

13. Tool, Automata, Material Methodology and Metaclass Role Model (TAM)

Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
<http://st.inf.tu-dresden.de/teaching/most>
WS 21-1.1, 29.01.22

- 1) Taxonomy of applications, tools and materials
- 2) TAM as metamodel pattern language
- 3) TAM for Layering of Applications
- 4) Basic Functions of Standalone Tools
- 5) Graph-Fact-Isomorphism

- ▶ Züllighoven, Heinz: Object-Oriented Construction Handbook; dpunkt.verlag 2005
- ▶ Riehle, D., Züllighoven, H.: Pattern Languages of Program Design; Reading, Massachusetts: Addison Wesley 1995, Chapter 2
- ▶ Helmut Balzert. Softwaretechnik I+II. Verlag Spektrum der Wissenschaft
- ▶ [GOPPR] J.-P. Tolvanen, P. Marttiin, and K. Smolander. An integrated model for information systems modeling. In J. F. Nunamaker and R. H. Sprague, editors, Proceedings of the 26th Annual Hawaii International Conference on Systems Science, Maui, Hawaii, January 1993. IEEE Computer Society Press.
<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=284345>



13.1 Tools, Workflows and Materials as Pattern Language for Applications

A Tool or a Material?

- ▶ With tears in his eyes the violinist Aaron Rosand left his soul behind in a London hotel suite last week.
- ▶ That is how he described the sale of the instrument he had played for more than 50 years, the ex-Kochanski Guarneri del Gesù. The buyer was a Russian billionaire whom Mr. Rosand declined to identify and who paid perhaps the highest price ever for a violin: about \$10 million.
- ▶ “I just felt as if I left part of my body behind,” Mr. Rosand said on Wednesday, overflowing with metaphors for what the instrument meant to him. “It was my voice. It was my career.”
- ▶ Daniel J. Wakin. New York Times Oct 21, 2009.
 - http://www.nytimes.com/2009/10/22/arts/music/22violin.html?_r=0

Human Beings Use Tools

A **tool (Werkzeug)** is a thing helping to do actions faster as by hand.
An **IT-tool** is a tool running on a computer. It is active and triggered by humans or programs.
A **data tool** is an IT-tool working with data.

A **software tool** is an IT-tool working on software.
A **modeling tool** is a software tool working on models.
An **application** contains several data or software tools.
A **tool class** is a class of an application providing functionality of a tool.
A **standalone tool** is a software tool persisting its materials.

A **machine tool (Werkzeugmaschine)** is a tool for production of other tools.

A **software machine tool (Software-Werkzeugmaschine)** is a software tool for production of other software-tools.

- ▶ SW-machine tools are the basis of all productivity and wealth in a society.

“Tools and Material”-Metapher (TAM) for Programming Applications

- ▶ **Tool:** A **tool(-object, -class)** is an active software object (-class) that can be used to change materials
 - Tools can be used by humans (interactively, batch) or by other tools, or by automata (workflows)
- ▶ **Material:** A **material (-object, -class)** is a passive object (-class) handled by a tool
- ▶ **Automaton (Workflow engine):** An **automaton** is an operational workflow orchestrating together several tools
- ▶ The **collaboration** of Tool, Automaton and Material classes can be described by a collaboration scheme (role model, Rollenmodell) (see Softwaretechnologie, DPF).

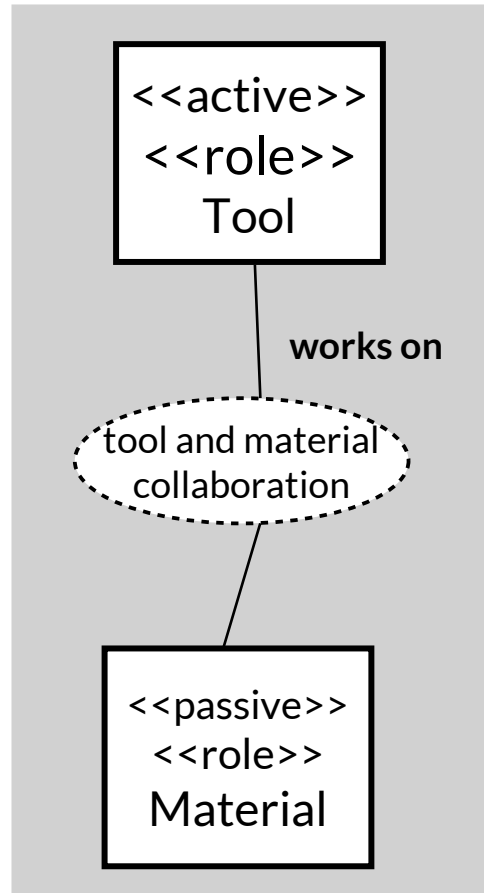
**All applications consist of tool-objects in workflows working on material.
(Züllighoven principle)**

[Züllighoven, Heinz: Object-Oriented Construction Handbook; dpunkt.verlag 2005]

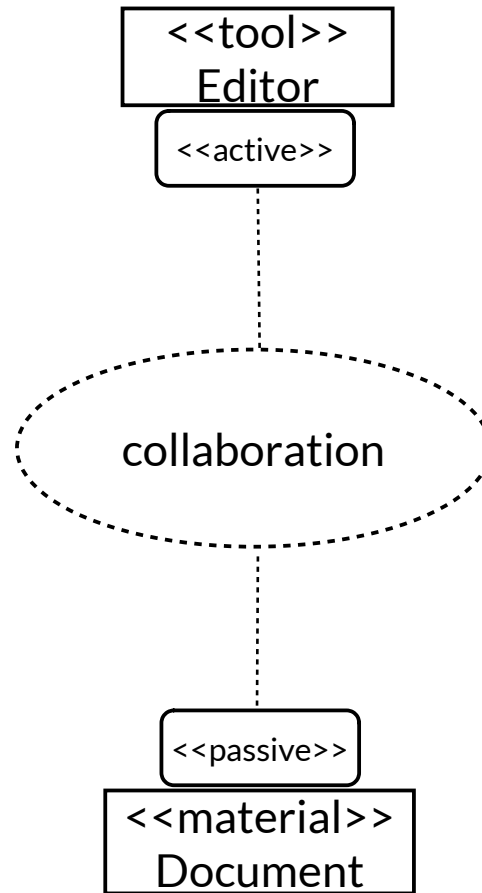
Are “Tools” and “Materials” Natural or Role Types?

- ▶ A thing can be a tool, but also a material of another tool
- ▶ Therefore, “tool” and “material” are *roles*.
 - Tool and Material and Workflow/Automaton form metaclasses on M2, instantiated from metametaclass roles on M3
- ▶ The TAM metaphor is a *role type model* indicating
 - which naturals can play which TAM role
 - How naturals play together in a TAM collaboration

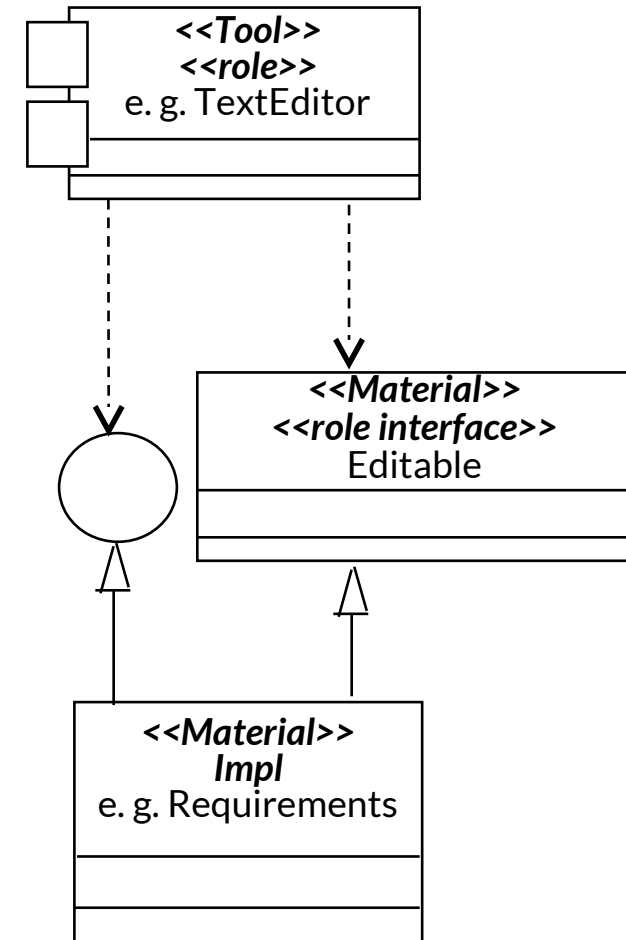
Tool and Material – Metaphor can be Realized in Many Designs of Tools



Conceptual Pattern



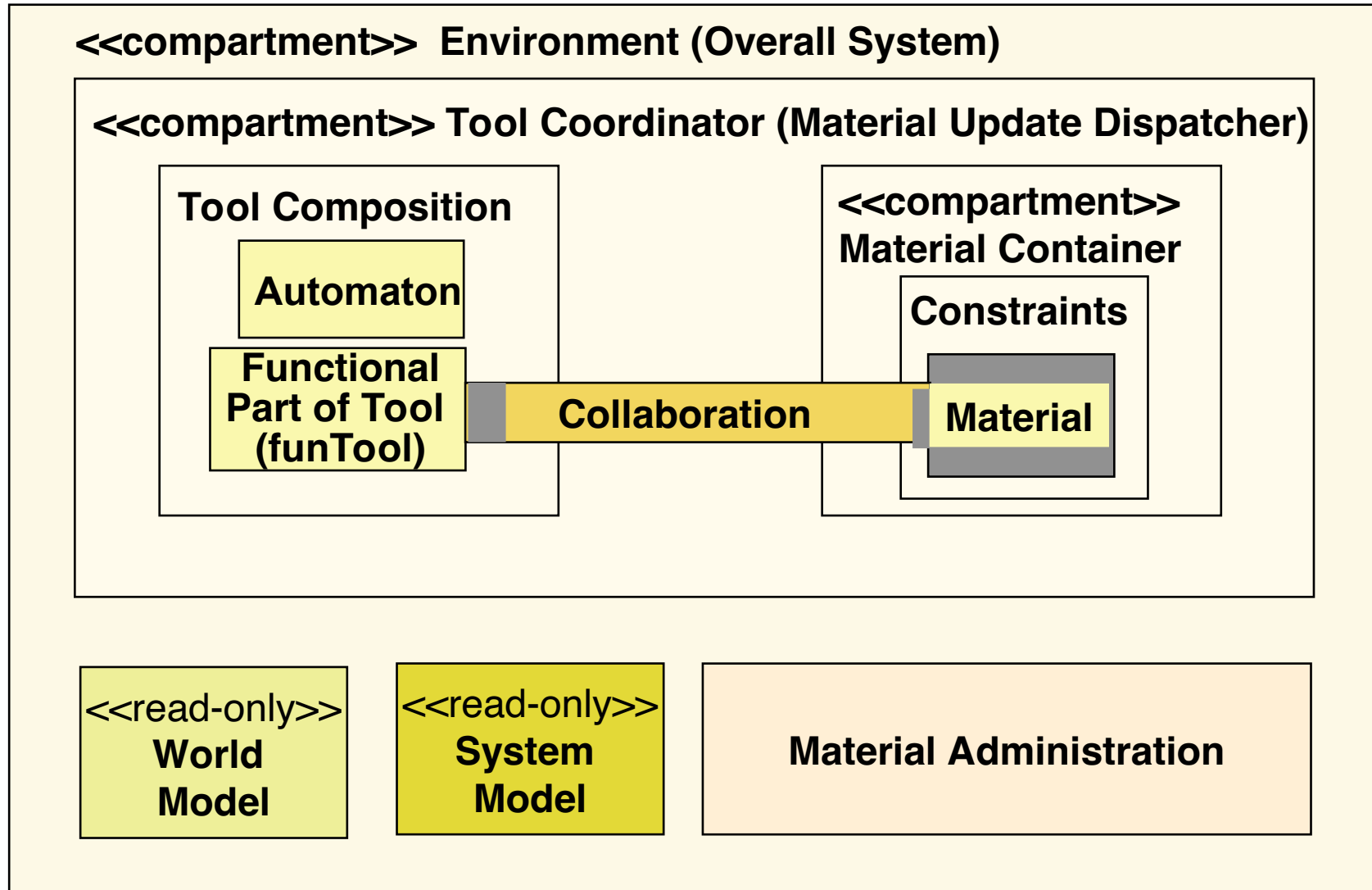
Design Pattern



Implementation

[Züllighoven, H.: Object-Oriented Construction Handbook; dpunkt.verlag Heidelberg 2005, S. 87]

