5. Model-Driven Development

Oscar Nierstrasz

Roadmap

> Introduction to Model Engineering
  - Models and metamodels
  - MDA
> Query/Views/Transformations
  - QVT languages: Relations, Core, Operational Mappings
  - Case study: flattening UML hierarchies

Sources

- Introduction to Model Engineering
  - Jean Bézivin
- Query/Views/Transformations
  - ATLAS group, INRIA & University of Nantes

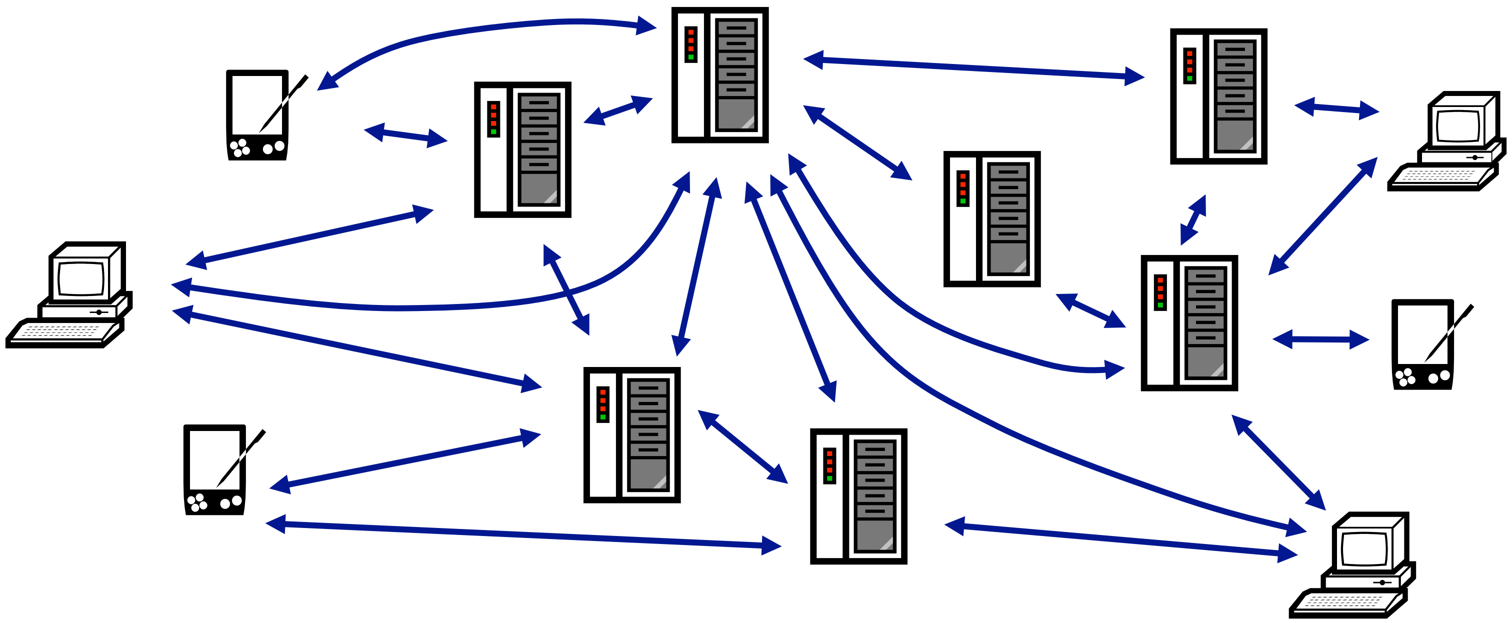
http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/ecesis-home/downloads/index.html

- Model Driven Development
  - Colin Atkinson, Universität Mannheim

Roadmap

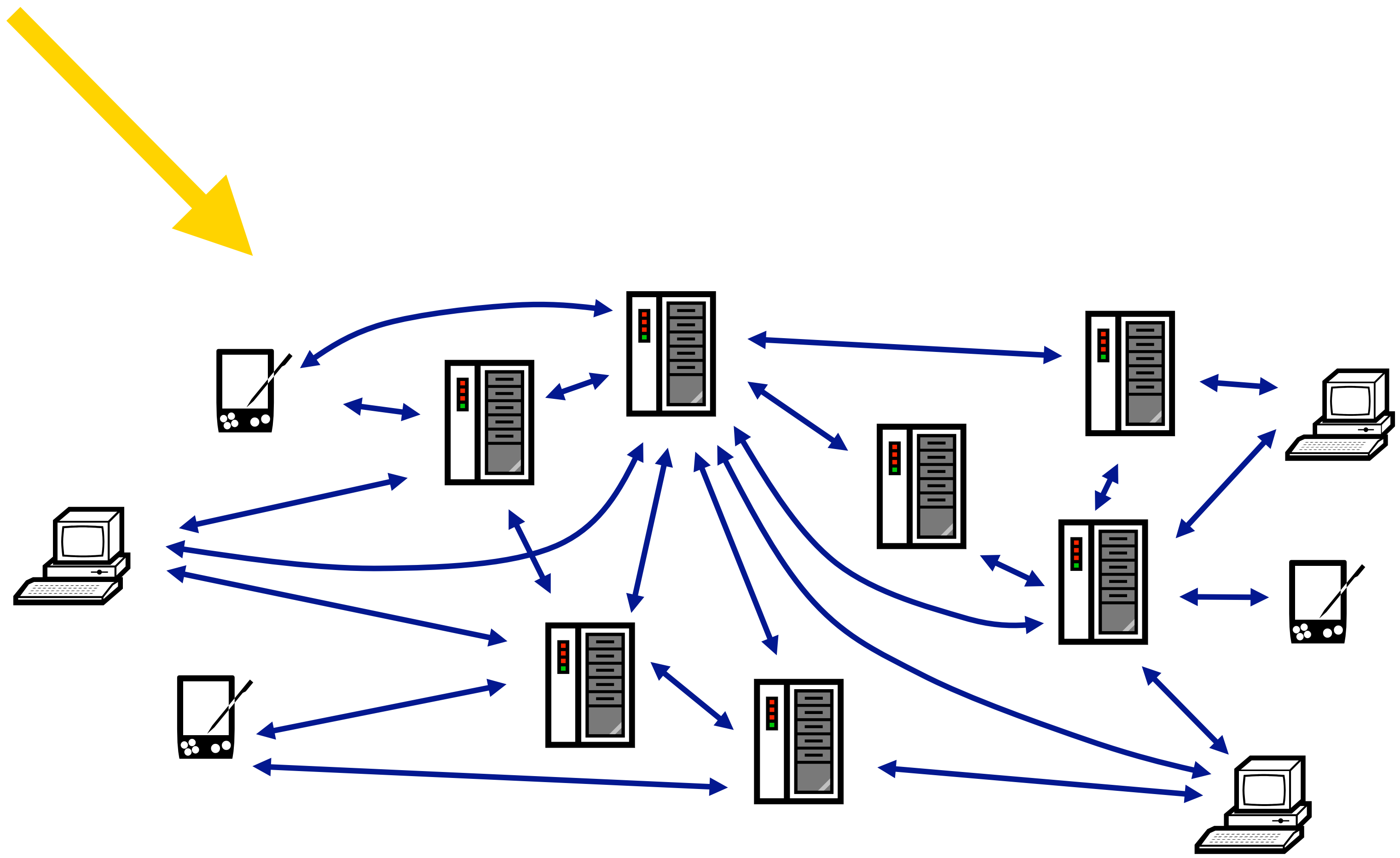
> **Introduction to Model Engineering**
  - Models and metamodels
  - MDA
> Query/Views/Transformations

