Theory and Basic Principles of MDE

Jean Bézivin

JBezivin@gmail.com

Lecture 2: Theory and Basic Principles of MDE

The second course will present the basic principles of MDE from a conceptual and theoretical point of view. MDE uses typed graphs for most of the representation tasks and rule-based declarative transformations for most of the manipulation tasks. It relies essentially on the concepts of metamodel and model transformation. Sometimes called Modelware, MDE is thus a graph-based and transformation based technology. A technical space will be defined and used to compare similarities and differences with other technologies like Grammarware. Metamodels provide some sort of typing system on top of the models themselves. The essence of MDE is however most related to the unification of abstract models. Terminal models, metamodels and metametamodels are the most important categories of abstract models. Different operations like storage, retrieval, transformation, etc. may be applied on all abstract models. One of the important characteristics of MDE is to represent systems by models. This relation of representation which is central to all modeling activities will be analyzed and discussed.

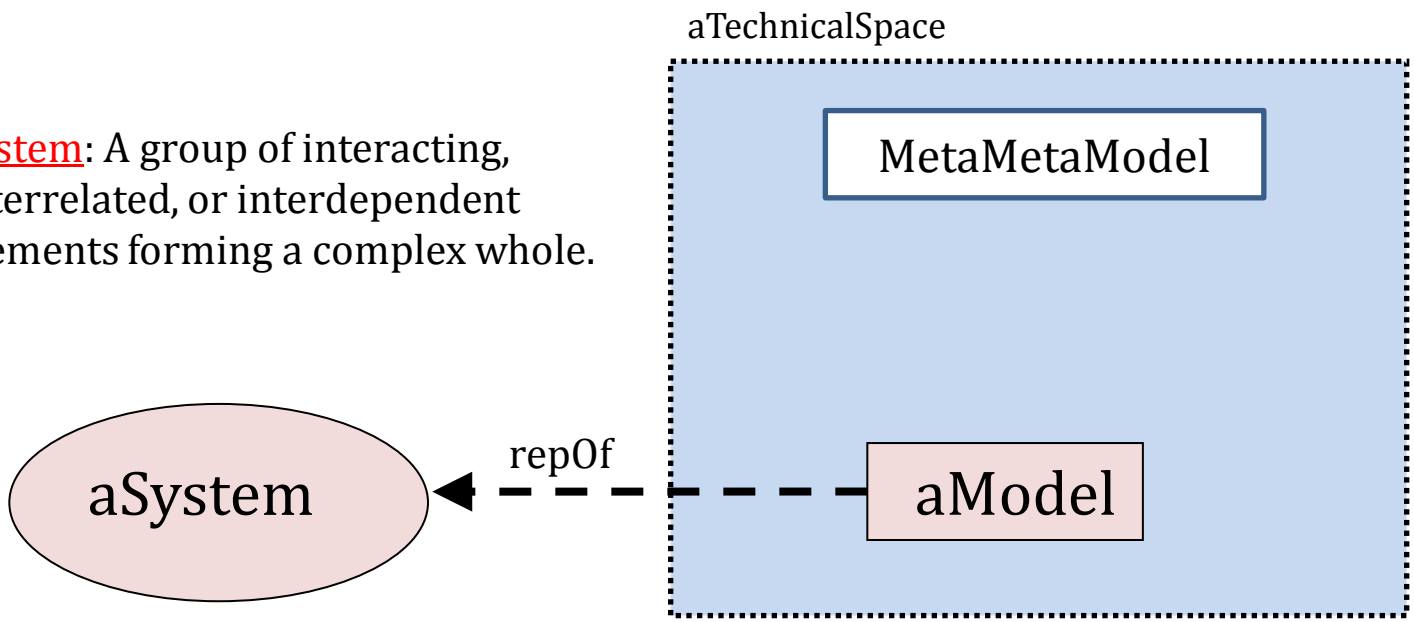
If metamodels provide a strong support for representation, they have however not the power to reason or more generally to act upon these representations. Of course it is always possible to use the MOF or the general purpose language to put these metamodel-based representations to work but this solution has many drawbacks that will be outlined. A more constructive MDE approach usually suggested consists in using model transformations as the core and main operation of the MDE. The evolution will be discussed; particularly in the case when the transformation language is declarative and rule-based. Additional advantages may also be reaped when the transformation program may be considered itself as a model. In this case it is much easier to deal with domain specific transformation languages, providing maximum impact to the regular MDE architecture based on metamodels and transformations.

- Advanced concepts
  - Two important additional concepts
    - Technical Spaces
    - DSL
  - Other concepts
    - Metamodels
    - Metametamodels
    - Multimodels
    - Transformations
    - Correspondence Models
    - Megamodels
    - etc.

Technical Spaces

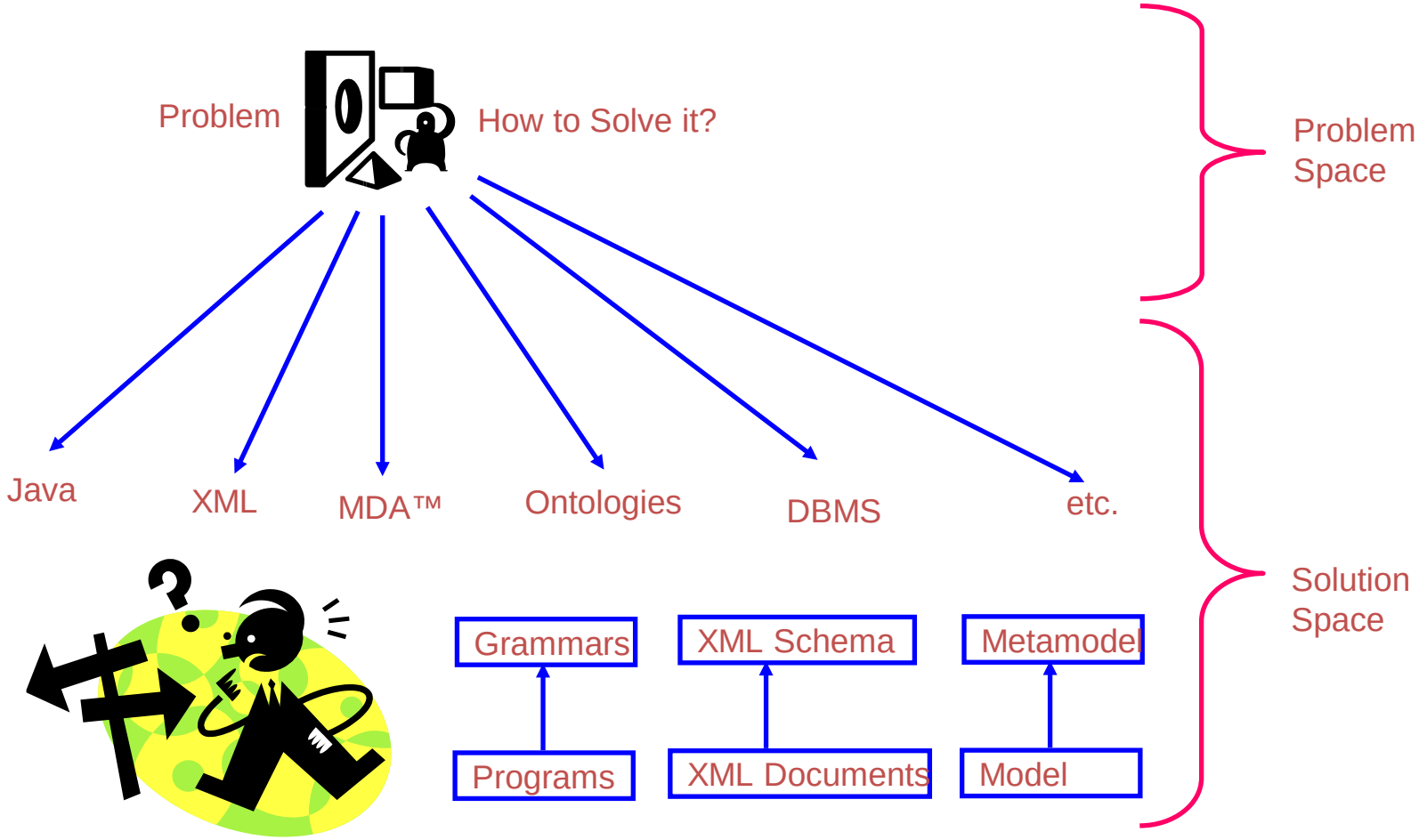
Technical Space: A representation system for models and a set of technical solutions to handle them. A framework usually based on some algebraic structures (relational tuples, trees, graphs, hypergraphs, etc.)

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

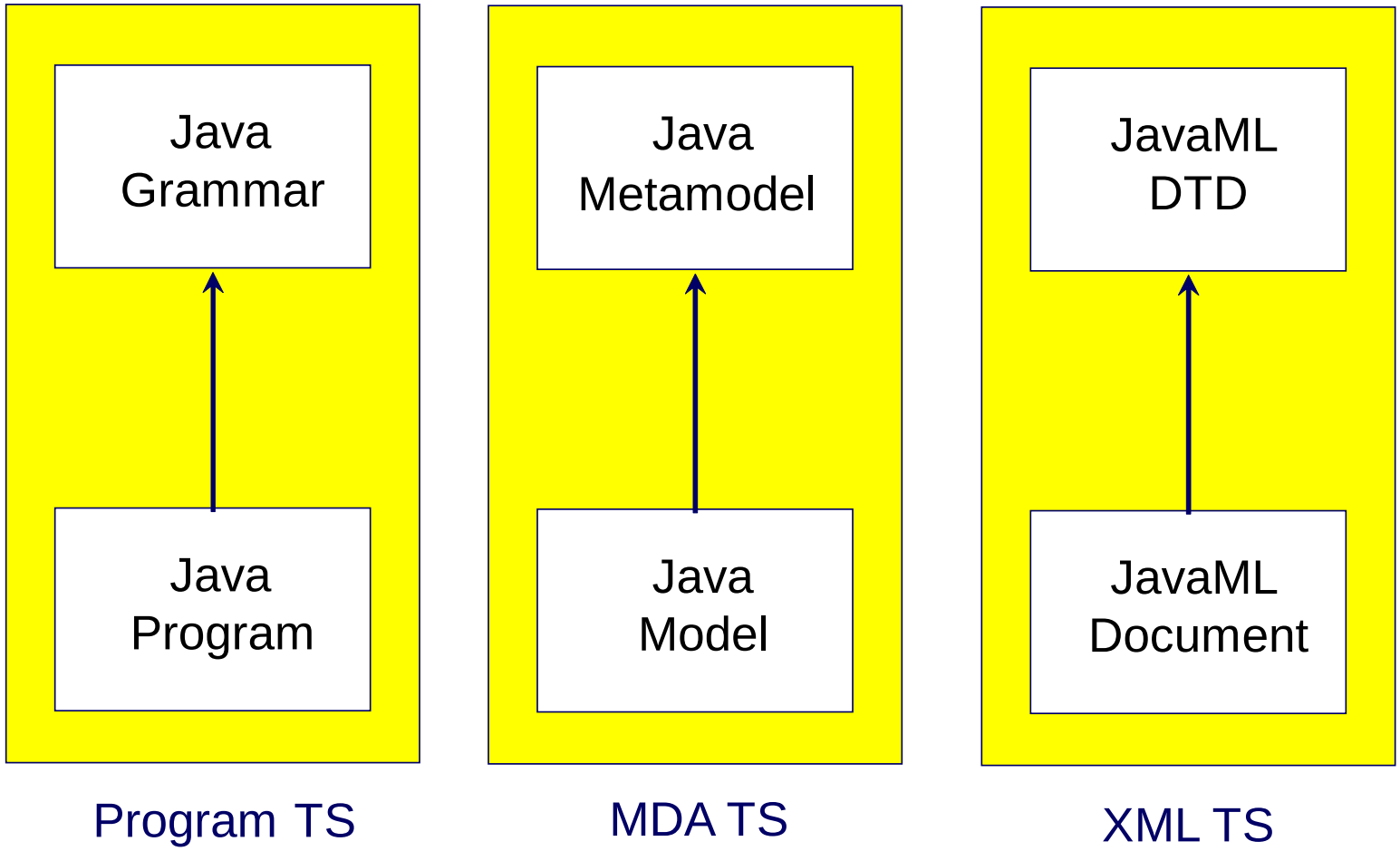


Model: An abstract representation of a system created for a specific purpose.

Structuring the solution space



Three representations for the same program



Three representations for the same program

```
1  import java.applet.*;
2  import java.awt.*;
3
4  public class FirstApplet extends Applet {
5    public void paint(Graphics g) {
6      g.drawString("FirstApplet", 25, 50);
7    }
8  }
```

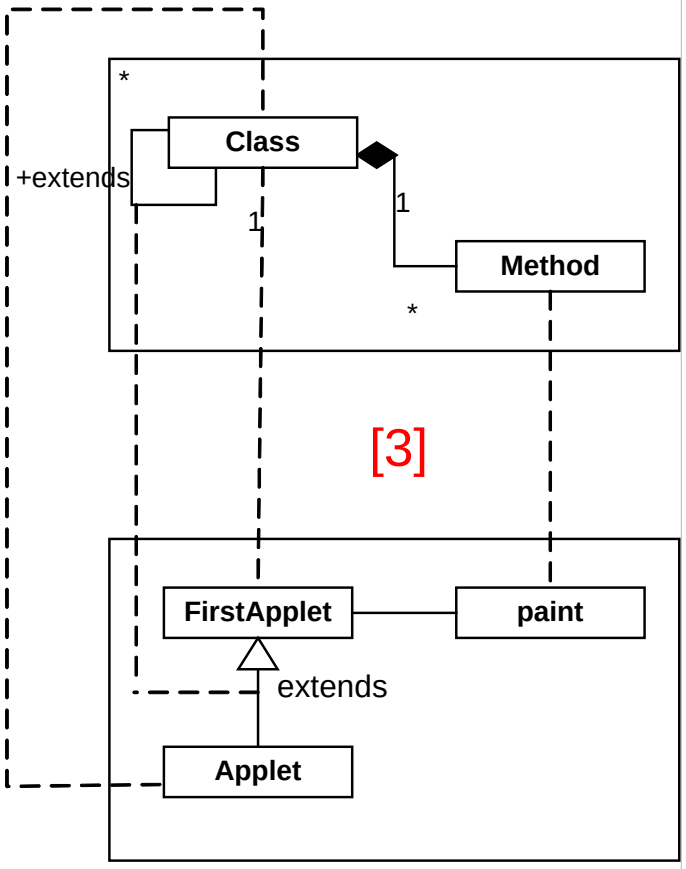
[1]