Full TAM Pattern Language Suggests an Architecture for Application Integration





13.2. 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

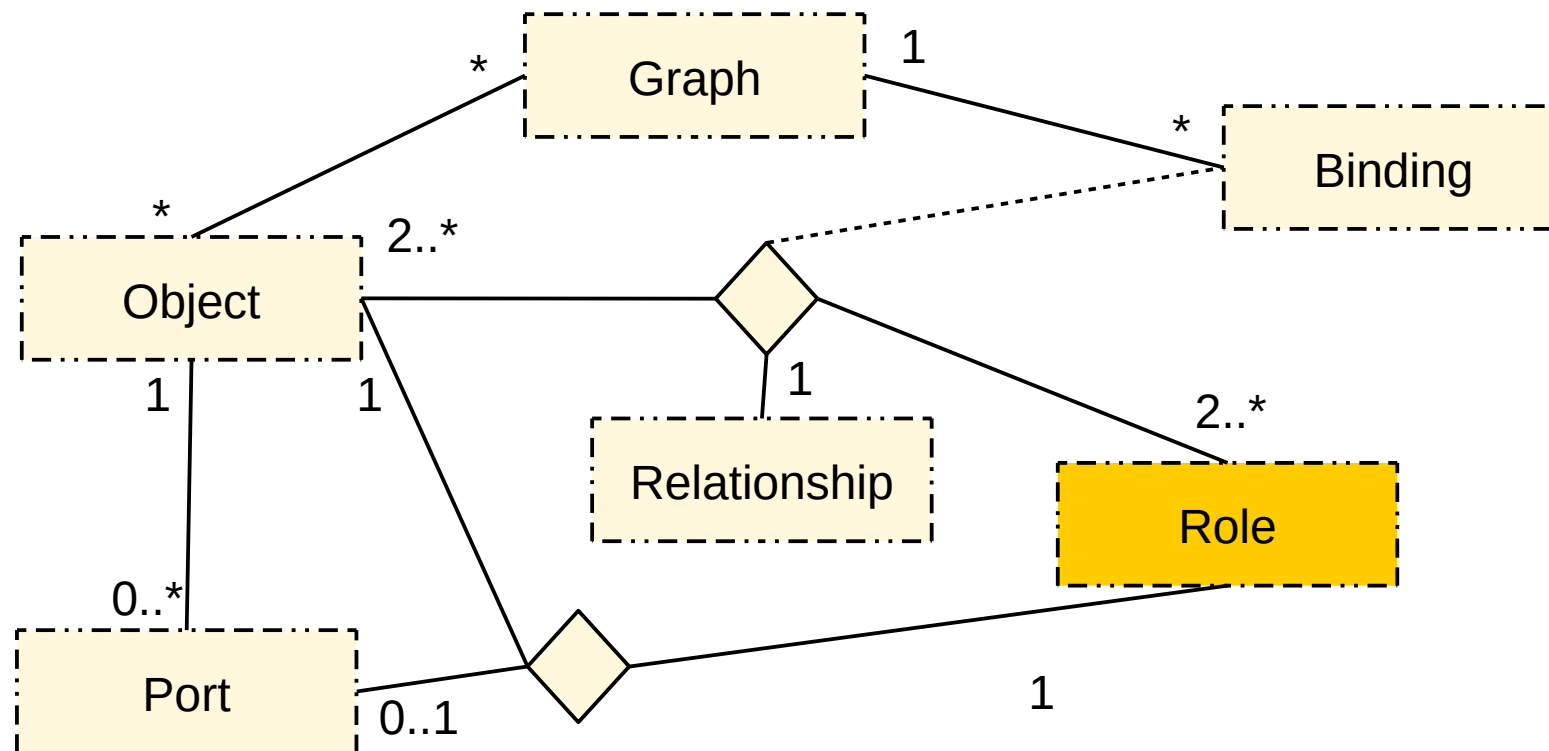


13.2.1 Roles in Metamodels and Metametamodels

- ▶ On M2 and M3, not only concepts and relations may be modeled.
 - It is a big old fight how “thick” M3 should be.
- ▶ Some M3 metametamodels have used other concepts:
 - **Roles:** A *role* is a context-dependent behavior of an object, a class, a metaclass or a metametaclass [GOPPR]
 - **Contexts:** A context groups many roles throughout an application, activates and deactivates them
 - **(Hyper-)Graphs:** A (hyper-)graph groups many classes or objects

Role-Based Graph Types in MetaEdit+

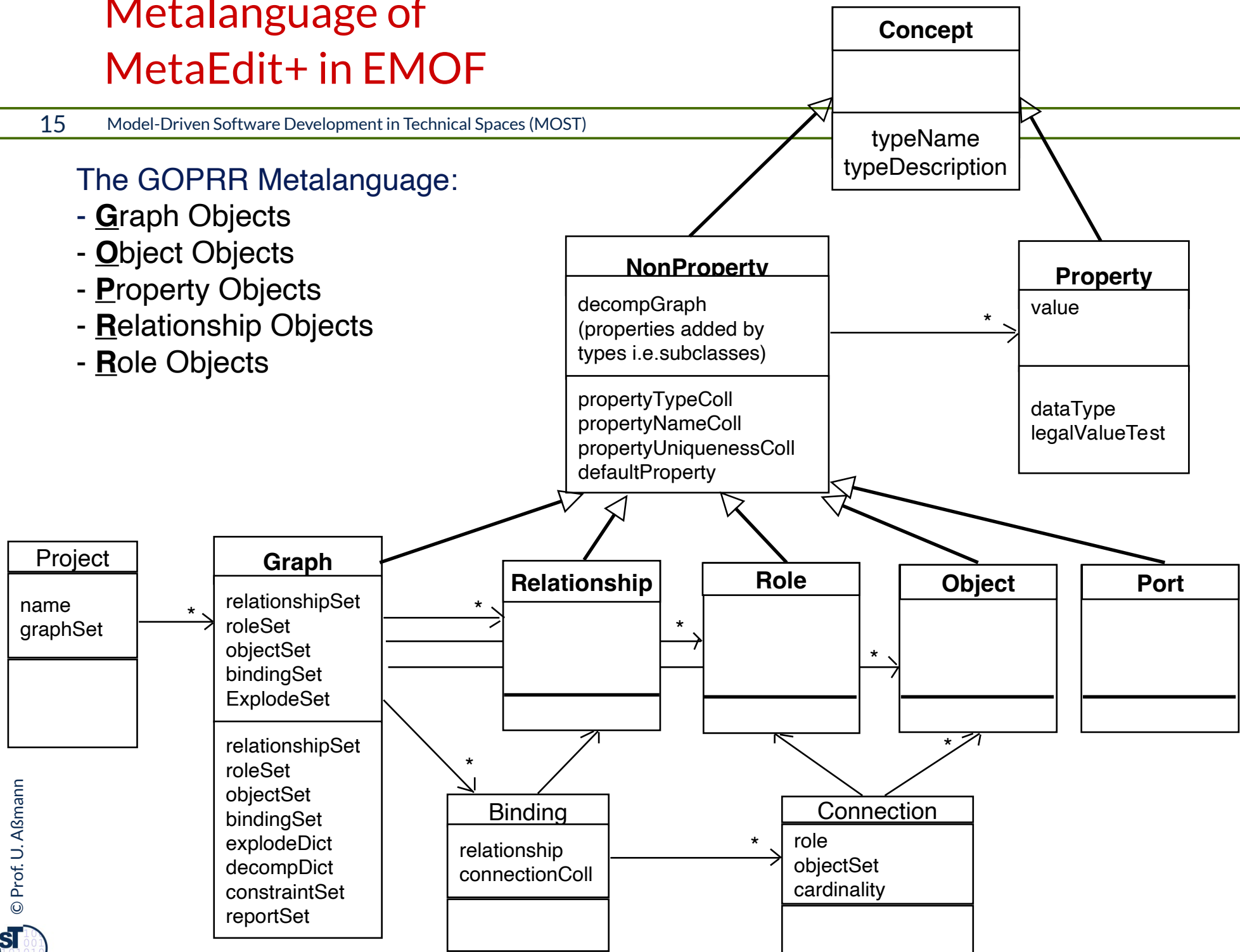
- ▶ [www.metacase.com]
- ▶ The tool MetaEdit+ uses the **graph schema (metalanguage) GOPRR**:
 - Objects and their Roles; Relationships
 - Allowed Bindings between all entities:
 - a binding consists of a relationship with roles and playing objects



Metalinguage of MetaEdit+ in EMOF

The GOPRR Metalinguage:

- **G**raph Objects
- **O**bject Objects
- **P**roperty Objects
- **R**elationship Objects
- **R**ole Objects





