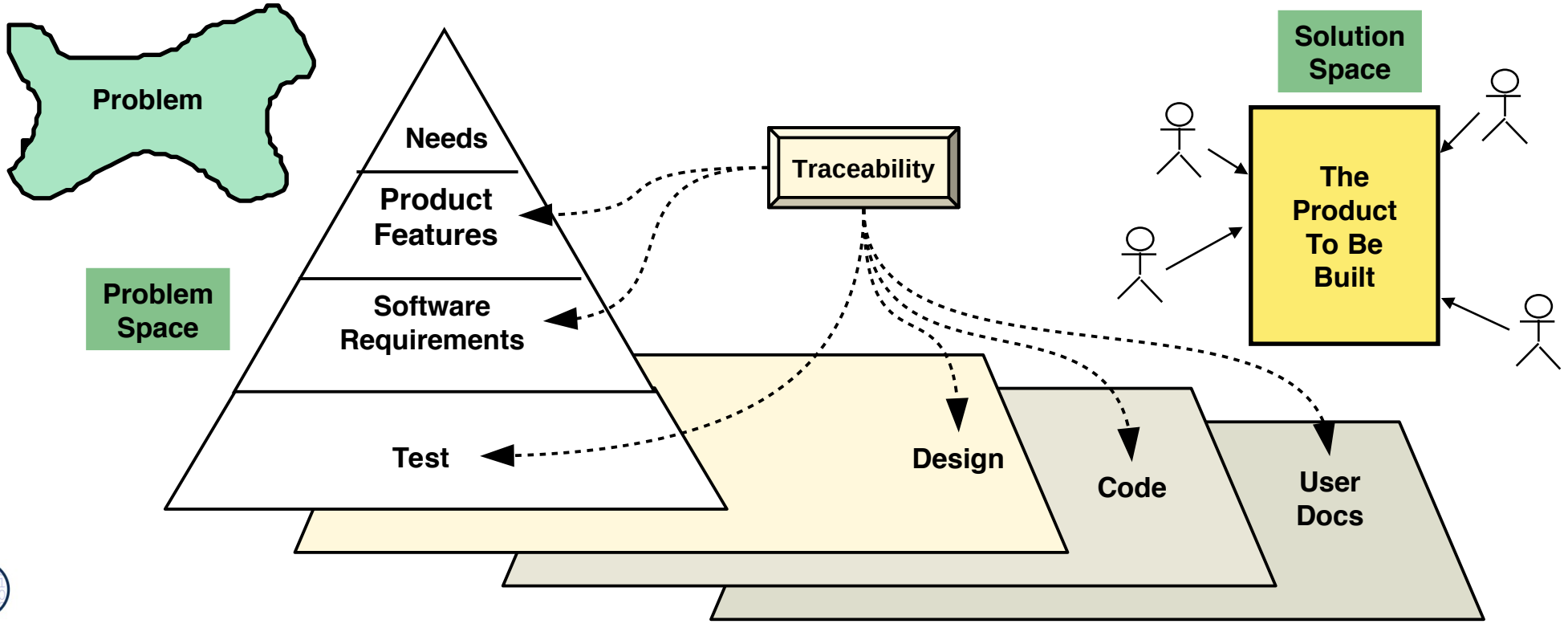
Hierarchical numbering of the classes in an inheritance tree:



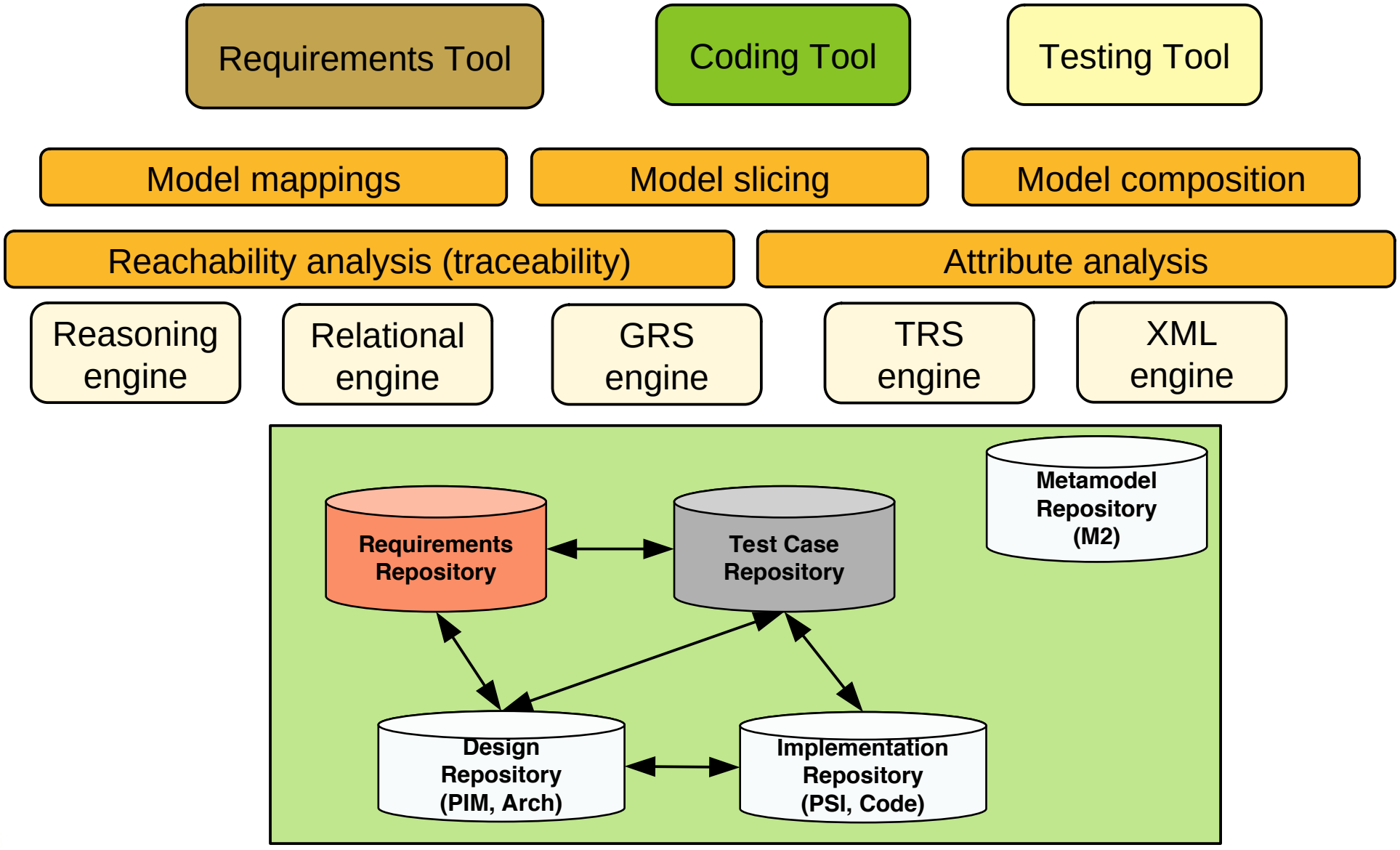
27.4 Traceability in Practical Requirements Management Tools