Java source code

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE java-source-program SYSTEM "java-ml.dtd">
4 <java-source-program name="FirstApplet.java">
5   <import module="java.applet.*"/>
6   <import module="java.awt.*"/>
7   <class name="FirstApplet" visibility="public">
8         <superclass class="Applet"/>
9         <method name="paint" visibility="public" id="meth-15">
10                  <type name="void" primitive="true"/>
11                  <formal-arguments>
12                        <formal-argument name="g" id="frmarg-13">
13                        <type name="Graphics"/></formal-argument>
14                  </formal-arguments> …….
```

[2]

JavaML document



Each of these representations may be more convenient to perform some operation on the program.

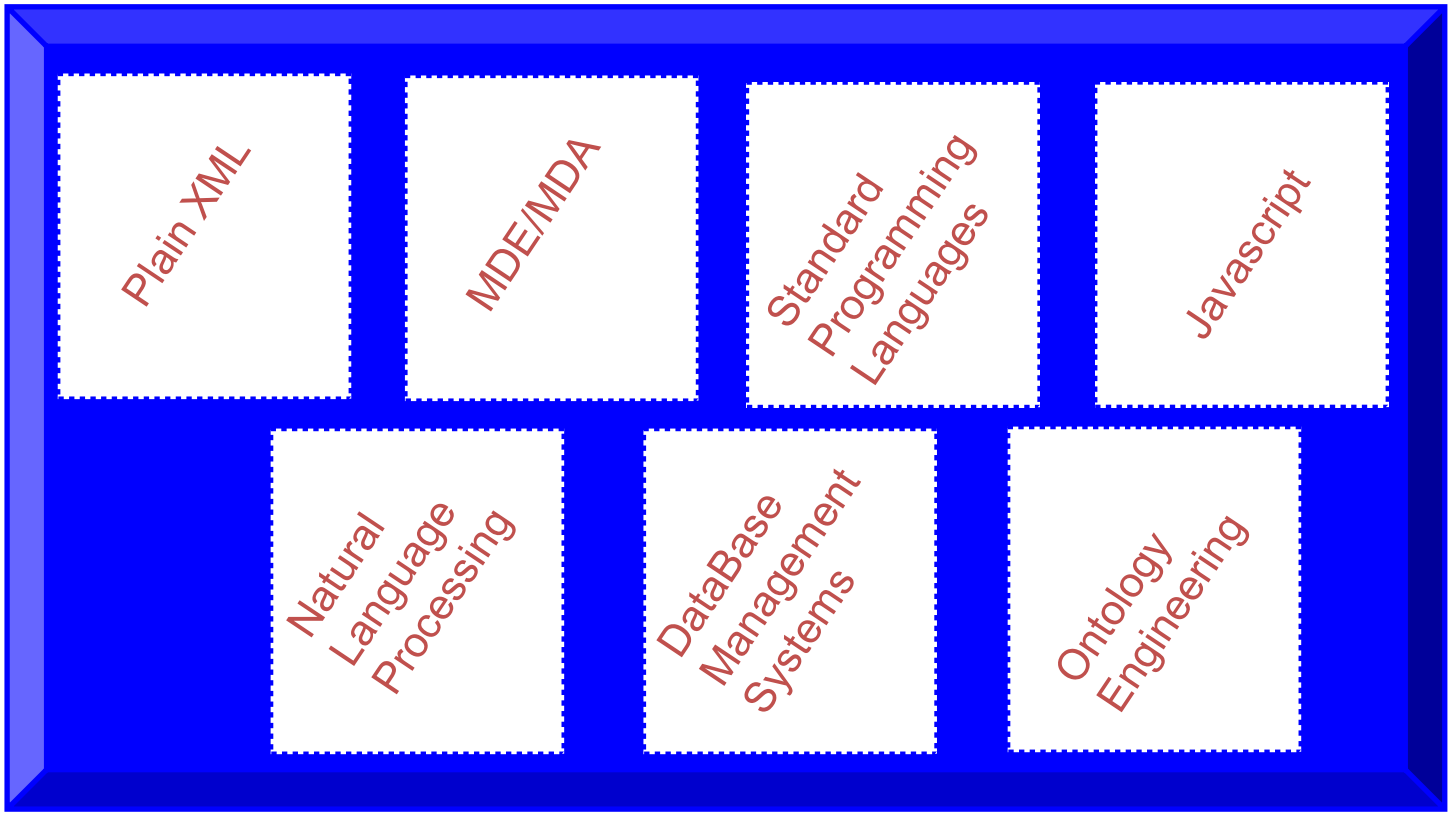
Multiple possible representation systems

- A Technological Space (abbr. TS) employs:
  - A set of concepts
  - A working context
  - A shared knowledge and know how
  - A set of associated tools
- It is usually related to a given community with an established expertise, research problems and known results.

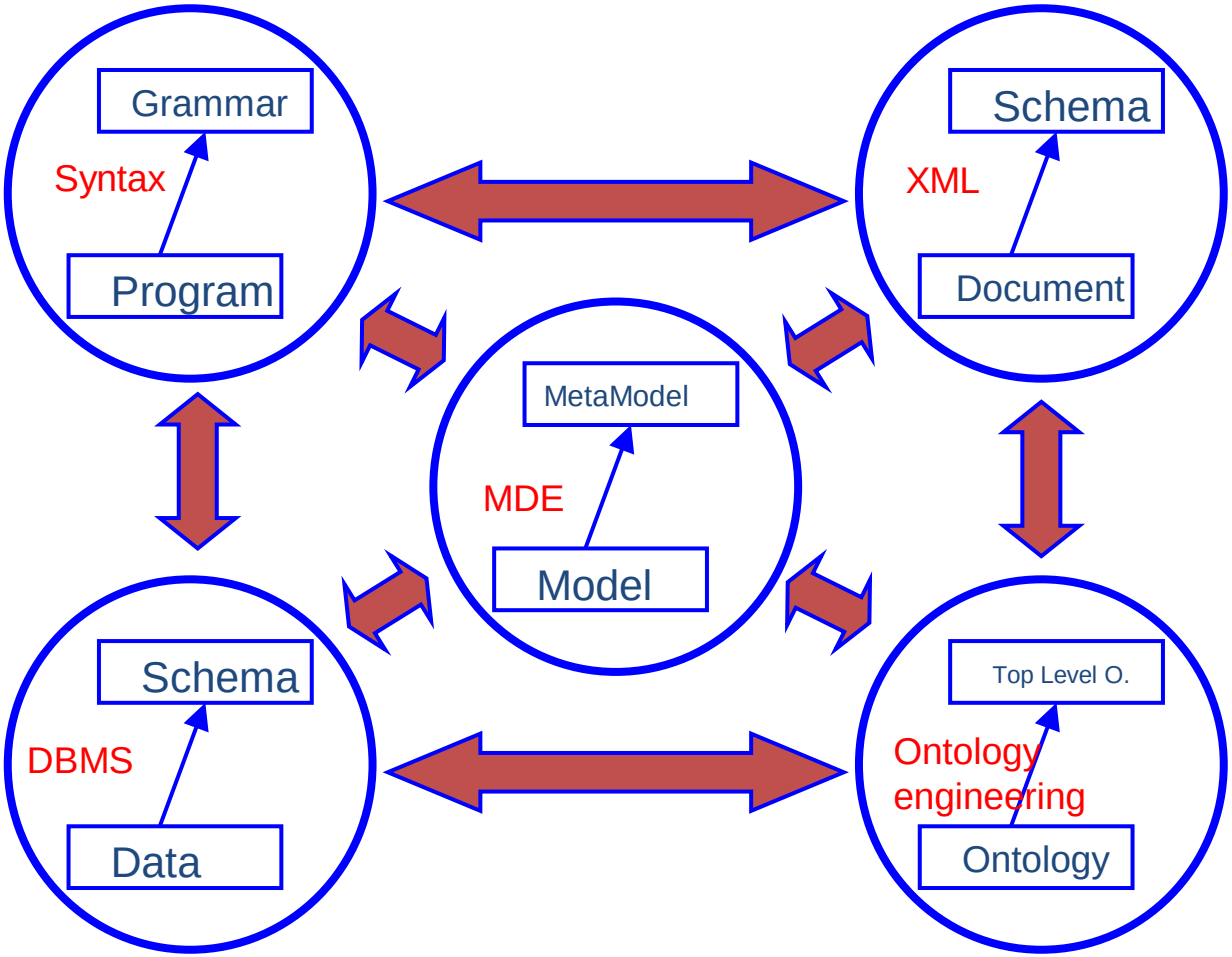
  Exemple: the Petri Net community, the Coq community, the Prolog community.

But it is mainly based on some common representation system.

Classical TSs (examples)

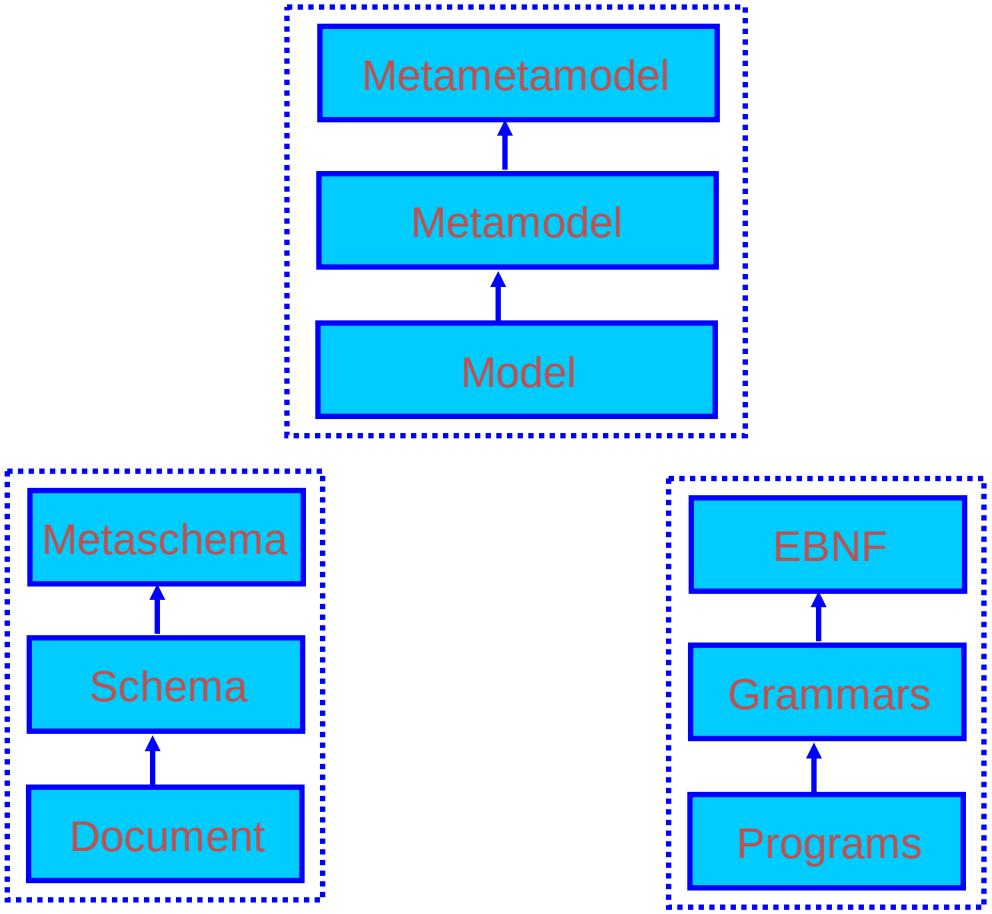


Alternative TSs



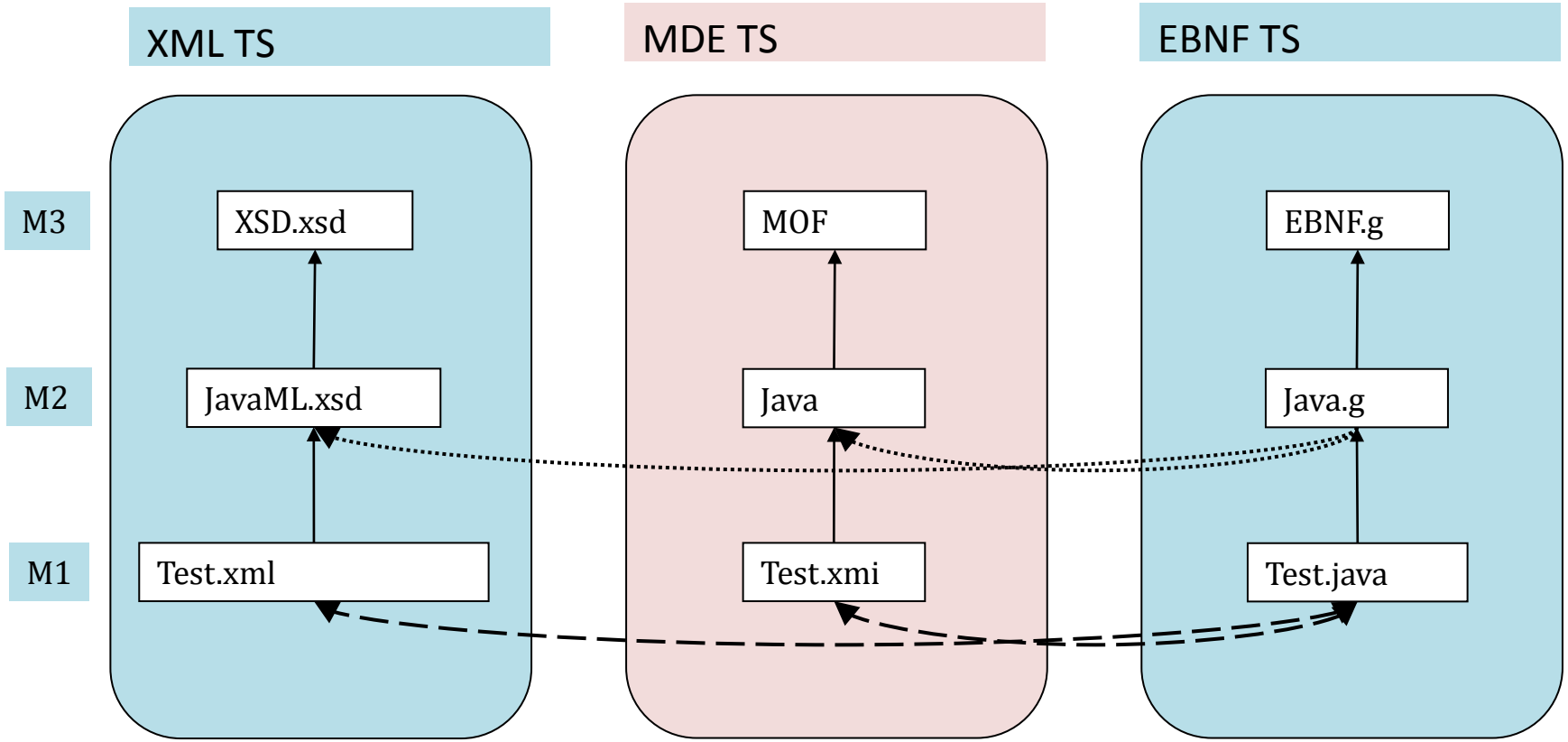
- A technological space is organized around a set of concepts
- Spaces are connected via bridges
- Spaces are similarly organized and operationally interoperable.

The 3-level conjecture

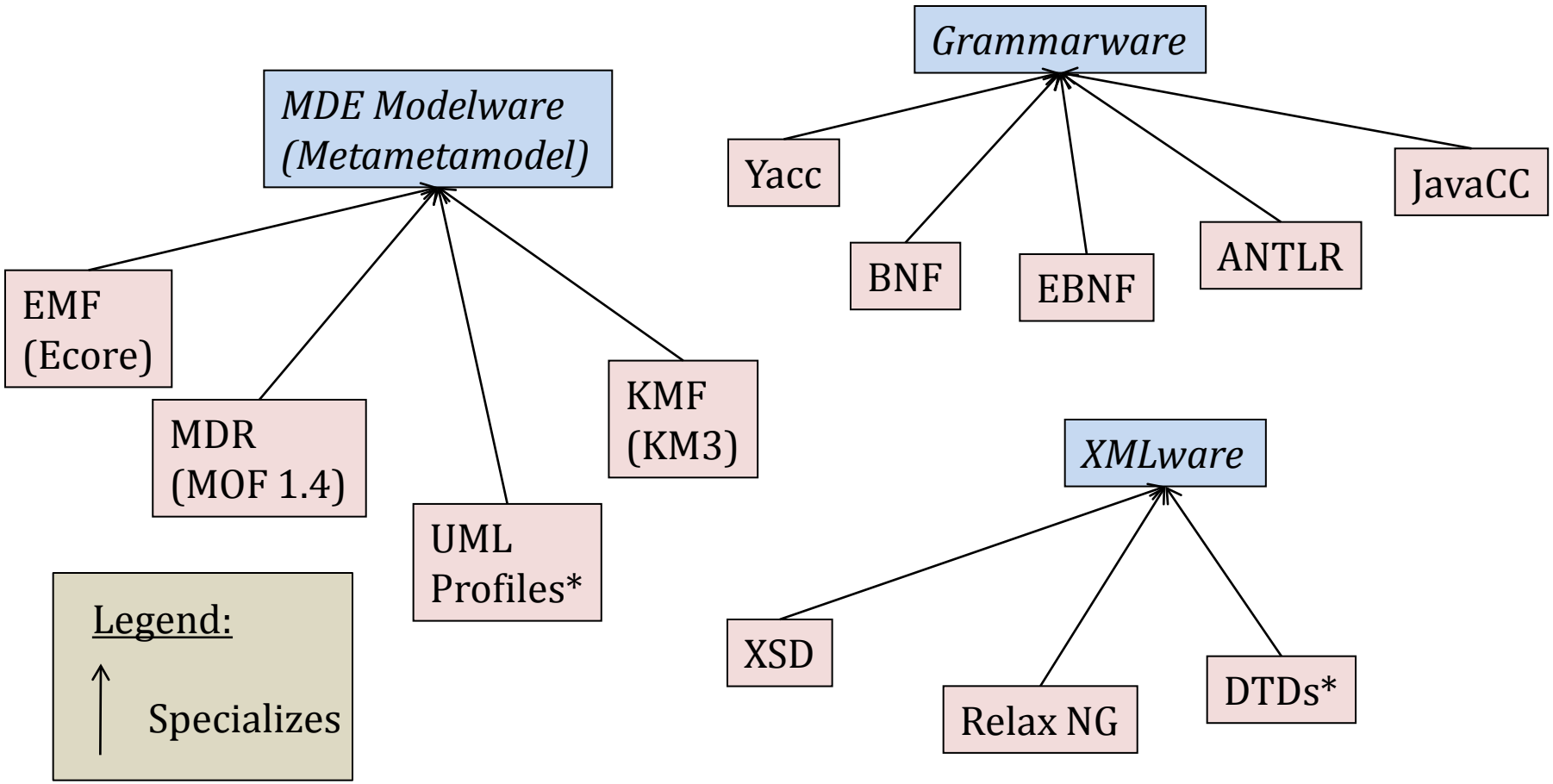


- Each space may be viewed as organized in three engineering "metalevels"
- These levels have nothing to do with abstraction levels
- If we can assume this organization, this will make much easier the conceptual and operational bridging between different technical spaces

Projections betwen EBNF, MDE and XML



Technical Space Variants – Some Examples



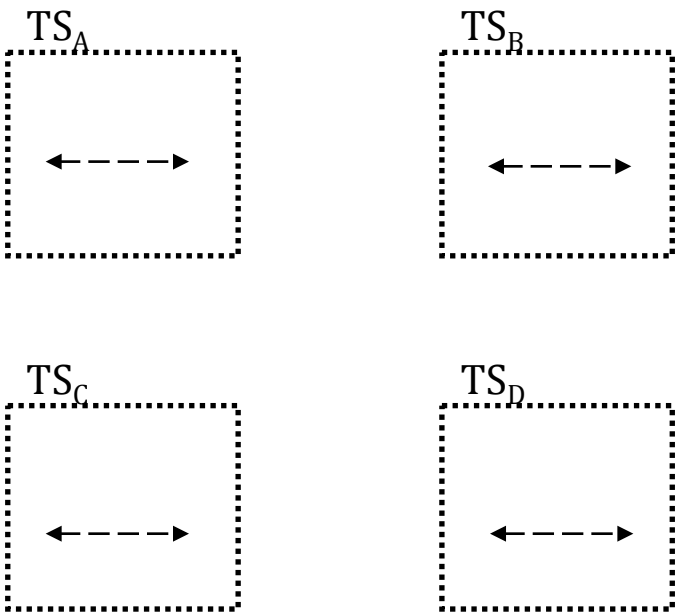
*Not 3-level based

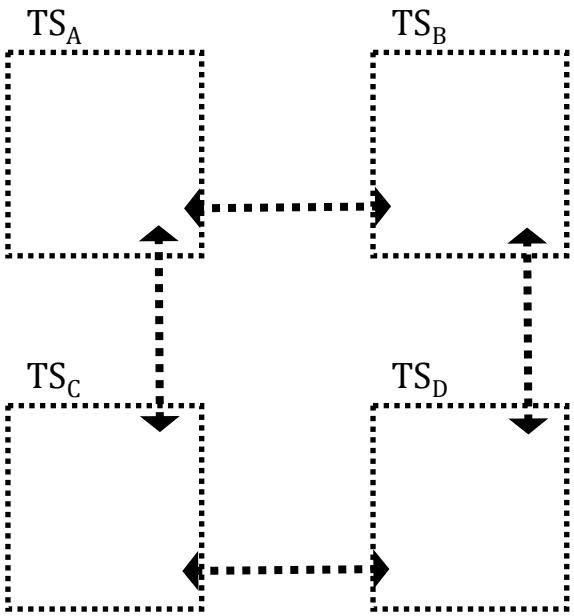
λ-Models and MDE-Models

- Technical Spaces and their representation systems
  - XML uses a specific kind of trees (XML trees),
  - RDBMS uses the relational model,
  - Ontology uses a specific kind of graphs,
  - EBNF (or Grammarware) uses formal grammars and Abstract Syntax Trees,
  - Model Driven Engineering uses a specific kind of graphs.
- The λ-model notation, where λ is the Technical Space of the model:
  - XML-models,
  - RDBMS-models,
  - EBNF-models,
  - Ontology-models,
  - MDE-models.
- Each TS has its own engineering advantages and drawbacks.

- **Everything is a model**
  - A λ-model
  - λ meaning the specific TS
  - An XML document is an XML-model
  - A Java source program is a Java-model
  - An UML model is a MDA-model
  - etc.
- Each TS is rooted in a metametamodel defining its representation scheme (representation ontology)
- Each is supposed to be organized in a 3-level architecture
- **Our default TS here is the MDE TS, but we can't ignore the other TSs**

Intra and Inter-TS transformations





- Most Technical Spaces have internal model transformation solutions:
  - XSLT and XQuery for XML,
  - Compilers & Program transformation for EBNF,
  - QVT or ATL for Modelware,
  - etc.

- But there is also a need for inter-TS transformations.
- Many transformations may be expressed as a <u>combination of intra and inter-TS transformations:</u>

$$t = f \circ g \circ h$$

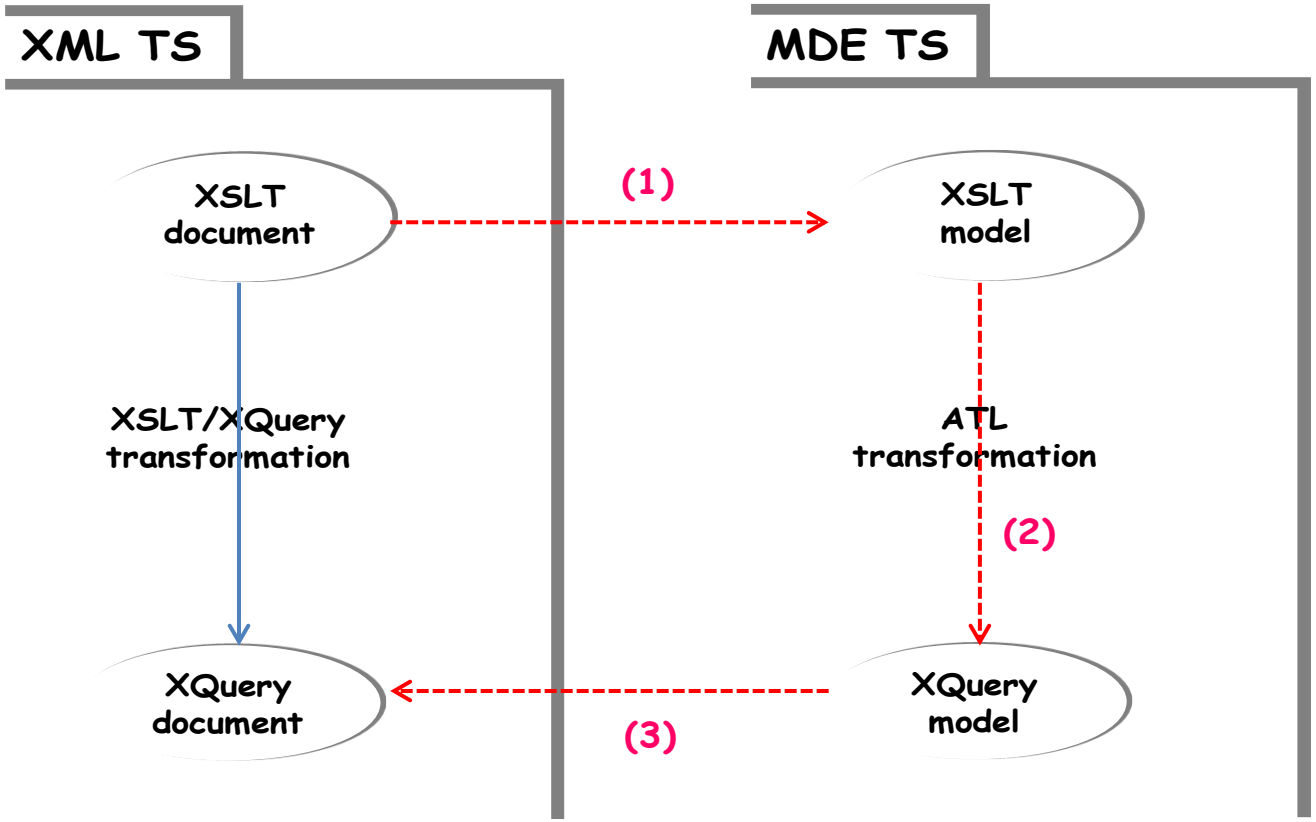
Communications between technical spaces

Any software artifact can be **injected** into a model

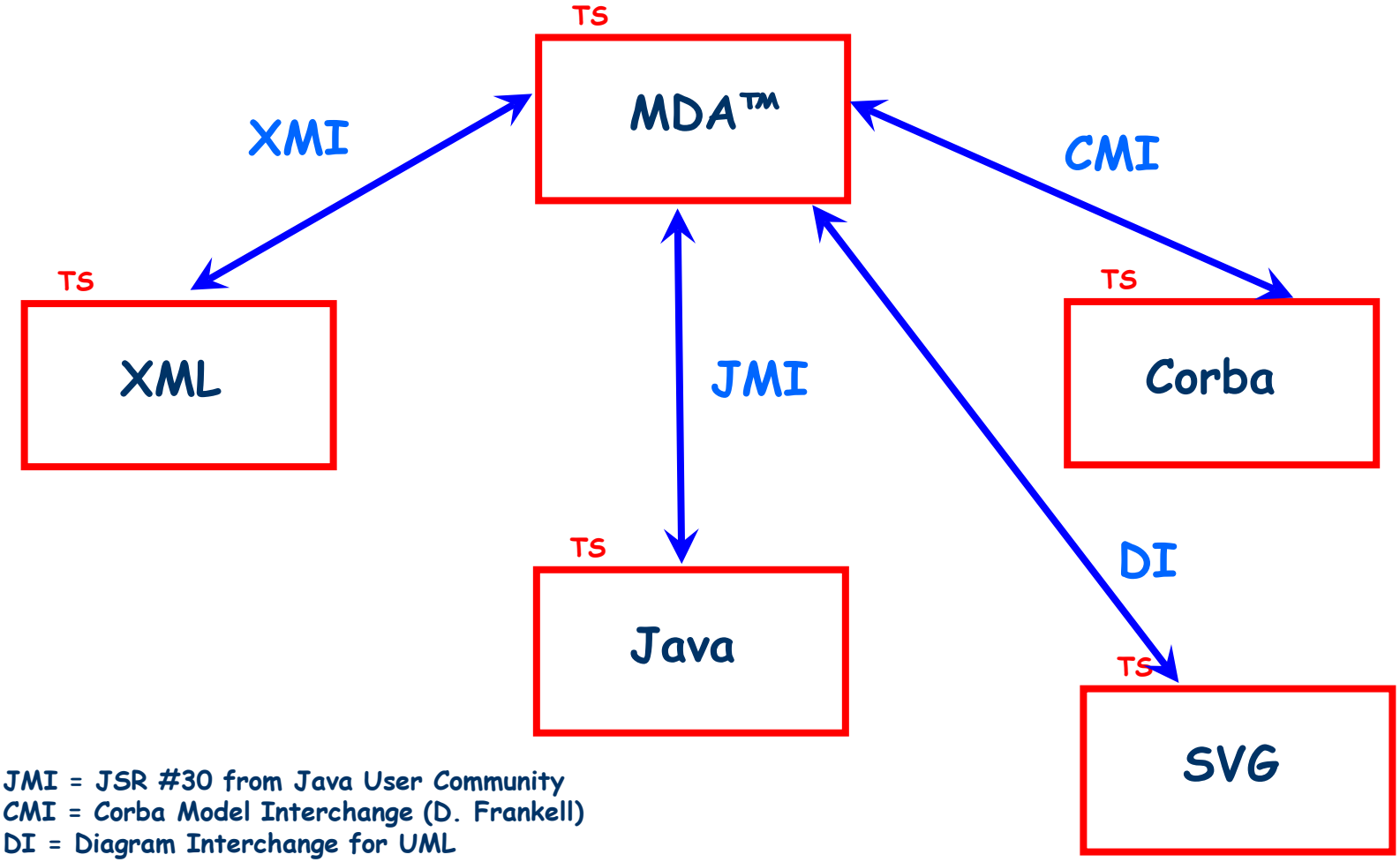
Any model can be **extracted** to a software artifact

Cost of solving a problem inside a given TS vs. exporting it to another TS, solving it and importing back the solution.

Economic equation



Collaborations between TS: Projections



JMI = JSR #30 from Java User Community
CMI = Corba Model Interchange (D. Frankell)
DI = Diagram Interchange for UML

Comparing spaces

	XML	MDE	Grammars	Ontologies
Executability	Poor	Poor	Excellent	Poor
Aspect separation	Good	Excellent	Poor	Fair
Formalization	Poor	Poor	Excellent	Fair
Specialization	Fair	Good	Poor	Fair
Modularity	Good	Good	Poor	Poor
Traceability	Good	Fair	Poor	Fair
Transformability	Excellent	Fair	Fair	Fair

(NB: marks are only indicative)

Domain Specific Languages