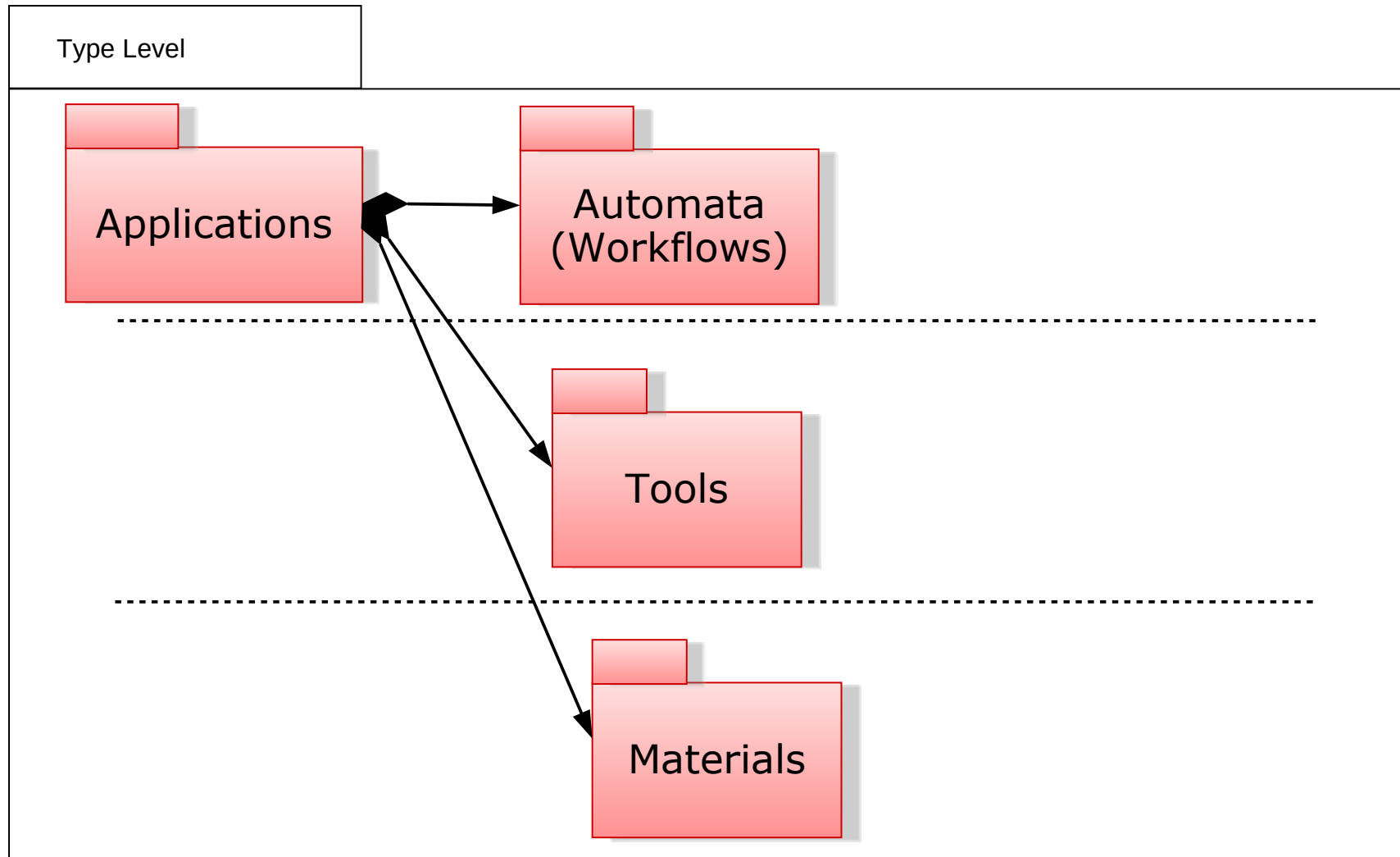
13.2.2 TAM Pattern Language

- 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

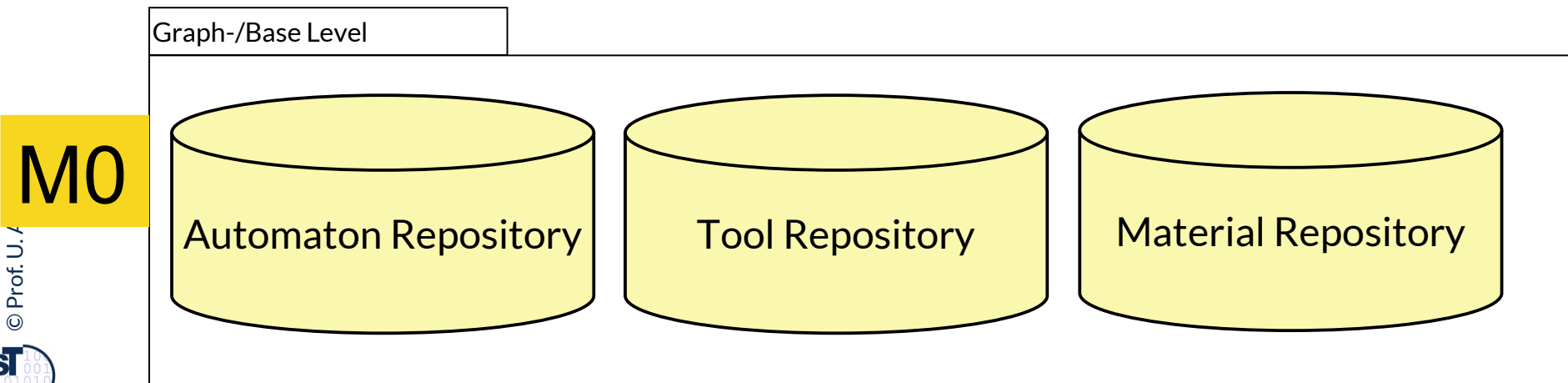
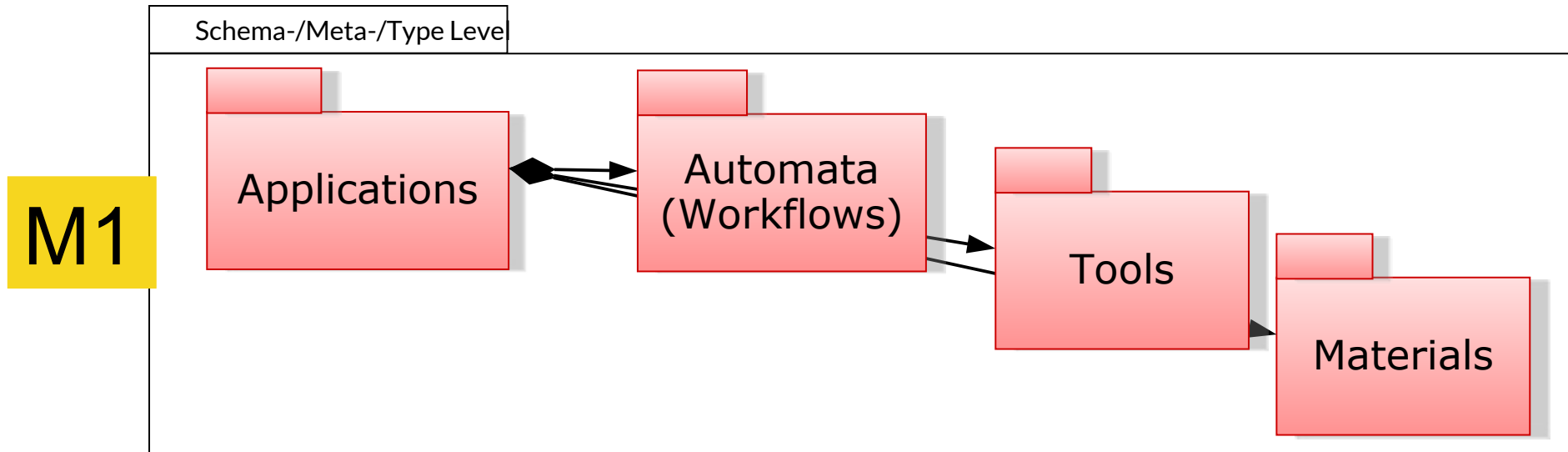
TAM Structures on M1

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



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

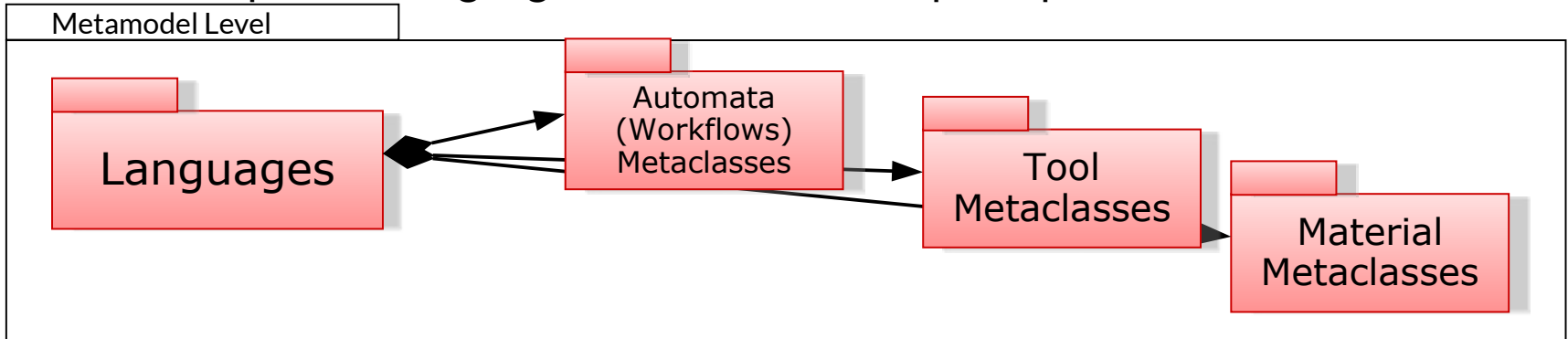
- ▶ On M1, application class models need to define (stereotype) tools, automata, and materials.



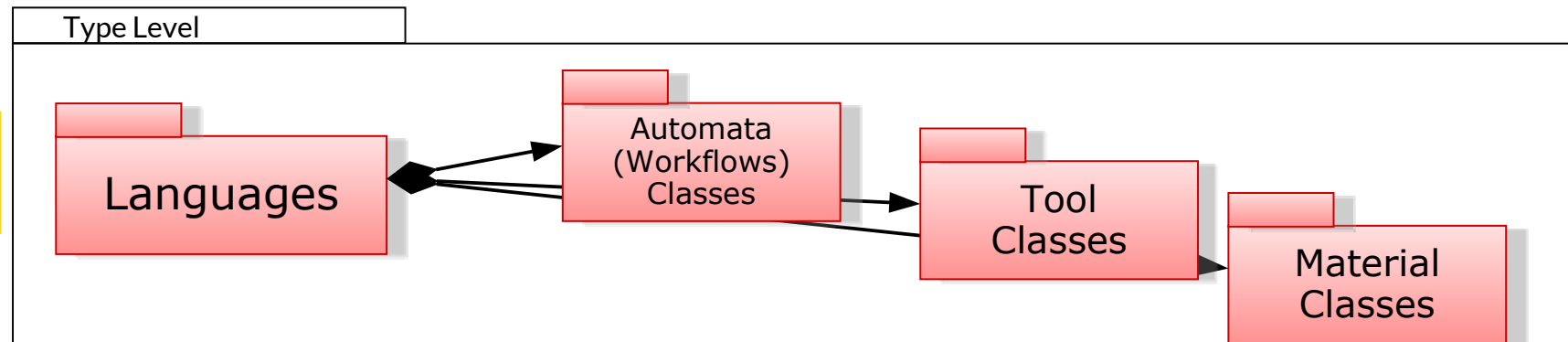
TAM Structures on M2 Provide Language Concepts for Stereotypes for Classes in M1

