4. Classical Metamodelling in Technical Spaces

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
Technische Universität Dresden
http://st.inf.tu-dresden.de/teaching/most

Version 15-1.3, 23.10.15

Obligatory Literature

- Kurtev, I., Bezivin, J., Aksit, M.: Technological Spaces: An Initial Appraisal. In: International Symposium on Distributed Objects and Applications, DOA Federated Conferences, Industrial track, Irvine. (2002)
- Model-based Technology Integration with the Technical Space Concept. Jean Bezivin and Ivan Kurtev. Metainformatics Symposium, 2005.
- Jean Bézivin. Model Driven Engineering: An Emerging Technical Space. In R. Lämmel, J. Saraiva, and J. Visser (Eds.): GTTSE 2005, LNCS 4143, pp. 36 – 64, 2006. Springer.
- Ed Seidewitz. What models mean. IEEE Software, 20:26-32, September 2003.
    - http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1231147&tag=1

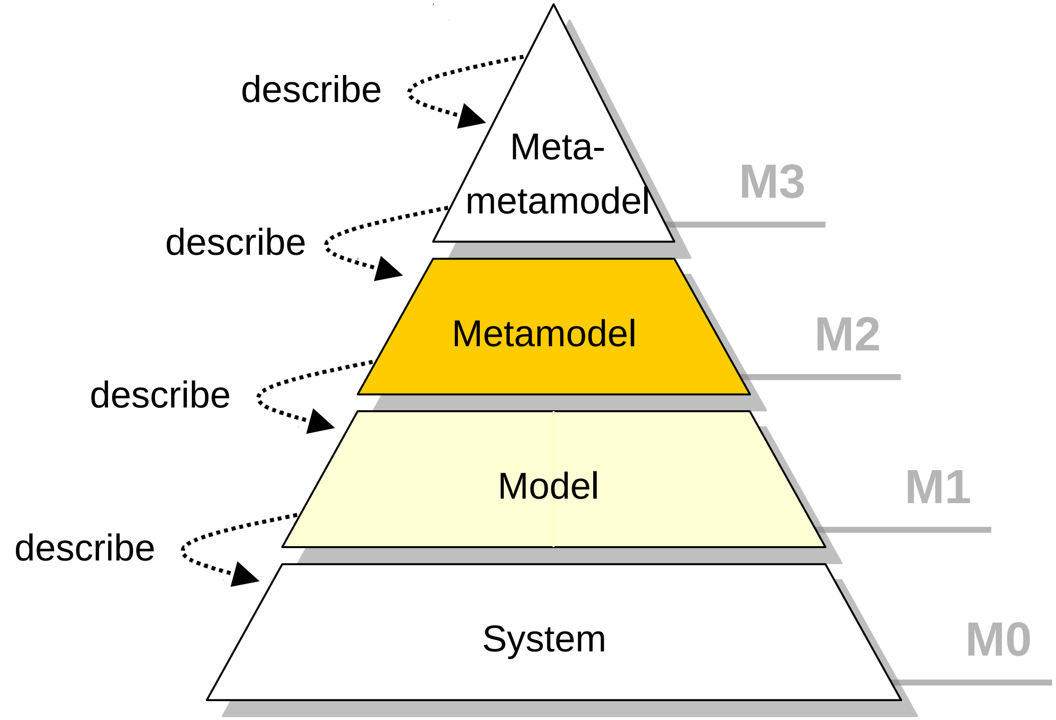
Other Literature

- Gaševic, Dragan, Djuric, Dragan, Devedžic, Vladan. Model Driven Engineering and Ontology Development, 2nd ed., 2009, ISBN 978-3-642-00281-6
  - http://www.springer.com/computer/swe/book/978-3-642-00281-6?cm_mmc=Google-_-Book%20Search-_-Springer-_-0
- [MOF] Metaobject Facility. OMG. 1.4 and 2.0. www.omg.org
- [Nill] C. Nill. Analysis and Design Modeling Using Metaphorical Modeling Entities. A Modeling Language for the Tools and Materials Approach. Diplomarbeit Technische Universität Dresden, 2006.
- [Atkinson/Kühne] Colin Atkinson and Thomas Kühne. Model-driven development: A metamodeling foundation. IEEE Software, 20(5):36-41, 2003.
- [Favre] Jean-Marie Favre. Foundations of model (driven) (reverse) engineering: Models. Technical report, ADELE Team, Laboratoire LSR-IMAG Université Joseph Fourier, Grenoble, France, 2004. vol. 1-3.
- [Flatscher] Rony Flatscher. Metamodeling in EIA/CDIF - meta-metamodel and metamodels. ACM Trans. Model. Comput. Simul, 12(4):322-342, 2002.
- [Kendall] D. T. Chang and E. Kendall. Metamodels for RDF Schema and OWL. Proceedings of the First International Workshop on the Model-Driven Semantic Web (MDSW 2004), Monterey, USA, September 21, 2004.

4.1 Metamodelling in the Classical Metapyramid

The Metamodel Hierarchy (Metapyramid, Metahierarchy)

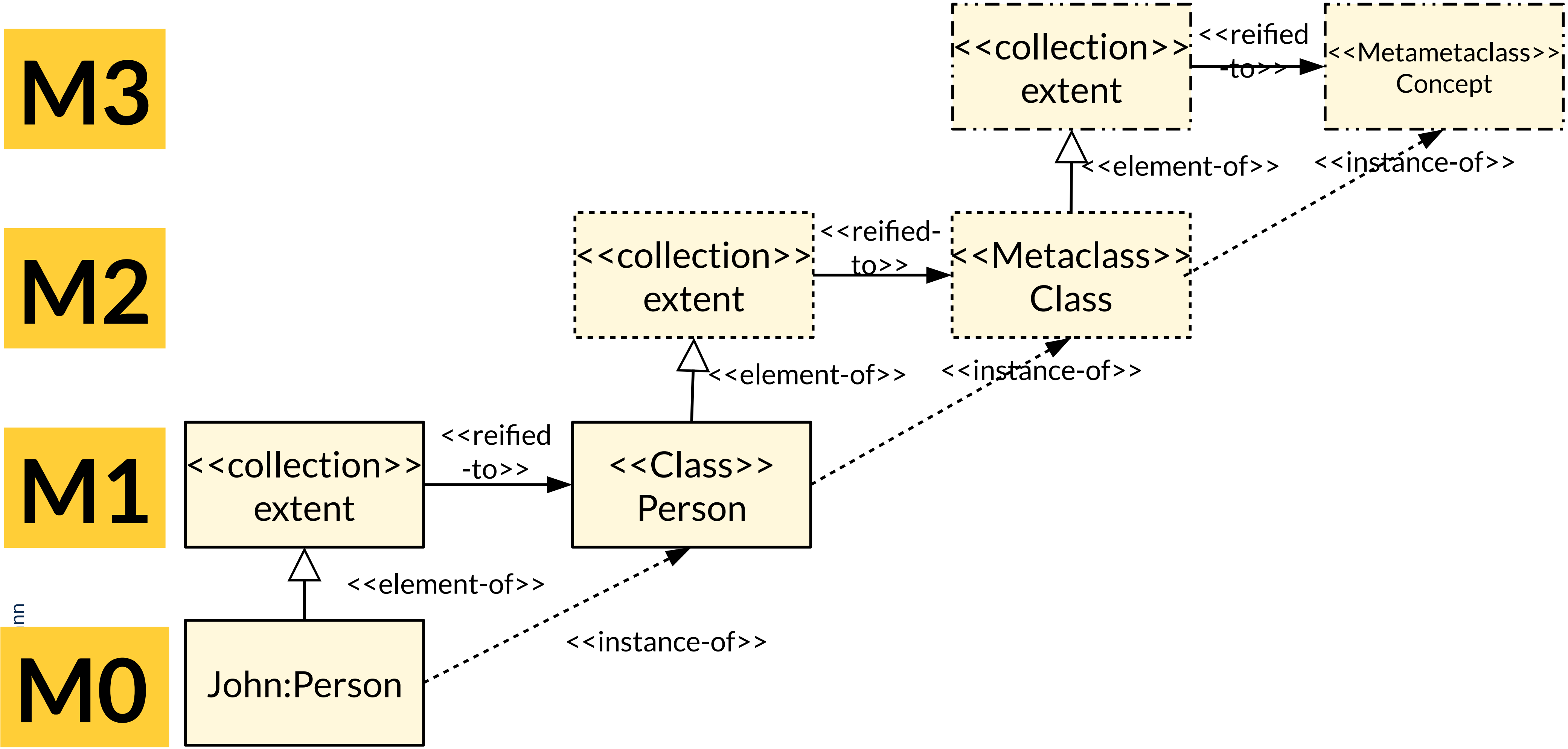
- Models are widely used in engineering disciplines
- Need for **tool support** that enables model-editing
- Domain experts want **domain specific languages** (DSL)
  → domain specific models with types from the domain
- Do not build model editors from scratch each time
  → **reuse** functionality
  → use meta-information



[F. Klar, TU Darmstadt]

Remember: The Clabject Metahierarchy and Metapyramids

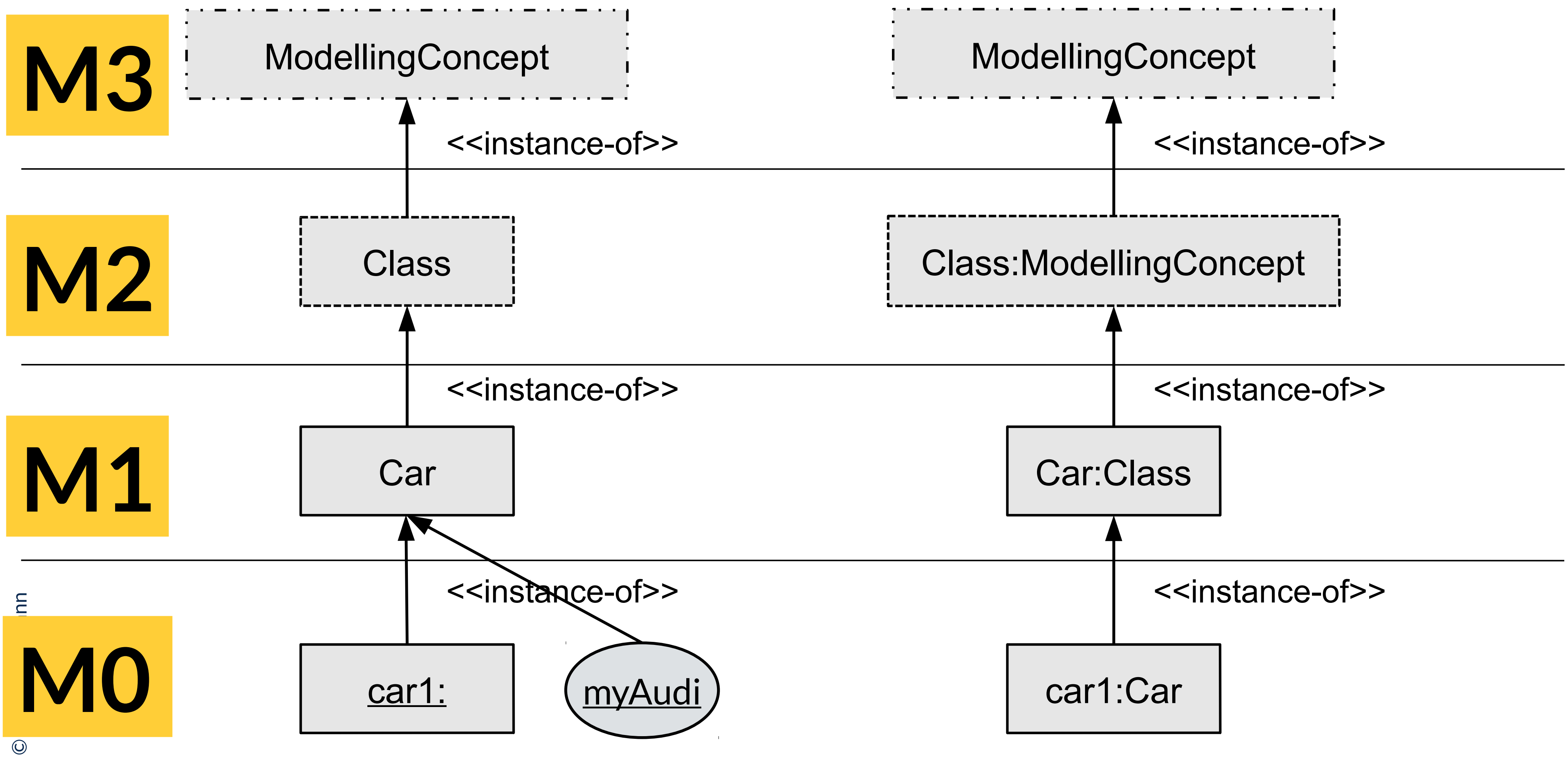
- We call a hierarchy of instance-of relationships a ***metahierarchy.***
- A ***metapyramid*** is a network of instance-of relationships



Notation

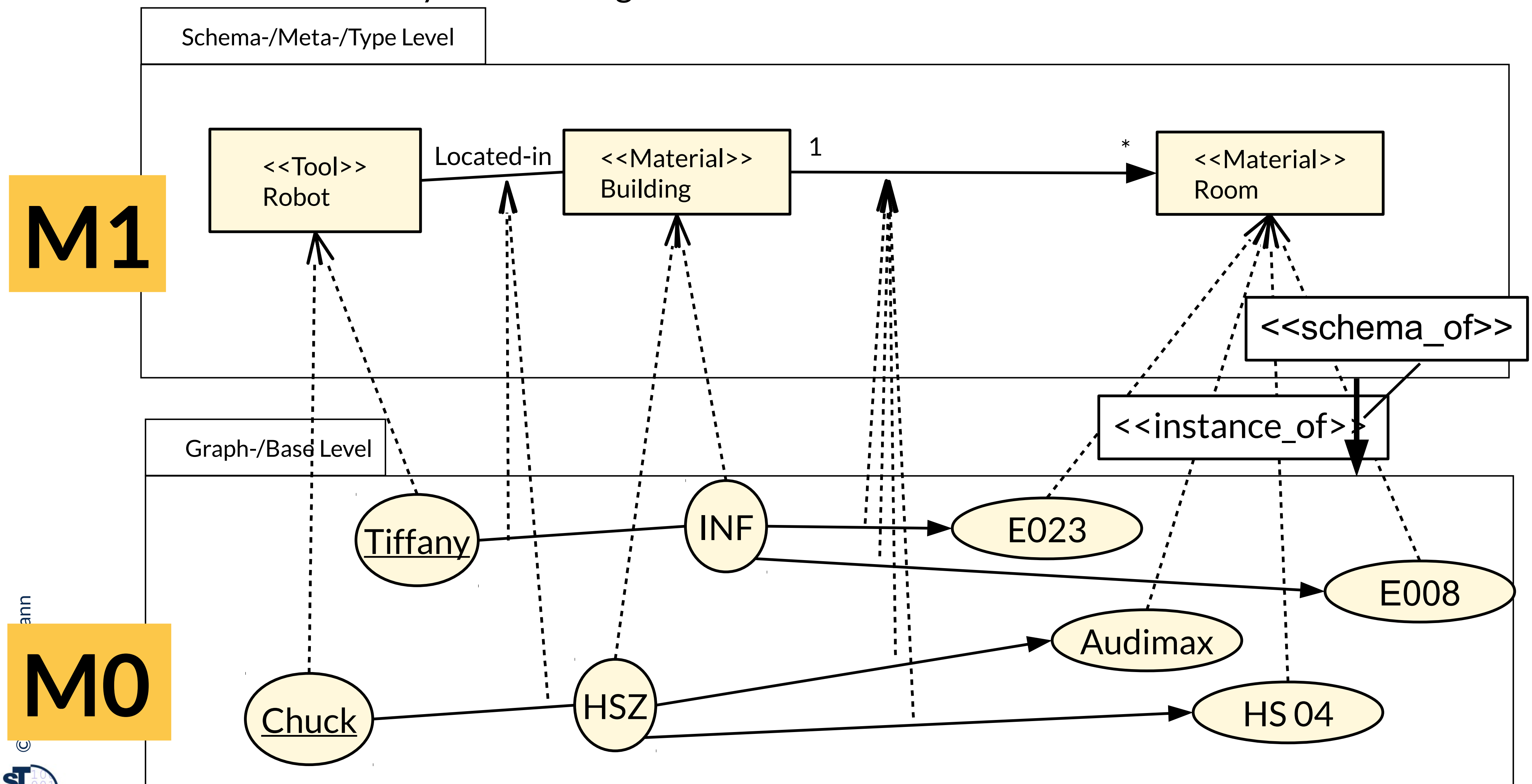
Clabject Hierarchy

- We write metaclasses (clabjects) with dashed lines, metametaclasses (clabjects) with dotted-dashed lines



Rpt.: Type Modeling for Application Types

- On M1, also other sets of the application world can be used as types
- Classes can carry the TAM tags



Models in Software Engineering

Models define abstractions of realities

- **Process models (Workflow models)** define workflows and other processes
- **Domain models** describe a domain of the world, or a problem domain from the world of the customer
- **System models** specify systems or artefacts:
  - **Software models** define the structure of code
  - **Architecture models** define computational units, distribution, runtime issues, design patterns or architectural styles
  - **Data models** define die structure of materials and the data (e.g. relational model)

Metamodels define types for model elements. They define the *structure* of models. Their instances are models.

- **Process metamodels** define concepts for workflows
- **Domain metamodels** define concepts of domains
- **System metamodels** define concepts of systems
- **Programming Language Metamodels** define concepts of programming languages
- **Modeling Language Metamodels** define concepts of modeling languages
- **Domain-specific language (DSL) metamodels** define concepts of DSL
- **Pattern Language Metamodels** define stereotypes for classes
- **Data metamodels** define concepts for materials

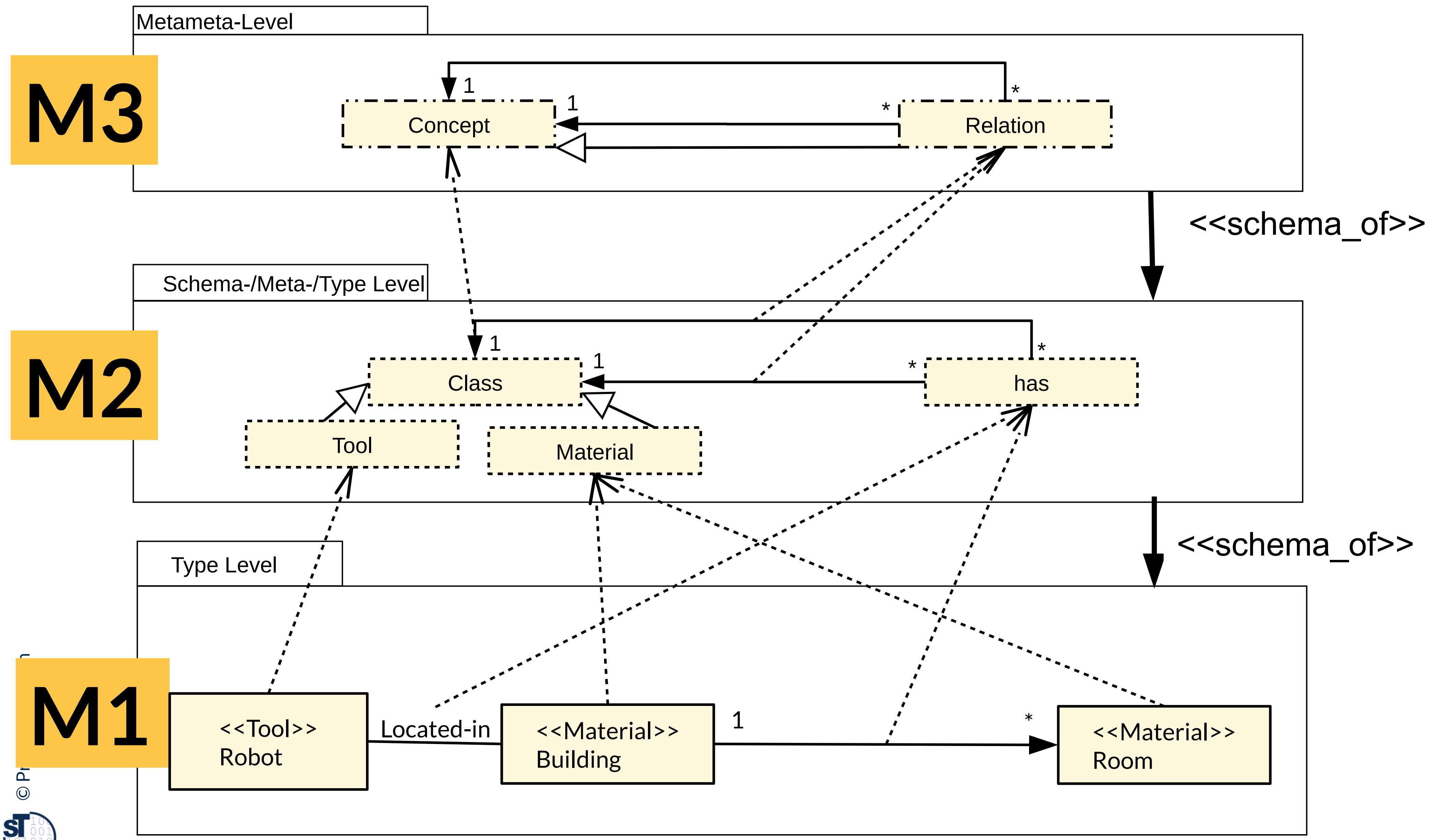
4.2 Metametamodels on M3

The Metametamodel (Metalanguage)

- A ***Metametamodel (MMM, Metalanguage)*** is a structural graph schema of a language
  - Defines types for the concepts of a language (the metaclasses on M2)
  - Contains the modeling concepts for languages
  - Structural – no behavior
  - Contains **wellformedness rules** for the graphs on M2
  - Via its **multiplicity constraints**, the metametamodel defines the form of data structure on M0 (sequence, list, table, tree, link tree, reducible graph, graph)
  - Should be minimalistic

Problem: All tools and materials heavily depend on the MMM of the technical space

Objects, their Clabjects in Models and Metamodels



Tower of Babel Problem

Tragically, no uniform metametamodell has appeared... (tower of babel)

Tools depend on their MMM

[Jan-Pieter Breughel (wikipedia)]

Metametamodels - Overview

- A ***metametamodel*** describes the context-free and -sensitive structure of a **metalanguage**. It can be augmented with wellformedness roles of the metalanguage.

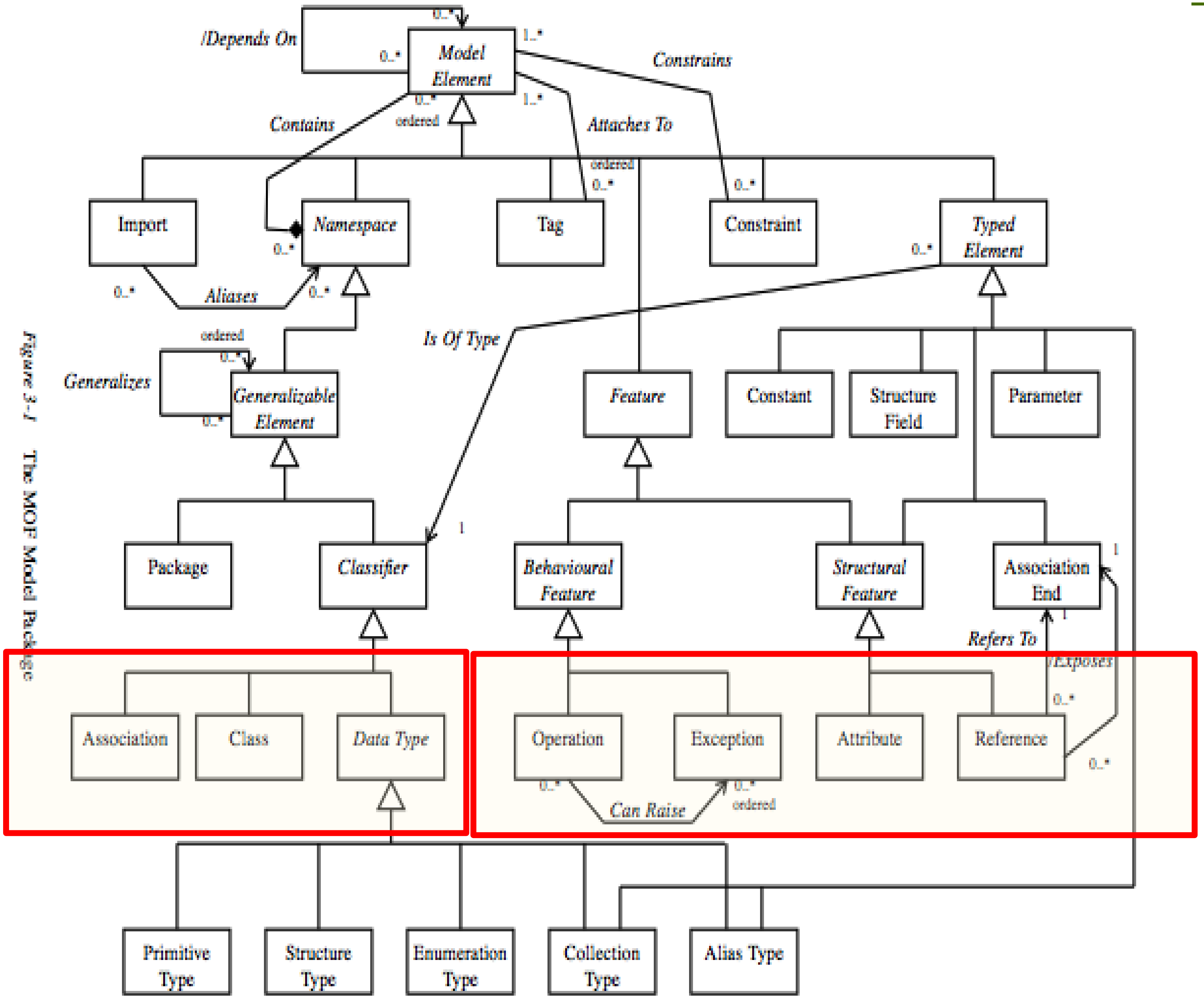
Examples:

- Meta Object Facility – MOF
  - Complete MOF – CMOF
    - **UML core**
  - Essential MOF – EMOF
    - **Ecore** (Eclipse implementation of EMOF)
- GOPRR – Graph Object Property Role Relation (MetaCase.com)
- CROM of ROSI (DFG training group at TU Dresden)
- GXL – Graph eXchange Language

Problem: All tools and materials heavily depend on the MMM of the technical space