- A domain-specific language (DSL) is a programming or modeling language dedicated to a particular domain, a particular problem, and/or a particular representation technique.

> ***"DSLs are not a new idea. It is exactly what people called POL (Problem Oriented Languages) 50 years ago."***
>
> **Alan Kay**

- DSLs or Special-purpose languages are becoming popular.
  - Several modeling languages are DSLs, some of them called *specification languages*.

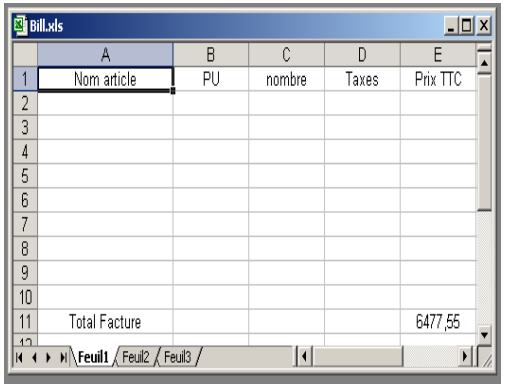
DSLs vs. General Purpose Languages

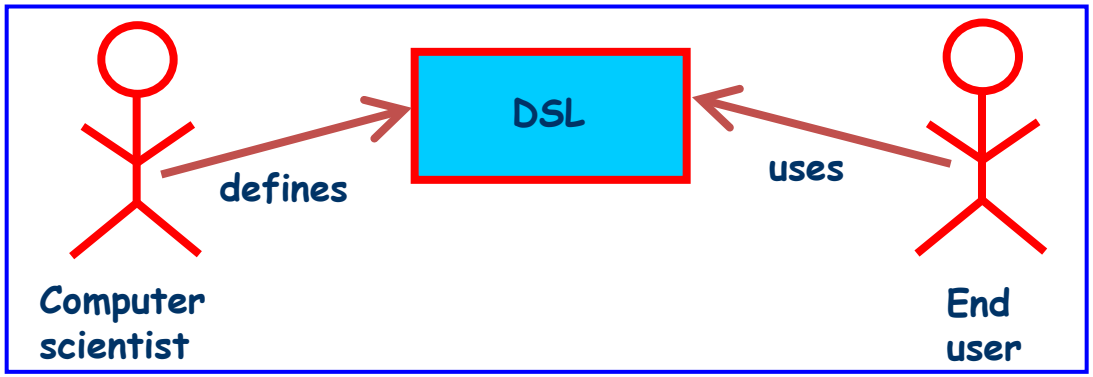
- The opposite of a DSL is
  - a *general-purpose programming language*, such as Pascal, C, Java or Python,
  - a *general-purpose modeling language* such as the Unified Modeling Language (UML).
- Language-Oriented Programming considers the creation of special-purpose languages for expressing problems a standard part of the problem solving process.
  - Creating a domain-specific language (with software to support it) for a repetitive type of problem.
  - The cost of creating such a language should be light.

Examples of DSLs

- HTML for describing Web pages
- Verilog and VHDL for describing hardware
- Mata for matrix programming
- SQL for relational database queries
- YACC for creating parsers
- Csound for sound and music synthesis
- Graphviz for graph layout
- R for statistical computing
- Awk for Unix-based data transformation
- Make for software configuration
- Postscript for printer page layout
- Excel for basic accounting and spreadsheet calculation
- etc.

A very successful DSL (Visicalc, Excel)





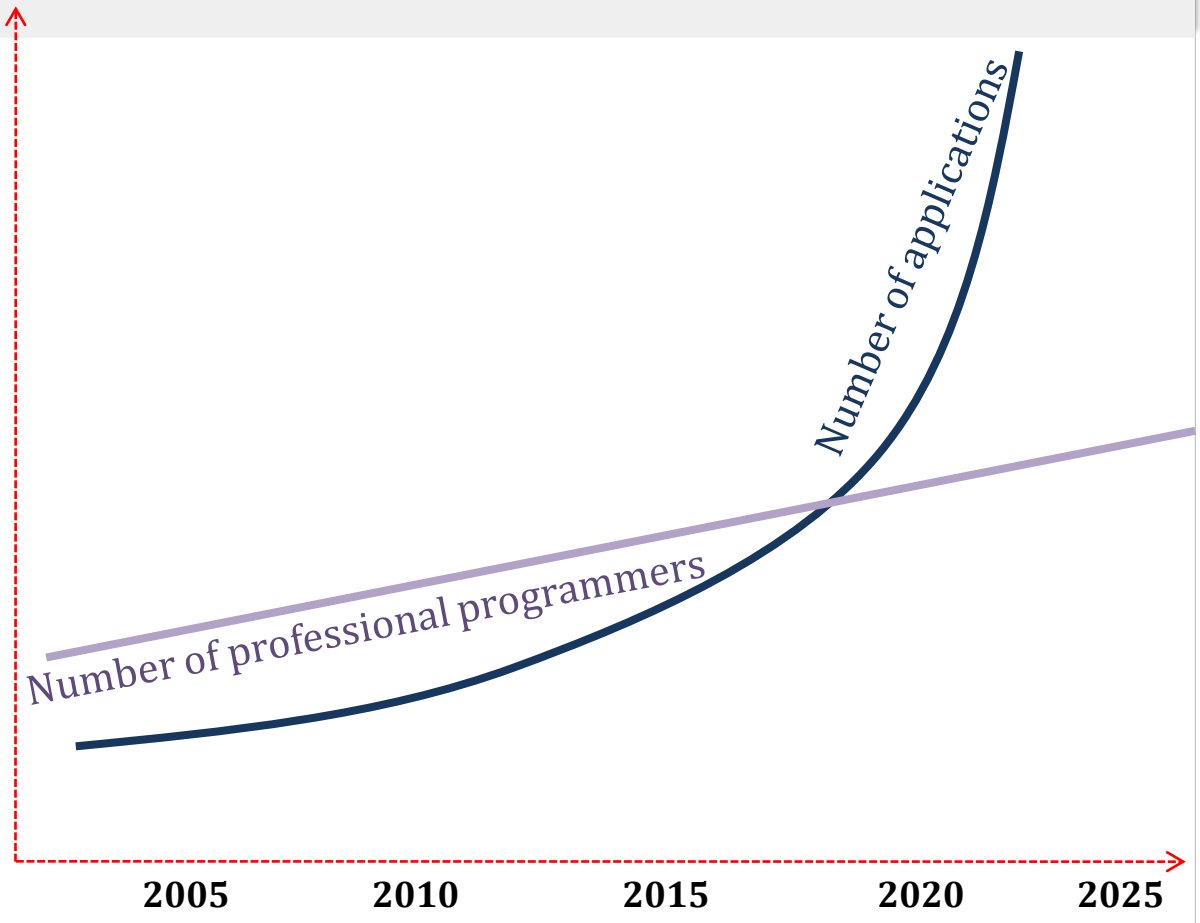
Initial domain = accounting

Why Johnny can't program?

❑ What is "programming"?

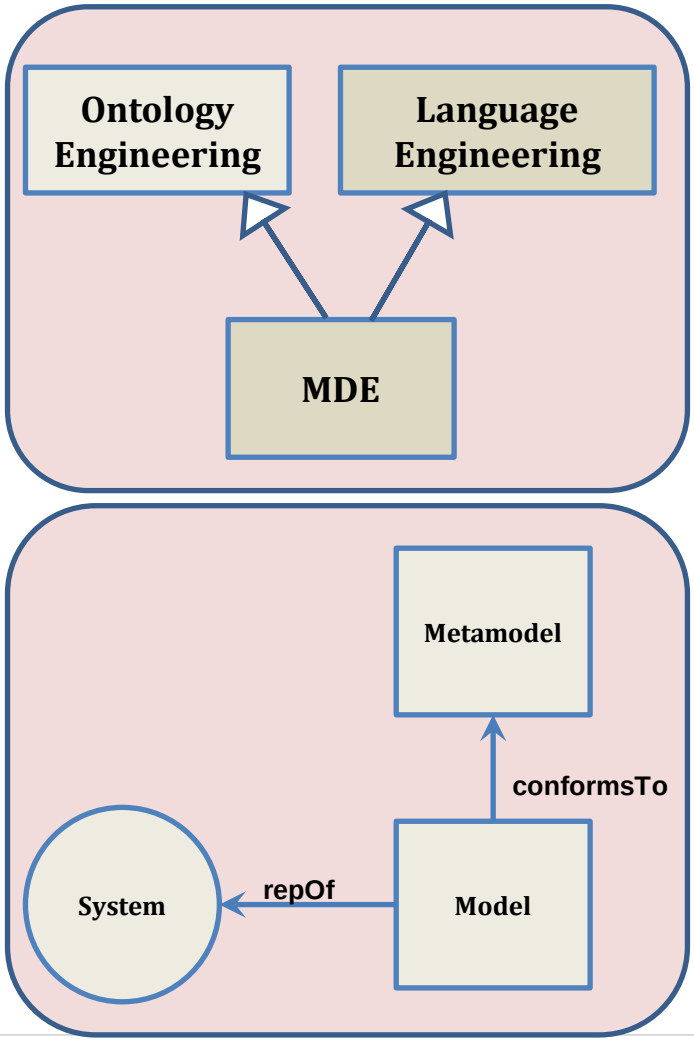
- The question really isn't "Why Johnny *can't* program" but rather "Why Johnny *won't* or *doesn't choose to* program".
- "Fill-in-the-forms and drop downs" and direct manipulation systems require less learning, testing of results, and debugging. There are fewer chances for error because of the constrained input choices. Rapid viewing of the results (in the domain vocabulary) makes it easier to make simple changes and see what happens.
- Programming languages like Assembler, FORTRAN, C, and Java are least likely to be well accepted by the masses.

The impossible equation



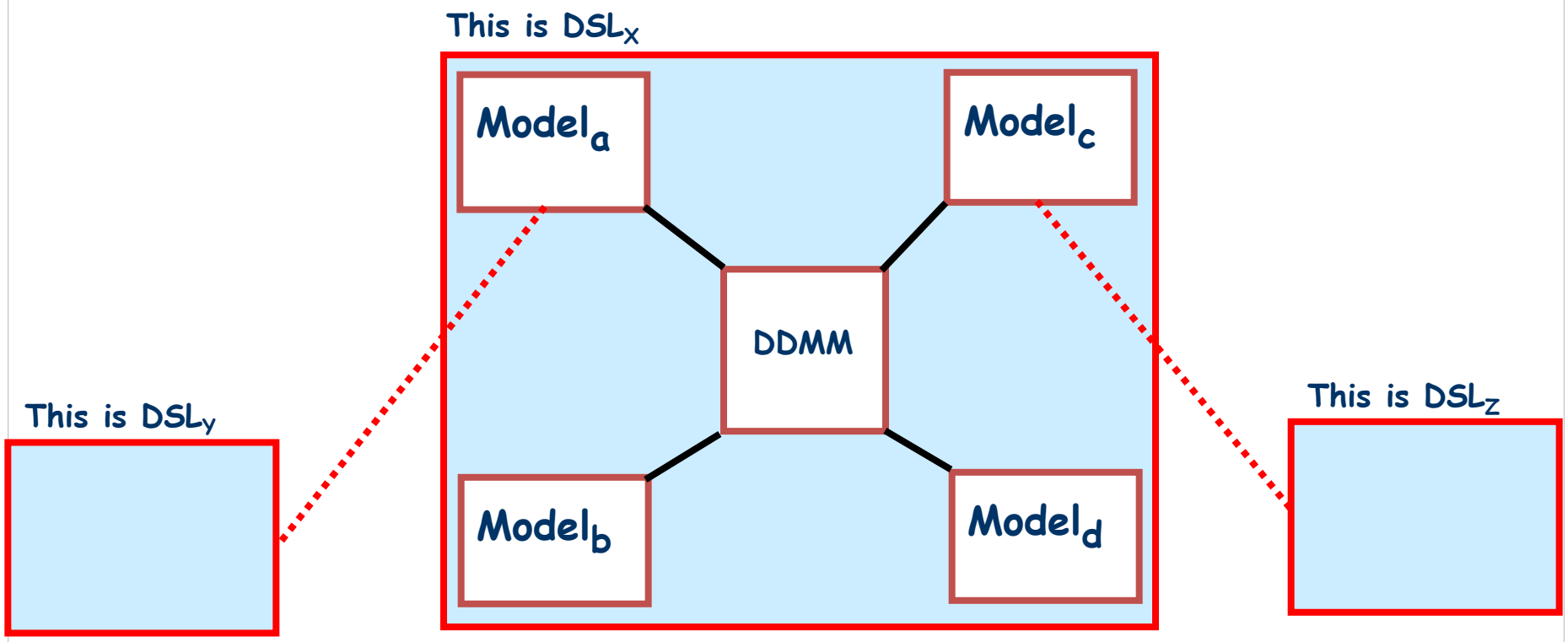
***USA**:*
90 Millions computer users;
50 Millions Spreadsheet & DB users;
12 Millions self described programmers;
3 Millions professional programmers;

DSLs and MDE



- What is the relation between DSLs and MDE?
- Both have a focus on domains
- MDE is a branch of language engineering, very close to Language Oriented Programming.
- In MDE, a language is created from a precise metamodel
- More precisely a language is composed of:
  - Abstract syntax
  - Concrete Syntax
  - Semantics
- In the MDE TS, these may mostly be represented by modeling artifacts

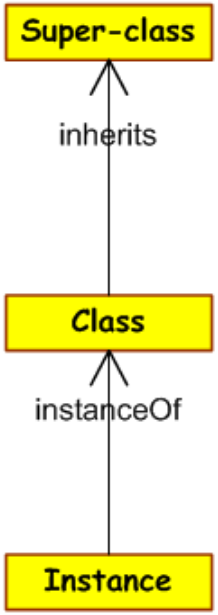
Relations between MDE and DSLs

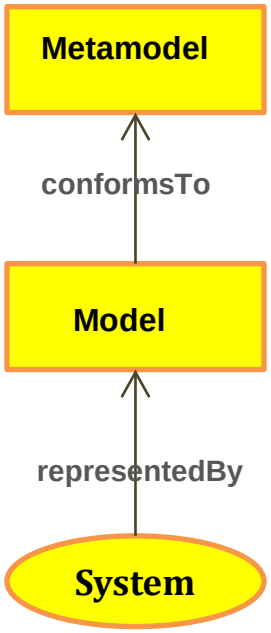


<u>Definition:</u> A DSL is a coordinated set of models
DDMM = Domain Definition MetaModel
Many models are mappings to other DSLs (semantic, concrete syntax, etc.)

Two TSs: MDE vs. OOT

The two central relations in object-oriented technology





The two central relations in model-driven engineering

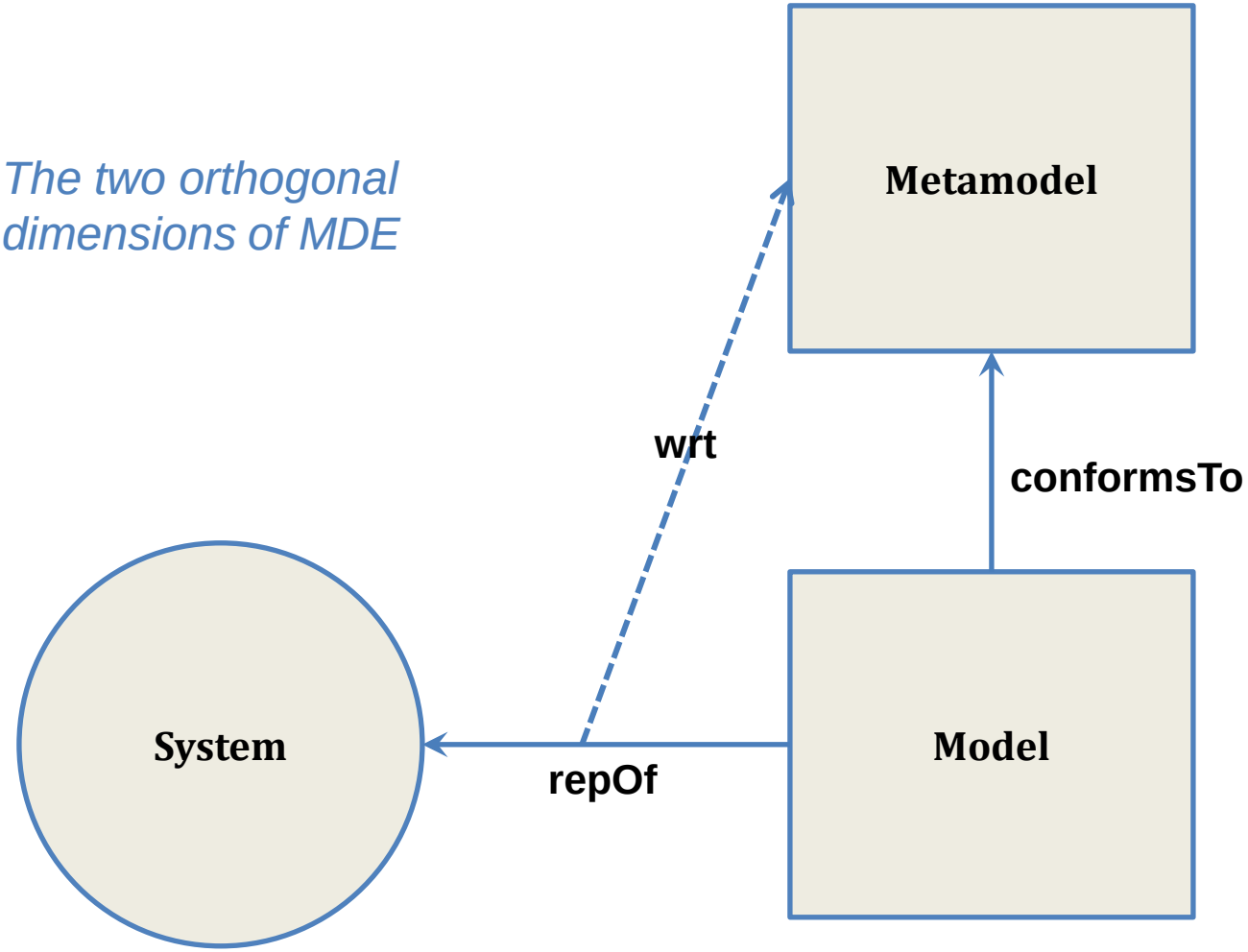
The set of concepts and central relations in model-driven engineering and object-oriented technology are basically different.

Caution: Trying to use one set of relations, as a central interpretation, in the wrong context, may lead to serious problems :

Example 1: A model being "an instance of" a meta-model
Example 2: A UML profile "inheriting from" the UML meta-model
etc.

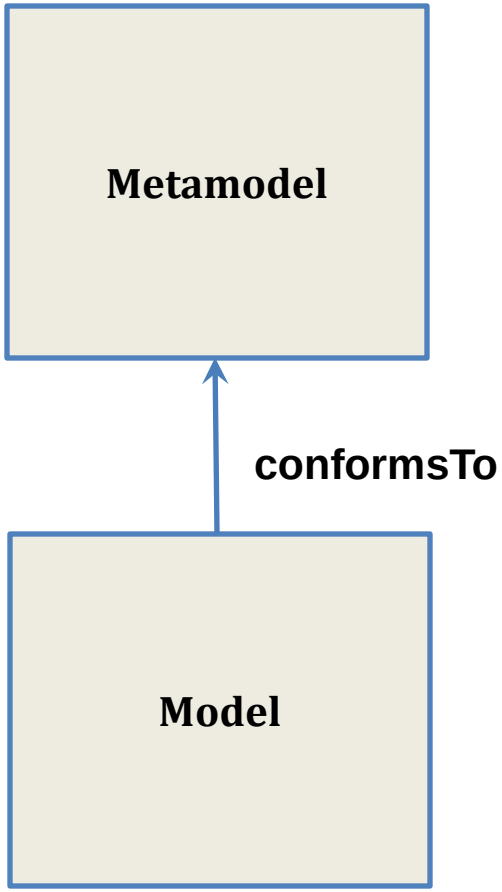
Representation and Conformance

The two orthogonal dimensions of MDE



Metamodels

A metamodel is also a graph. It is a simplified ontology, i.e. a set of concepts and relations between these concepts (bipartite graph).



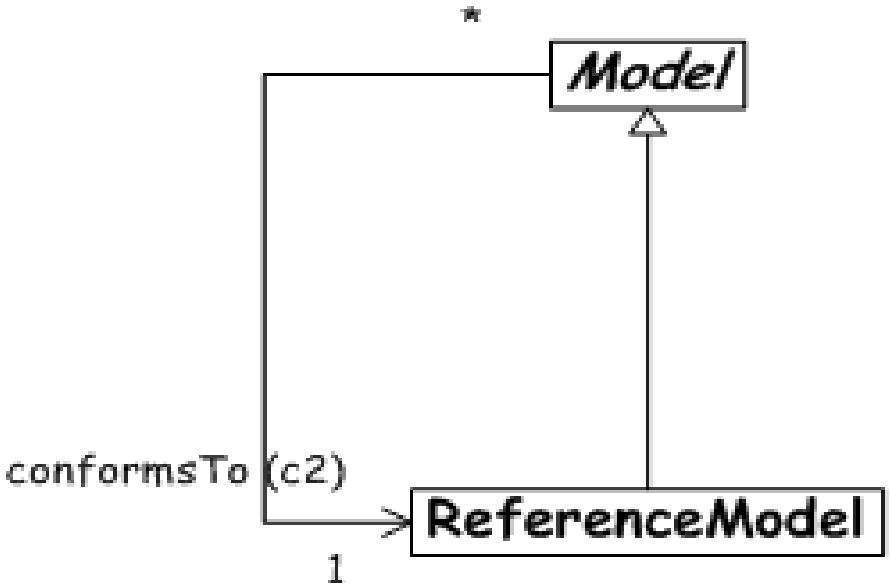
A model is a graph composed of elements (nodes and edges). Each such element corresponds to a concept in the metamodel.

Structural definition of a MDE-model

- **<u>Definition 1</u>.** A directed multigraph $G = (N_G, E_G, \Gamma_G)$ consists of a set of distinct nodes N_G, a set of edges E_G and a mapping function $\Gamma_{G:}: E_G \rightarrow N_G \times N_G$