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

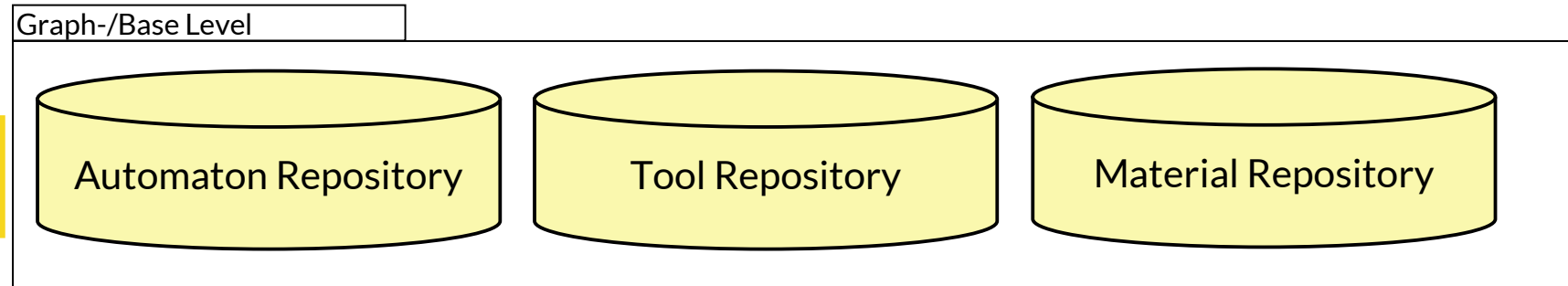
M2



M1



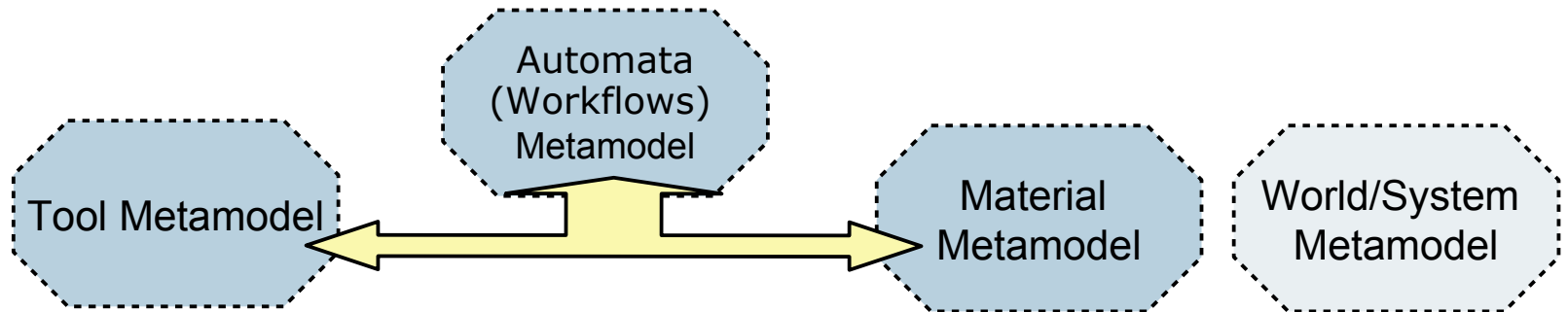
M0



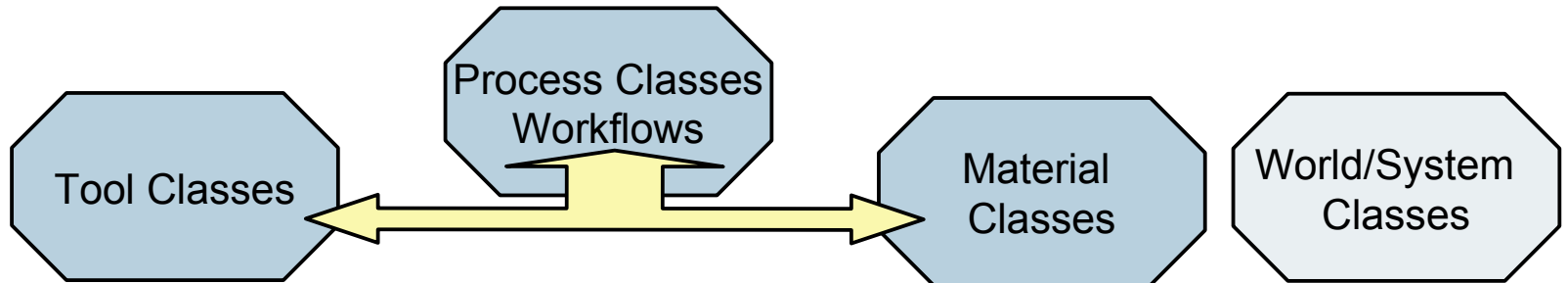
TAM in the Metapyramid

- ▶ TAM is a **pattern language** on M2 (metaclass role model)
- ▶ Thereby, TAM structures M2, M1, M0

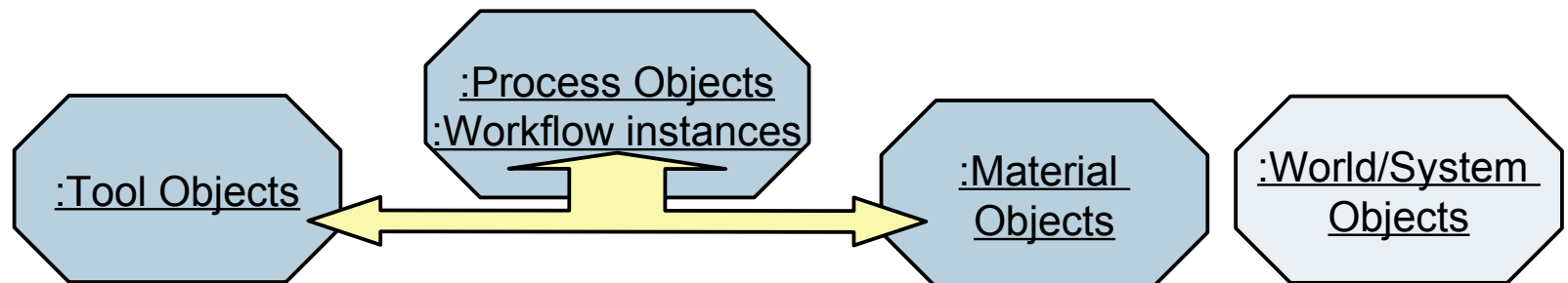
M2



M1



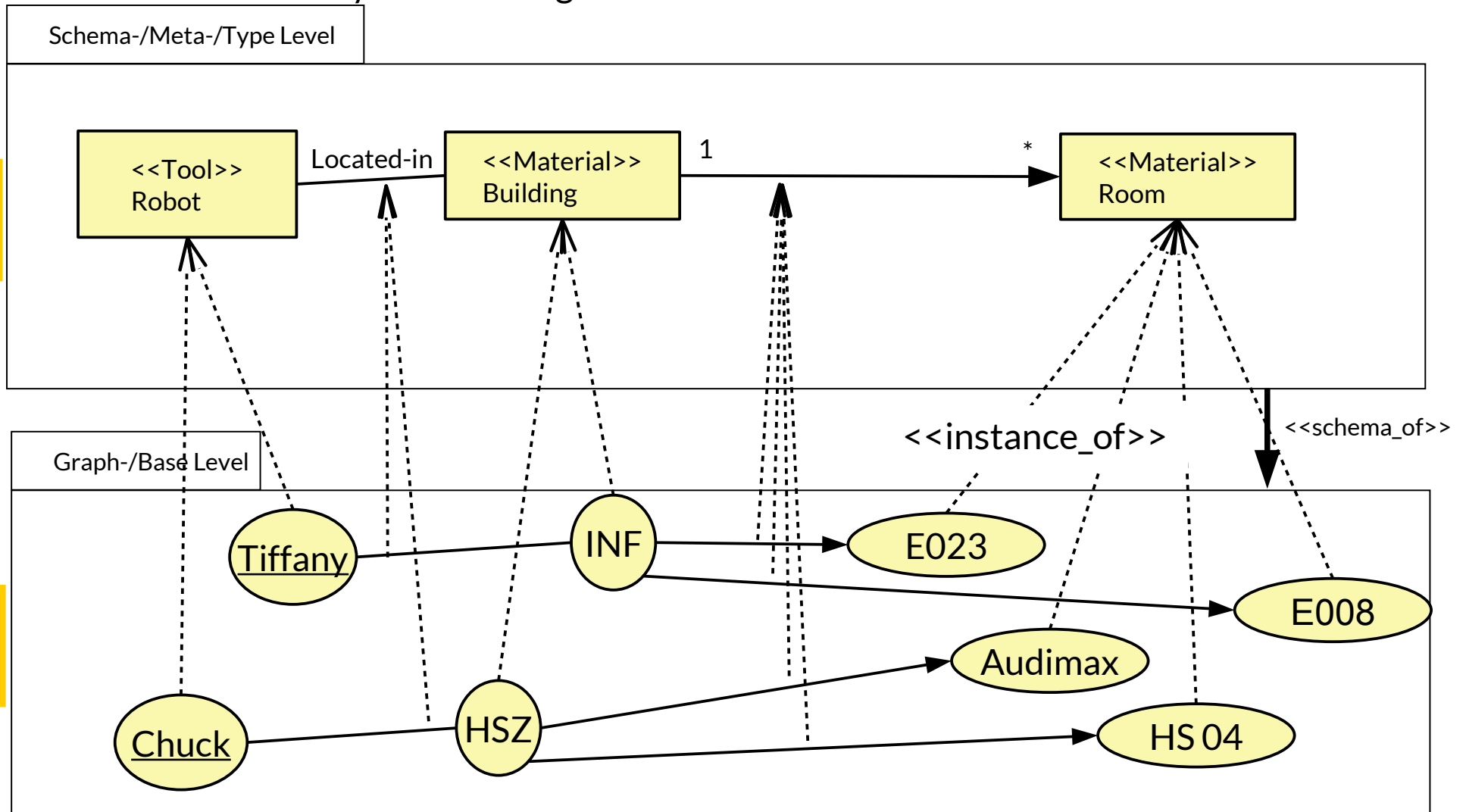
M0



Type Modeling for Application Types (with TAM Stereotypes for Classes)

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

M1

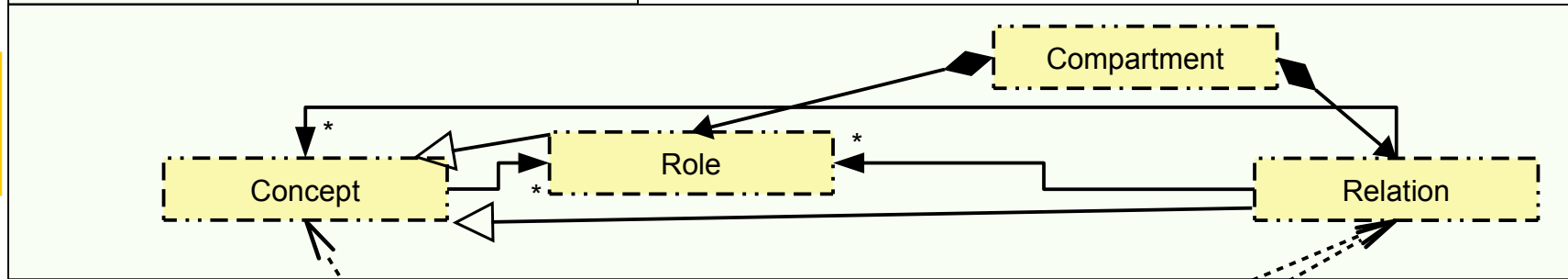


M0

Objects, their Clabjects in Models and Metamodels and TAM

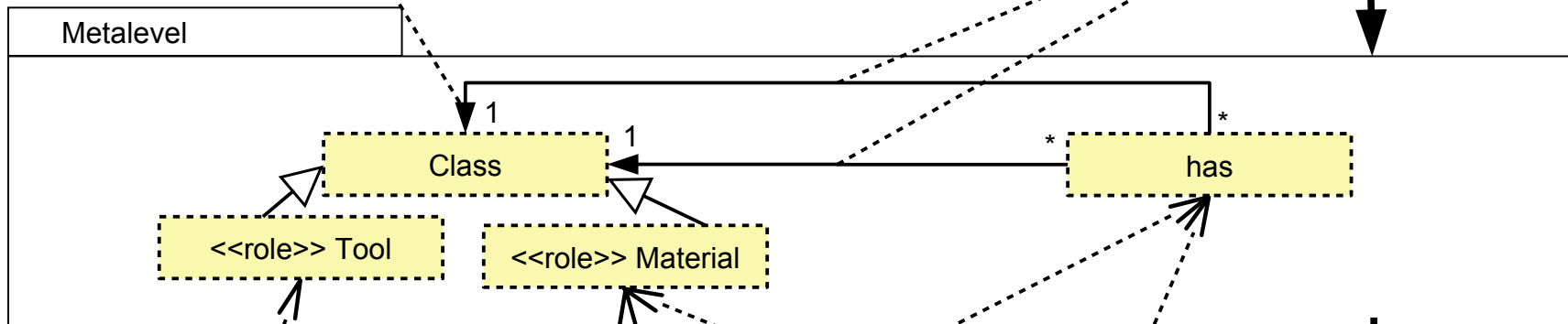
Metameta-Level with CROM-like language

M3



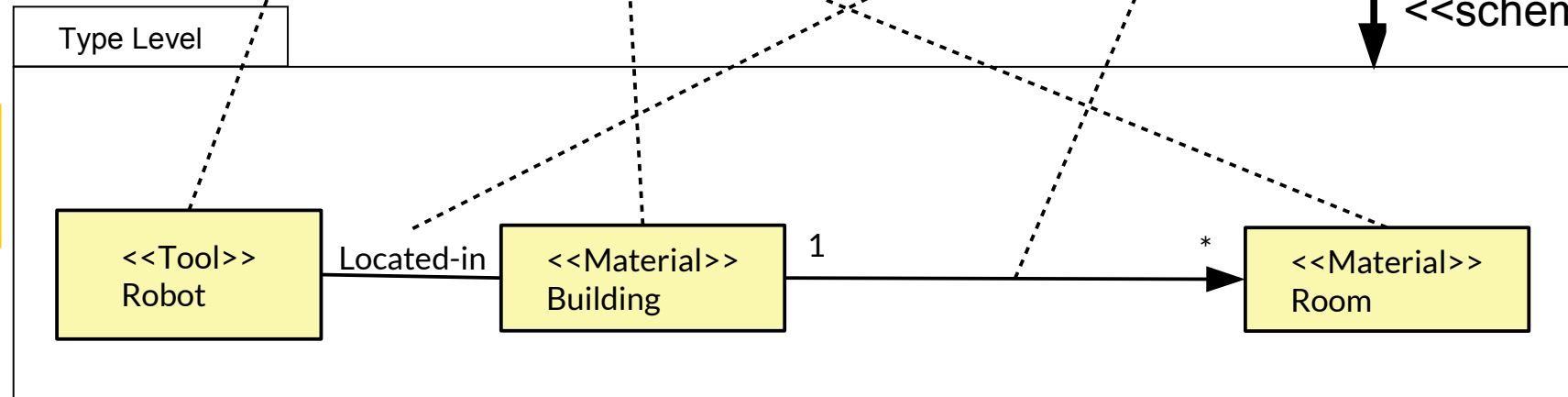
<<schema_of>>

M2



<<schema_of>>

M1



Law of Pattern Languages for M2

A pattern language on M2 requires the Role concept on M3.