4.2.1 Ecore and MOF as Simple Metametamodels

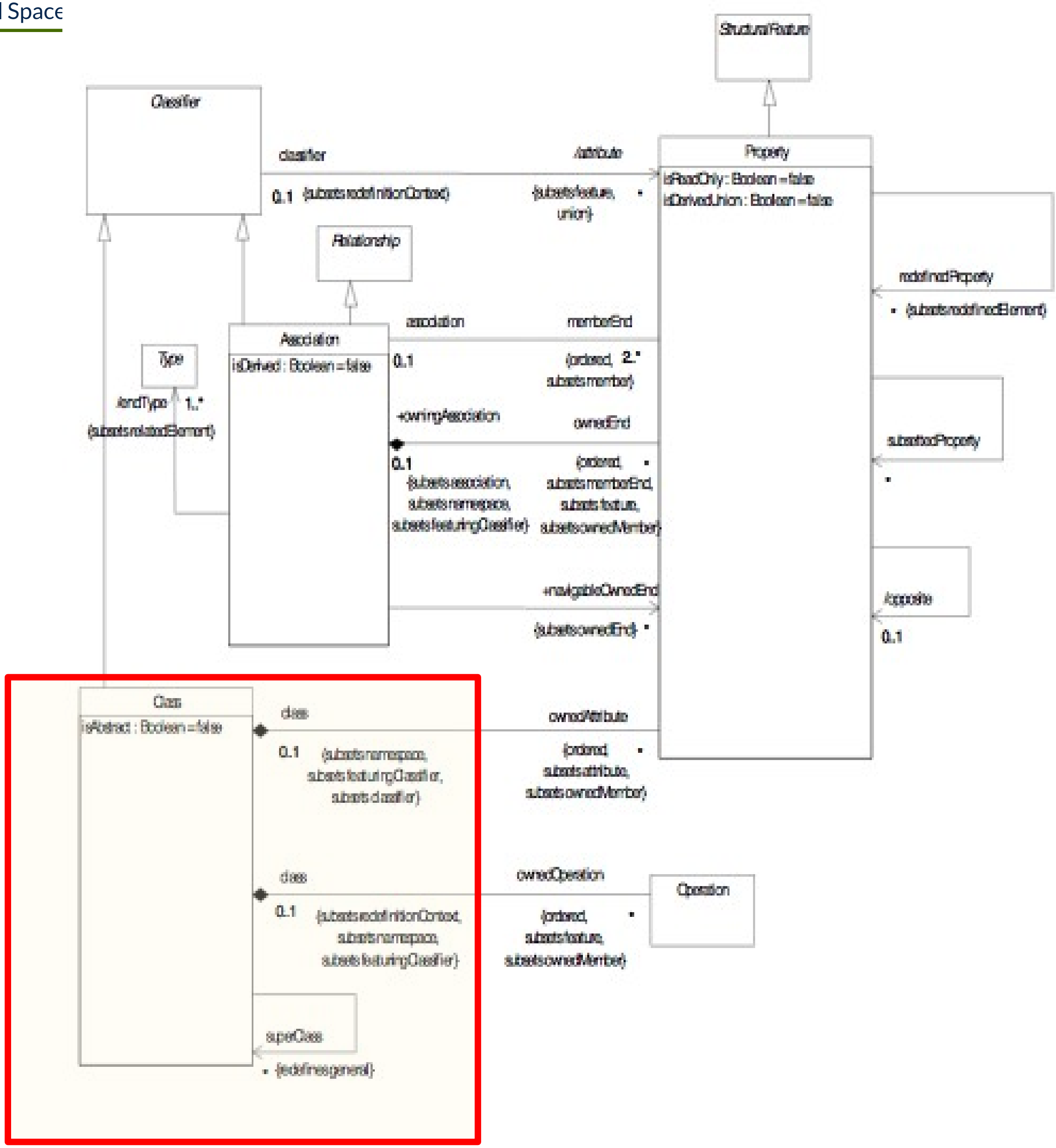
Overview of Metalanguage MOF (CMOF: Complete MOF)



[MOF]

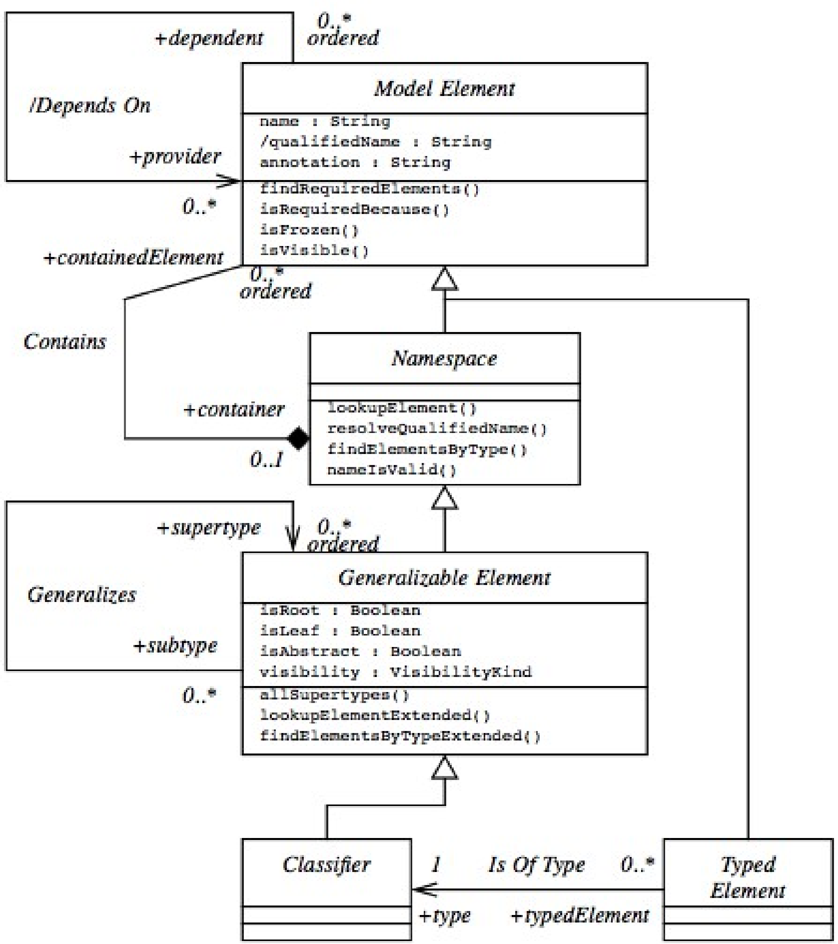
UML Core

- UML core is subset of MOF, and UML-CD
- It is rather minimalistic



[MOF]

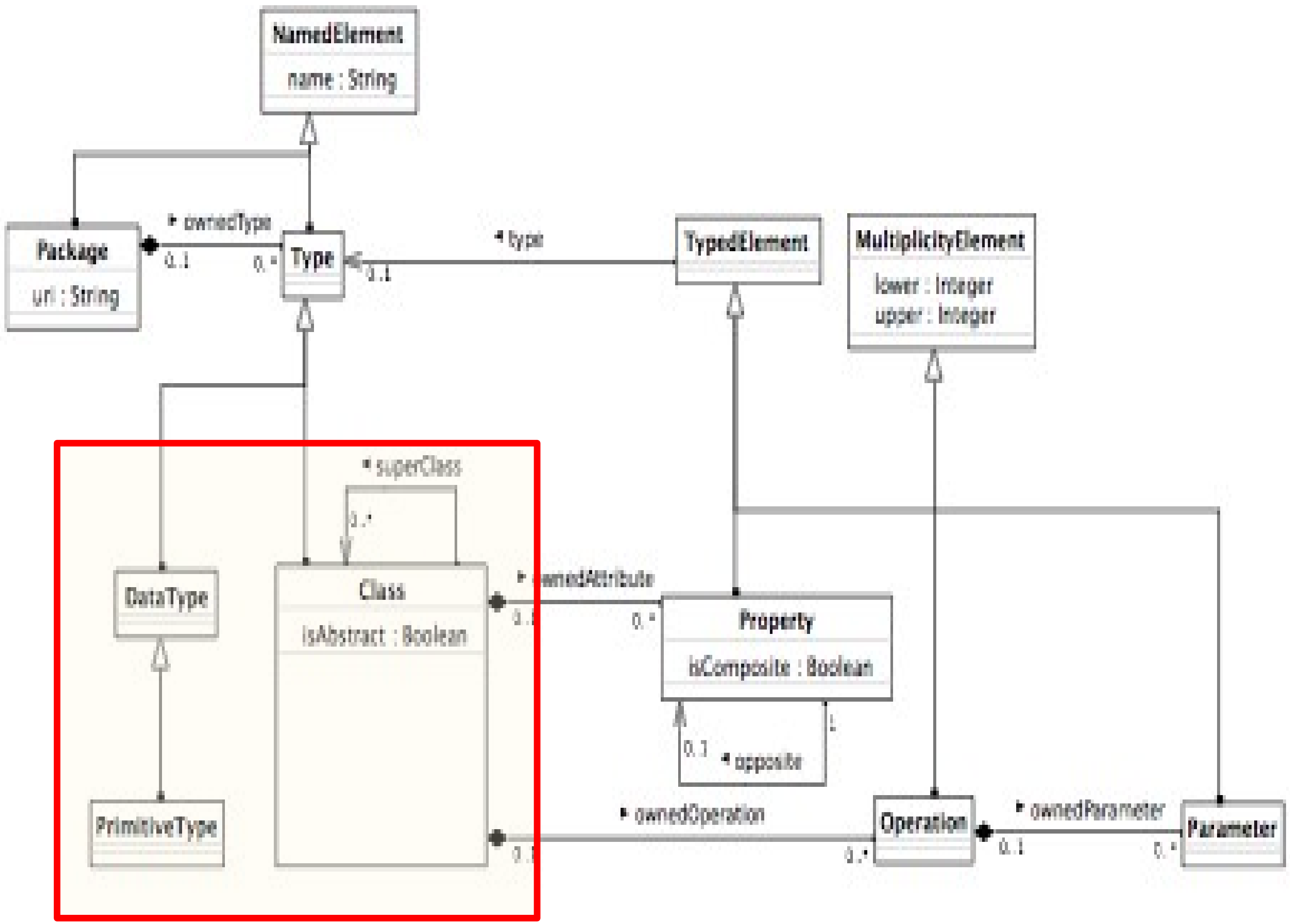
MOF Central Types



- MOF is for modeling of material, tools, automata (not distinguished)

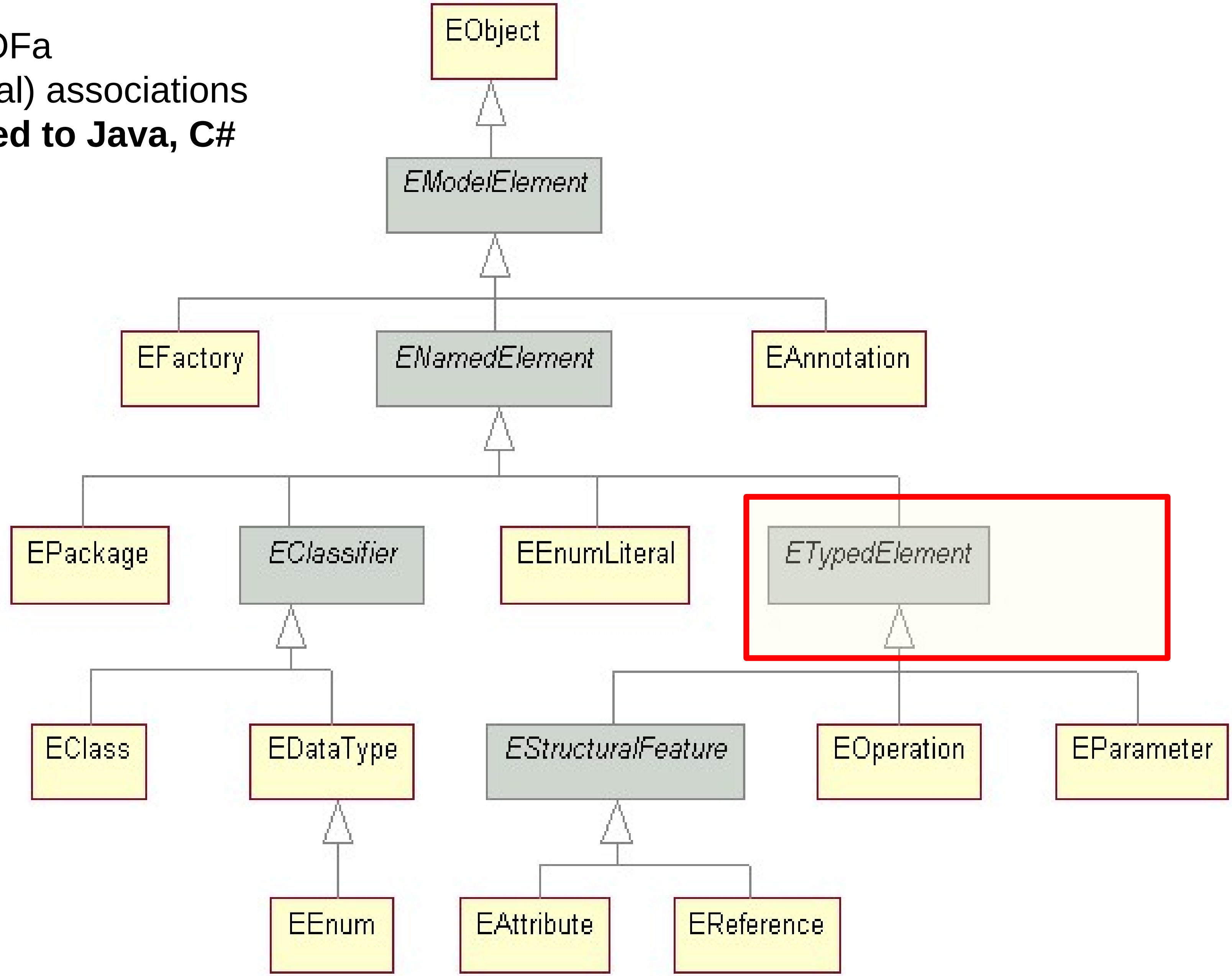
[MOF]

Central MOF Metaclasses with Associations



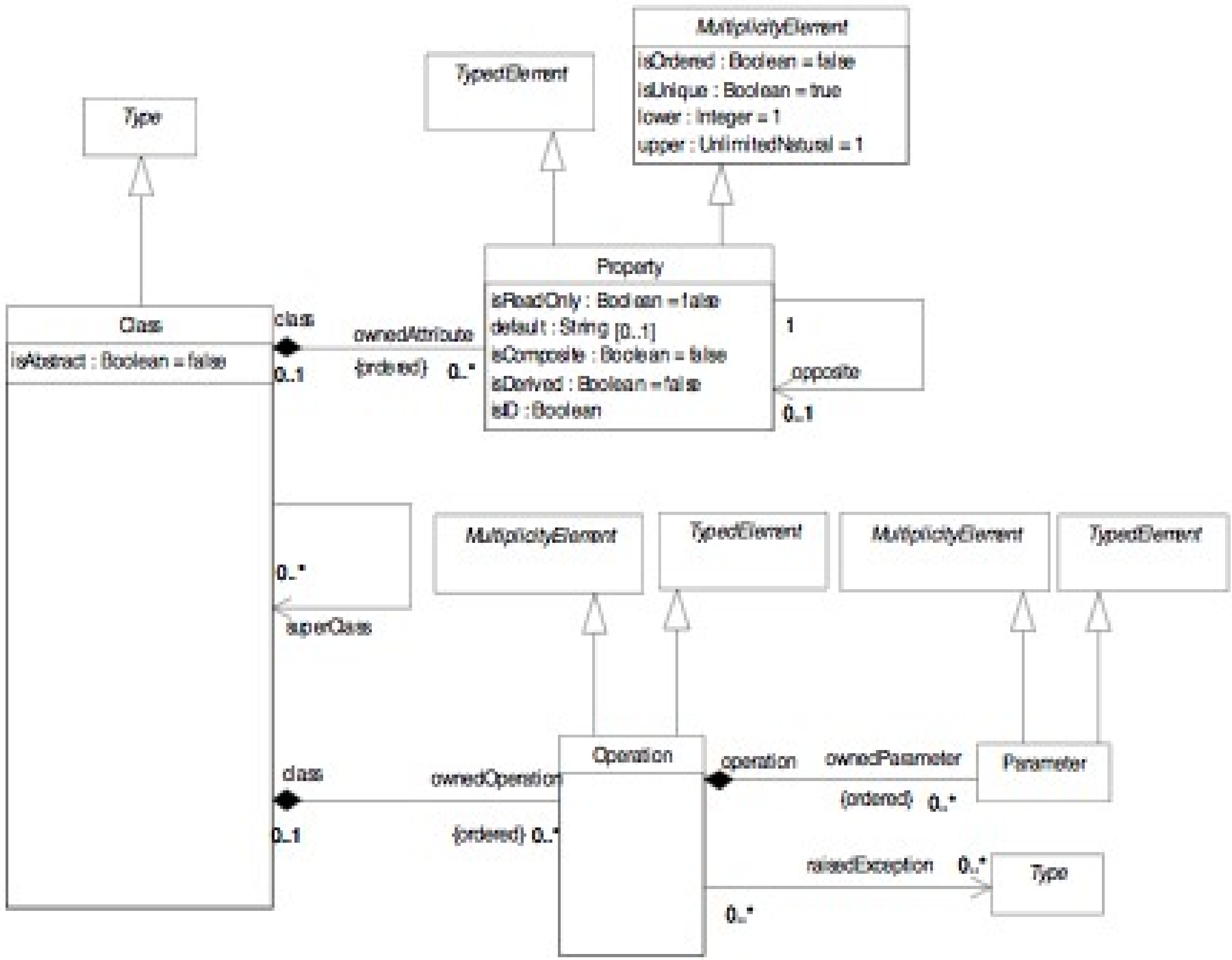
EMOF (Essential MOF)

Subset of CMOFa
No (bidirectional) associations
Can be mapped to Java, C#



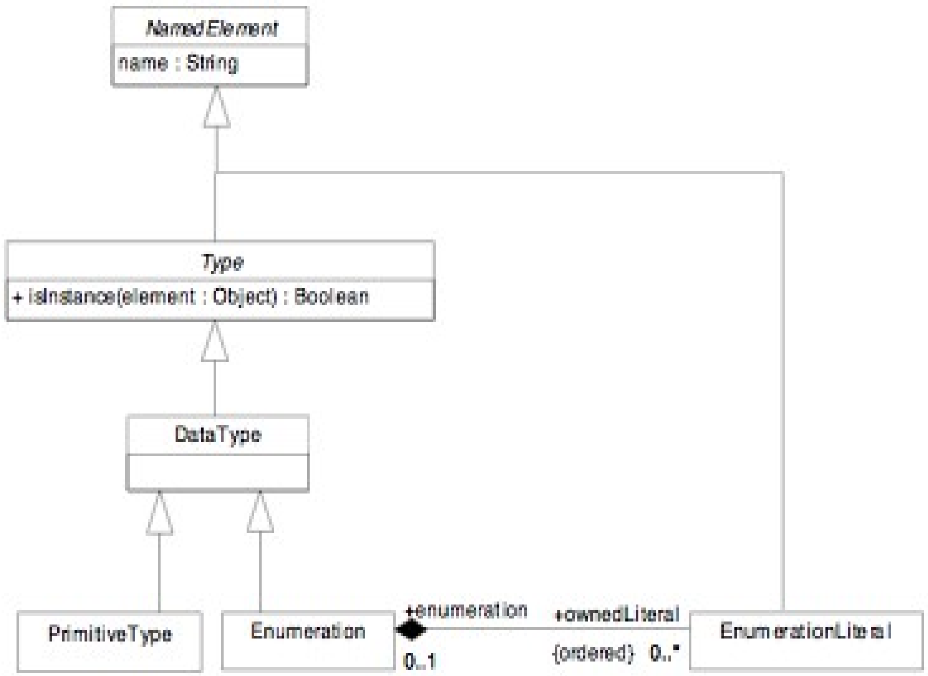
[MOF]

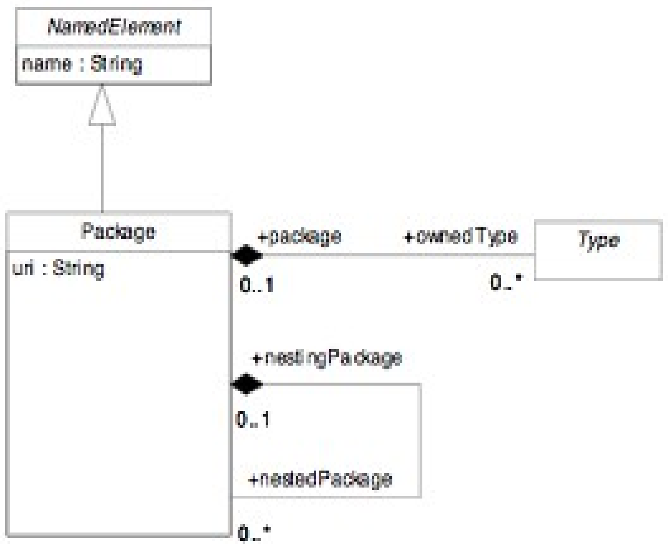
EMOF Classes in Detail



[MOF]

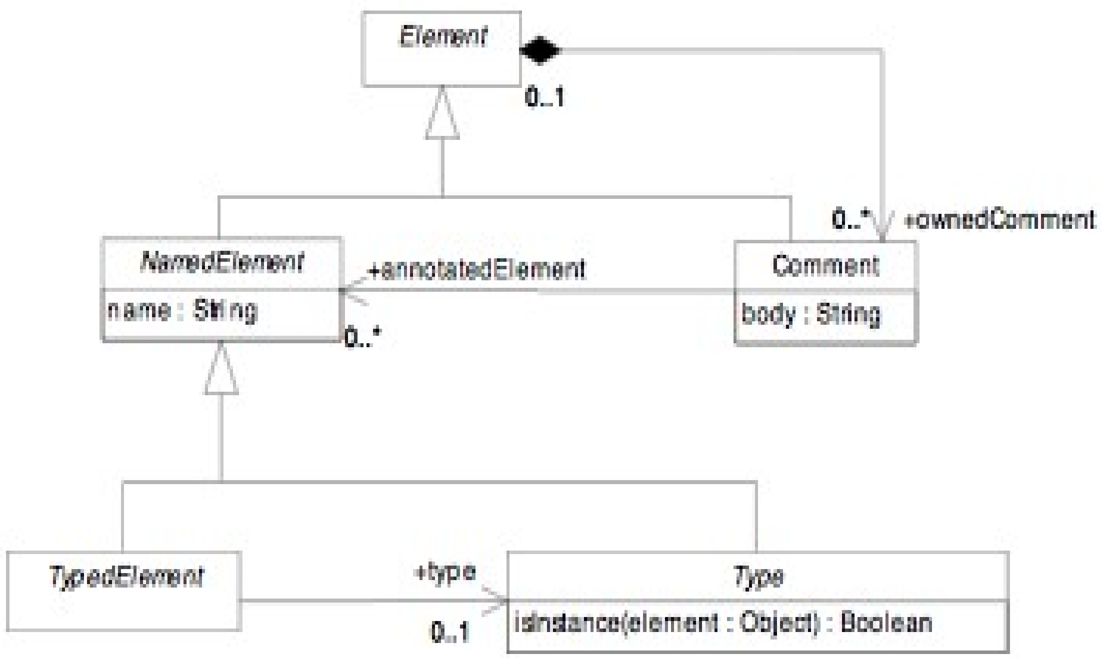
EMOF Data Types and Packages





[MOF]

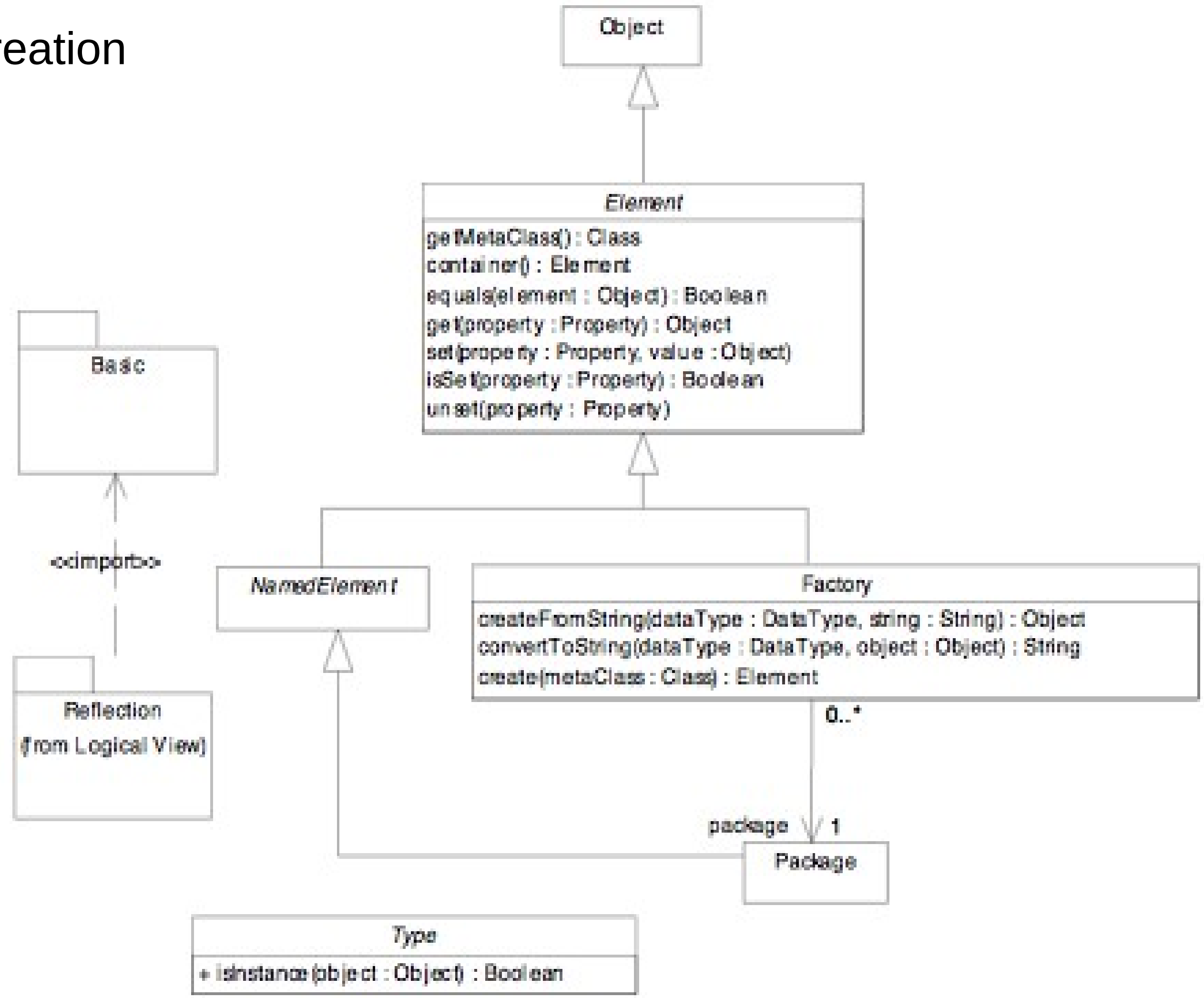
EMOF Types



[MOF]