- **<u>Definition 2</u>.** A model $M = (G, \omega, \mu)$ is a triple where:
  - **$G = (N_G, E_G, \Gamma_G)$ is a directed multigraph**
  - **ω is itself a model, called the <u>reference model</u> of M, associated to a graph $G_\omega = (N_\omega, E_\omega, \Gamma_\omega)$**
  - **$\mu: N_G \cup E_G \rightarrow N_\omega$ is a function associating elements (nodes and edges) of G to nodes of G_ω (metaElements)**

The central role of metamodels



Each model conforms to its reference model (metamodel), but the metamodel is a model.

Definitions

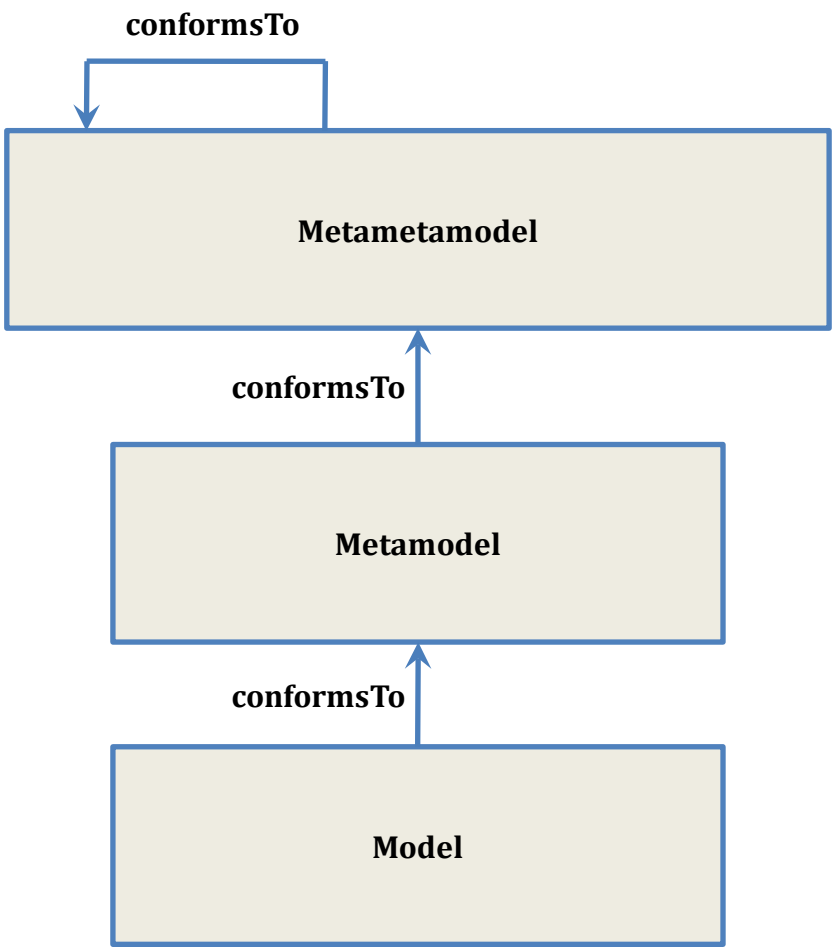
- In most technologies, a three level engineering organization has been chosen (see technical spaces later).

- In MDE, the following is assumed:

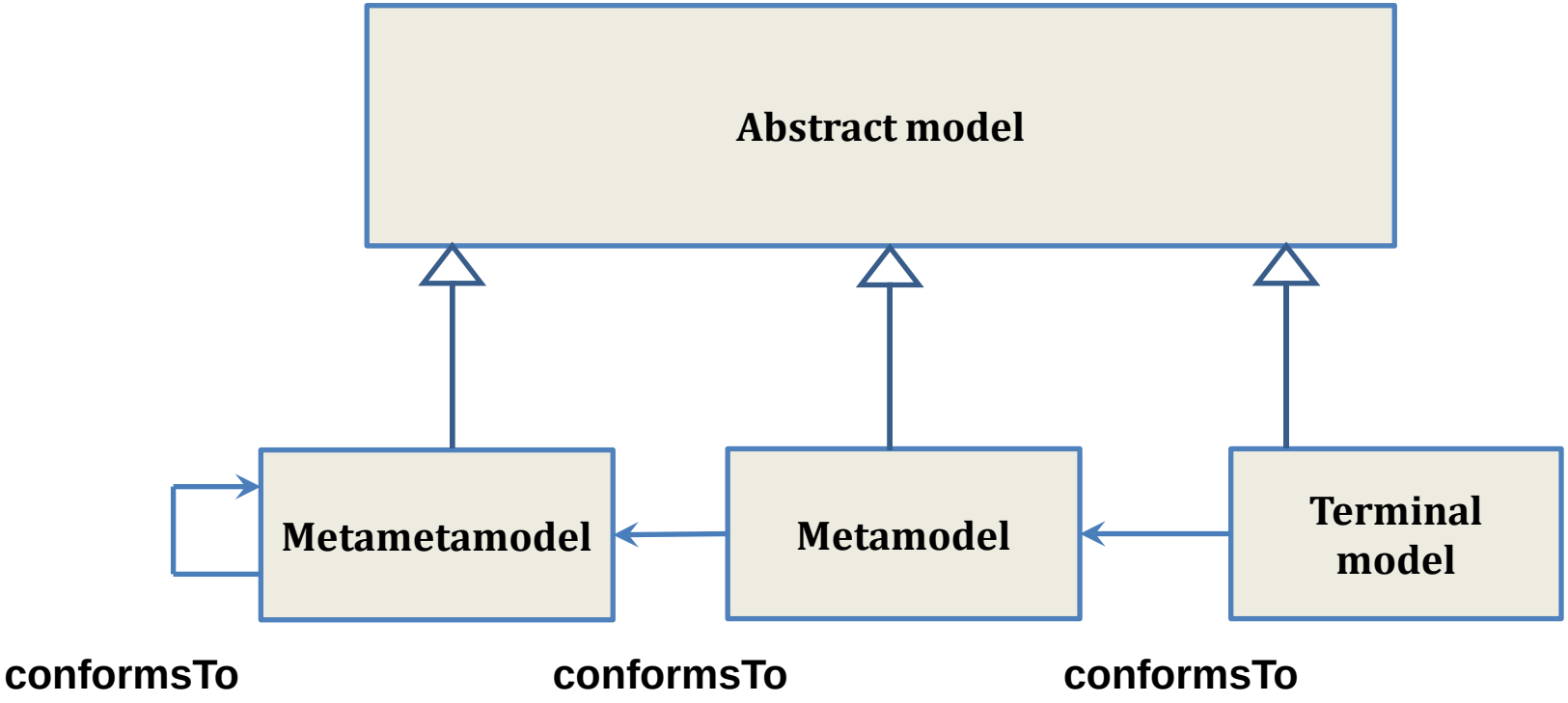
  - **Definition 3.** A metametamodel is a model that is its own reference model (i.e. it conforms to itself).

  - **Definition 4.** A metamodel is a model such that its reference model is a metametamodel.

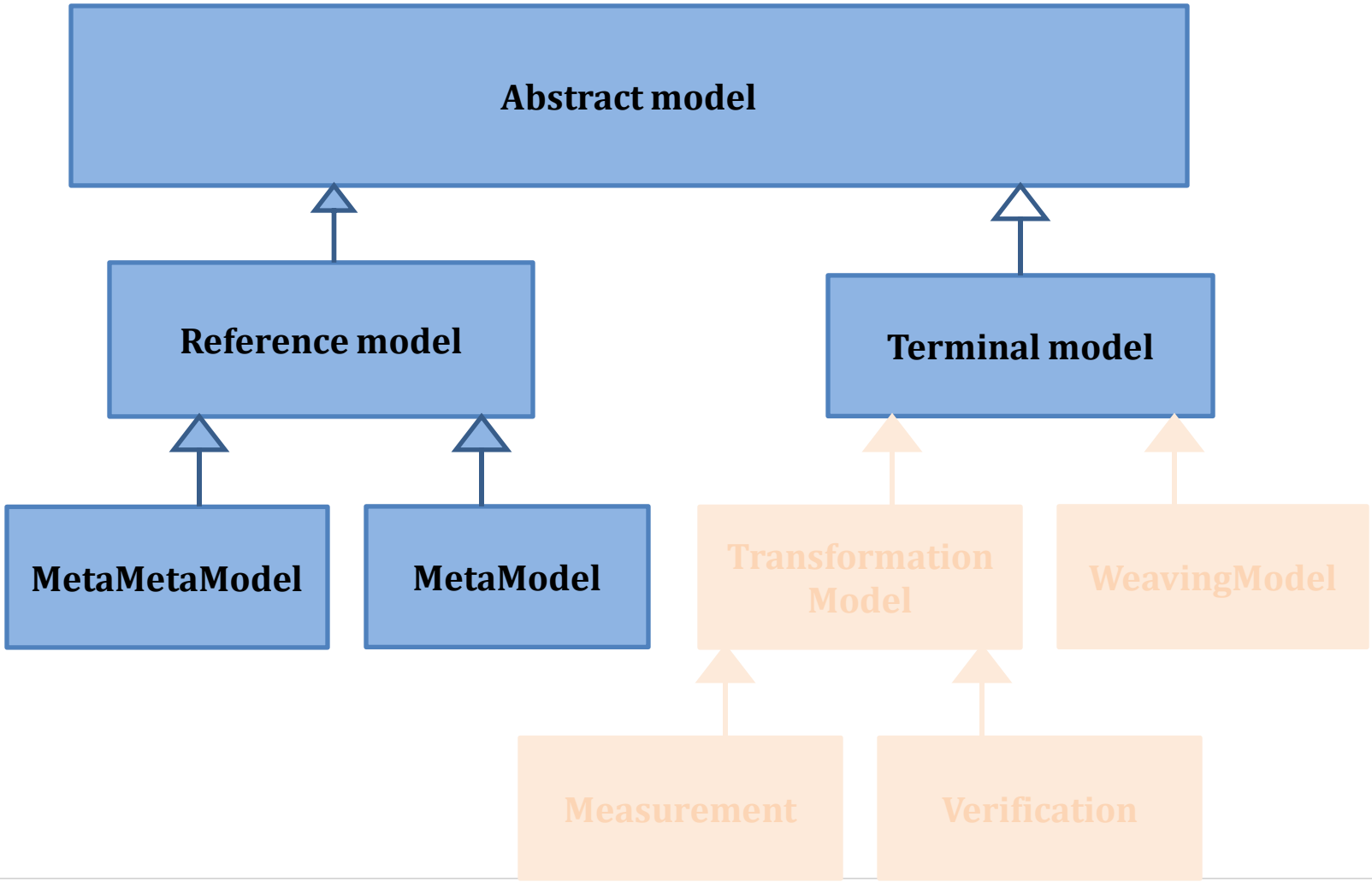
  - **Definition 5.** A terminal model is a model such that its reference model is a metamodel.



Abstract models



A more complete view



Collections of metamodels

- How to organize collections of reusable metamodels?
  - Several possible answers
    - ✓ Atlanmod zoo with 300+ unorganized metamodels of unequal quality (empirical material)
    - ✓ To a regularly organized lattice of metamodels
    - ✓ With all the intermediate possibilities

Zoo Federation - All in one Zoo.

Atlantic Zoo - A zoo of metamodels expressed in **KM3**. Basic measurements available at KM3

AtlantEcore Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in **EMF XMI 2.0**, c

Atlantic Microsoft SQL Server Modeling "M" Zoo - An ATL-auto-generated mirror of AtlantEco version only uses the **MSchema** part of the language).

AtlanticClojure Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in Clojure.

AtlanticSBVR Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed as SBVR vocat

Atlantic UML 2 Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in **UML 2.1**, co

Atlantic yUML Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in **yUML**, confo

Atlantic Raster Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in **PNG** bitmap

Atlantic GraphML Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in **GraphML**

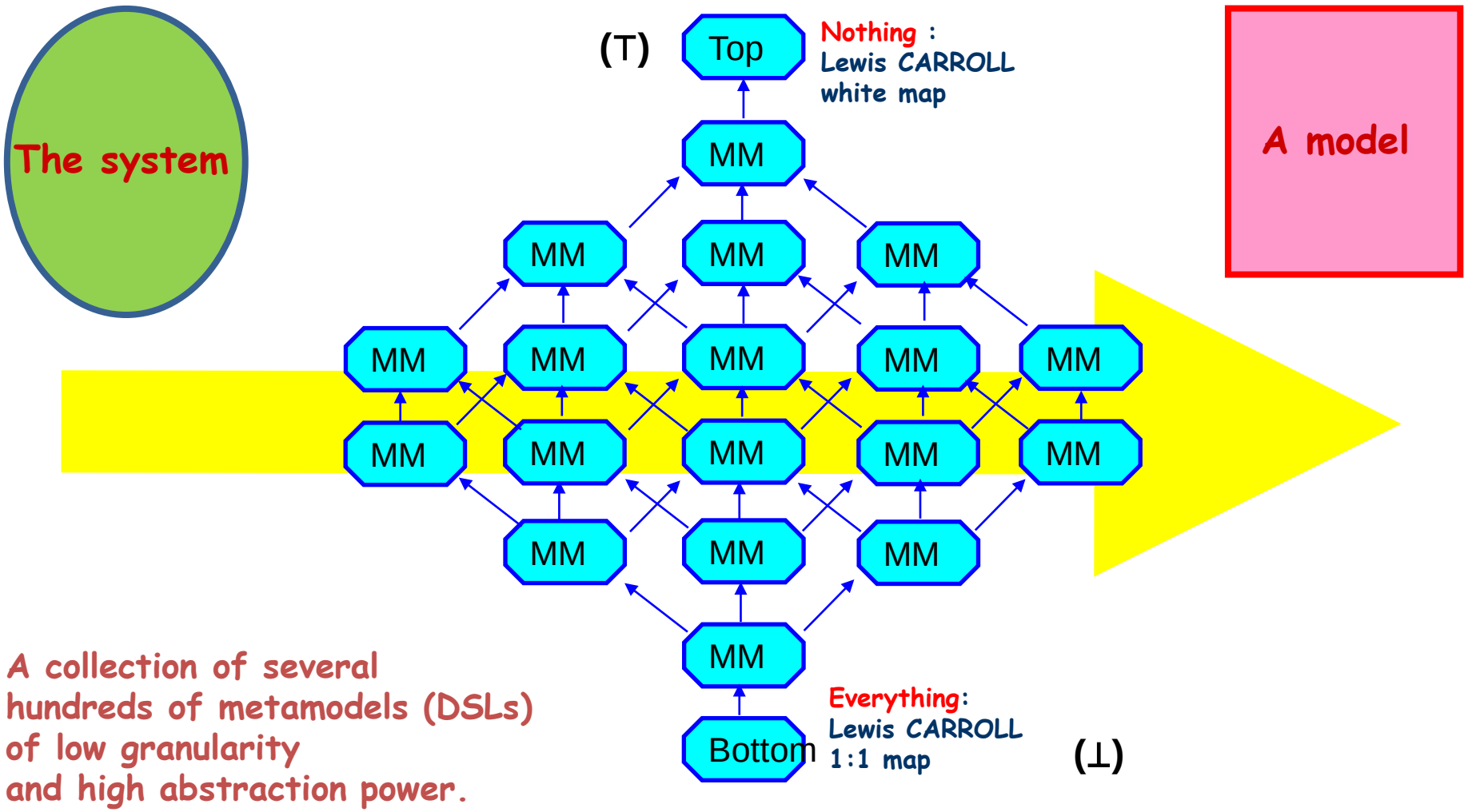
Atlantic Measurement Zoo - An ATL-auto-generated mirror of Atlantic zoo presenting measu

Atlantic MOF/MDR Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in **MDR XM**

Atlantic UML Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in **MDR XMI 1.2**,

Atlantic SQL DDL Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in **SQL DDL**,

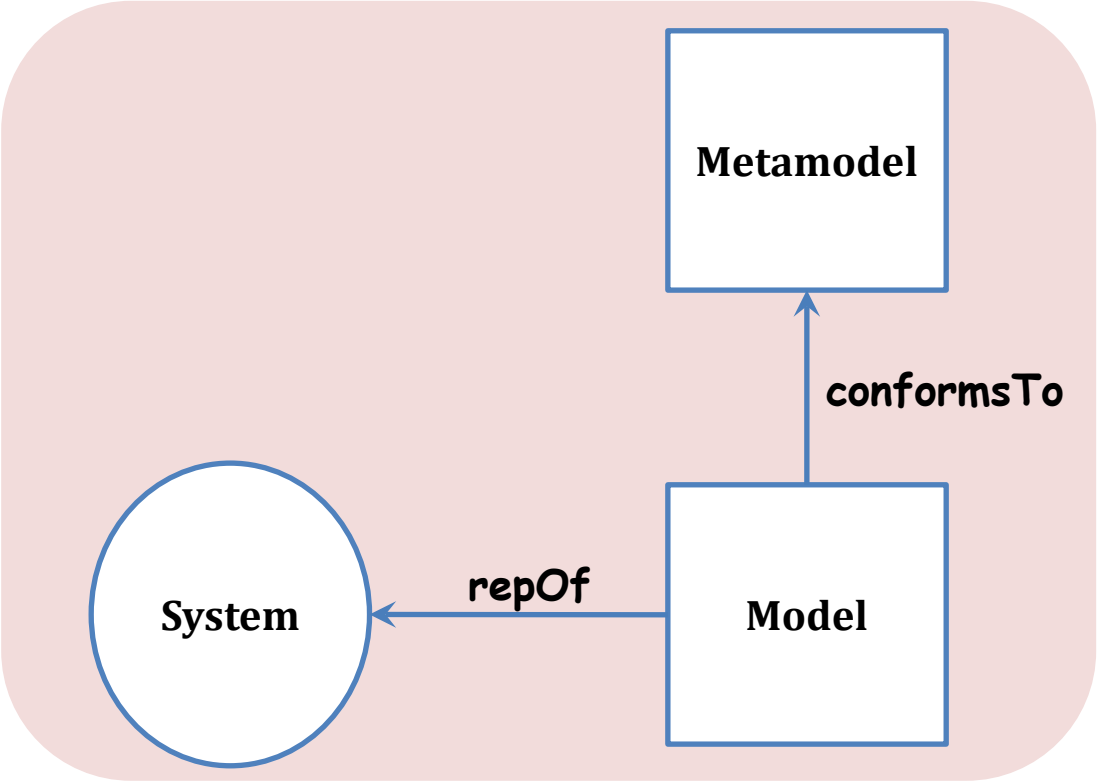
The vision: a lattice of metamodels



Utilization definition

Two orthogonal definitions for a model

- ✓ Structural definition (language dimension)
- ✓ Utilization definition (ontology dimension)



Utilization definition

The objective here is to define the possible usages of a model. Consequently, in all the present subsection, model will mean "terminal model".

- **Definition 6.** A system S is a delimited part of the world considered as a set of elements in interaction.

- **Definition 7.** A model M is a representation of a given system S, satisfying the substitutability principle (see below).

- **Definition 8.** (Principle of substitutability). A model M is said to be a representation of a system S for a given set of questions Q if, for each question of this set Q, the model M will provide exactly the same answer that the system S would have provided in answering the same question.

Principle of limited subtituability

❑We use the term "model" in the following sense: To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A. …

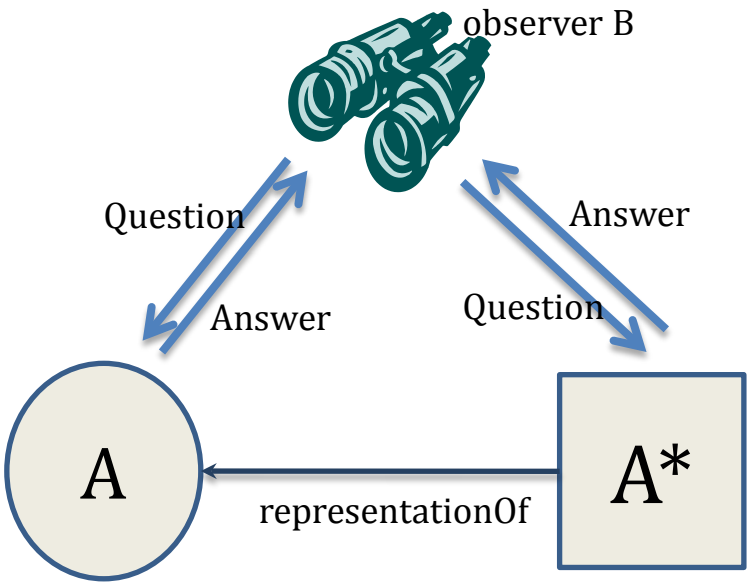
❑ It is understood that B's use of a model entails the use of encodings for input and output, both for A and A*. If A is the world, questions for A are experiments. …

❑ A* is a good model of A, in B's view, to the extent that A*'s answers agree with those of A's, on the whole, with respect to the questions important to B. …

Marvin L. Minsky

Matter, Mind and Models Semantic Information Processing,

MIT Press, 1968

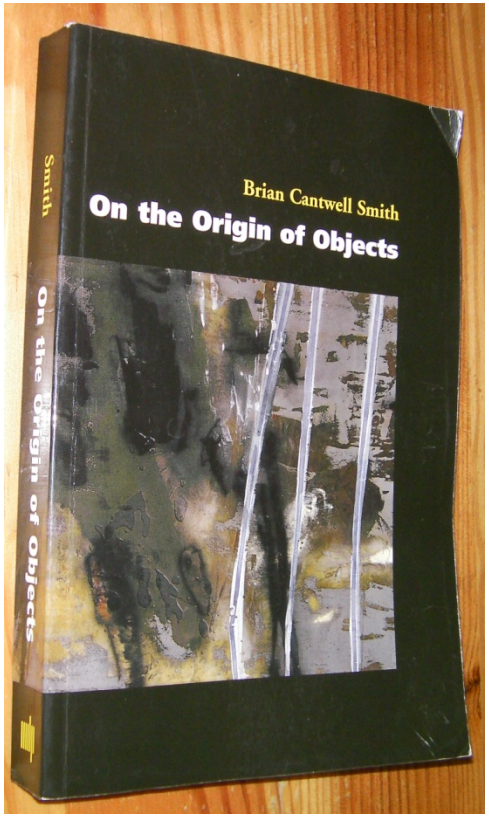


Taking the *representation* relation seriously

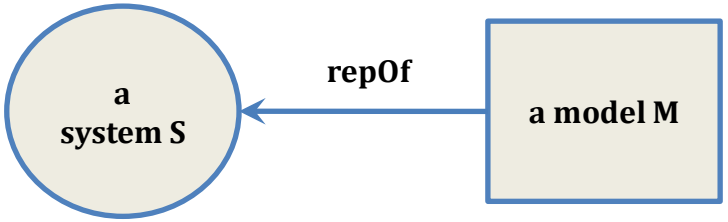