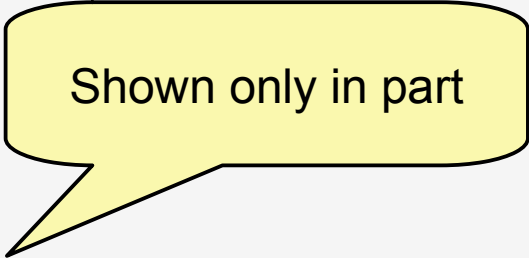
Different pattern languages on M2 requires the Compartment concept on M3.
Compartments structure their metaclass role models on M2.

CROM is a good metalanguage for technical spaces with pattern languages on M2.



13.3 Identification of Tools, Materials for Layering of Applications

Representing TAM as stereotype profile in UML diagrams – marking up special kinds of tools, workflows, materials

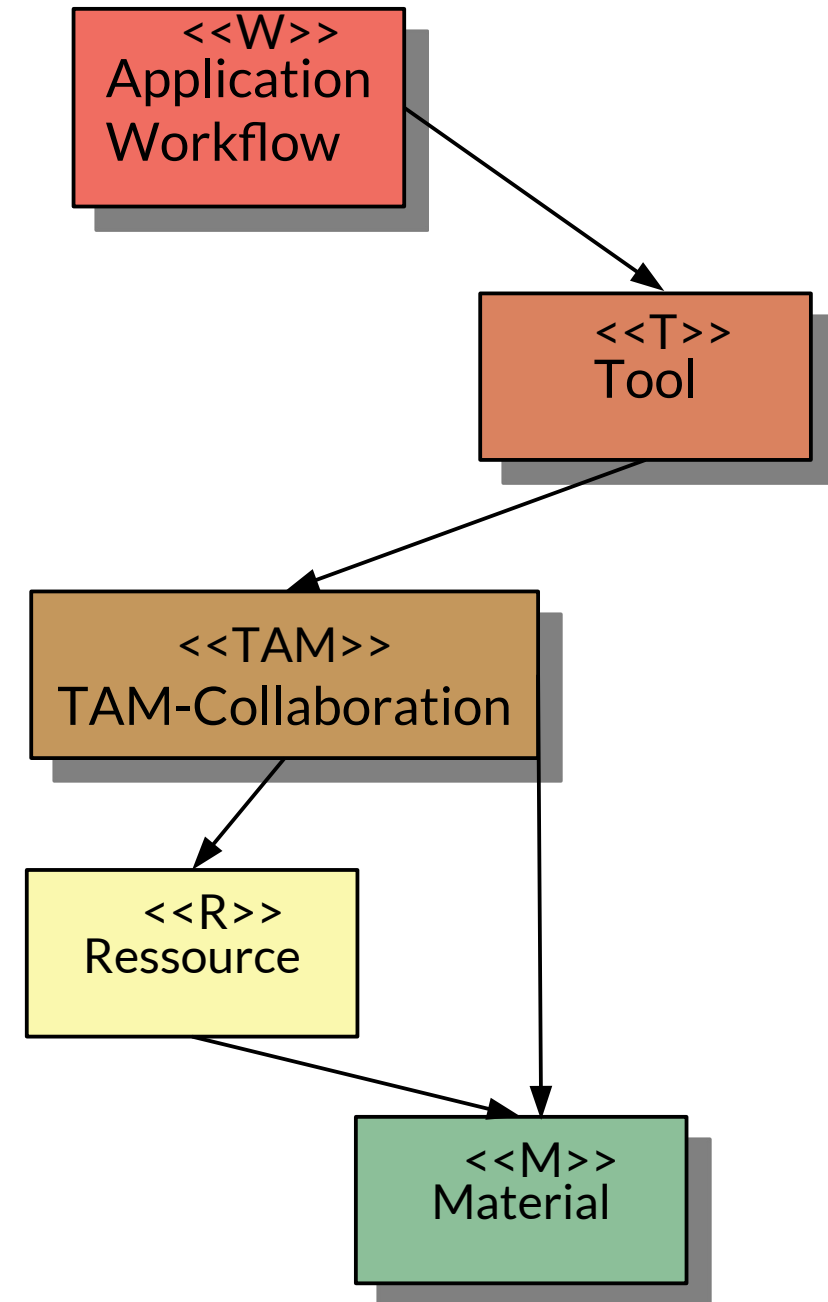


Shown only in part

Perspektive Model TAM: Separation of active and passive Components

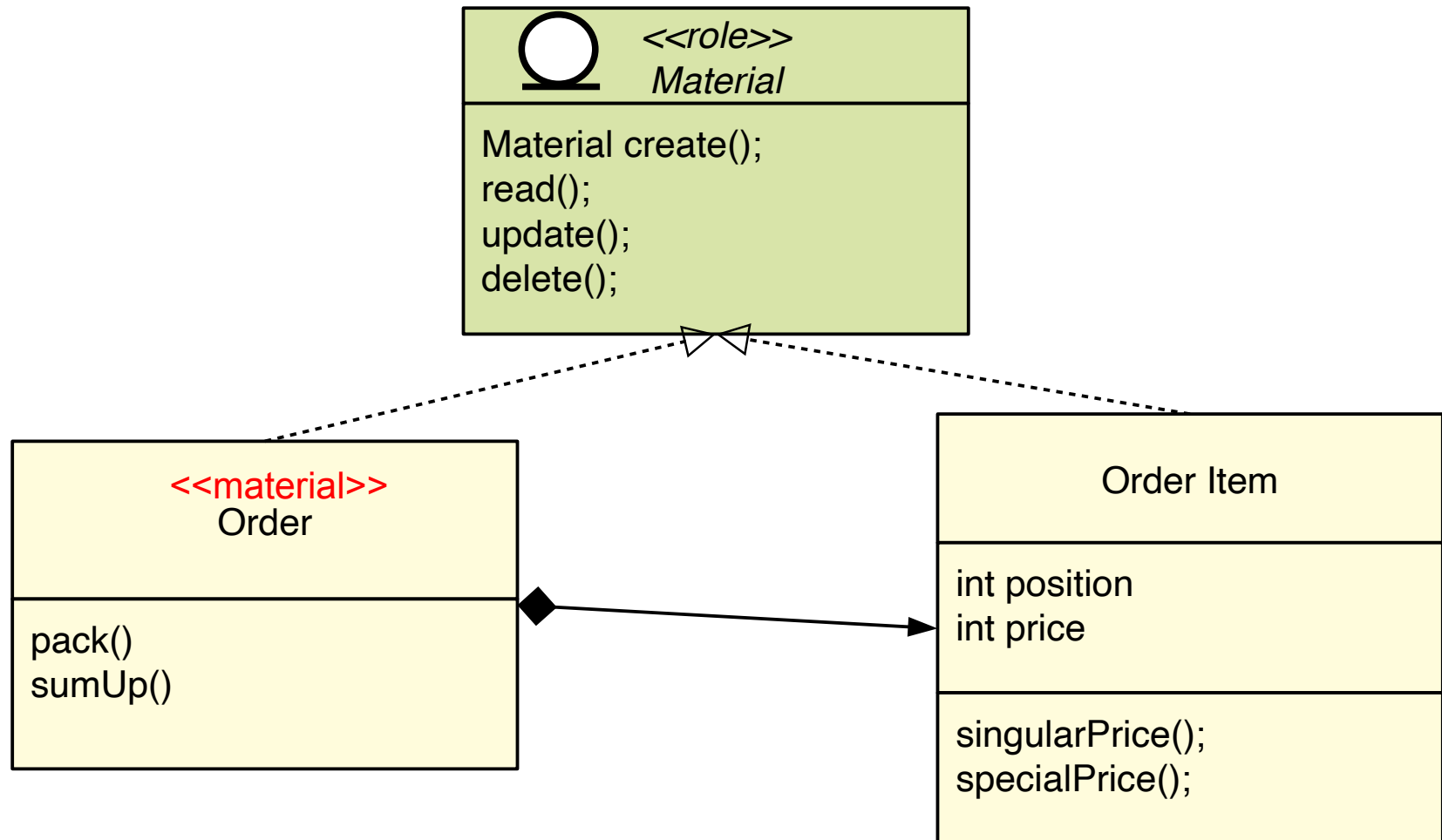
Tools-and-Materials [Züllighoven] is a perspektive model with the following aspects:

- 1) Tools (active processes)
 - 2) Ressourcen (allocatable)
 - 3) Materials (passive data)
 - 4) TAM-Collaboration
 - 5) Workflows (Automata) coordinate Tools
- All program units, such as classes, modules, components, packages can be attributed with these aspects **as stereotypes**