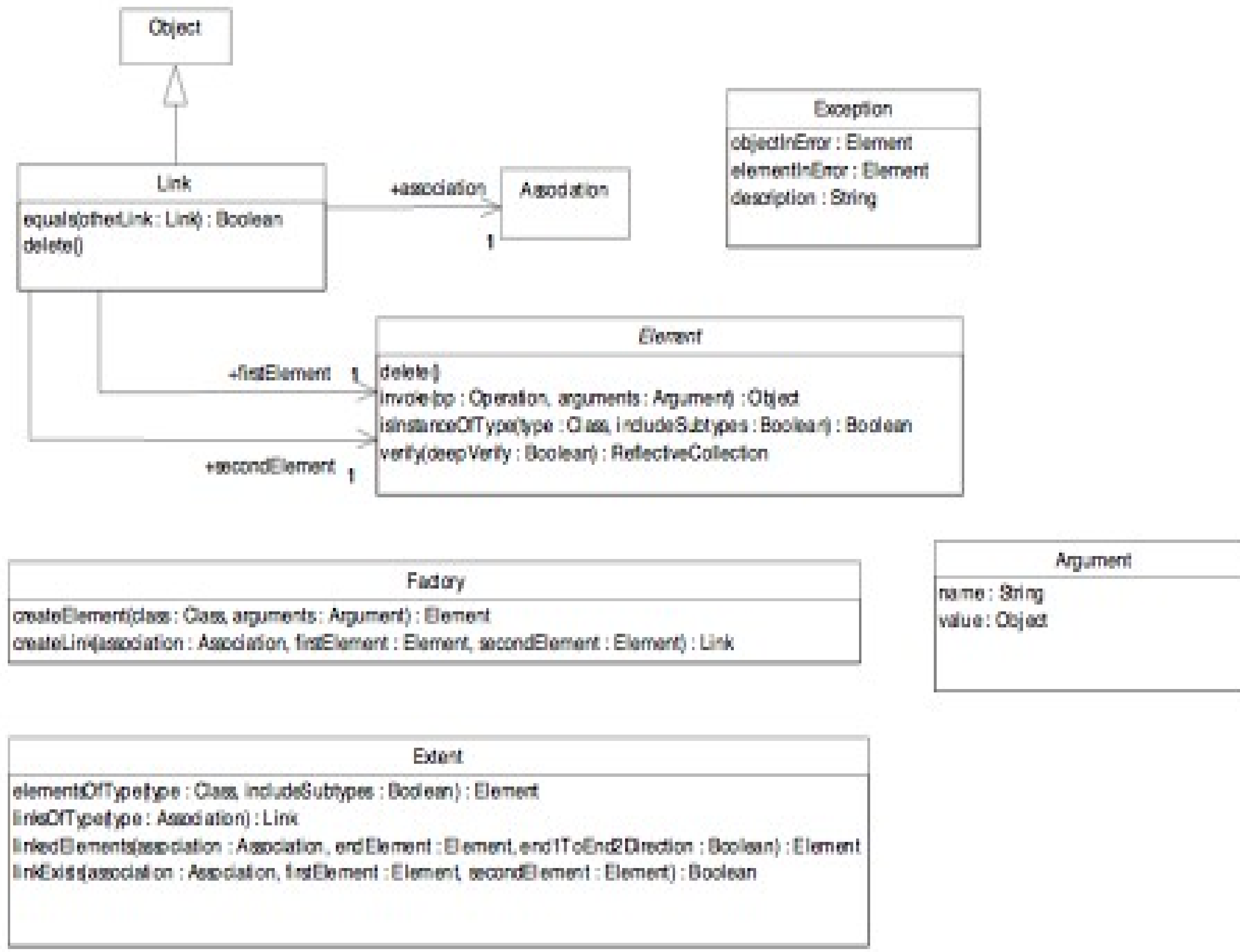
EMOF Reflection

offers access to the metamodel (getMetaClass())
provides a Factory, for creation of a Class from String



[MOF]

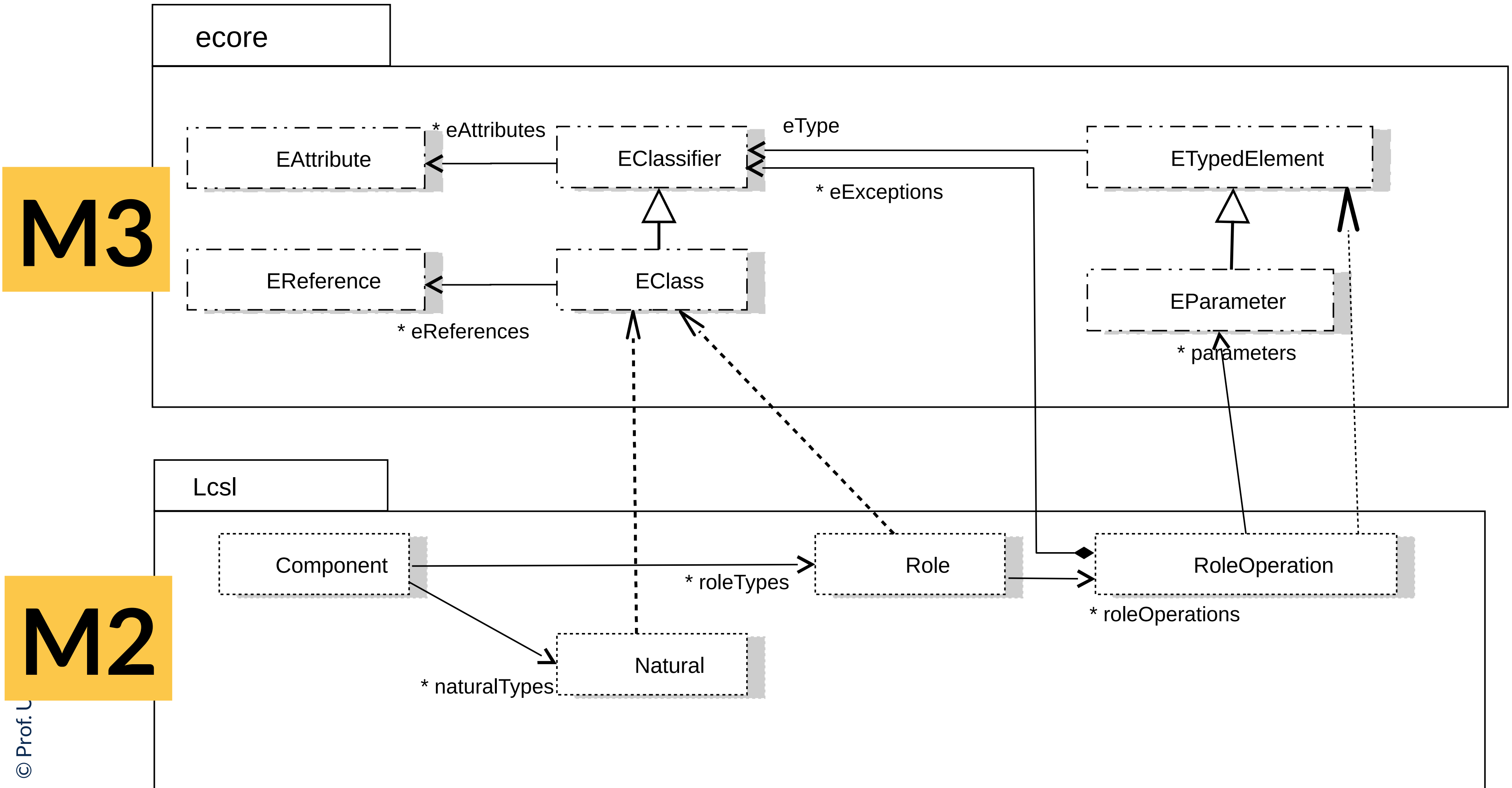
CMOF Reflection



[MOF]

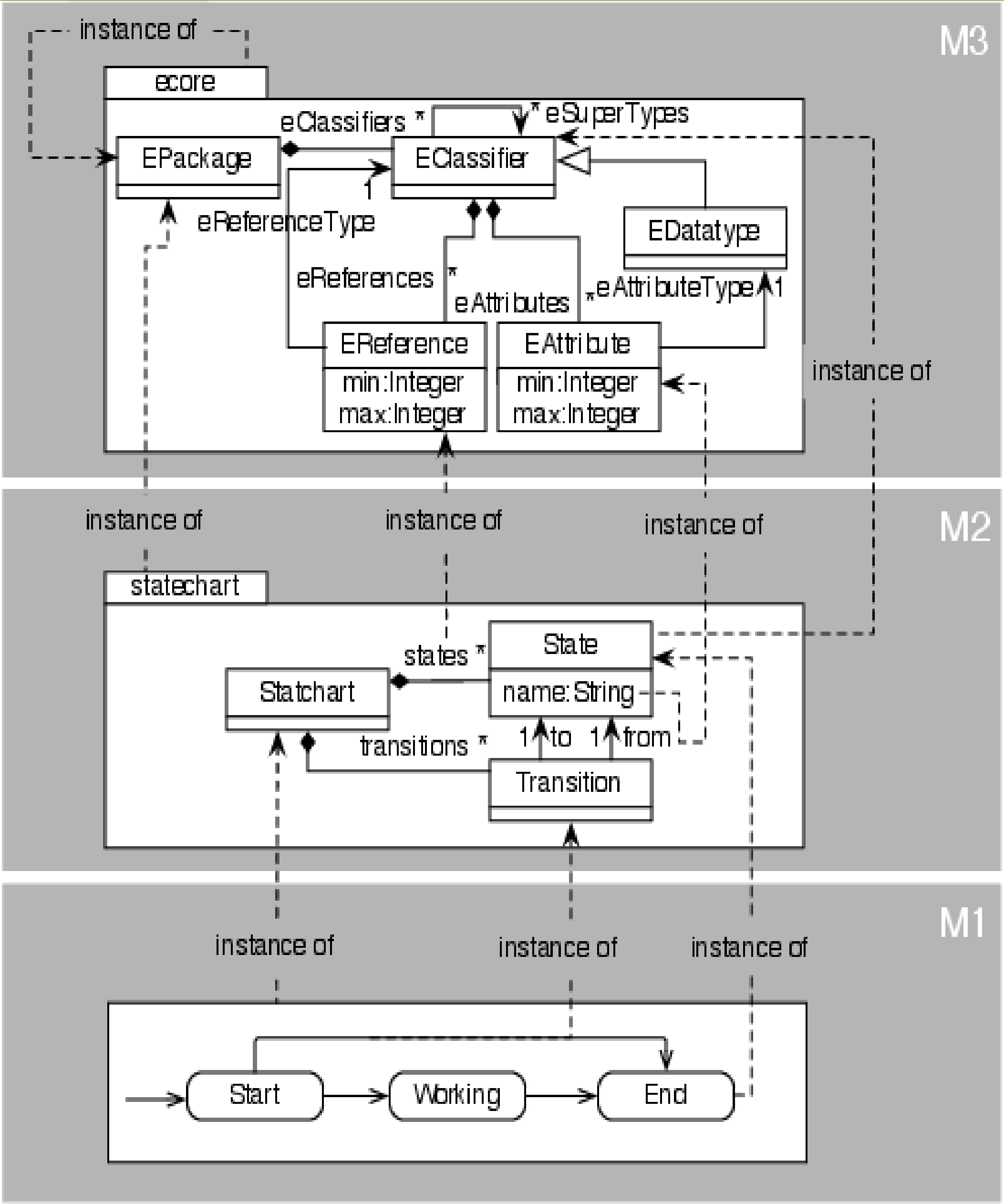
Ex.: Deriving a DSL from EMOF and its Implementation Eclipse ecore

- lcsl is a domain-specific language for component-based modeling [C. Wende]



Ex. EMOF/Ecore based Metamodel of Statecharts

- Ecore is the Eclipse implementation of EMOF, provided by the Eclipse Modeling Framework (EMF) on M3
- Here: a metamodel of statecharts (M2), (which is a little DSL)
- a set of states and their transitions (M1)



4.2.2 Lifting of a Metamodel to a Metametamodel

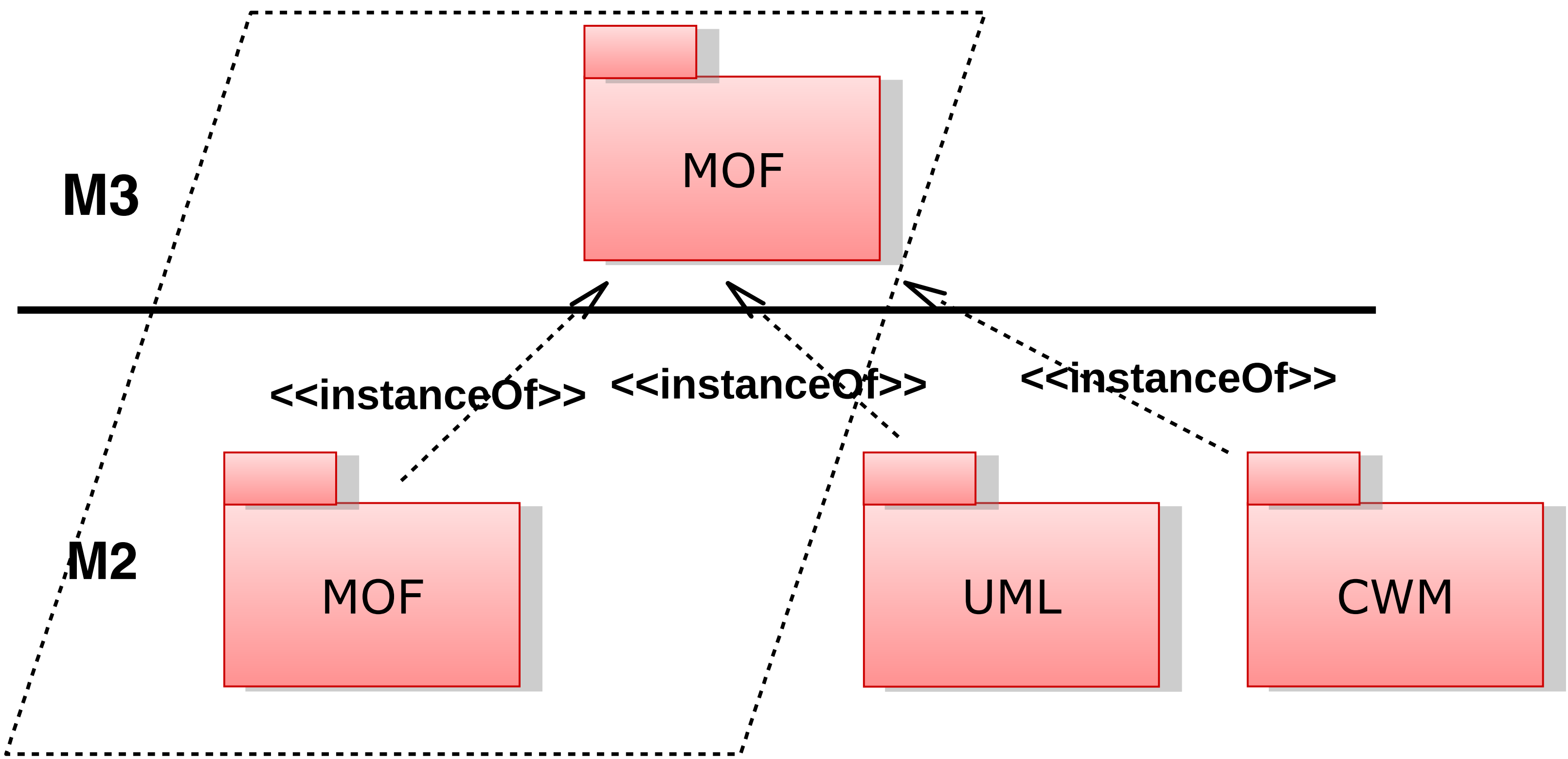
Lifting of Metamodels

A Metamodel of a data definition language in M2 is being ***lifted (promoted)***, if it is used as metametamodel on M3

- Ex. MOF is a simple DDL (Datendefinitionssprache, structural language) for graphs
  - It can be used on M2 to define new languages with package merge (see UML)
  - It can be used on M3 to define metamodels on M2 as instances
  - MOF is self-descriptive

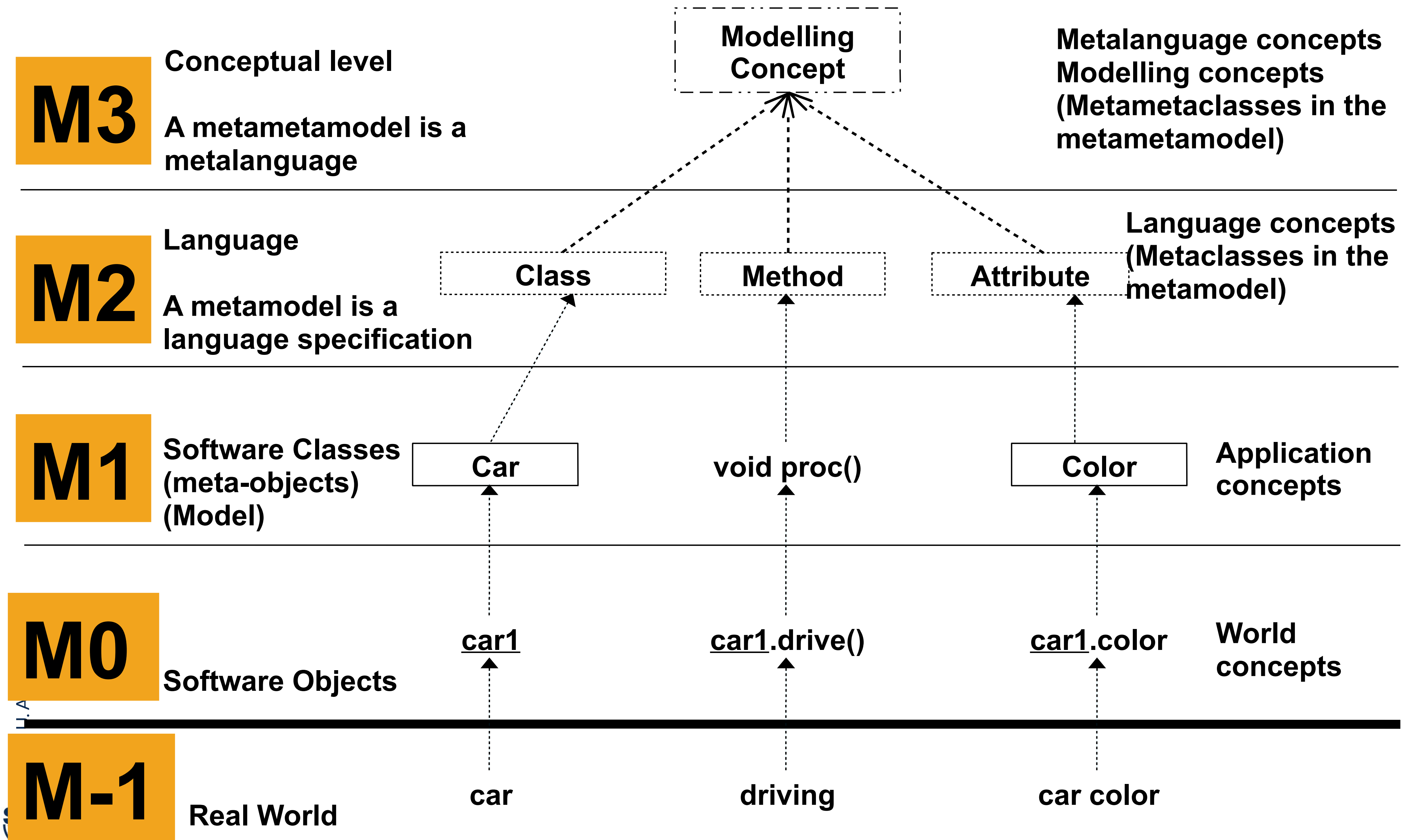
Self-Descriptive MOF

- MOF is ***self-descriptive (selbstbeschreibend)***, because the structure of MOF (M2) is defined in the lifted MOF (M3)
- MOF is *lifted*, because it is used on M2 and M3
- Many other metamodels are also lifted, e.g., ERD

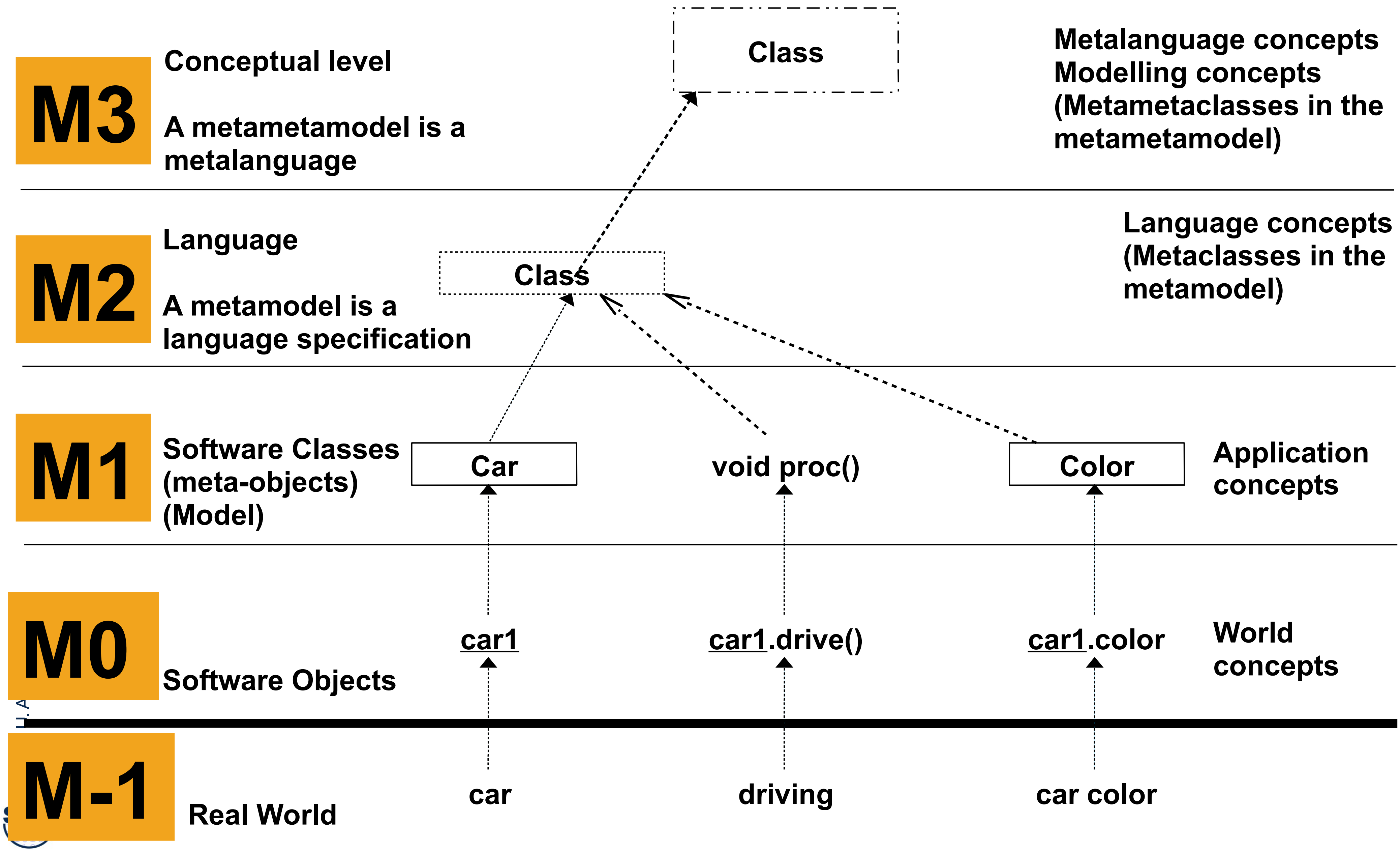


4.2.3 More Examples of Metahierarchies and their Metametamodels

Metalevels in Programming Languages (The Metahierarchy of Programming)

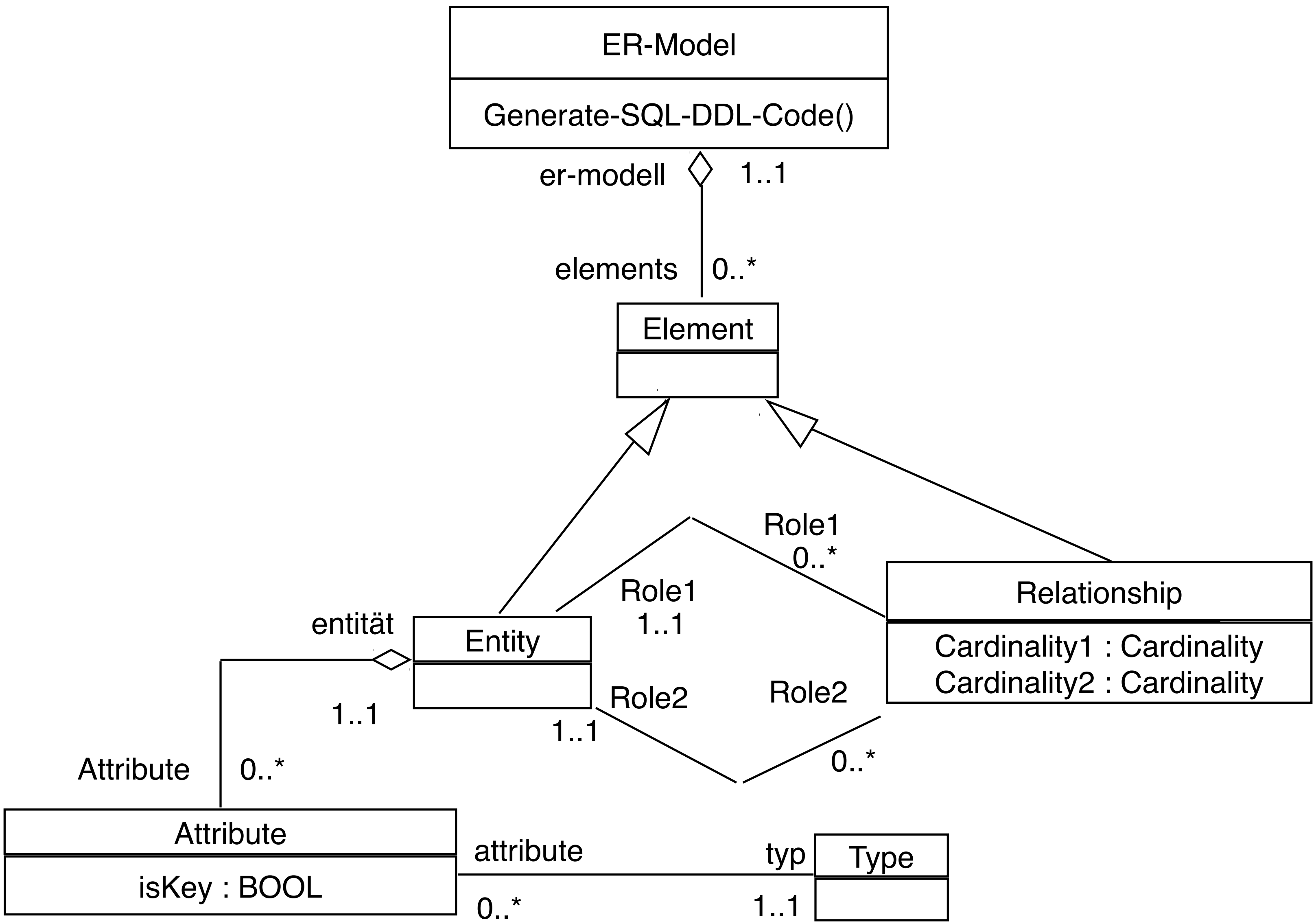


Metalevels in Smalltalk



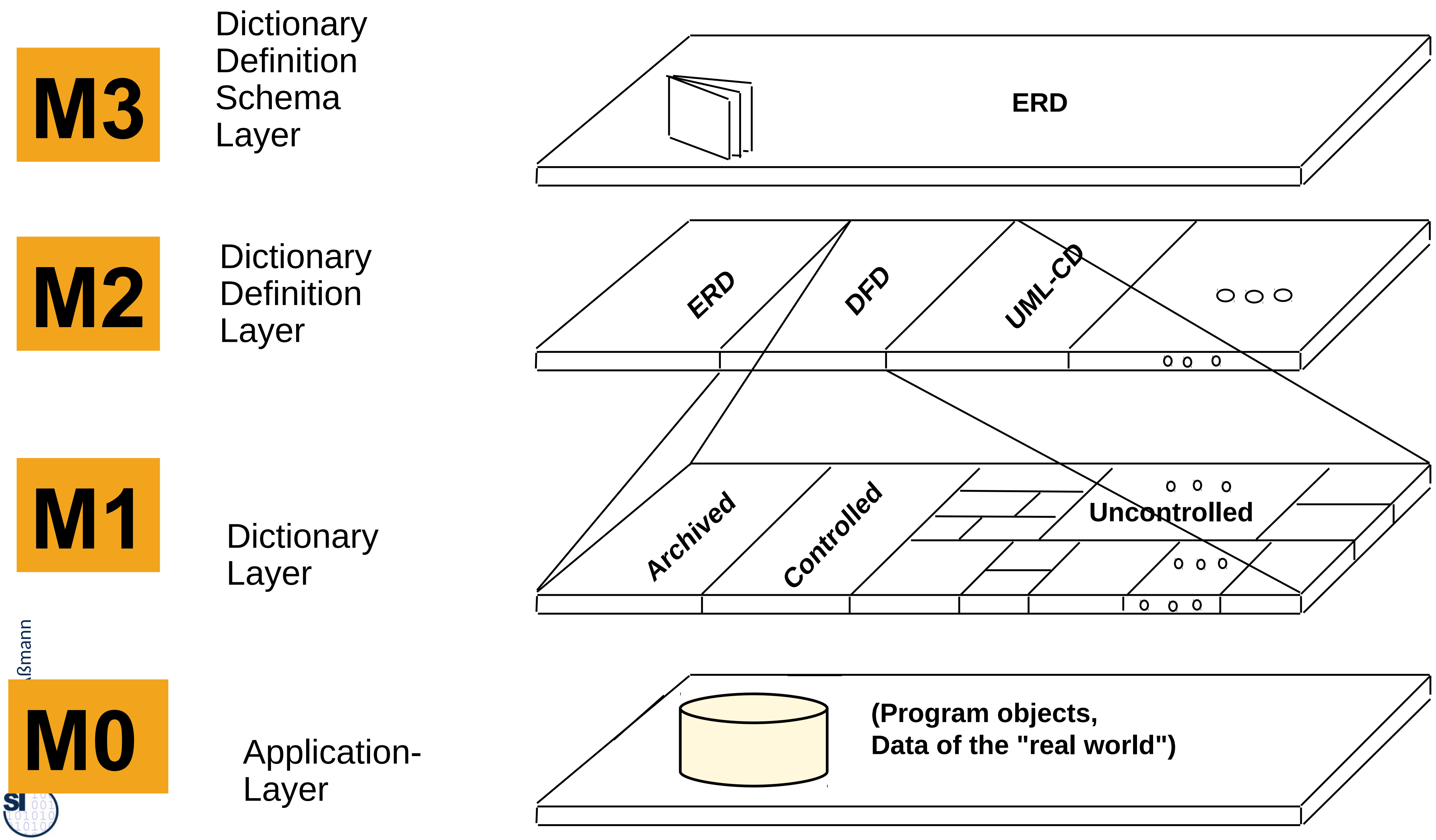
Metamodel of EntityRelationship Diagrams (ERD-ML)