"*What about the [relationship between model and real-world]? The answer, and one of the main points I hope you will take away from this discussion, is that, at this point in intellectual history, we have no theory of this [...] relationship*".

Brian Cantwell Smith **The Limits of Correctness**
a paper prepared for the Symposium on Unintentional Nuclear War, Fifth Congress of the International Physicians for the Prevention of Nuclear War, Budapest, Hungary, June 28 – July 1, 1985.

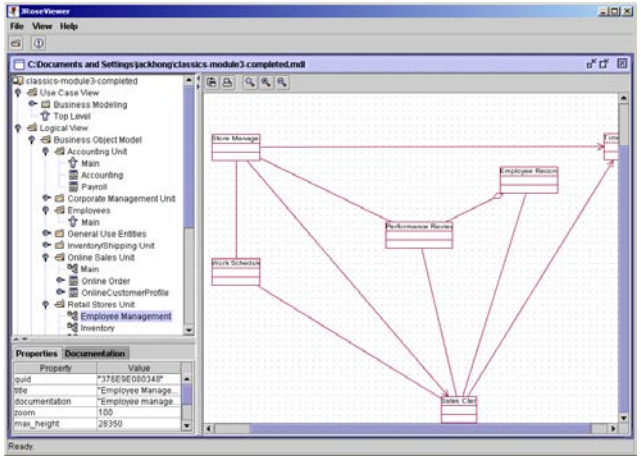
See also "***On the origin of objects***"
ISBN 0-262-69209-0

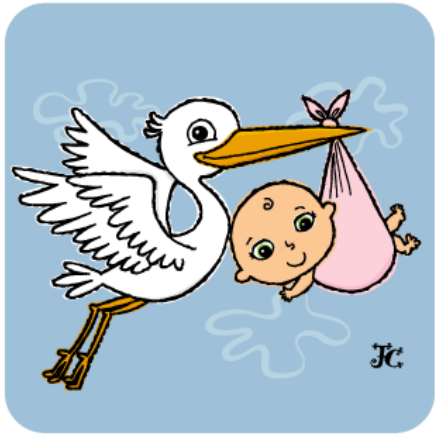


On the origin of models



Naïve answer:
from Rational Rose

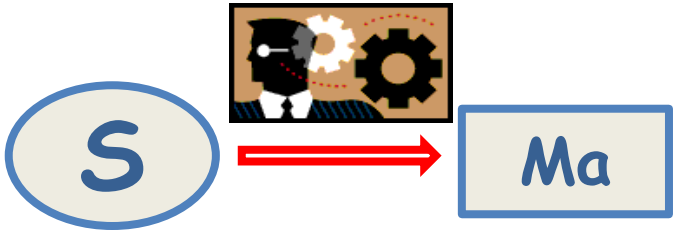




How to assess the fidelity (or faithfulness) of models?

- Two possible sources for models (only two, <u>initial</u> and <u>derived</u>)
  1. <u>Transformation</u> of other models

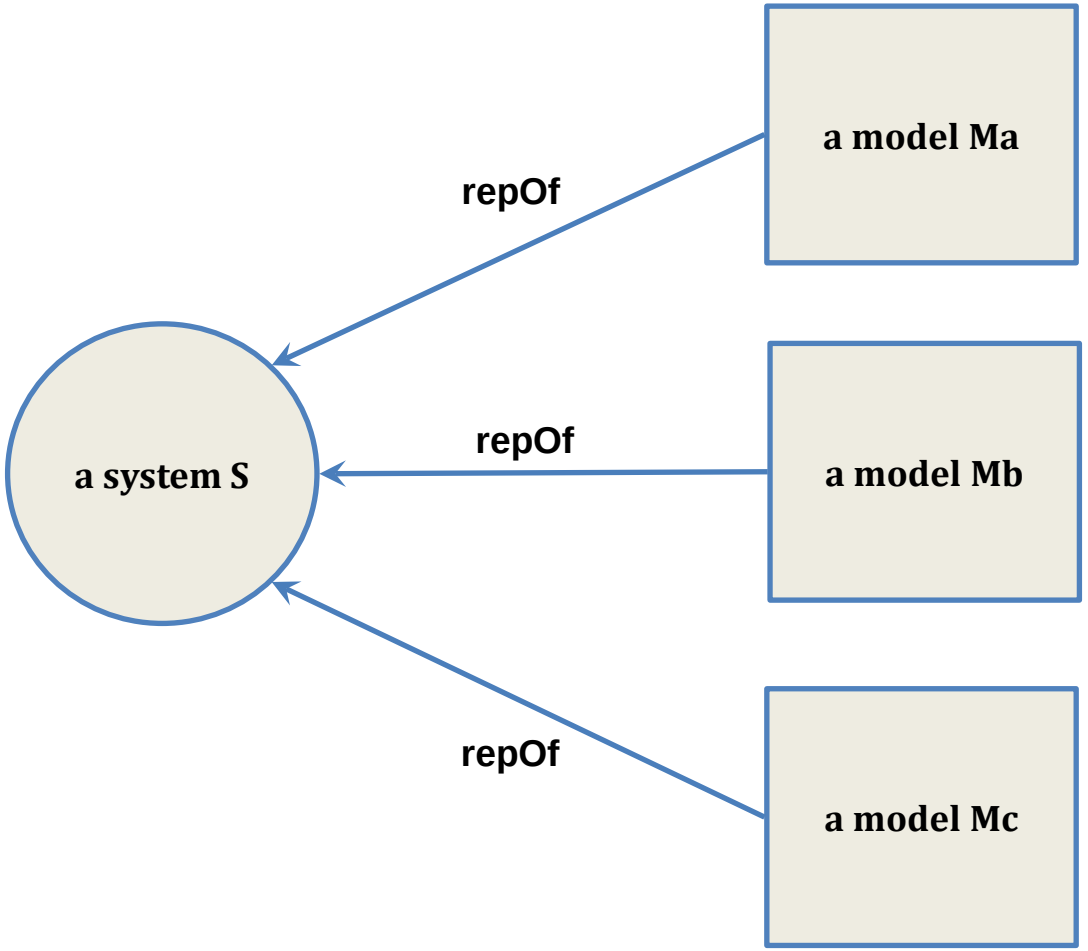
  2. <u>Observation</u> of a system



- Because we gave too much emphasis to our technology and not enough to its applications, computer science has got a bad reputation of buggy and loose engineering.
- The real question of the origin of objects has never been seriously addressed by the community.
- A software can be written in the most wonderful language and completely misunderstand the reality it is supposed to control and miss its goal.
- We have never seriously invested time and energy in seriously studying the faithfulness or fidelity of models to systems and this is one reason why we deserve the bad reputation of our software systems. It's about time we start studying ways to evaluate how accurate a copy is to its source.
- If we want to improve the public image of software engineering, this is an important field on which we should invest in future research.

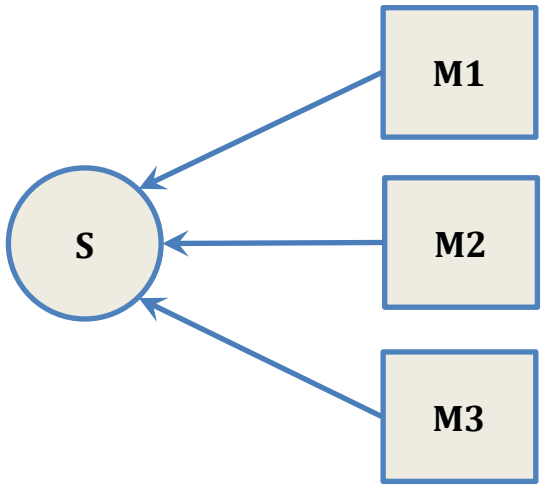
Multimodeling

- ✓ Multimodeling is the joint exploitation of different models representing the same system.
- ✓ **These models usually conform to different metamodels.**
- ✓ Multimodeling suggests to manage complex systems by collaborative reasoning based on multiple models, each one encompassing a specific type of knowledge (e.g. structural, behavioral, functional) and representation.
- ✓N.B.: the different models (Ma, Mb, Mc, …) may also be linked together.

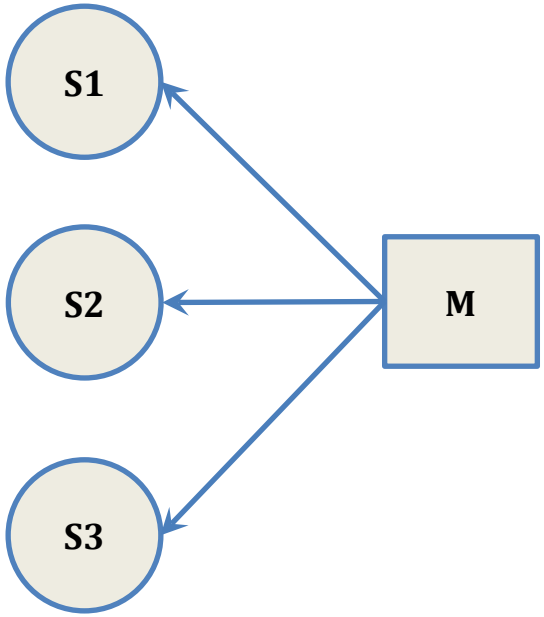


Not to be confused

Multimodeling



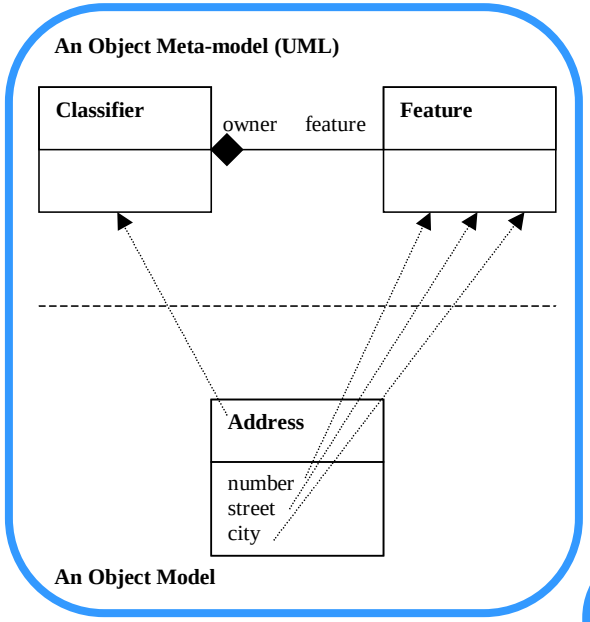
Composite Modeling



A usual program is an example of a composite model, representing at the same time the domain, the execution platorm, and the application user requirements.

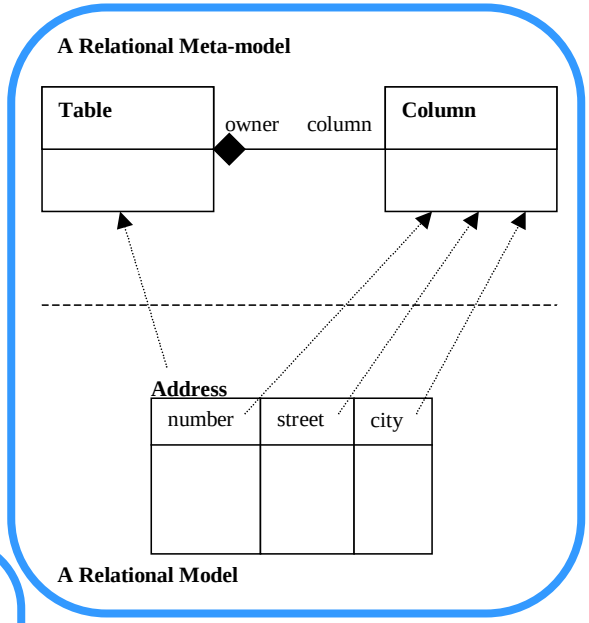
Model transformations

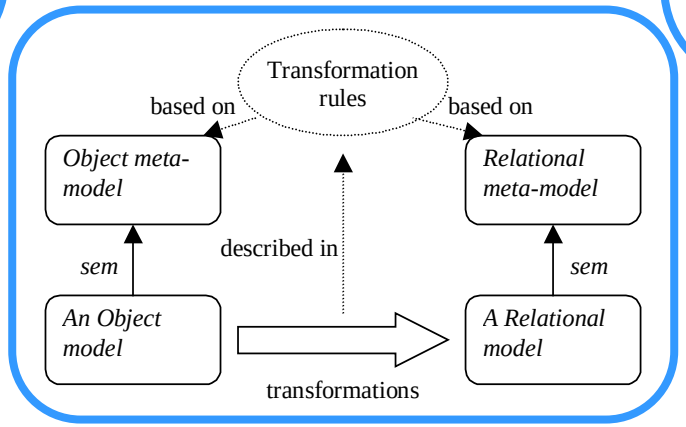
Transformation Rules Based on Meta-modeling



Lemesle Richard
Laboratoire de Recherche en Sciences de Gestion, Université de Nantes

Proceedings Second International
ENTERPRISE DISTRIBUTED OBJECT COMPUTING





ENTERPRISE DISTRIBUTED OBJECT COMPUTING
EDOC '98
WORKSHOP

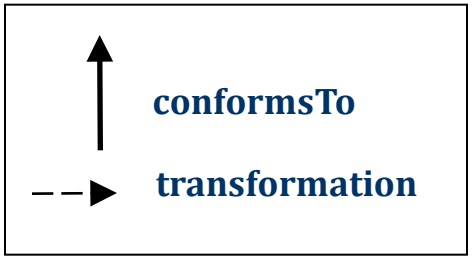
Endogeneous and exogeneous transformations

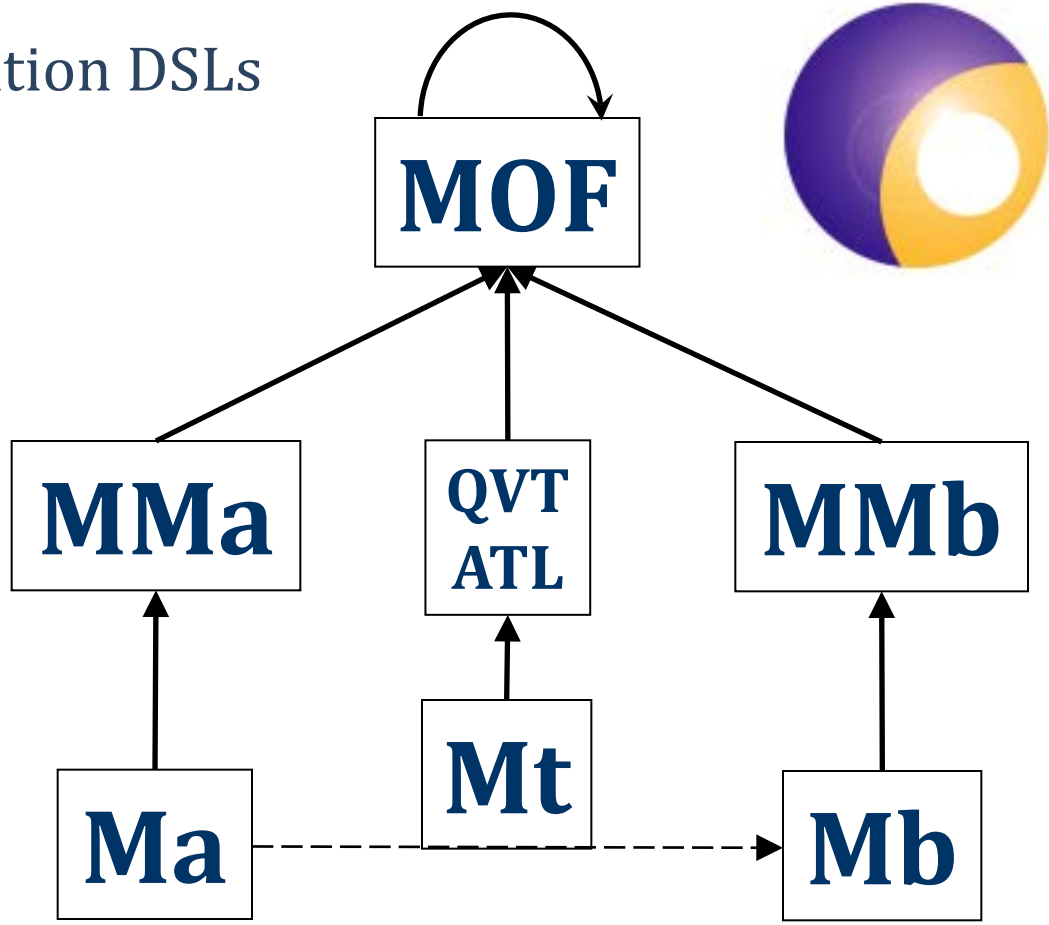
- Endogeneous: MMa = MMb
- Exogeneous: MMa ≠ MMb
  - Changing the source and target metamodels to MMa ∪ MMb may change the type from exogeneous to endogeneous

Model transformations as models

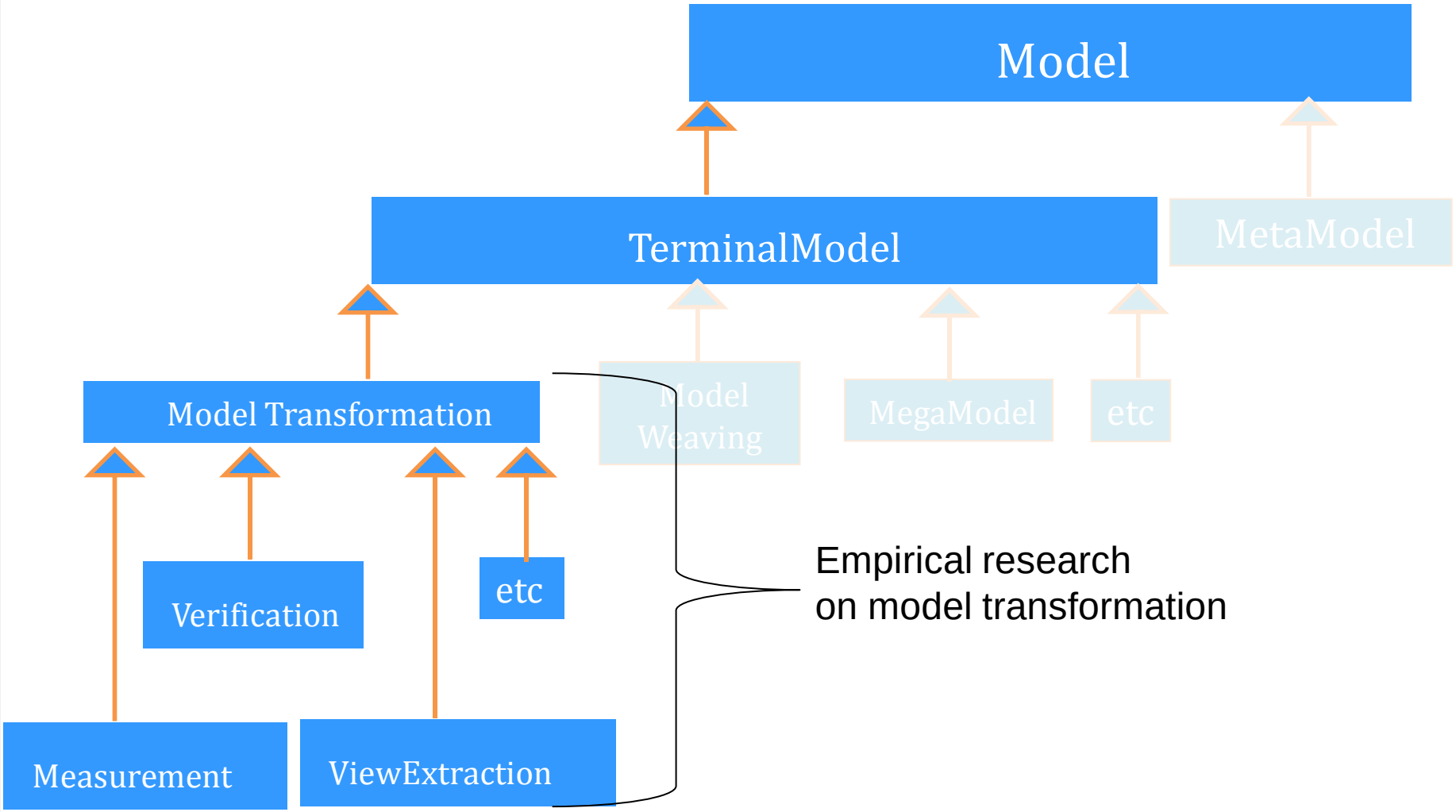
Model-to-Model transformation DSLs

- A transformation (*Mt*) builds a target model (*Mb*) from a source model (*Ma*)
- Model transformations are transformation models
  - Each model conforms to a metamodel
  - A transformation is a model (*Mt*) conforming to a metamodel (*MMt*)

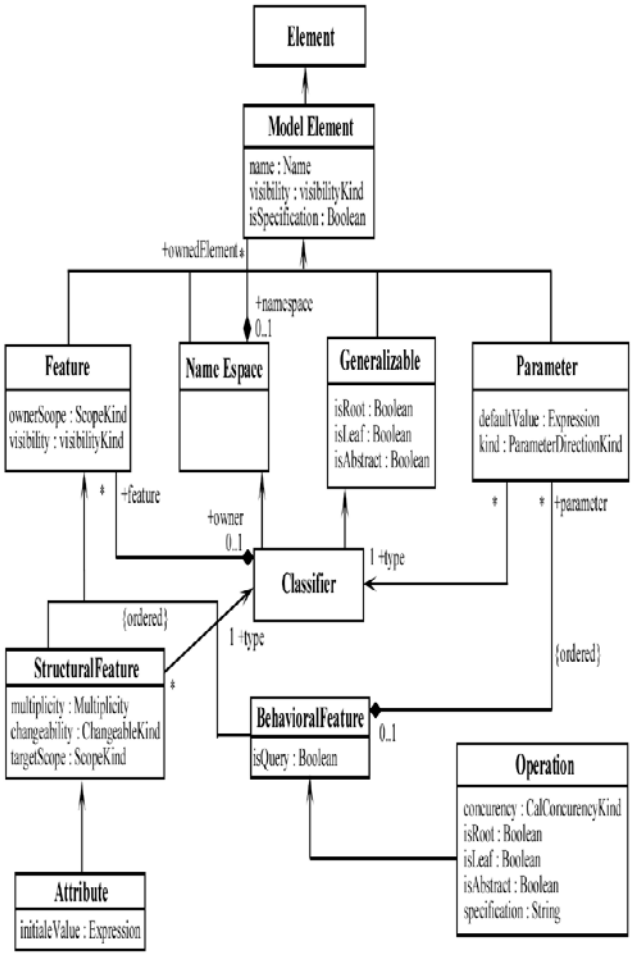




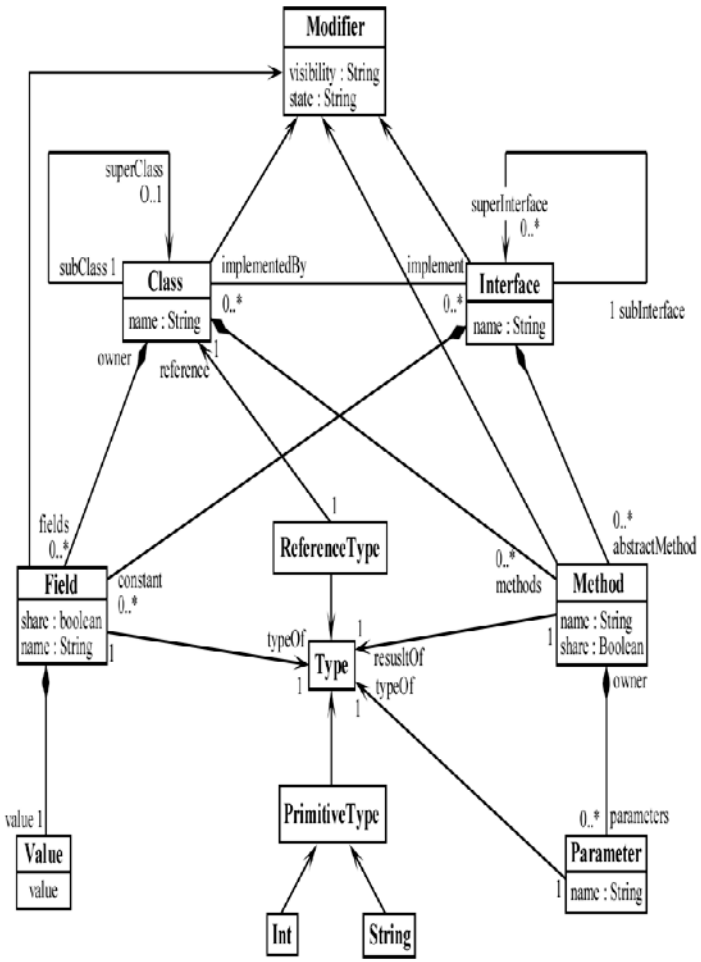
Experiments in Model Transformation



Example : UML to Java



UML Metamodel

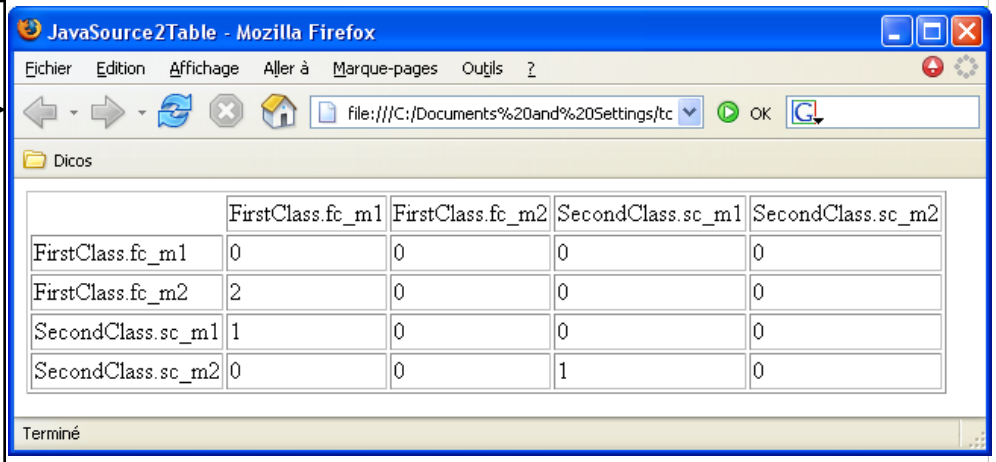


Java Metamodel

Example : Java to Excel

FirstClass.java	SecondClasss.java

`public class FirstClass {`
`  public void fc_m1(){`
`  }`
`  public void fc_m2(){`
`  this.fc_m1();`
`  this.fc_m1();`
`  }`
`}`
	`public class SecondClass {`
	`  public void sc_m1(){`
	`  FirstClass a = new FirstClass();`
	`  a.fc_m1();`
	`  }`
	`  public void sc_m2(){`
	`  this.sc_m1();`
	`  }`
	`}`



	FirstClass.fc_m1	FirstClass.fc_m2	SecondClass.sc_m1	SecondClass.sc_m2
FirstClass.fc_m1	0	0	0	0
FirstClass.fc_m2	2	0	0	0
SecondClass.sc_m1	1	0	0	0
SecondClass.sc_m2	0	0	1	0

Example: XSLT to XQuery

XSLT