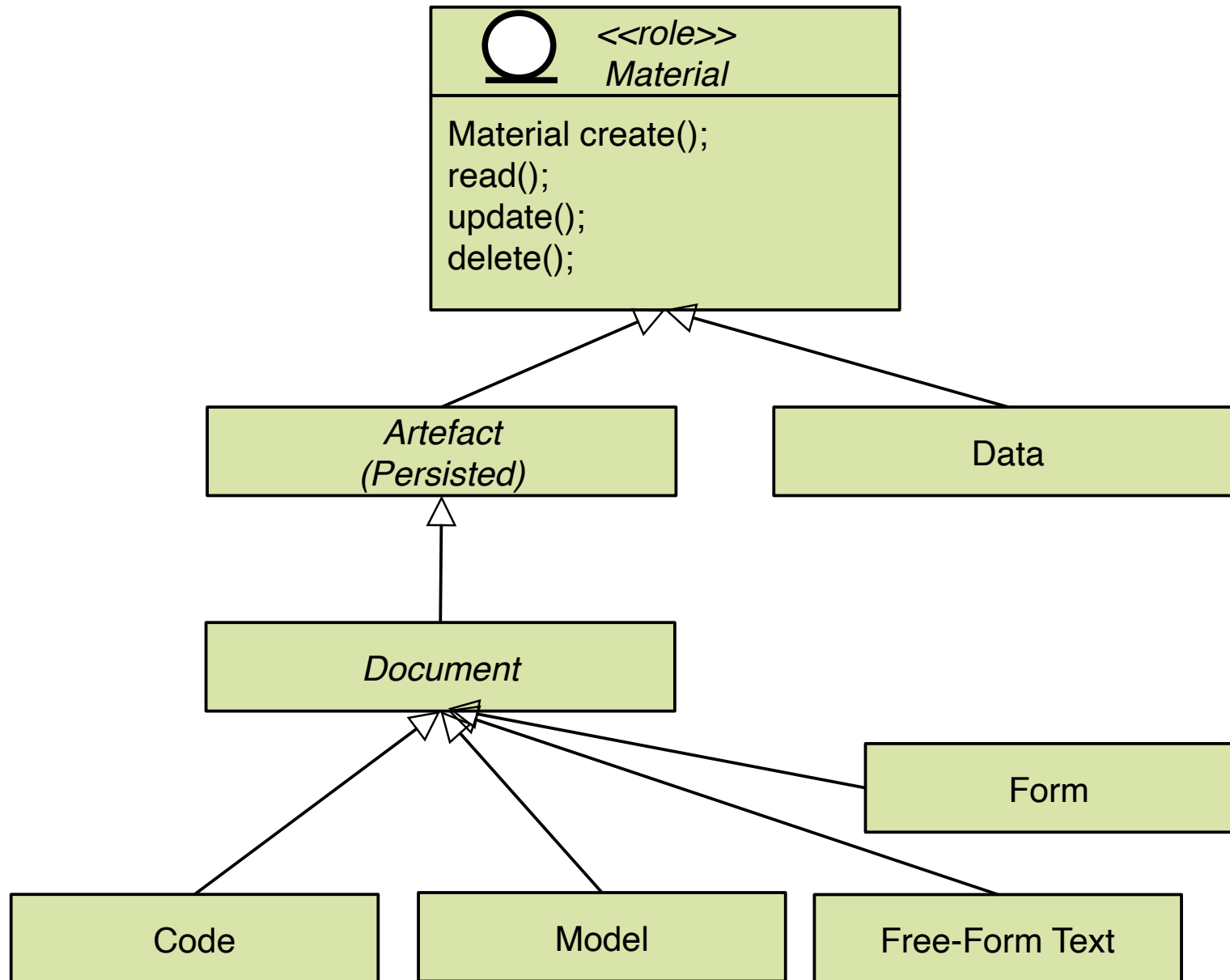


Material-Classes and Interfaces

- ▶ Material objects (M0) are passive, e.g., are called from outside
- ▶ Material objects can be composite (Pattern Composite or Bureacracy)
- ▶ Materials have a CRUD-interface

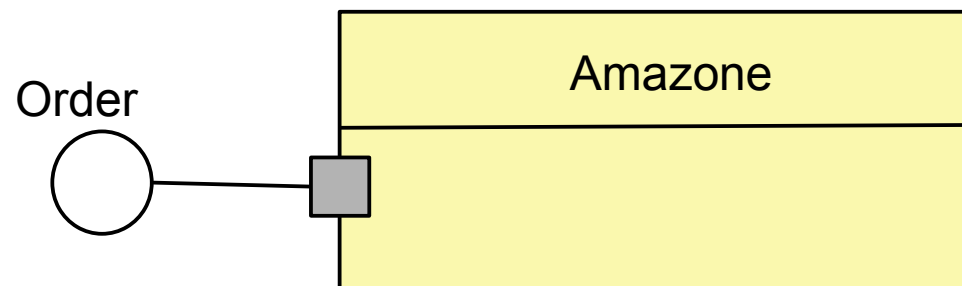


The Material Hierarchy



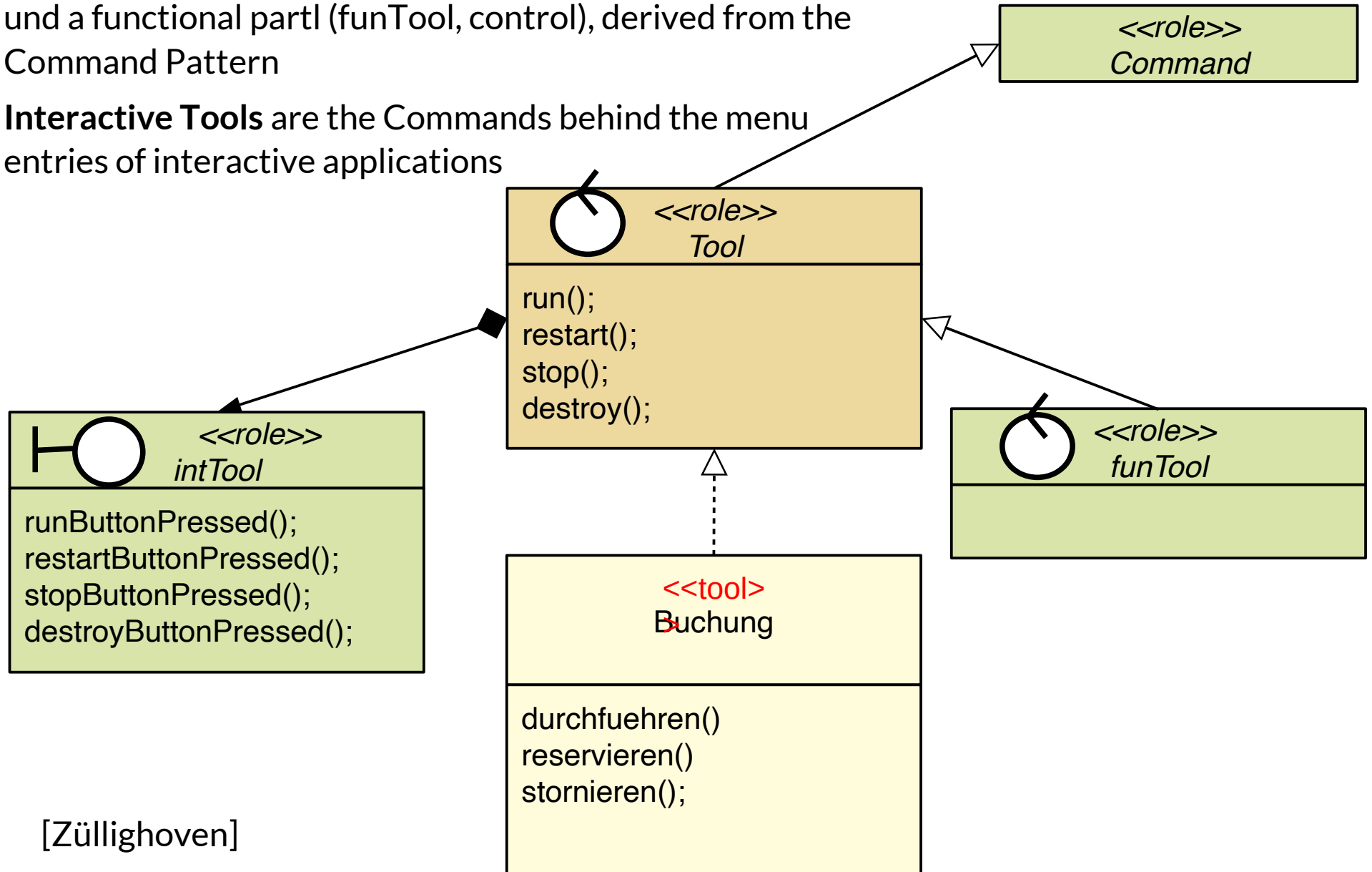
Material-Classes and Interfaces

- ▶ Material Classes can appear as interfaces in Ports of UML-components



Tool-Classes and Interfaces

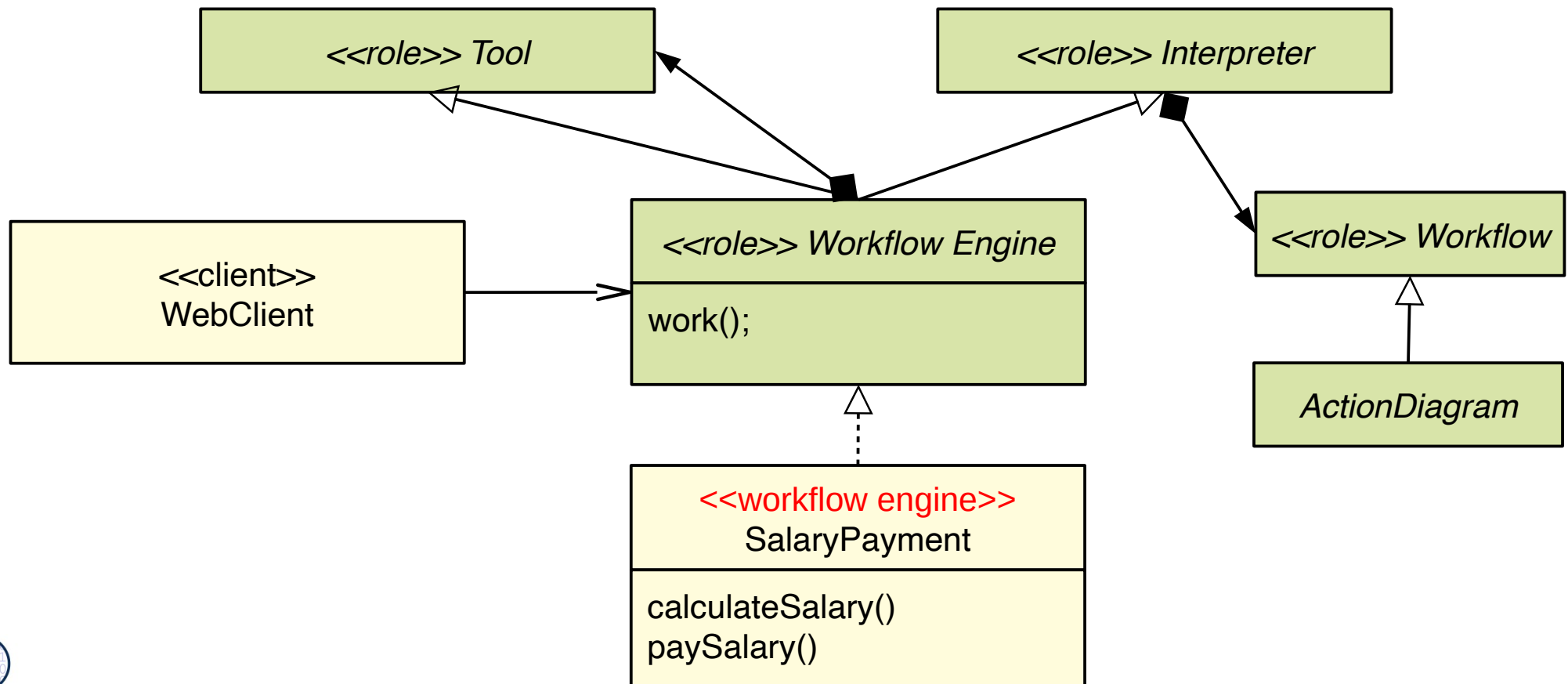
- ▶ Tool-objects have an interactive Teil (intTool, boundary) und a functional part (funTool, control), derived from the Command Pattern
- ▶ **Interactive Tools** are the Commands behind the menu entries of interactive applications



[Züllighoven]

