32. Macromodels 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 21-0.2, 22.01.22

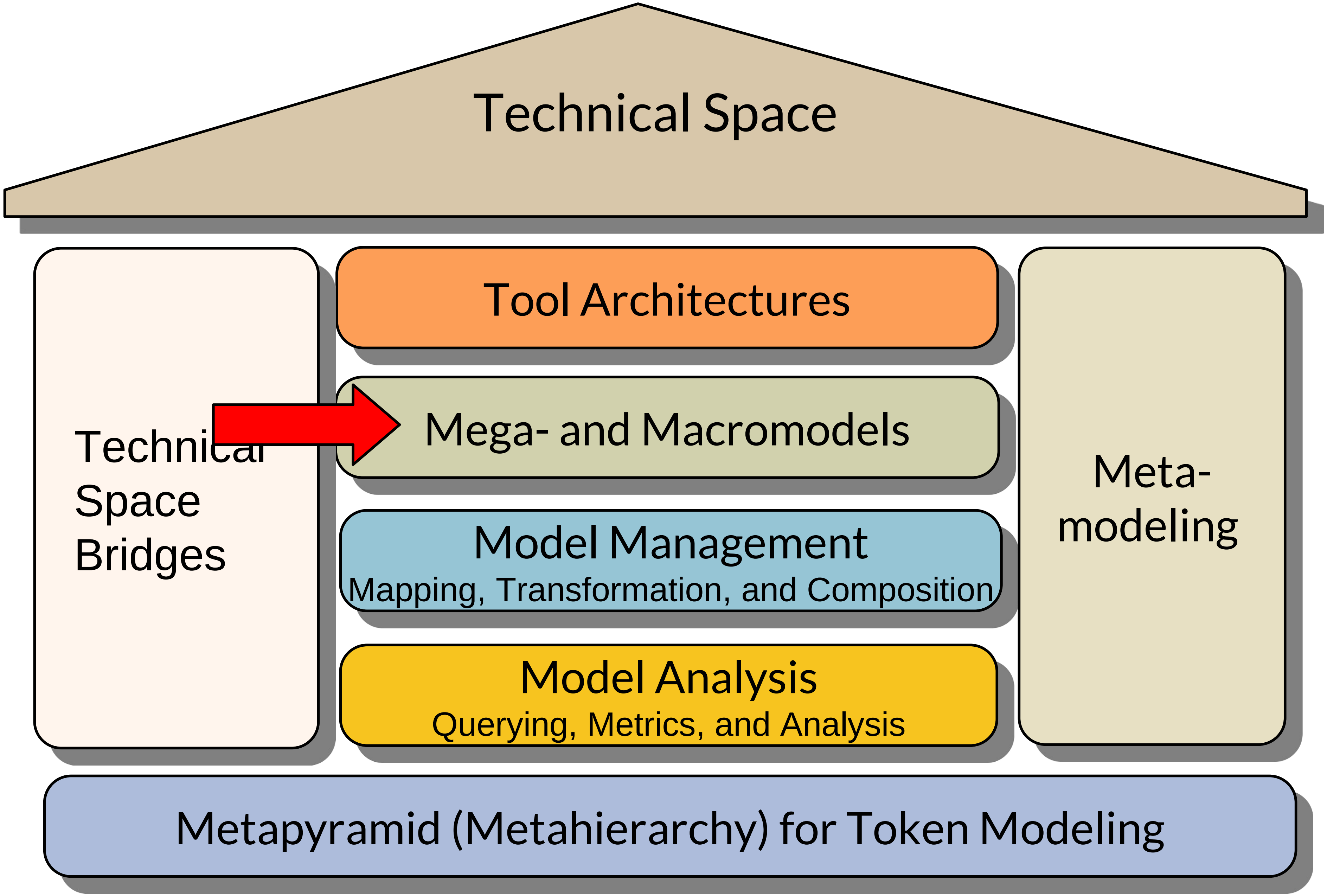
Literature

- [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
- [MID] MID Innovator Tutorial https://www.mid.de/fileadmin/mid/PDF/Kundenbereich/11_R3/de/Innovator_11.3_Leitfaden.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

Q10: The House of a Technical Space



Software Factories

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

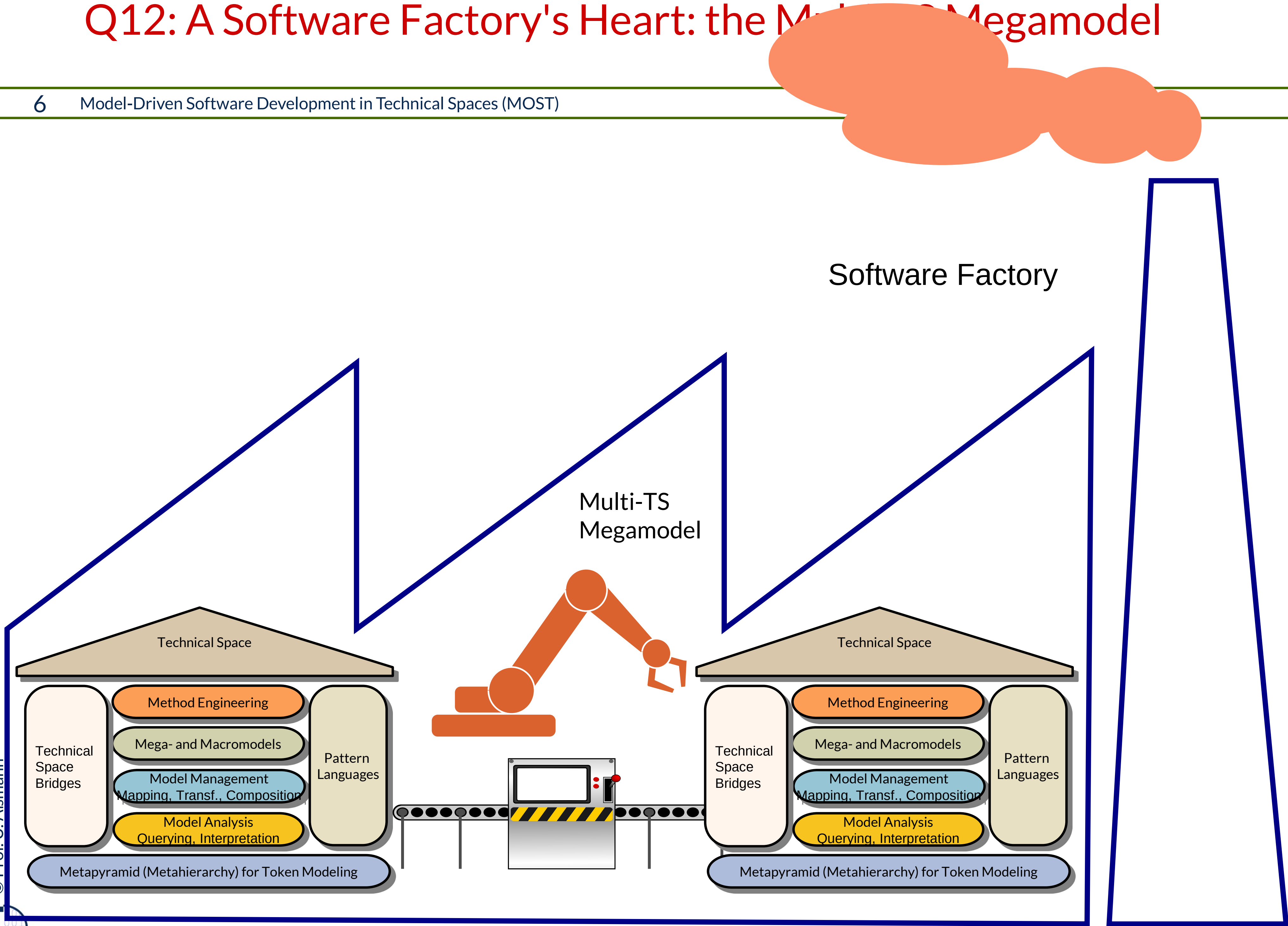
Jack Greenfield

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

In this course:

A **software factory** combines the languages and tools of several technical spaces to create software and cyber-physical systems product families.

Q12: A Software Factory's Heart: the M... Megamodel

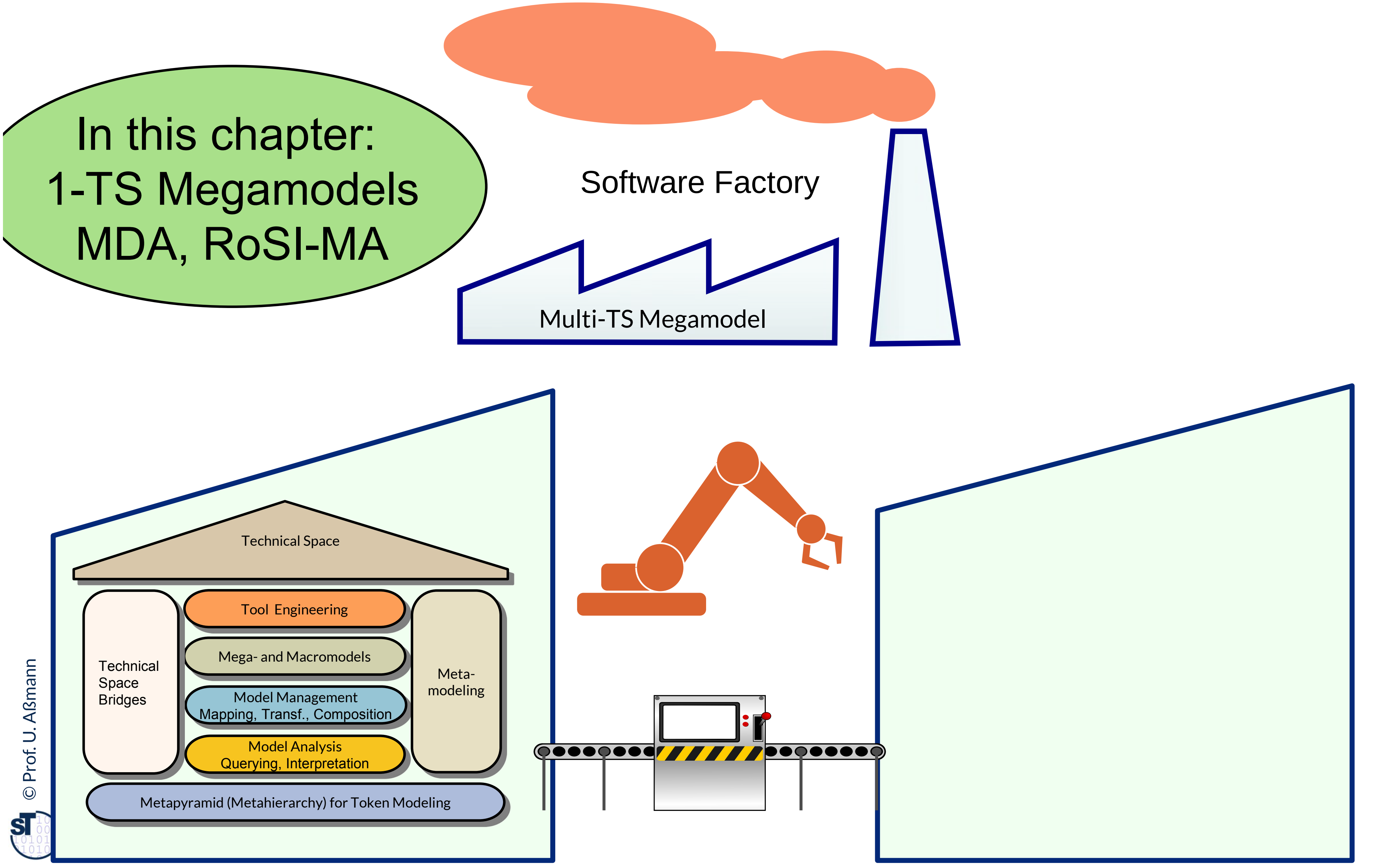


32.1 Model-Driven Architecture (MDA) (Modellgetriebene Architektur)

MDA is a trademark of OMG

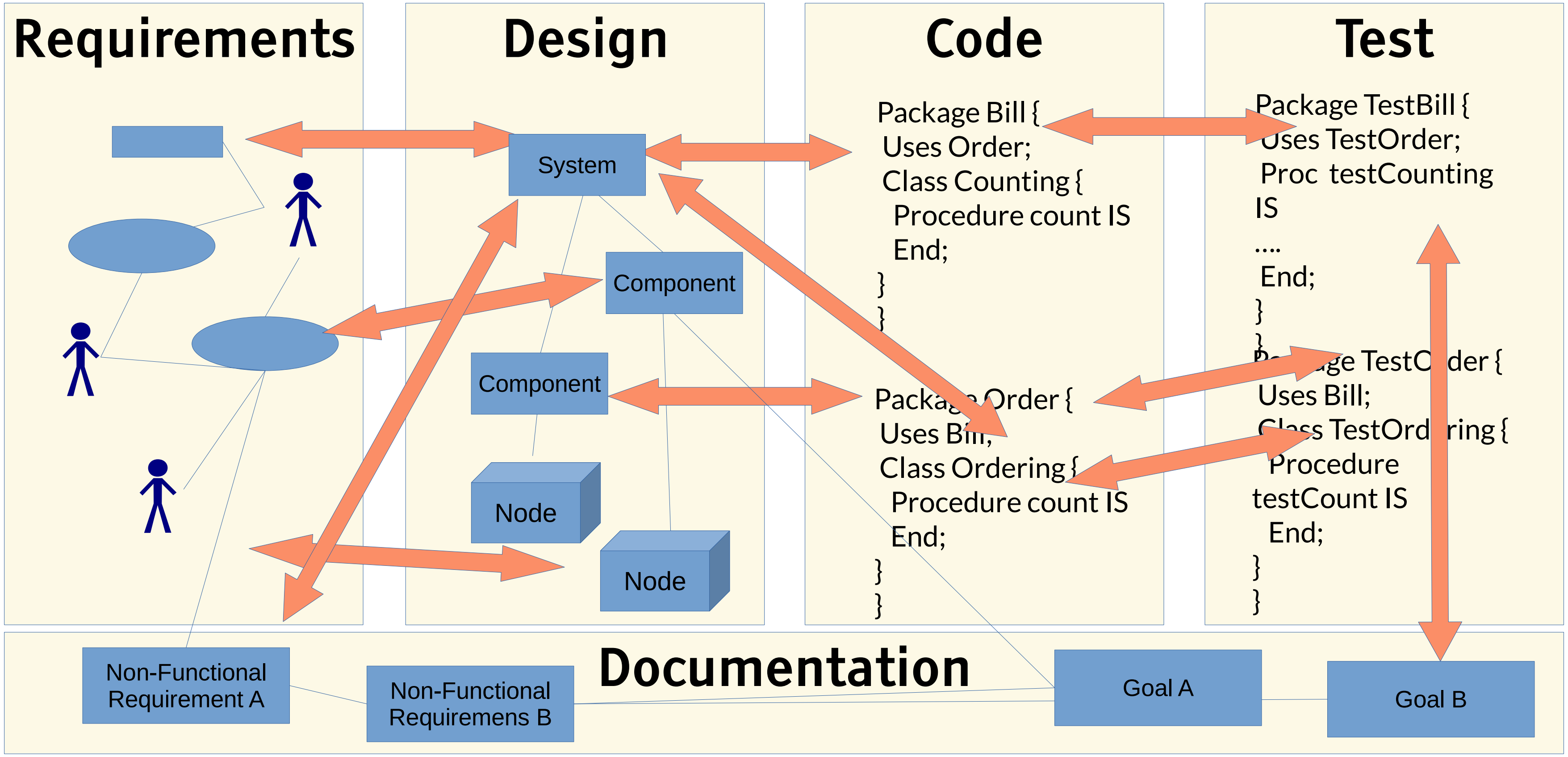
MDA is an industrial megamodel in the spirit of ReDeCT.
Its instances in software product are multimodels, connecting several ***model abstraction levels.***