Introduction to Requirements Management (RM)

- RM bridges the needs of the customer to testing, design, coding, and documentation
- RM continuously manages requirements in the entire software life cycle
- RM relies on inter-model mappings between requirements, test cases, design, and code



Tools in an Integrated Development Environment (IDE)

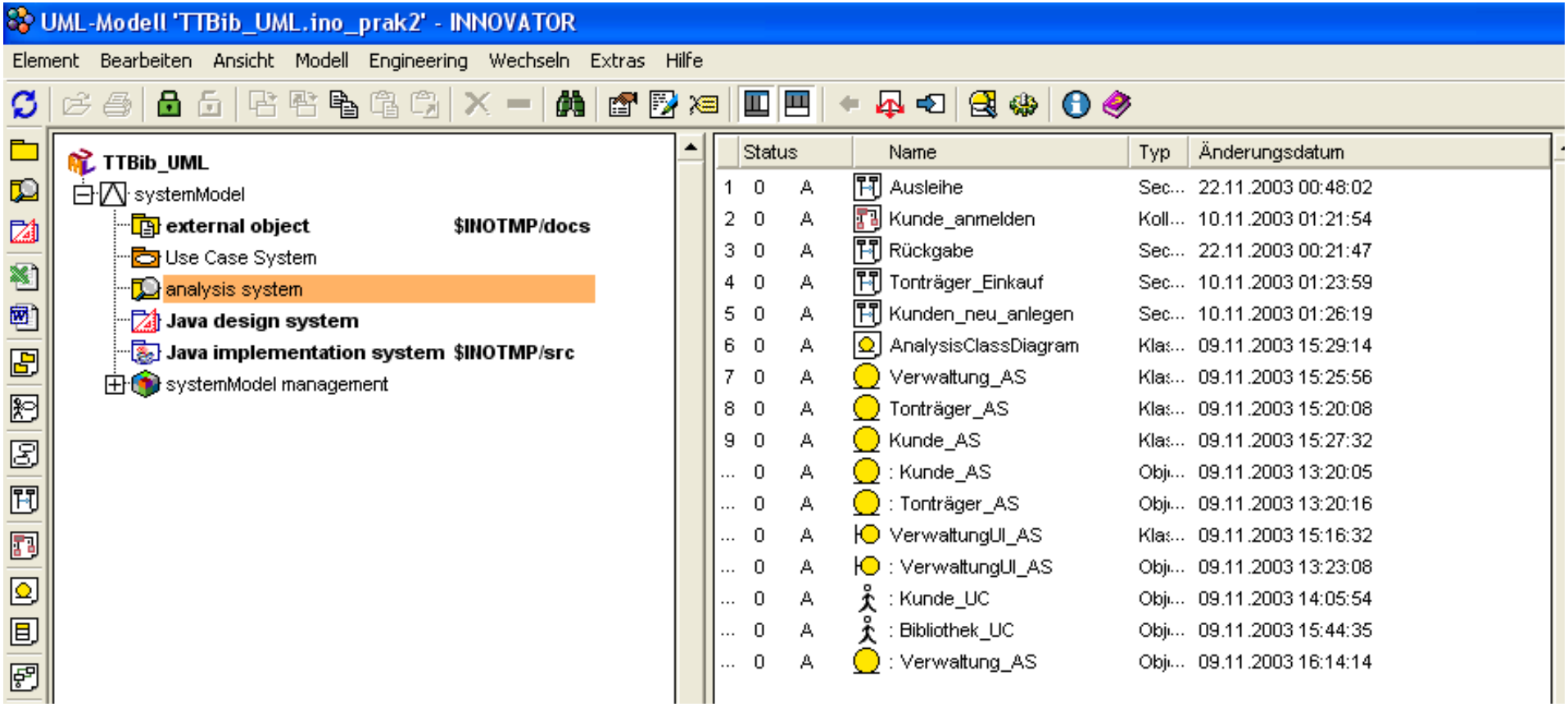


Deficiencies of Current RE Methods

- Relationships among requirements are inadequately captured
  - Causal relationship between consistency, completeness and correctness [Zowghi2002]
  - Completeness and consistency are not verified
- Requirement problems (e.g. conflicts, incompleteness) are detected too late or not all
- Relationships between requirements and dependent artifacts are insufficiently managed (test, documentation, design, code)
- Desirable:
  - Models for RE need richer and higher-level **abstractions** (goals, problems, needs) to validate that they are fulfilled [Mylopoulos1999]
    - Metamodels can be used to define these concepts
    - Ontologies deliver reasoning services
  - **Model mappings (direct and indirect)** between the artifacts (design, code) and the goals, problems, needs of the customer
    - Based on the model mappings, the requirements are consistently managed with design, code, and documentation

Model Mapping in MID INNOVATOR

- Innovator can be employed simultaneously for requirements, design and implementation models
- How to relate these models?



Direct Traceability

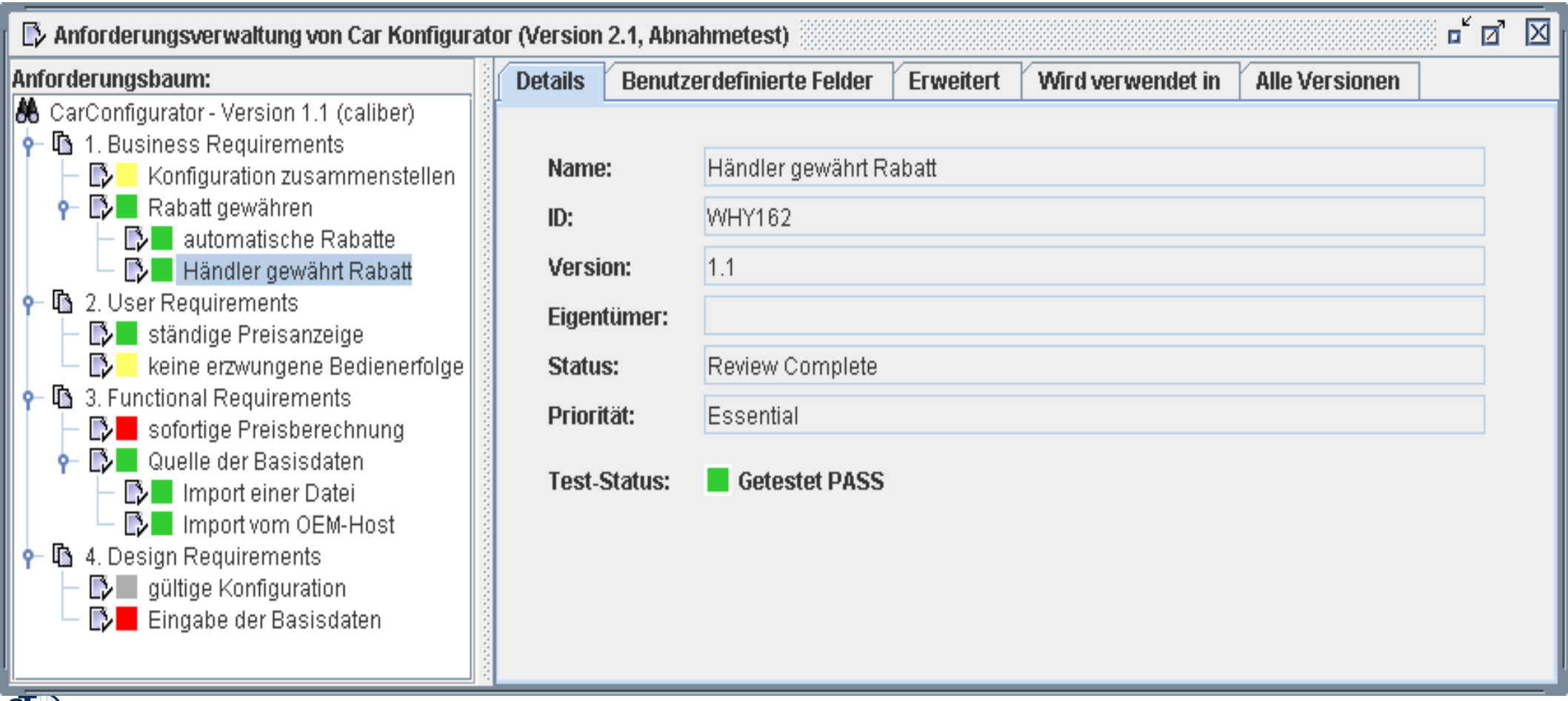
- With a **direct model mapping**, a requirements model can be linked
  - to a test case specification
  - to a documentation
  - to an architectural specification
  - via the architectural specification, to the classes and procedures in the code