The Vision of MDA

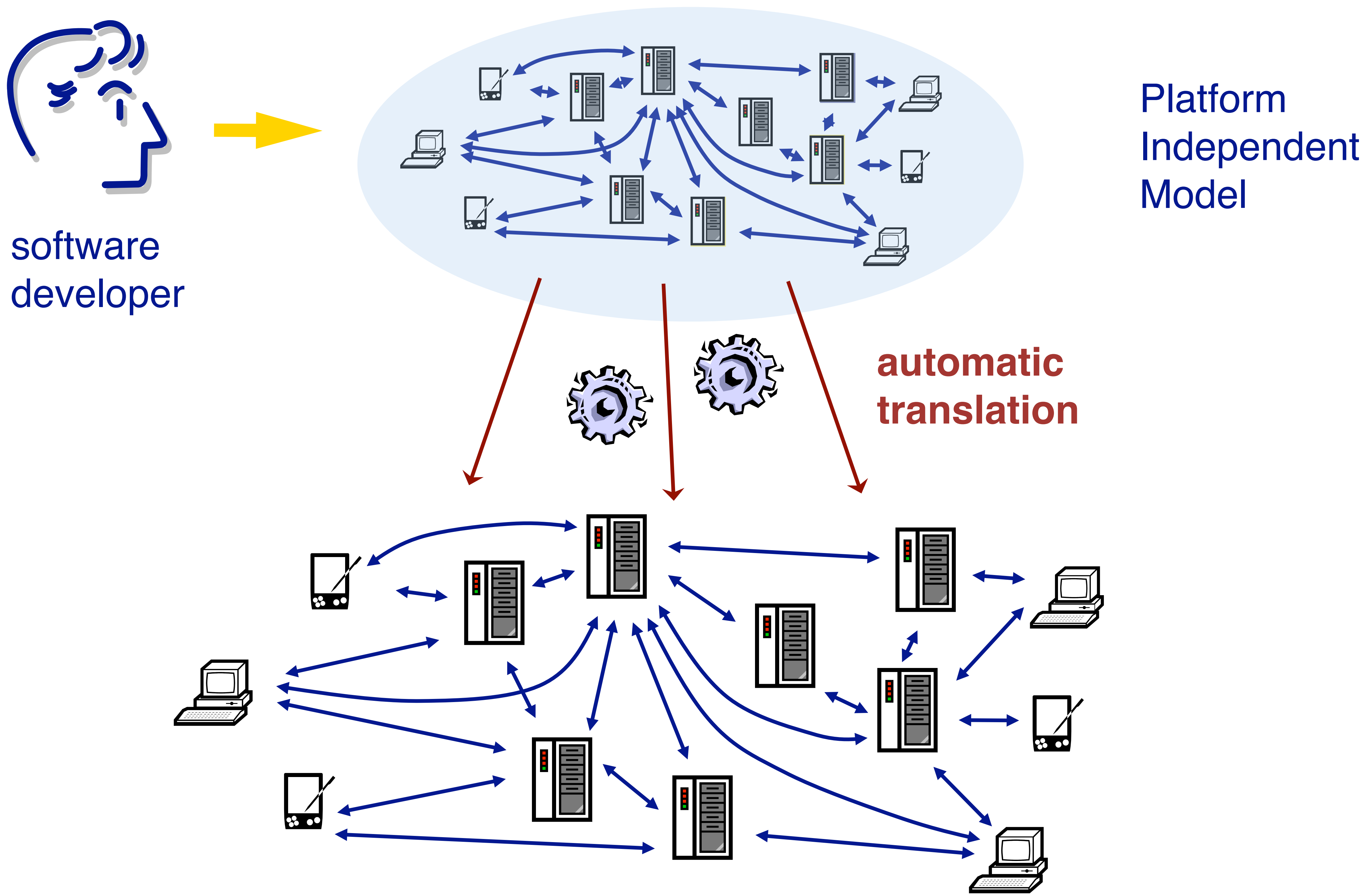


The Vision of MDA

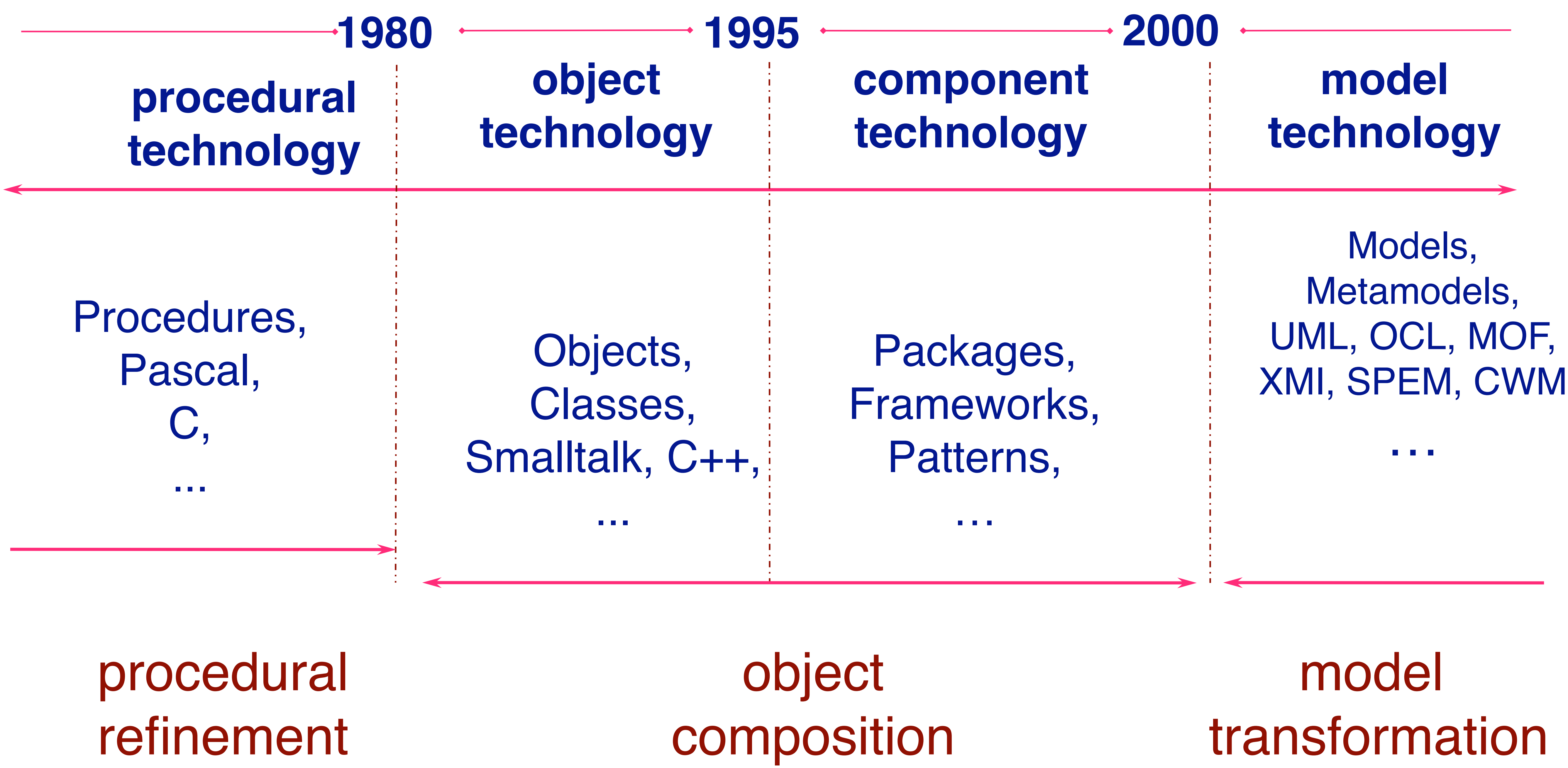
software
developer



The Vision of MDA



A global view of software engineering evolution



Roadmap

> Introduction to Model Engineering
  — **Models and metamodels**
  — MDA
> Query/Views/Transformations

Modeling is essential

Modeling is essential to human activity because every action is preceded by the construction (implicit or explicit) of a model.

> The medical technique of *bloodletting* was based on an *incorrect model of the body* [1]. If the model is incorrect, the action may be inappropriate [2].

Hippocrates and others believed that the four elements earth, air, water and fire were balanced within the human body as the four humors: blood, phlegm, and