Software Factories with Only 1 Technical Space



Q12: The ReDoDeCT Problem and its Macromodel

- The **ReDoDeCT problem** is the problem how requirements, documentation, design, code, and tests are related (→ V model)
- Mappings between the Requirements model, Documentation files, Design model, Code, Test cases
- A **ReDoDeCT macromodel** has maintained mappings between all 5 models



Overview Table for Link-Tree Macromodels

The Link-Treeware TS is well apt for macromodel construction in a software factory

- A tree node abstracts a subtree (representant)
  - Attributes and attributions are *composable partial mappings* from treenodes
- RAGs are useful for all kinds of structure- and function-modeling in Link-Tree Macromodels, because they abbreviate dependencies in several models with cross-model relations.
  - In a macromodel under an artificial root (rooted macromodel), attributions can work on the SUM to ensure the constraints
- Relational RAGs (RelRAGs) are useful, because they have bidirectional constraints

	(Plain) MDA	General SUM	Skeleton SUM (partial function extension)
RAGs in Repositories	Markings		Repository-SUM: get/put as higher-order attributions of link trees
			• Javadoc-SUM
RAGs in Data-flow architectures	Needs trace models	get/put as model transformations (lenses)	Flow-SUM: Communicating link trees; In-place transformations of SUM
			• Google Docs, Stream-Based MDA

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

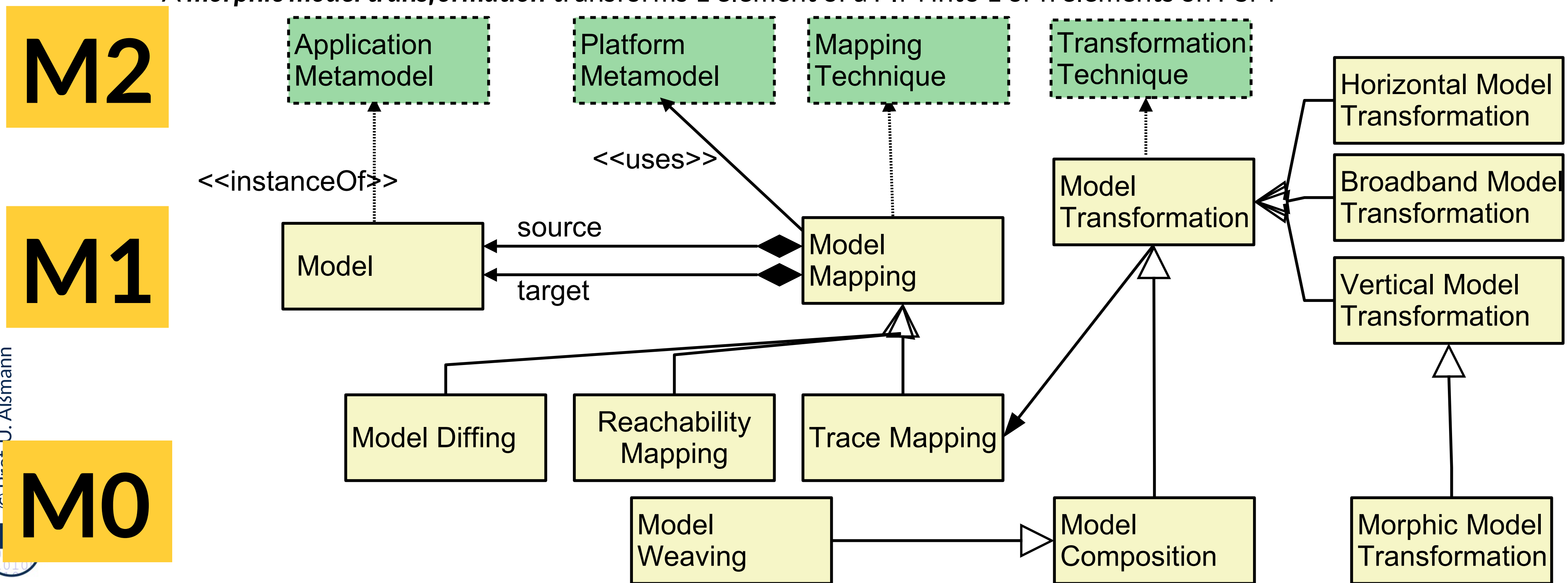
- **MDSD in 1-TS** falls into several main development methods with a macromodels:
  - Engineering with metamodels in ReDeCT-like megamodels (integrated software life-cycle management tools):
    - for integrated requirements, documentation, and testing along the life-cycle
    - Model-Driven Architecture (MDA) (MDA toolkits)
  - Engineering with DSL (domain-specific modeling, DSM) (Meta-CASE toolkits)
    - For simplifying the specification of domain-specific software
- **Model mappings** correlate models
  - capturing *reachability* informations (path abbreviations)
  - 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 (lowerings)** transform a model from a higher-level set into a lower-level set of a broadband (wide-spectrum) language
- **Model compositions** compose models with extensions
  - **Model weavings** extend models by other models and weave them together

Model-Driven Architecture (MDA)

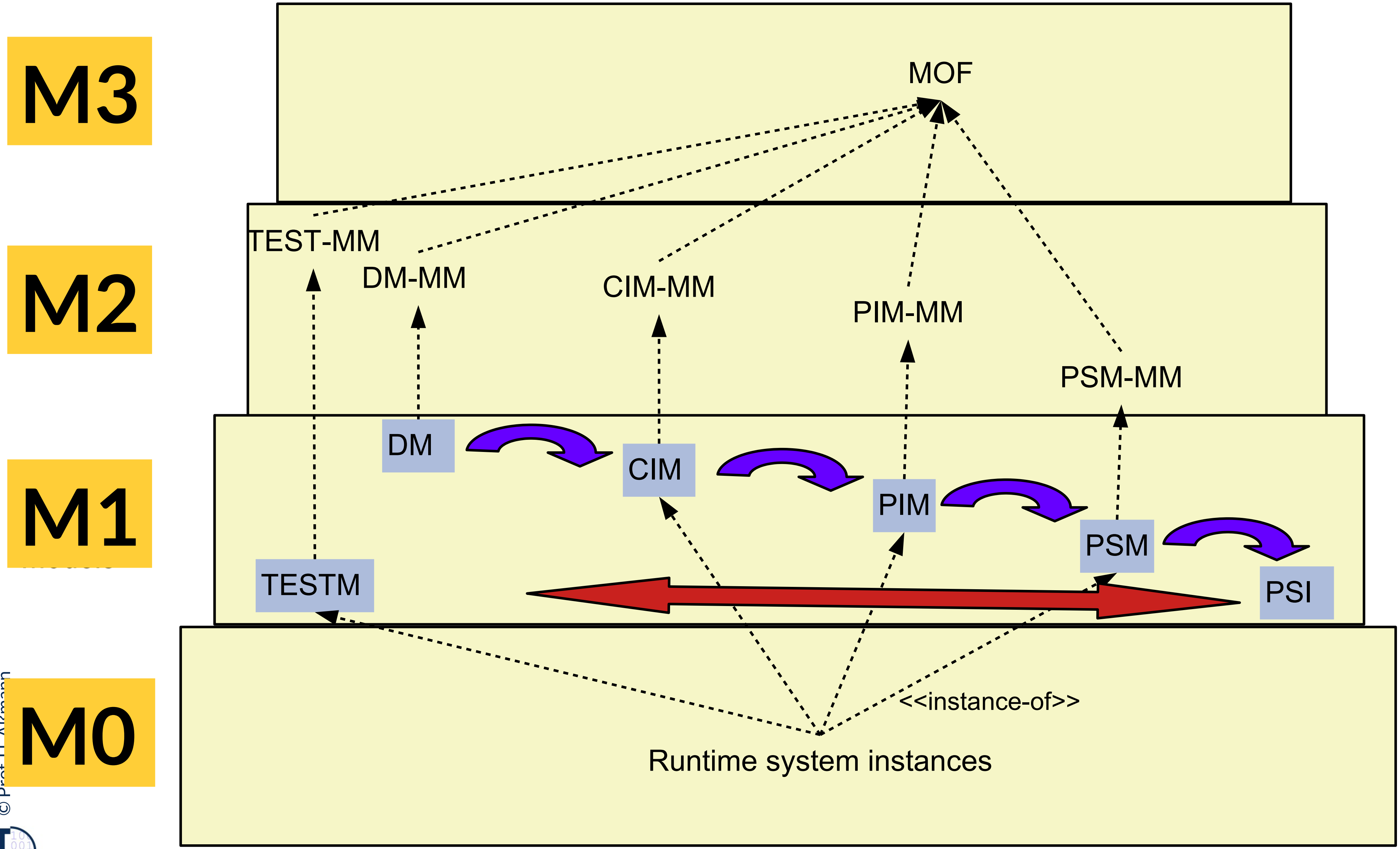
- Model-Driven Architecture (MDA) is a macromodel similar to ReDoDECT, but distinguishes more models:
  - Platform-independent model (architectural)
  - Platform-specific model (in modeling language equivalent to coding language)
  - Platform-specific implmentation (in coding language)
- On the other hand, documentation is neglected :-(
- MDA uses *model mappings, horizontal and vertical model transformations*, as well as *code generation*

What are Model Mappings?

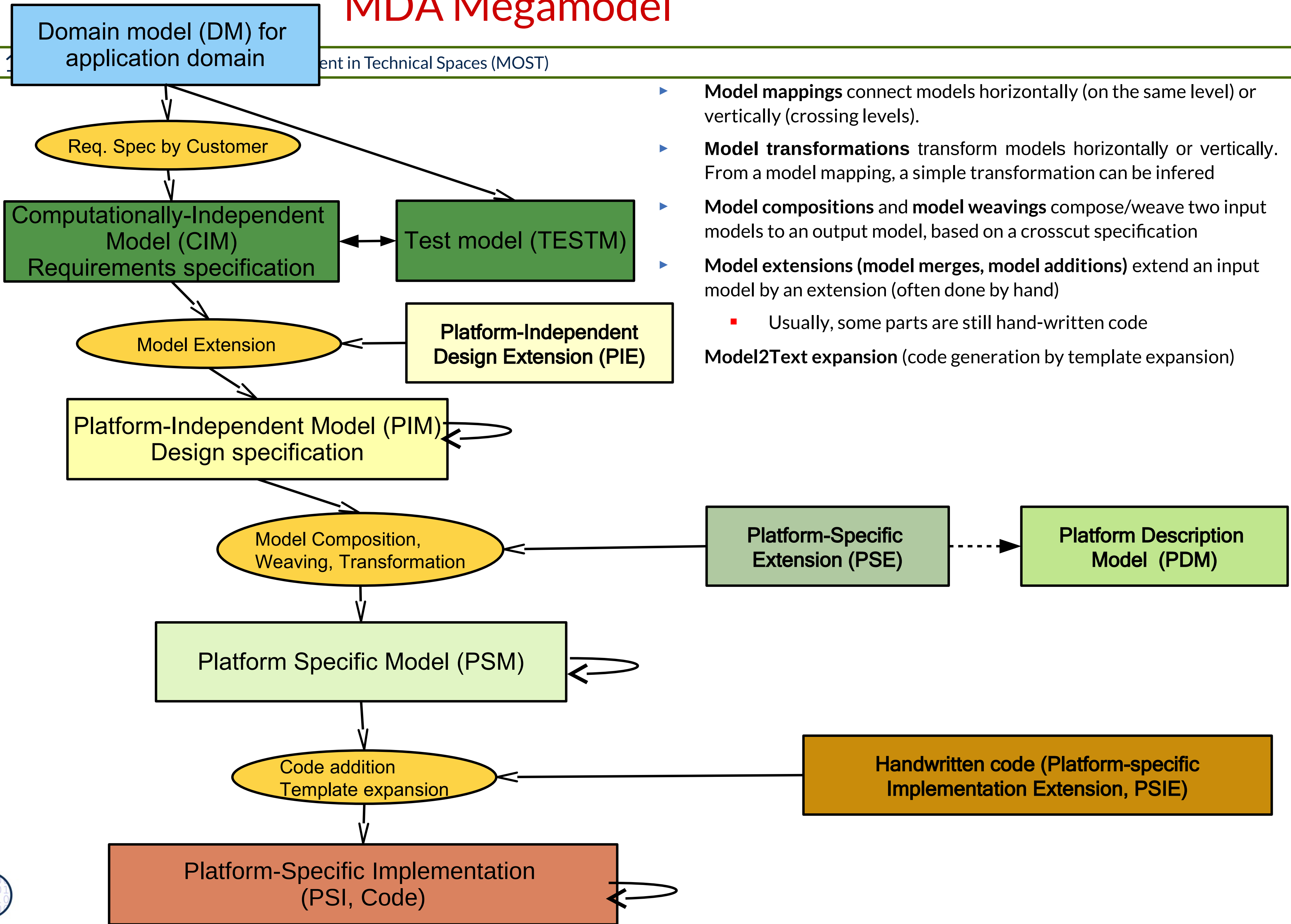
- Model mappings are link graphs between model elements of different models
- Mappings are ***automatic*** or ***semi-automatic***:
  - A model mapping can be generated from a model difference analysis
  - Some are step-wise refinement of the model by transformation (in MDA)
- A model mapping is ***horizontal***, 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



The MDA Megamodel, a Specific Variant of ReDoDeCT, Embedded in the MOF Metapyramid



Q9: Model Mappings and Model Weavings in the MDA Megamodel

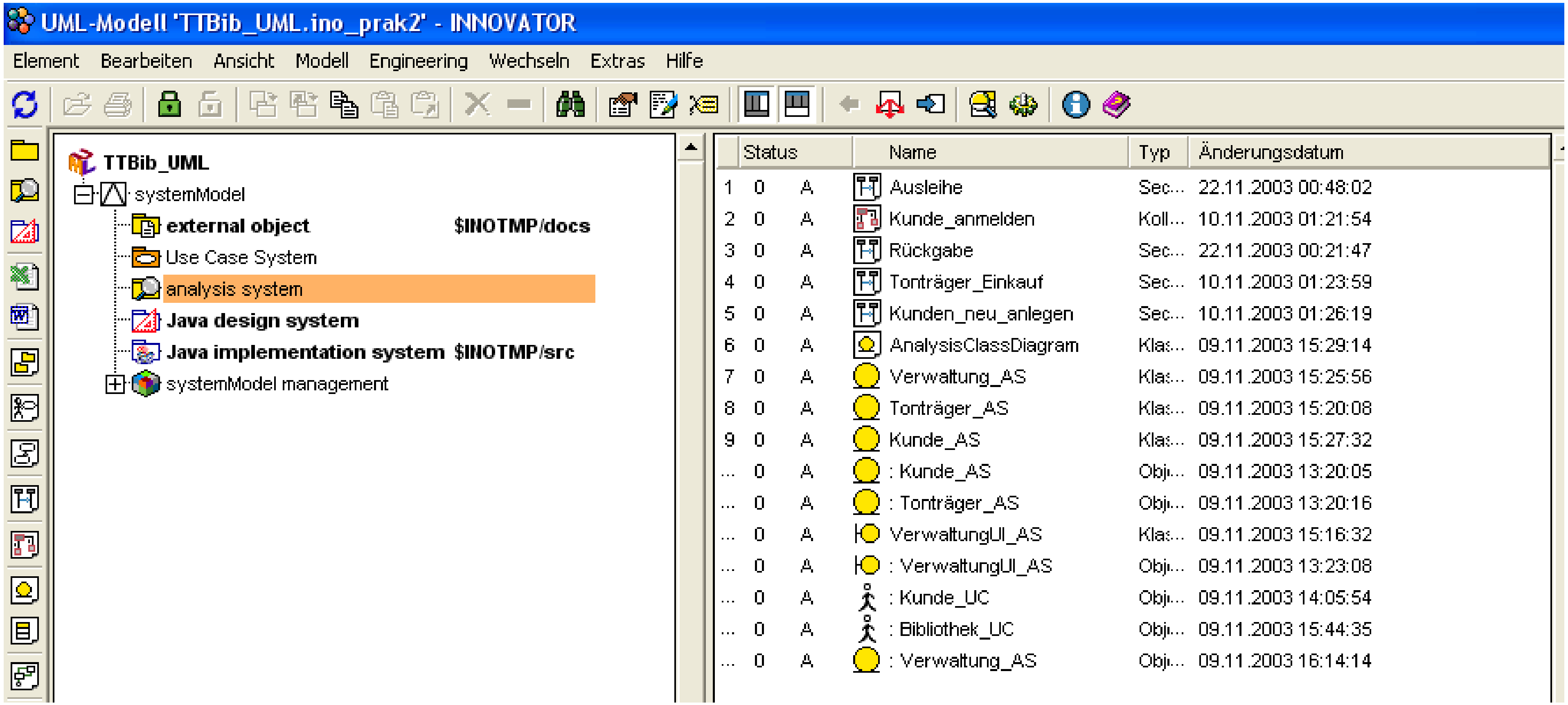


- **Model mappings** connect models horizontally (on the same level) or vertically (crossing levels).
- **Model transformations** transform models horizontally or vertically. From a model mapping, a simple transformation can be infered
- **Model compositions** and **model weavings** compose/weave two 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 [MID]