```
<xsl:stylesheet […] >
    <xsl:template match="/">
          <emps>
            <xsl:apply-templates
    select="employee"/>
          </emps>
    </xsl:template>
    <xsl:template match="employee">
      <xsl:if test="salary&gt;2000">
        <emp>
          <xsl:value-of select="name"/>
          <xsl:value-of
    select="firstname"/>
        </emp>
      </xsl:if>
    </xsl:template>
</xsl:stylesheet>
```

XQuery

```
define function fctemployee($paramVar)
{
  for $var in $paramVar
  return
    let $var := $var
    where $var/salary>2000
    return

     <emp>{$var/name}{$var/firstname}</emp
     >
}
for $var in document("xmlFile.xml")/*
return

     <emps>{fctemployee($var/employee)}</e
     mps>
```

Note: TS crossing needed

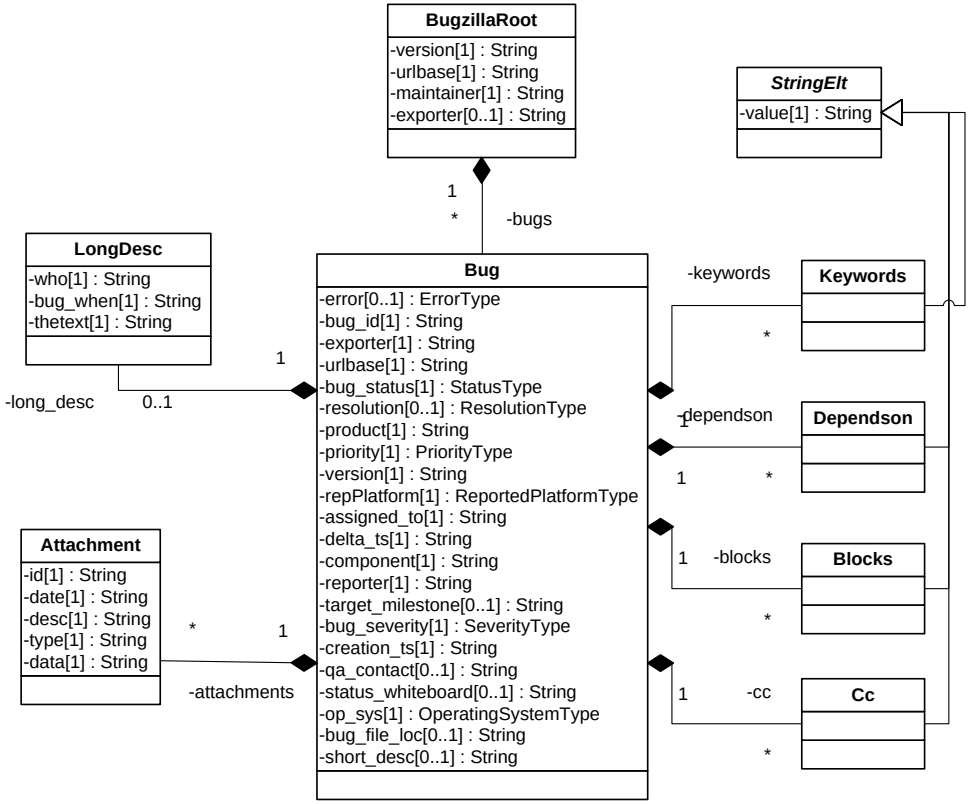
Example: Tool Interoperability

About 30 open source tools to choose among for bug tracking (Mantis, Bugzilla, etc.)

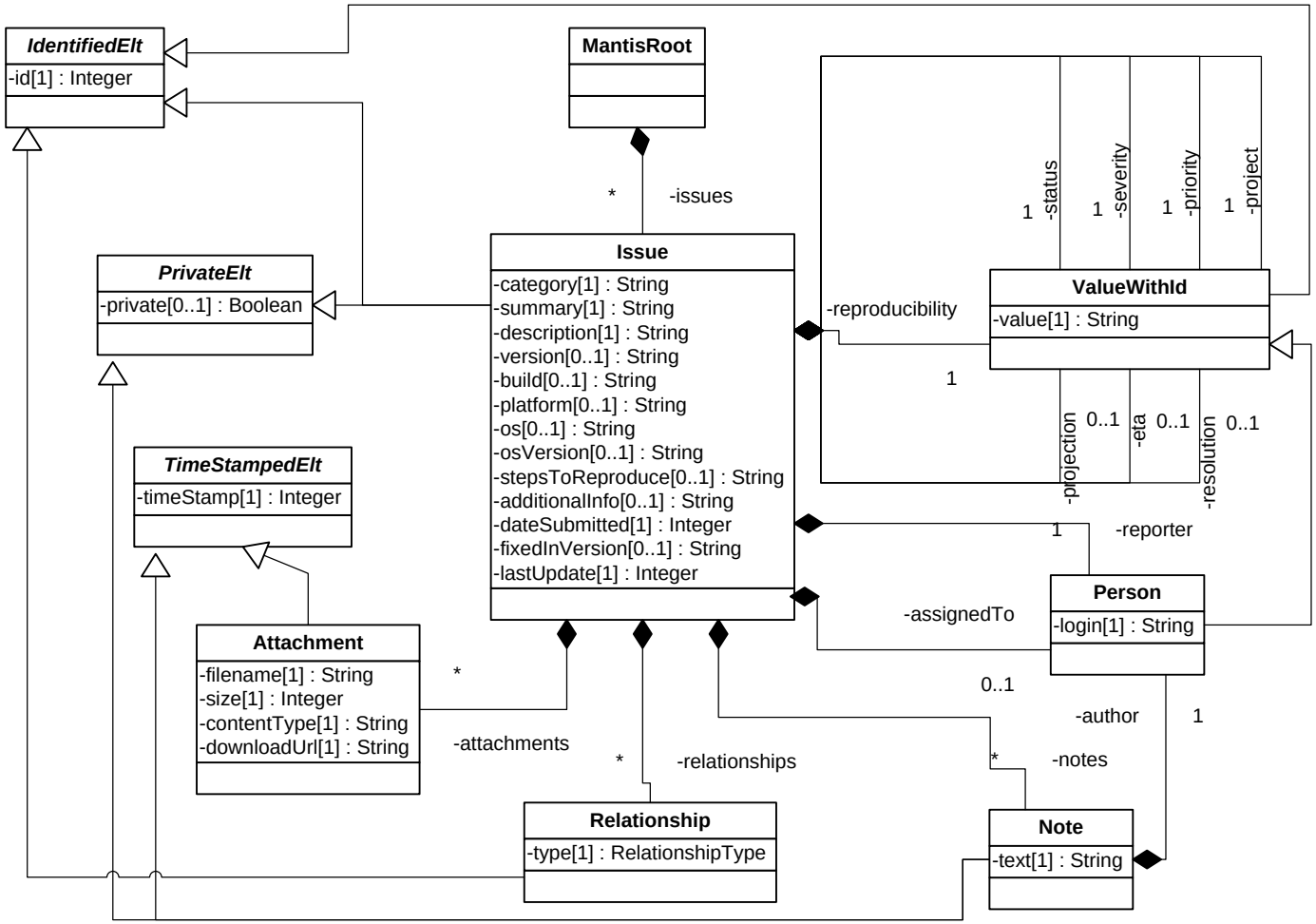
Different data models and functionalities

Tool	Lang	Ver	Cust	Temp	Search	RSS	Not	Rep	Hist	Attach	Updated	Demo	Score
ASP.NET Starter	C#/VB		Yes		Yes							No	2
Bug-a-Boo	CGI			Yes	Yes		Yes				Feb 05	Yes	4
Bug Base	Java										Aug 03		0
BugIn	PHP										May 04		0
Bugs Online	ASP				Yes			Yes	Yes		Jan 02	No	3
BugTracker	Java										Apr 01	No	0
BugTracker.NET	C#		Yes	No	Yes	No	Yes	Yes	Yes	Yes	Apr 05	No	7
Bugzilla	Perl		Yes		Yes	No	Yes		Yes	Yes	Jan 05	No	5
Eventum	PHP	1.5.4	Yes		Yes		Yes	Yes	Yes	Yes	Jun 05	No	7
EZ Ticket	PHP							Yes			Jan 04	No	1
Flyspray	PHP				Yes		Yes			Yes	Jan 05	Yes	4
GNATS											Mar 05	Yes	1
Issue Tracker	PHP			Yes			Yes			Yes	Feb 04	Yes	4
ITracker	Java		Yes		Yes		Yes	Yes	Yes	Yes	Aug 04	Yes	7
JTracker	Python		No								May 04	No	0
Mantis	PHP		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	May 05	Yes	10
Midge	Python				Yes						Oct 04	No	0
OpenPSA Support	PHP				Yes			Yes			Jan 05	Yes	3
OTRS	Perl			Yes	Yes		Yes		Yes	Yes	Oct 04	Yes	6
phpBugTracker	PHP			Yes	Yes			Yes		Yes	Nov 04	Yes	5
PloneCollector-NG	Python		Yes		Yes	Yes	Yes	Yes		Yes	Apr 04	No	6
