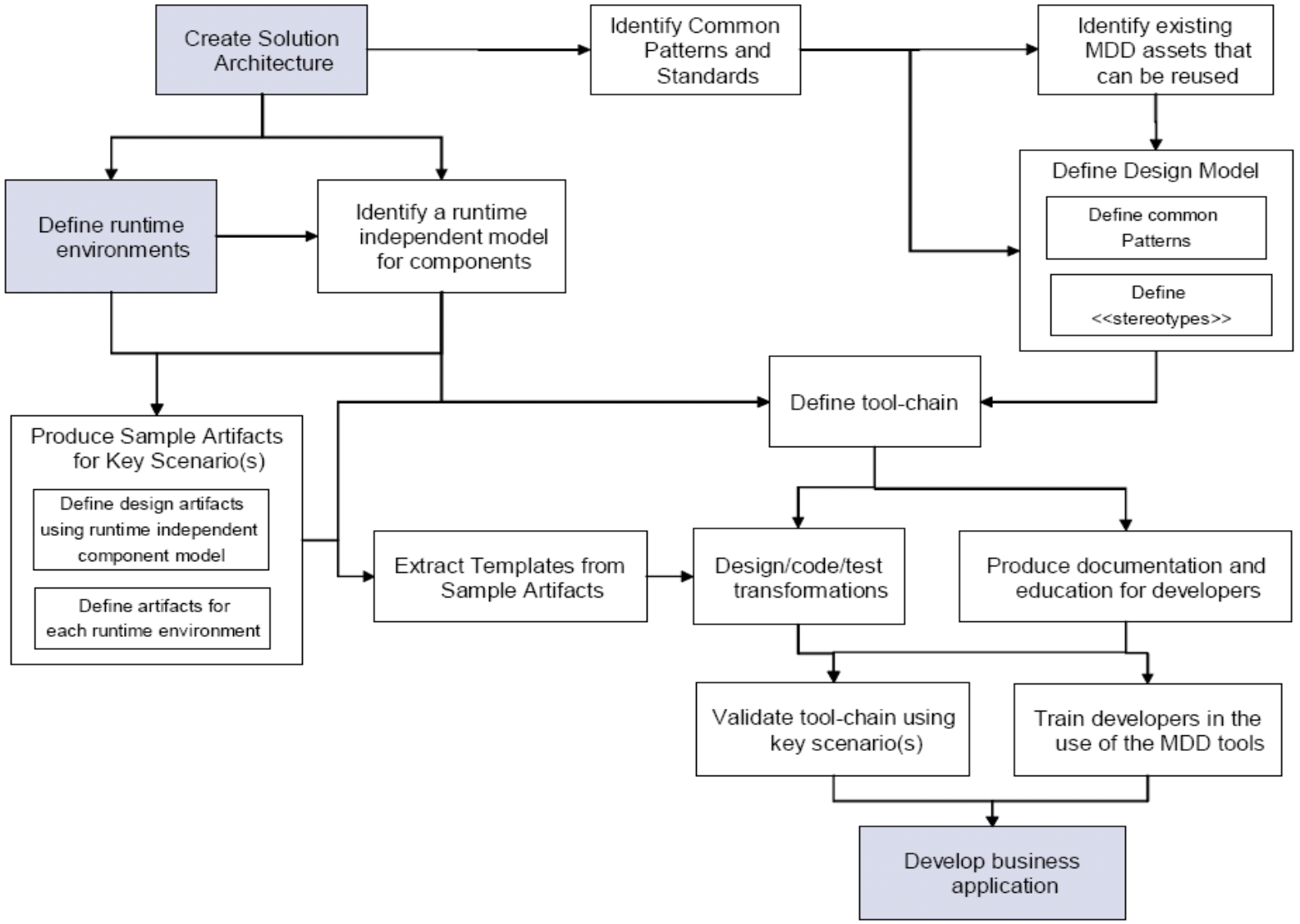
MDSD: Process

- **Changed development process**
  - Two stages of development – infrastructure and application
    - Setting up/developing infrastructure: modelling languages, platform (e.g., frameworks), model transformations, …
    - Application development: modelling, efficient reuse of infrastructure, less coding
  - Simplified application development
    - Automated code generation makes implementation tasks obsolete.
    - Tasks on code level (implementation, test, maintenance, etc.) are drastically reduced.

- **New development tools**
  - Tools for language definition, especially meta-modelling
  - Editors and transformations engines
  - Customizable tools and suites: Model editors, repositories, tools for simulation, verification, and test, etc.

Set-up of MDSD project and tooling



MDSD approaches: A short overview

- **Approaches**
  - Computer-Aided Software Engineering (CASE)
  - Executable UML
  - Model-Driven Architecture (MDA)
  - Architecture-Centric Model Driven Software Development (AC-MDSD)
  - MetaCASE
  - Software Factories

Computer-Aided Software Engineering (CASE)

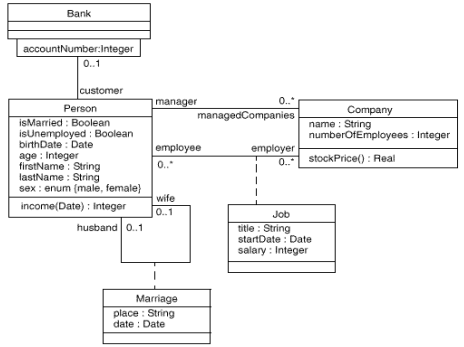
- Historical approach (end of 20th century)
  - Example: Computer Associates' AllFusion Gen
    - Support Information Engineering Method of James Martin through different diagrams types
    - Fully automatic code-generation for 3-tier architecture and some execution platforms (Mainframe, Unix, .NET, J2EE, various databases, …)
    - Advantage/disadvantage: changes to target platform not necessary/possible
- Differences to the basic architecture of MDSD
  - Meta-level description not supported or accessible to modeller
  - General-purpose graphical language representations with tool specific variants
  - Modelling languages mapped poorly onto the underlying platforms
  - No or fixed description of execution platform
- Advantages
  - Productivity, development and maintenance costs, quality, documentation
- Disadvantages
  - Proprietary modelling languages
  - Tools not interoperable and rather complex
  - Support of platforms and new features strongly depends on tool vendors
  - No standardization, no (real) abstraction levels, and DSLs
  - Limited to programs written by a single person or by a team that serializes its access to files

Executable UML

- “CASE with UML”
  - Subset of UML: class diagrams, state charts, component diagrams
  - UML Action Semantic Language (ASL) as programming language
- Niche products
  - Some specialized tool vendors like Kennedy/Carter
  - Used e.g. for developing embedded systems
- Realizes parts of the MDSD basic architecture
  - There is one predefined modelling language (xUML)
  - Transformation definitions can be changed and adapted (with ASL)
- Advantages compared to CASE
  - Standardized modelling language based on UML
- Disadvantages compared to CASE
  - Modelling language has less modelling elements

Model-Driven Architecture (MDA)

- MDA is a standard promoted by the OMG
  - A *set of specifications* defined by OMG's open, worldwide process
  - MDA looks at software development from the point of view of models
- *Models* are the core; design is the focus
  - MDA supports technology-independent design
  - MDA divides domain knowledge and platform knowledge
- Advantages
  - Portability to different platforms and technologies
  - Re-usability
  - Open Source
- Disadvantage
  - General-purpose approach, sometimes specific solutions perform better



Architecture-Centric Model Driven Software Development

- Efficient reuse of architecture
  - Focus on efficient reuse of infrastructure/frameworks (= architecture) for multiple applications
  - Concrete methodology
    - Development of reference architectures
    - Analysis of code that is individual, has schematic repetitions, or is generic
    - Extraction of necessary modelling concepts and definition of modelling language, transformations, and platform
    - Tool support (e.g. www.openarchitectureware.org)
- Advantages to MDA
  - Supports development of individual platforms and modelling languages
- Disadvantages to MDA
  - Little support for portability

MetaCASE/MetaEdit+

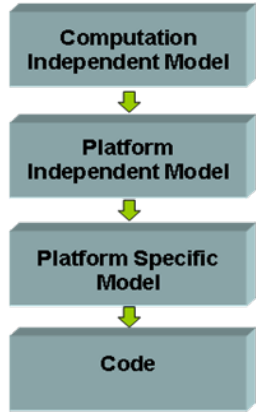
- Individual configurable CASE
  - Metamodeling for developing domain-specific languages (DSLs)
  - Focuses on best support of application domain (intentional programming for e.g. cell phone software)
  - Methodology defined through DSL development
- Good (meta-)modelling support
  - Good meta-modelling support, incl. graphical editors
  - No separated support for platform development, but suggests to use components and frameworks
- Advantage
  - Domain-specific modelling
- Disadvantages
  - Tool support focused on graphical modelling
  - No tool interoperability, since proprietary M3-level (meta-meta-model)

Software Factories

- (Industrial) manufacturing of software products
  - Combines ideas of different approaches (e.g. MDA, AC-MDSD, MetaCASE/DSLs) as well as common SW-engineering technologies (patterns, components, frameworks)
  - Objective is to support the development of software product lines (SPLs) through automation, i.e. a set of applications with a common application domain and infrastructure
  - "A **software factory** is a **software product line** that **configures extensible tools**, processes, and content […] **automates** the development and maintenance of variants of an archetypical product by **adapting, assembling, and configuring framework-based components**."
- Advantages
  - Focuses on domain-specific solutions
- Disadvantages
  - Little tool support

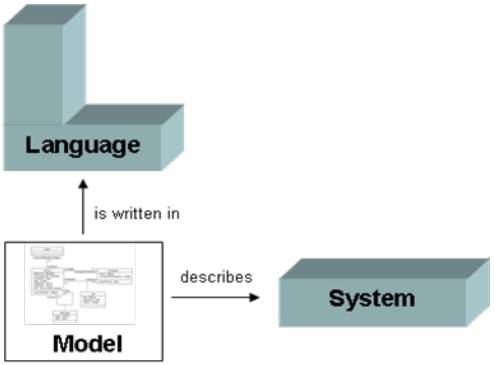
Model-Driven Architecture (MDA): Overview

- *Separates* the operational specification of a system from the details such as how the system uses the platform on which it is developed
- MDA provides the means to
  - *Specify* a system independently of its platform
  - *Specify* platforms
  - Choose a platform for the system
  - *Transform* the system specifications into a platform dependent system
- Three fundamental objectives
  - Portability
  - Interoperability
  - Reuse
  - *Productivity* (derived objective)

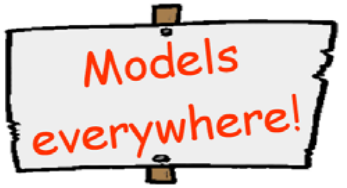


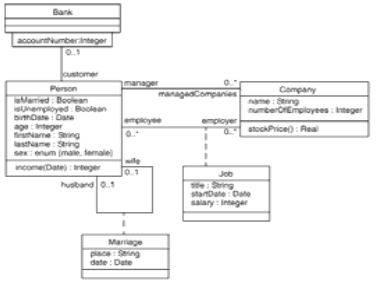
MDA basic elements: Models

- *Cornerstone* of MDA
  - *Abstraction* of reality, different from it, and that can be used for (re)producing such reality
- Expressed in a *well-defined language* (syntax and semantics) which is suitable for automated interpretation



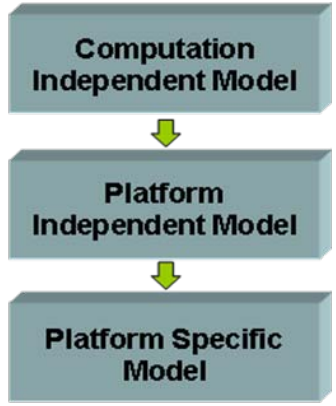
- In MDA, "**everything is a model**"
- One model may describe only part of the complete system
- A model helps
  - Focusing on essentials of a *problem* to better understand it
  - Moving towards an effective *solution*





MDA basic elements: Models

- Types of models
  - **Business models** or Computation Independent Models (**CIM**)
    - Define domains identifying fundamental **business entity types** and the **relationships** between them
    - Say *nothing* about the software systems used within the company
  - **System models**
    - These models are a description of the software system
    - Platform **independent** models (**PIM**)
      - resolves functional requirements through purely problem-space terms.
      - *No platform-specific details* are necessary.
    - Platform **specific** models (**PSM**)
      - It is a *solution model* that resolves both functional and non-functional requirements.
      - A PSM *requires information on specific platform* related concepts and technologies.
    - *Platform independence* is a relative term.

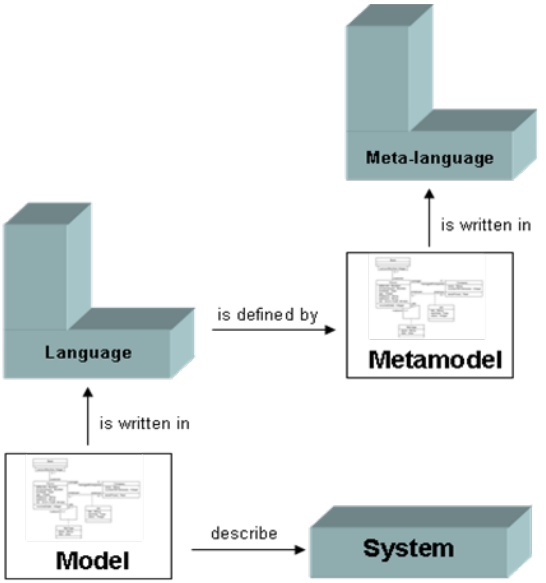


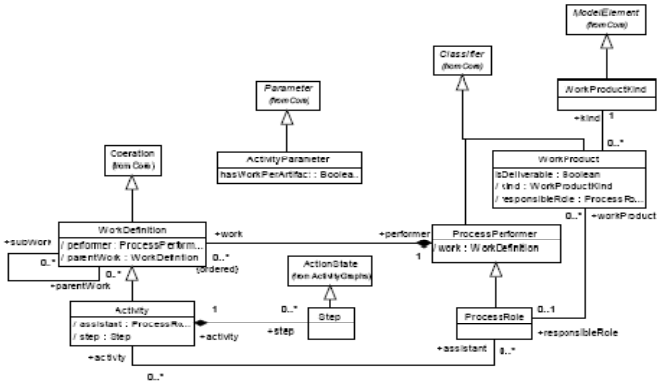
MDA basic elements: Meta-models (1)

- Meta-models allow the exchange of models among modelling tools.

- Meta-models represent specific domain elements.
  - Use of a common terminology
  - Reduce misunderstandings
  - Production of a complete documentation
  - Check of consistent processes
  - Traceability of process artefacts: impact analysis

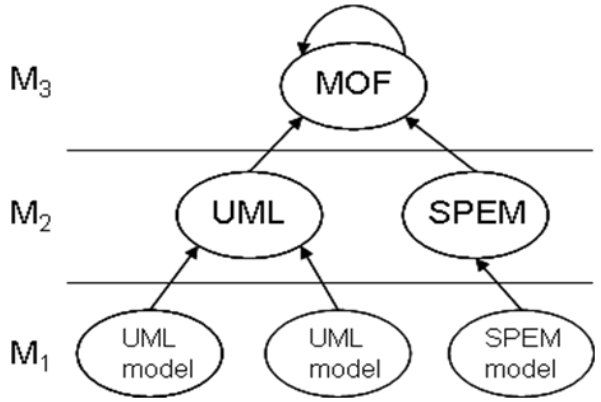
- A **meta-model**
  - is also a *model* and must be written in a well-defined language;
  - defines *structure*, *semantics* and *constraints* for a family of models.