black and yellow bile. Disease was due to an imbalance in the four humors and treatment involved restoring their balance through bloodletting.

Georges Washington died after heavy blood loss sustained in a bloodletting treatment for laryngitis.

What is a model?

A <u>model</u> is a *representation of a system*

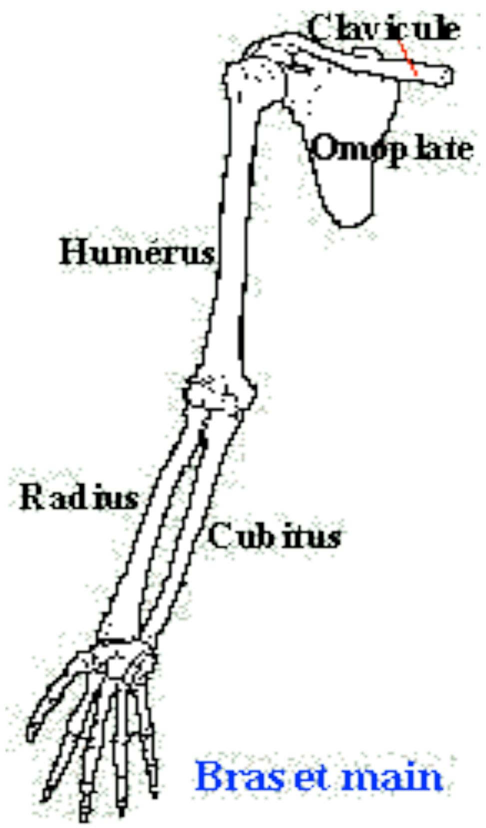
- A model is written in the language of its unique metamodel
- A metamodel is written in the language of its unique metametamodel
  - *The unique MMM of the MDA is the MOF*
- A model is a constrained directed labeled graph
- A model may have a visual graphical representation (sometimes)

A model is a partial view of a system

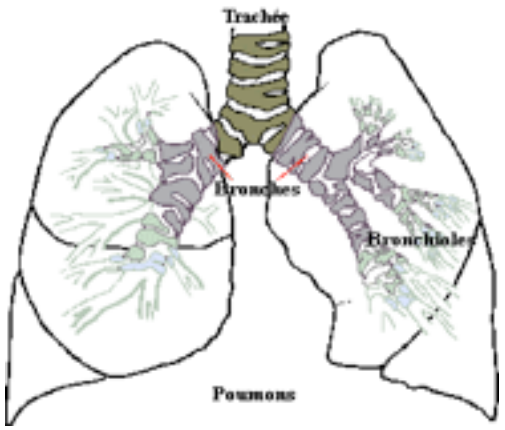
A system

repOf

Several models
of this system
(partial views)