- ERD is like MOF without inheritance



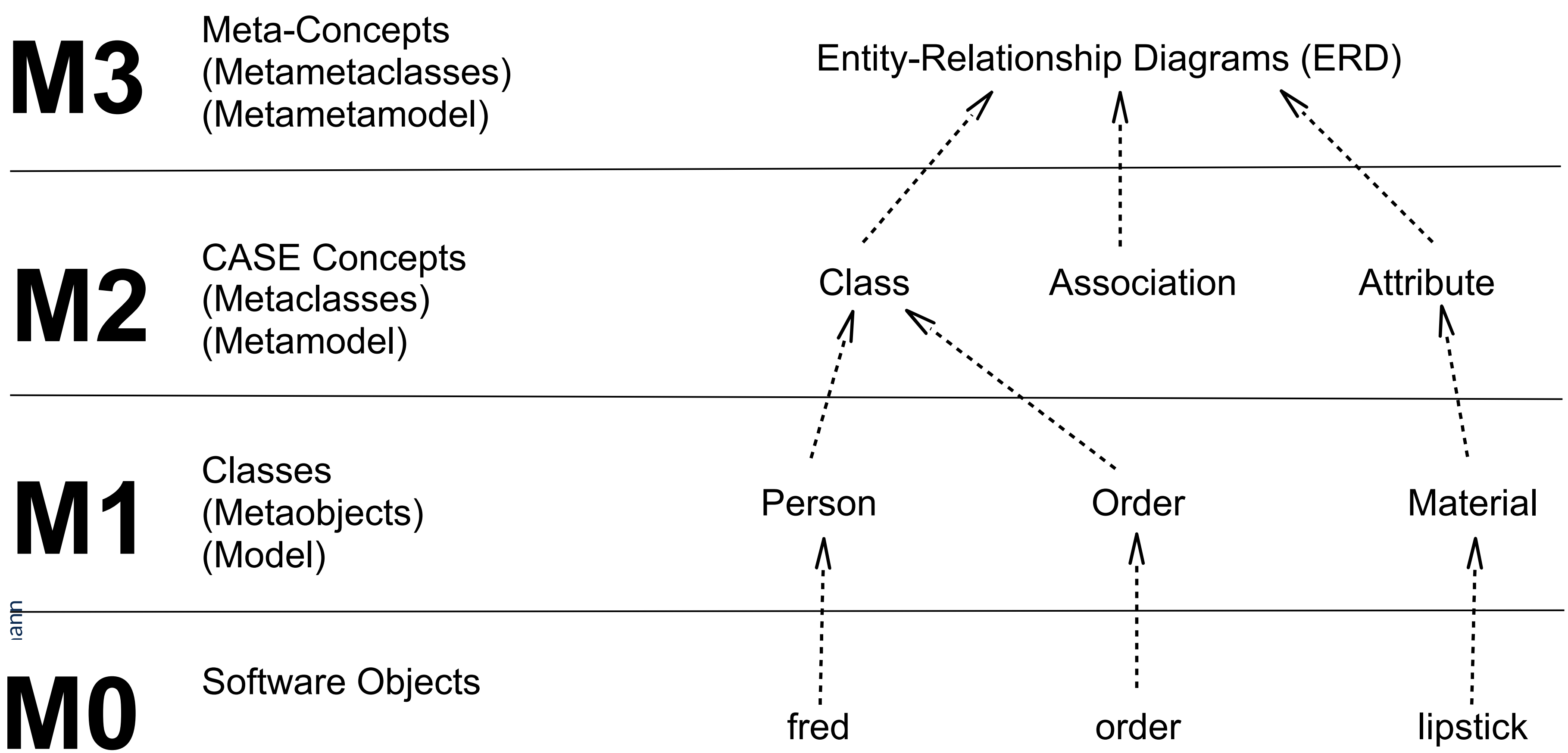
Ex.: IRDS/MOF Metahierarchy for Data Dictionaries in the Structured Analyse (SA)

- IRDS was defined in the 70s to model (persistent) data structures of applications



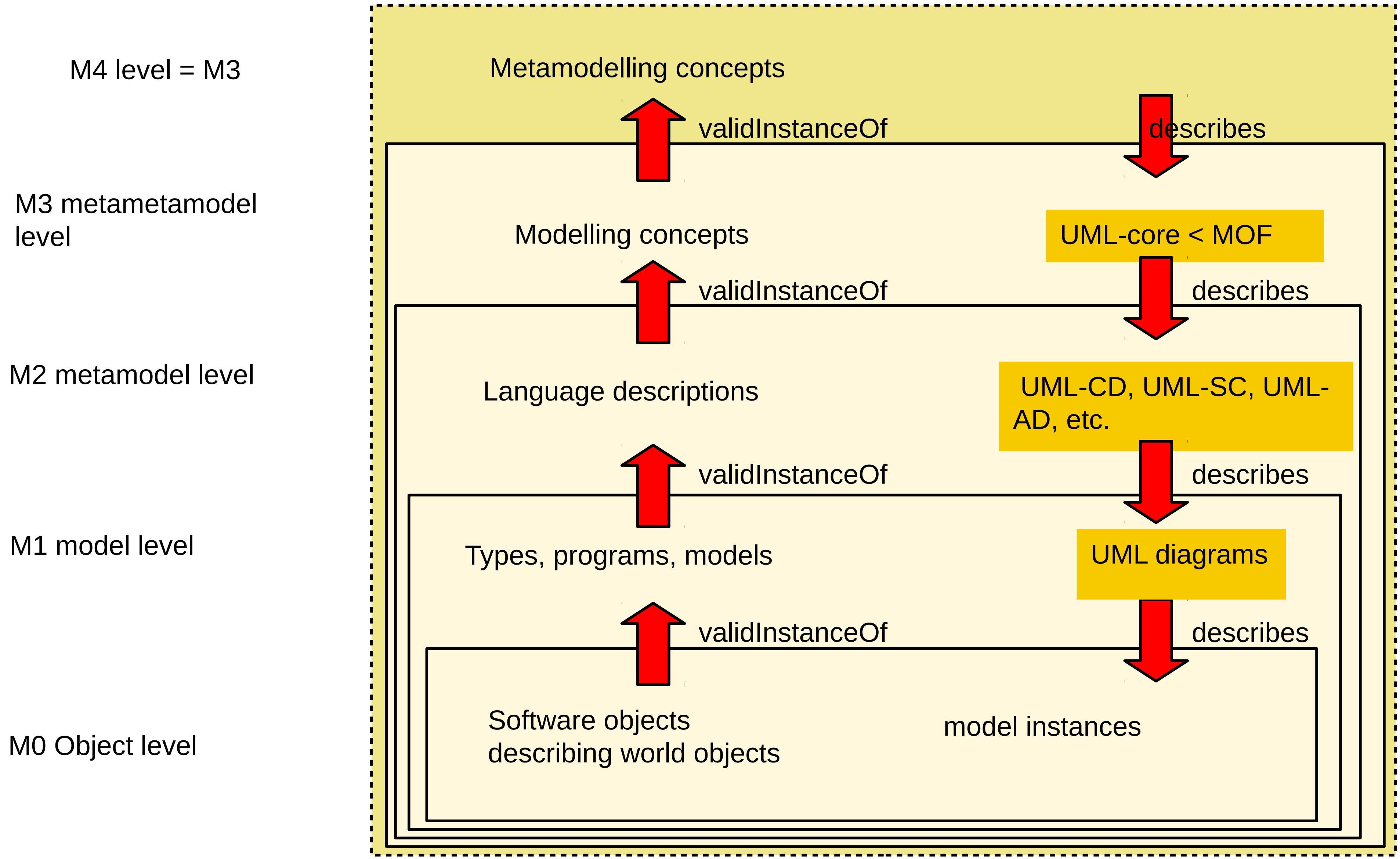
Ex.: Metahierarchy CASE Data Interchange Format (CDIF)

- CDIF was defined in the 80s as industrial interchange format [Flatscher]
- uses entities and relationships on M3 to model CASE concepts on M2

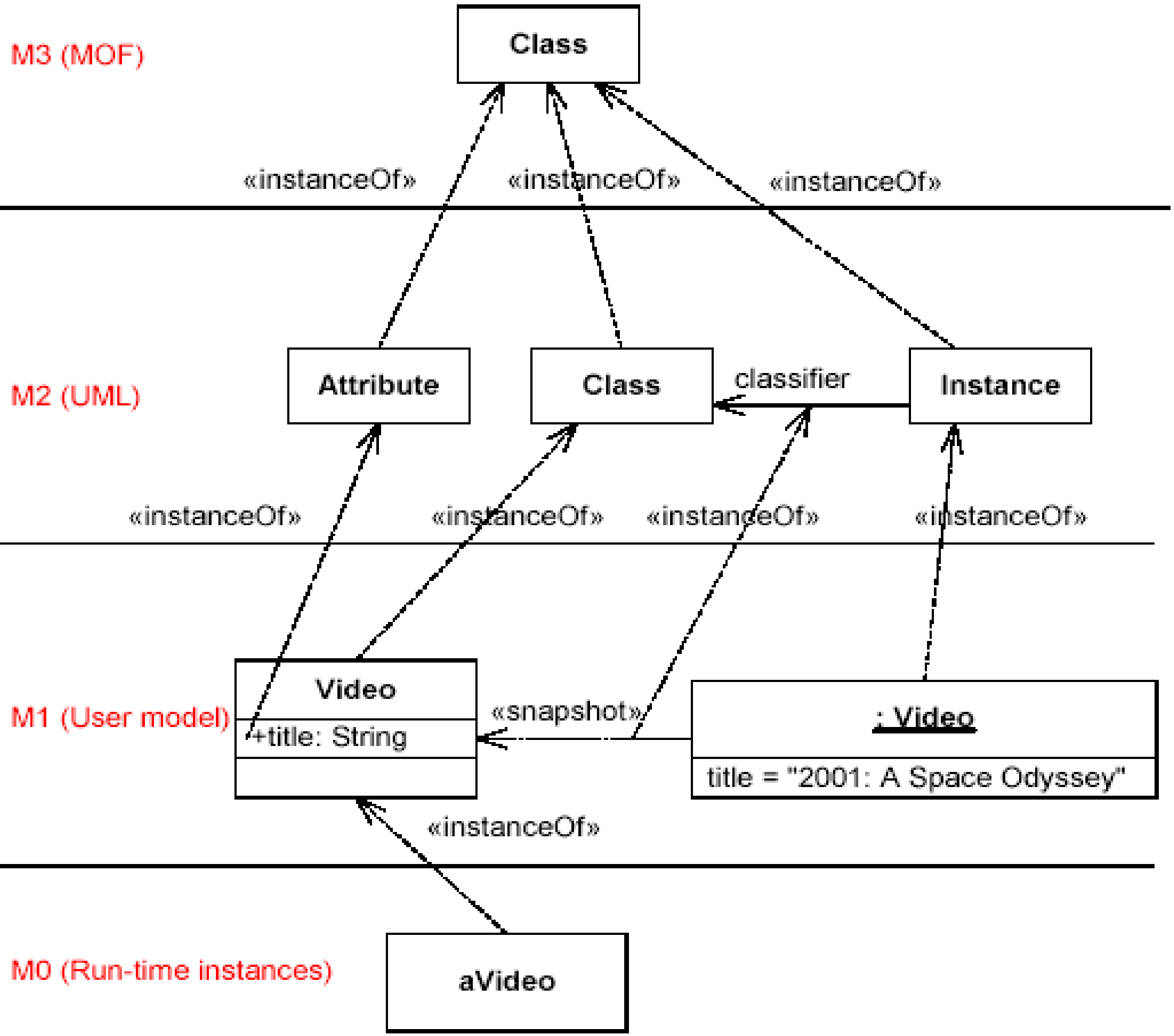


The UML-Core/MOF Metahierarchy

- The UML language manual uses UMLcore, a subset of MOF, as metalanguage



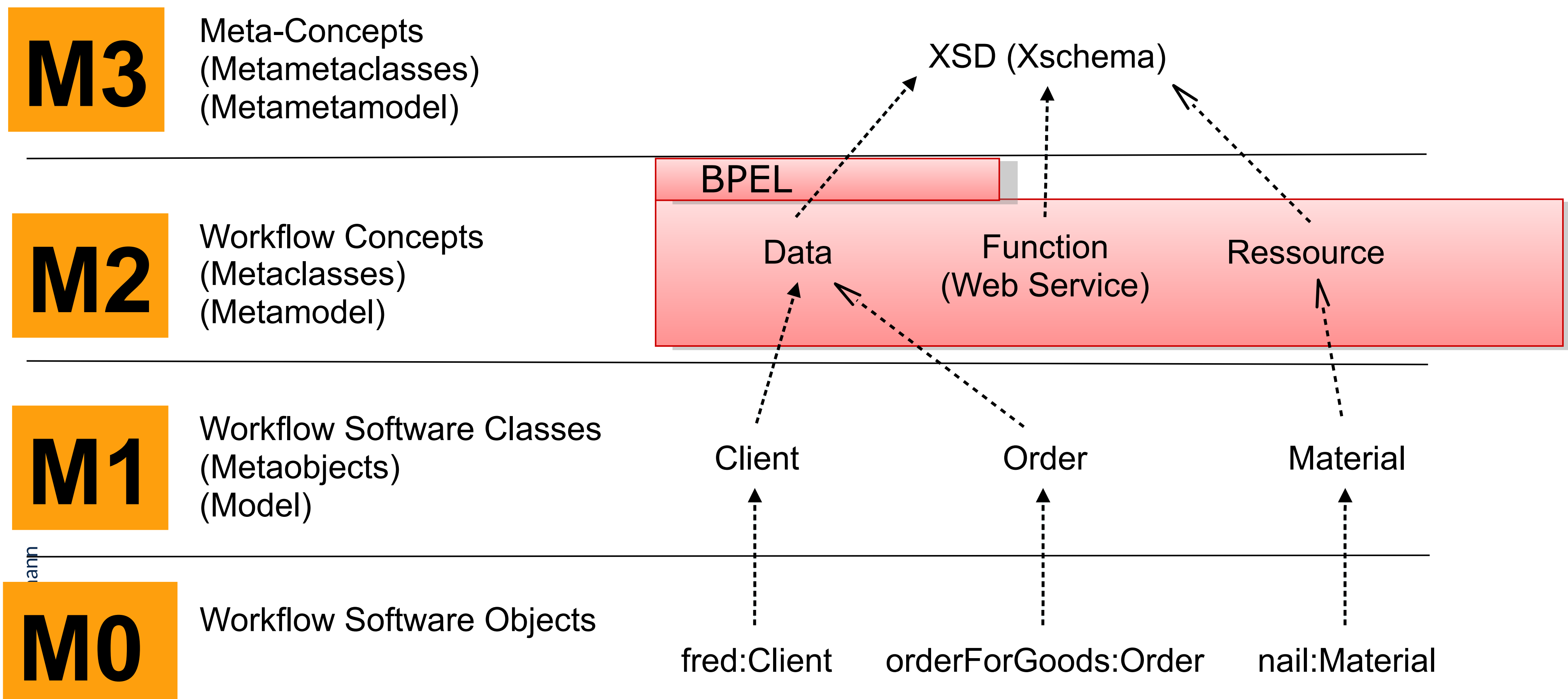
Ex.: MOF-Metahierarchy for UML



From: UML 2.0 Infrastructure Specification; OMG Adopted Specification ptc/03-09-15

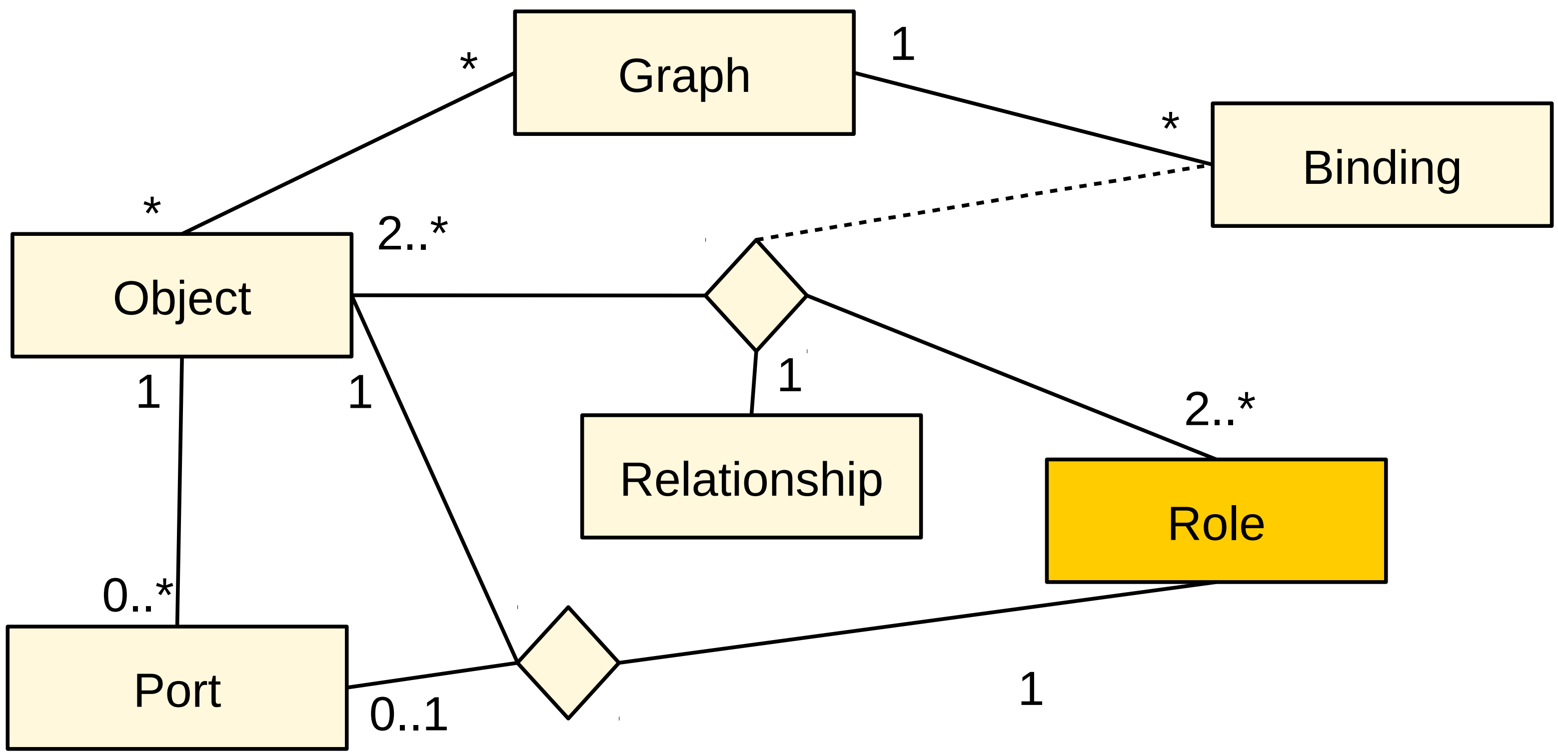
Ex.: Metahierarchy in Workflow Systems and Web Services (e.g., BPEL, BPMN, ARIS-EPK)

- It is possible to specify workflow languages with the metamodelling hierarchy
- BPEL and other workflow languages can be metamodeled
- BPEL is metamodeled with XSD



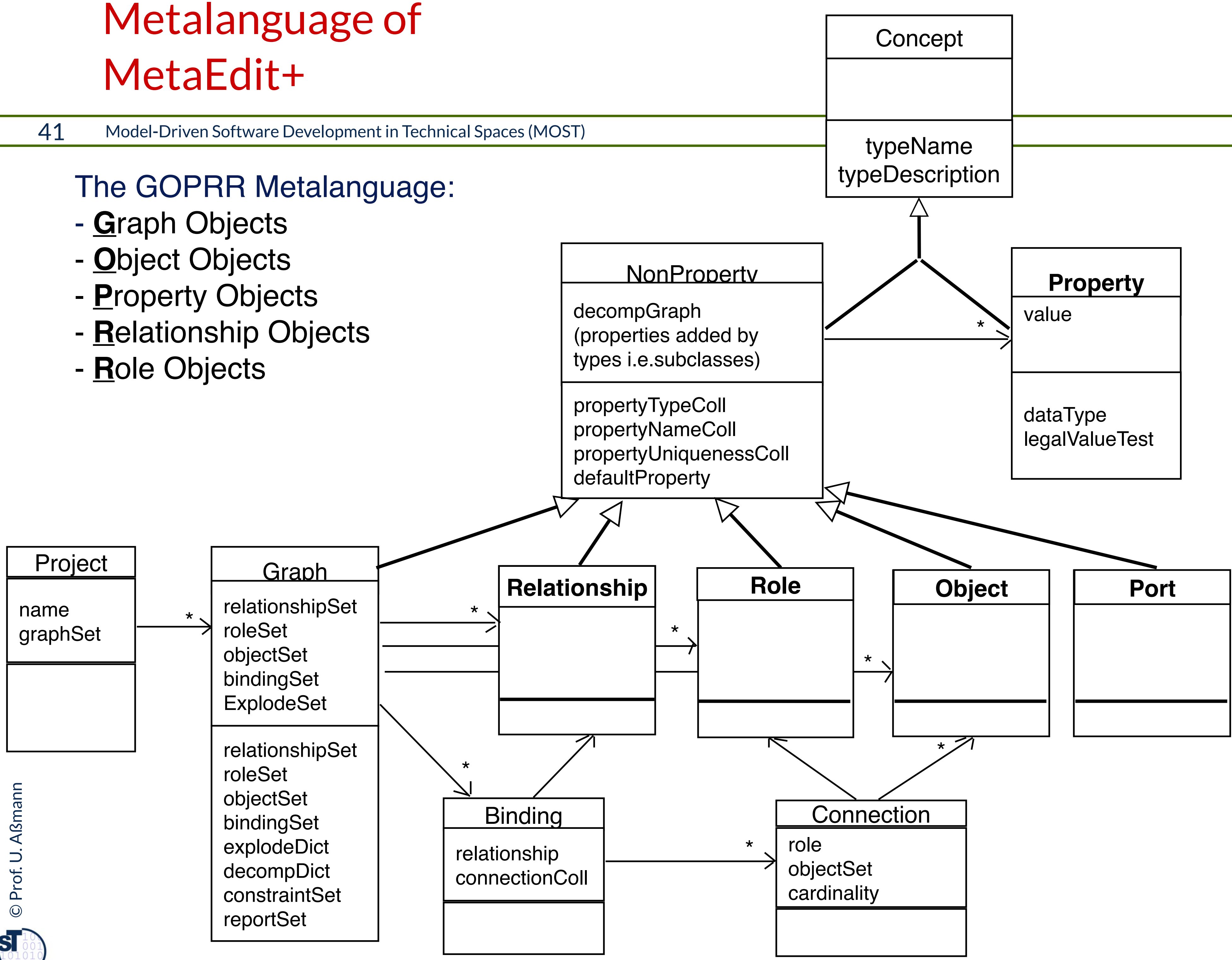
Role-Based Graph Types in MetaEdit+

- [www.metacase.com]
- The tool MetaEdit+ uses the **graph schema (metalanguage) GOPRR**:
  - Objects and their Roles; Relationships
  - Allowed Bindings between all entities:
    - a binding consists of a relationship with roles and playing objects