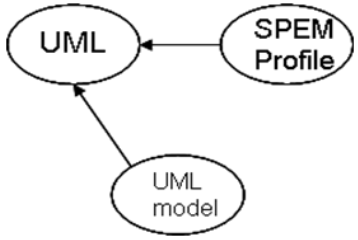


MDA basic elements: Meta-models (2)

- The three-layer architecture
  - (M3) Meta-meta-model
    - One unique meta-meta-model, the *Meta-Object Facility* (**MOF**).
    - It is some kind of “top level ontology”.
  - (M2) Meta-model
    - defines *structure*, *semantics* and *constraints* for a family of models.
  - (M1) Model
    - Each of the models is defined in the language of its *unique meta-model***.**

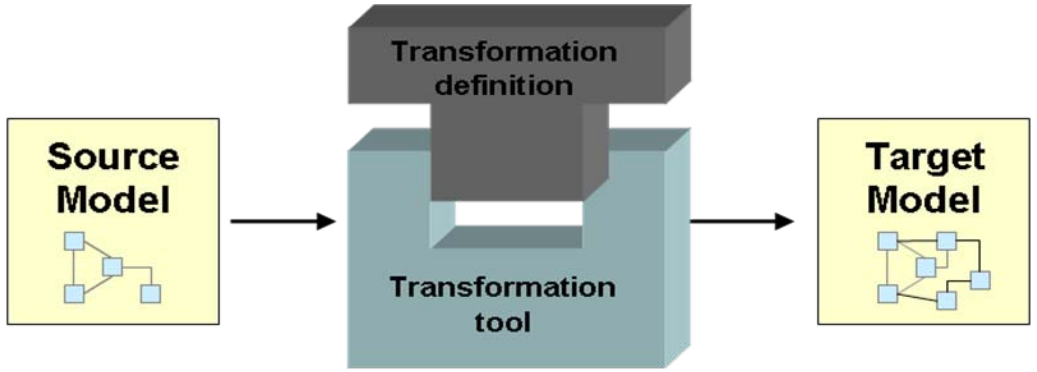


- UML profiles are *adapted modelling languages*.

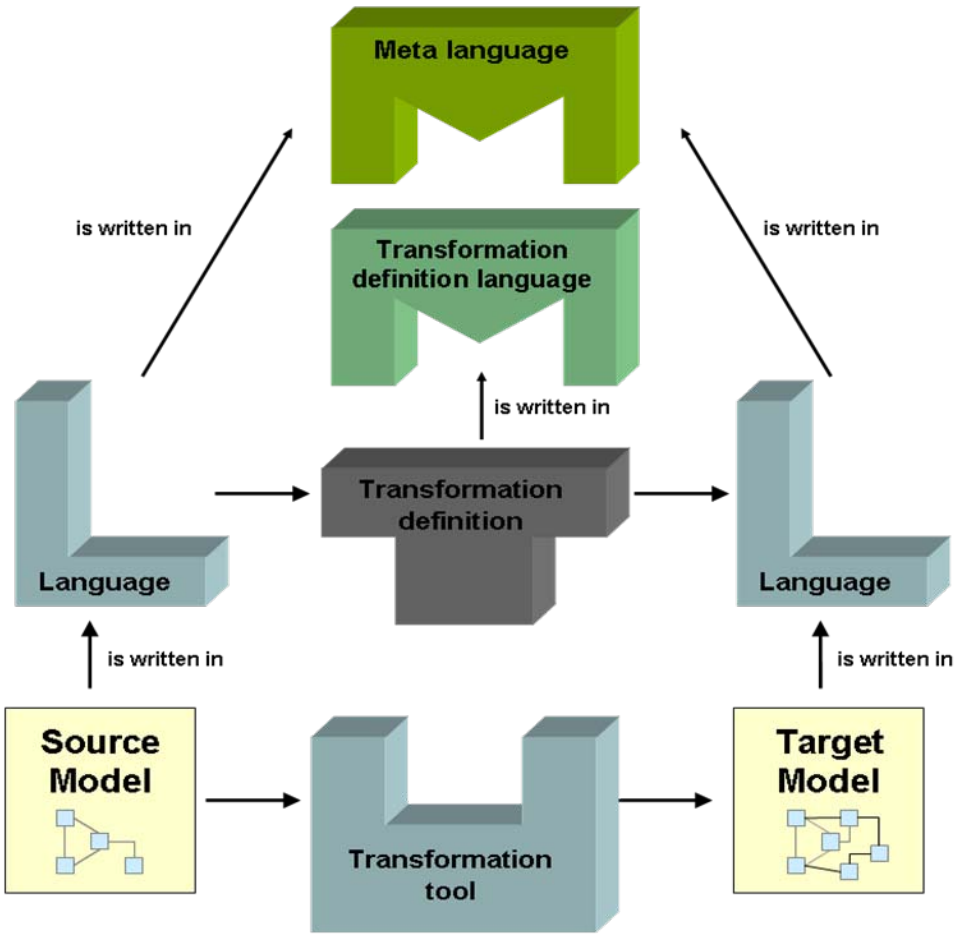


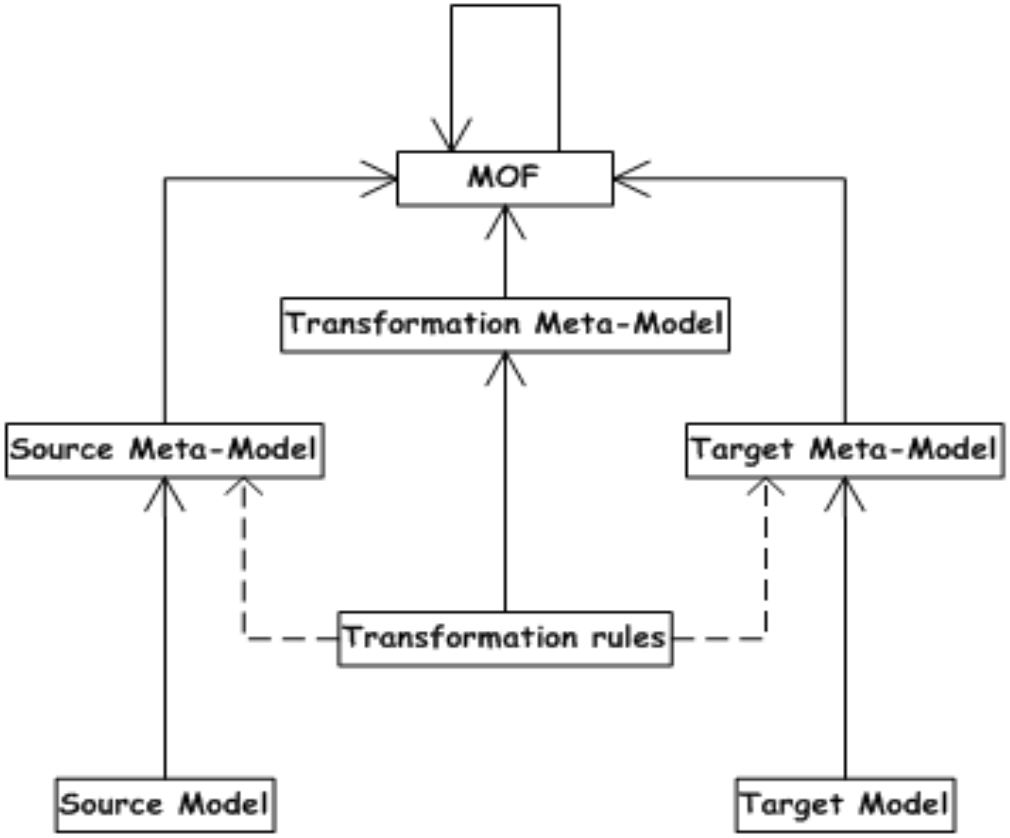
MDA basic elements: Transformations (1)

- A **transformation** is the automatic generation of a *target model* from a *source model*, according to a transformation definition.
- A *transformation definition* is a set of *transformation rules* that together describe *how* a model in the source language can be transformed into a model in the target language.
- A *transformation rule* is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.



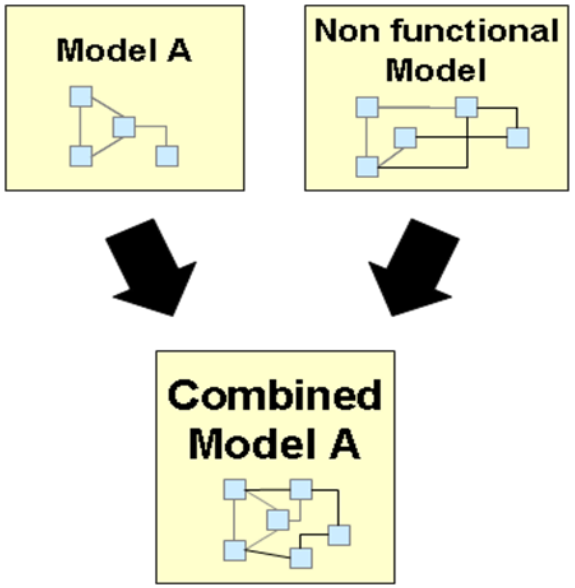
MDA basic elements: Transformations (2)

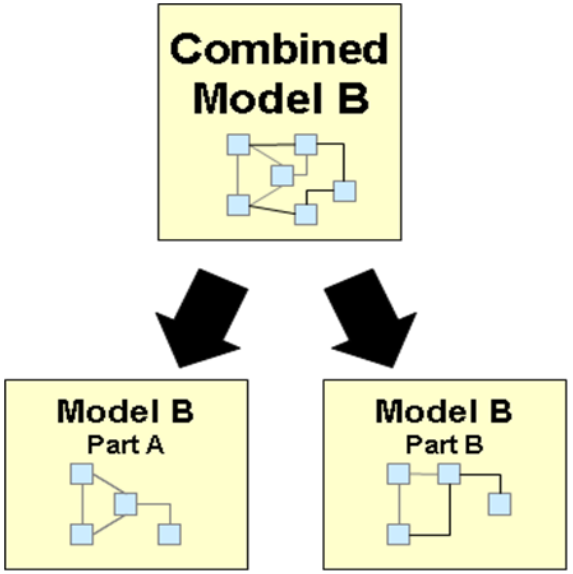




MDA basic elements: Transformations (3)

- **Composition**
  - Special case of transformation
  - allows bringing new details or “aspects” into a model.
  - allows splitting functionality across several platforms.

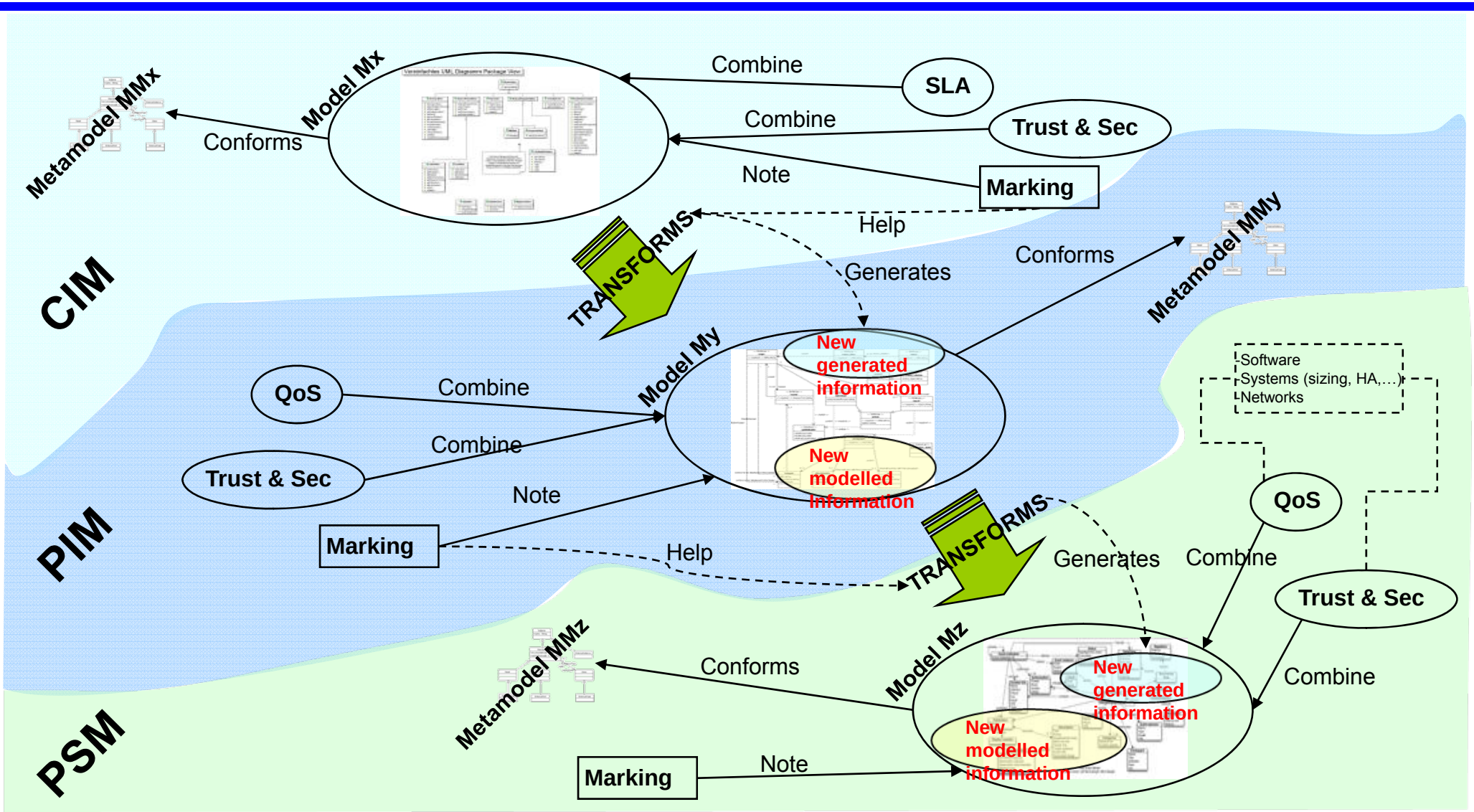




MDA technologies and standards

- **MOF**: Meta-modelling language, repository interface (JMI), interchange (XMI)
- **UML**: Standard modelling language; instance of the MOF model; for developers and "meta-developers"
- **CWM**: modelling languages for data warehousing applications (e.g. Relational DBs)
- **OCL**: expression language, extends the expressive power of UML and MOF
- **QVT**: Transformations definition language; also for Queries and Views of models.
- **SPEM**: metamodel and a UML profile used to describe a concrete software development process.