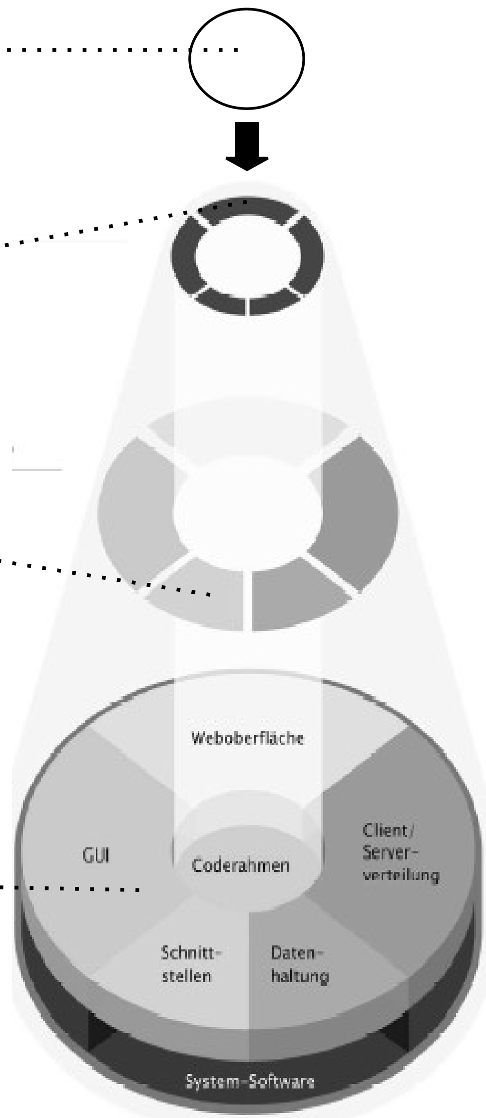
Example: PIM and PSM Extend the CIM in the Janus Toolkit

Domain model (DM) 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 DM - 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

Model Management in Megamodels

- In the MDA megamodel,because MDA *enriches models from top to bottom,* 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

32.1.2 Different Forms of MDA

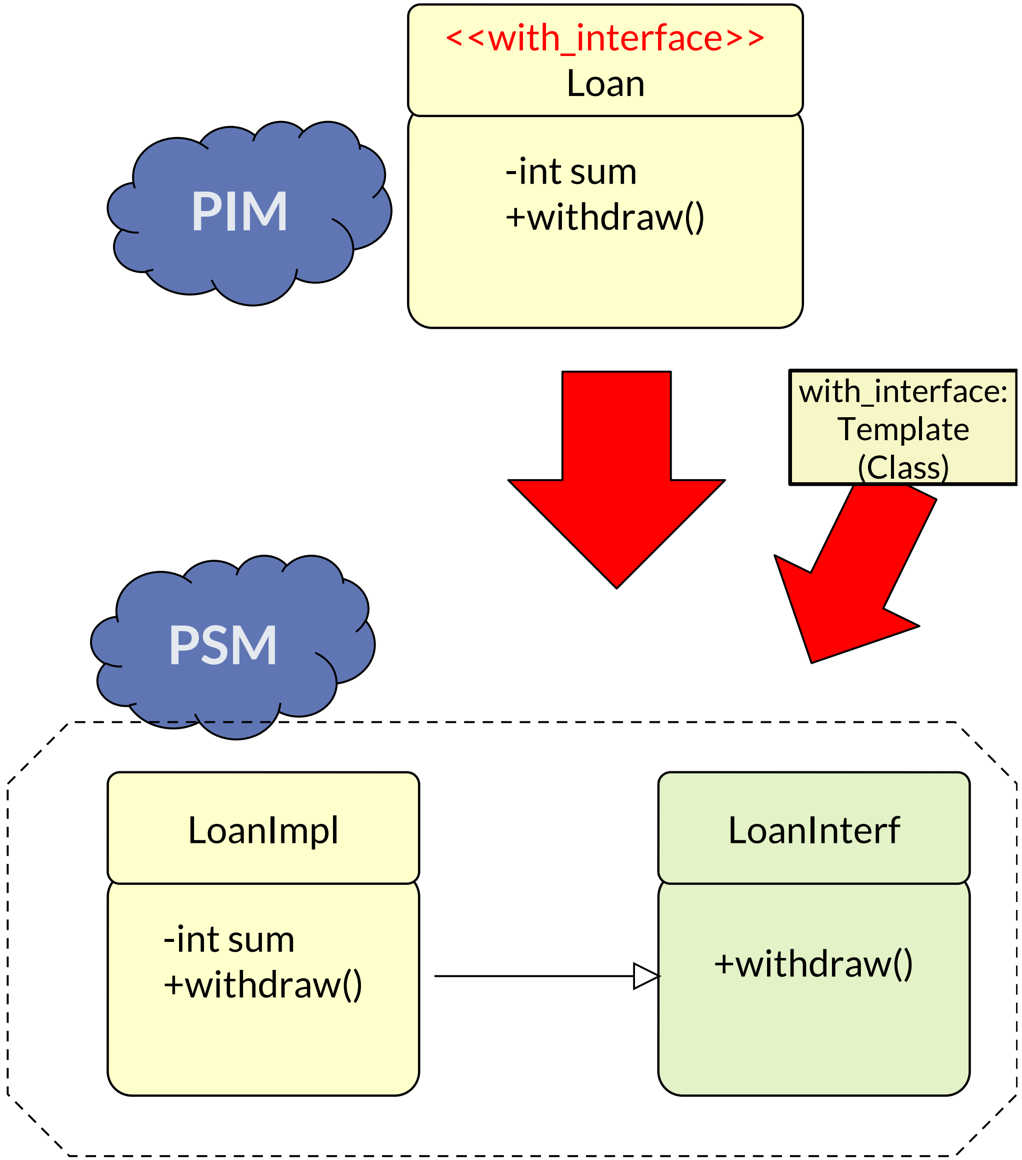
Different forms of MDA

- A ***transformative MDA*** uses refinement transformations for variation
  - introduces trace links (32.3)
- An MDA is called ***component-based (CoMDA)*** if the variation action is the exchange of an implementation behind an interface, or if the component model is used for exchange
  - RoSIMDA MDA (32.5)
- A ***transformative CoMDA*** uses point-wise refinement transformations on a model-based component model
  - for instance, refinements in Petrinets
    - combining trace links and component-based MDA (32.3 and 32.5)
- A **MDA-SUM** uses transformative or component-based MDA for realizing *views* on a *single underlying model (SUM)* (next chapter)

32.1.3 Morphic Model Mappings and Transformations

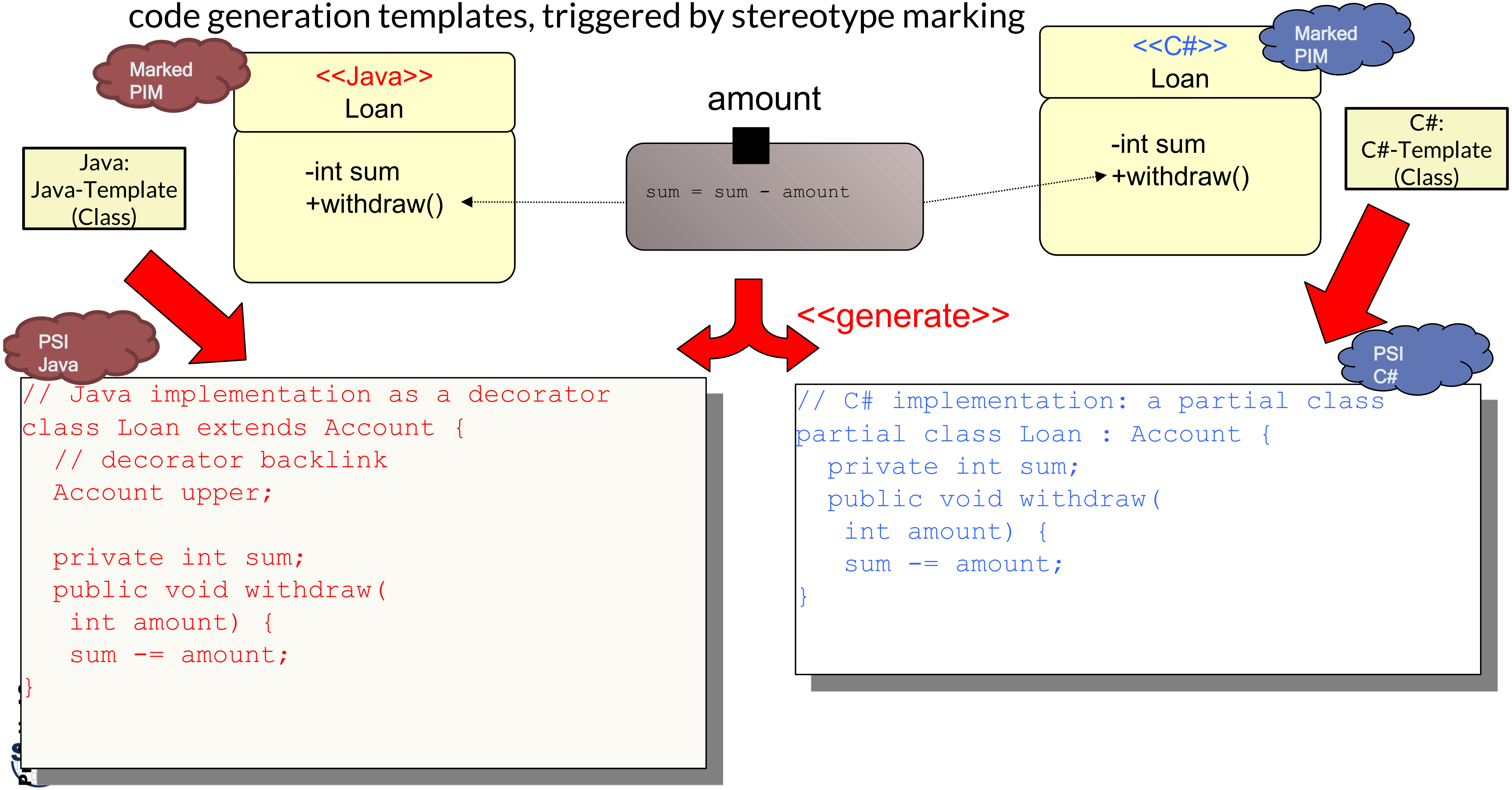
Morphic Mappings and Pointwise Transformations on Marked PIMs

- **Morphic mappings (1:1 or 1:n)** are defined by ***marked PIMs***:
  - Stereotypes introduce a mapping 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
- Easy traceability by morphic mapping



Example of a Marked PIM and the Induced Pointwise 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



Cartridges are Transformation Libraries for Marked PIMs

- A **Cartridge** is a plugin to an MDA tool defining both the model mapping and the model transformation
  - For vertical and horizontal transformations
  - Definition of stereotypes for PIM markings in vertical 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

32.1.4 Cartridges (Platform Extensions) 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 Main Tree Spec
// Domain Model
class Loan extends Account {
   eq ..
   syn ..
   inh ..
}
class Saving extends Account {
  eq ..
  syn ..
  inh ..
}
```

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

}
```

Intertype declarations

Ex.: JastAdd Aspects are Cartridges

- A JastAdd Aspect, like a cartridge, extends a set of Main 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

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

```
// JastAdd Additional Tree Spec
aspect CIM {
  eq Loan.fun1()
  eq ..
  syn ..
  inh ..
}
```

```
// JastAdd Additional Tree Spec
aspect TestM {
  eq Loan.test_fun1()
  eq ..
  syn ..
  inh ..
```

```
// JastAdd Additional Tree Spec
aspect PIM {
  eq Loan.fun2()
  eq ..
  syn ..
  inh ..
}
```

```
// JastAdd Additional Tree Spec
aspect PSM {
  eq Loan.fun3()
  eq ..
  syn ..
  inh ..
}
```

```
// JastAdd Additional Tree Spec
aspect PSI {
  eq Loan.fun4()
  eq ..
  syn ..
  inh ..
}
```

MDA by Composition of RAG Aspects

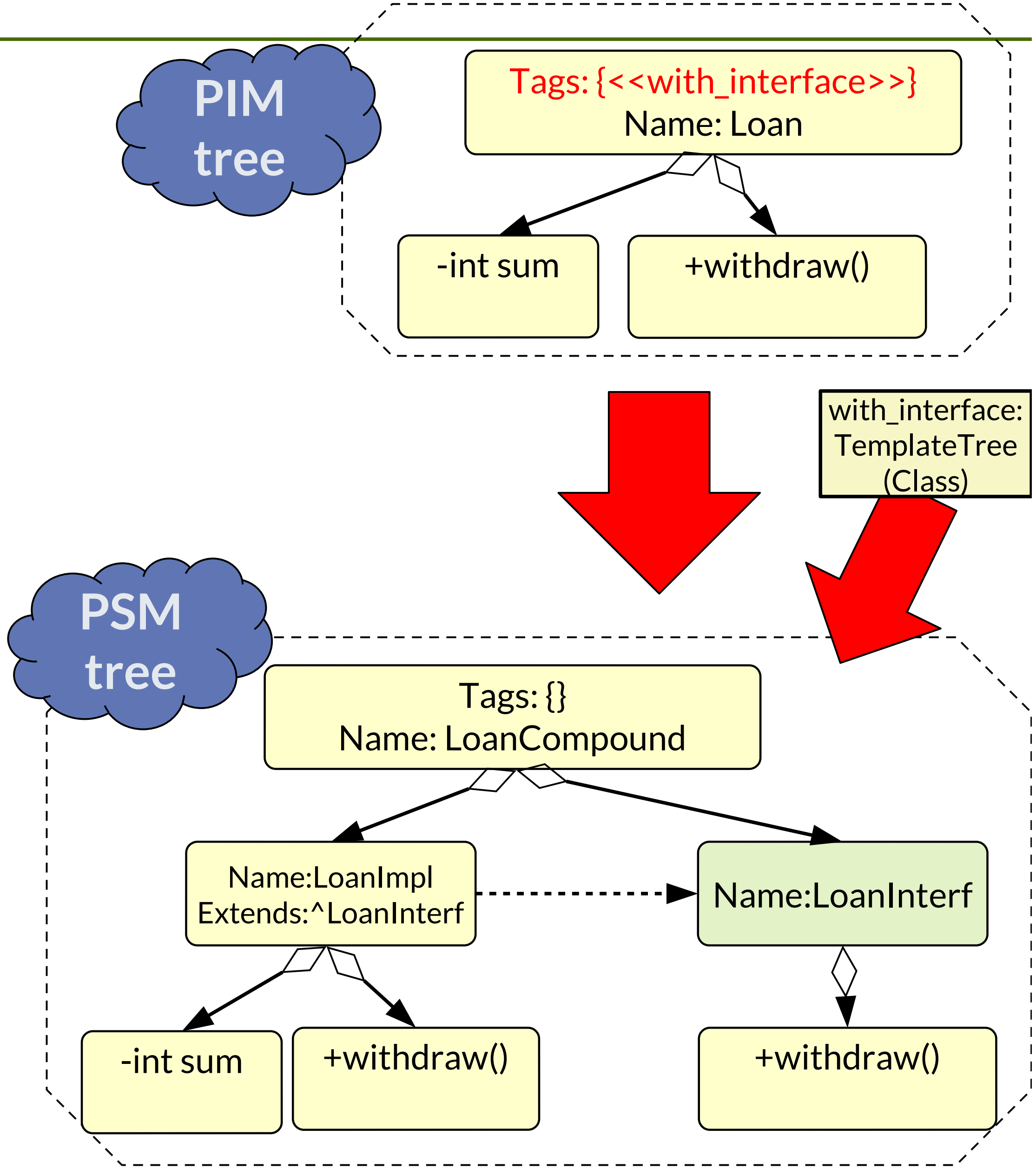
- RAG modules, e.g., JastAdd aspects, can be used as MDA cartridges
  - They compose class extensions “around” class names
  - Model weaving is done by class composition
  - Intertype declarations introduce “mixins” into classes of main syntax tree
- Model Refinement (in MDA) is done by modular composition (aspect composition) with intertype declarations
  - Model synchronisation is done by re-composition
  - RAG-MDA supports **composable macromodels**
- Model mappings achieved by common class names
  - Tracing is easy (common classes for extensions)