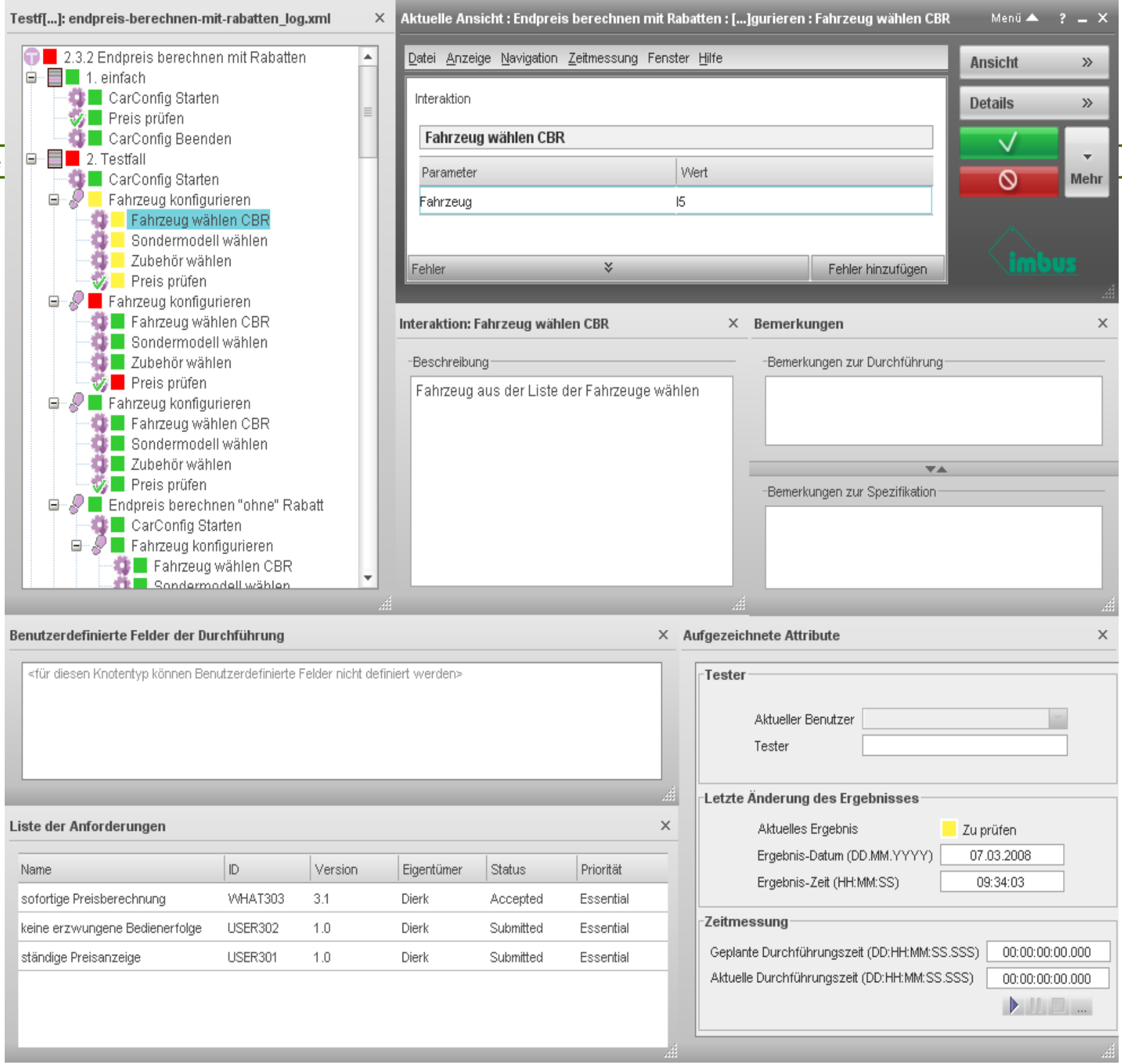
Example: imbus TestBench

http://www.imbus.de/produkte/imbus-testbench/hauptfunktionen/

Requirements get “red-yellow-green” Test Status Attribute

- Test status is an attribute in the requirements tree that contains a **direct link** to the result of a corresponding test case





Direct Model Mappings between Requirements and Test Tools

- Most often, these tools are in Link-treeware (hierarchical requirements, hierarchical test cases and test suites)
- → The trace models can be stored externally in the megamodel
  - Every trace link refers to link-tree node numbers in the requirements and test specifications