MDA development process

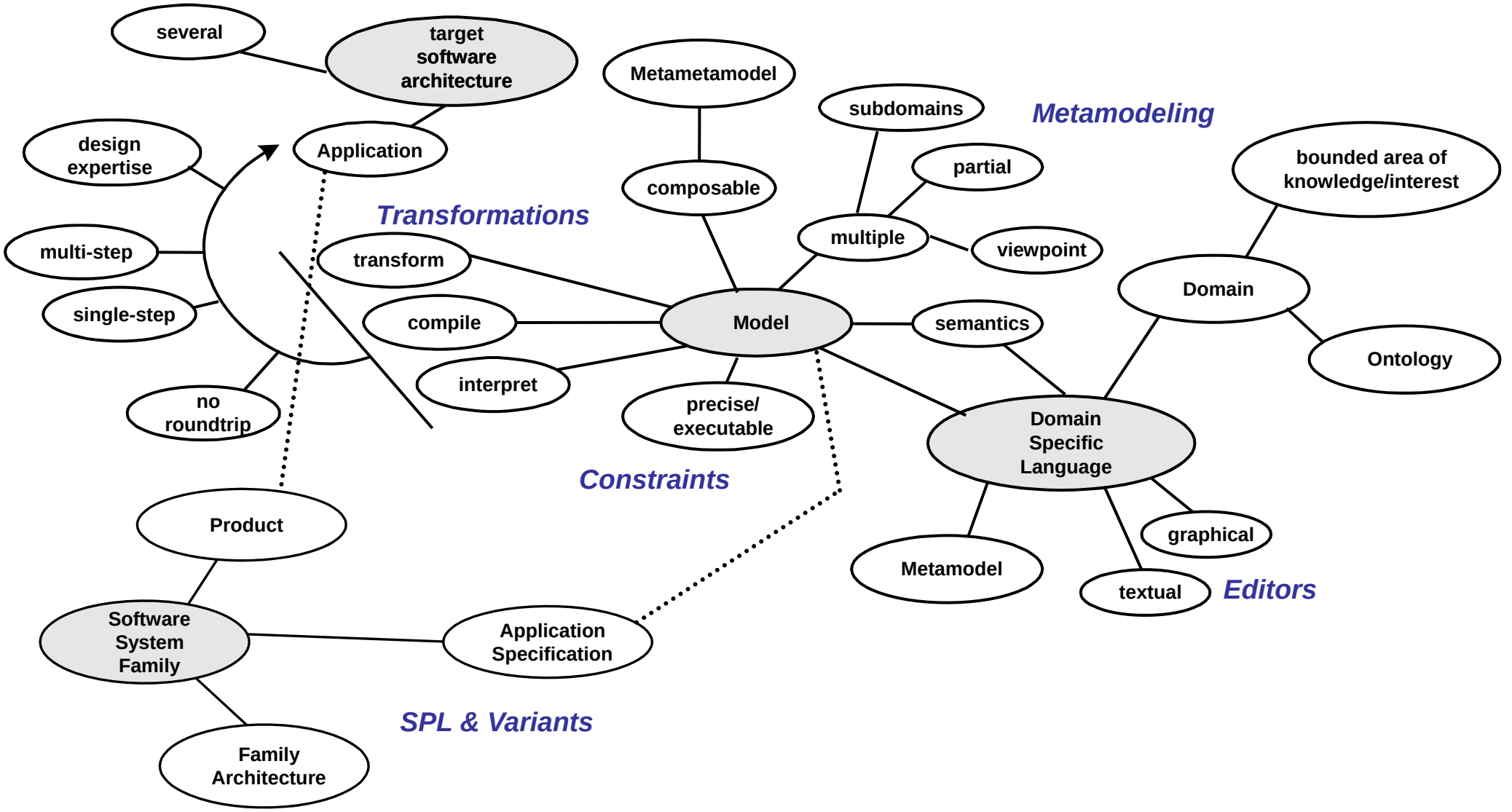


Acronyms / Definitions

- MDE: Model-Driven Engineering
- ME: Model Engineering
- MBDE: Model-Based Data Engineering
- MDA: Model-Driven Architecture
- MDD: Model-Driven Development
- MDSD: Model-Driven Software Development
- MDSE: Model-Driven Software Engineering
- MM: Model Management
- ADM: Architecture-Driven Modernization
- DSL: Domain-Specific Language
- DSM: Domain-Specific Modelling
- etc.

- MDE is a generic term.
- ME and MDSE more or less synonyms of MDE
- MDA™ and MDD™ are OMG trademarks; MDD is a protection trademark (no use as of today/just reserved by OMG for future use).
- MDSD like MDE is sometimes used instead of MDD when one does not wish to be associated to OMG-only technology, vocabulary and vision.
- ADM is another standard intended to be the reverse of MDA: MDA covers forward engineering while ADM covers backward engineering.
- MM mainly used in data engineering like MBDE
- DSM is more Microsoft marked but of increasing use by the academic and research community.

Map of MDSD concepts



References

- Grady Booch, Alan Brown, Sridhar Iyengar, James Rumbaugh, Bran Selic. “An MDA Manifesto”. MDA Journal, May 2004. http://www.ibm.com/software/rational/mda/papers.html
- Marco Brambilla, Jordi Cabot, Manuel Wimmer. Model-Driven Software Engineering in Practice. Morgan & Claypool, 2012.
- Jack Greenfield, Keith Short, Steve Cook, Stuart Kent. Software Factories. John Wiley & Sons, 2004.
- Chris Raistrick, Paul Francis, John Wright, Colin Carter, Ian Wilkie. Model-Driven Architecture with Executable UML. Cambridge University Press, 2004.
- Thomas Stahl, Markus Völter, Sven Efftinge, Arno Haase. *Modellgetriebene Softwareentwicklung*. dpunkt.verlag, ²2007.
- Peter Swithinbank, Mandy Chessell, Tracy Gardner, Catherine Griffin, Jessica Man, Helen Wylie, Larry Yussuf. Patterns: Model-Driven Development Using IBM Rational Software Architect. IBM Redbooks, 2005. http://www.redbooks.ibm.com/redbooks/sdbooks/pdfs/sg247105.pdf

- Stephan Roser. Vorlesung „Modellgetriebene Softwareentwicklung“. Universität Augsburg, Sommersemester 2008.

Meta-Modelling

Model vs. System

René Magritte. La trahison des images. 1928–29.

Model of a model — The correspondence continuum

Meaning is rarely a simple mapping from a symbol to an object; instead it often involves a ***continuum of (semantic) correspondences*** *from symbol to (symbol to)* object.* [Barry Smith. The correspondence continuum. 1987]

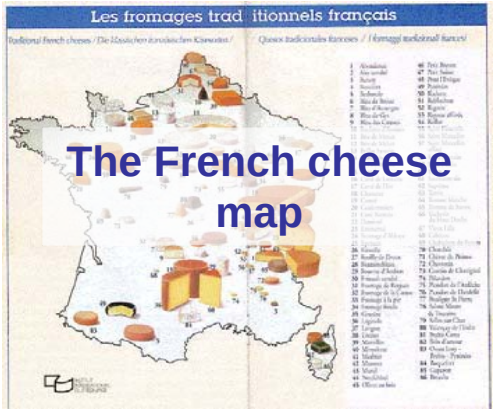
- Example
  - A photo of a landscape is a model of the landscape.
  - A photocopy of the photo is model of a model of the landscape.
  - A digitalization of the photocopy is a model of the model of the model of the landscape.
  - etc.

Basic entities of MDE and MDSD

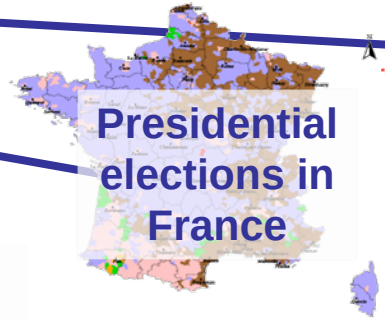
***System*: a group of interacting, interrelated, or interdependent elements forming a complex whole.**

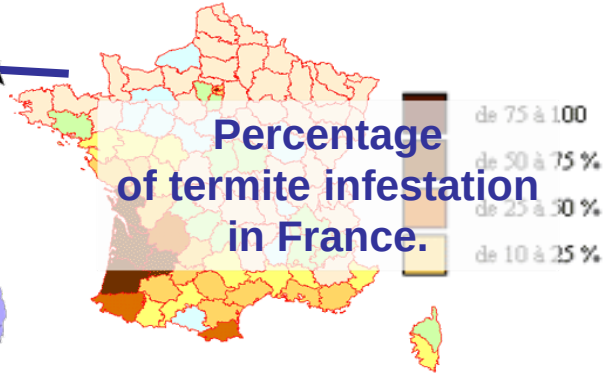
***Model*: an abstract representation of a system created for a specific purpose.**

A very popular model: Geographical maps



La France des "Huit présidents"
candidat arrivé en tête à l'issue du premier tour





Models

System ← repOf — Model

Limited substitutability principle

- The purpose of a model is always to be able to answer some specific sets of questions in place of the system, exactly in the same way the system itself would have answered similar questions.

- A model represents certain specific aspects of a system and only these aspects, for a specific purpose.

Lewis Carroll and the 1:1 map

“That’s another thing we’ve learned from your Nation” said Mein Herr, “map-making. But we’ve carried it much further than you. What do you consider the ***largest*** map that would be really useful?”

“About six inches to the mile.”

“Only *six inches*!” exclaimed Mein Herr. “We very soon got to *six yards* to *the mile*. Then we tried a hundred yards to the mile. And then came the grandest idea of all! We actually made a map of the country, on the scale of ***a mile to the mile***!”

”***Have you used it much***?” I enquired.

“***It has never been spread out***, yet” said Mein Herr: “the farmers objected: they said it would cover the whole country, and shut out the sunlight! So we now use the country itself, as its own map, and I assure you it does nearly as well.“

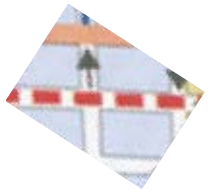
Lewis Carroll. Sylvie and Bruno concluded.

Lewis Carroll and the blank map

He had bought a large map representing the sea,
Without the least vestige of land:
And the crew were much pleased when they found it to be
A map they could all understand.
“What's the good of Mercator's North Poles and Equators,
Tropics, Zones, and Meridian Lines?”
So the Bellman would cry: and the crew would reply
“They are merely conventional signs!
Other maps are such shapes, with their islands and capes!
But we've got our brave Captain to thank:”
(So the crew would protest) “that he's bought us the best—
A perfect and absolute blank!”

Lewis Carroll. The Hunting Of The Snark — An Agony in Eight Fits.

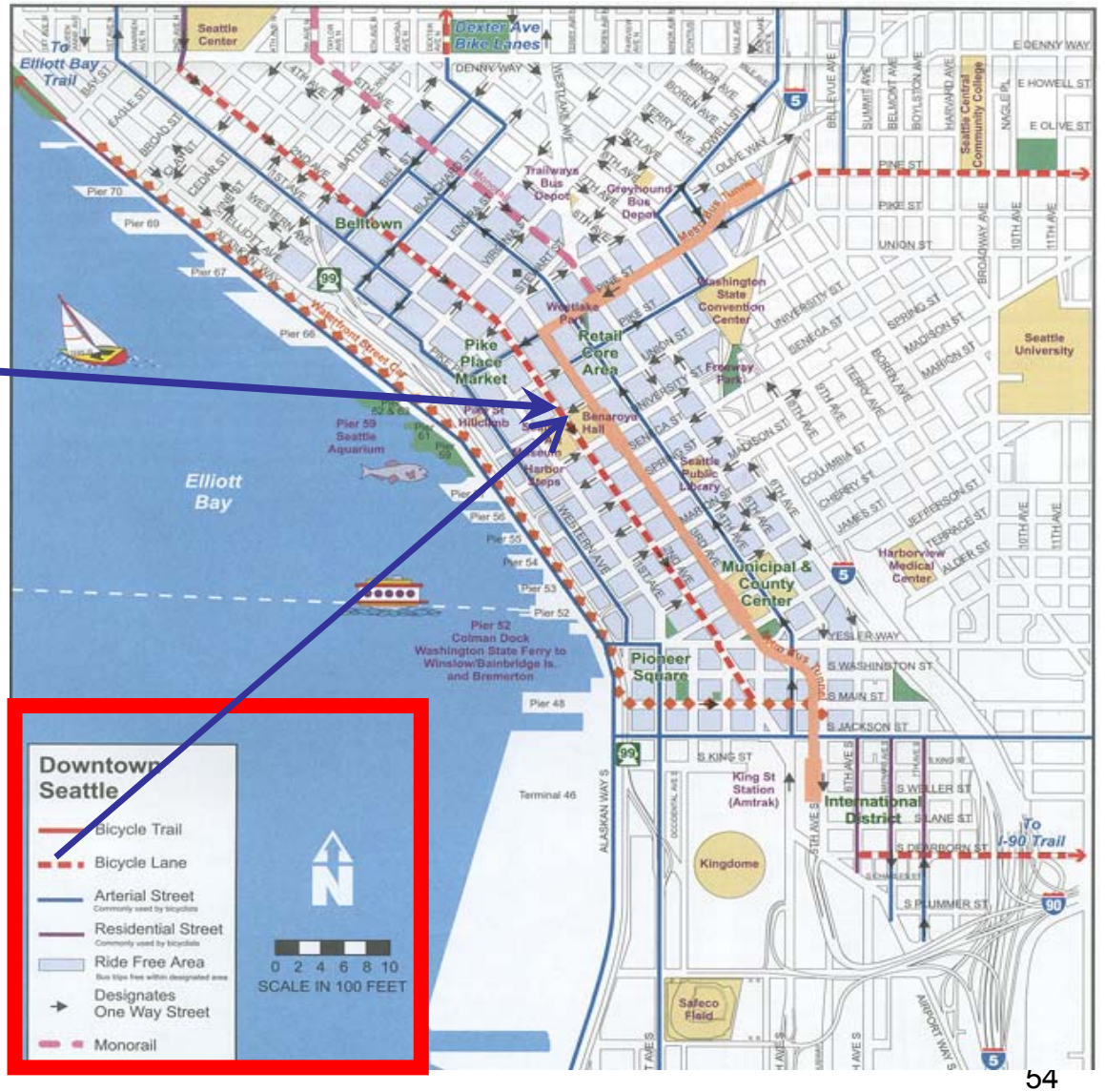
Every map has a legend (implicit or explicit)



Bicycle Lane

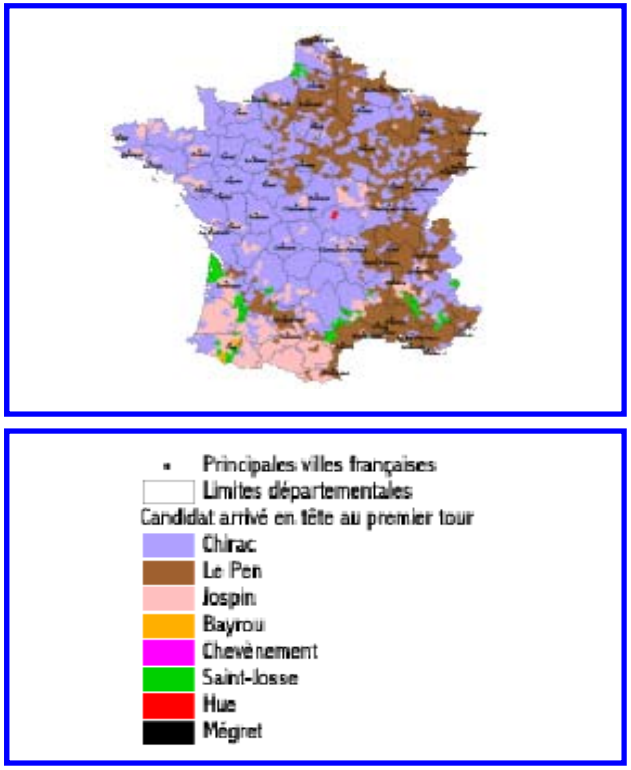
Same visual notation, different context, different meaning