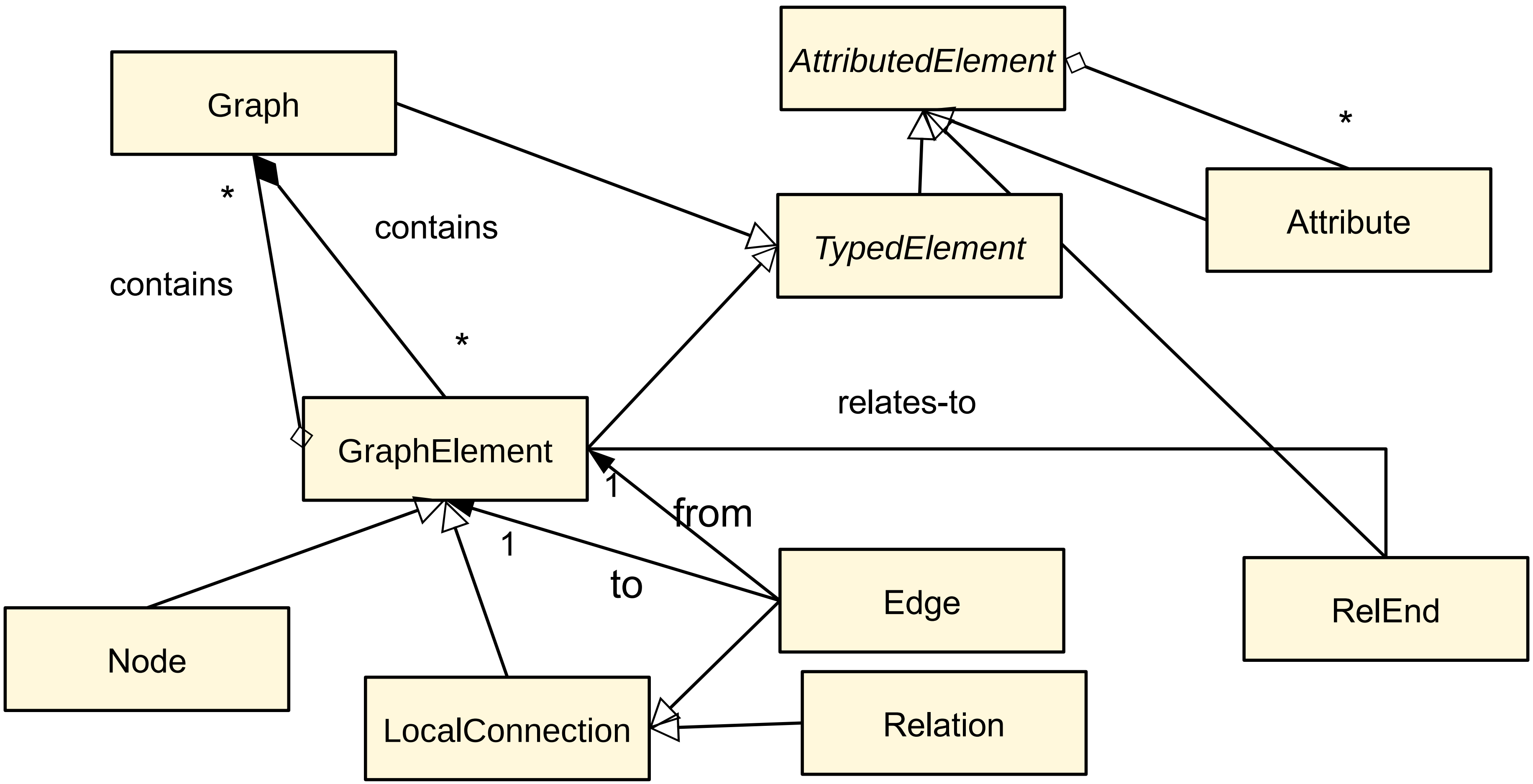
Metalanguage of MetaEdit+

The GOPRR Metalanguage:
- Graph Objects
- Object Objects
- Property Objects
- Relationship Objects
- Role Objects



GXL Graph eXchange Language – a Technical Metametamodel

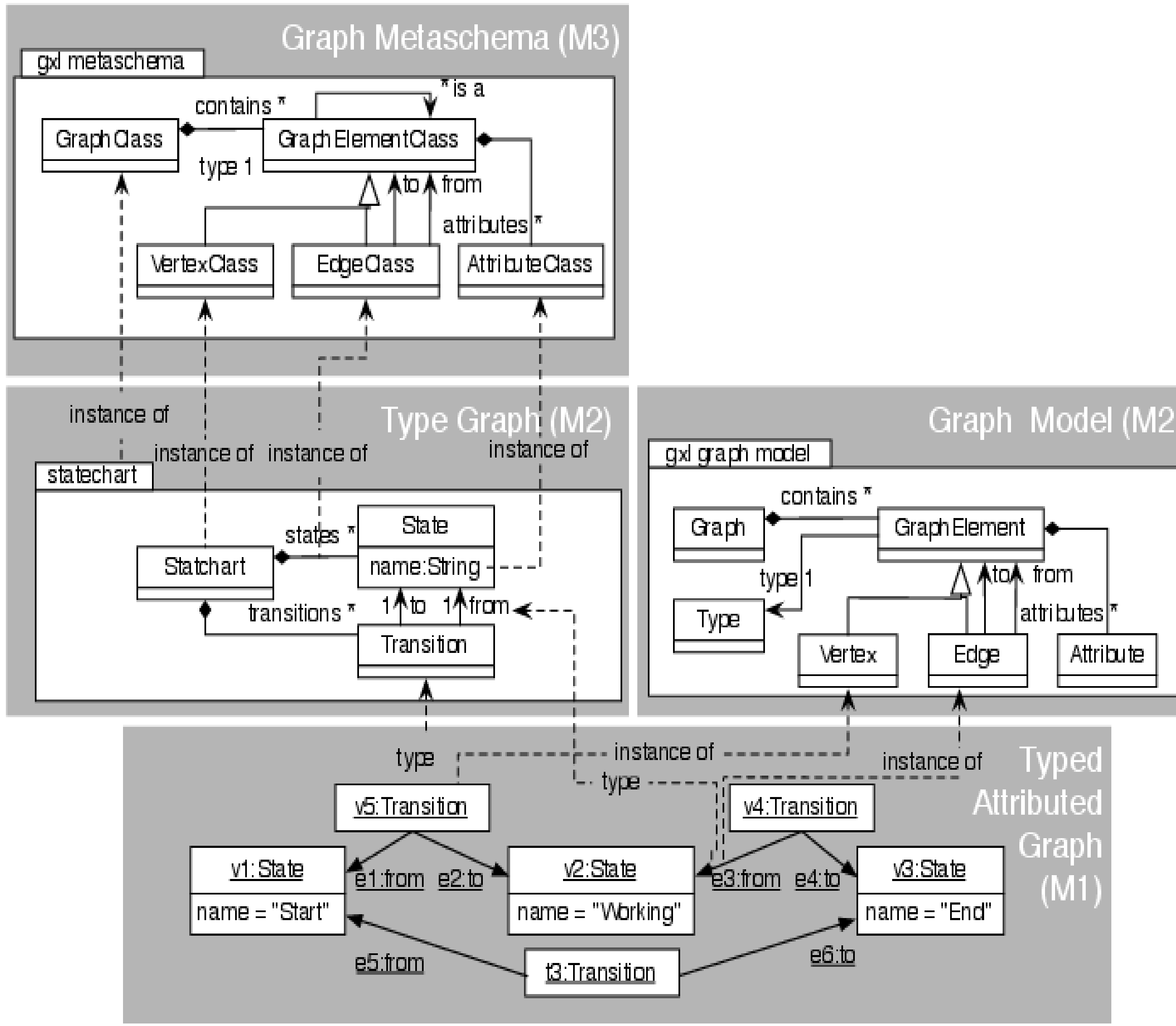
- GXL is a modern graph-language (graph-exchange format)
- Contains abstractions for elements of graphs usable for generic algorithms (e.g., flexible navigation)



Richard C. Holt, Andy Schürr,Susan Elliott Sim, Andreas Winter. GXL: A graph-based standard exchange format for reengineering. Science of Computer ProgrammingVolume 60, Issue 2, April 2006, Pages 149-170

GXL-based Metamodel of Typed Attributed Graph

- GXL can be used as metalanguage (Metametamodel) on M3, to type metamodels and DSL on M2
- For example, state machines
- Alternatively, GXL can also be used as DDL on M2 (it is a lifted metamodel)



Packeting on all Layers

- All layers can be structed into packages

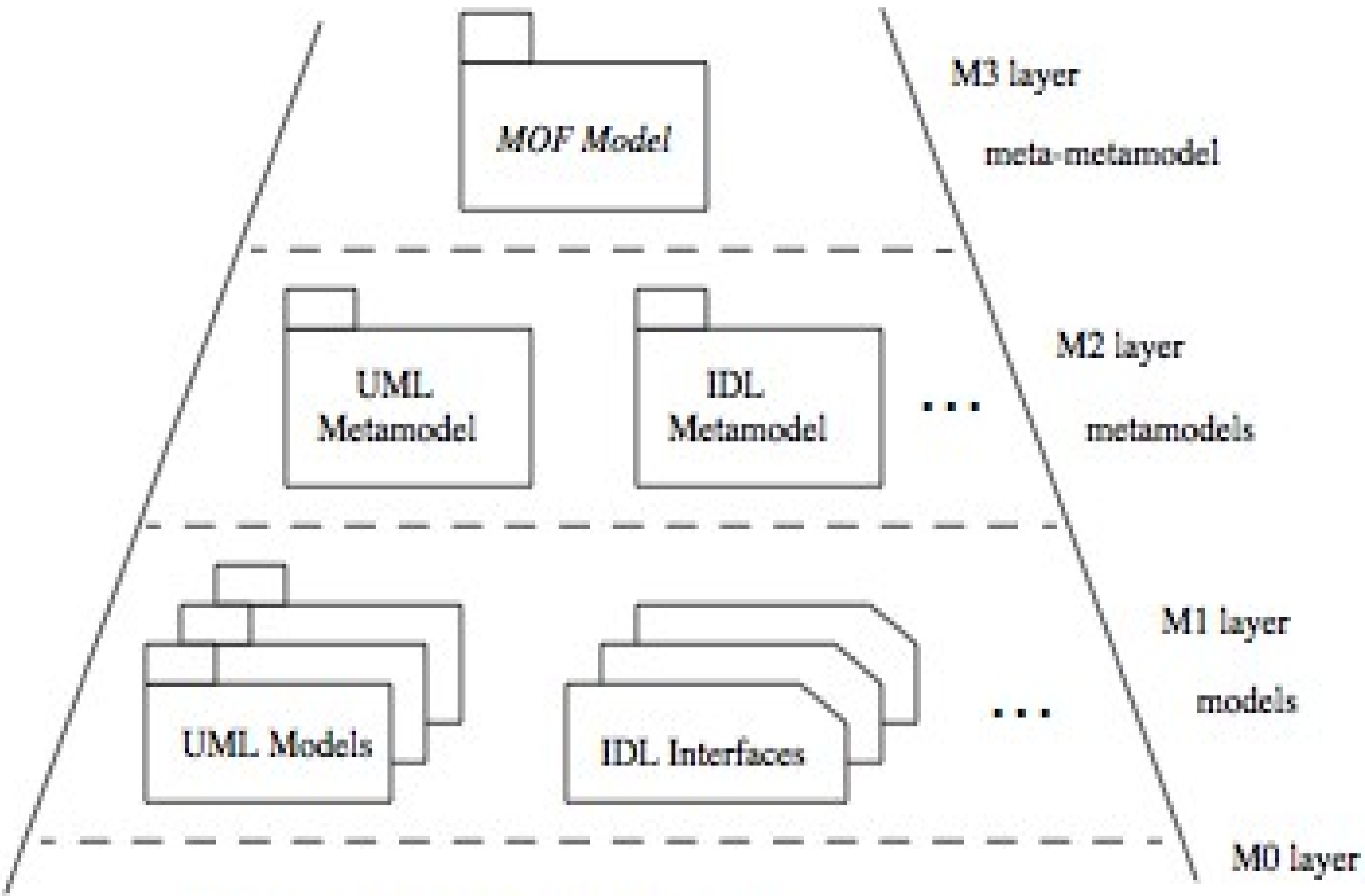
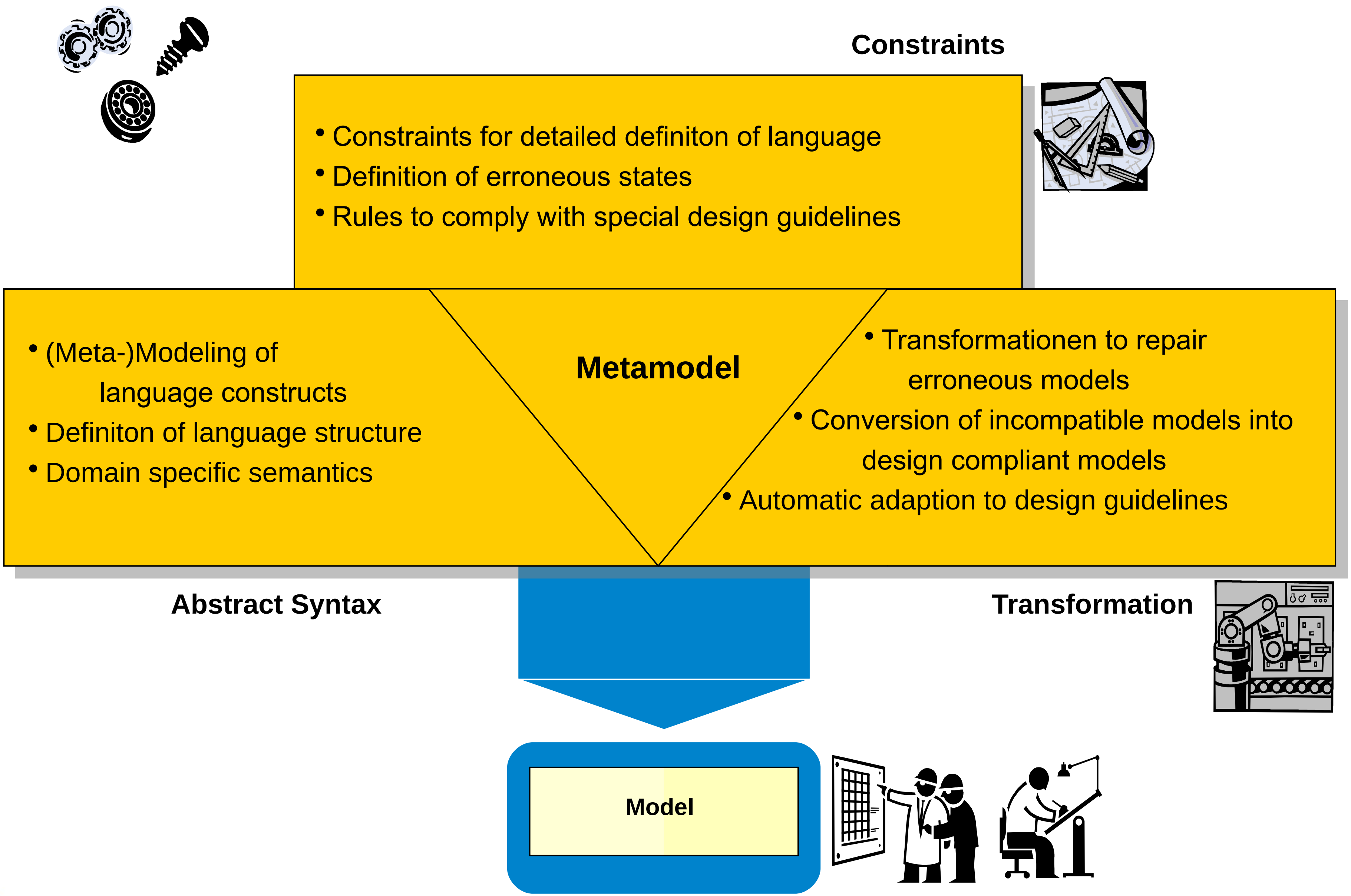


Figure 2-2 MOF Metadata Architecture

[MOF]

Metamodeling – Benefits



[F. Klar, TU Darmstadt]

Excursion: Metaprogramming

- **Metaprograms (reflective programs)** generate code on the basis of a metamodel of their own language (self model)
- **Metaprogram-Procedures** (Semantic Macros, Hygenic Macros, Programmable Macros [Weise/Crew], Orchestration Style Sheets) can be typed by a metamodel
    - Parameter types and return types of prodedures are metaclasses
- → See course CBSE

4.3 Technological & Technical Spaces

Technological Spaces

A ***technological space*** is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities.

- It is often associated to a given user community with shared know-how, educational support, common literature and even workshop and conference regular meetings.
  - Ex. compiler community, database community, semantic web community, automotive community
  - [Technological Spaces: an Initial Appraisal. Ivan Kurtev, Jean Bézivin, Mehmet Aksit. CoopIS, DOA'2002 Federated Conferences, Industrial Track. (2002) http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.332&rep=rep1&type=pdf]

Technical Spaces

A ***technical space*** is a metamodeling framework (in a technological space) with a metapyramid (metahierarchy), accompanied by a set of tools that operate on the models definable within the framework.

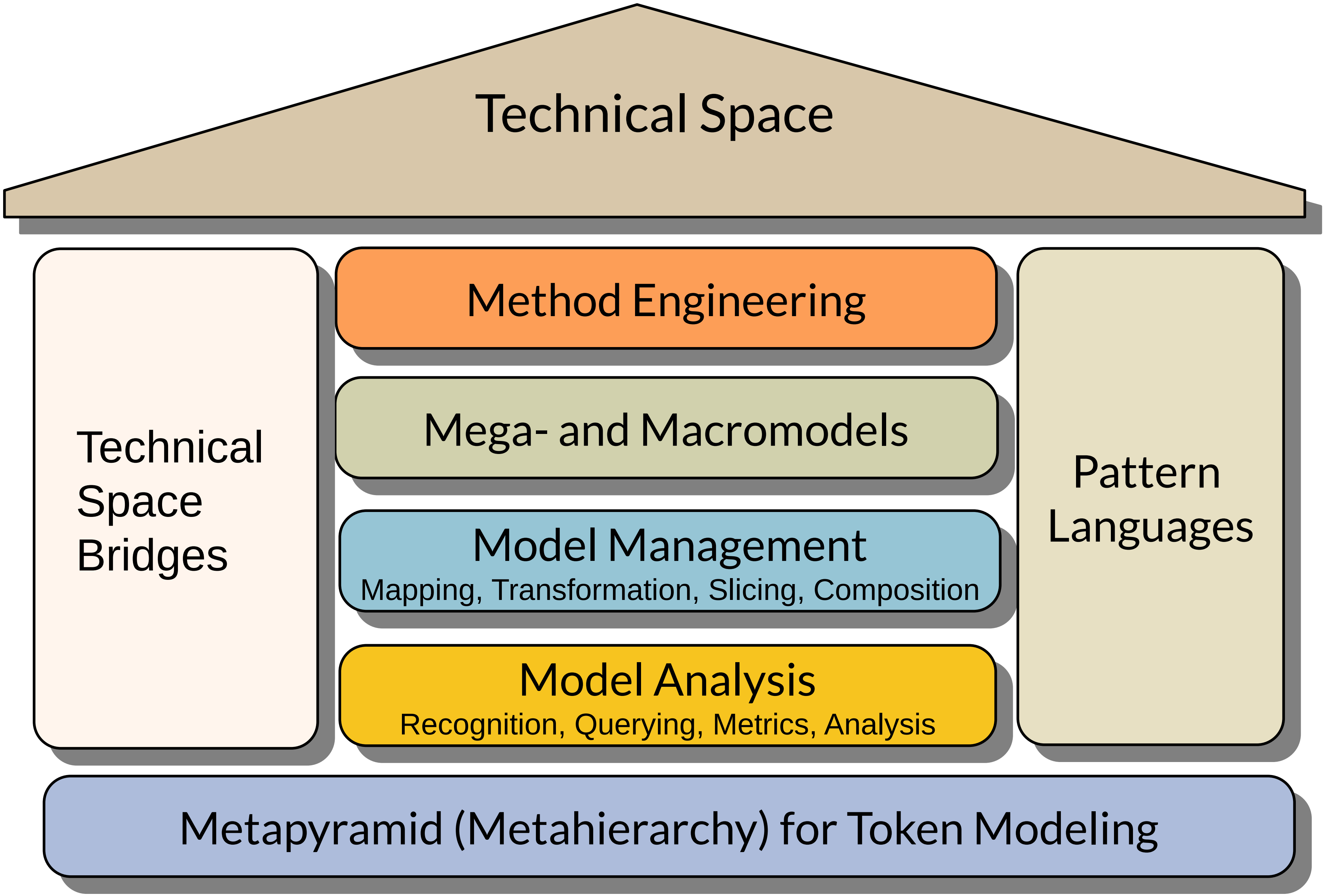
- [Model-based Technology Integration with the Technical Space Concept. Jean Bezivin and Ivan Kurtev. Metainformatics Symposium, 2005.]
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.1366&rep=rep1&type=pdf
- Ingredients of a Technical Space (Technikraum):
  - A **metapyramid** (or **metahierarchy**) with data (tools, workflows, and materials on M0), Code and models (M1), languages (M2), and metalanguages (M3)
  - A **model management unit** (**model algebra** or **model composition system**)
  - A **macromodel**
- Be aware: A technological space may contain several technical spaces:
  - Compiler community: Grammarware, Tree-Ware, Graph-Ware
  - Database community: Relational database model, csv-tables, XML
  - Business software: Reports in TextWare, TableWare

The Trick of the Metapyramid

Observation:
In the metapyramid of a technical space, tools can be applied on every level.

- Tools on level M[n-1] can work on M[n]
- Tools can be *lifted* from the object to the class to the metaclass level to the metametaclass level:
- Object-manipulating tools on M0 work for clabjects in models on M1
  - Graph-manipulating tools on M0 for models on M1
- Class-manipulating tools on M1 work for clabjects in metamodels on M2
  - Model-manipulating tools on M1 work for metamodels on M2
- Metaclass-manipulating tools on M2 work for clabjects in metamodels on M3
  - Metamodel-manipulating tools on M2 work for metametamodels on M3

Q10: The House of a Technical Space



Q10: Overview of Technical Spaces in the Classical Metahierarchy

	Grammarware (Strings)	Text-ware	Table-ware		Treeware (trees)			Graphware/Modelware			Role-Ware	Ontology-ware
	Strings	Text	Text-Table	Relational Algebra	NF2	XML	Link trees	MOF	Eclipse	CDIF	MetaEdit+	OWL-Ware
M3	EBNF	EBNF		CWM (common warehouse model)	NF2-language	XSD	JastAdd, Silver	MOF	Ecore, EMOF	ERD	GOPPR	RDFS OWL
M2	Grammar of a language	Grammar with line delimiters	csv-header	Relational Schema	NF2-Schema	XML Schema, e.g. xhtml	Specific RAG	UML-CD, -SC, OCL	UML, many others	CDIF - languages	UML, many others	HTML XML MOF UML DSL
M1	String, Program	Text in lines	csv Table	Relations	NF2-tree relation	XML-Documents	Link-Syntax-Trees	Classes, Programs	Classes, Programs	CDIF - Models	Classes, Programs	Facts (T-Box)
M0	Objects	Sequences of lines	Sequences of rows	Sets of tuples	trees	dynamic semantics in browser		Object nets	Hierarchical graphs	Object nets	Object nets	A-Box (RDF-Graphs)

4.4. Model Analysis in a Technical Space with Model Querying, Model Metrics, and Model Analysis

Discussing the internals of models and their model elements

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

The Internals of a Model

Model analysis techniques reveal the inner details of models.

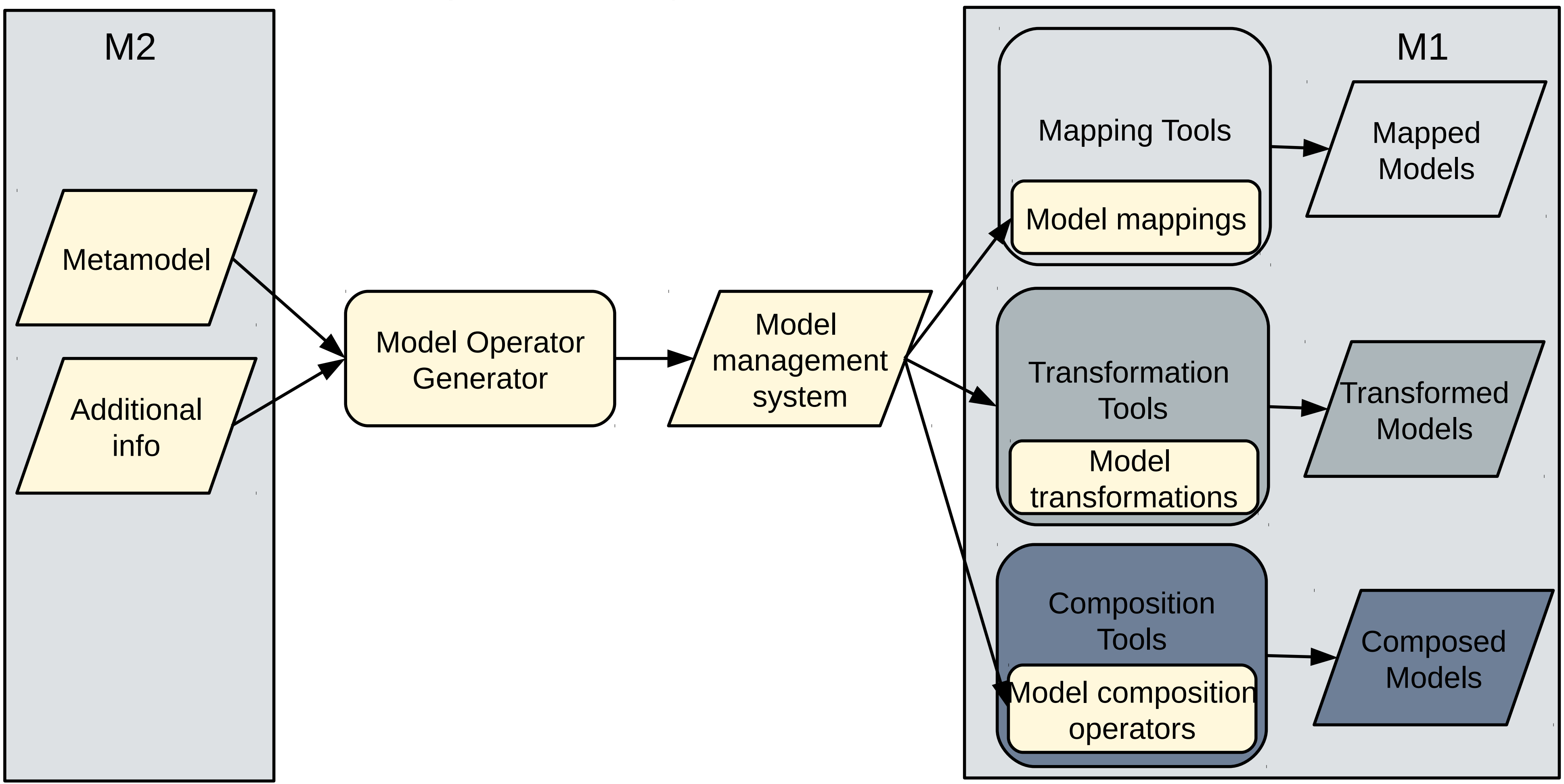
- **Model querying** searches patterns in models, described by a query or pattern match expression.
  - Searching for a method with a specific set of parameters
- **Model metrics** counts patterns in models
  - Counting the depth of the inheritance hierarchy
- **Model analysis** analyzes hidden knowledge from the models, making implicit knowledge explicit
  - Value flow analysis between variables in programs

4.5. Model Management in a Technical Space with Model Mapping, Transformation and Composition

Discussing the relationships of models and their model elements

Model Management in a Technical Space

- A ***model management system*** manages the relationships of models, metamodels, metametamodels of a technical space as well as the relationships of their elements
  - Model mapping subsystem
  - Model transformation subsystem
  - Model composition subsystem

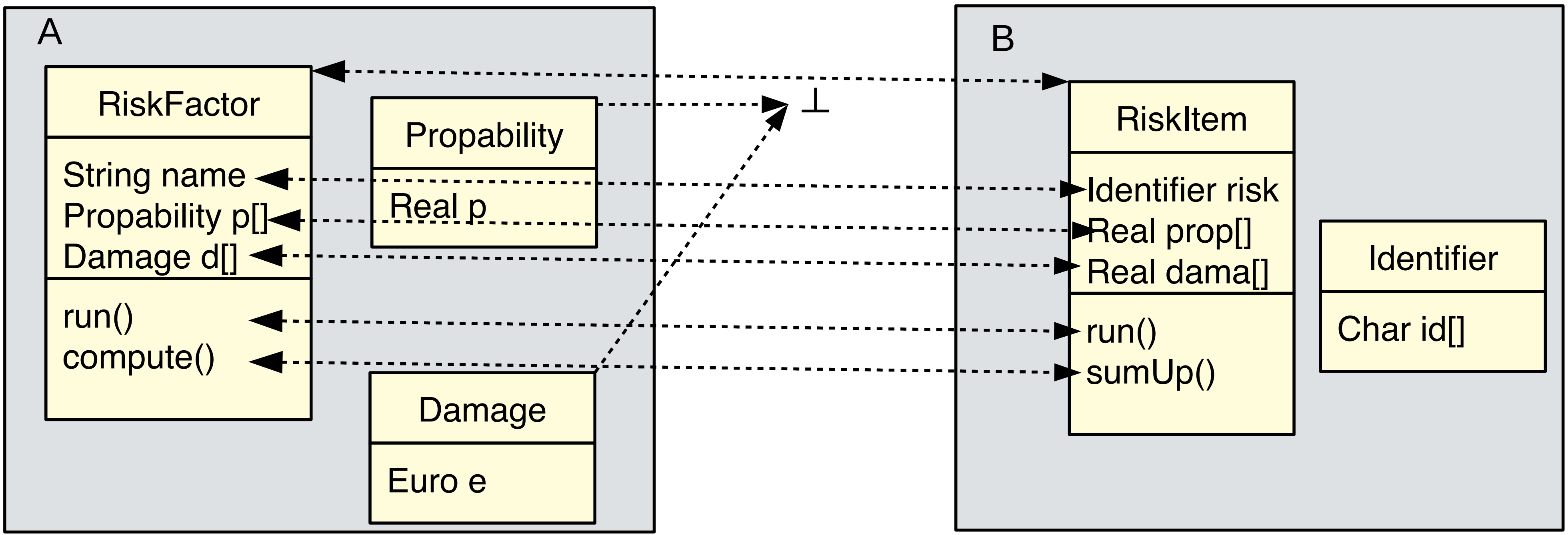


4.5.1. Model Mapping

Model Mappings

A ***model mapping*** is a mapping between the model elements of several models.