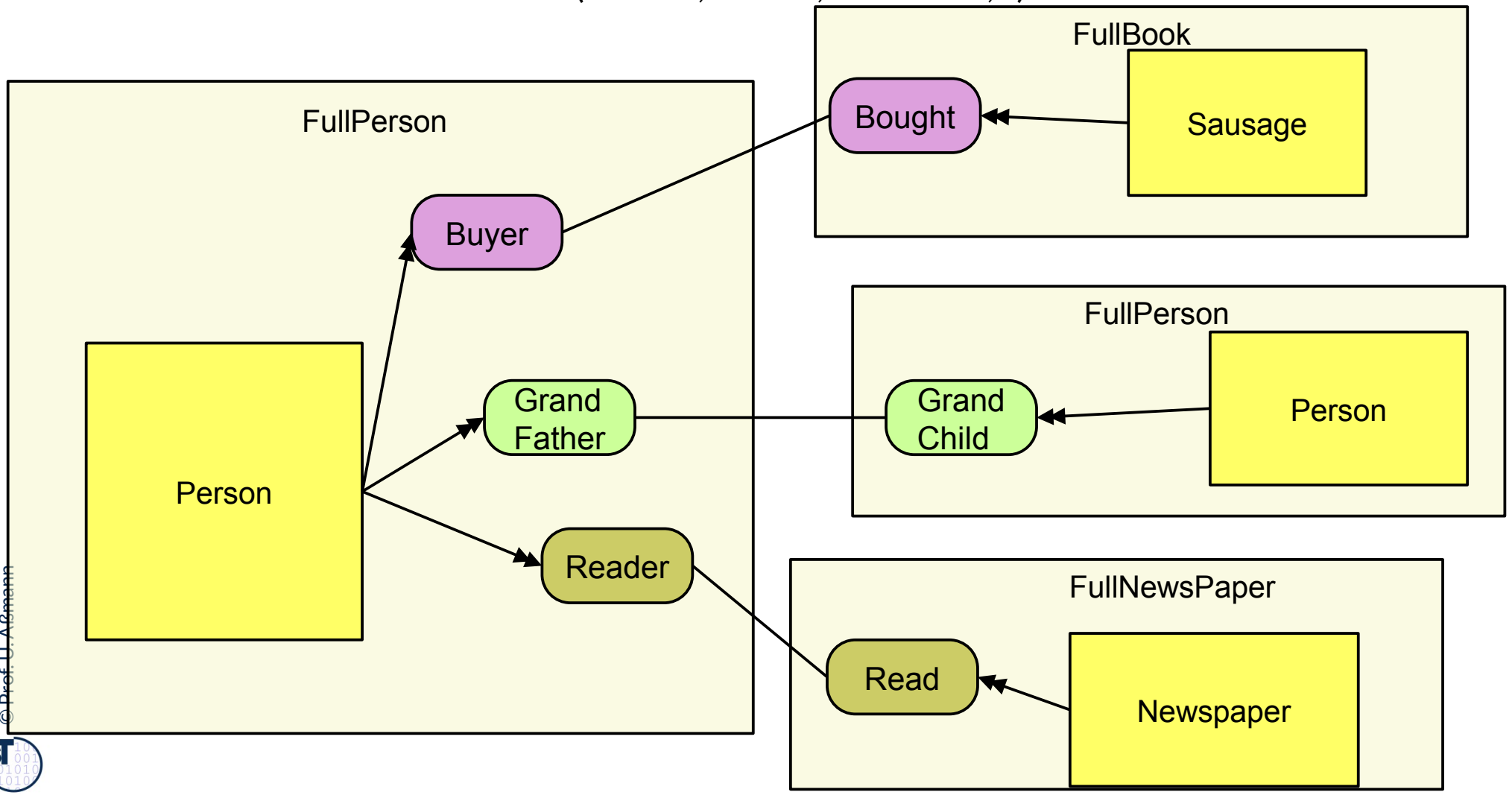
27.5 The Megamodel of RoSI: RoSIMa

- What happens if contexts and roles are available in models?
- The Megamodel of RoSI and its traceability of model elements is extremely simple, because the role-based models and metamodels are factorizing objects