The legend is the metamodel

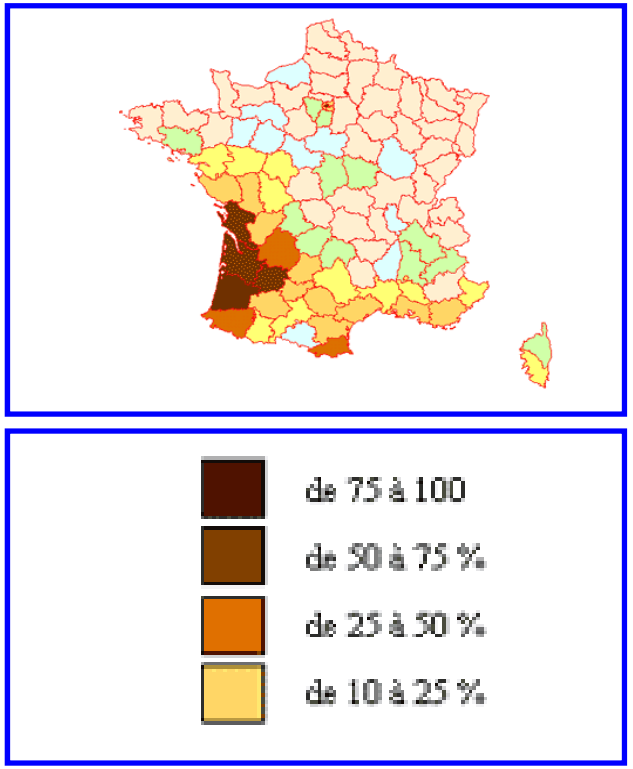


Maps without legends are meaningless

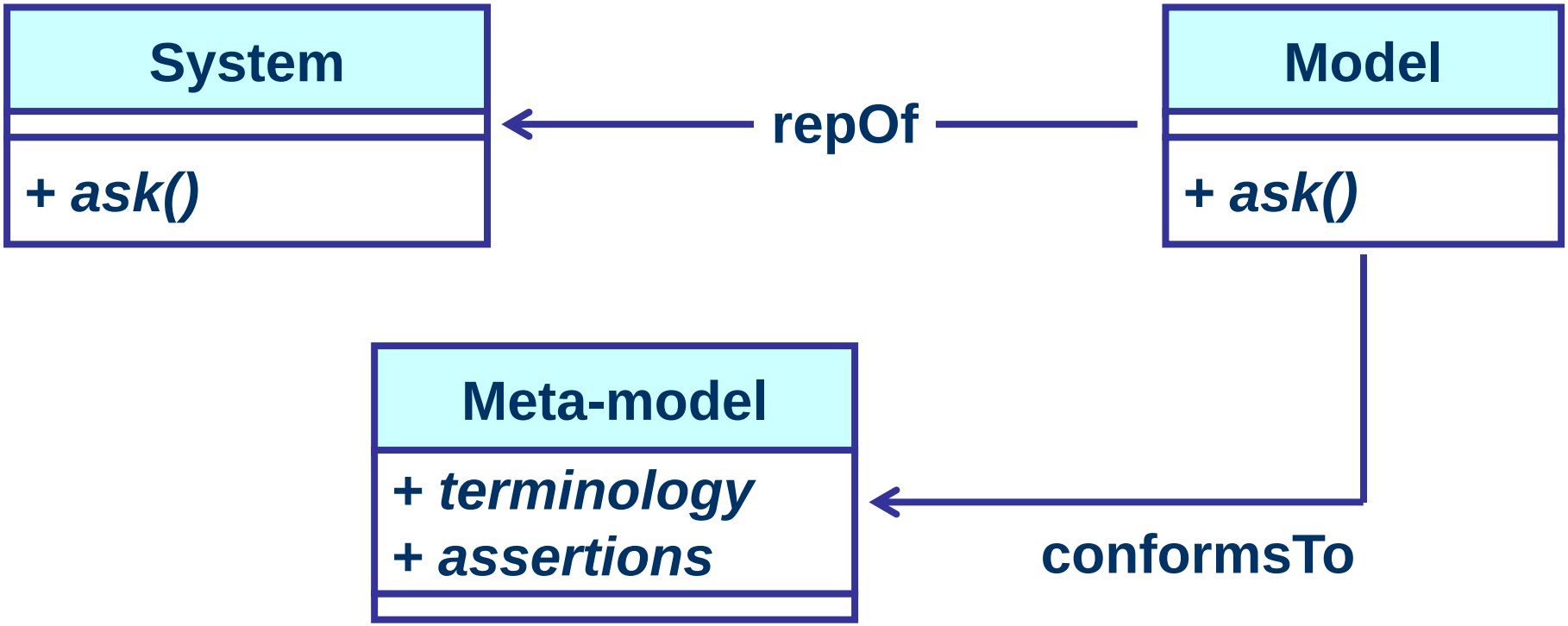
First round of political election in France in 2002



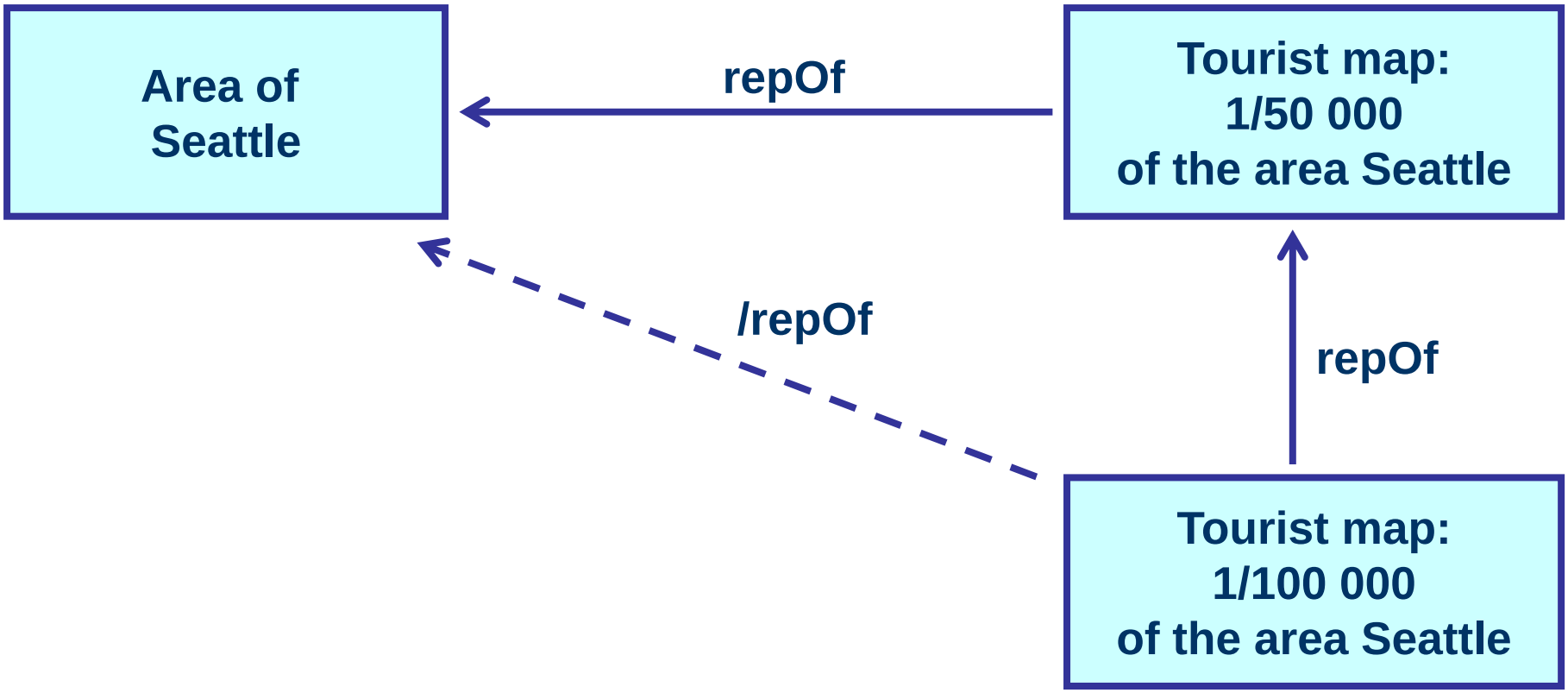
Percentage of places infested by termites in France



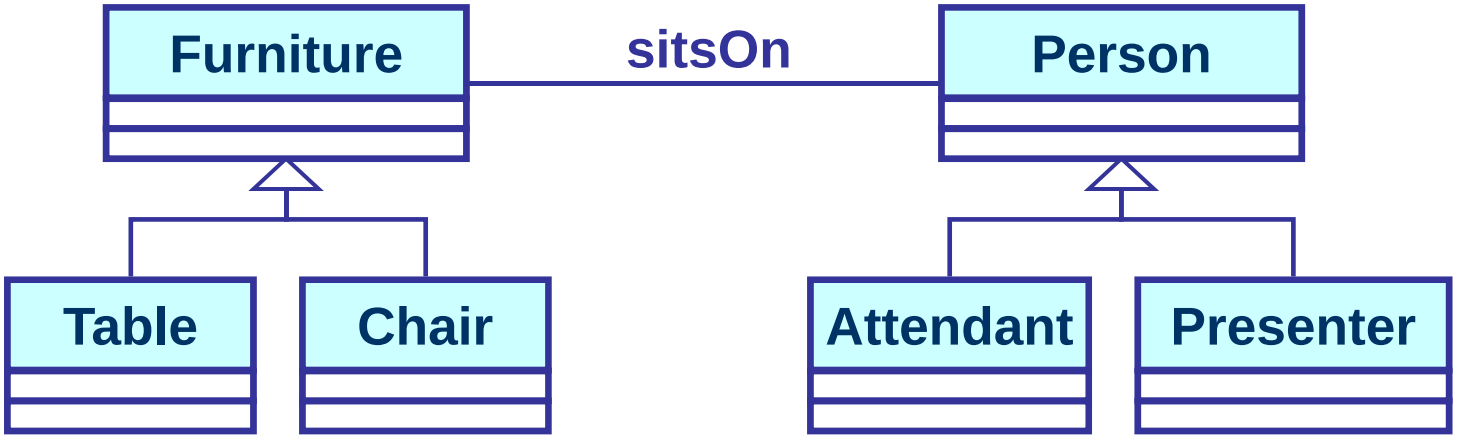
The legend is a meta-model



The model of a model is not a meta-model



Meta-models act as filters



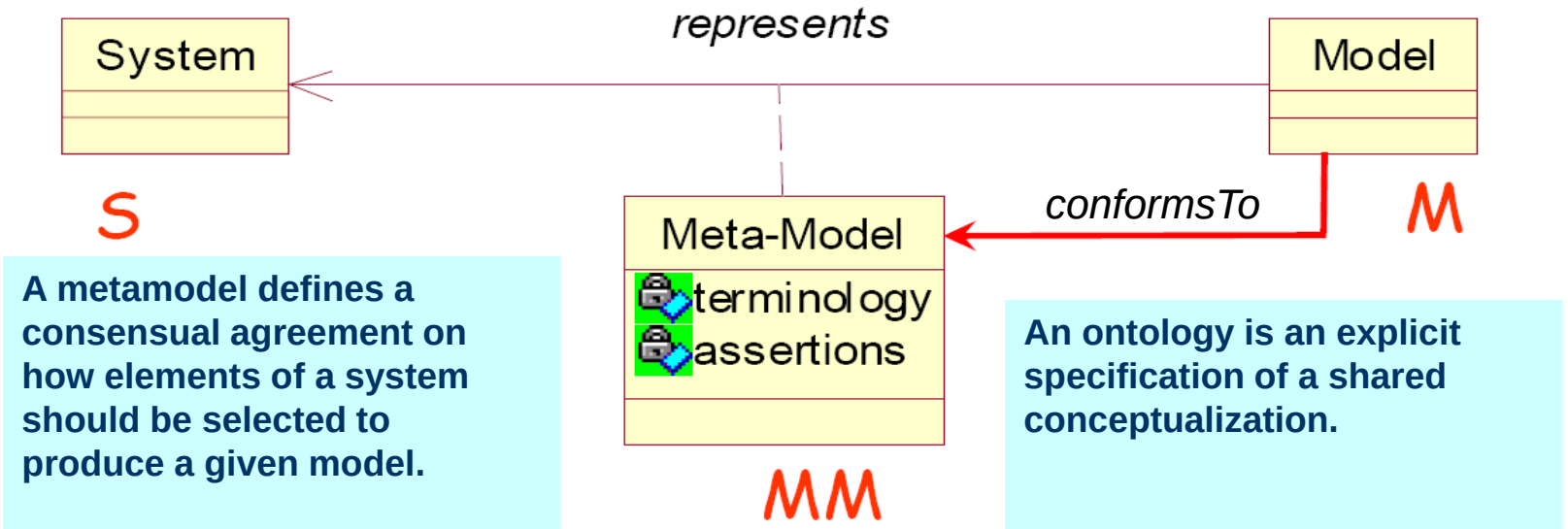
The metamodel

Mary
Table 237
Chair 34
Paul
Victor
Emily

A model

Meta-models as simple ontologies

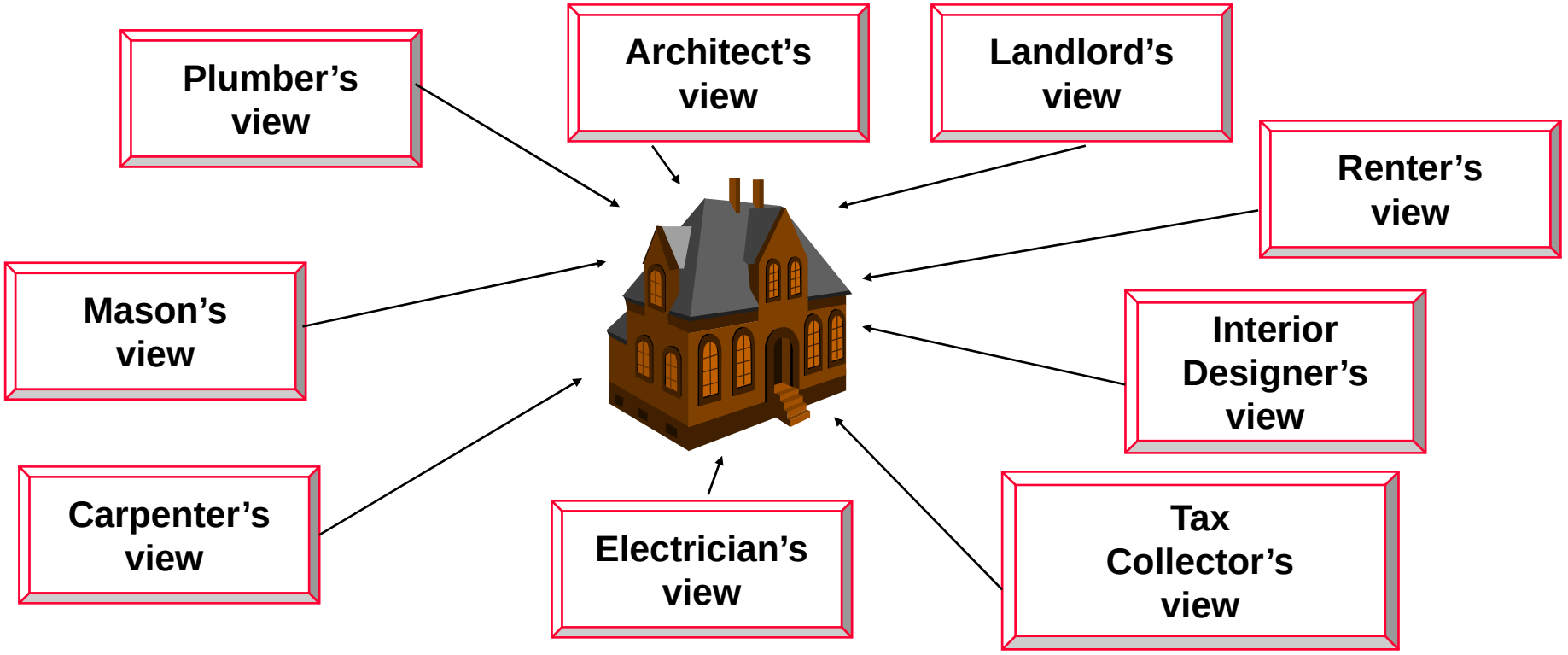
- Meta-models are precise abstraction filters.
- Each meta-model defines a domain-specific language.
- Each meta-model is used to specify which particular “aspect” of a system should be considered to constitute the model.