RAG modules, e.g., JastAdd aspects, can be used as MDA cartridges

32.1.5 Morphic Model Transformations in JastAdd

Morphic Transformations on Marked PIMs

- **Morphic mappings (1:1 or 1:n)** can be realized by JastAdd Rewrite operations or Term rewrite operations (Stratego, Xcerpt)
  - If Users add a stereotype to a node of a PIM
  - Rewrites can reduce them
- The rewrite is a replace operation of the marked node by its "implementation"
- Rewrite rule transforms redex of upper model to snippet in lower model
- Easy traceability by morphic mapping
- The PIM tree as well as the PSM tree are represented by the top node
- The PIM tree snippet and the PSM tree snippet are *homomorphic regions*



32.2 MDA Toolkits

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]

Important Features of MDA Toolkits

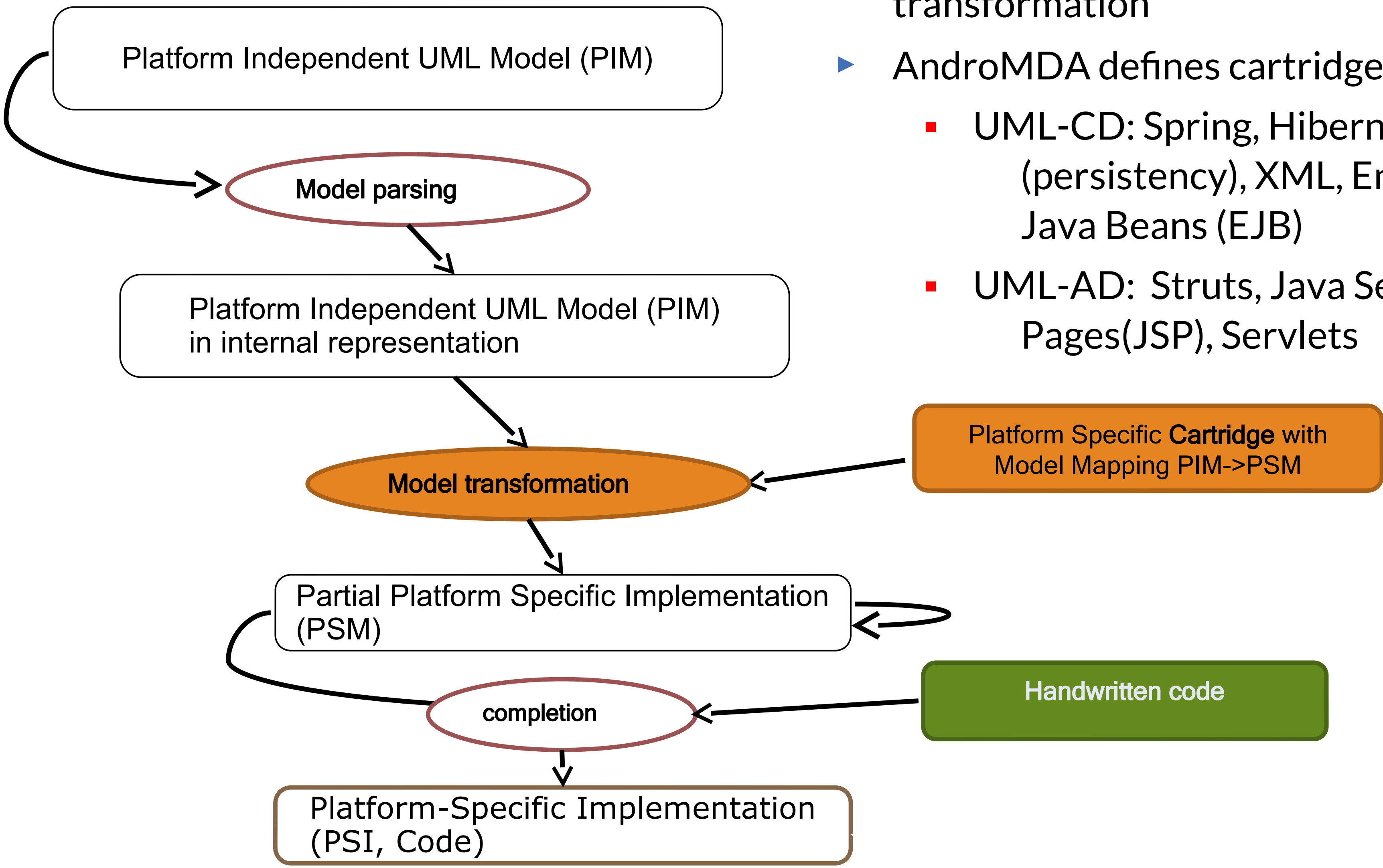
- **Model-to-Model Mapping** bzw. **Model-to-Model Transformation** (e.g., PIM to PSM) with cartridges
- **User definition of model transformation cartridges** with query and transformation languages
  - e.g., with QVT, ATL, Graph writing or XML Rewriting
- **Forward- und Reverse-Engineering**
  - Code generation (Model-to-Code Transformation, PSM to PSI)
    - Mapping to a programming language (e.g., with JMI)
- **Roundtrip-Engineering** between models and code
- **Single underlying model (SUM):** forming views by get and put operations
- **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]

32.2.1 AndroMDA, a Leading MDA Toolkit Focusing on PIM-PSM Transformations

[www.androMDA.org]

- 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

32.2.2 MDA Toolkit ArcStyler

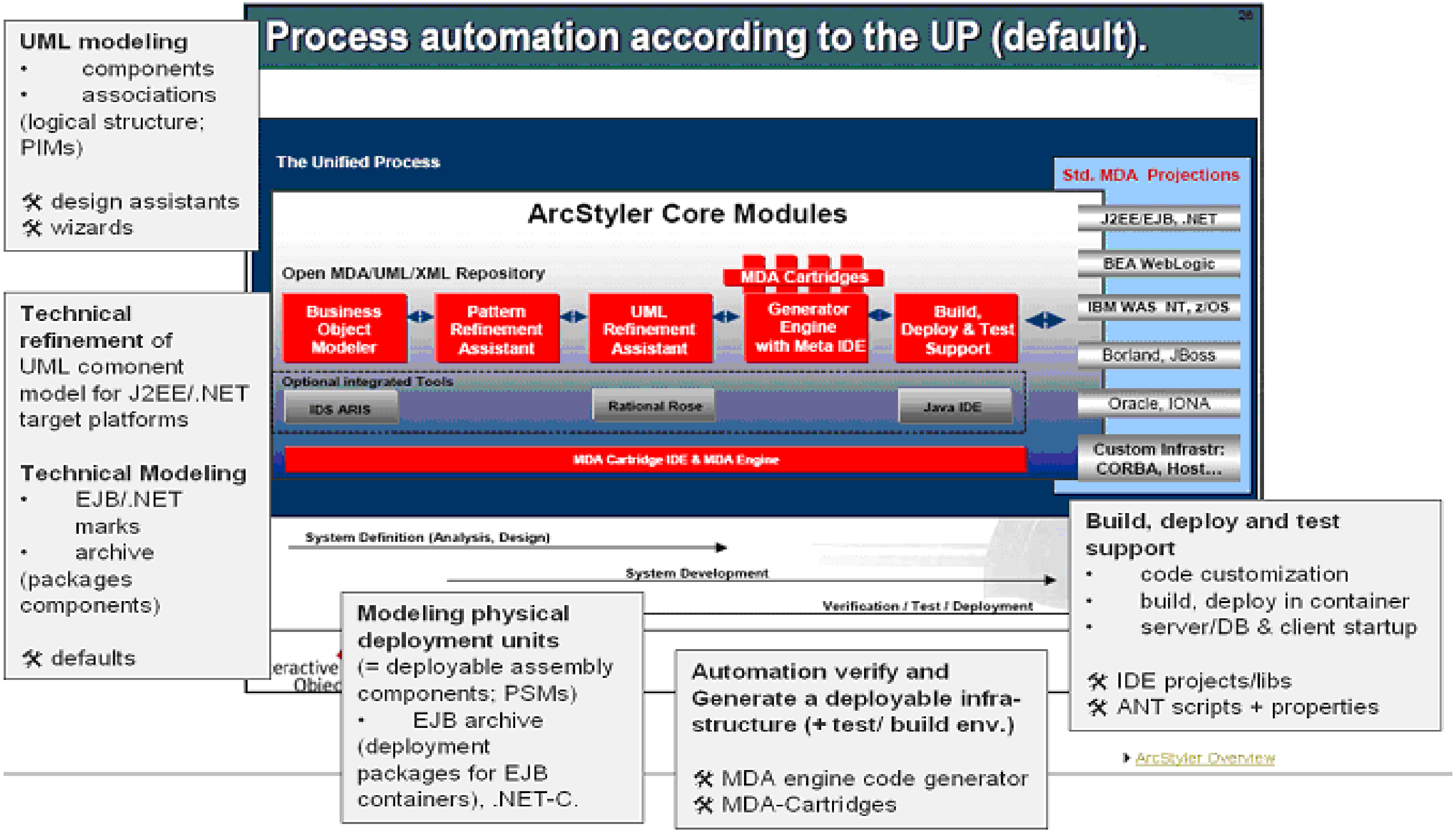
ArcStyler is a toolkit working with several UML-editors such as MagicDraw or Rational Rose

- Cartridges for model mappings and transformations
- **Object Modeler** for requirements modeling; based on CRC-Cards
- **Pattern Refinement Assistant** 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

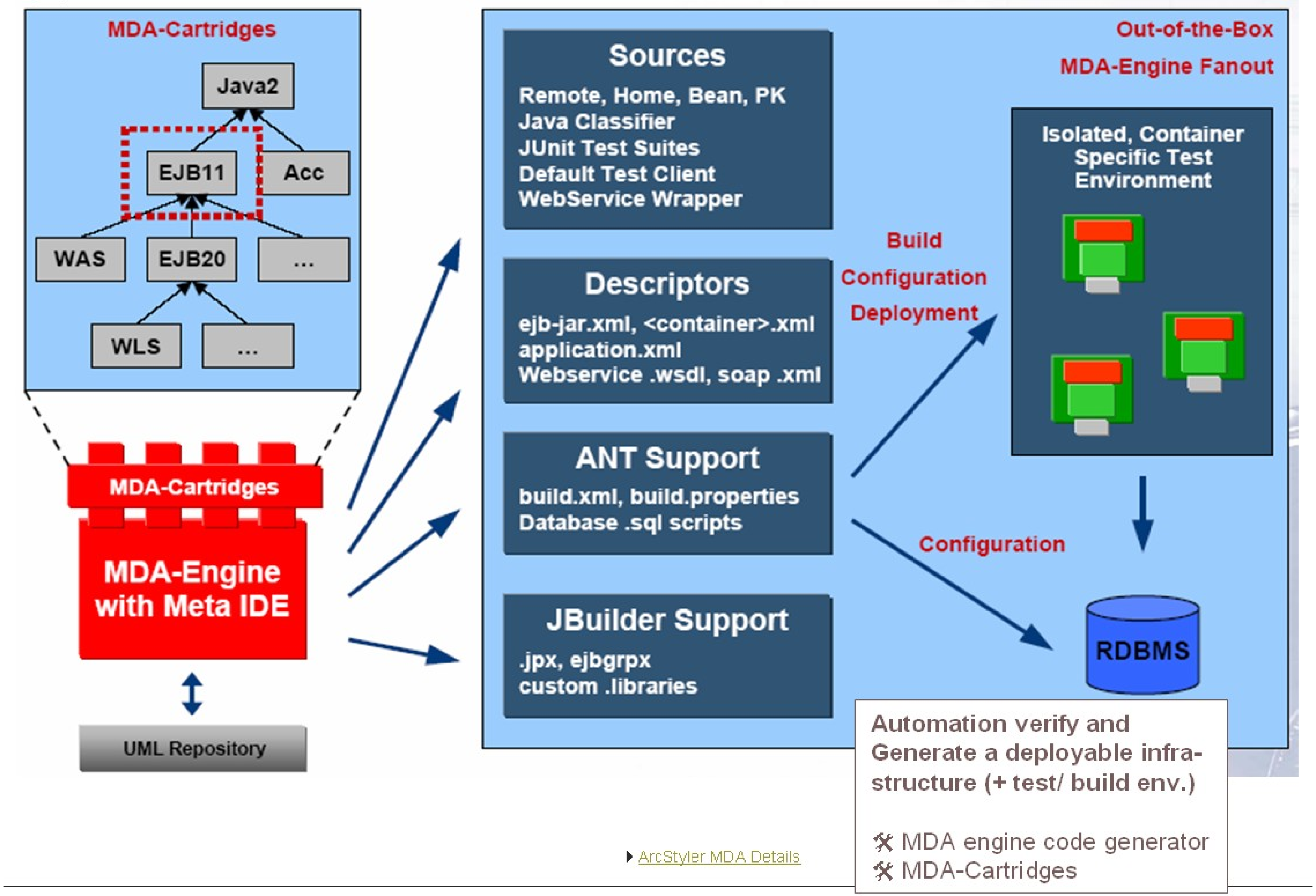


https://www.omg.org/mda/mda_files/P2A_Tutorial.pdf

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 werkzeuggestützte Softwareentwicklung nach der Model-Driven-Architecture; Großer Beleg an der Fakultät Informatik der TU Dresden 2004

32.3 Traceability between Models

- Model transformations generate trace mappings

omitted in 2021/22

Advantages of Model Mappings

- **Error tracing**
  - When an error occurs during testing or runtime, we want to trace back the error to a design element or requirements element
- **Traceability**
  - We want to know which requirement (feature) influences which design, code, and test elements, so that we can demarcate modules in the solution space (product line development)
- **Synchronization in Development:**
  - Two models are called **synchronized**, if the change of one of them leads automatically to a hot-update of the other
- **Cohesion of Distributed Information:**
  - Two related model elements may contain distributed information about a thing. The relation allows for reconstructing the full information
  - Example:
    - Storing two roles of an object in two different models (See "Amoeba Object Pattern")
    - Splitting the representation of the requirements on an object and its design in requirements vs design model

Different Forms of Model Mappings

- **Directly specified mappings** specify a deterministic mapping function between a source and target model.
  - Direct mappings are specified in GUI or text files
  - Direct mappings may be ***complete*** or ***incomplete***
- **Recursive mappings** are defined in a functional language
  - **Denotational semantics** is a complete direct mapping of two languages
  - The **coverage** of the source model must be ensured (completeness of specification)
- **General mappings** may be intensionally specified. Source and target models are mapped
  - With graph reachability expressions (QVT-R, TgreQL, EARS)
  - With query expressions (Semmle.QL)
  - With expressions in a logic (F-Datalog)
- **Inter-model mappings** are defined between model elements of different models
- **Lifted inter-model mappings** are lifted from intra-model element mappings

Why Traceability in a Macromodel?