Skeleton model

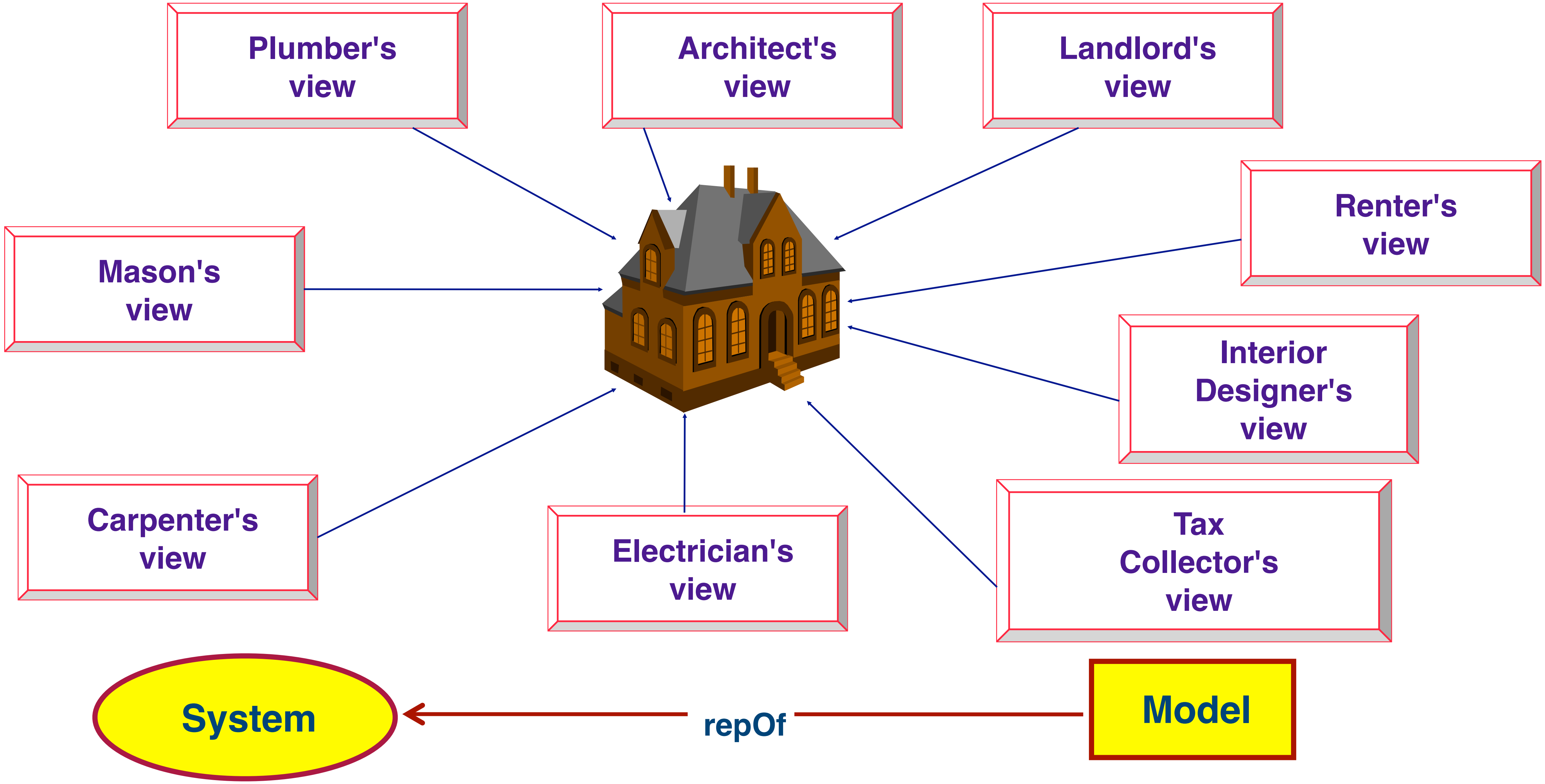


Respiratory model

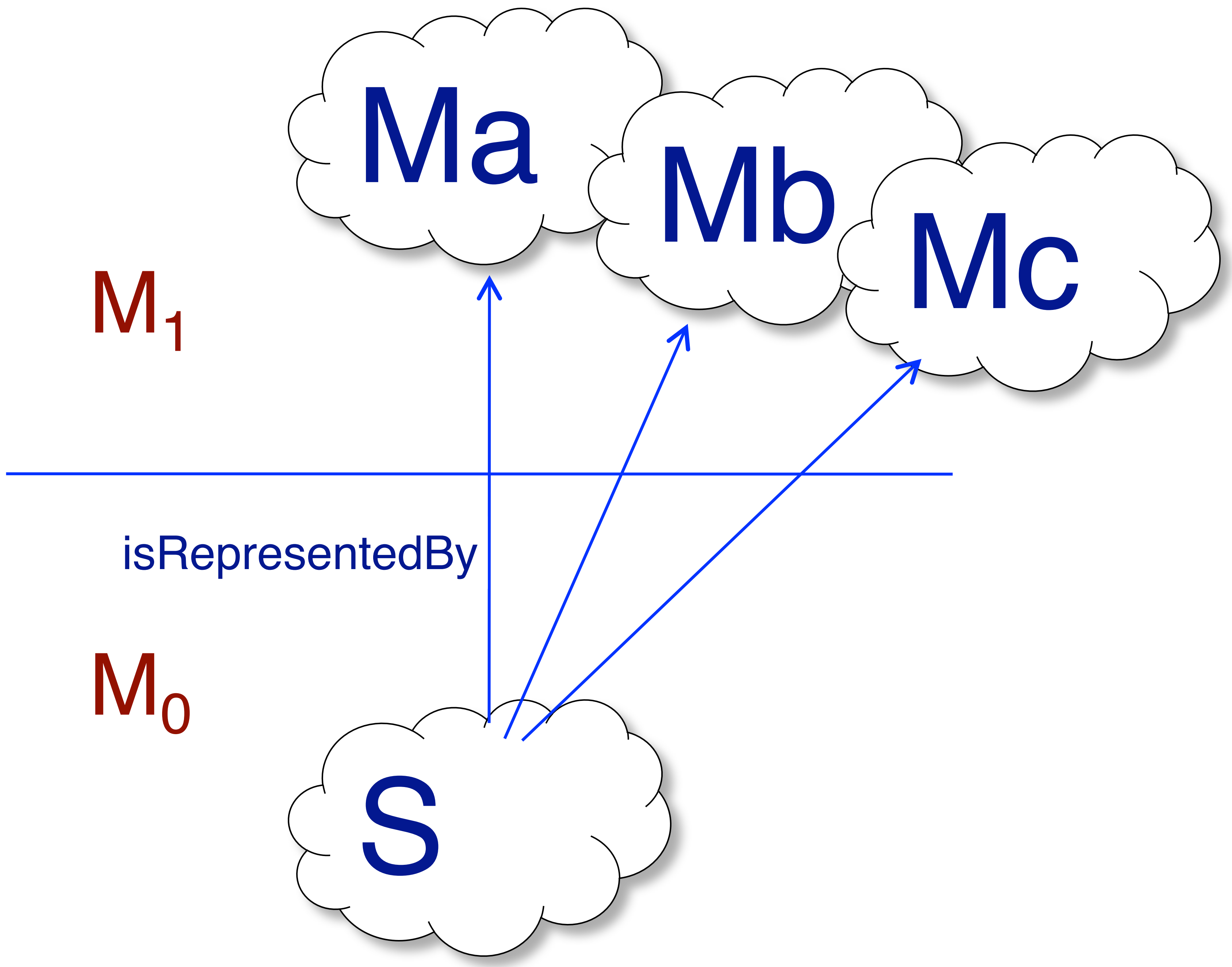
Other
Models
muscular,
nervous,
circulatory,
digestive,
endocrinous,
etc.

Multiples views and coordinated DSLs

Each view is expressed in a given domain language (DSL).
Vocabularies of different corporations are different;
However they allow talking about a common building.



Aspects of a system represented by models



A given system may have plenty of different models.

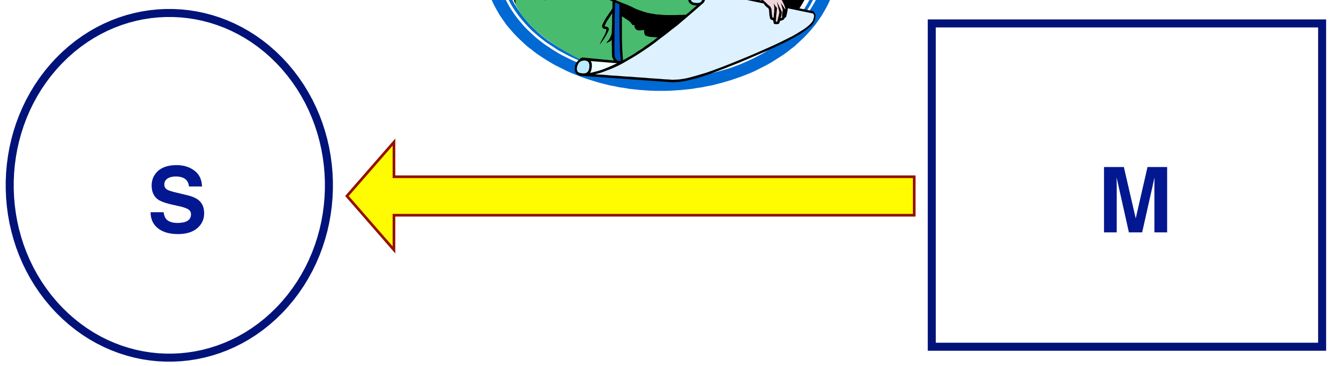
Each model represents a given *aspect* of the system.