Workflow-Engine-Classes and Interfaces

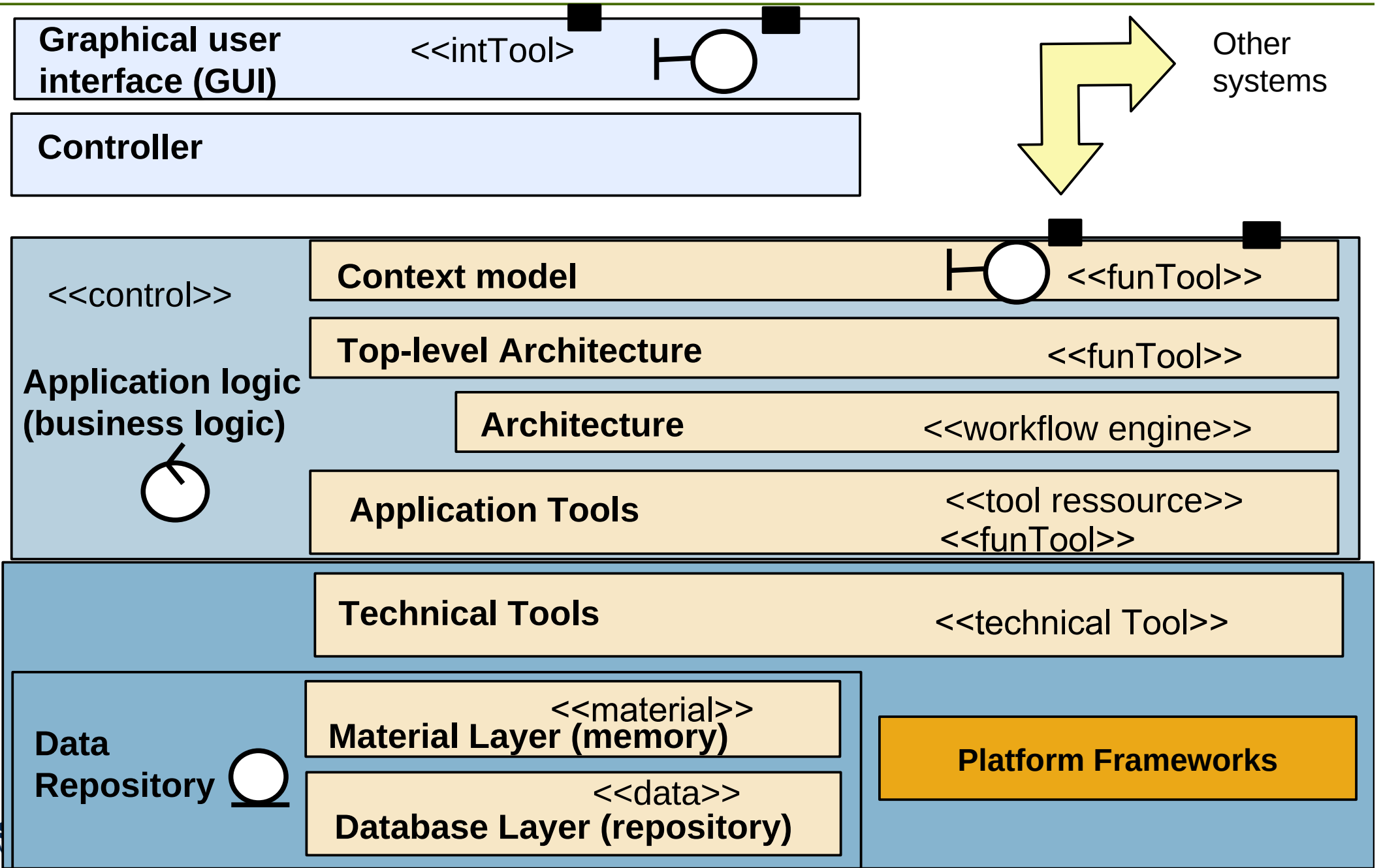
- ▶ **Workflow-Engines** are special tools, automata objects organizing a workflow.
 - Workflow-engines interpret the workflow
- ▶ Workflow-Engines call other tools
- ▶ Their workflows are specified by a behavioral language (action diagrams, statechart, BPMN)



M0 Layers and TAM-Classification

- ▶ Die TAM-classification enables to position objects in the layer cake of the application (M0 layer cake)

Q3: M0-Layer Cake





13.4 Basic Functions of Standalone Tools, their internal Tools and Materials

- „Tools and Materials“ mark *objects*
- The pattern is also applied to Standalone Tools in an Integrated Development Environment (IDE)

Standalone Tools and Artefacts

Def.: A **Standalone Tool** is a *persistent* tool object wrapped with externalization of its input and output data

Def.: An **artefact** is a persisted material.

Artefacts can be documents, models, or code.

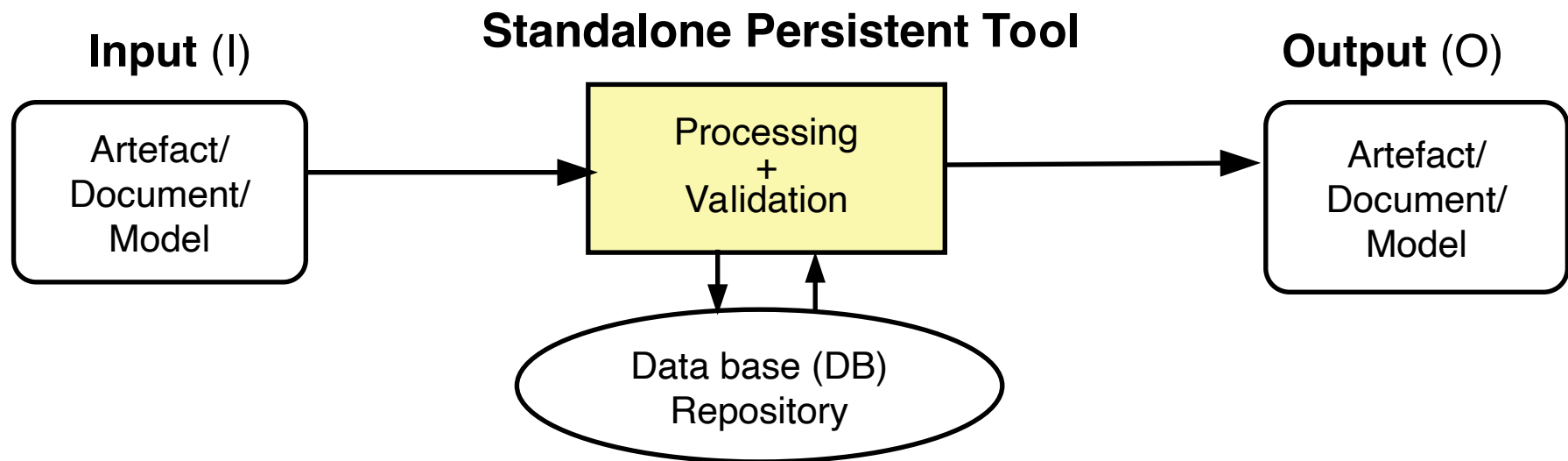
Standalone Tools on Different Kinds of Materials (Artefacts):

- ▶ **Code-centered standalone tools:**
 - **Software tools** are programs with documentation and test architecture
- ▶ **Document-centered standalone tools**
 - Document tools are needed for software development
- ▶ **Model-centered standalone tools (Modeling Tools)**
 - Basic components for MDSD IDE

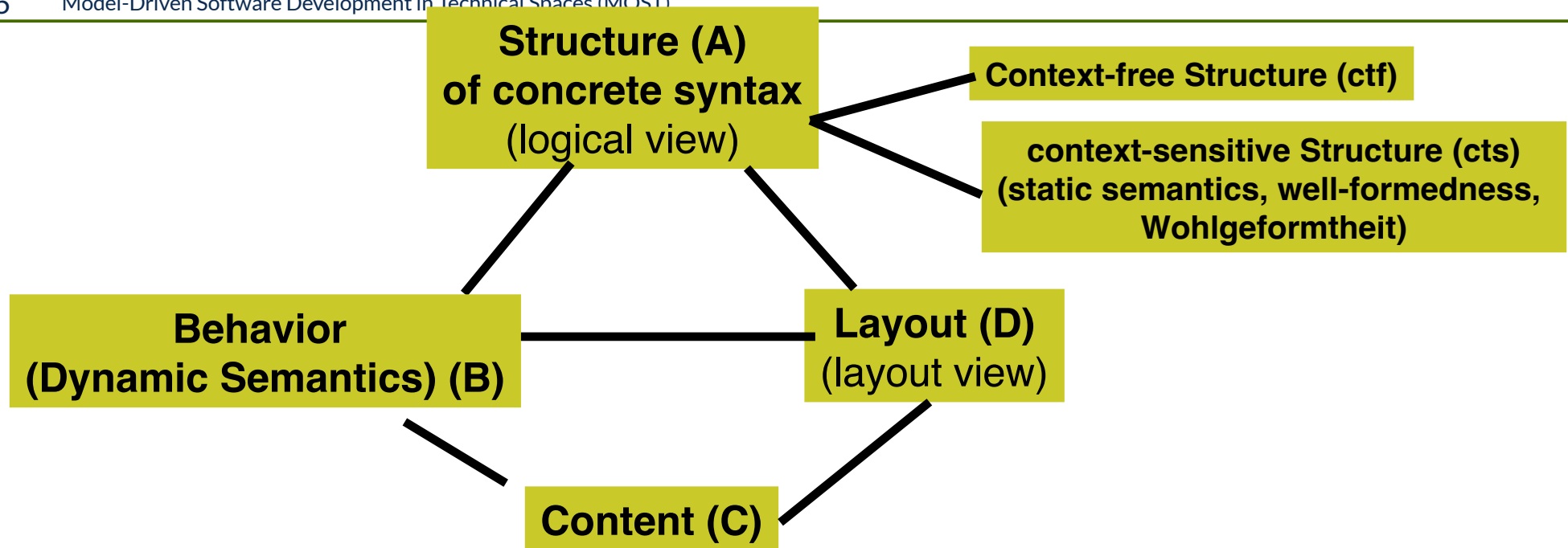
Standalone Tools are Deterministic Functions

- ▶ Standalone Tools analyze an input and produce an artefact as output
- ▶ Standalone Tools transform an input to an output

$$\text{sttool: } I \times \text{DB} \rightarrow \text{DB} \times O$$



Aspects of Materials and Artefacts (Persisted Documents, Models, Code)



- ▶ **Structure of concrete syntax:** log. Units of the model, conform to a metamodel
 - Context-free: Hierarchic structure
 - Links: cross links, references
 - Context-sensitive structure mit consistency conditions for well-formedness (static semantics)
- ▶ **Content:** Text, grafics, images, videos
- ▶ **Layout:** Placement of content
- ▶ **Dynamic Semantics:** Programs have a meaning (behavior)

Well-Formedness of Artefacts and Materials (Models, Documents, Code)

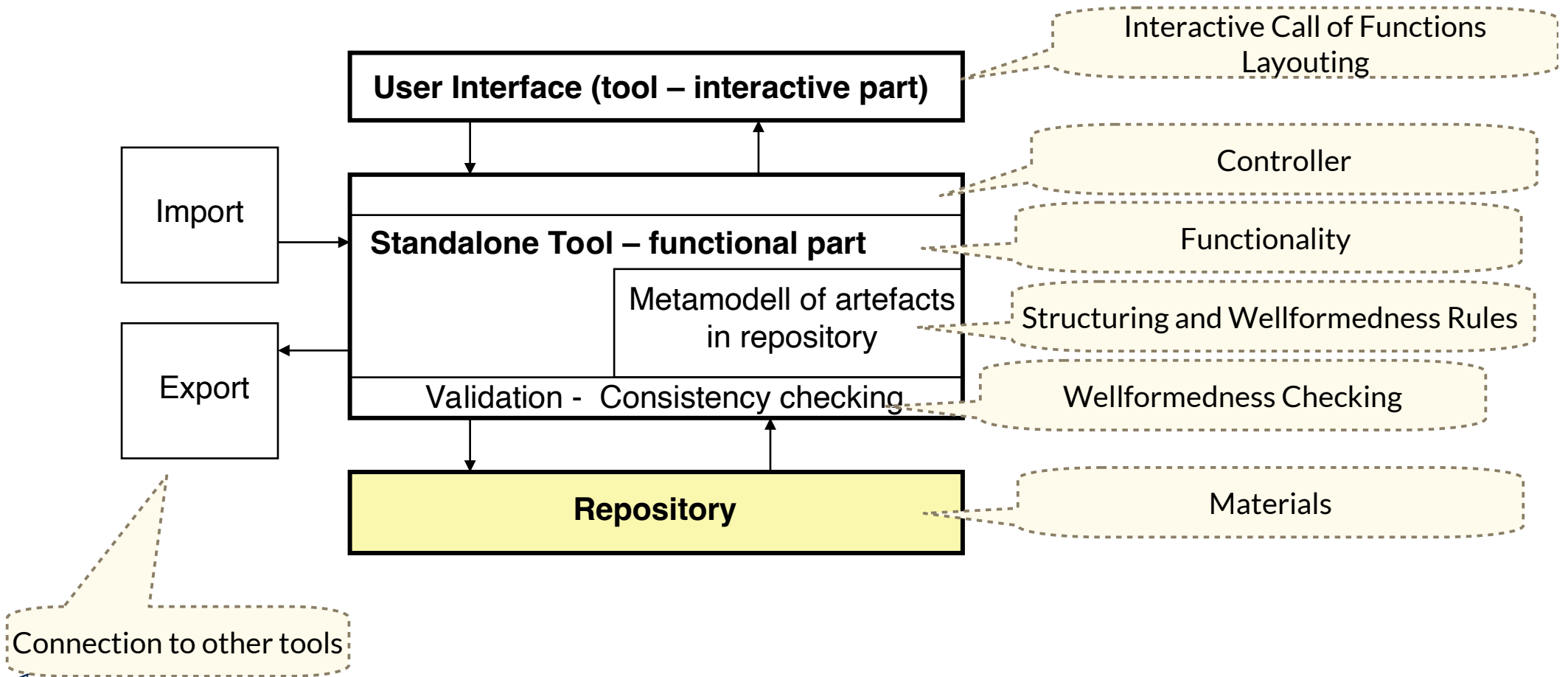
An artefact is **well-formed (consistent)**, if it fulfils context-sensitive constraints (integrity rules, consistency rules) of its metamodel.