Remember: The Steimann Factorization of Natural and Role Types

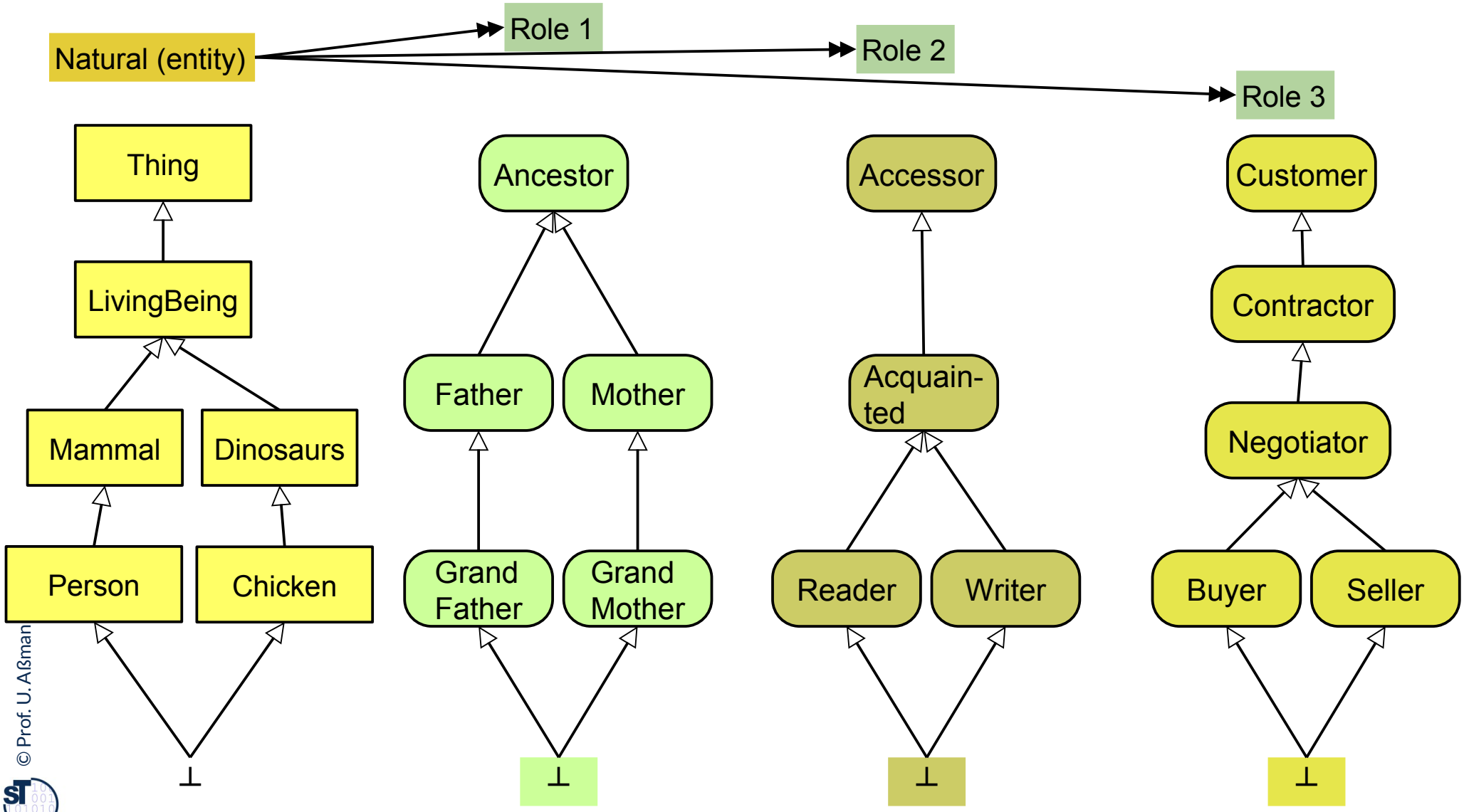
Splitting a full type into its *natural* and *role-type* components

- FullType = Natural x (role-type, role-type, ...)
- FullPerson = Person x (Reader, Father, Customer, ..)



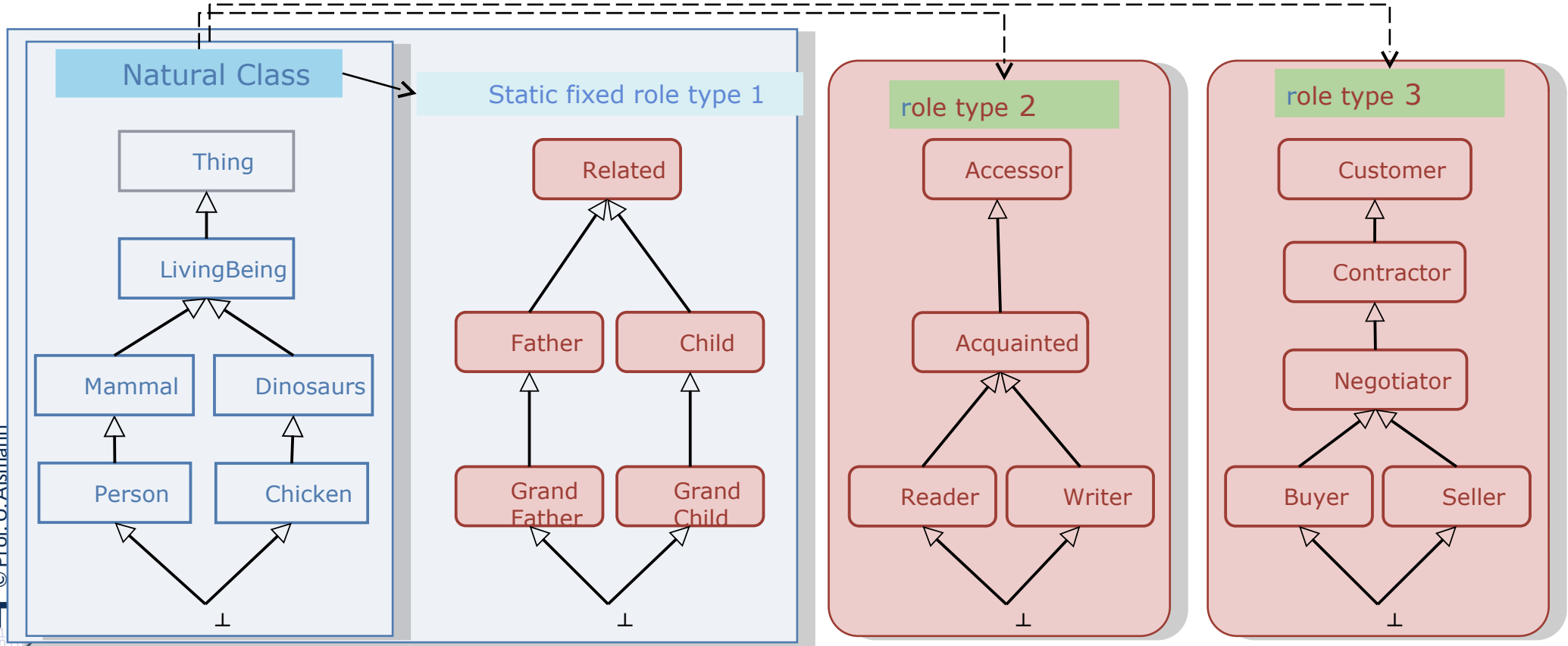
Full Type is from Inheritance Product Lattice