Don't confuse the model and the system

Roadmap

> Introduction to Model Engineering
  — Models and metamodels
  — **MDA**
> Query/Views/Transformations

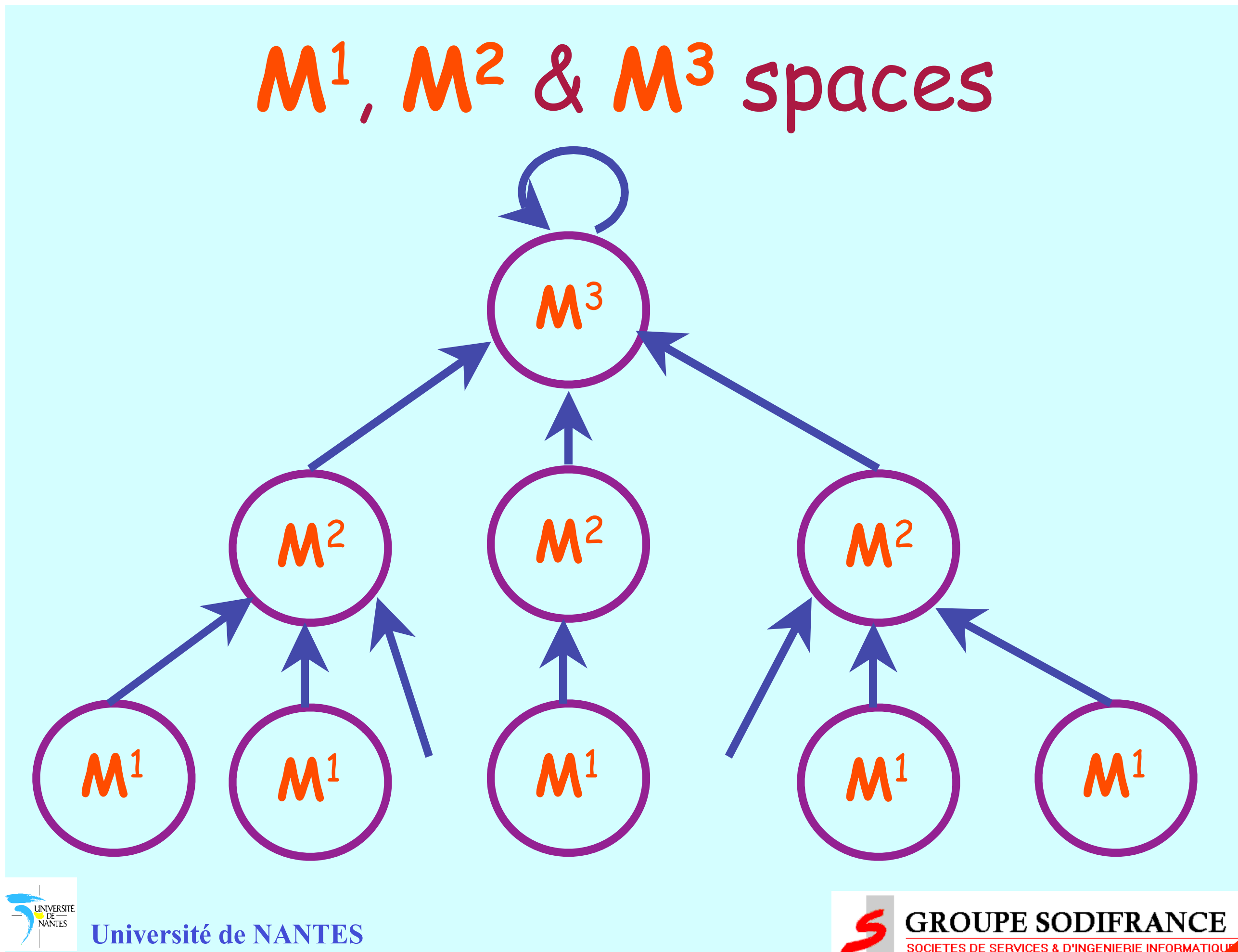
Production of a system from a model

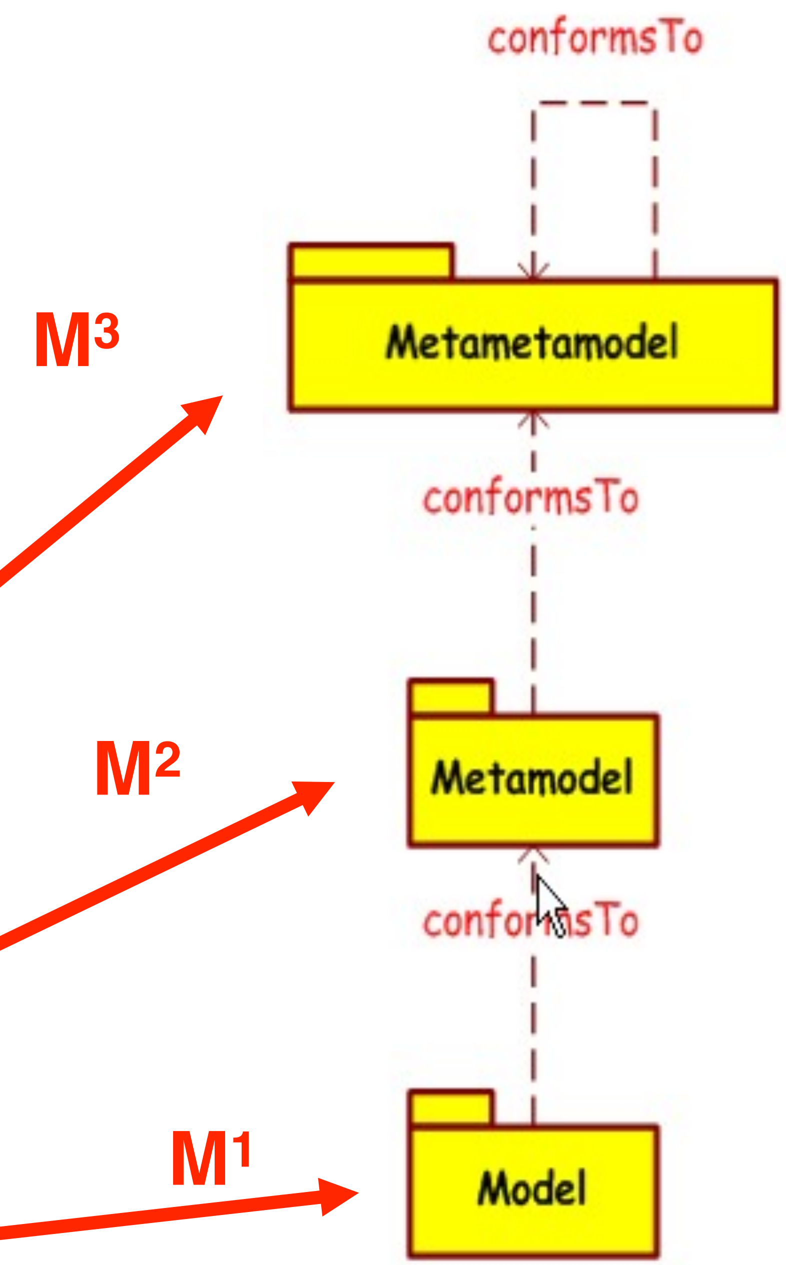


Production of a system from a model.
For example construction a building from its floor plans.