Request Tracker	Perl		Yes	Yes	Yes	Yes	Yes		Yes	Yes	Feb 05	No	7
Roundup	Python		Yes	Yes	Yes		Yes		Yes	Yes	Mar 05	Yes	7
sBugs	Java										Jan 02		0
Scarab	Java		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Apr 04	Yes	9
Subissue											N/A	No	0
SugarCRM			Yes	Yes	Yes						Jun 05	Yes	5
Trac	Python				Yes	Yes		Yes	Yes	Yes	Nov 04	Yes	6
Whups	PHP			Yes	Yes				Yes	Yes		Yes	5
Workbench	PHP										Apr 02	No	1
Zope Issue Tracker	Python										Dec 03	Yes	1
Zwiki Tracker	Python								Yes			Yes	2

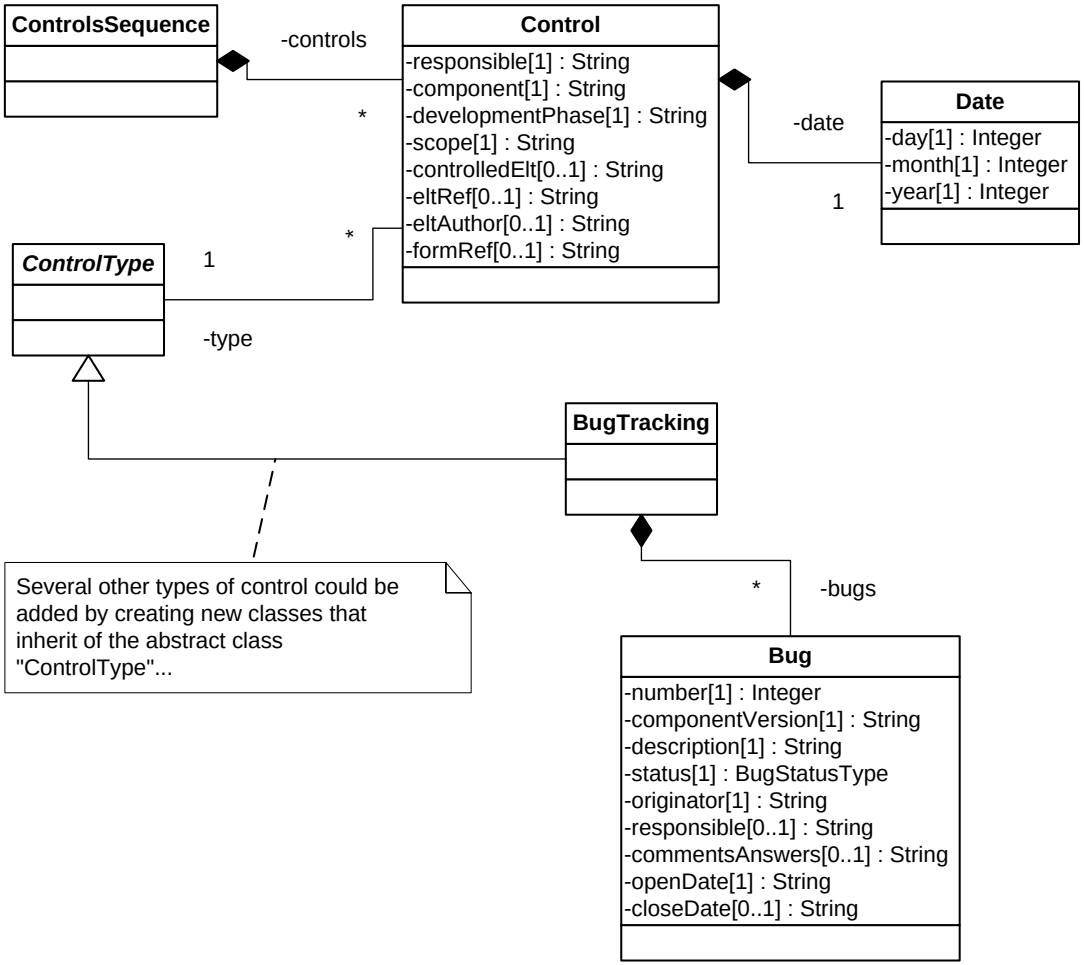
Bugzilla metamodel (simplified)



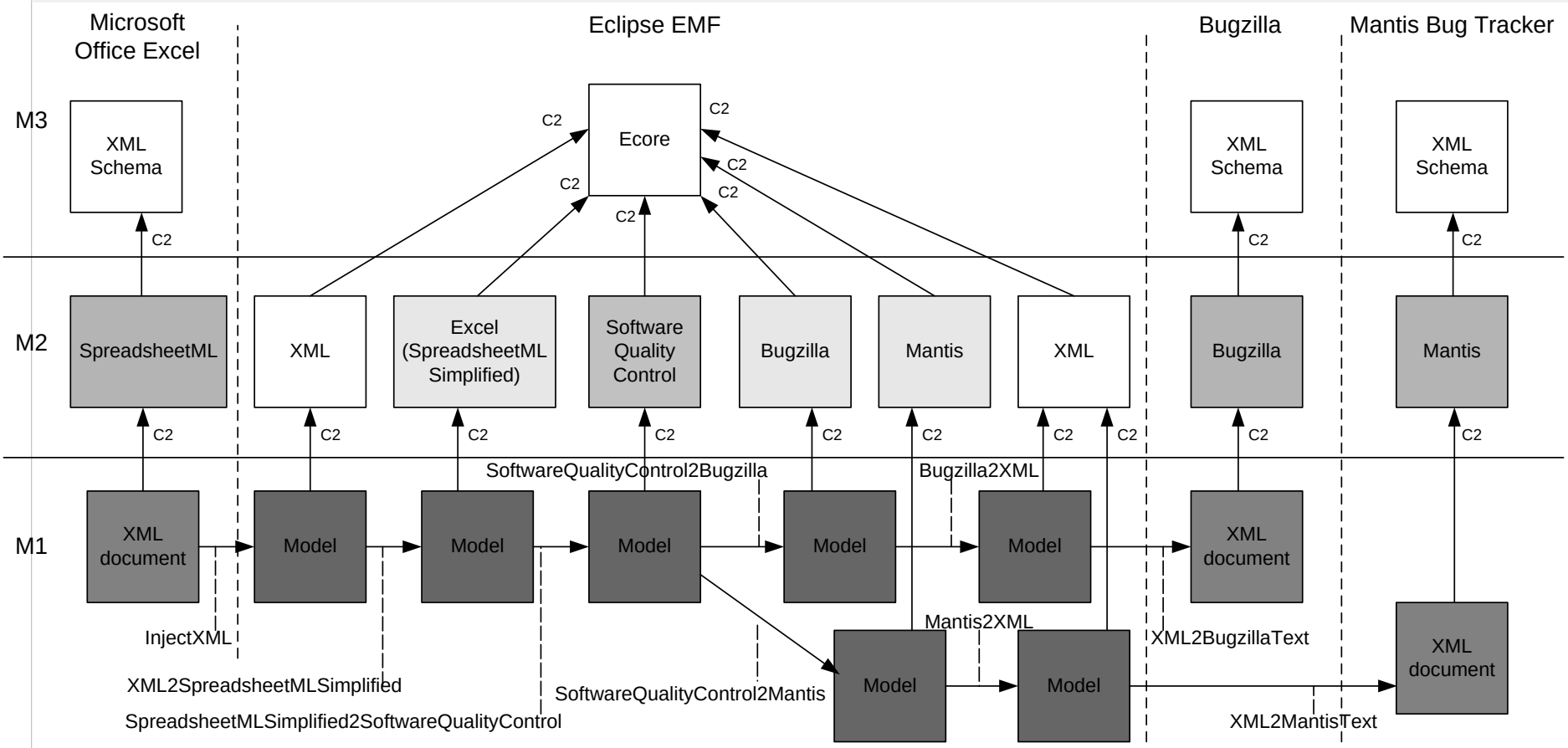
Mantis metamodel (simplified)



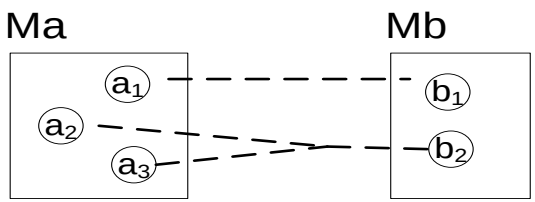
Bug control metamodel (pivot)

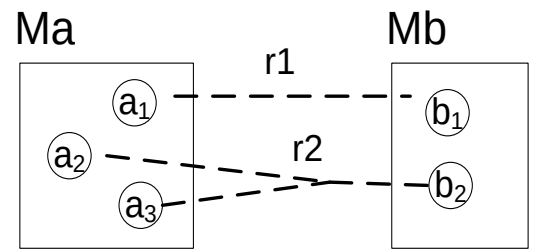


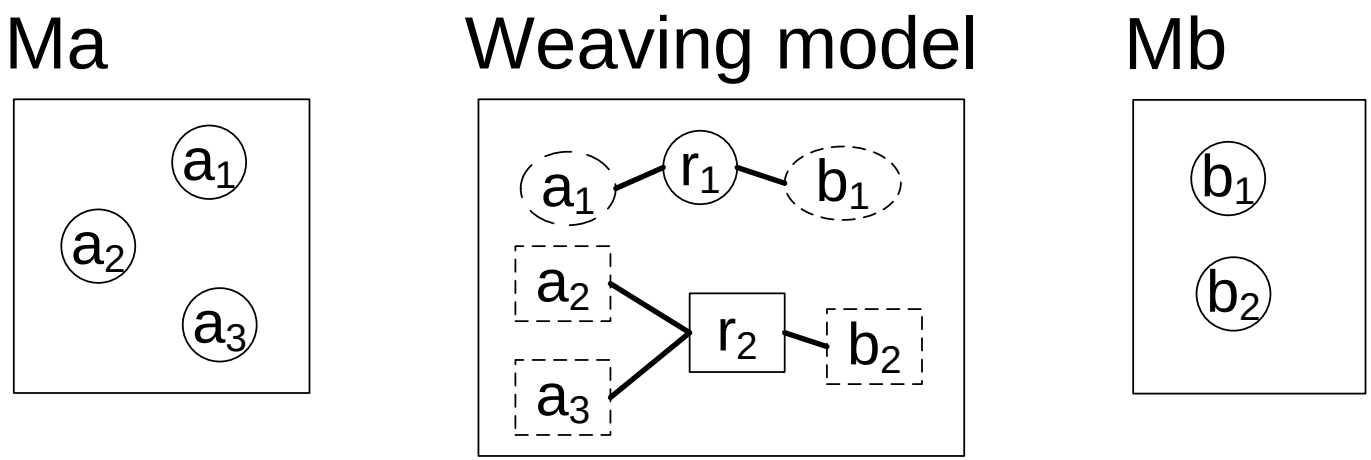
Excel-to-Bugzilla and Excel-to-Mantis bridges



Partial Correspondences or weavings

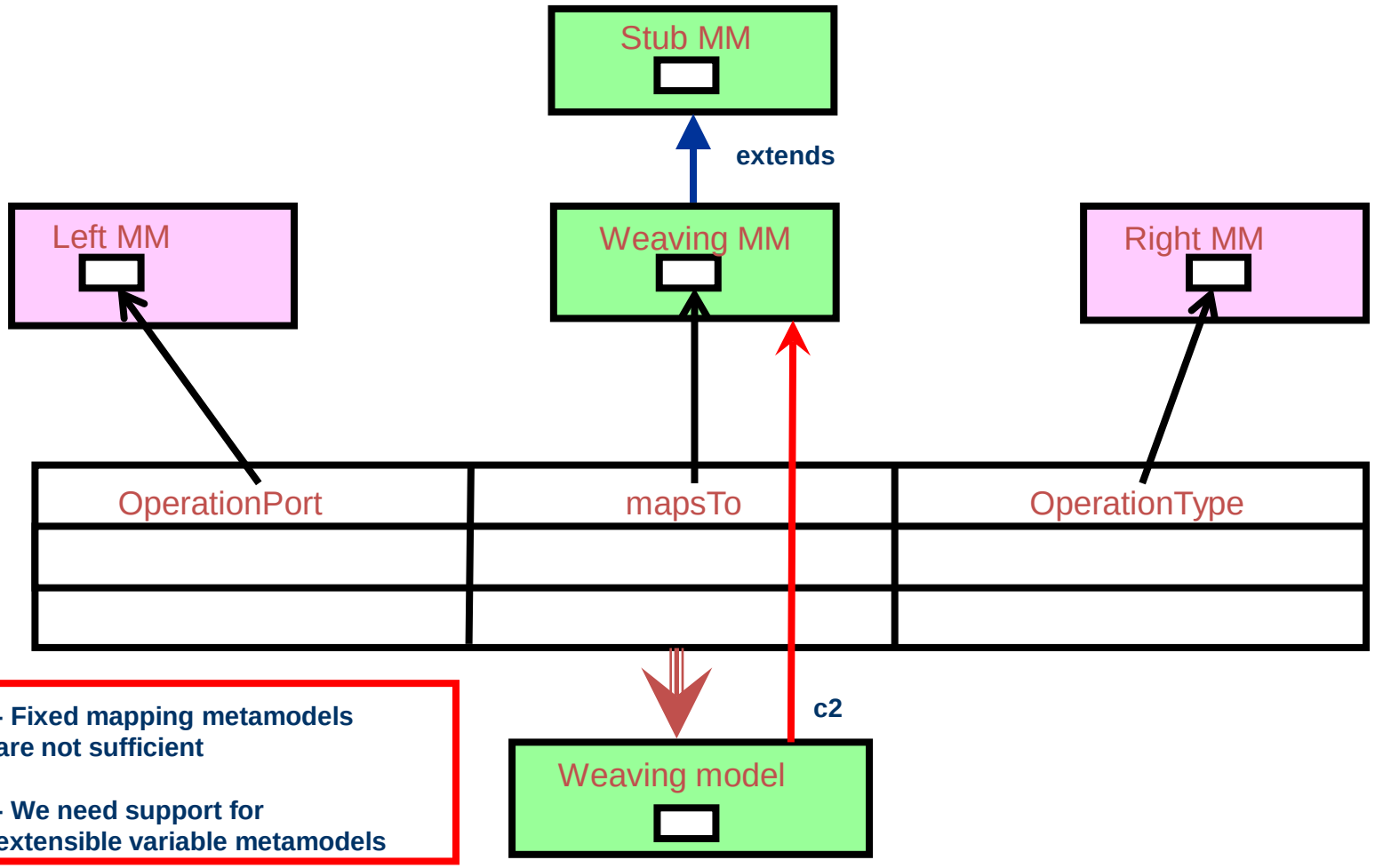






1. How to represent the partial correspondences?
   (M c2 an extensible MM)
2. How to compute the partial correspondences? (model matching)
3. How to use the partial correspondences? (HOTs)

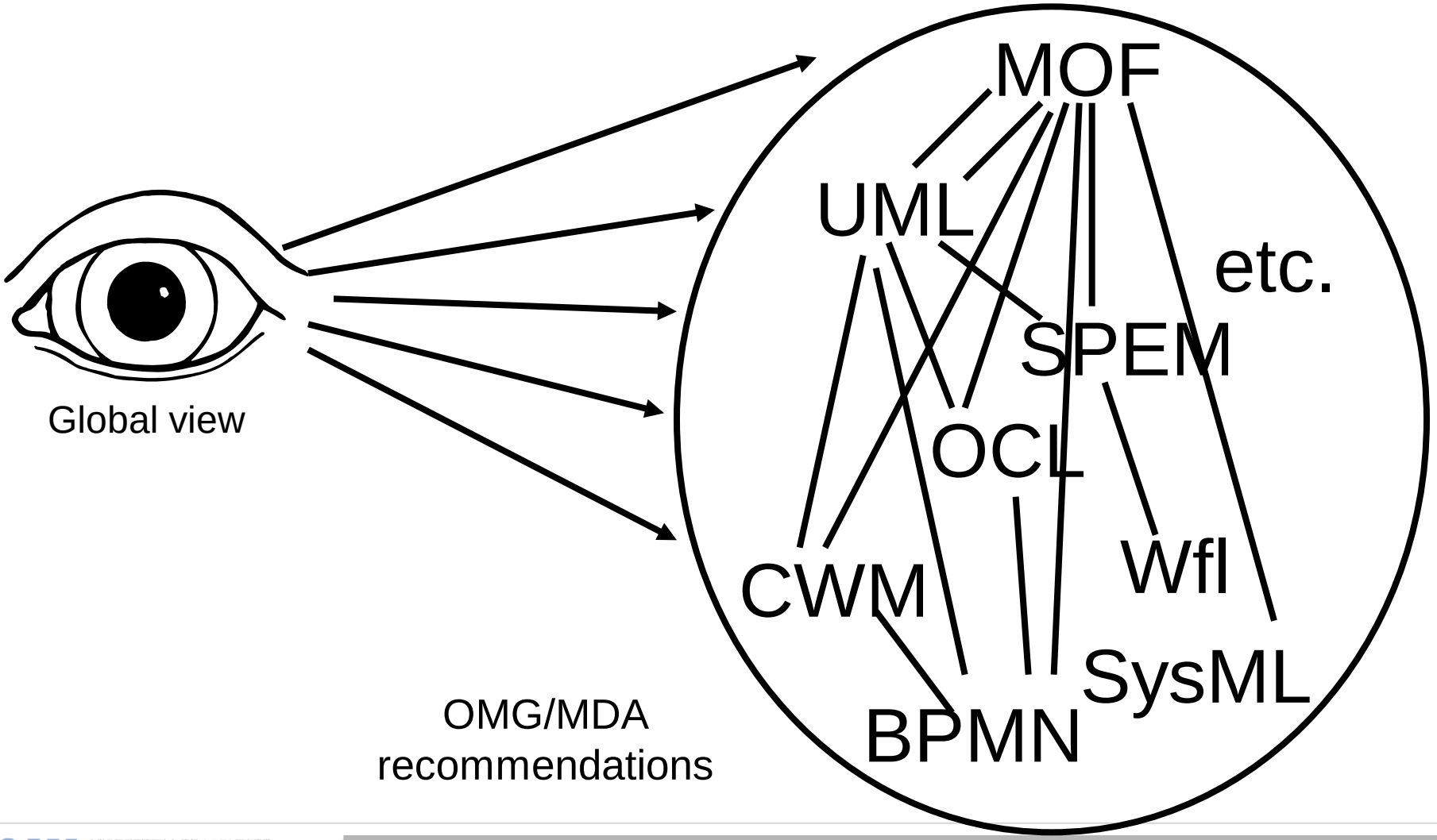
Principles of Model Weaving



- Fixed mapping metamodels are not sufficient

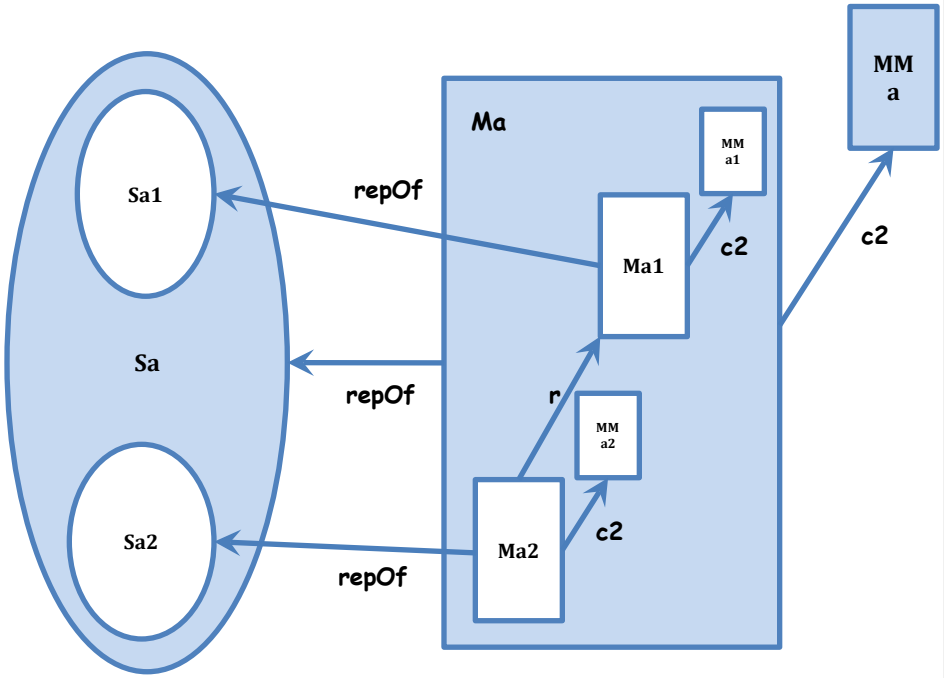
- We need support for extensible variable metamodels

Need for a global view: Megamodels



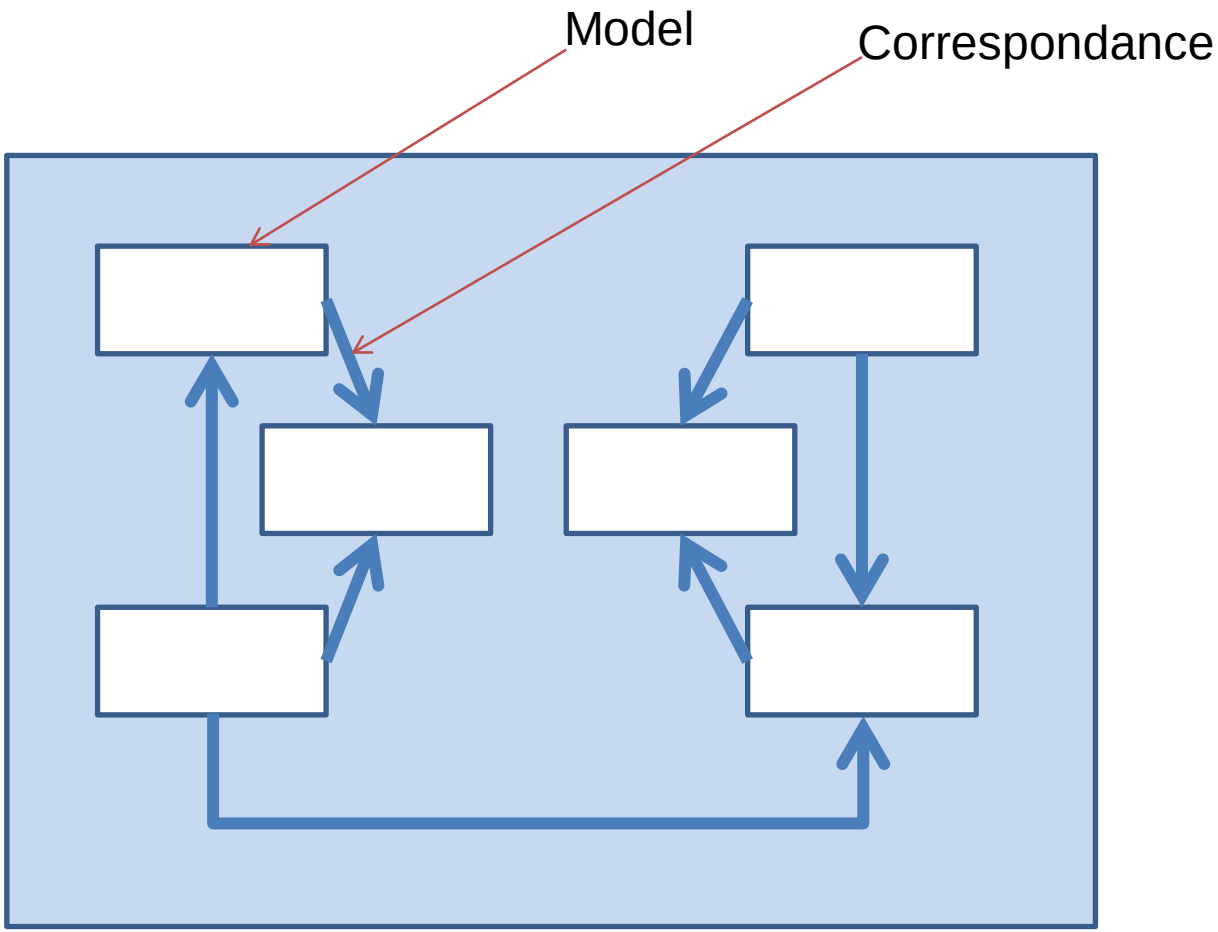
Megamodeling

- With megamodeling, a given model may describe a set of other models, their metadata and the mutual relationships between them.
- Since a megamodel is itself a model, this allows to represent arbitrarily deeply nested systems of systems.
- MDE solution for complex systems management

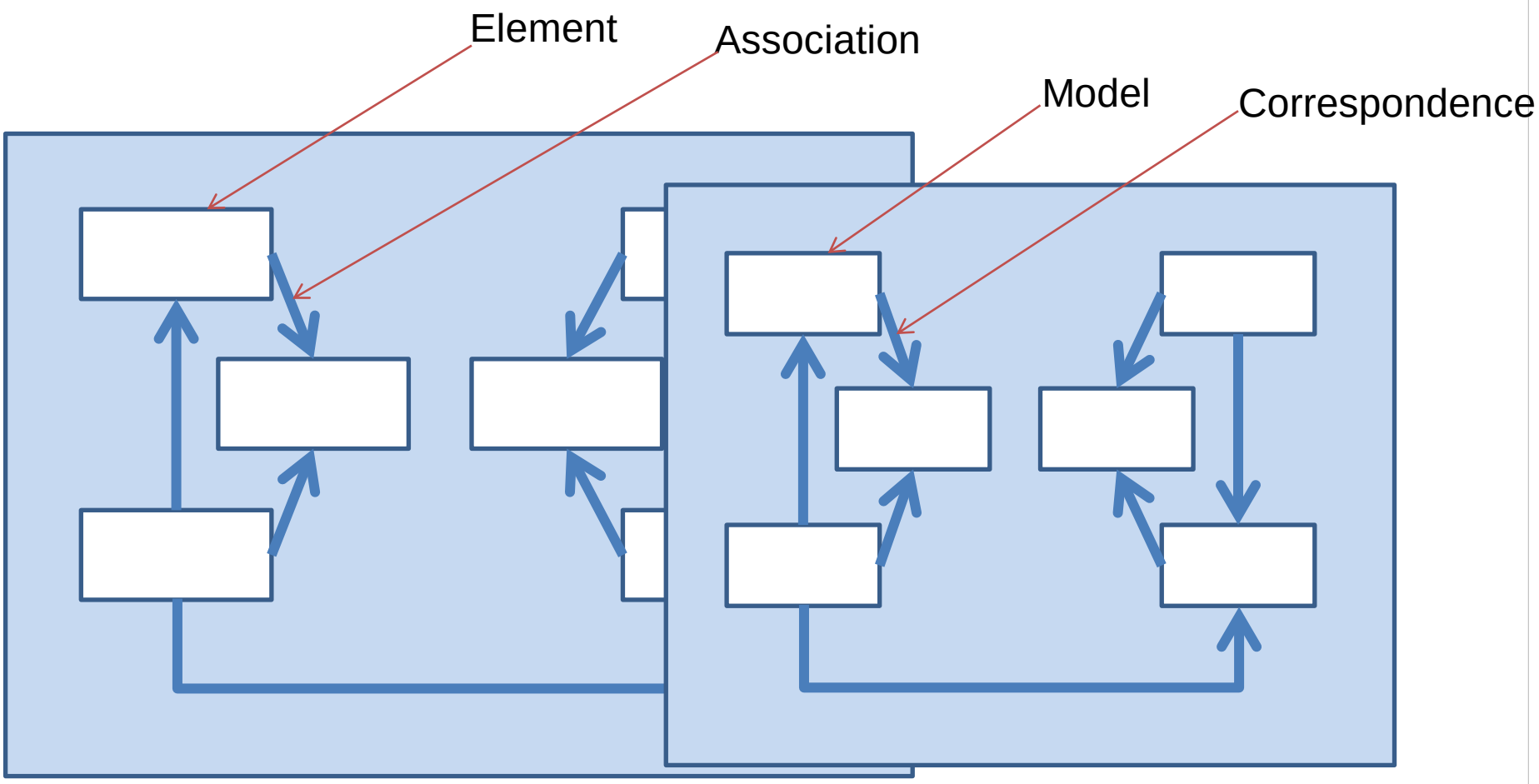


Area of Interest

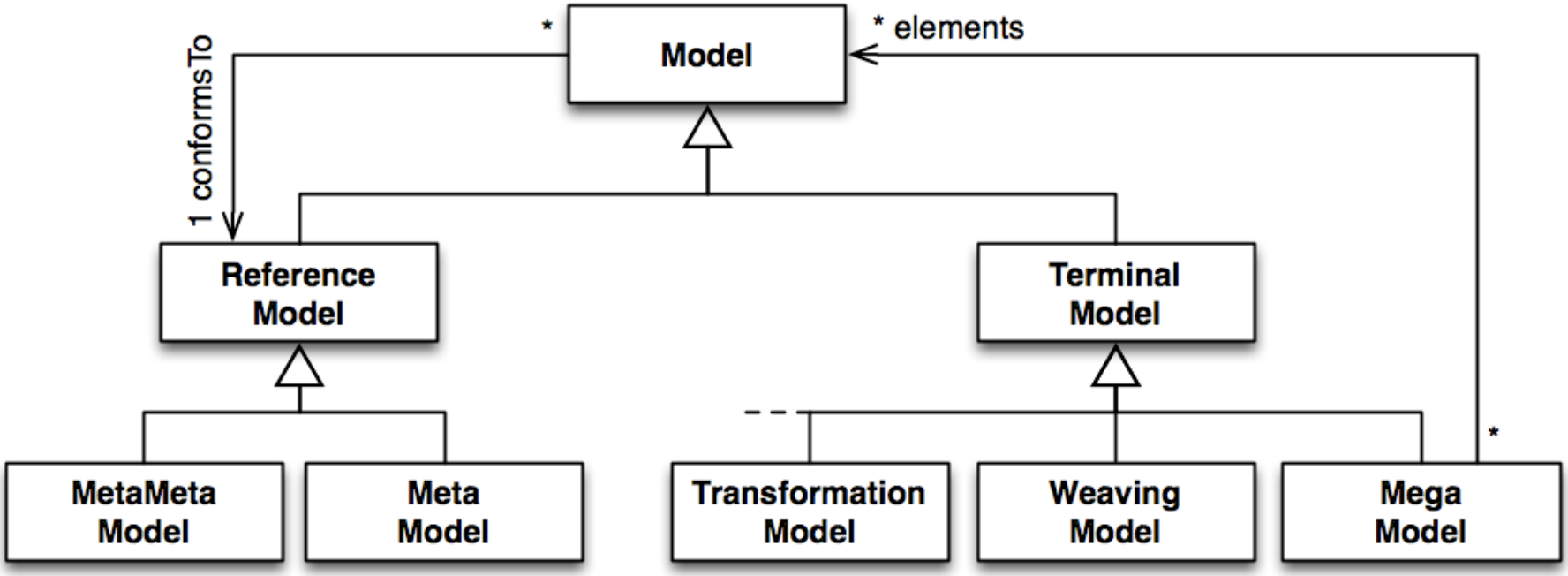
A model map (megamodel)



A model

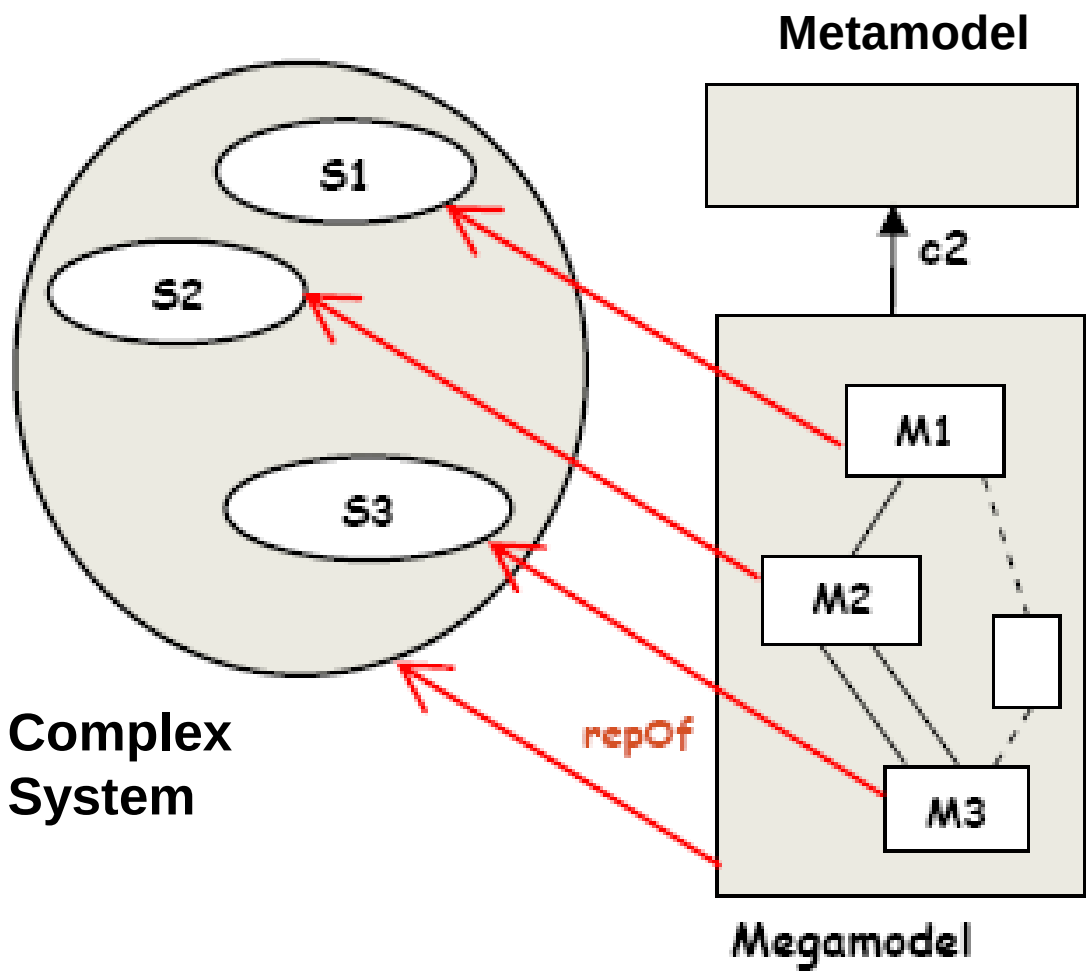


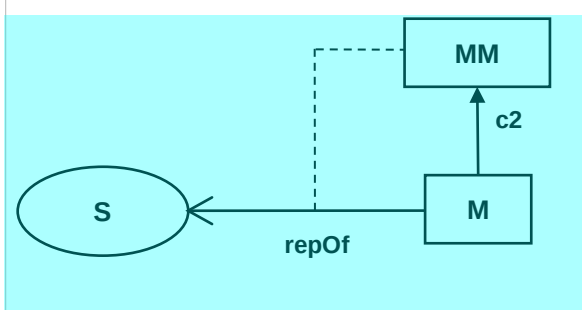
General organization



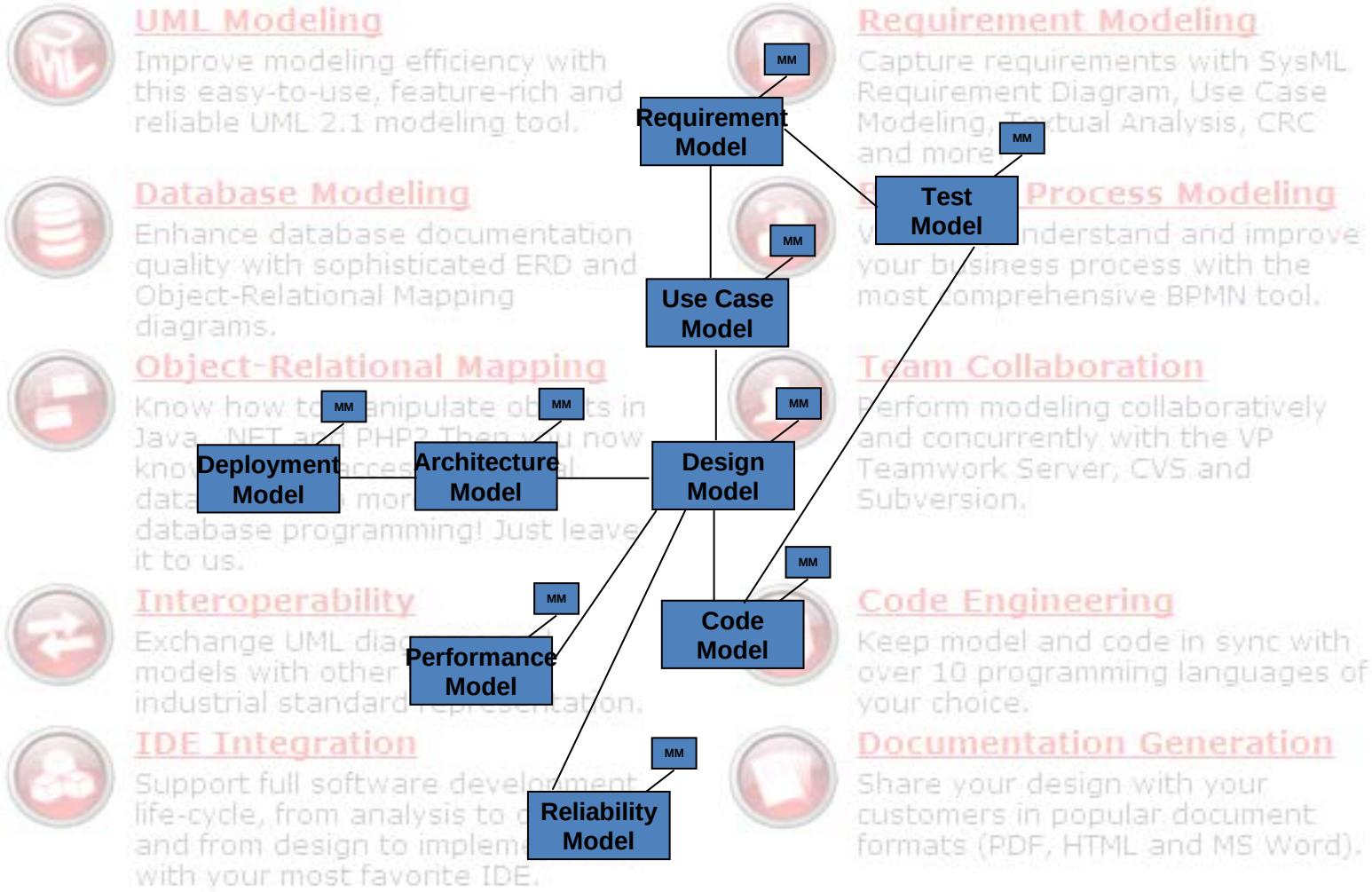