Q: What is a reading buying grandfather person? (A: tuple type)



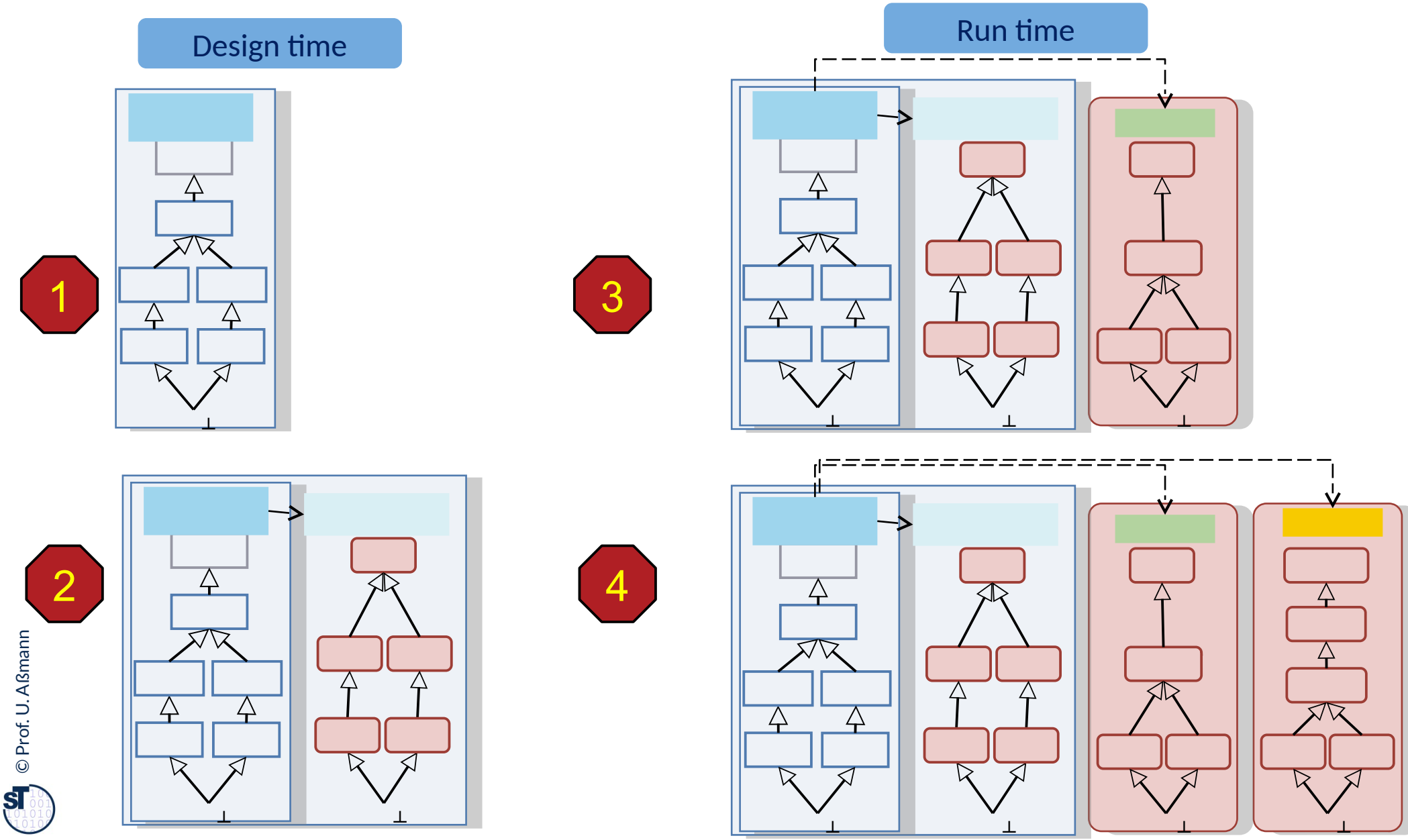
Scalable Bindung Time of Contexts with the Factorization

- **Scalable Binding**: Roles can also be bound statically, if mixins are used as implementation (fixing the context)
- Consequences for object life time, cohesion, allocation, adaptation, reconfiguration