- A **trace mapping** records during a model elaboration, model restructuring or model transformation, *which model elements are copied from model A to model B, or created in B.*
- A **synchronization mapping** records *hot-links* model elements from model A to model B.

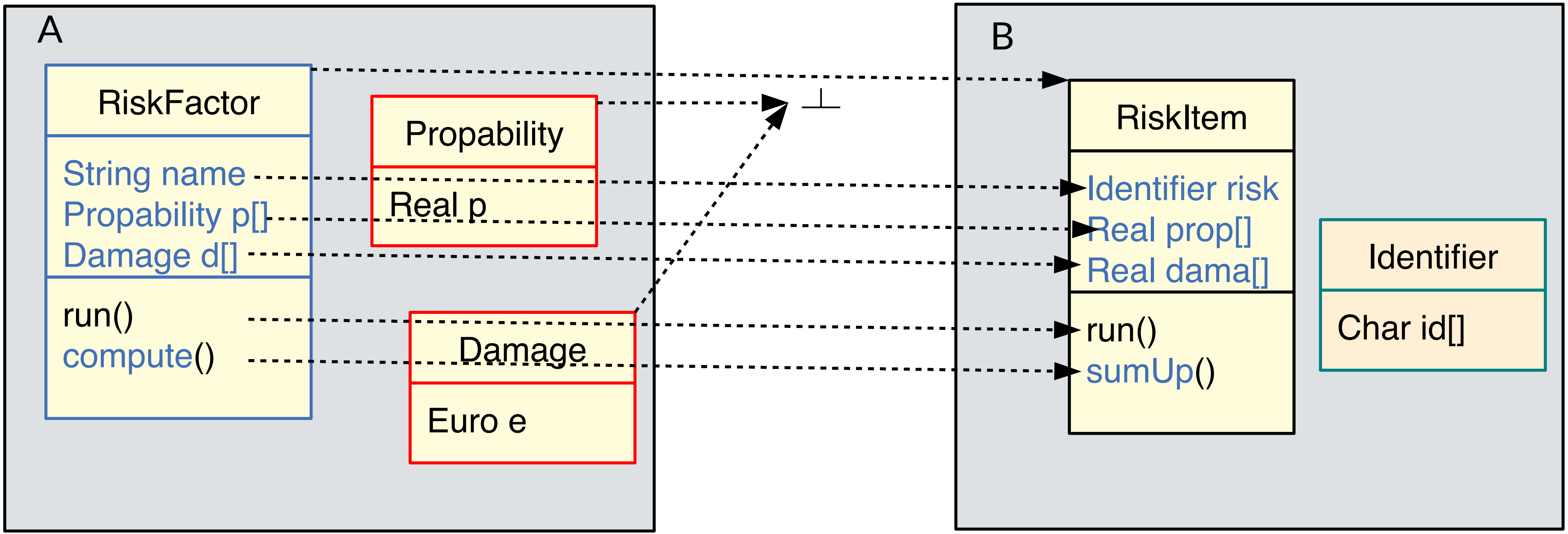


4.5.2. Model Transformation

Model Transformations

A ***model transformation*** is a program (or a specification how) to derive a model A from a model B.

- From a model mapping, two (partial) model transformations (forward and backward) may be derived.
- Deleted model elements are framed red, added elements are framed green, modified blue

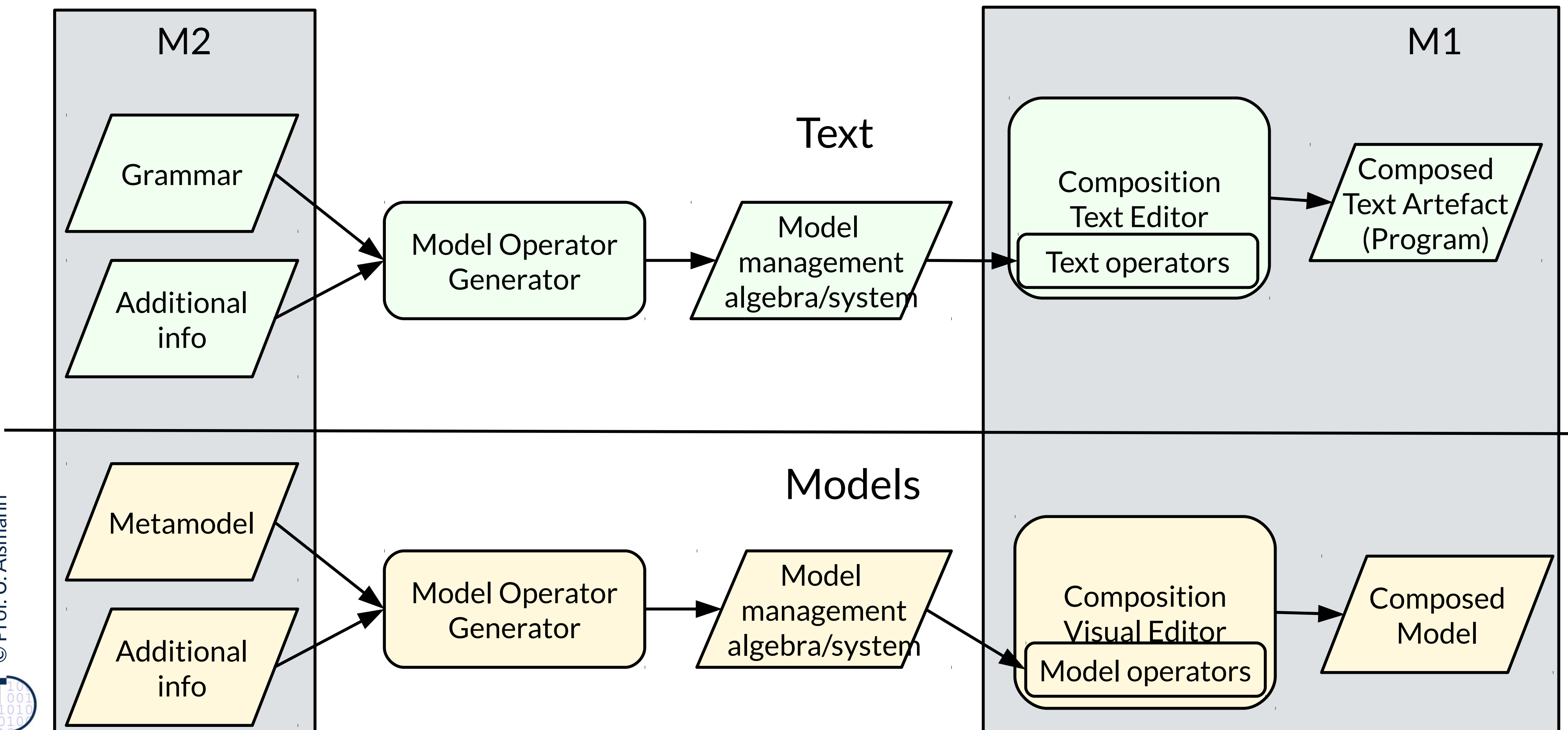


4.5.3. Model Composition with Model Algebrae and Composition Systems

Component-based Model Engineering (CBME)

Model Composition in a Technical Space

- A ***model composition system*** manages the relationships of models, metamodels, metametamodels of a technical space with a uniform model algebra
  - Operators on M1 can be generated from M2
  - Operators on M2 can be generated from M3

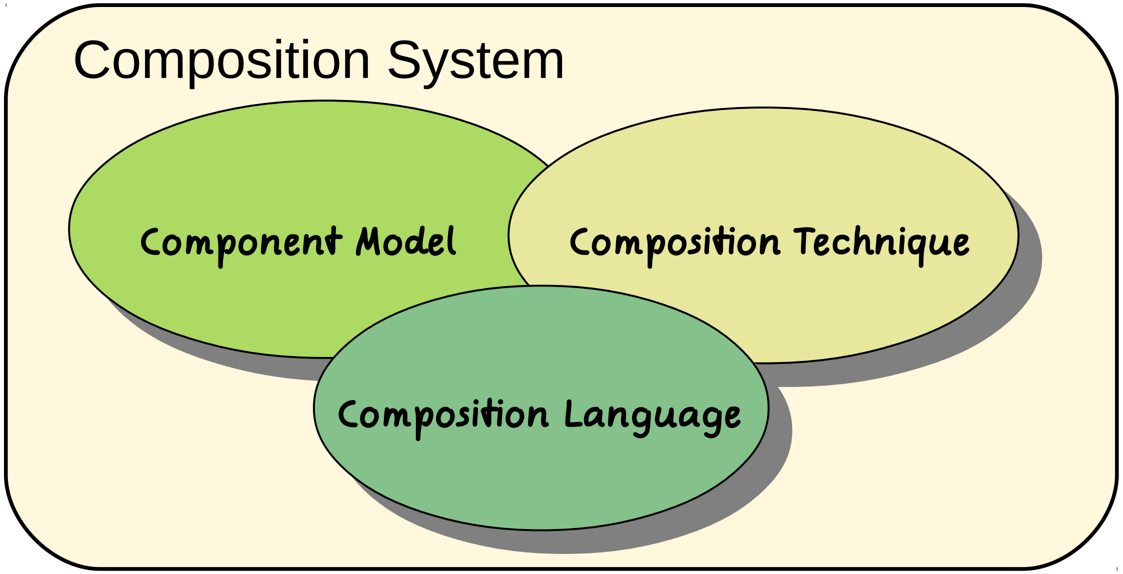


Simple Algebra for Models (on M1) and Metamodels (on M2)

- Models and metamodels can be grouped in packages (module)
  - A simple component model and composition system (see CBSE)
- Algebraic composition technique with operators on packages:
  - use (import)
  - merge (union)
  - Instance-of (element-of-reified-set)

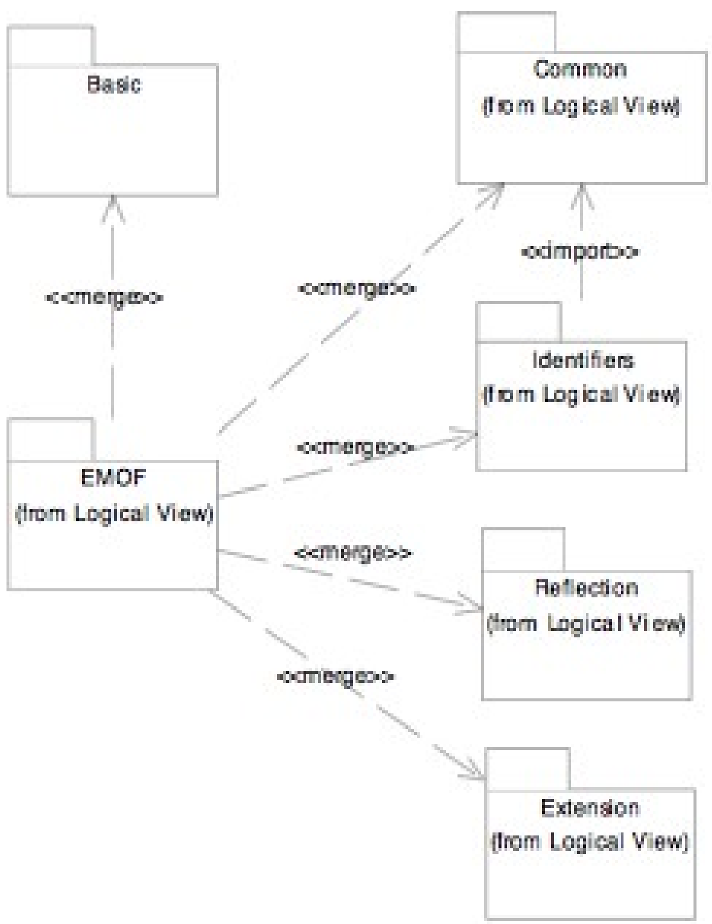
→ Metamodels are composed by unifying their views in the different packages

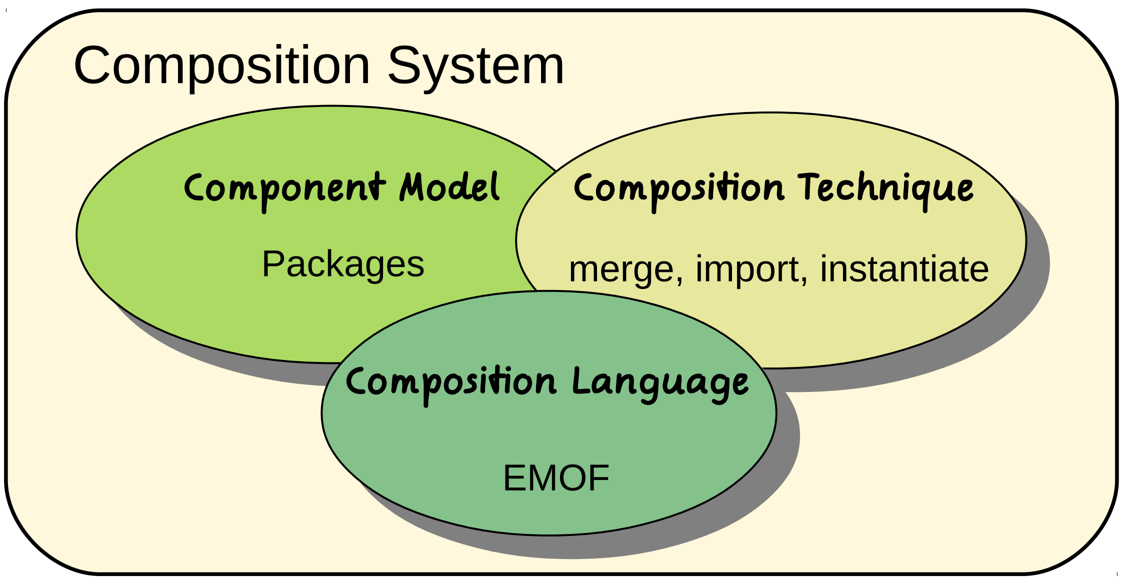
→ Metamodels can be composed from packages



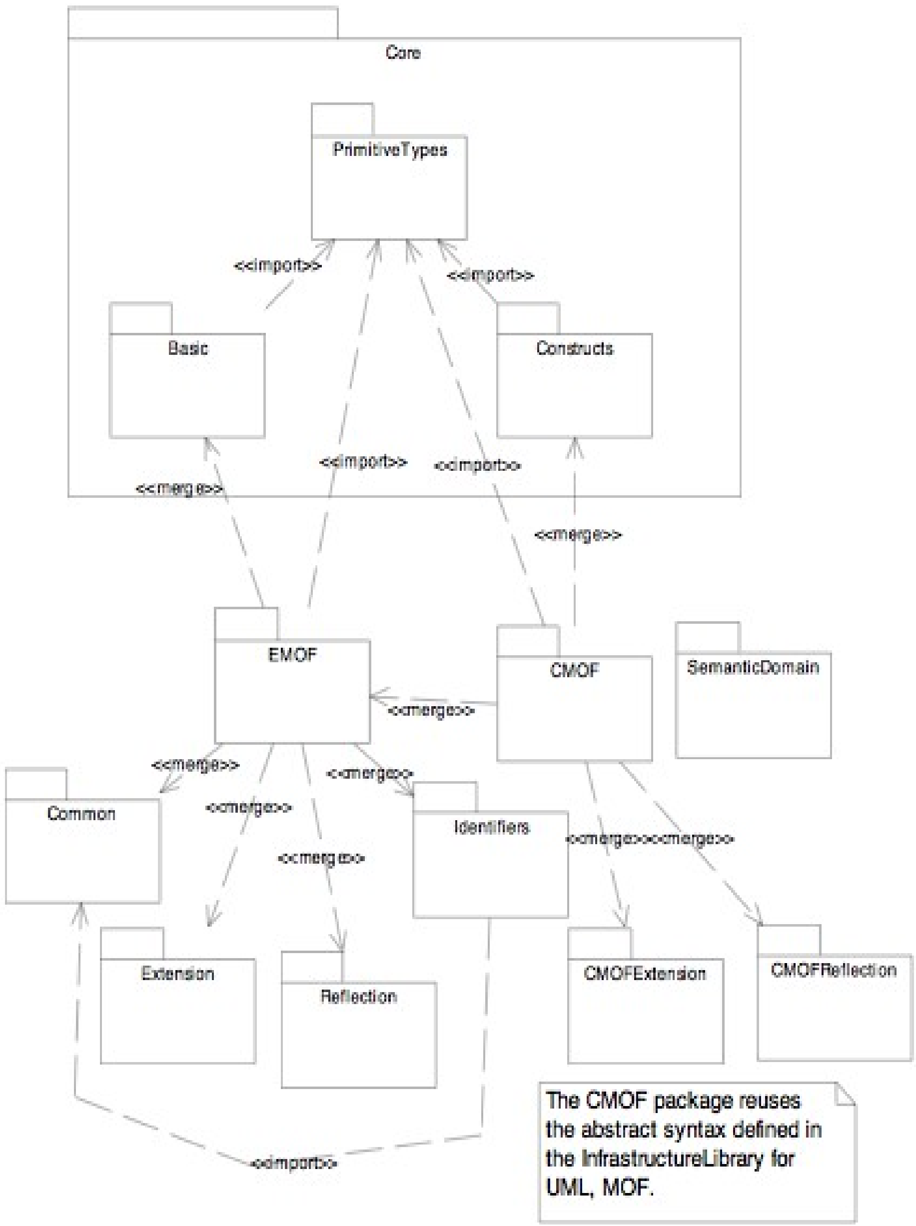
Ex.: EMOF Class Composition by EMOF Package Merge

[MOF]





Ex: CMOF Package Composition from UML Core and EMOF



[MOF]

4.5.3.a Composing UML Metamodels in the MOF Technical Space

Benefit of UML-Metamodeling for MDSD Tools and Model-Driven Applications

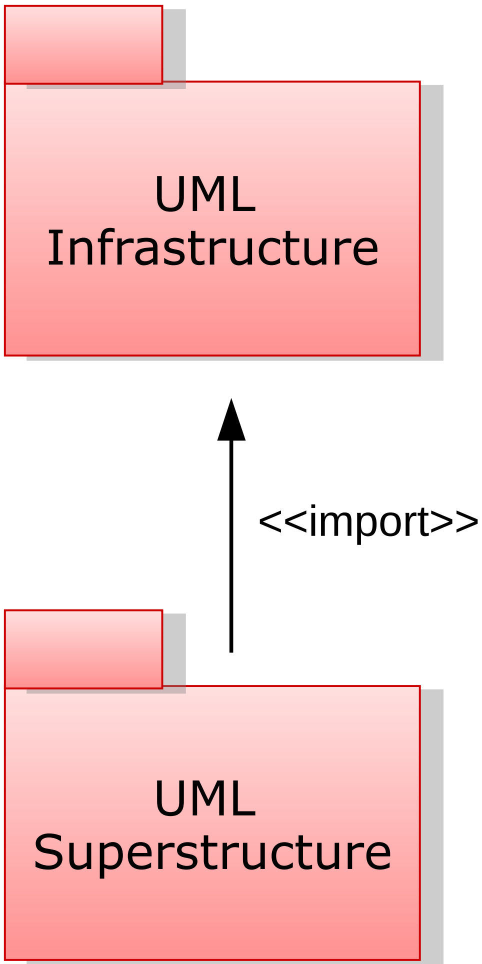
The language report of UML uses a simple metamodel algebra for the bottom-up composition of UML language.

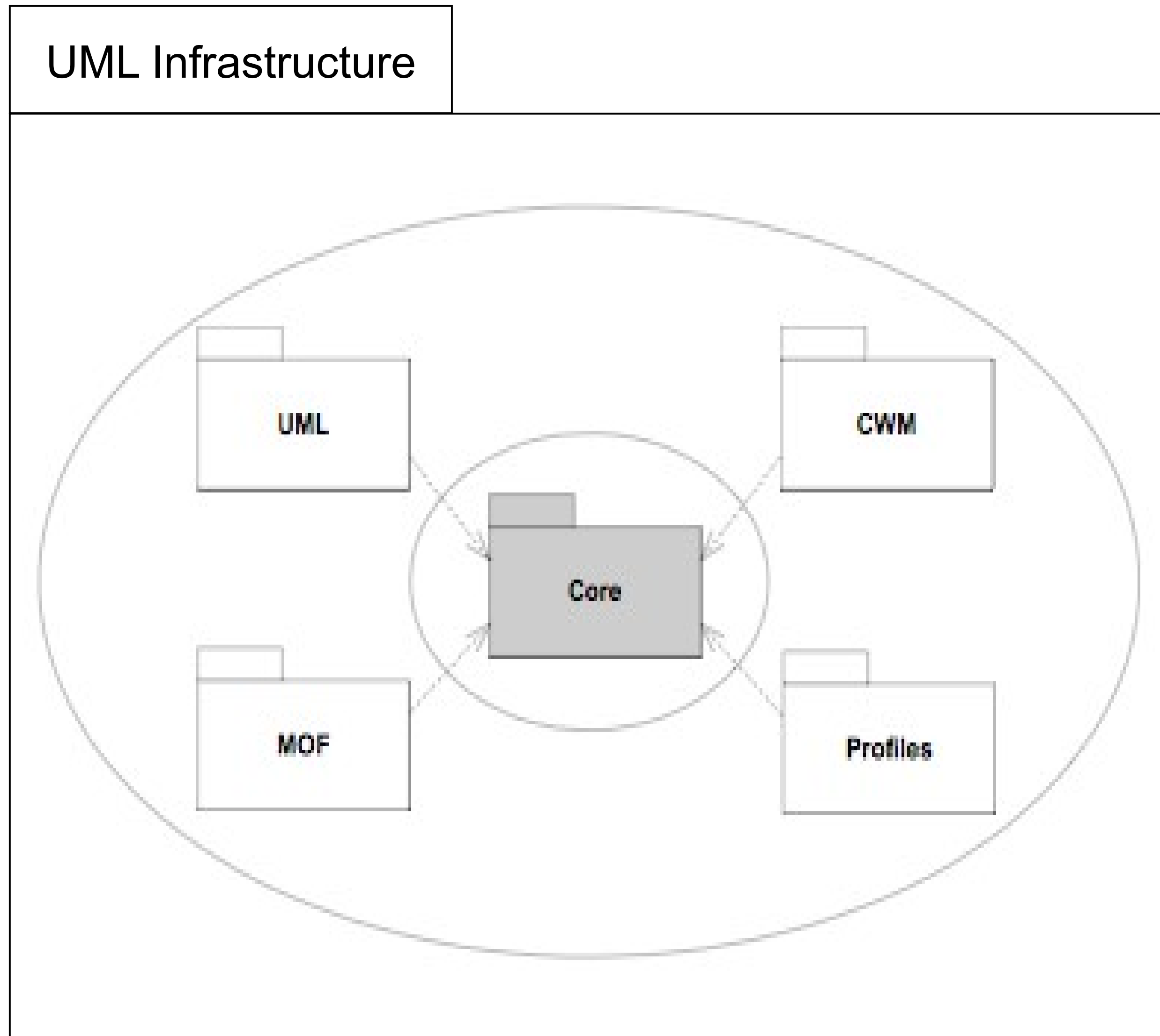
The UML-metamodel is a "logic" metamodel, because it is *composed:*

- Definition of merge operator composing metaclasses and metaclass-packages
- Defined in composable **packages**
  - With a clear **CMOF**-package architecture
  - uniform **package structure** and context-sensitive semantics for all diagrams such as Statecharts (UML-SC), Sequence Diagrams (UML-SD), etc.
- **Schemata for repositories** for uniform description of tools, materials, code, models (**metamodel-driven repositories)**
- **Exchange format** (XMI)

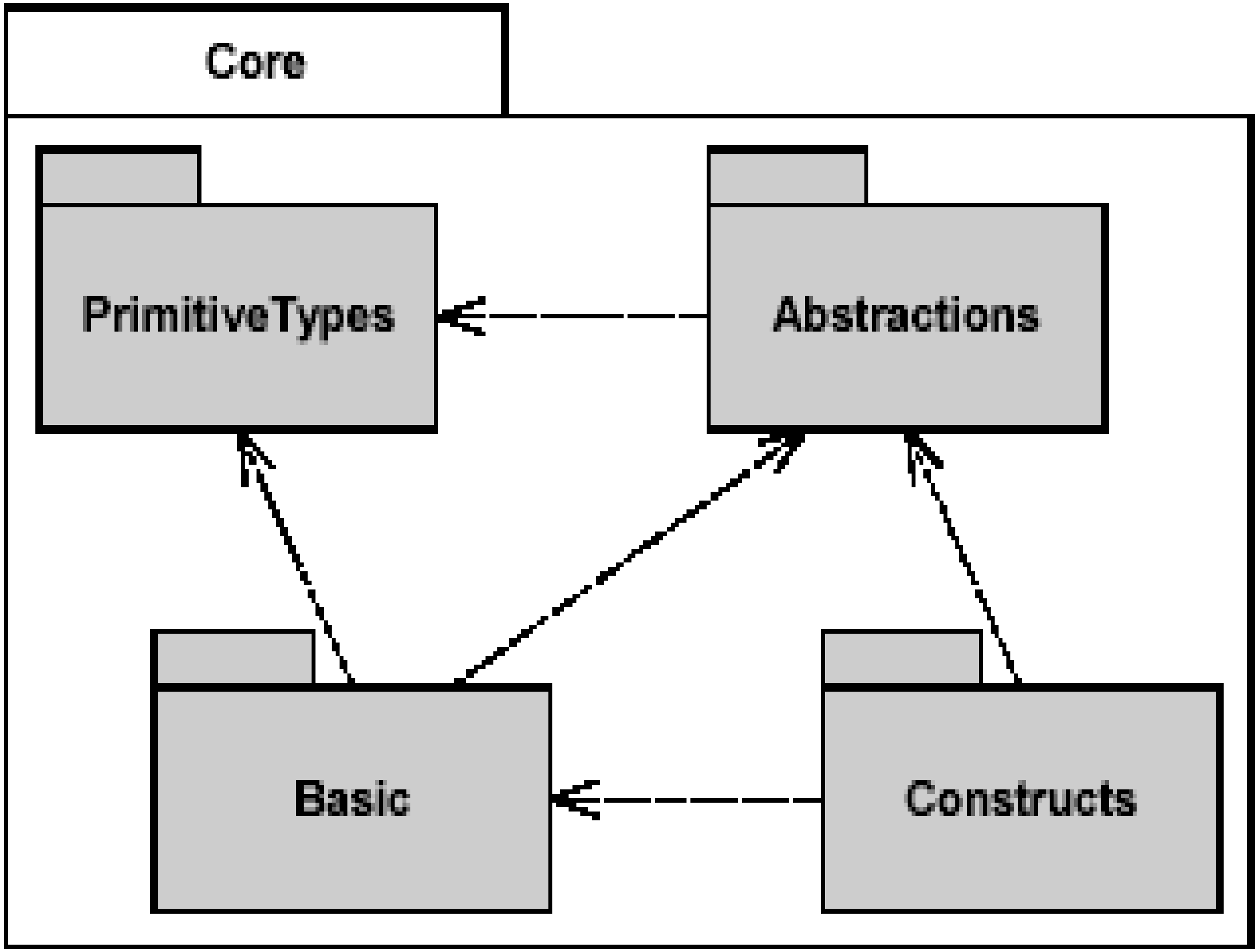
- The UML infrastructure can be used by MDSD applications