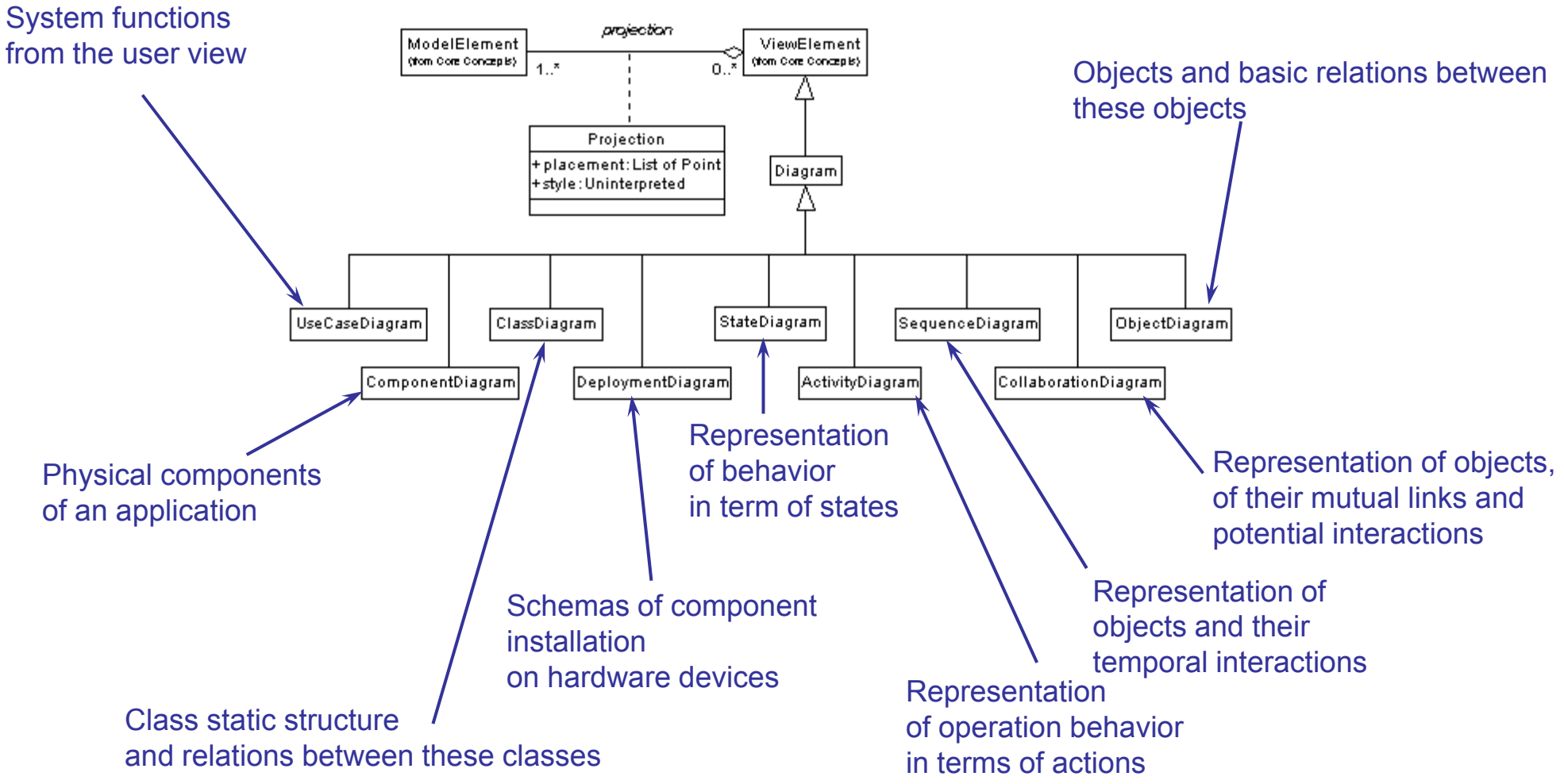
- The correspondence between a system and a model is precisely and computationally defined by a meta-model.

Multiple views and coordinated DSLs

- 1:1 map vs. blank map
- Limited substitutability principle
- A model has no meaning when separated from its meta-model.



Multiple views and aspects of a software system



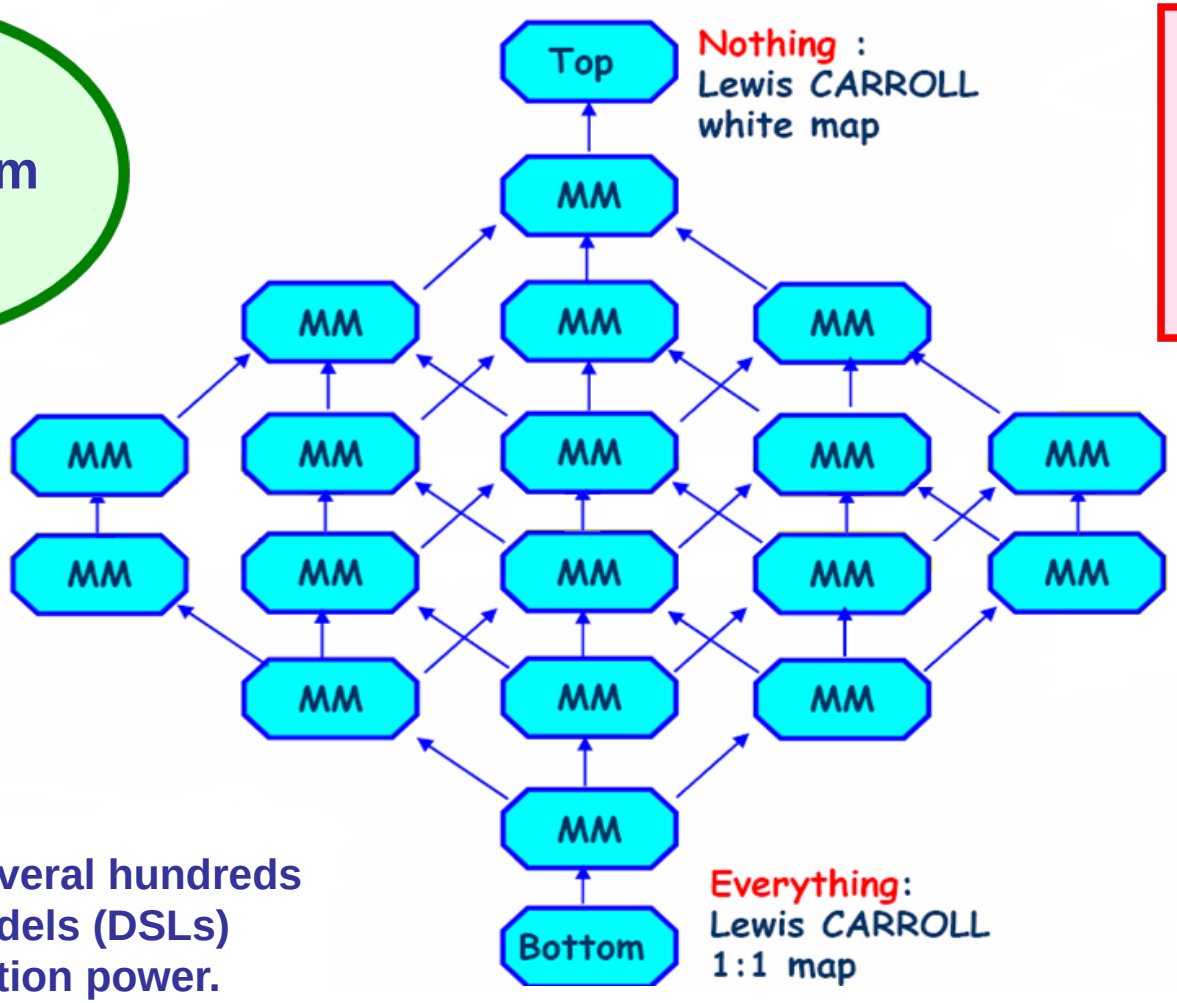
Meta-models

- A meta-model is just another model.
  - **Model of a set of models**
- Meta-models are specifications.
  - Models are valid if no false statements according to meta-model (e.g. well-formed)
  - Meta-models typically represent domain-specific models (real-time systems, safety critical systems, e-business)
- The domain of meta-modelling is language definition.
  - A meta-model is a model of some part of a language
  - Which part depends on how the meta-model is to be used
  - Parts: syntax, semantics, views/diagrams, ...

- **Meta-meta-model**
  - Model of meta-models
  - Reflexive meta-models expressed using itself

A “lattice” of meta-models

The system

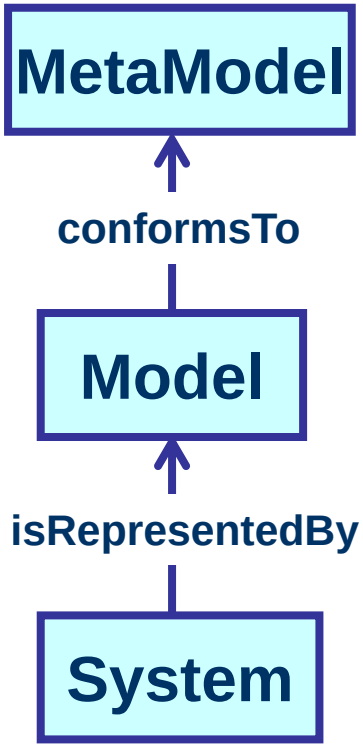


A model

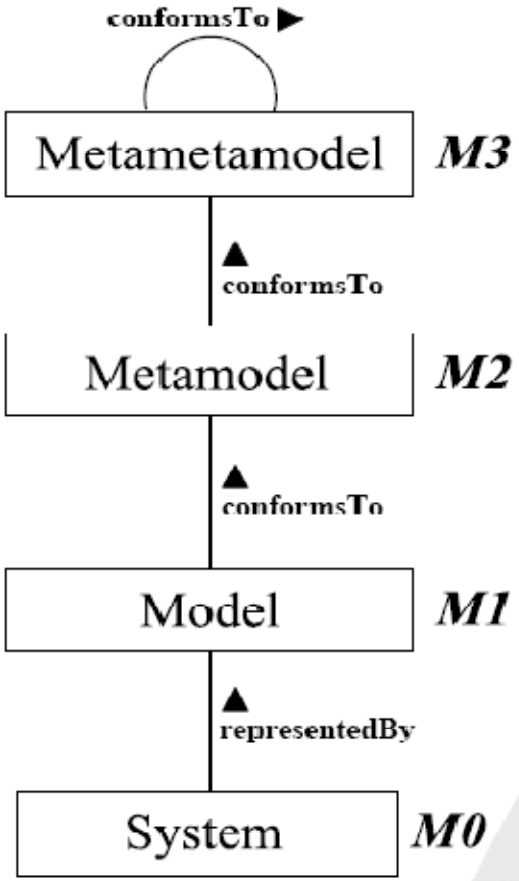
A collection of several hundreds of small meta-models (DSLs) with high abstraction power.

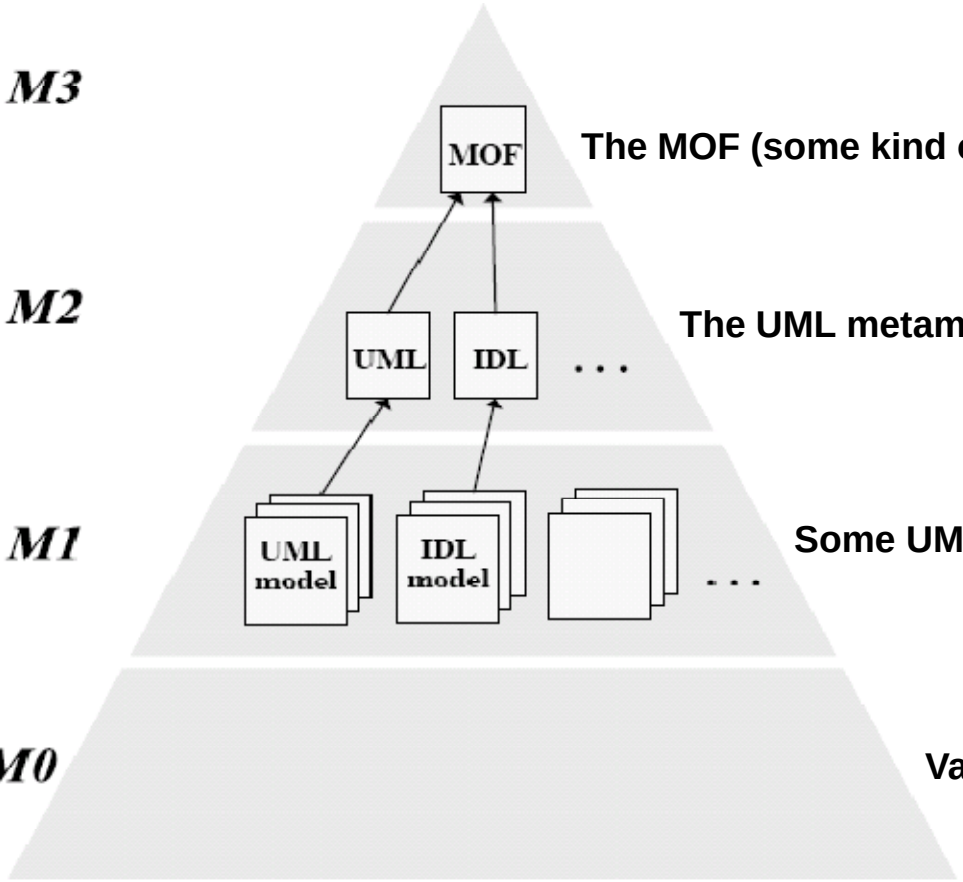
The basic assumptions of MDE and MDSD

- Models as first class entities
- ***Conformance*** and ***Representation*** as kernel relations central to MDE
  - MDSD as a special case of MDE



Meta-modelling hierarchy or the meta-modelling stack





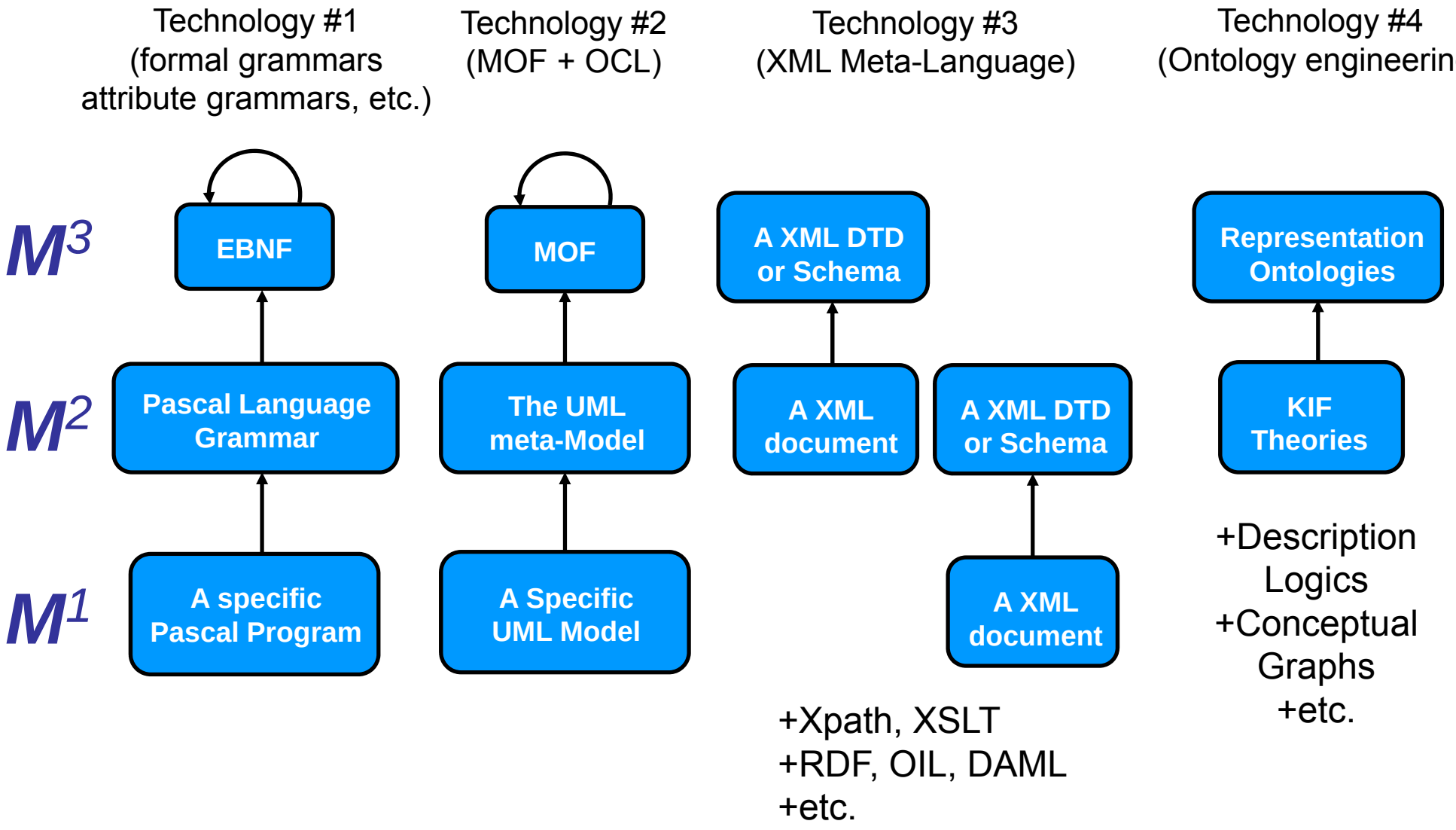
The MOF (some kind of "representation ontology")