Wellformedness is checked by **static semantic analysis (context analysis)**:

- ▶ **Name analysis (Namensanalyse)** finds the meaning of a name
 - **Type analysis (Typanalyse)** finds the meaning of a type
 - **Type checking** checks the use of types with their definition
 - **Links** are not pointing into Nirwana
- ▶ **Invariant checks (Invariantenprüfung)**
 - **Range checks** (Bereichsprüfungen) test the validity of variables in ranges
 - **Structuring** of data structures: Acyclicity, layering, connected components, reducibility
- ▶ **Forbidden combinations** of constructs
- ▶ **Replicated definitions** of variables and objects

- ▶ Documents:
 - Free text
 - Word documents, requirement specifications, user stories, comments
- ▶ Models
 - Textual models
 - Forms, Templates, Canvases
 - Trees and ordered trees (terms)
 - S-Expressions (Lisp, Scheme)
 - Link trees (XML-trees, JSON-trees), Feature terms
 - Ontologies
 - Diagrammatic models, usually specific graphs
 - Analysis documents and design specifications (UML-diagrams), Petri-Nets, statecharts
 - Tables: Relations, test case tables
- ▶ Graphics: Visualizations in 2-D or 3-D
- ▶ Code: e.g., Pseudocode, code templates, source code

Q4: Logic View of Standalone Tool Architecture

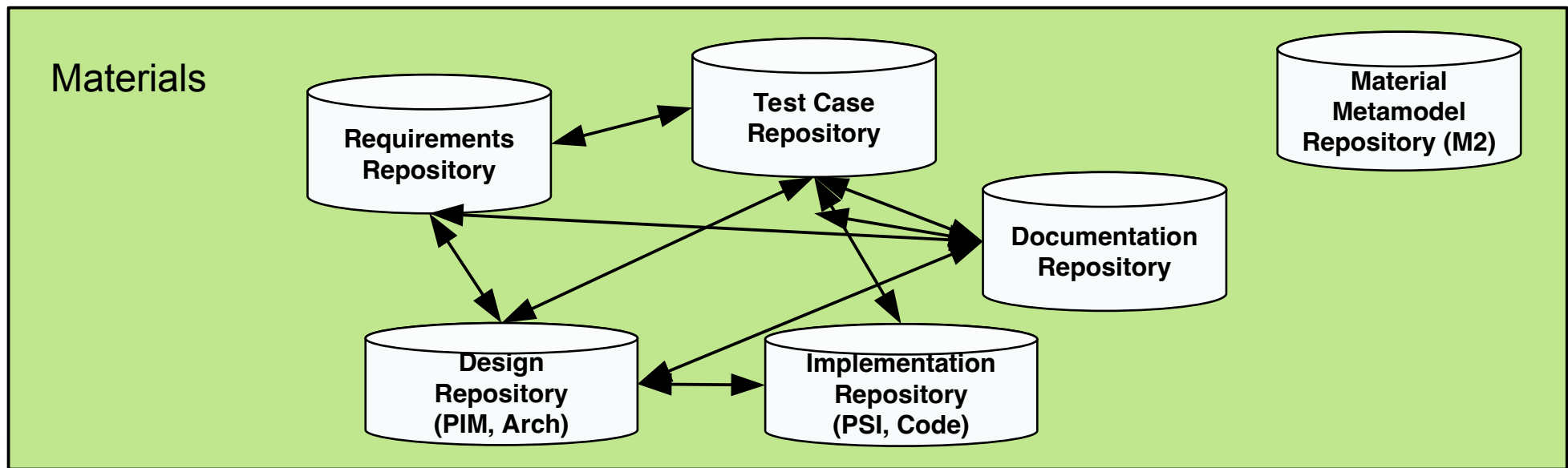
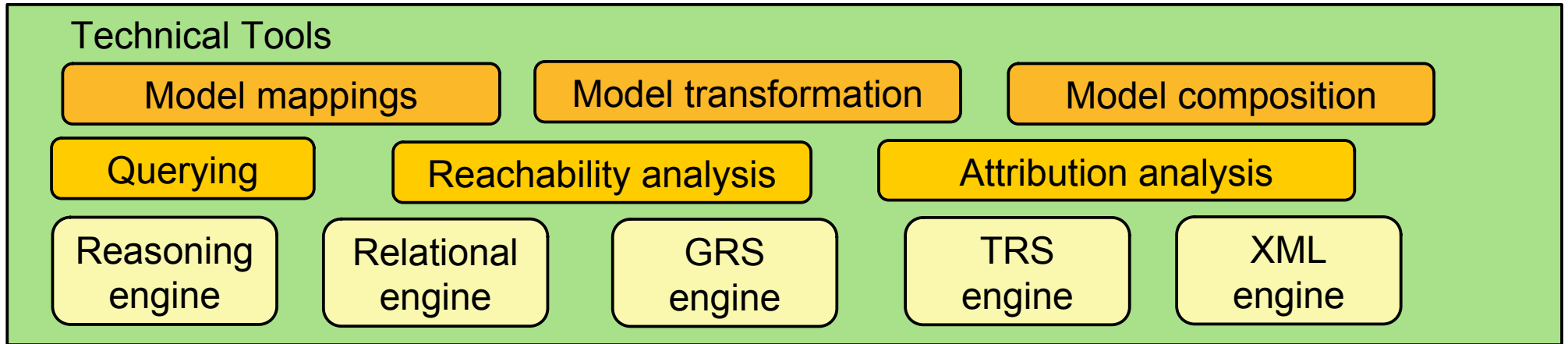
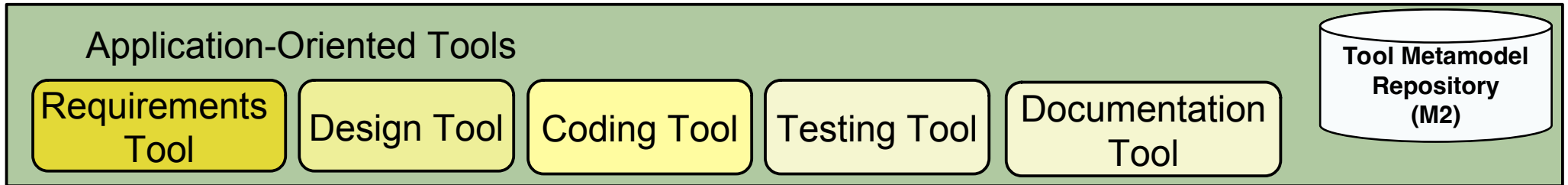


MDSD Applications

An Model-driven application consists of a structured set of integrated Standalone Tools working on a integrated set of artefacts, possibly in a world model.

- ▶ An MDSD application is also structured with TAM, but uses heterogeneous models and persisted artefacts.

Q2: Tool-Objects and Materials in an Integrated Development Environment (IDE, SEU) for MDSD



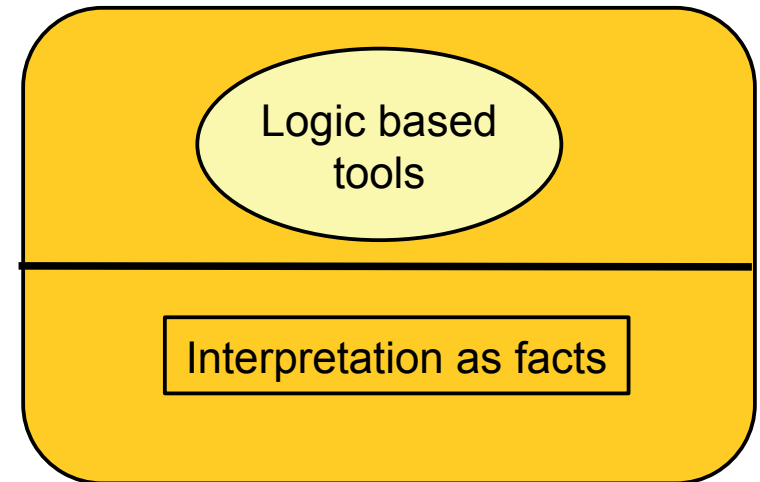
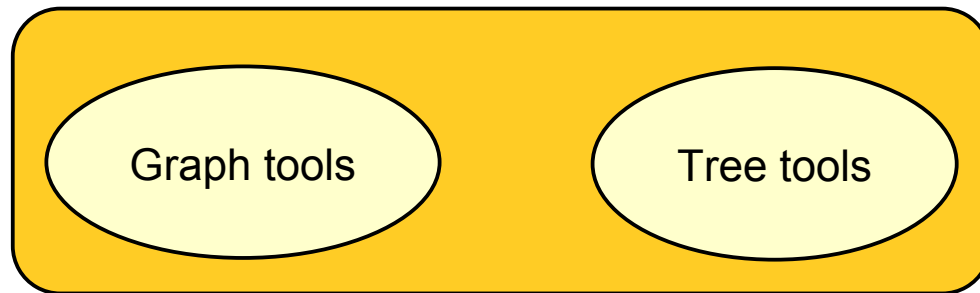


13.3.2 The Graph-Fact-Isomorphism for Materials

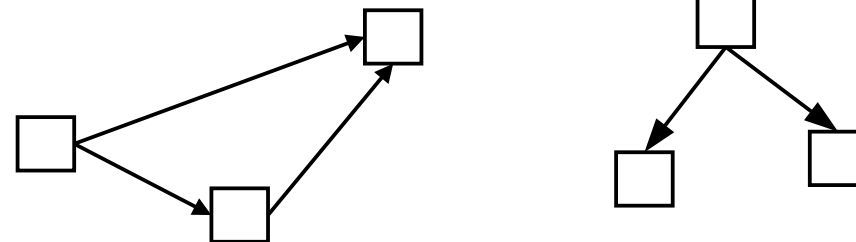
The Graph-Fact-Isomorphism

- ▶ Every graphbase can be represented as a fact base of a logic inference engine (reasoner)
- ▶ Every fact base (with material) can be interpreted as graph base
 - binary: Graph
 - n-ary: Hypergraph
- ▶ Therefore, logic inferencers and graph transformation tools can be used on the same data and artefacts
- ▶ Materials can be seen as facts of a reasoner or graphs of a modeling environment
- ▶ *Metamodeling* uses both kinds of technologies

IDE with Logic-based and Graph-based Tools



Trees and graphs in memory



Persistent trees and graphs (artefacts)

The End

- ▶ Explain the consequences of the Züllighoven principle for the construction of heterogeneous applications
- ▶ Why does the TAM pattern language cross the metapyramid?
- ▶ Which concepts belong to a process metamodel in contrast to a tool or material metamodel?
- ▶ Why is static semantics divided into context-free structure and context-sensitive wellformedness conditions?
- ▶ Why is it possible to store a model in a database or an inferencer?