[Grammel]

System Comprehension:

- Trace mappings improve orientation in multimodels by navigating via trace links along model transformation chains

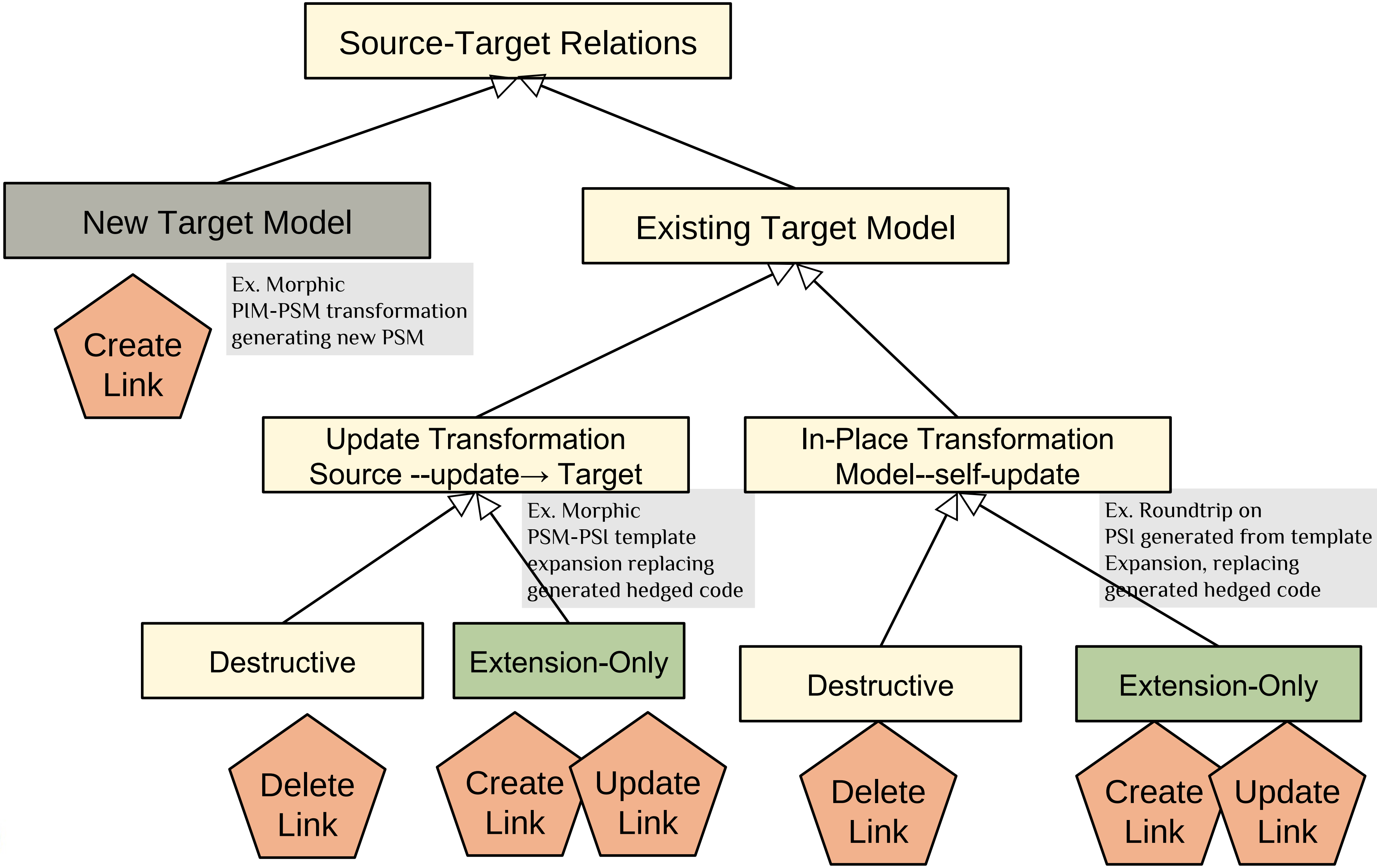
- **Change Impact Analysis**:
  - to analyze the impact of a model change on other models
  - to analyze the impact of a model change on existing *generated* or *transformed* output
  - To enable to do model synchronization (hot updating dependent parts)
- **Orphan Analysis:** finding orphaned elements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

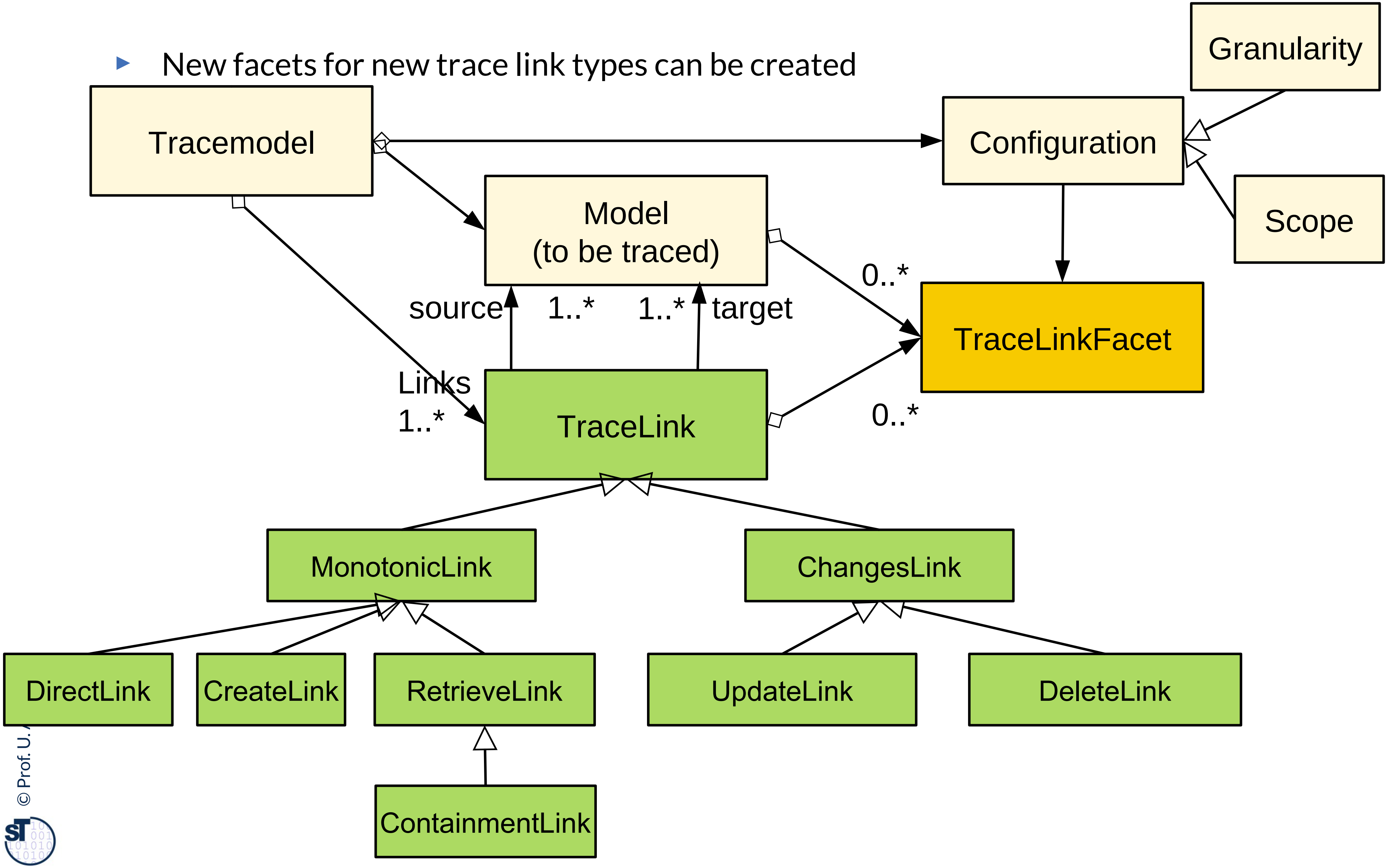
Traceability Metamodel: CRUD Types of Trace Links between Model Elements of Different Models

[Grammel]



Extensible Traceability Metamodel acc. to Grammel

- New facets for new trace link types can be created



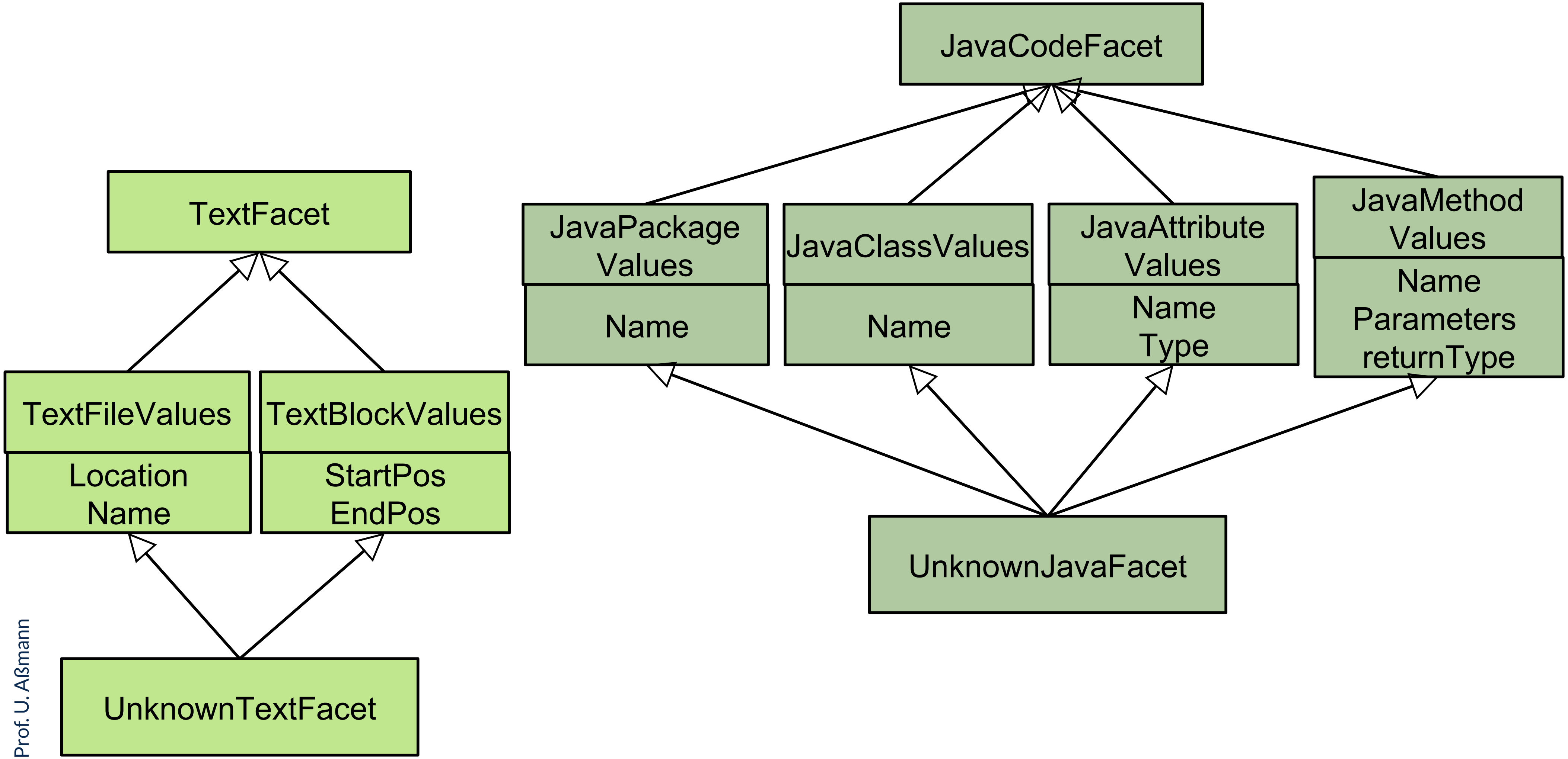
Traceability in Macromodels

- Piecemeal growth of macromodels in the software process:
  - Start with requirements, then add more stuff and models
- **Add links**
  - **Symmetric "Direct" (auto-drawn) links** are drawn between model element MA from model A and model element MB whenever MB is related to MA
    - Specified by hand or 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

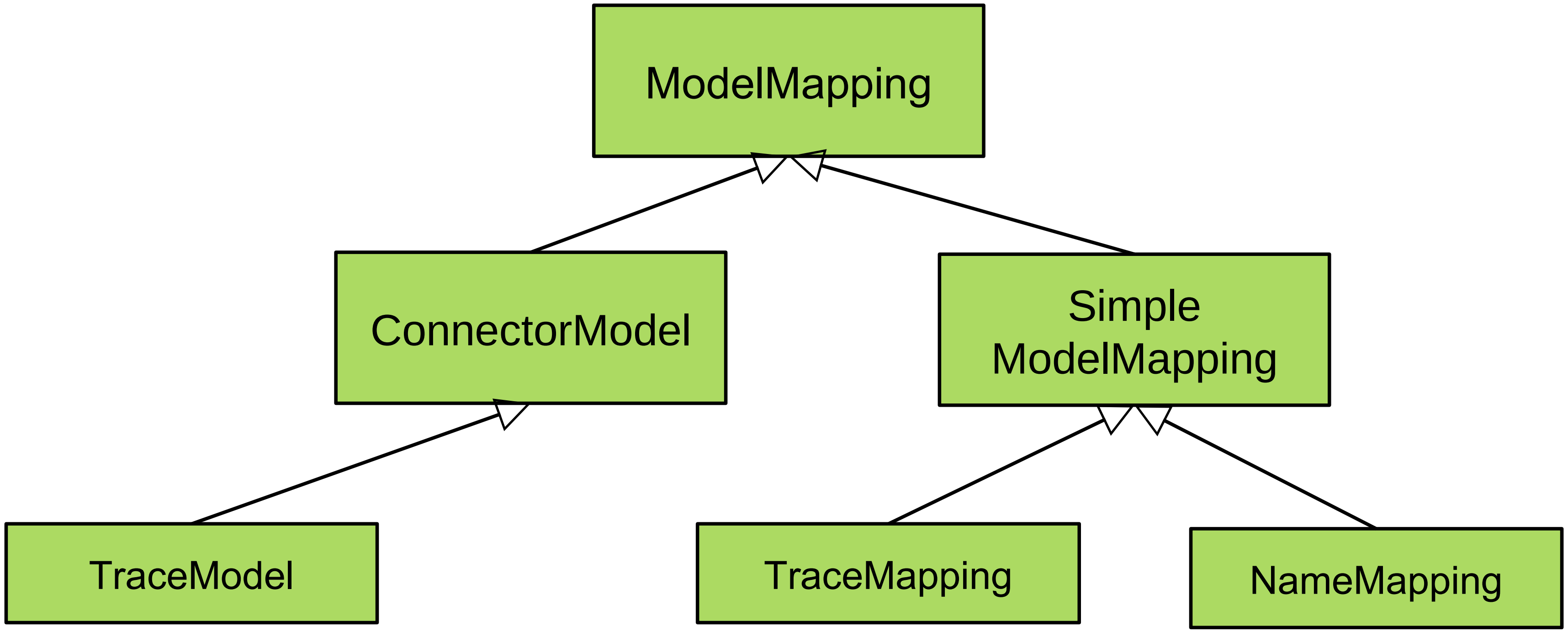
[Grammel]