The UML metamodel and other MMs

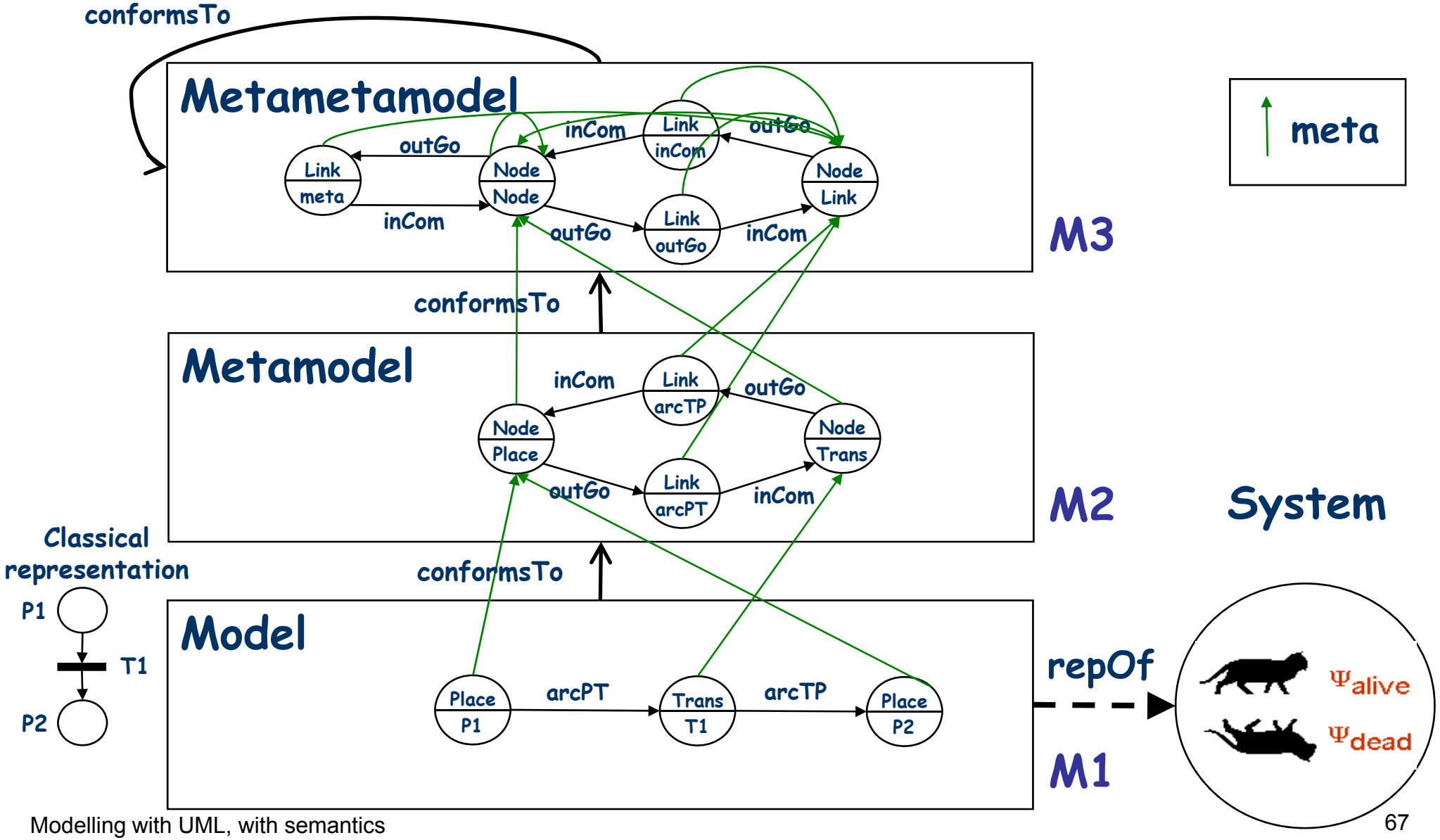
Some UML Models and other Ms

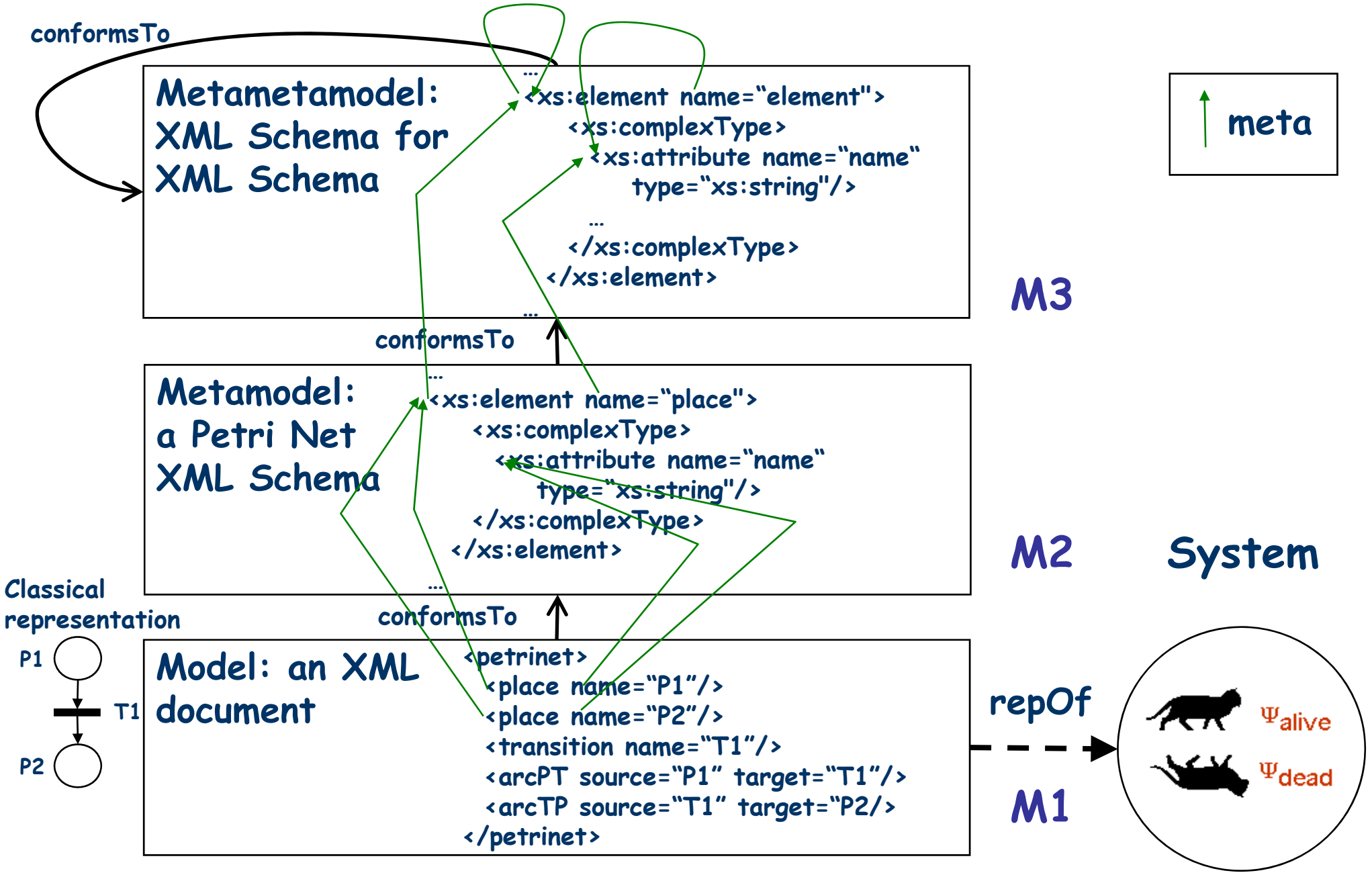
Various usages of these models

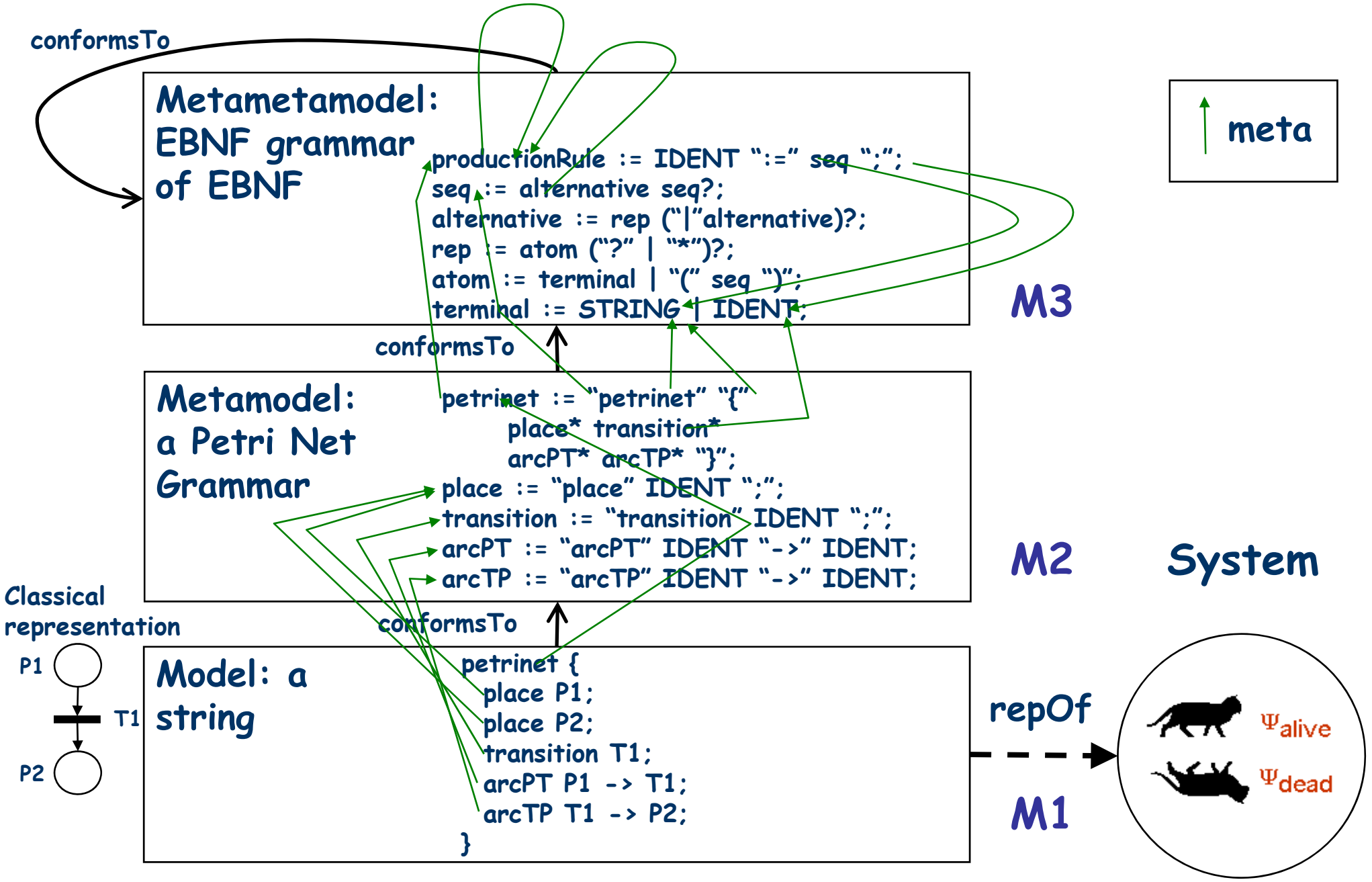
Abstract Syntax Systems Compared



Three-level hierarchy: Example — Petri-nets

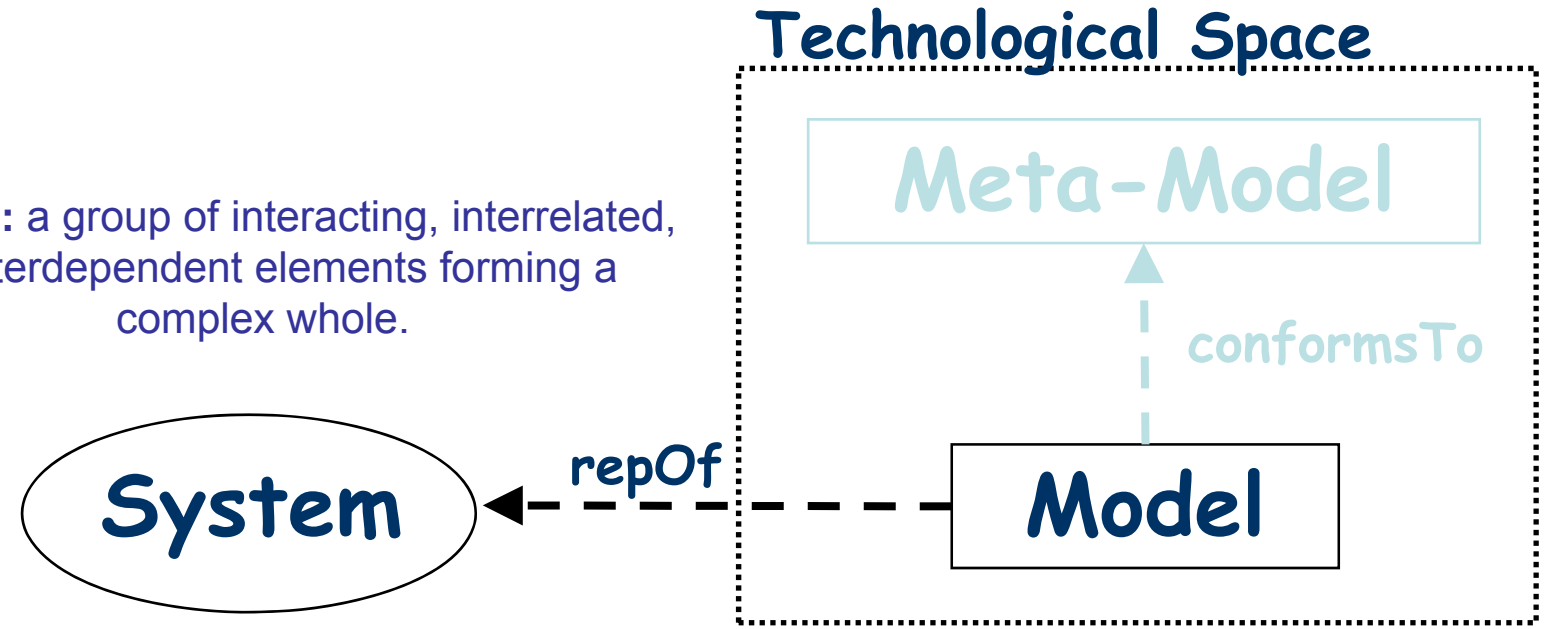



conformsTo
Metametamodel: XML Schema for XML Schema
<xs:element name="element">
<xs:complexType>
<xs:attribute name="name"
type="xs:string"/>
</xs:complexType>
</xs:element>
meta
M3
conformsTo
Metamodel: a Petri Net XML Schema
<xs:element name="place">
<xs:complexType>
<xs:attribute name="name"
type="xs:string"/>
</xs:complexType>
</xs:element>
M2
System
Classical representation
P1
T1
P2
conformsTo
Model: an XML document
<petrinet>
<place name="P1"/>
<place name="P2"/>
<transition name="T1"/>
<arcPT source="P1" target="T1"/>
<arcTP source="T1" target="P2/>
</petrinet>
repOf
M1
Ψalive
Ψdead


conformsTo
Metametamodel:
EBNF grammar
of EBNF
productionRule := IDENT ":=" seq ";";
seq := alternative seq?;
alternative := rep ("|"alternative)?;
rep := atom ("?" | "*")?;
atom := terminal | "(" seq ")";
terminal := STRING | IDENT;
meta
M3
conformsTo
Metamodel:
a Petri Net
Grammar
petrinet := "petrinet" "{"
place* transition*
arcPT* arcTP* "}";
place := "place" IDENT ";";
transition := "transition" IDENT ";";
arcPT := "arcPT" IDENT "->" IDENT;
arcTP := "arcTP" IDENT "->" IDENT;
M2
System
Classical
representation
P1
T1
P2
conformsTo
Model: a
string
petrinet {
place P1;
place P2;
transition T1;
arcPT P1 -> T1;
arcTP T1 -> P2;
}
repOf
M1
Ψalive
Ψdead