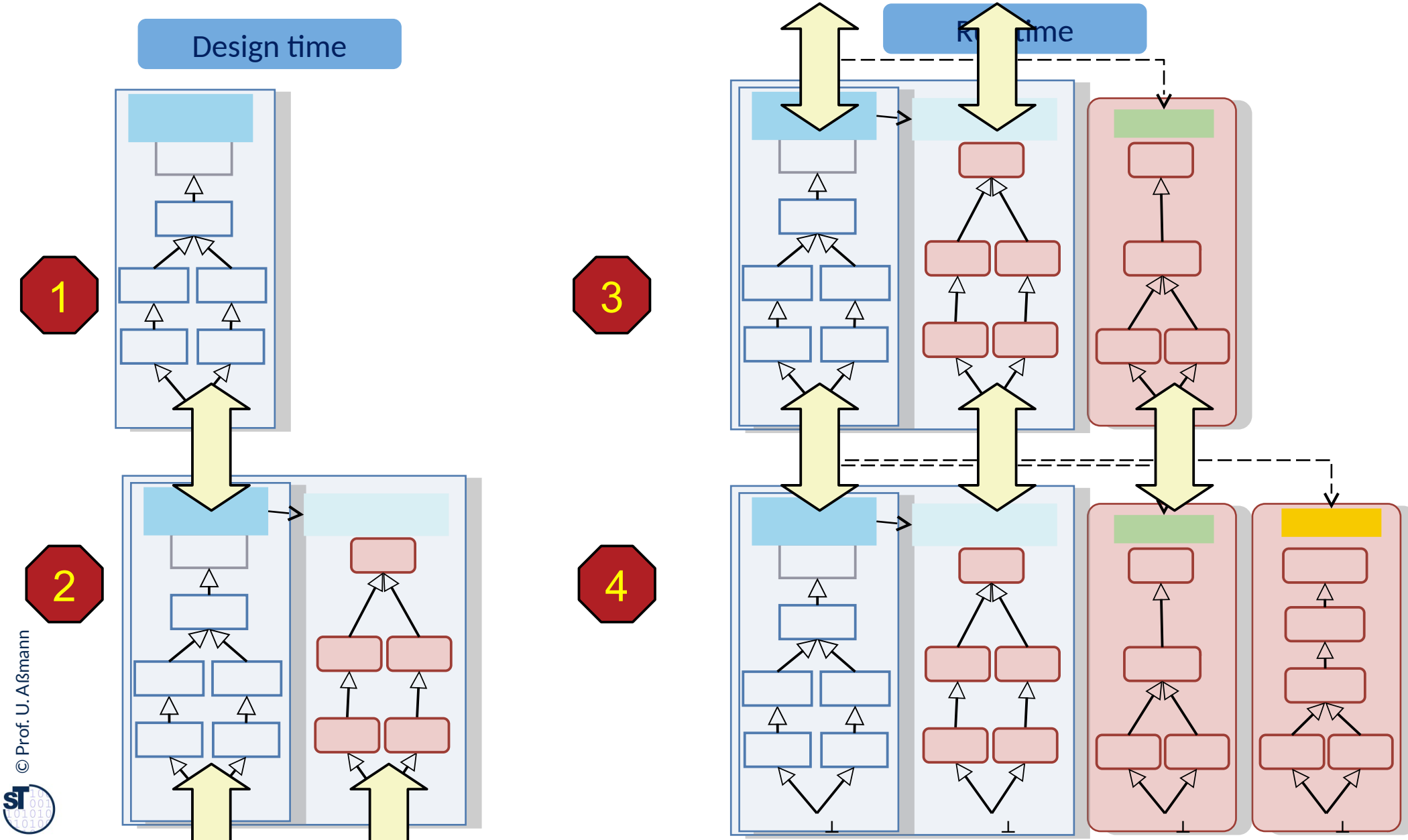
RoSI Megamodel (RoSIMa): Refinement by Role Allocation

- ***Refinement*** by allocation of further roles – static roles at design time, dynamic roles at runtime



RoSIMa: Traceability in Refinement by Role Allocation

- ***Refinement*** by allocation of further roles – static roles at design time, dynamic roles at runtime



RoSI Megamodel (RoSIMa): Cross-Layer Role-Based Refinement in the Software Life Cycle

- Refinement by allocation of roles provides **simple traceability** because Natural objects STAY the same
- Platform properties are „technical“ roles of the objects
  - Technical plattforms are static contexts
  - Dynamic contexts (place, time, service quality)

Causal Mapping of contexts and fludity From requirements level to runtime

Domain Model

Requirements

Design

PSM

Implementation

Run time context 1

Run time context 2

Run time context 3

Natural	Fixed Role 1	Fixed Role 2	Fixed Role 3	Fixed Role 4	Dynamic role 1	Dynamic role 2	Dynamic role 3
Person							
Person	Customer						
Person	Customer	Customer Design					
Person	Customer	Customer Design	Platform-specific Behavior				
Person	Customer	Customer Design	Platform-specific Behavior	Full static behavior			
Person	Customer	Customer Design	Platform-specific Behavior	Full static behavior	Behavior in Context 1		
Person	Customer	Customer Design	Platform-specific Behavior	Full static behavior	Behavior in Context 1	Behavior in Context 2	
Person	Customer	Customer Design	Platform-specific Behavior	Full static behavior	Behavior in Context 1	Behavior in Context 2	Behavior in Context 3

Advantages of RoSIMA (Role-Based MDA)

- Very simple MDA principle, easy traceability:
  - Cores of objects map 1:1 from CIM via PIM and PSM into the application PSI
  - Difference are new roles for PIM, PSM, PSI
- “object fattening”

End

- Why do the models of MDA form a macromodel, while MDA is a megamodel?
- Which trace link types are important for MDA?
- Why is a role-bsed model better for traceability?
- How does JastAdd aspects achieve MDA refinement?
  - How is traceability achieved?
  - How model synchronisation?
- How does RoSIMa achieve global traceability from requirements to run time?
- How will megamodel look like that provides Link-tree-based models and Role-based factorization of objects?
  - How does a trace link look like?
  - Where are the trace links stored?
  - Why can XML be used as simple exchange format in these megamodels?