- Facets factorize inheritance hierarchies; new facets extend inheritance hierarchies



Different Kinds of Trace Models

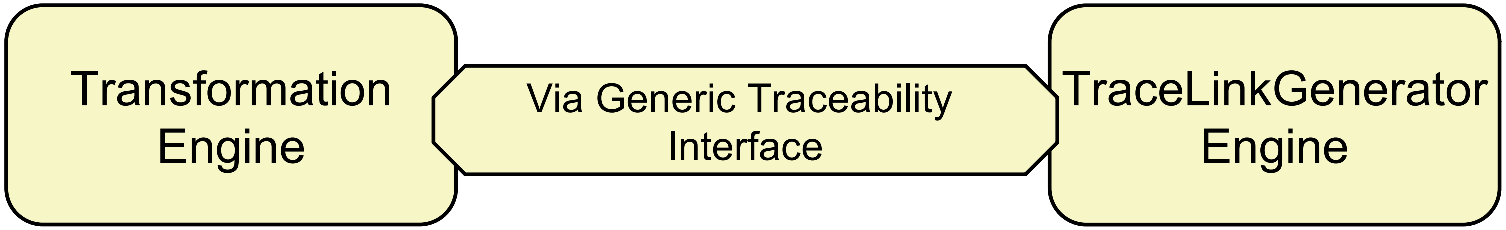
- So far, trace mappings were realized as associations in a **simple model mapping**
- The trace metamodel can be extended to describe a ***trace model***, a specific form of ***connector model***

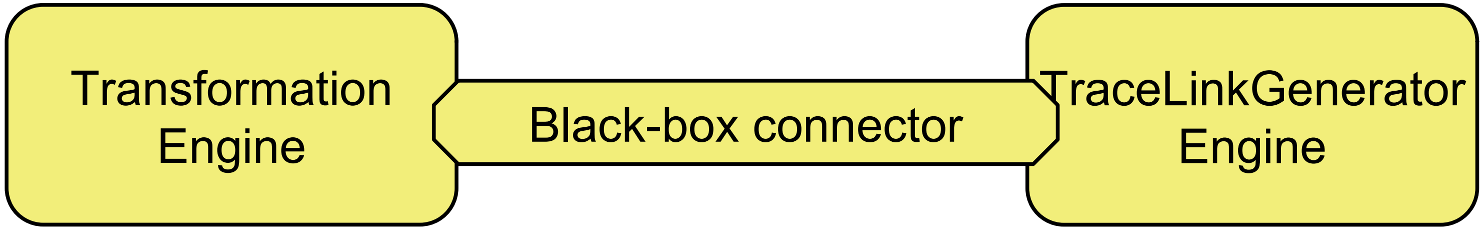


Adding a Trace Link Generator to Tools

[Grammel]

- TraceLinkGenerators for Trace Models must be written by hand
- They can be connected to transformation engines and cartriges in three ways, following a ***generic traceability interface***:





Transformation engine must know and call the generator

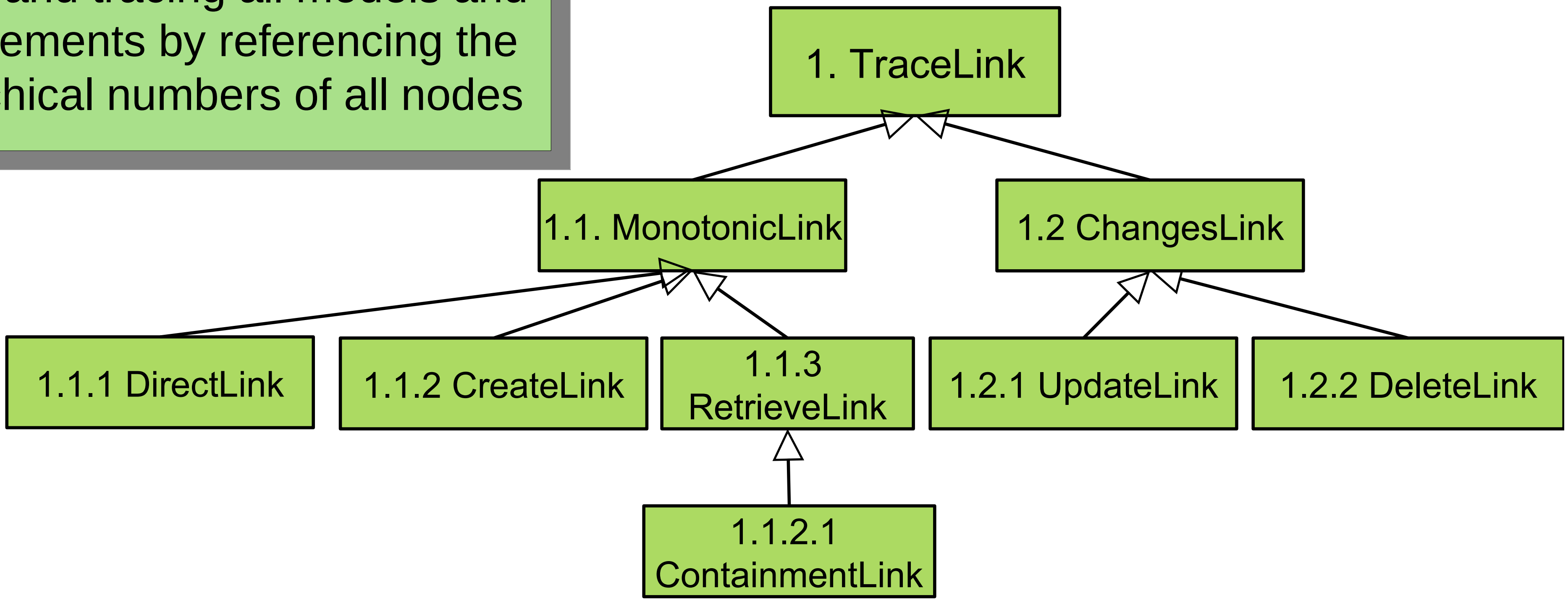
Transformation engine need not know but is extended Invasively or woven By AOP

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 *trace(link) 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:

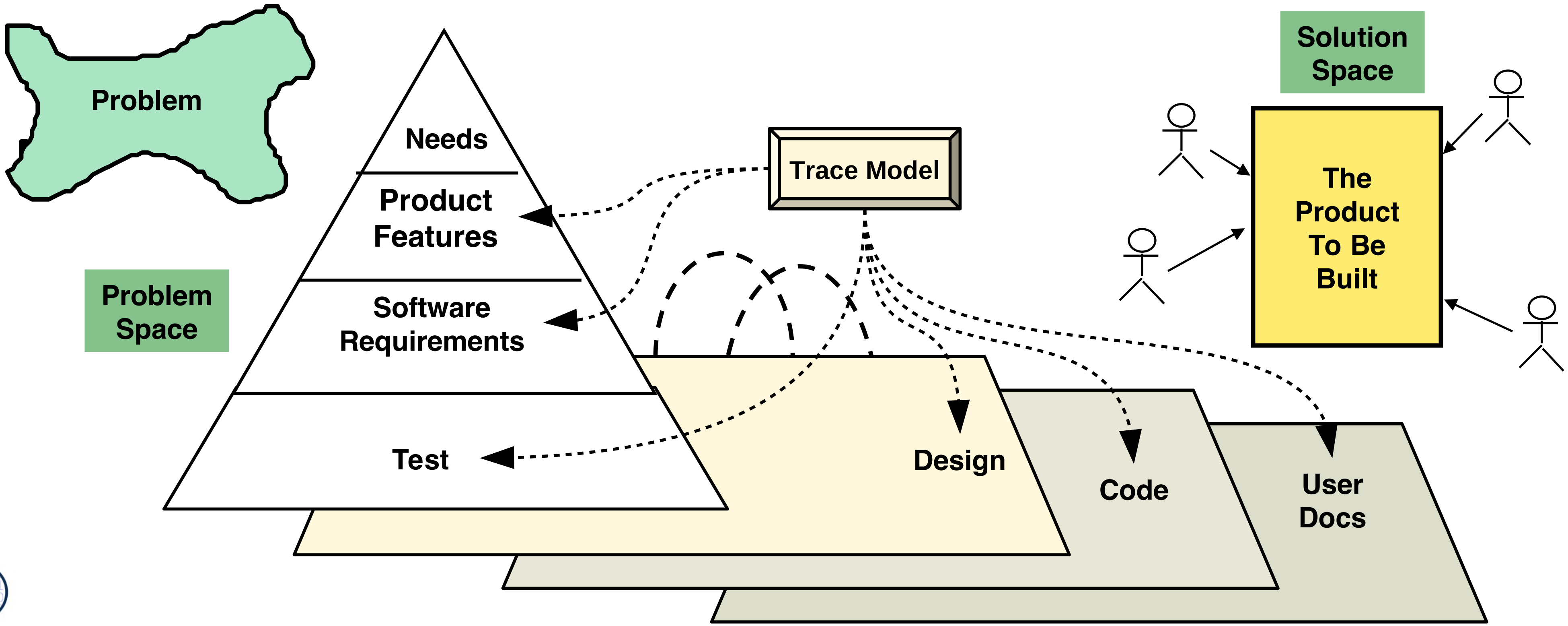


32.4 Traceability in Practical Requirements Management Tools

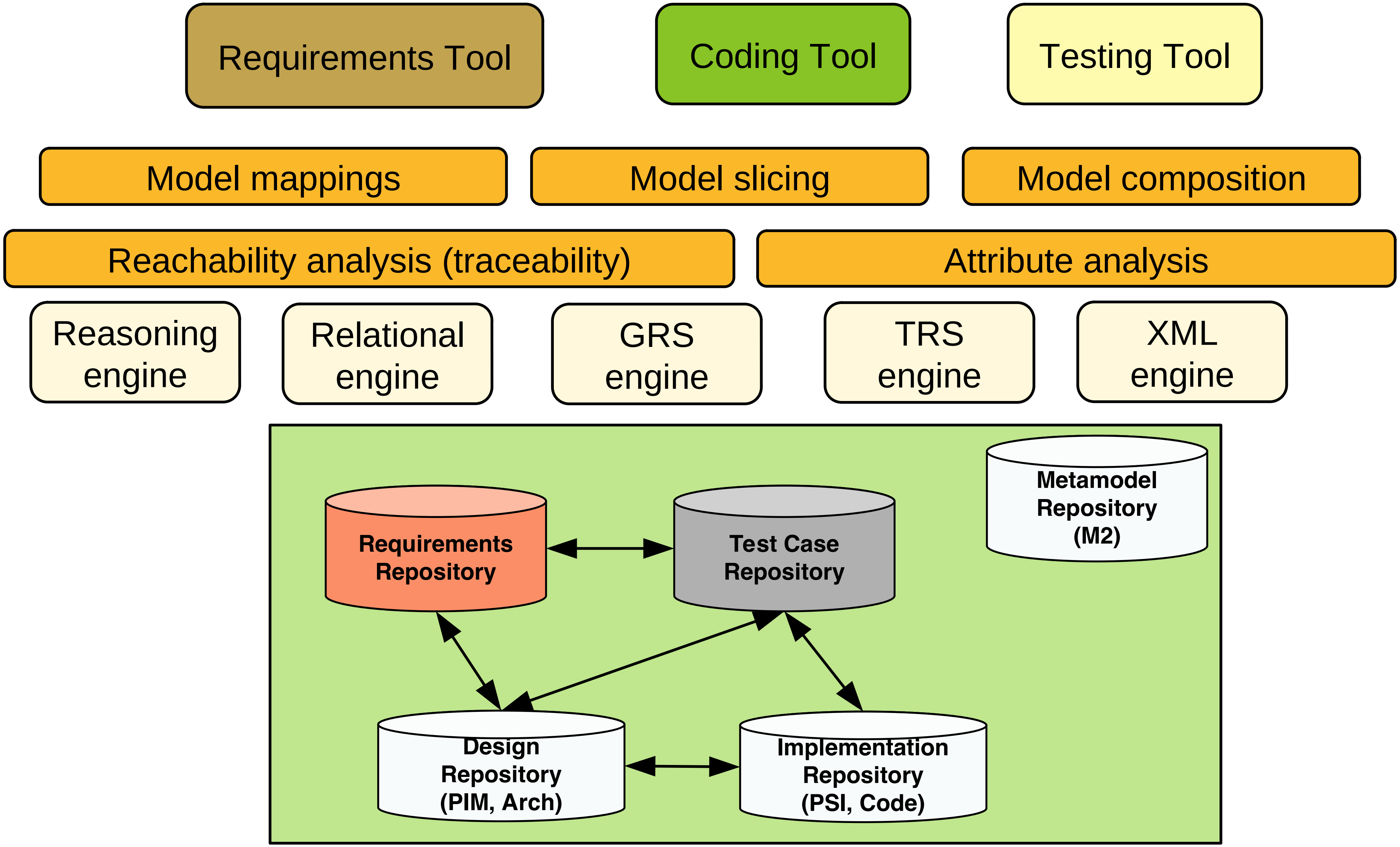
omitted in 2021/22

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)