Basic entities of MDE and MDSD

***Technological Space*: a model management framework usually based on some algebraic structures (trees, graphs, hypergraphs, etc.).**

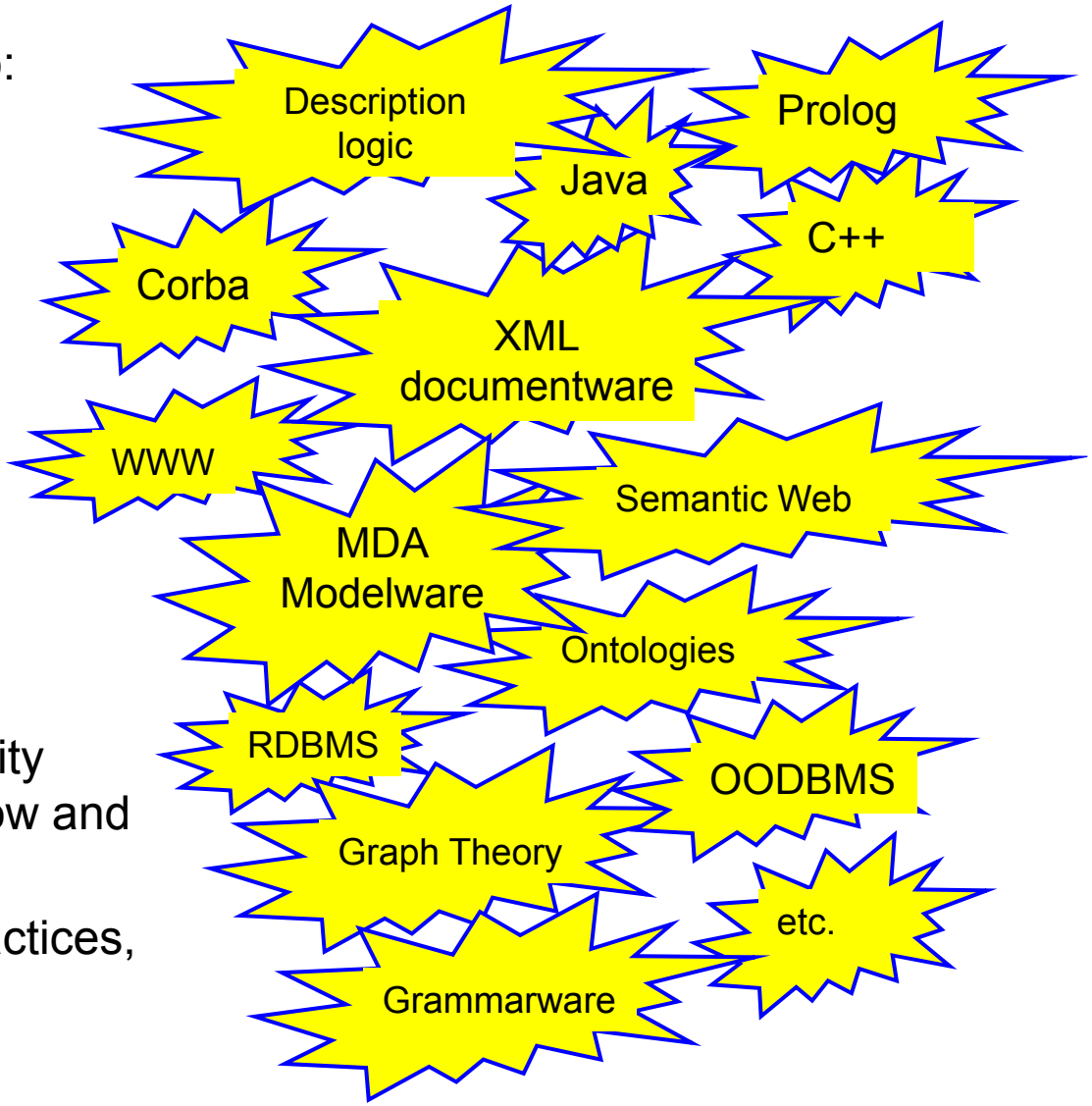


***System*: a group of interacting, interrelated, or interdependent elements forming a complex whole.**

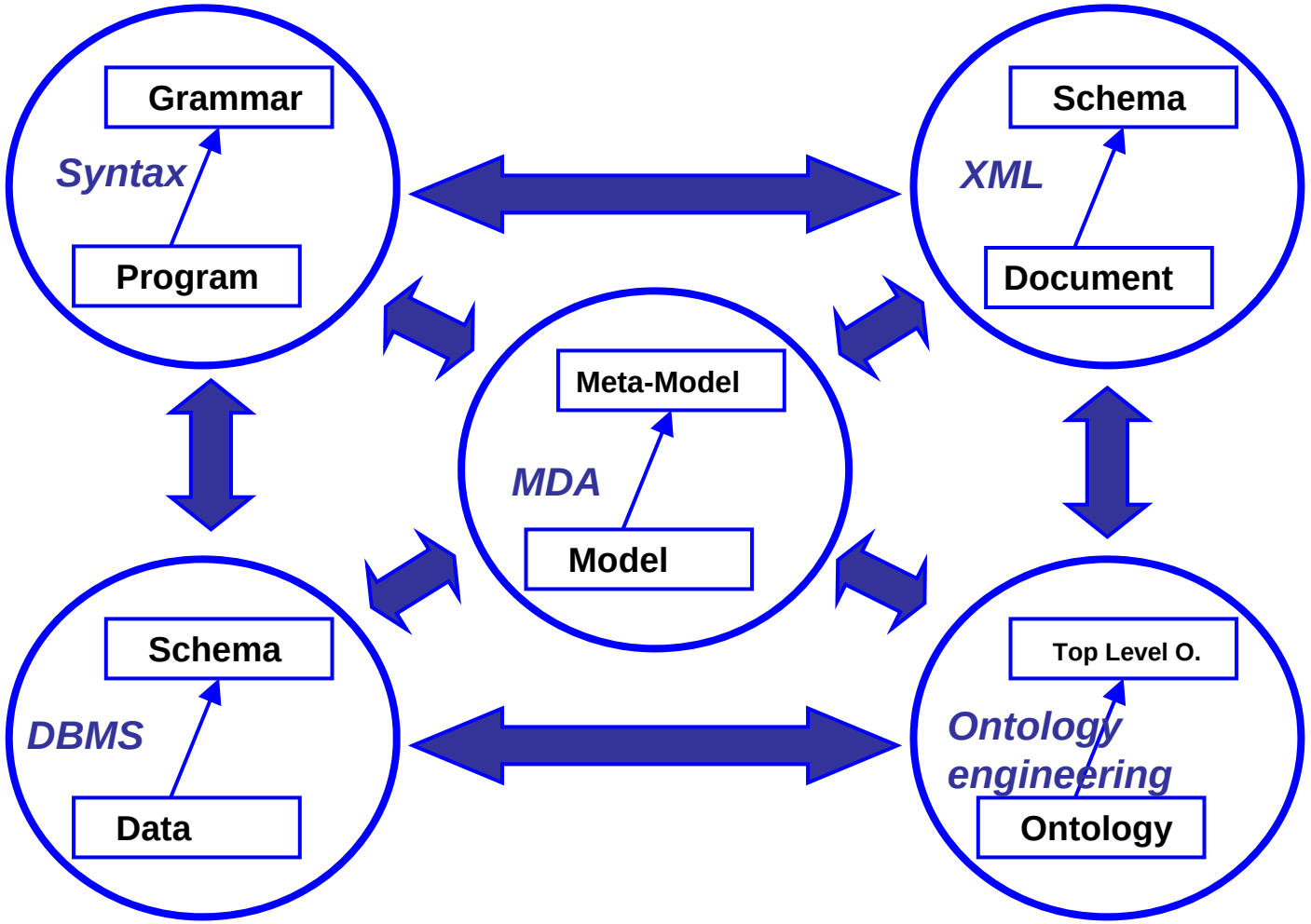
***Model*: an abstract representation of a system created for a specific purpose.**

The notion of Technological Space (TS)

- A Technological Space corresponds to:
  - A **uniform representation system**
    - Syntactic trees
    - XML trees
    - Sowa graphs
    - UML graphs
    - MOF graphs
  - A **working context**
  - A **set of concepts**
  - A **set of methods**
  - A **shared knowledge and know-how**
  - etc.
- It is usually related to a given community with an established expertise, know-how and research problems.
- It has a set of associated tools and practices, etc.
  - Protégé, Rational Rose, …



Main Technological Spaces

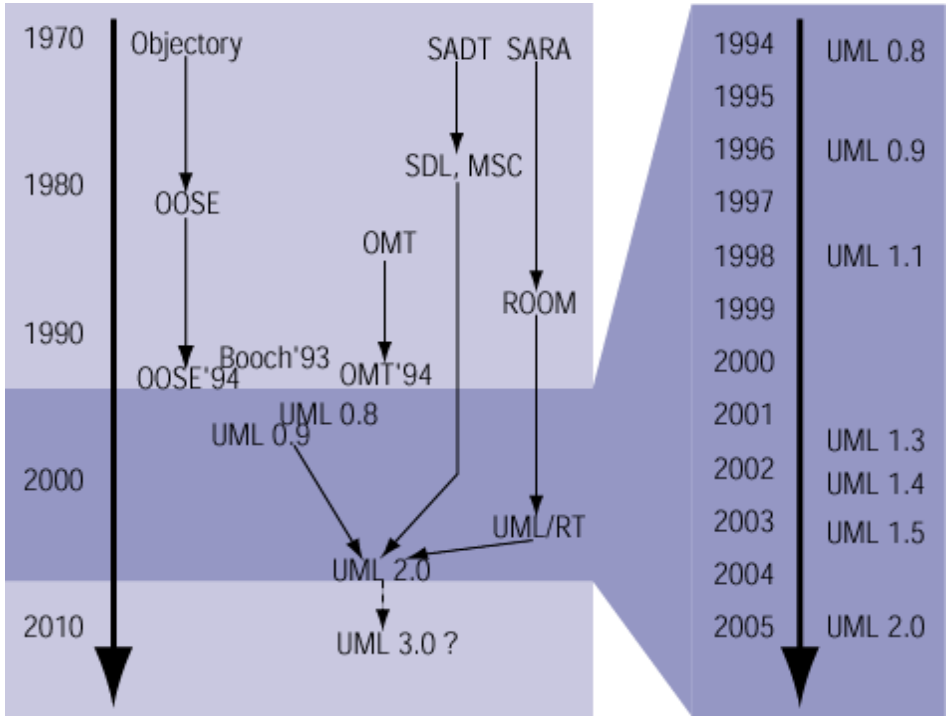


TS's may be connected via **bridges**

Unified Modeling Language 2

History and Predecessors

- The UML is the “lingua franca” of software engineering.
- It subsumes, integrates and consolidates most predecessors.
- Through the network effect, UML has a much broader spread and much better support (tools, books, trainings etc.) than other notations.
- The transition from UML 1.x to UML 2.0 has
  - resolved a great number of issues;
  - introduced many new concepts and notations (often feebly defined);
  - overhauled and improved the internal structure completely.
- While UML 2 still has many problems, it is much better than what we ever had before.

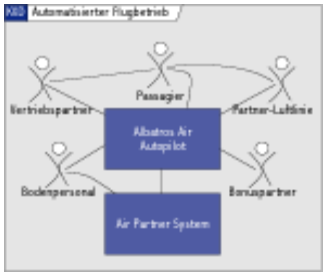


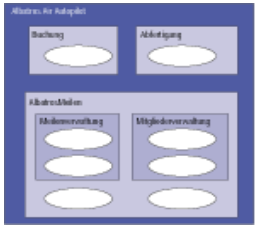
current version (“the standard”) UML 2.4.1
formal/2011-08-06 of August ’11

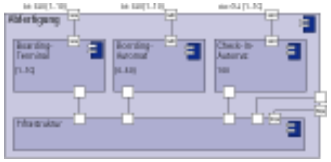
Usage Scenarios

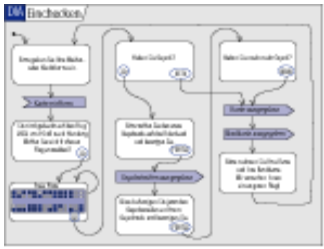
- UML has not been designed for specific, limited usages.
- There is currently no consensus on the rôle of the UML:
  - Some see UML only as tool for sketching class diagrams representing Java programs.
  - Some believe that UML is "*the prototype of the next generation of programming languages*".
- UML is a really a system of languages ("notations", "diagram types") each of which may be used in a number of different situations.
- UML is applicable for a multitude of purposes and during all phases of the software lifecycle – to varying degrees.

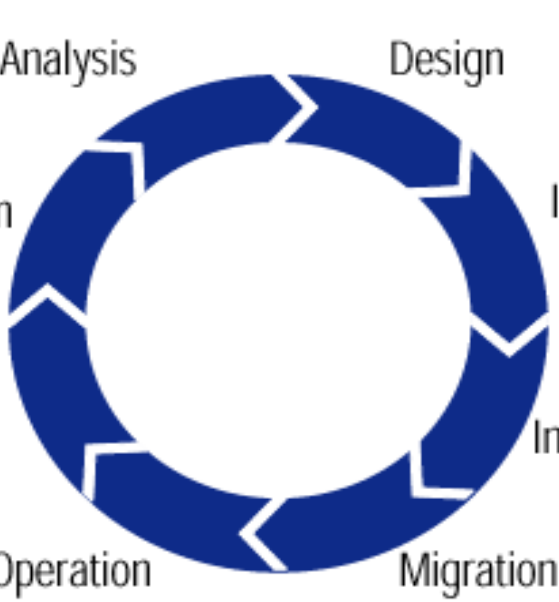
Usage Scenarios

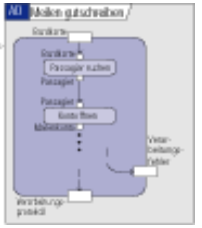


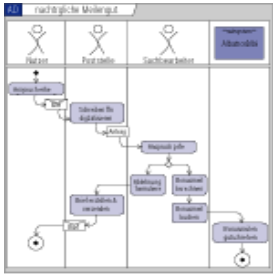


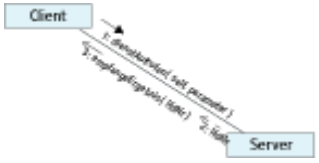


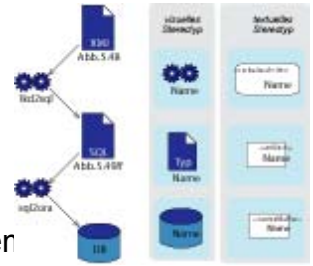


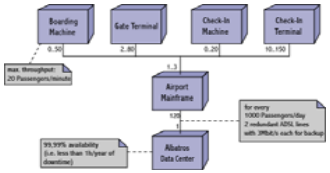


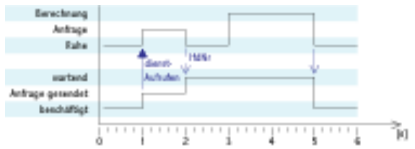


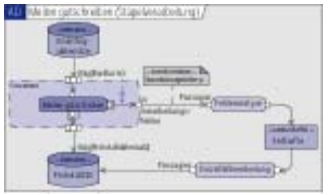


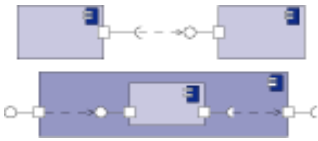












ML, with ser

Diagram types in UML 2

UML is a coherent system of languages rather than a single language.
Each language has its particular focus.