{postcondition}

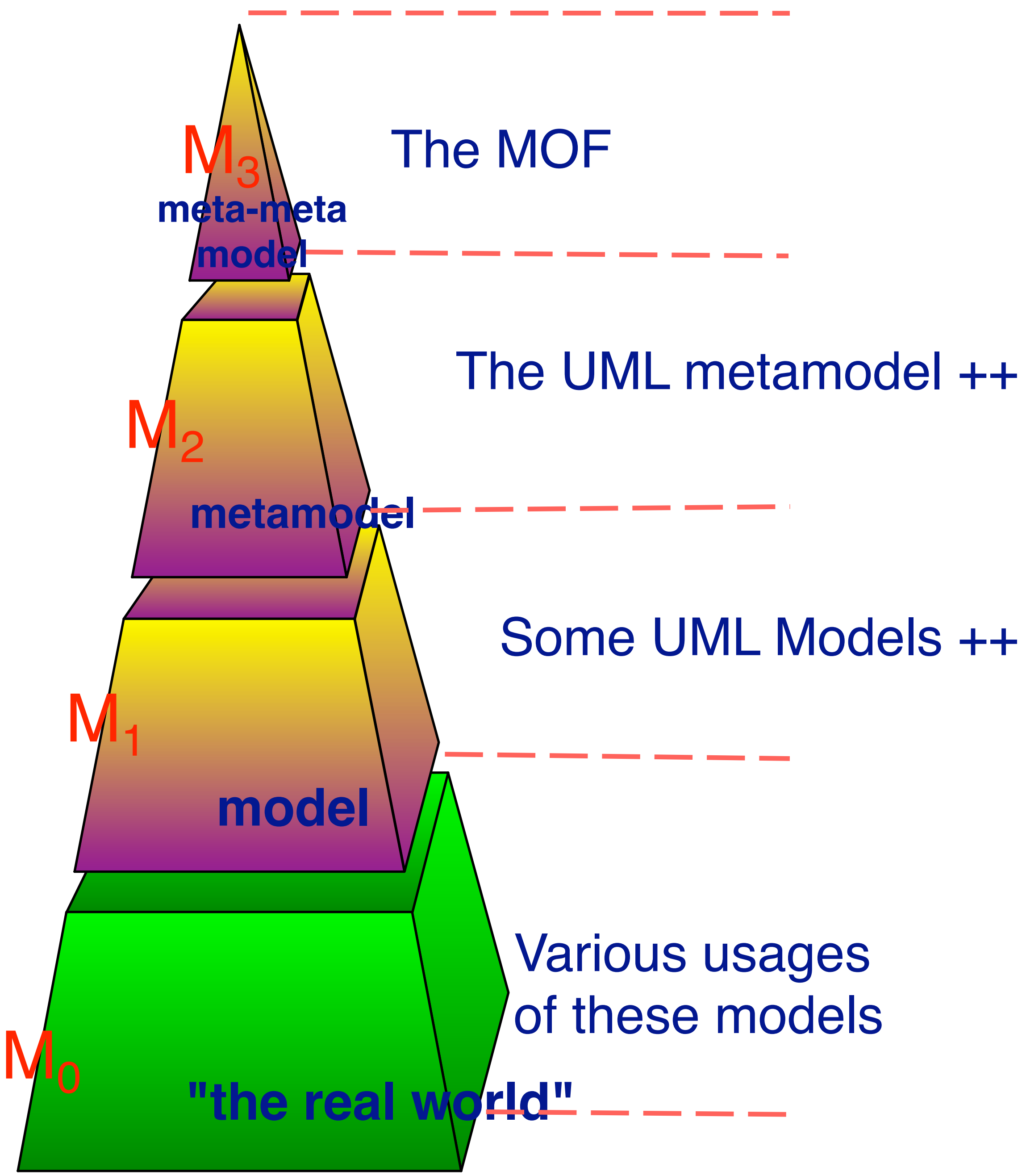
MDA in a nutshell

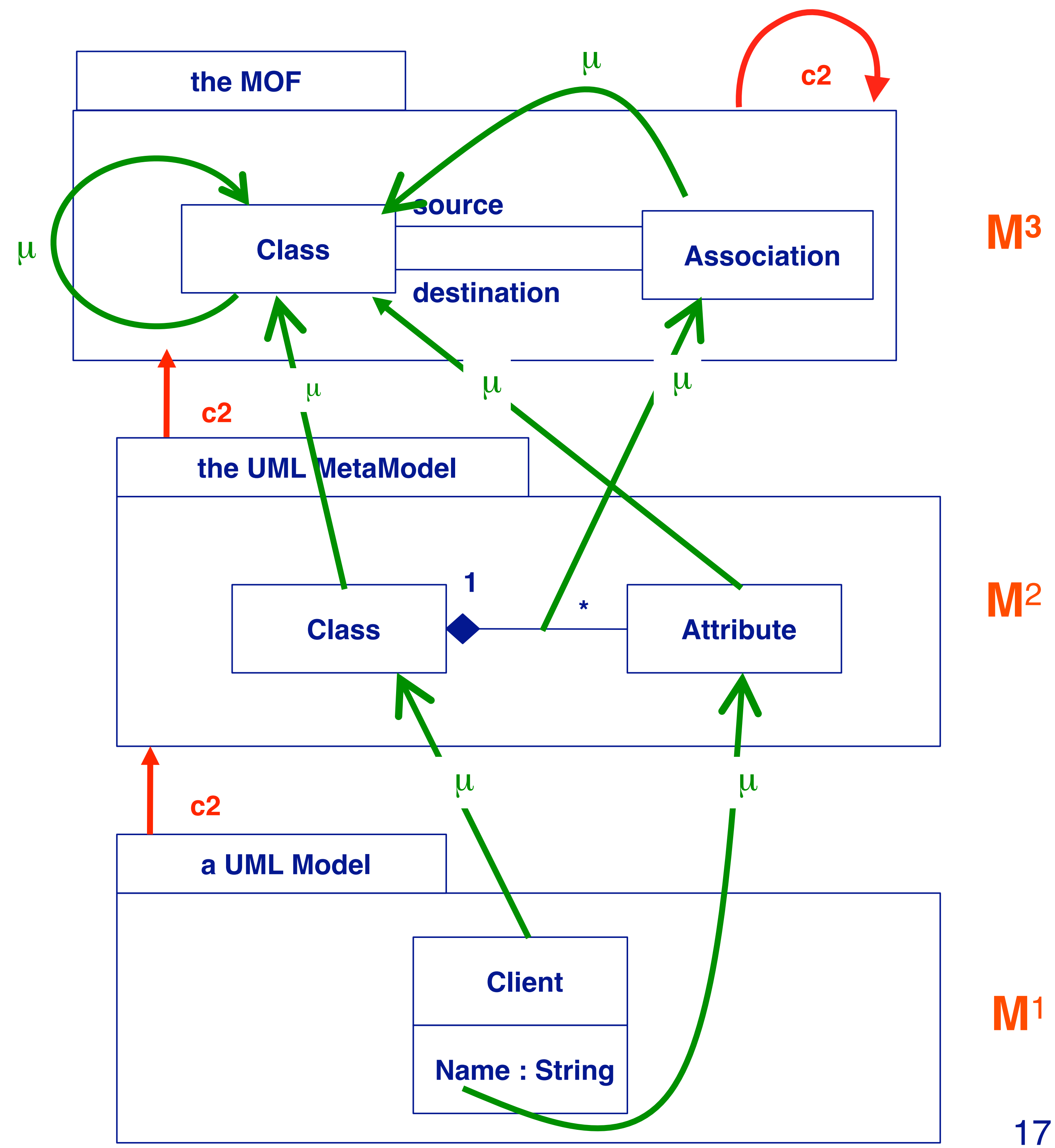




- One unique Metametamodel (the MOF)
- An important library of compatible Metamodels, each defining a DSL
- Each of the models is defined in the language of its unique metamodel

The OMG / MDA Stack

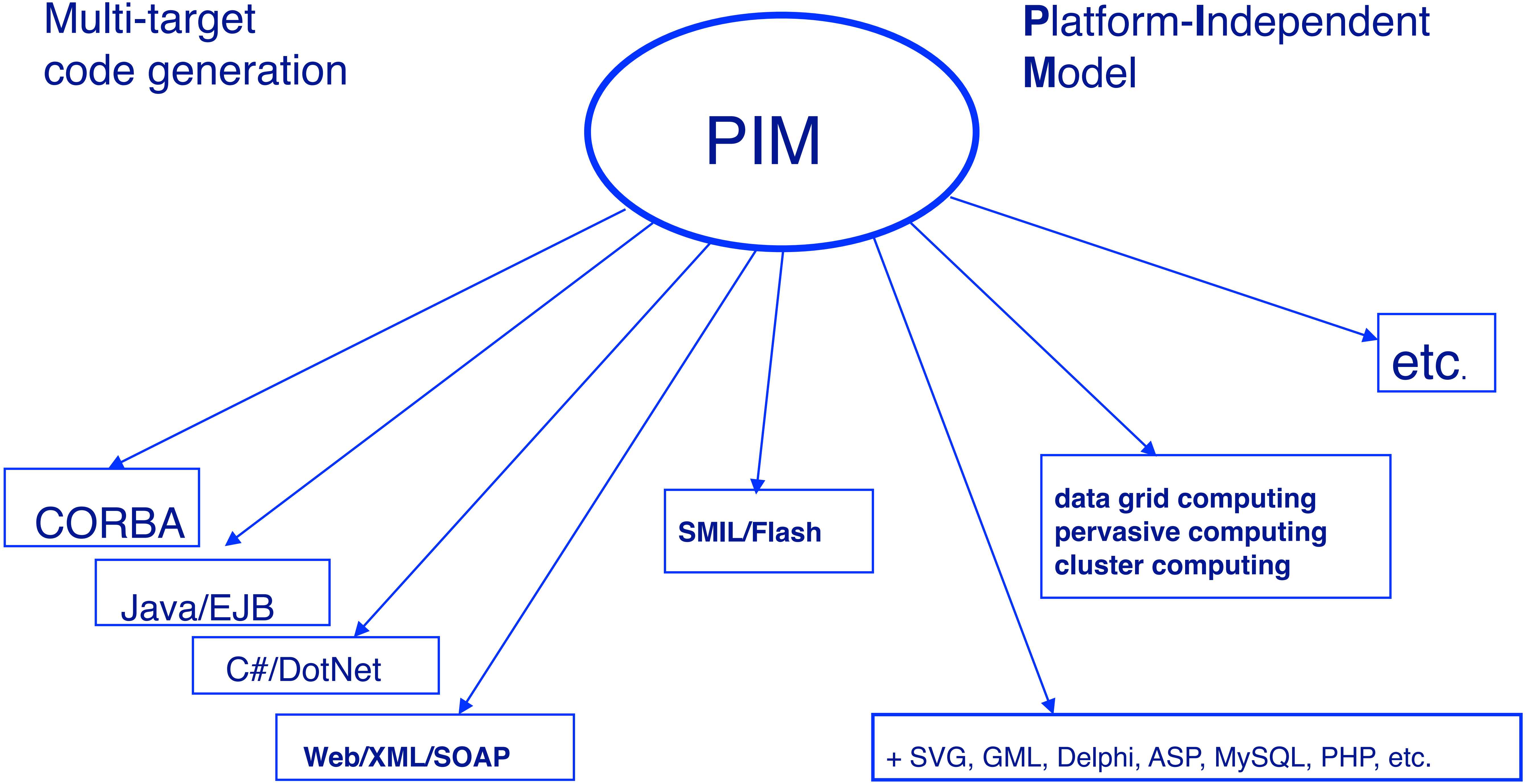




Write Once, Run Anywhere
Model Once, Generate Anywhere

Multi-target
code generation

Platform-**I**ndependent
Model



Roadmap

> Introduction to Model Engineering
> **Query/Views/Transformations**
  — QVT languages: Relations, Core, Operational Mappings
  — Case study: flattening UML hierarchies

Overview

- QVT stands for **Q**uery/**V**iews/**T**ransformations
- OMG standard language for expressing queries, views, and transformations on MOF models
  - OMG QVT Request for Proposals (QVT RFP, ad/02-04-10) issued in 2002
  - Seven initial submissions that converged to a common proposal
  - Current status (June, 2006): final adopted specification, OMG document ptc/05-11-01