Coarse-Grain Structure of UML on M2





Core Package of the UML-Infrastructure Metamodel (M2)

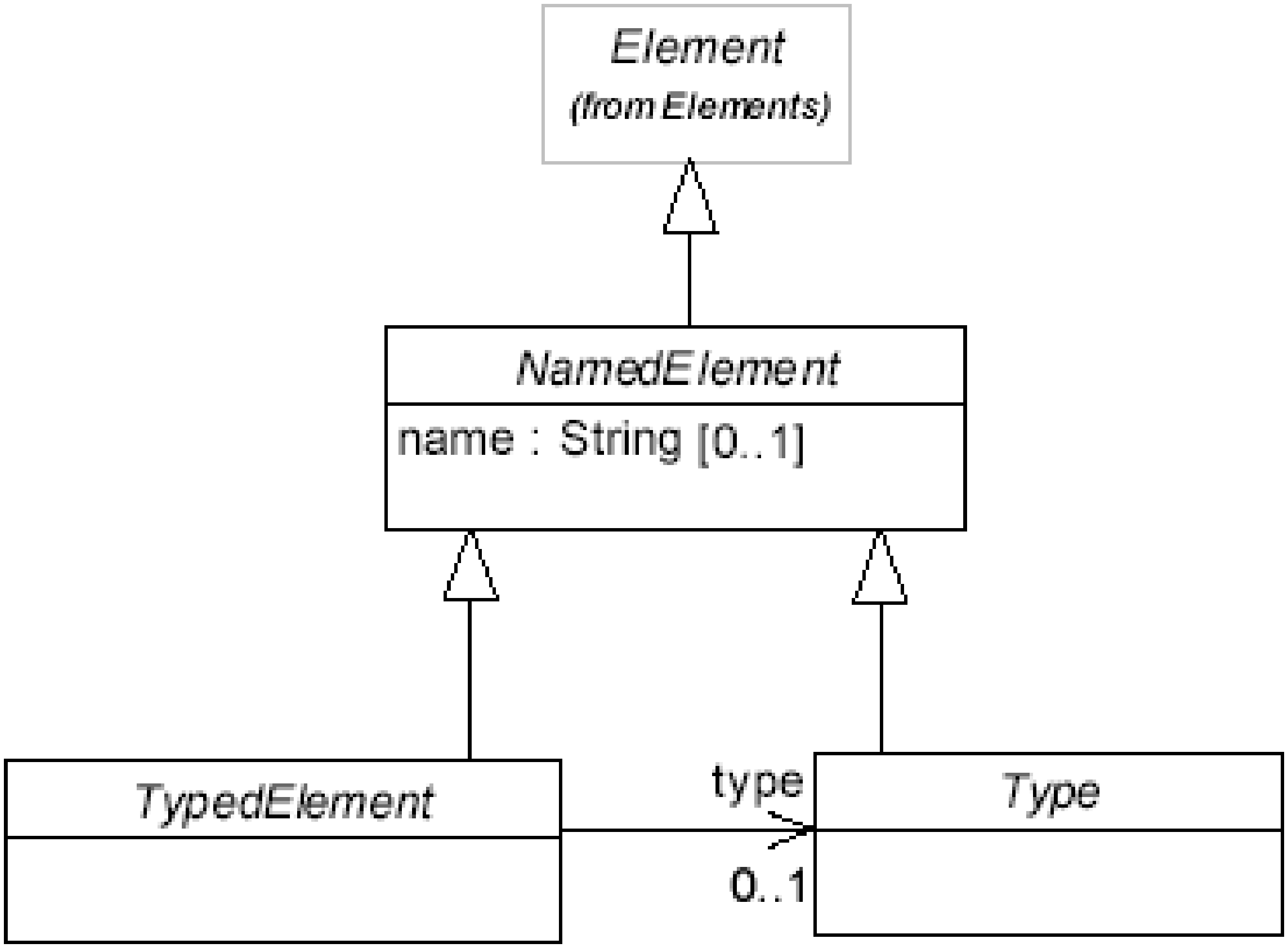


Basic: basic constructs for XMI
Constructs: Metaclasses for modeling

Abstractions: abstract metaclasses
Primitive Types: basic types

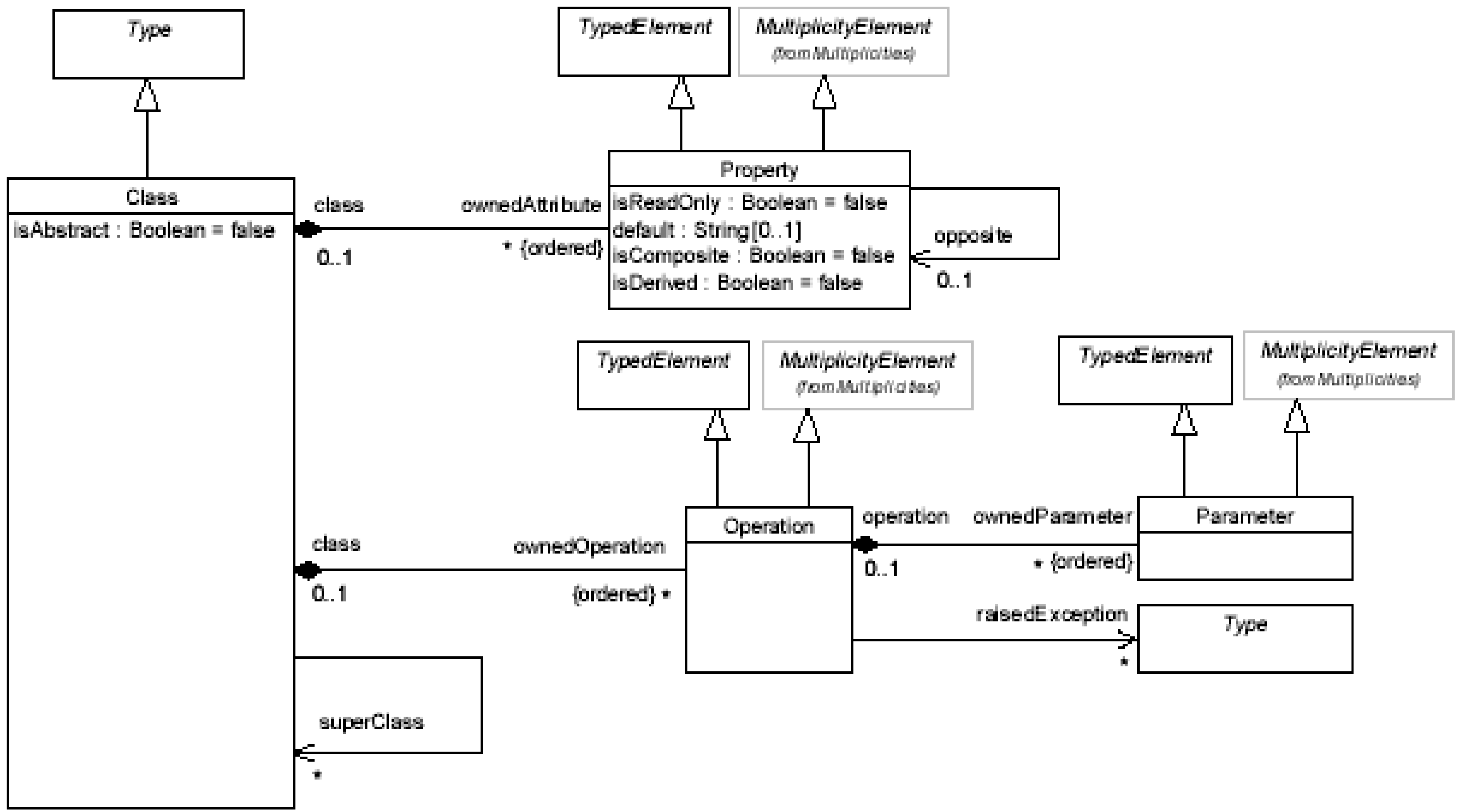
From: UML 2.0 Infrastructure Specification; OMG Adopted Specification ptc/03-09-15

Package Basic: Uses Types from CMOF



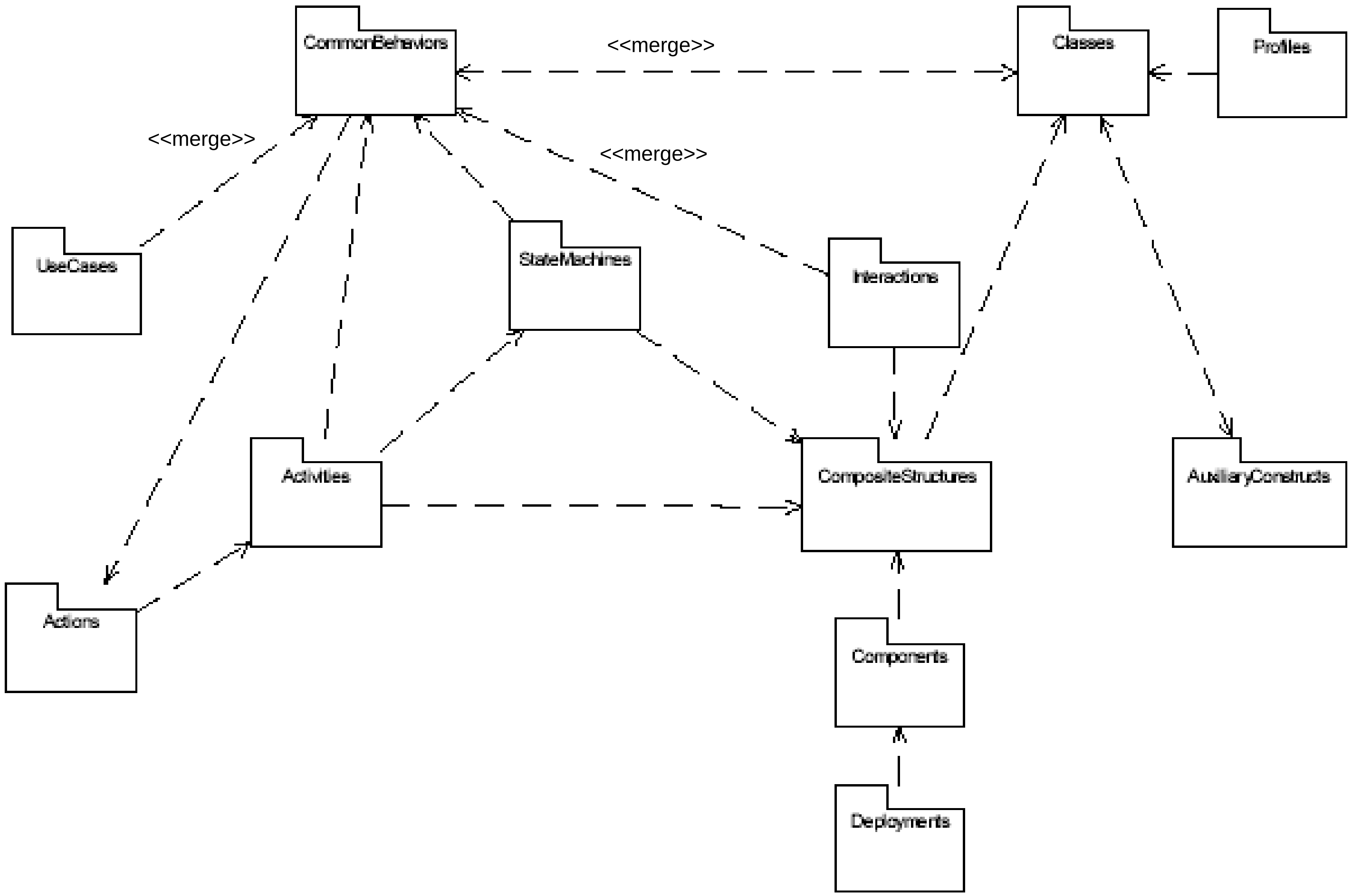
From: UML 2.0 Infrastructure Specification; OMG Adopted Specification ptc/03-09-15

Package Basic: Classes



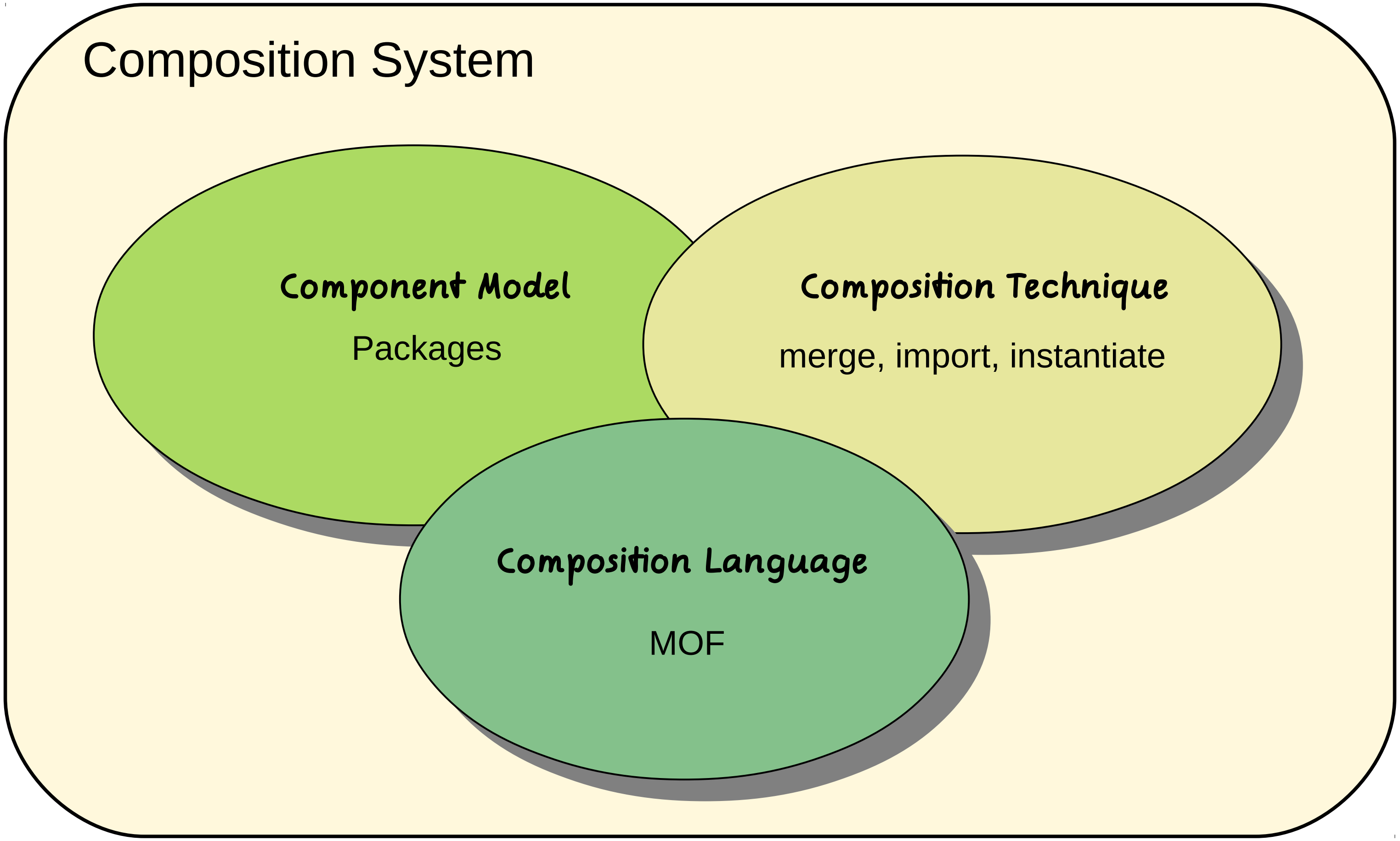
From: UML 2.0 Infrastructure Specification; OMG Adopted Specification ptc/03-09-15

Package Composition Architecture UML 2.0 (M2)



From: UML 2.0 Infrastructure Specification; OMG Adopted Specification ptc/03-09-15

Metamodel Composition – the Composition System of the UML Language Report



4.6 Mega- and Macromodels

In a technical space, a ***megamodel*** is an infrastructure for models and metamodels, systematically linking a set of models

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Megamodels

- A ***megamodel*** is a model for a set or graph of models.
  - The graph of models is an instance of the megamodel (element of the of the language)
- Usually, a technical space has one or several megamodels on M1, linking many models on M1
  - Clearifying the relationships of the M1 models by model transformations, model mappings, and model compositions
  - A megamodel uses the model management system of the technical space

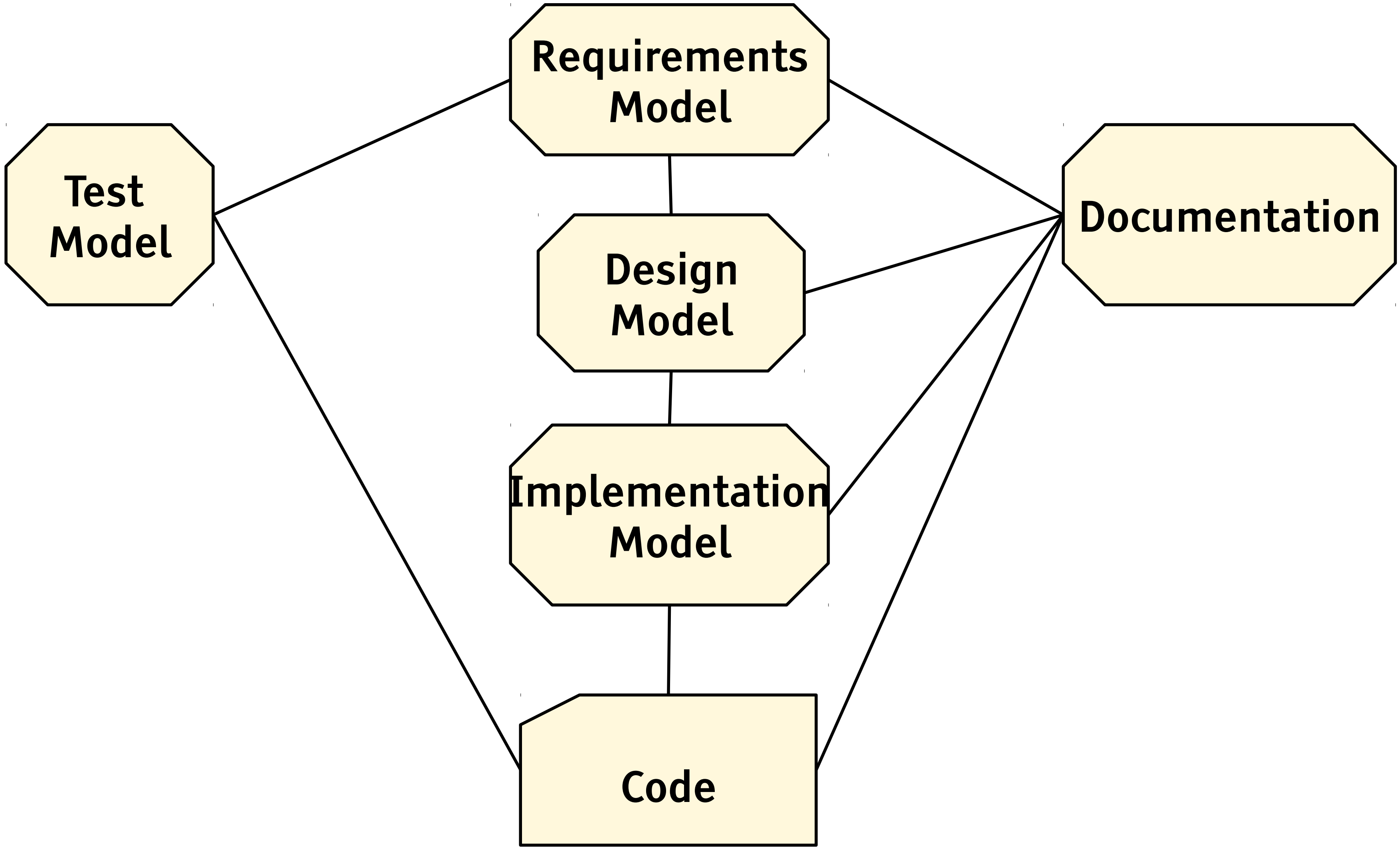
> The idea behind a *mega-model* is to define the set of entities and relations that are necessary to model some aspect about model-driven engineering (MDE).
>
> [Favre]

Macromodels – Megamodels with Consistency Rules

- A ***macromodel*** is a model for a set or graph of models *fulfilling some consistency constraints over the models and their elements*
  - The graph of models is an instance of the megamodel (element of the of the language)
  - The graph of models obeys wellformedness constraints
  - There are **fine-grained relations** between model elements of the models, which also follow *consistency constraints*
    - **Trace mappings** between tools, materials, automata
    - **Synchronization relations** for updating

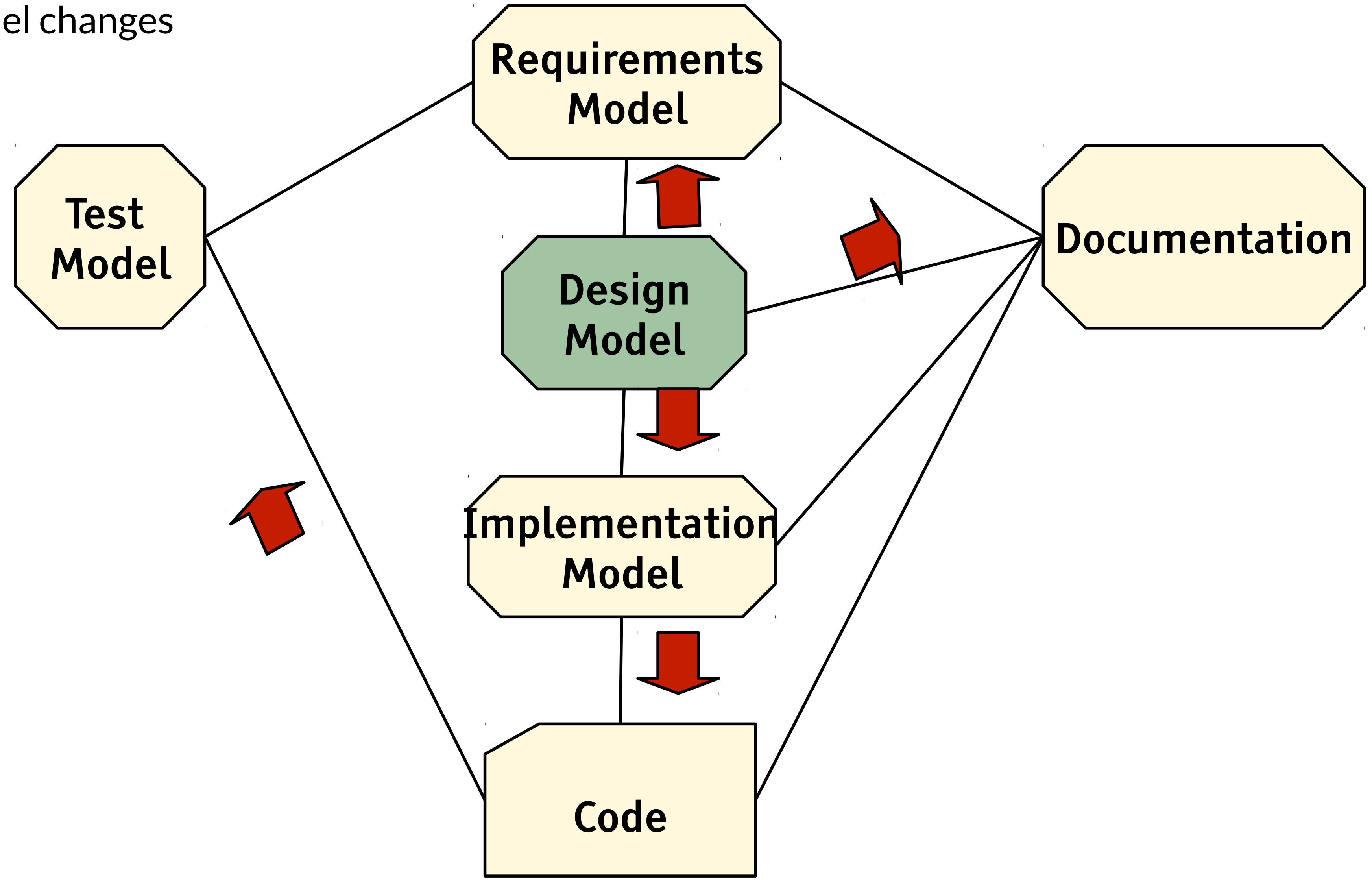
Model Synchronization in Macromodels

- ***Model synchronization*** keeps a set of connected models (the ***crowd***) *in* sync, i.e., consistent