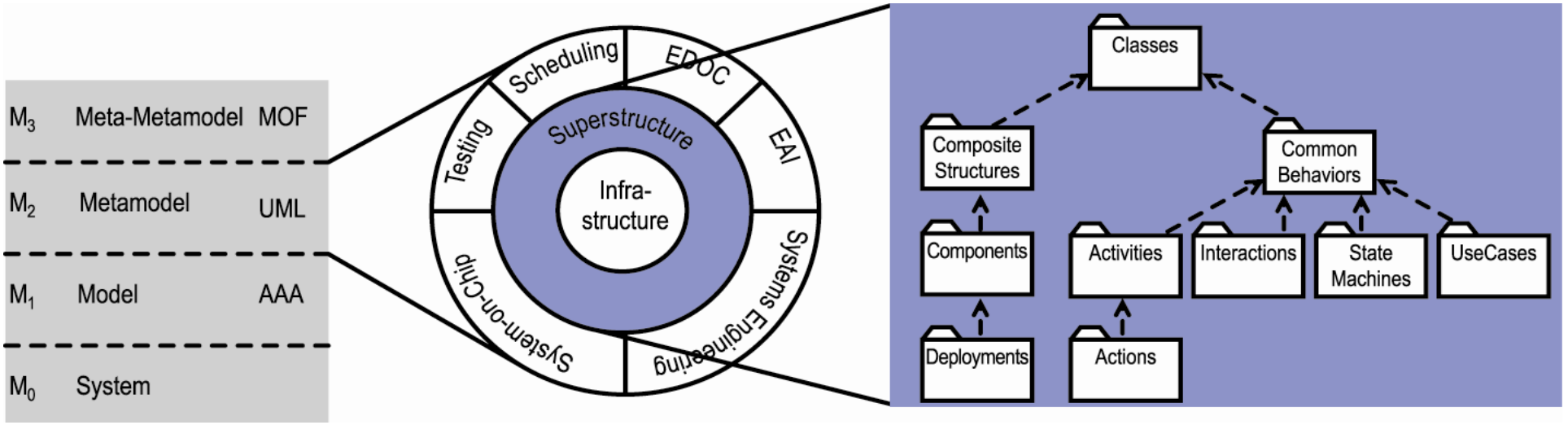
Structure	Class Diagram		static structure (generic/snapshot)
	Composite Structure Diagram		logical system structure
	Component Diagram		physical system structure
	Deployment Diagram		computing infrastructure / deployment
	Package Diagram		containment hierarchy
Behavior	Use Case Diagram		abstract functionality
	Activity Diagram		controlflow and dataflow
	Interaction	Sequence Diagram	*interactions by message exchange* message exchange over time
		Communication Diagram	structure of interacting elements
		Timing Diagram	coordinated state change over time
		Interaction Overview Diagram	flows of interactions
	State Machine Diagram		event-triggered state change

Internal Structure: Overview

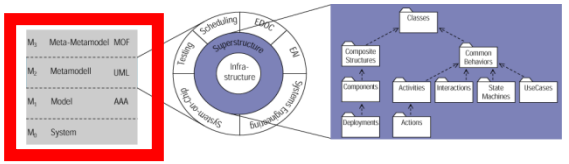
- The UML is structured using a metamodeling approach with four *layers*.
- The M_2-layer is called metamodel.

- The metamodel is again structured into *rings*, one of which is called superstructure, this is the place where concepts are defined ("the metamodel" proper).

- The Superstructure is structured into a tree of *packages* in turn.

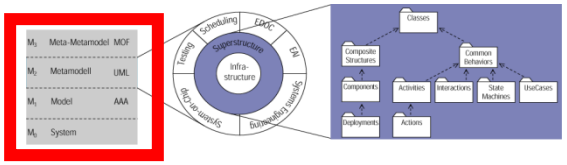


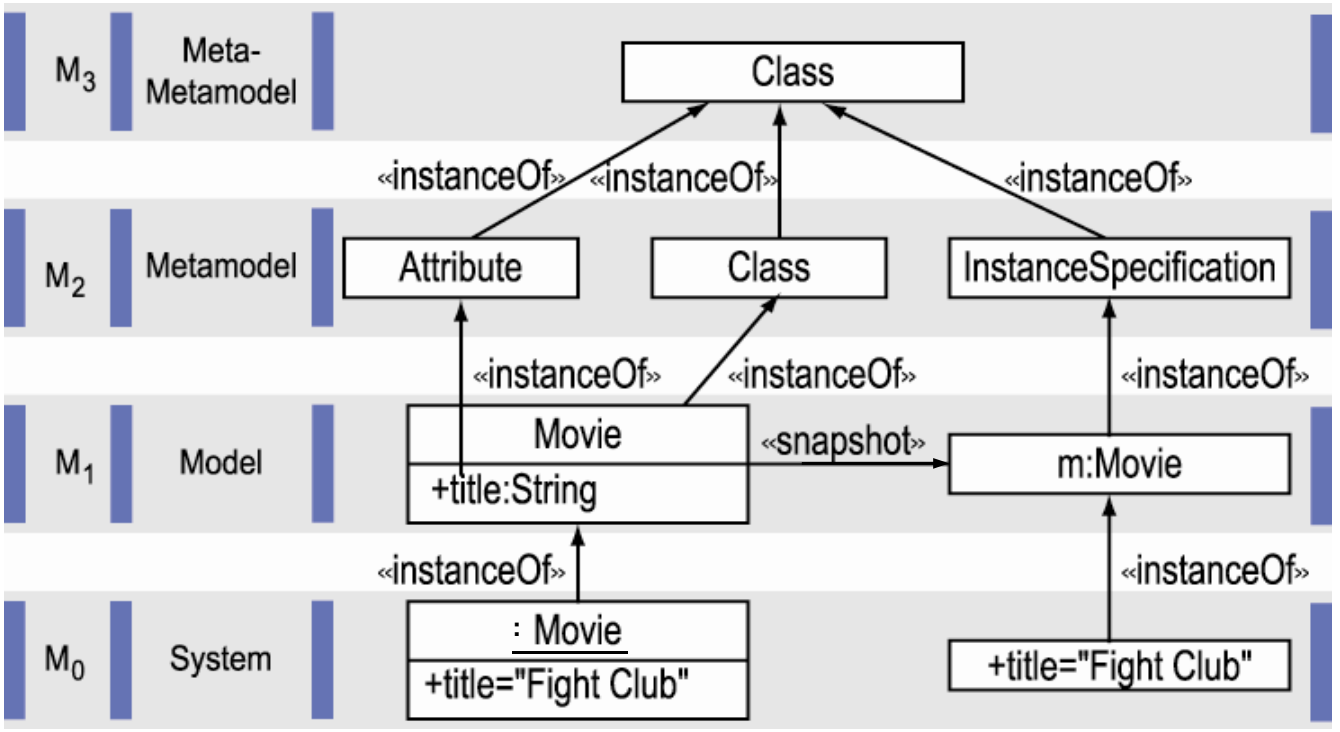
Internal Structure: Layers



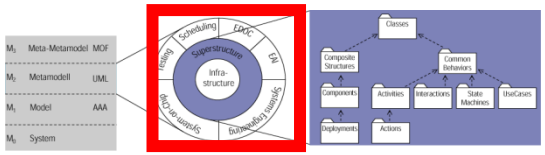
M_3	Meta-Metamodel	EBNF	Meta Object Facility (MOF)
M_2	Metamodel	Java grammar	Unified Modeling Language (UML) Common Warehouse Metamodel (CWM)
M_1	Model	a Java program	Albatros Air Autopilot
M_0	System	an execution of a Java program	a runtime state in a deployment of Albatros Air Autopilot

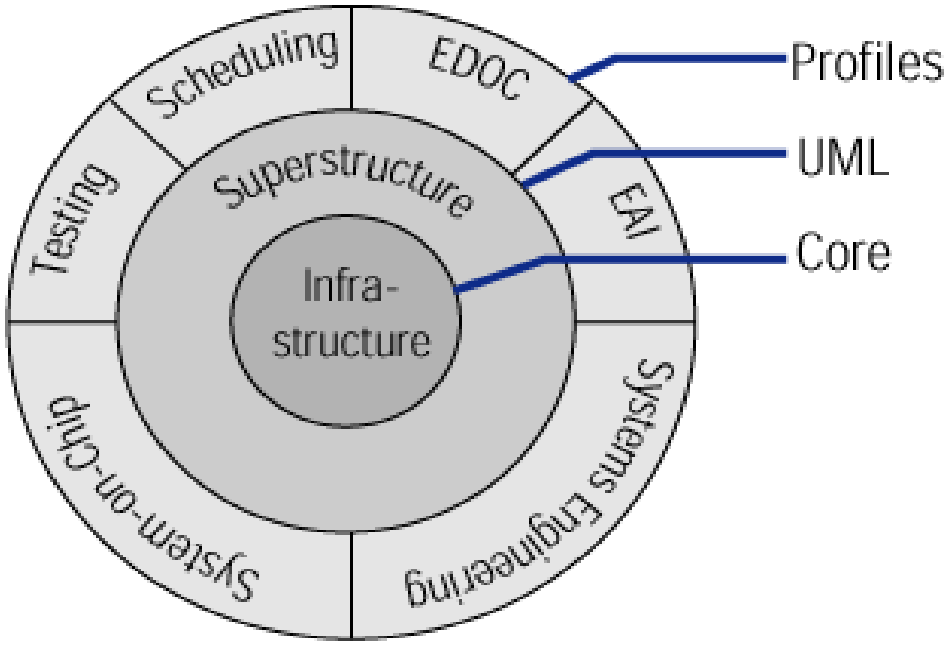
Internal Structure: Layers



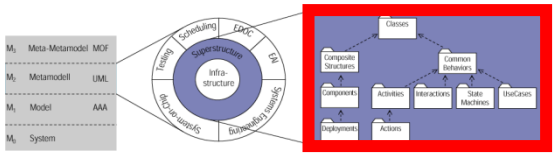


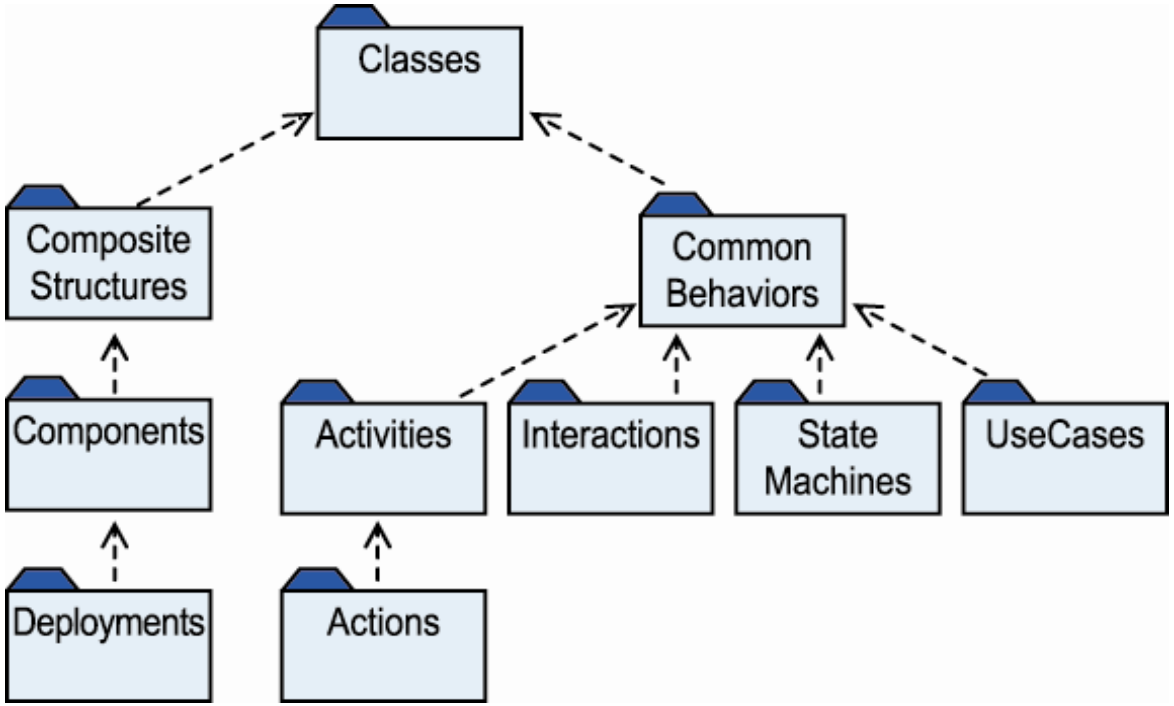
Internal Structure: Rings





Internal Structure: Packages





UML is not (only) object oriented

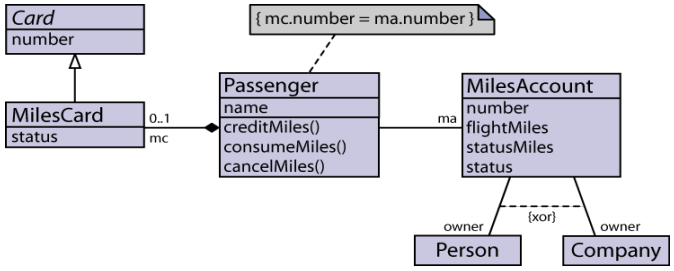
- A popular misconception about UML is that it is “object oriented” by heart – whatever that means.

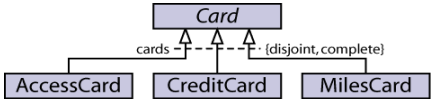
- It is true that
  - UML defines concepts like class and generalization;
  - UML is defined using (mainly) a set of class models;
  - UML 2 rediscovers the idea of behaviour embodied in objects.

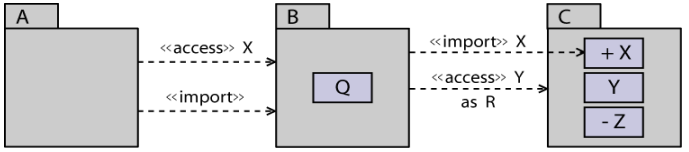
- However, UML 2
  - also encompasses many other concepts of non- or pre-OO origin (Activities, StateMachines, Interactions, CompositeStructure, …);
  - may be used in development projects completely independent of their implementation languages (Java, Cobol, Assembler, …);
  - is not tied to any language or language paradigm, neither by accident nor purpose.

Unified Modeling Language 2

Classes and packages







History and predecessors

- **Structured analysis and design**
  - Entity-Relationship (ER) diagrams (Chen 1976)
- **Semantic nets**
  - Conceptual structures in AI (Sowa 1984)
- **Object-oriented analysis and design**
  - Shlaer/Mellor (1988)
  - Coad/Yourdon (1990)
  - Wirfs-Brock/Wilkerson/Wiener (1990)
  - OMT (Rumbaugh 1991)
  - Booch (1991)
  - OOSE (Jacobson 1992)

Usage scenarios

- Classes and their relationships describe the vocabulary of a system.
  - **Analysis**: Ontology, taxonomy, data dictionary, …
  - **Design**: Static structure, patterns, …
  - **Implementation**: Code containers, database tables, …

- Classes may be used with different meaning in different software development phases.
  - meaning of generalizations varies with meaning of classes

	Analysis	Design	Implementation
Concept	√		×
Type		√	√
Set of objects		√	√
Code	×		√

Classes

- Classes describe a set of instances with common features (and semantics).
  - Classes induce types (representing a set of values).
  - Classes are namespaces (containing named elements).

- **Structural features** (are typed elements)
  - properties
    - commonly known as attributes
    - describe the structure or state of class instances
    - may have multiplicities (e.g. **1, 0..1, 0..*, *, 2..5**)

- **Behavioral features** (have formal parameters)
  - operations
    - services which may be called
    - need not be backed by a method, but may be implemented otherwise