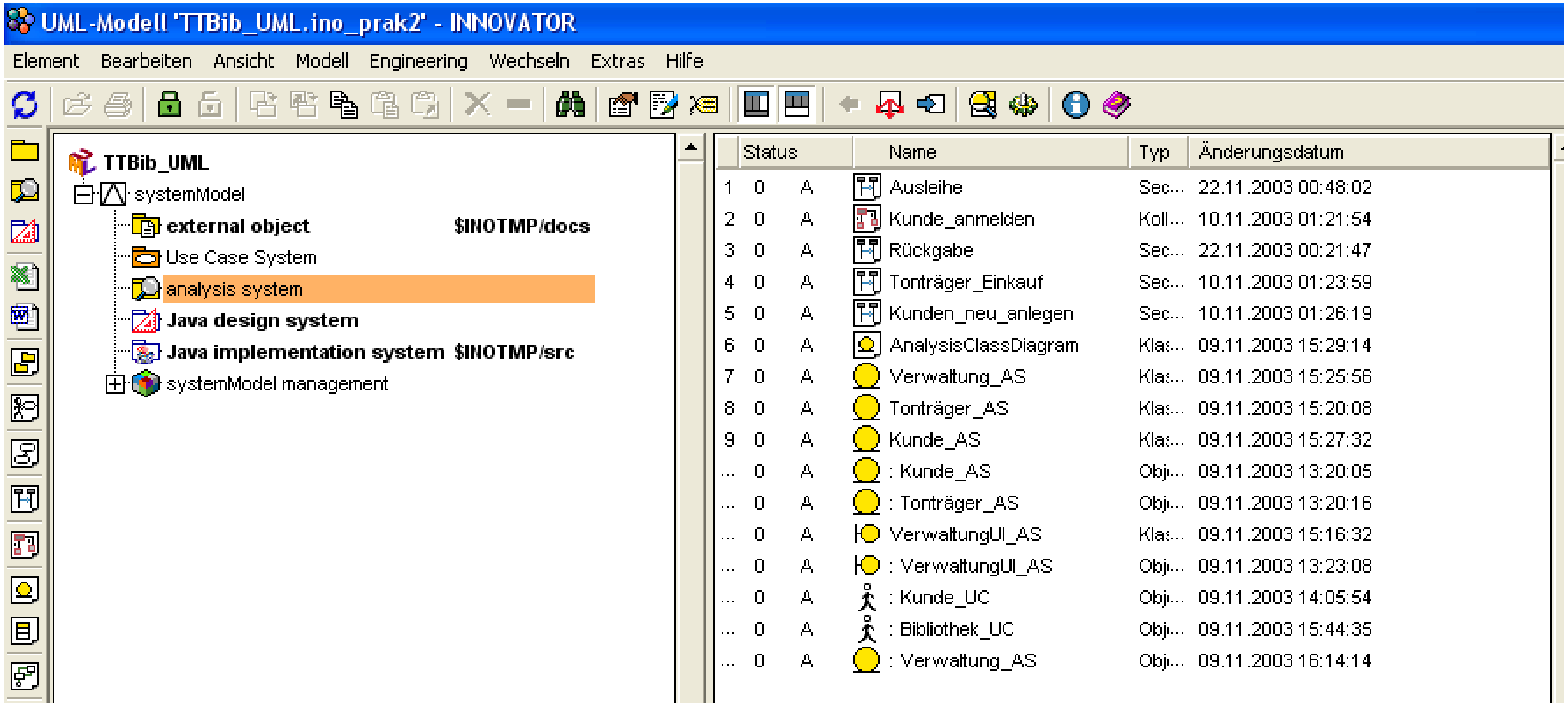


Deficiencies of Current RE Methods

- 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

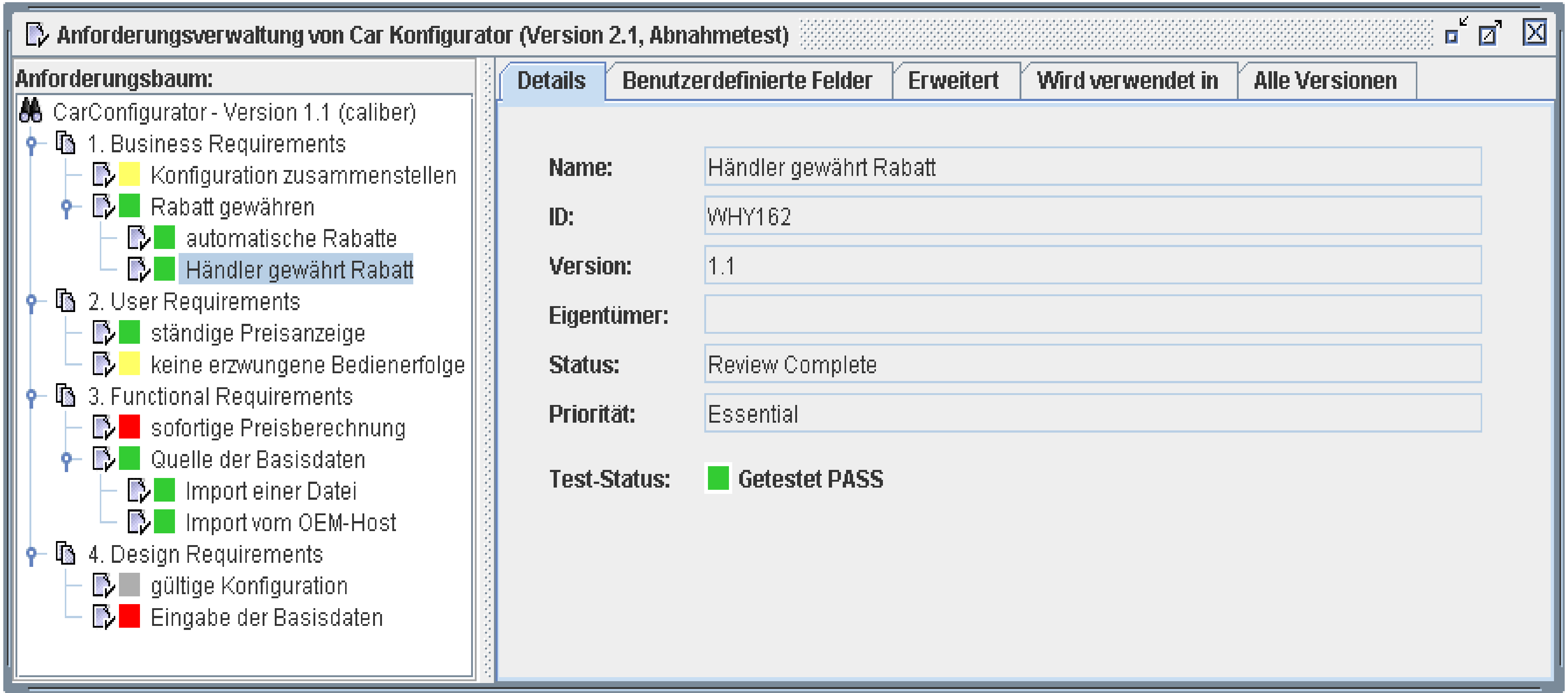
- With a **direct model mapping**, a requirements model can be linked
  - to a test case specification
  - to a documentation
  - to an architectural specification
  - via the architectural specification, to the classes and procedures in the code

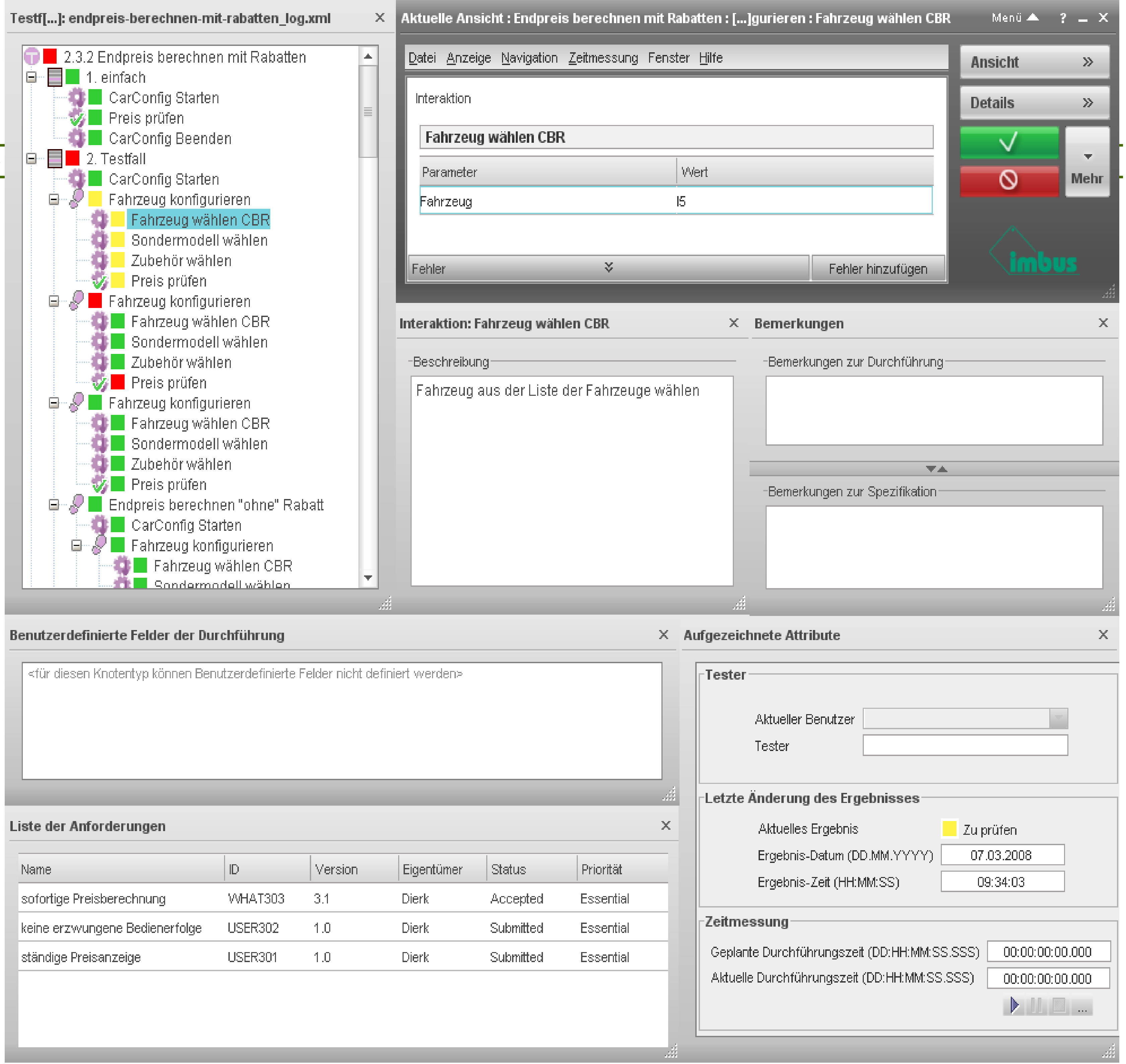
Example: imbus TestBench

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




Testf[...]: endpreis-berechnen-mit-rabatten_log.xml
2.3.2 Endpreis berechnen mit Rabatten
1. einfach
CarConfig Starten
Preis prüfen
CarConfig Beenden
2. Testfall
CarConfig Starten
Fahrzeug konfigurieren
Fahrzeug wählen CBR
Sondermodell wählen
Zubehör wählen
Preis prüfen
Fahrzeug konfigurieren
Fahrzeug wählen CBR
Sondermodell wählen
Zubehör wählen
Preis prüfen
Fahrzeug konfigurieren
Fahrzeug wählen CBR
Sondermodell wählen
Zubehör wählen
Preis prüfen
Endpreis berechnen "ohne" Rabatt
CarConfig Starten
Fahrzeug konfigurieren
Fahrzeug wählen CBR
Aktuelle Ansicht : Endpreis berechnen mit Rabatten : [...]gurieren : Fahrzeug wählen CBR
Menü
Datei Anzeige Navigation Zeitmessung Fenster Hilfe
Ansicht
Details
Mehr
Interaktion
Fahrzeug wählen CBR
Parameter
Wert
Fahrzeug
I5
Fehler
Fehler hinzufügen
imbus
Interaktion: Fahrzeug wählen CBR
Beschreibung
Fahrzeug aus der Liste der Fahrzeuge wählen
Bemerkungen
Bemerkungen zur Durchführung
Bemerkungen zur Spezifikation
Benutzerdefinierte Felder der Durchführung
<für diesen Knotentyp können Benutzerdefinierte Felder nicht definiert werden>
Liste der Anforderungen
Name | ID | Version | Eigentümer | Status | Priorität
sofortige Preisberechnung | WHAT303 | 3.1 | Dierk | Accepted | Essential
keine erzwungene Bedienerfolge | USER302 | 1.0 | Dierk | Submitted | Essential
ständige Preisanzeige | USER301 | 1.0 | Dierk | Submitted | Essential
Aufgezeichnete Attribute
Tester
Aktueller Benutzer
Tester
Letzte Änderung des Ergebnisses
Aktuelles Ergebnis
Zu prüfen
Ergebnis-Datum (DD.MM.YYYY)
07.03.2008
Ergebnis-Zeit (HH:MM:SS)
09:34:03
Zeitmessung
Geplante Durchführungszeit (DD:HH:MM:SS.SSS)
00:00:00:00.000
Aktuelle Durchführungszeit (DD:HH:MM:SS.SSS)
00:00:00:00.000

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

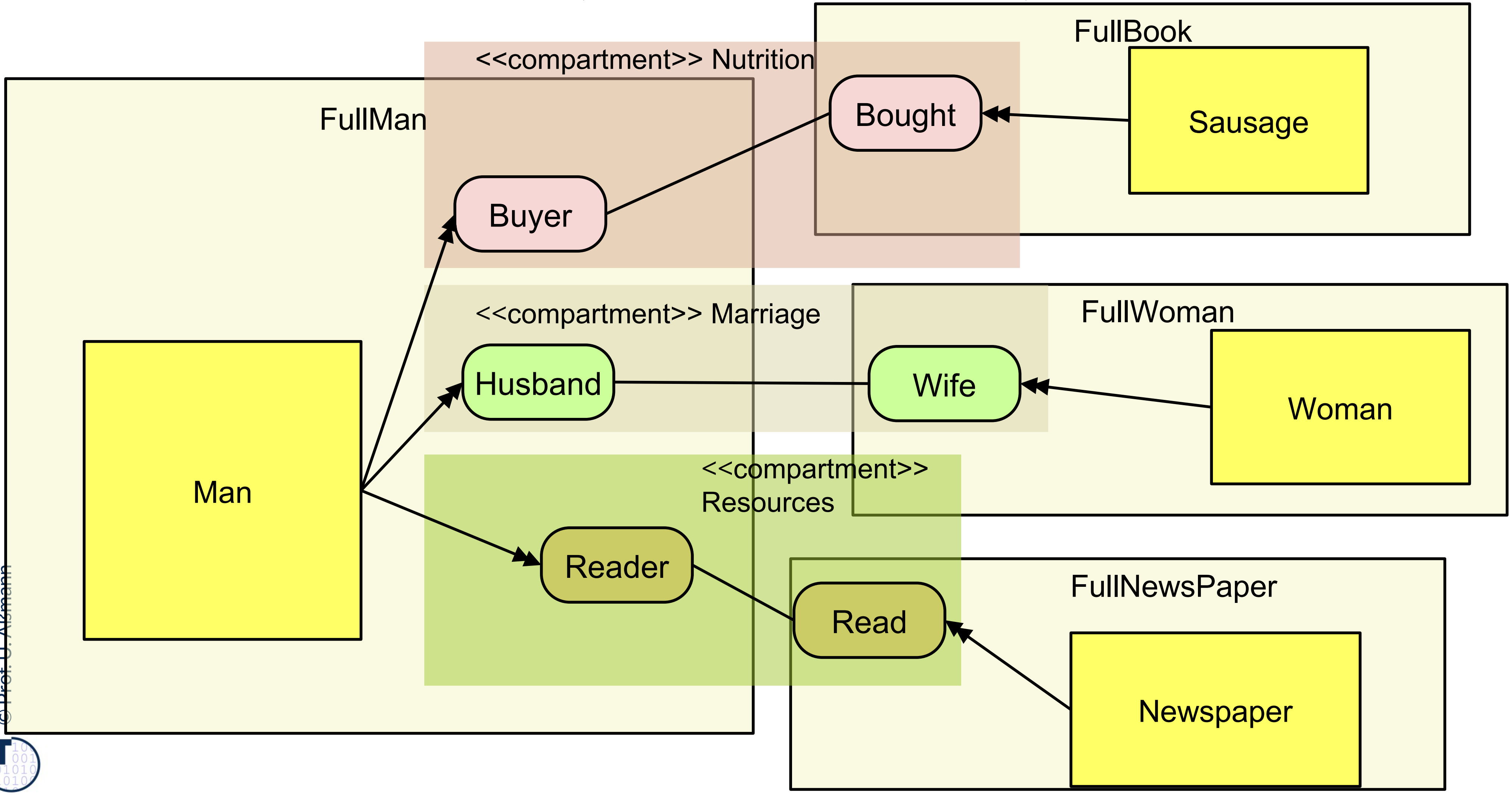
32.5 The MDA Macromodel of RoSI (RoSI-MDA): Representing Trace Mappings as Role-Playing

- 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
- RoSI-MDA is homogeneous Macromodel