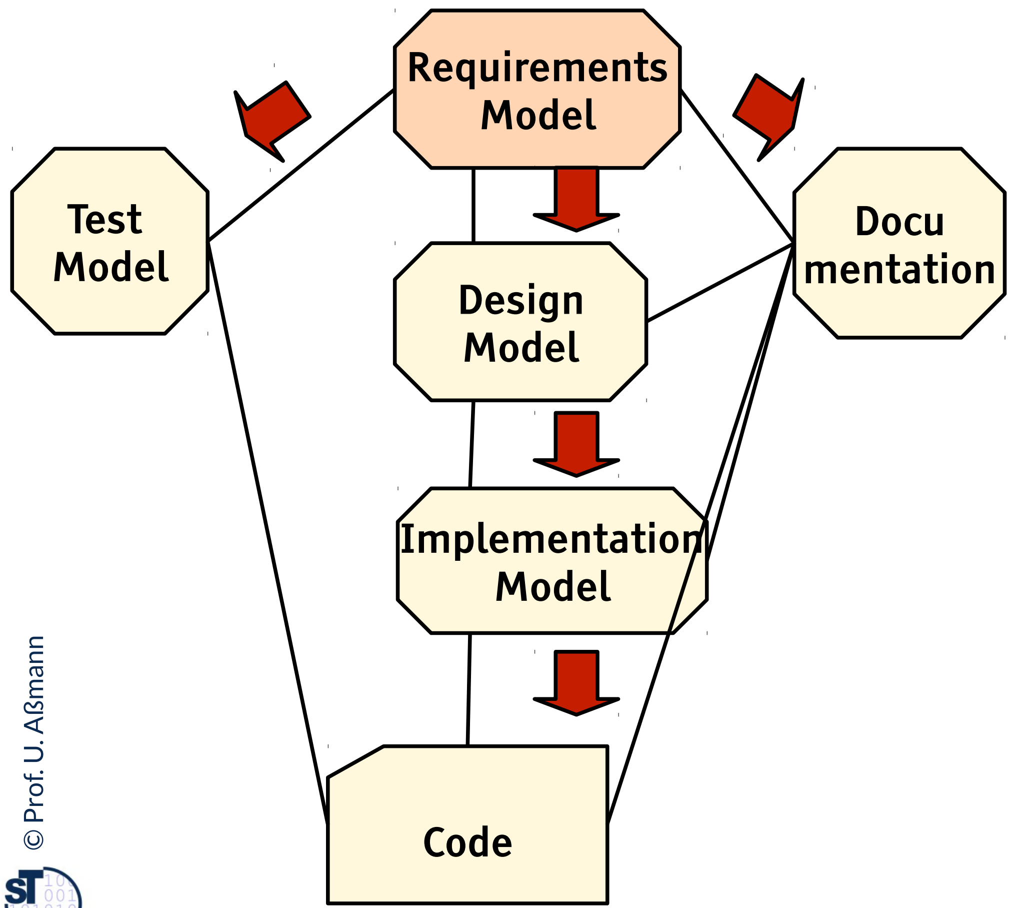
Model Synchronization in Macromodels

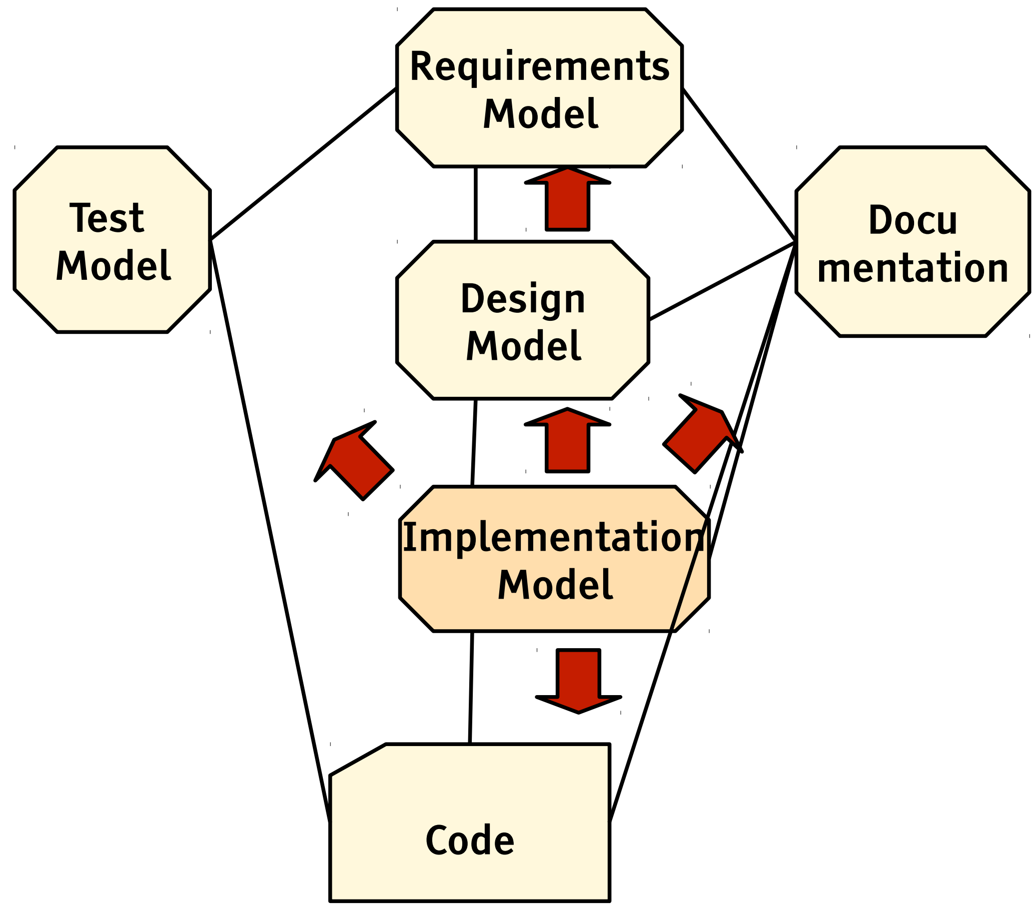
- In model synchronizsation, if an edit has occurred in a ***origin model***, all other connected models of a crowd (***dependent models***) are updated instantaneously, when one focus model changes



Round-Trip Engineering Changes the Model-in-Focus of the Crowd

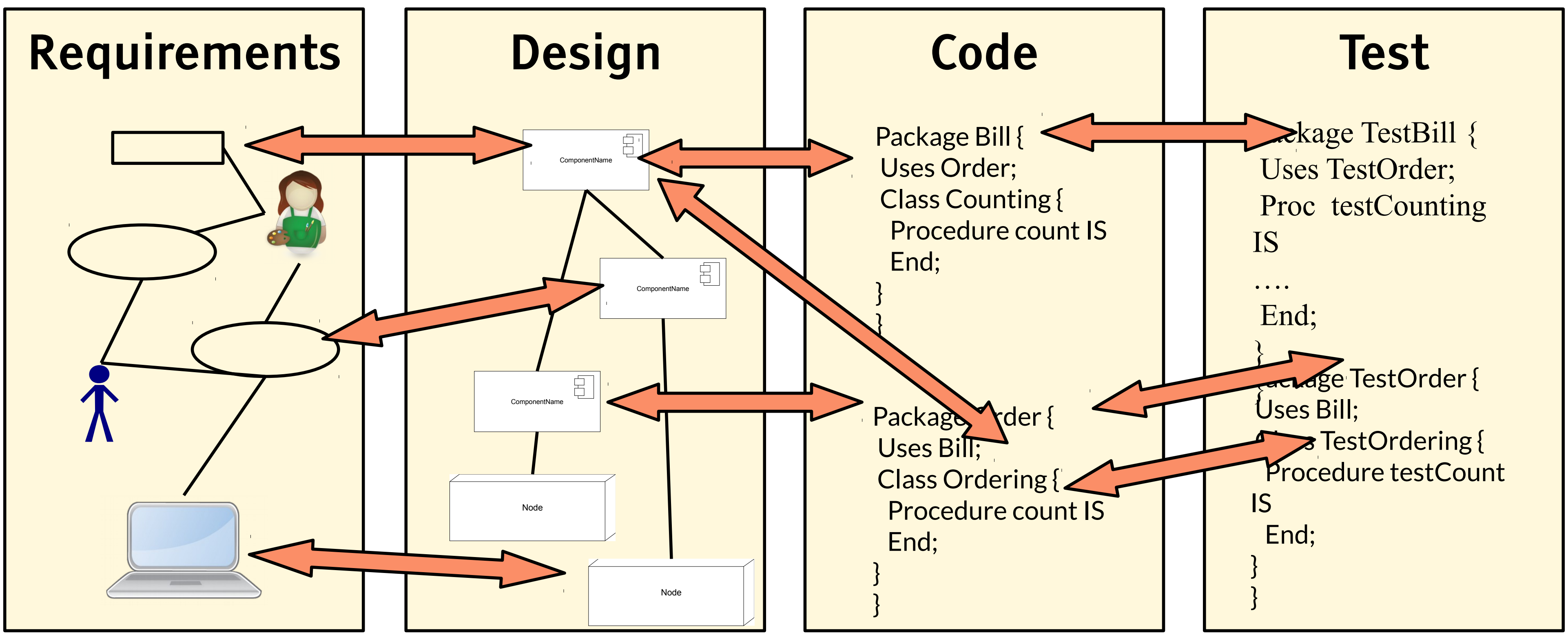
- But always performs model synchronization as a basic step





Q12: The ReDeCT Problem and its Macromodel

- The **ReDeCT problem** is the problem how requirements, design, code and tests are related (→ V model)
- Mappings between the Requirements model, Design model, Code, Test cases
- A **ReDeCT macromodel** has maintained mappings between all 4 models

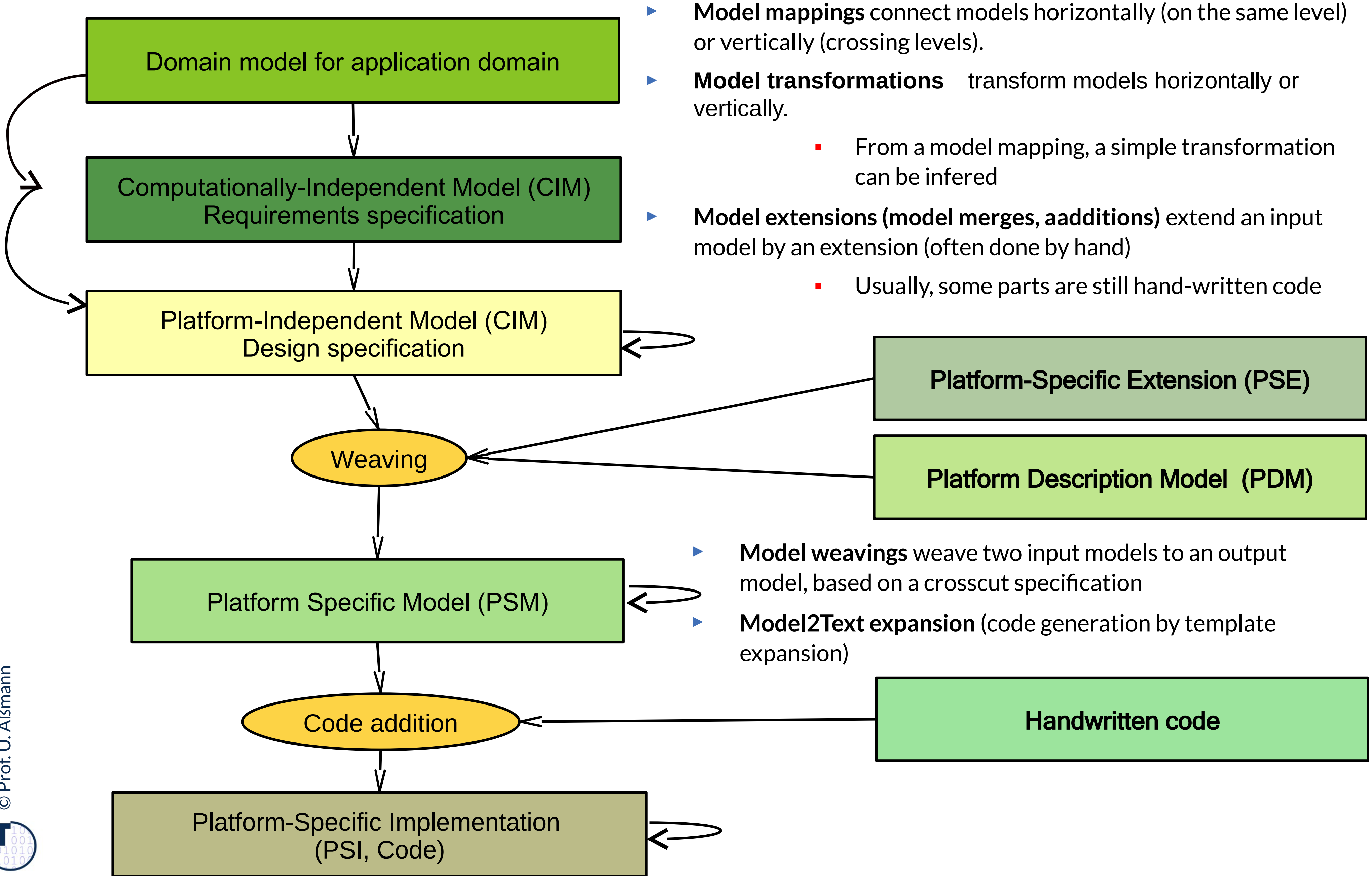


Advantages of Model Mappings in Macromodels

- **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

Q9: Model Mappings and Model Weavings in the MDA Macromodel

- **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 extensions (model merges, aadditions)** extend an input model by an extension (often done by hand)
  - Usually, some parts are still hand-written code



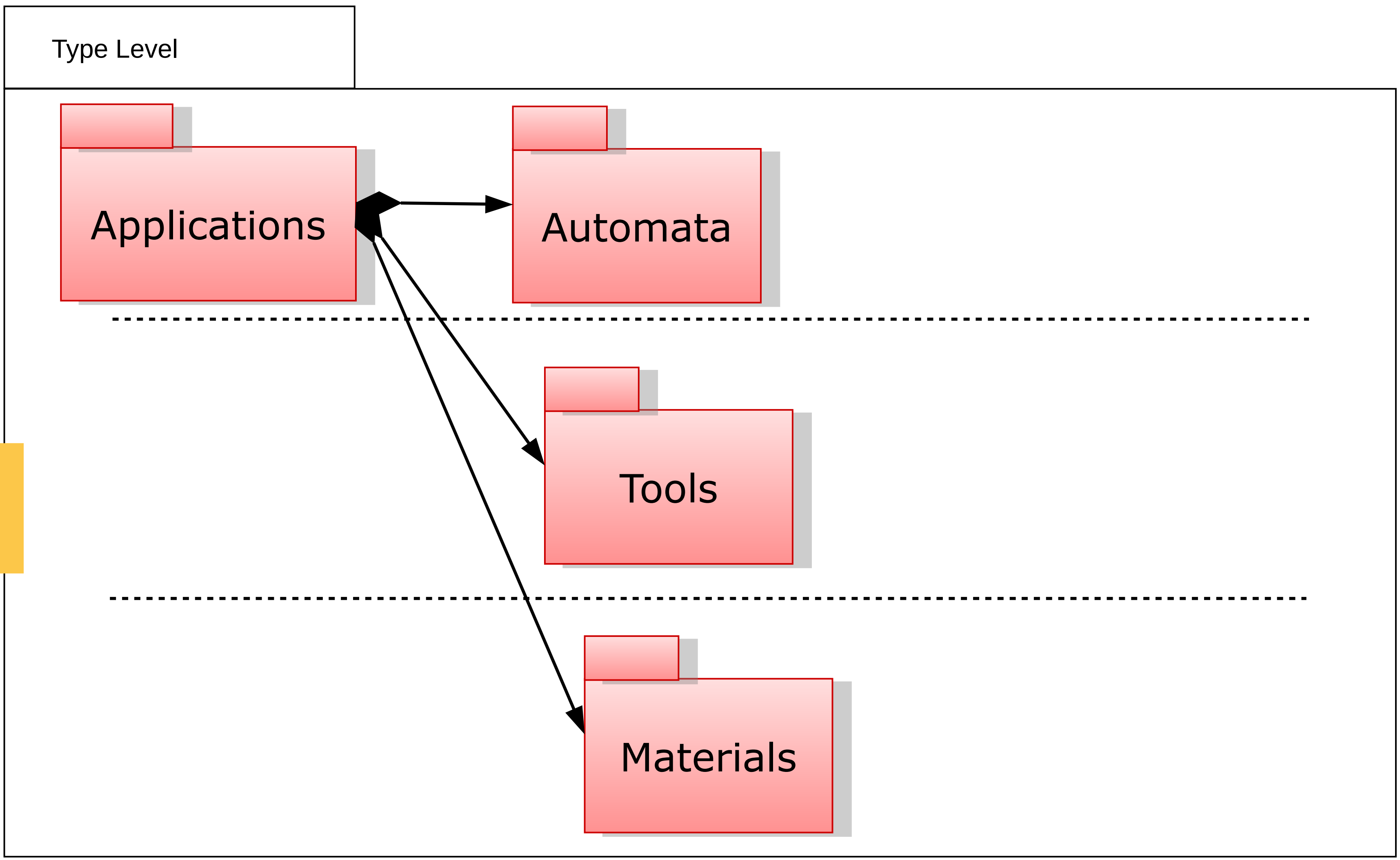
- **Model weavings** weave two input models to an output model, based on a crosscut specification
- **Model2Text expansion** (code generation by template expansion)

4.7. Pattern Languages in a Technical Space

- In a TS, several pattern languages may be used to structure the relationship of models and metamodels
- TAM can be used as Pattern Language on all levels in the metahierarchy
- However, there may be more pattern languages associated to a technical space
- Pattern languages can be expressed as stereotypes

A Pattern Language Useful for all Technical Spaces TAM Structures on M1

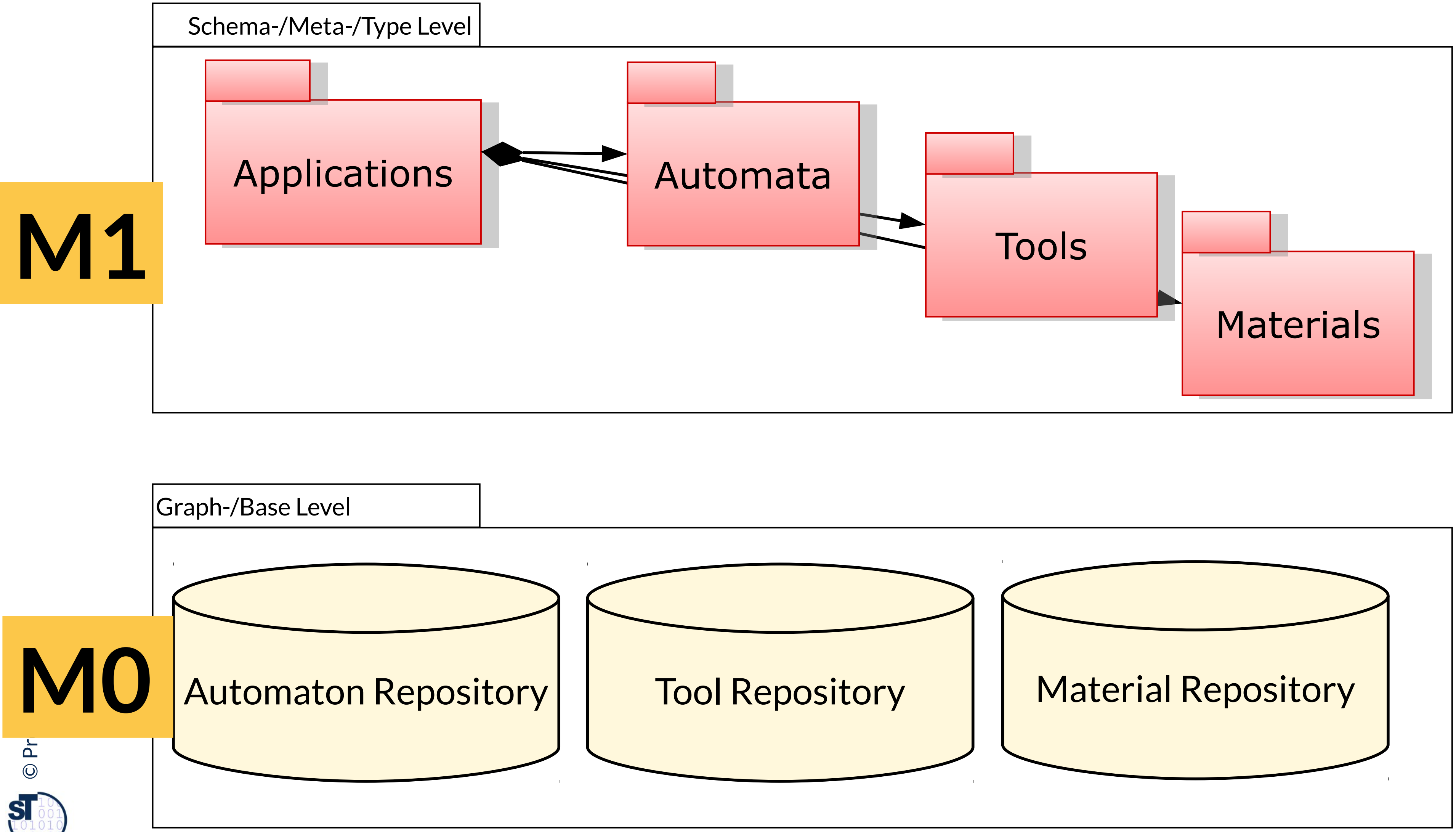
- On M1, application class models need to define (stereotype) tools, automata, and materials.



M1

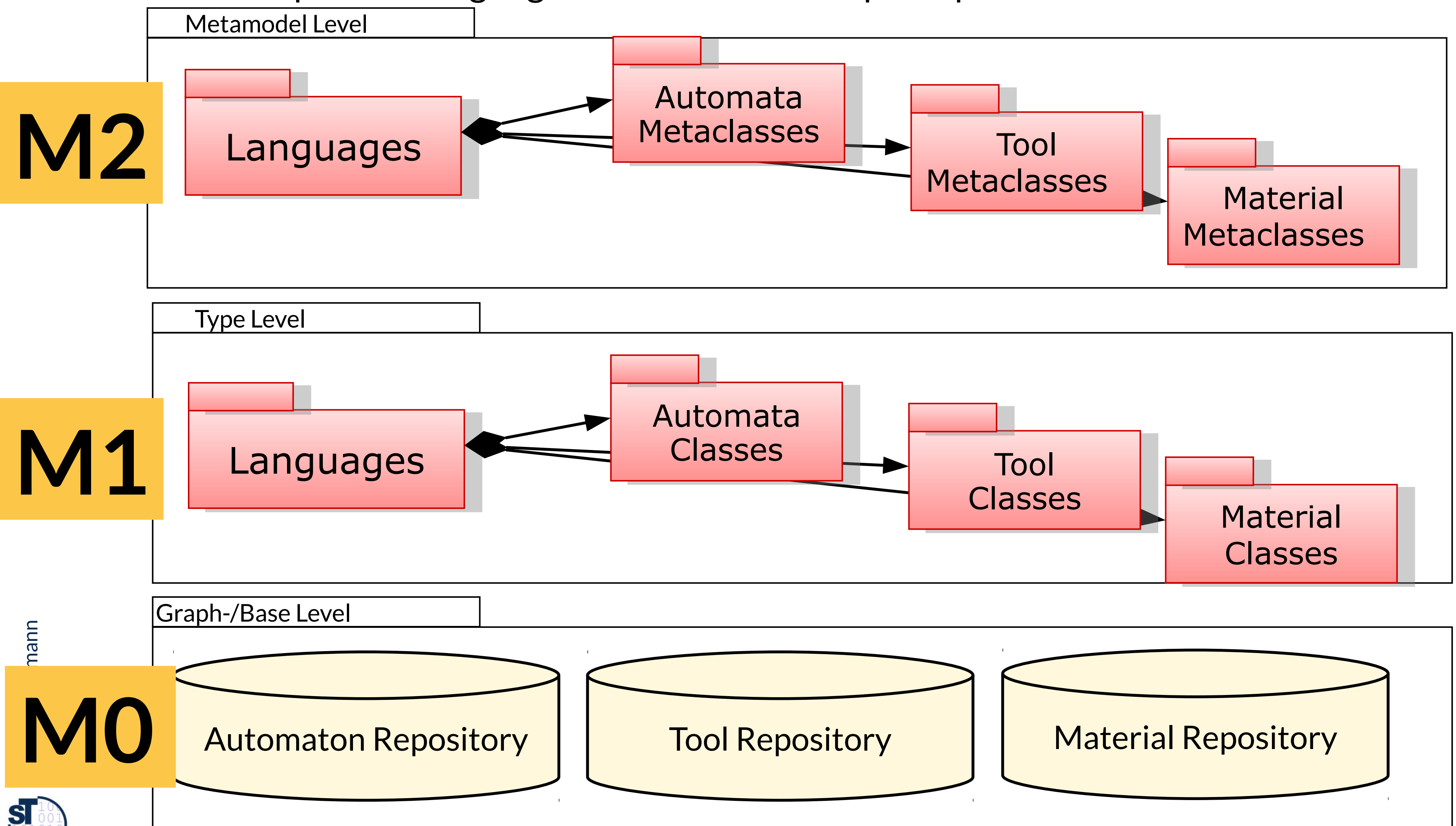
TAM Structures on M1 Provide Types for Objects in Repositories on M0

- On M1, application class models need to define (stereotype) tools, automata, and materials.



TAM Structures on M2 Provide Language Concepts for Stereotypes for Classes in M1

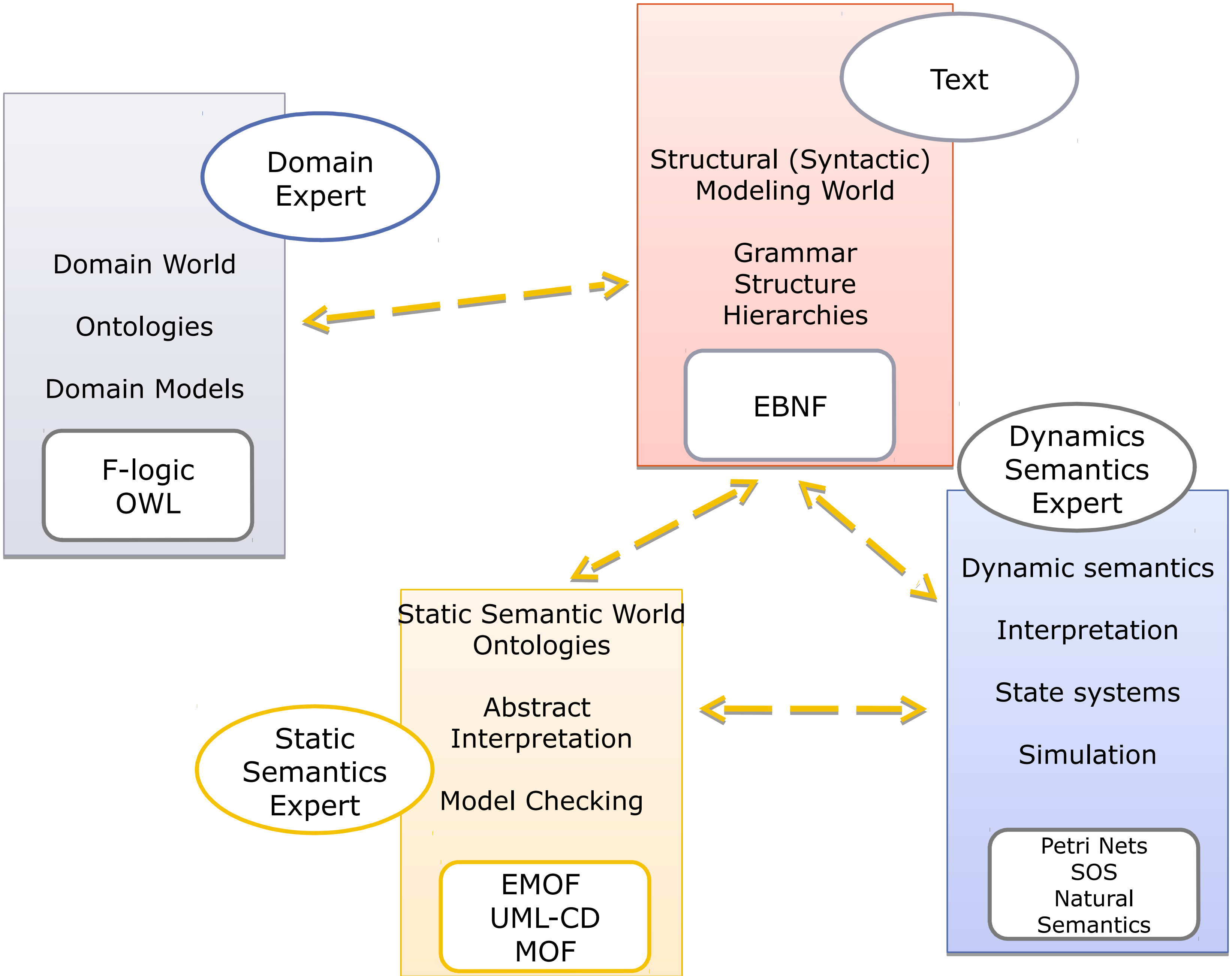
- On M2, TAM forms a DSL for stereotypes on M1
- Other pattern languages can use the same principle



4.8. Briding Technical Spaces

- While one tool/application may live in one TS, for the communication with other tools/applications, **technical space bridges** have to be built.
- Usually, a technical spaces has a subsystem for **technical space bridging.**

An Application May Need Several Technical Spaces



The End