```
context MetaMetaModel inv: self.conformsTo = self
context MetaModel inv: self.conformsTo.oclIsKindOf(MetaMetaModel)
context TerminalModel inv: self.conformsTo.oclIsKindOf(MetaModel)
```

Megamodels for complex systems





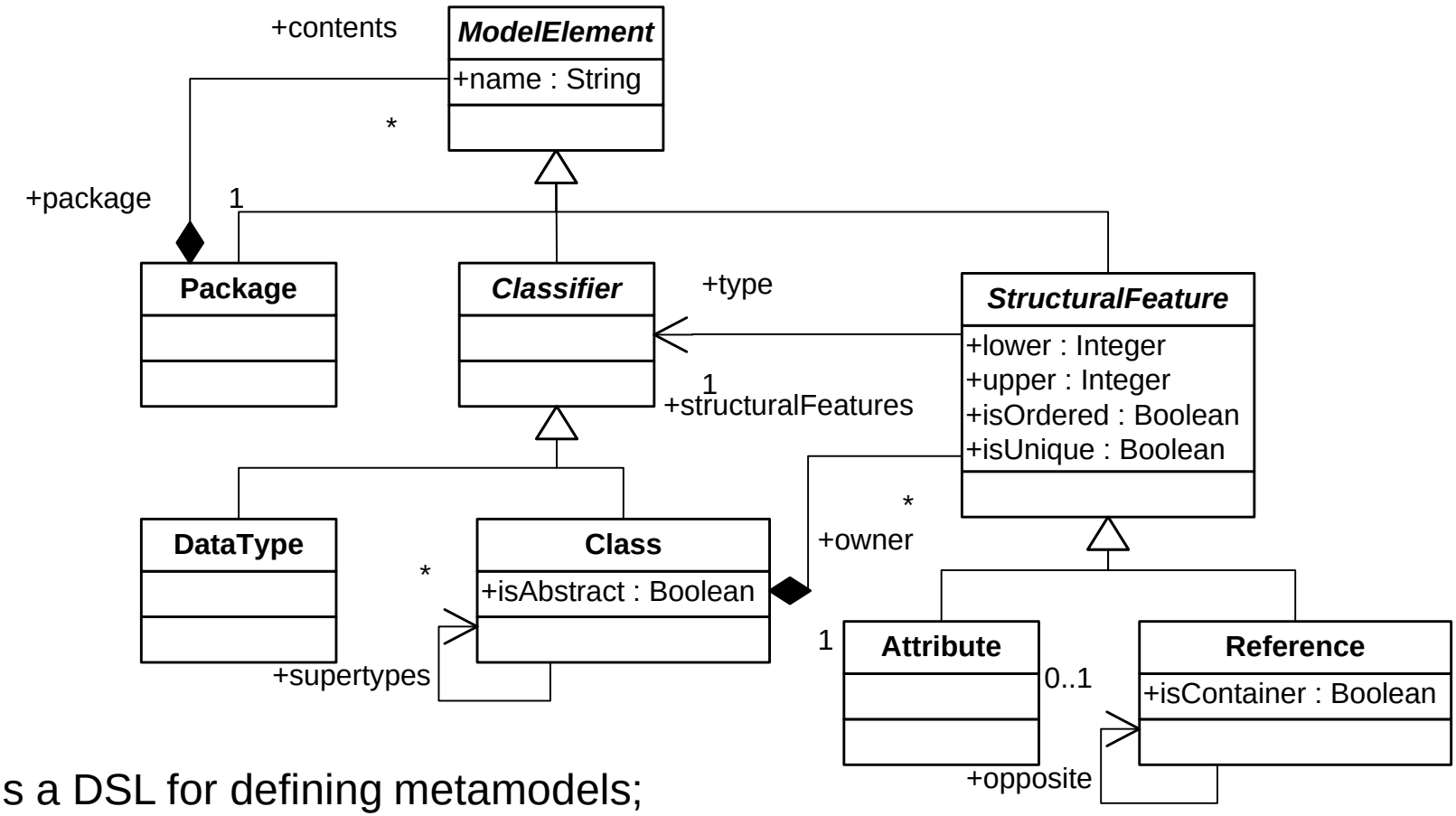
Software Project



How to define a metametamodel?

- Several examples
  - MOF, EMOF, CMOF, SMOF, ECORE, etc.
- Two main approaches
  - Universalist
  - Minimalist
- The design of a metametamodel is a difficult exercise in engineering (many tradeoffs)
- Two examples
  - KM3
  - sNets

The KM3 metamodel (compromise)



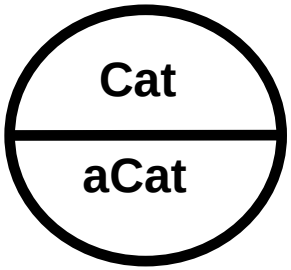
KM3 is a DSL for defining metamodels;
It's a M3 notation (self-defined)

[The presentation above is simplified.]

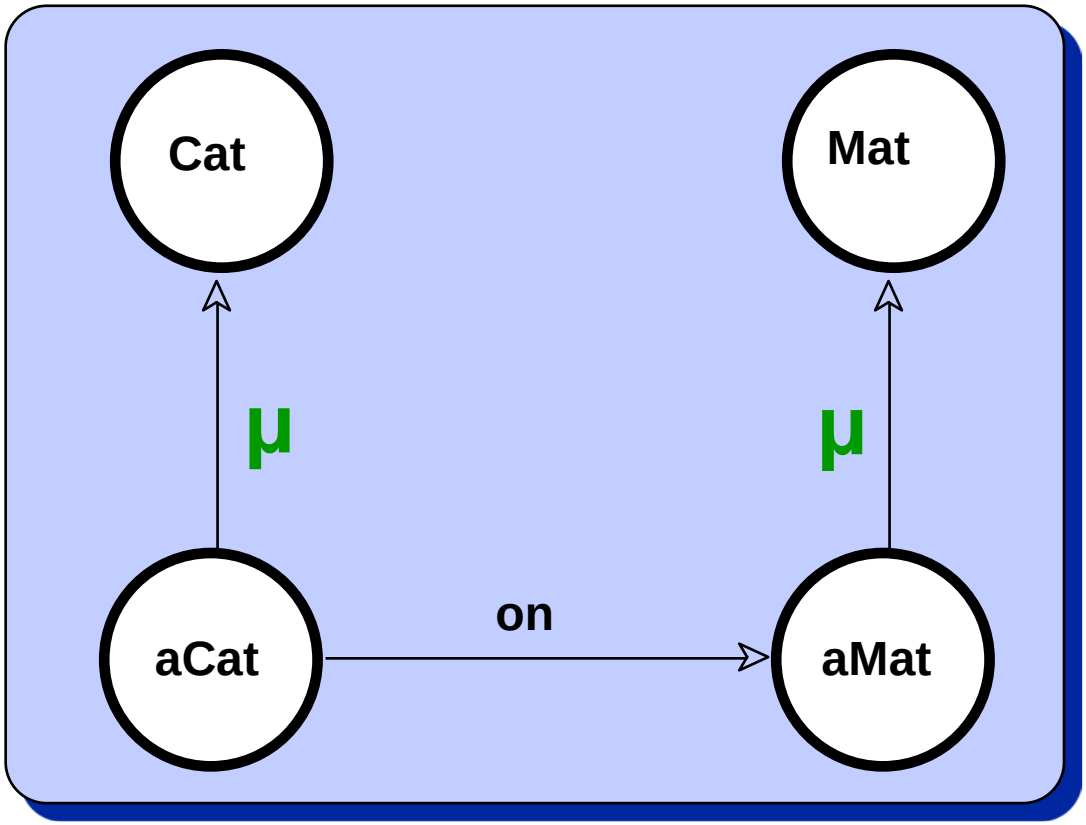
Self-definition of the KM3 metamodel

```
package KM3 {

    class Package extends ModelElement {
        reference contents[*] ordered
                container : ModelElement oppositeOf "package";
    }

    class Classifier extends ModelElement {}

    class DataType extends Classifier {}

    class Class extends Classifier {
        attribute isAbstract : Boolean;
        reference supertypes[*] : Class;
        reference structuralFeatures[*] ordered
                container : StructuralFeature
                oppositeOf owner;
    }
...
```

The sNets minimalist approach

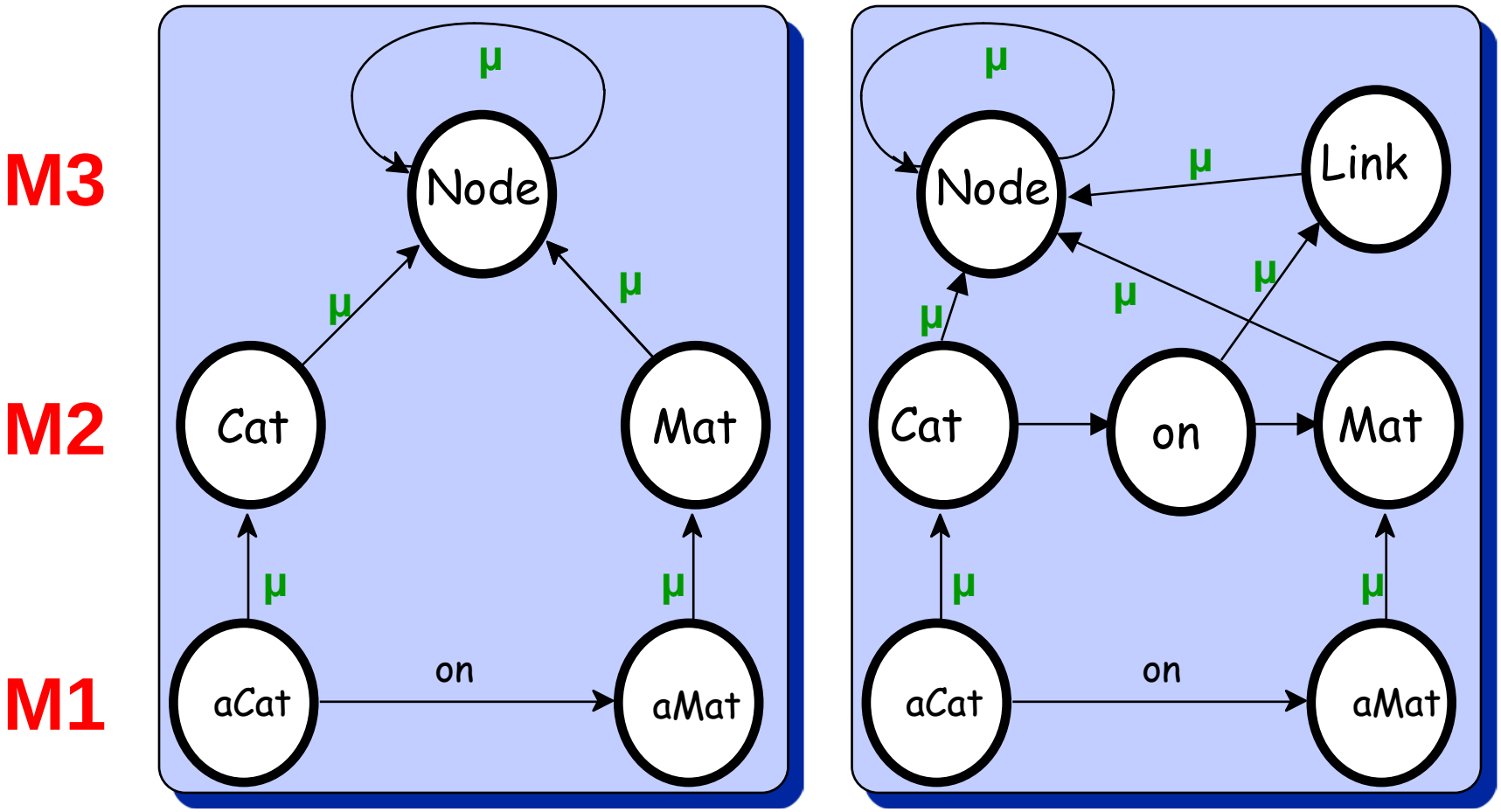


sNets :
Typed
Partitioned
Reflective
Semantic
Networks.

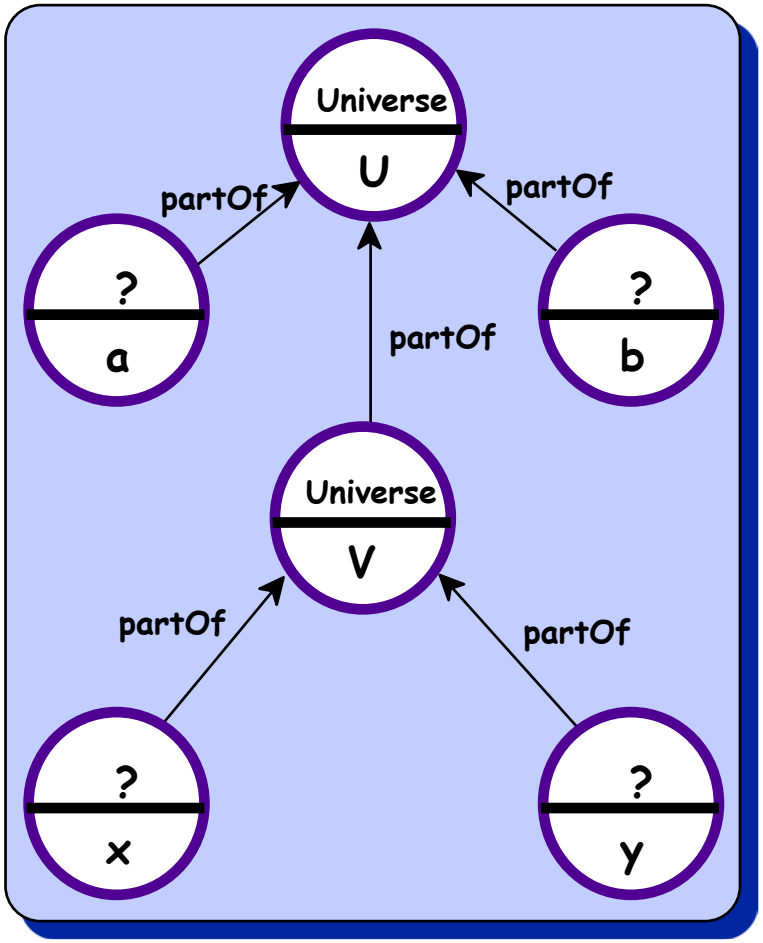


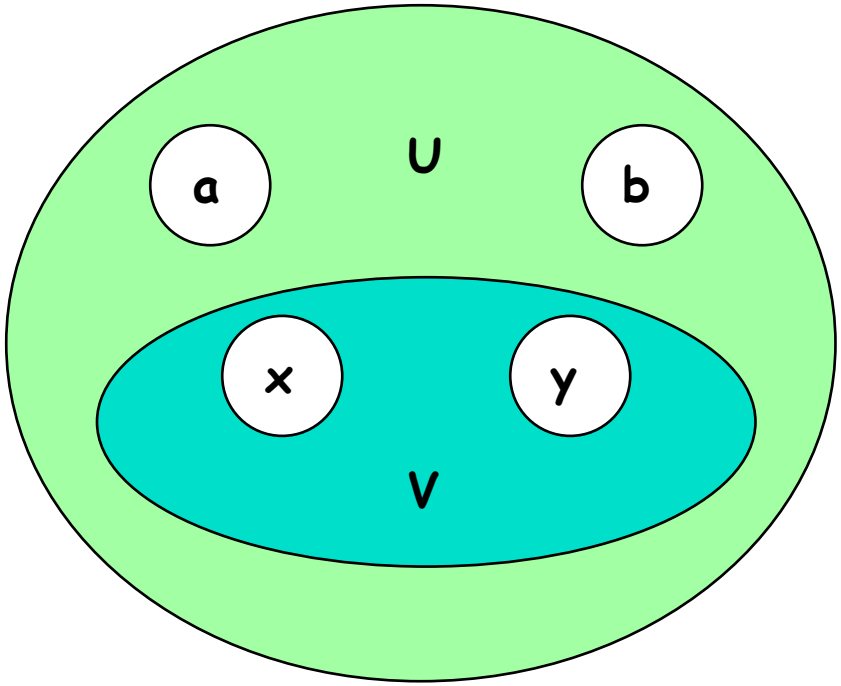
$$\exists\, x, \exists\, y : Cat\,(x) \wedge Mat\,(y) \wedge on\,(x,y)$$

A minimal M3

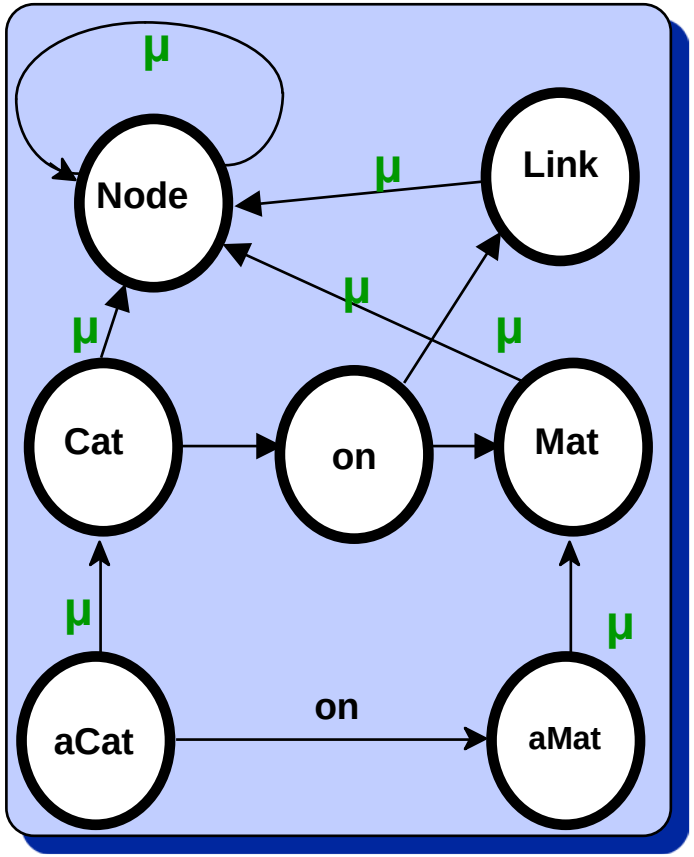


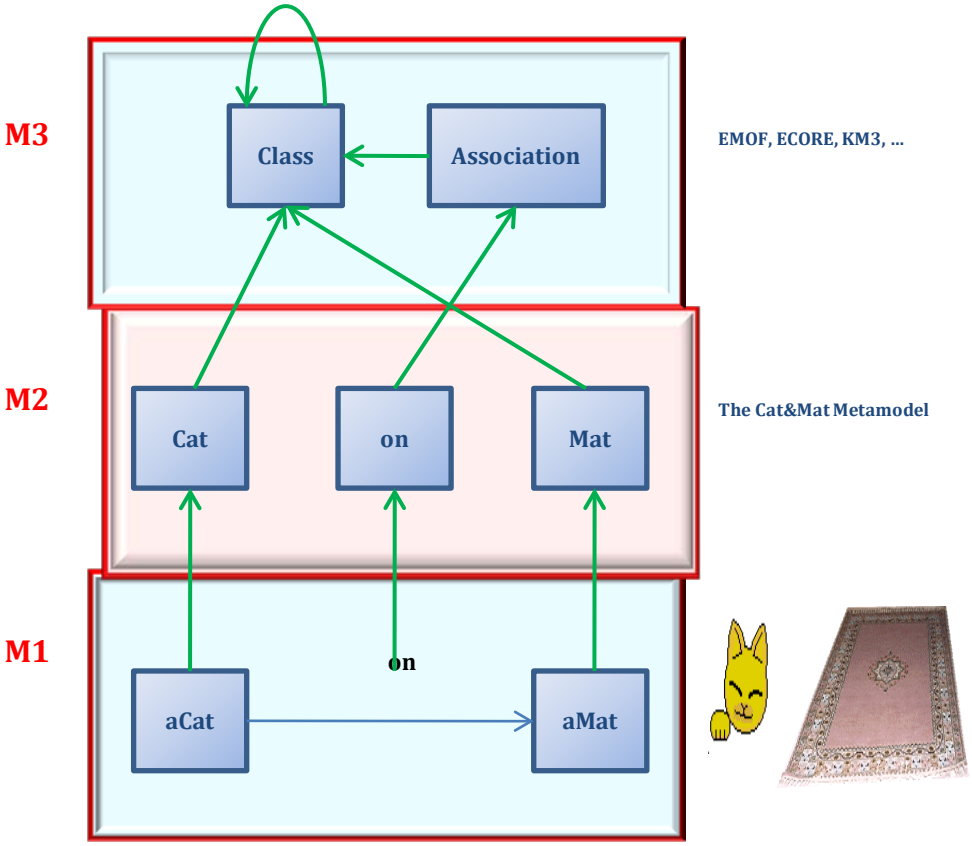
Explicit built-in modularity





Alignment on OMG standards





$\exists x, \exists y : Cat(x) \wedge Mat(y) \wedge on(x,y)$

Main messages of the lecture

- Technology evolution may be characterized by different Technical Spaces (TSs).
  - Based on one representation system, these TSs may communicate and offer various cost alternative for similar solutions.
  - MDE is one of these TSs, based on graphs and performs well in some operation, less on others
- Domain Specific Languages (DSLs) are opposed to General Purpose Languages on their special purpose and their focus on a given domain.
  - DSLs
- Metamodels are the keystone of MDE
  - They are type graphs, playing the role of grammars or Schemas in other TSs
  - The MDE TS is organized around one central representation principle (models and reference models)
  - As most other TSs, MDE is organized in three layers
- Transformations are the main operation on models
  - Various kinds of transformations
  - Purely intuitive conjecture: 80% of the needs of an IT design could be covered with pure model transformations.

Thanks

- Questions?
- Comments?

JBezivin@gmail.com