Remember: The Steimann Factorization of Natural and Role Types

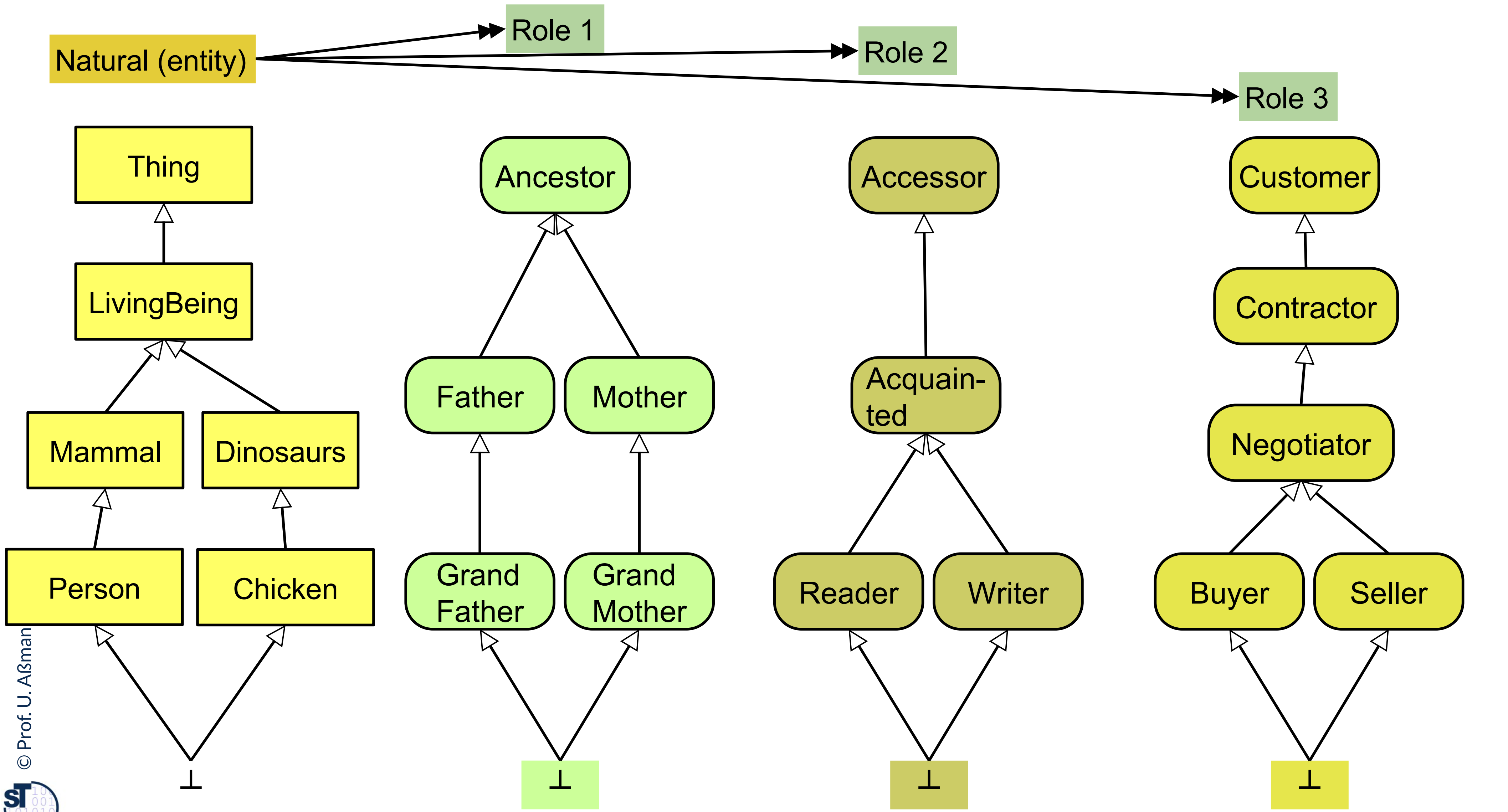
Splitting a full type into its *natural* and *role-type* components

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



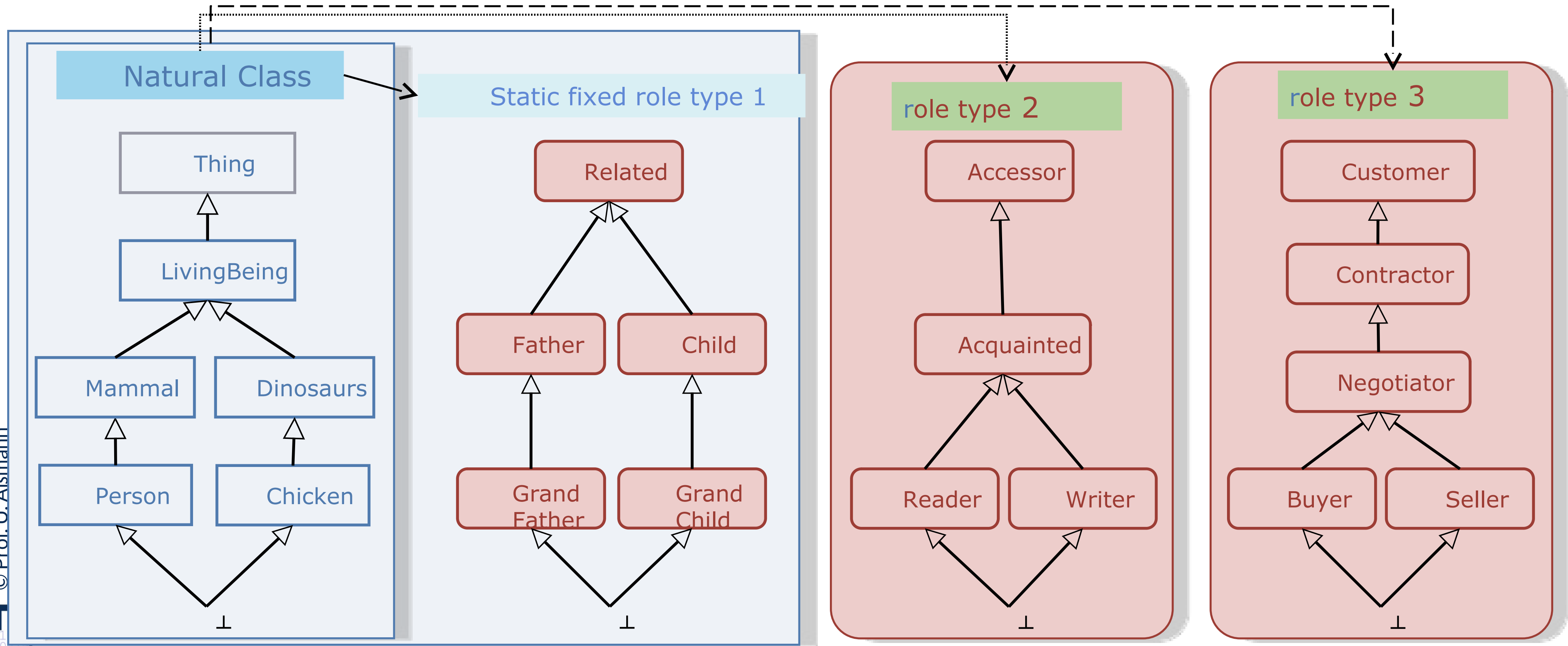
Remember: Full Type is from Inheritance Product Lattice

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



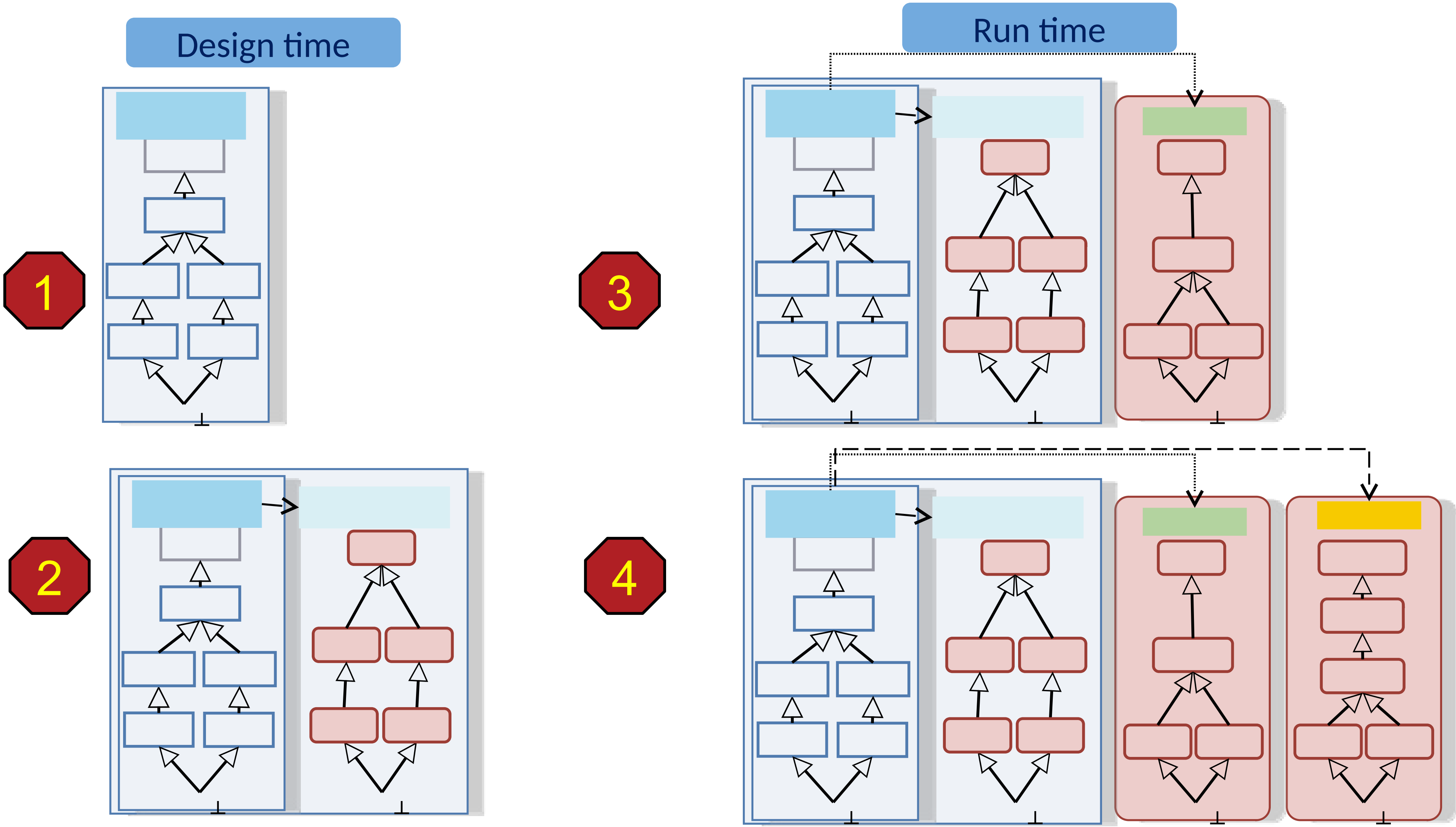
Scalable Bindung Time of Contexts with the Factorization

- **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



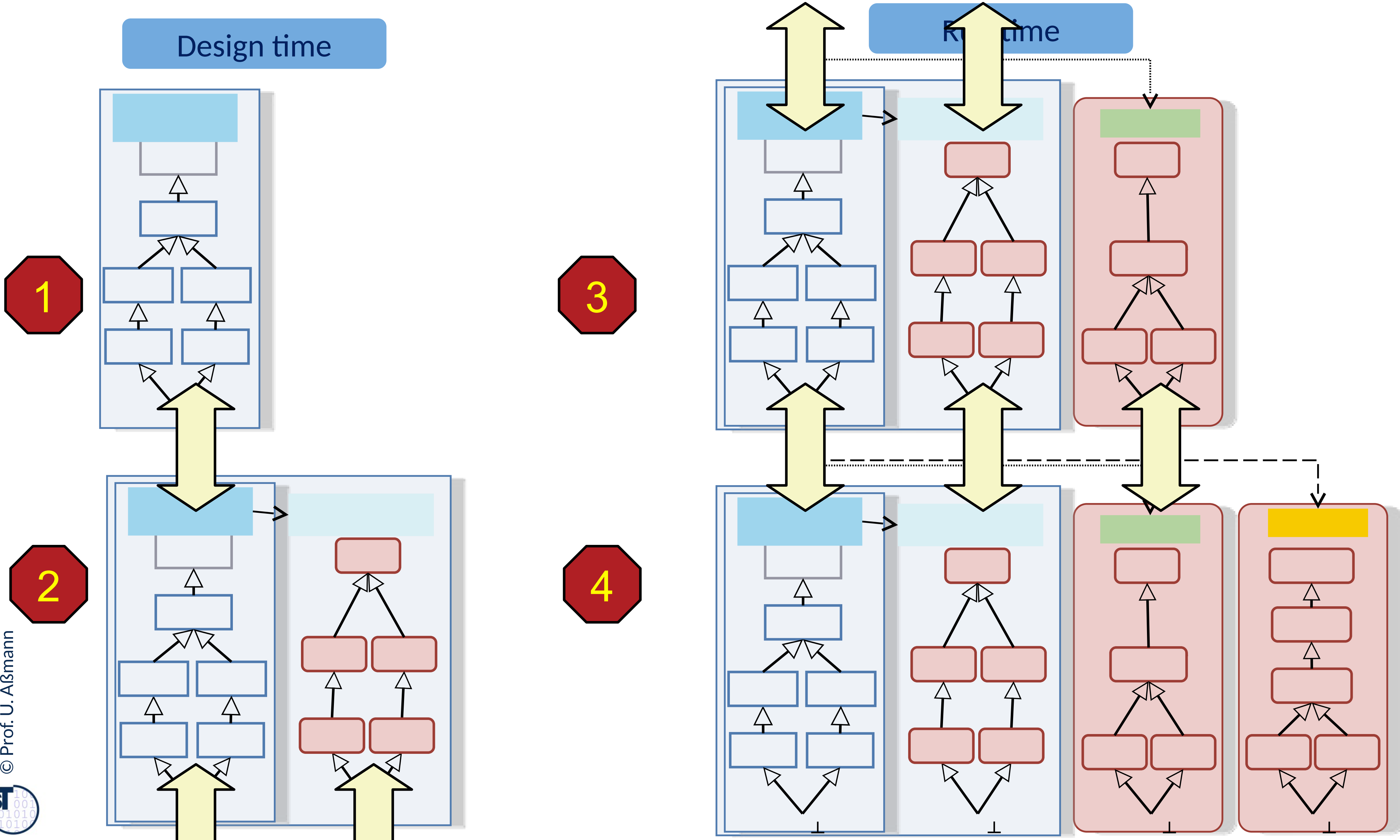
RoSI Macromodel (RoSI-MDA): Refinement by Role Allocation

- ***Refinement*** by allocation of further roles – static roles at design time, dynamic roles at runtime
- In RoSI-MA, the role-play relation is subset of the traceability relation



RoSI-MDA: Traceability in Refinement by Role Allocation

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



RoSI Macromodel (RoSI-MDA): Cross-Layer Role-Based Refinement in the Software Life Cycle

- Refinement by allocation of roles provides **simple traceability** because Natural objects STAY the same
- Trace mapping *is* role-play relation joined with context-role matrix
- 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

	Natural	Fixed Role 1	Fixed Role 2	Fixed Role 3	Fixed Role 4	Dynamic role 1	Dynamic role 2	Dynamic role 3
Domain Model	Person							
Requirements	Person	Customer						
Design	Person	Customer	Customer Design					
PSM	Person	Customer	Customer Design	Platform-specific Behavior				
Implementation	Person	Customer	Customer Design	Platform-specific Behavior	Full static behavior			
Run time context 1	Person	Customer	Customer Design	Platform-specific Behavior	Full static behavior	Behavior in Context 1		
Run time context 2	Person	Customer	Customer Design	Platform-specific Behavior	Full static behavior	Behavior in Context 1	Behavior in Context 2	
Run time context 3	Person	Customer	Customer Design	Platform-specific Behavior	Full static behavior	Behavior in Context 1	Behavior in Context 2	Behavior in Context 3

Advantages of RoSI-MDA (Role-Based MDA)

- Very simple, component MDA with easy traceability:
  - Cores of objects map 1:1 from CIM via PIM and PSM into the application PSI (context-role matrix)
  - Variability via new roles for PIM, PSM, PSI
  - "object fattening" through the MDA
- Projection (get) and reintegration (put) is simple for MDA-SUM

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 context-role-based model better for traceability?
- How does JastAdd aspects achieve MDA refinement?
  - How is traceability achieved?
  - How model synchronisation?
- How does RoSI-MDA 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?