Roadmap

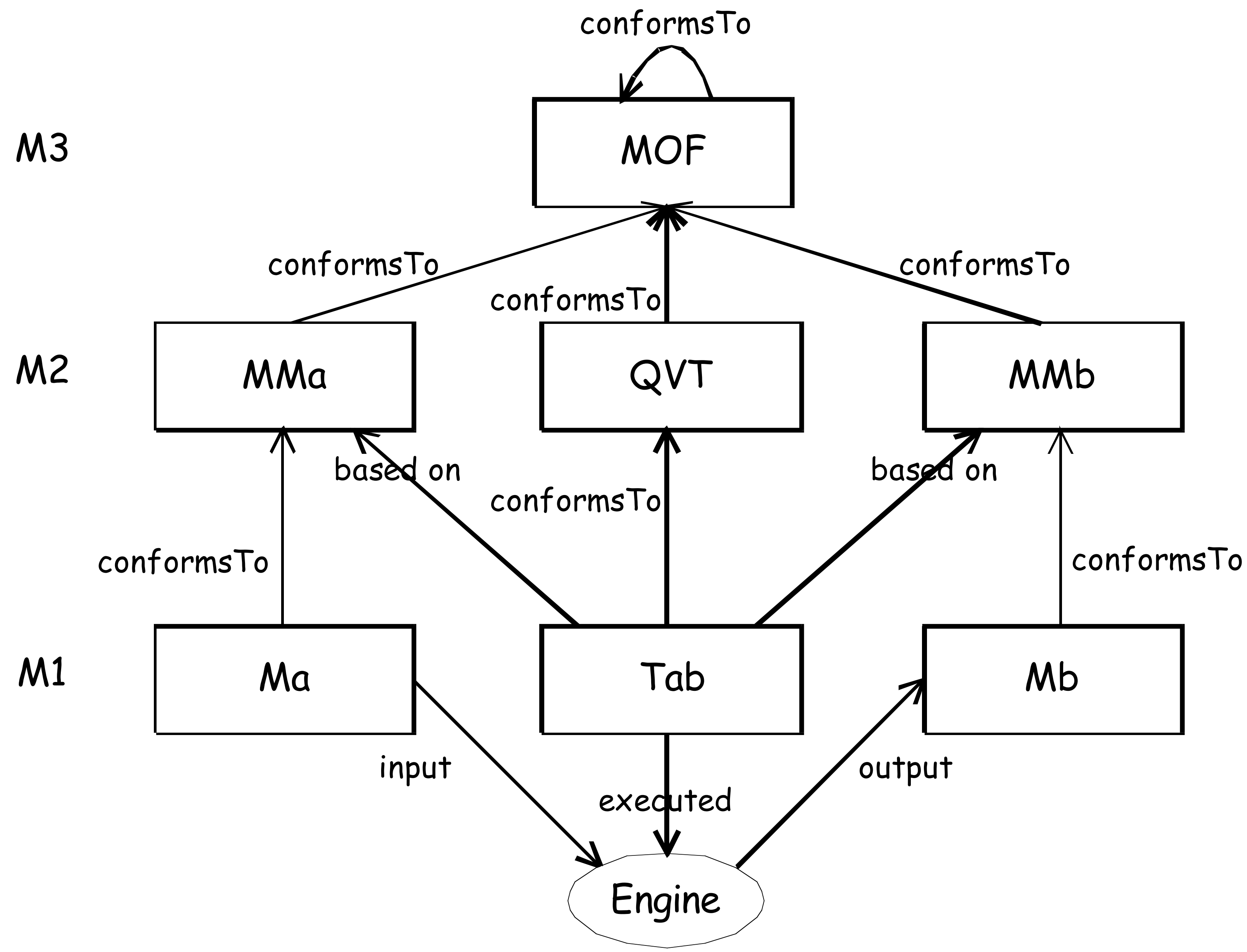
> Introduction to Model Engineering
> Query/Views/Transformations
  — **QVT languages: Relations, Core, Operational Mappings**
  — Case study: flattening UML hierarchies

QVT Operational Context

> Abstract syntax of the language is defined as MOF 2.0 metamodel

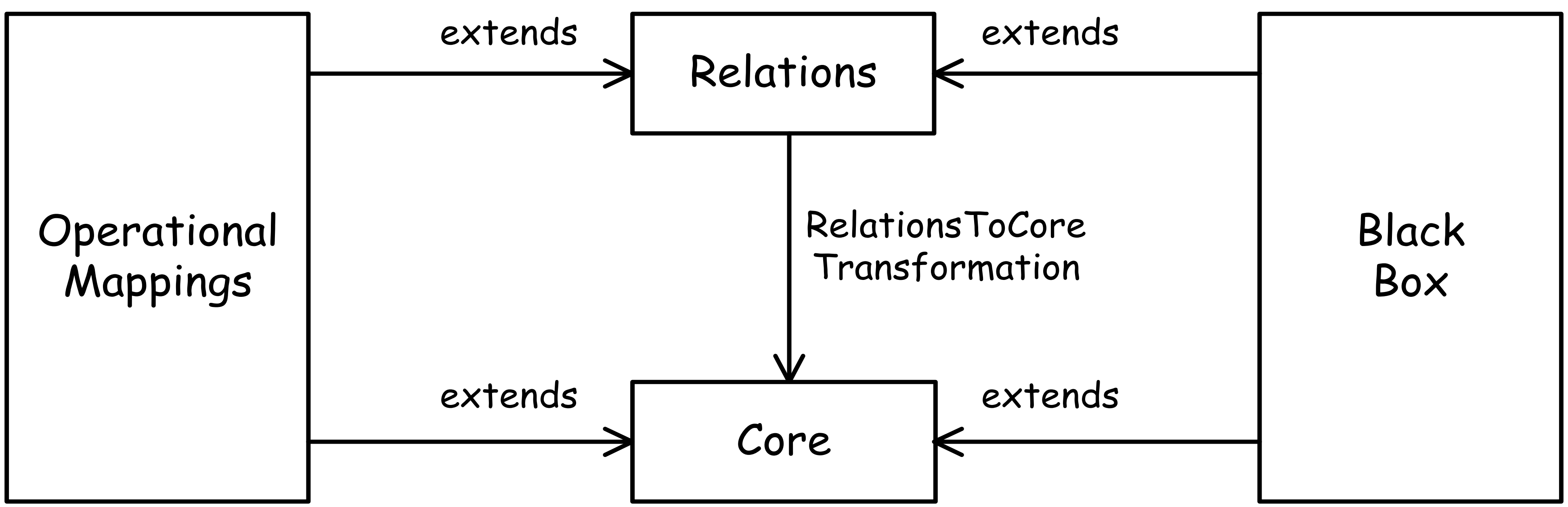
> Transformations (*Tab*) are defined on the base of MOF 2.0 metamodels (*MMa*, *MMb*)

> Transformations are executed on instances of MOF 2.0 metamodels (*Ma*)



QVT Architecture

- Layered architecture with three transformation languages:
  - Relations
  - Core
  - Operational Mappings
- Black Box is a mechanism for calling external programs during transformation execution



QVT Languages

> Relations
  — Declarative transformation language
  — Specification of relations over model elements
> Core
  — Declarative transformation language
  — Simplification of Relations language
> Operational Mappings
  — Imperative transformation language
  — Extends Relations language with imperative constructs

QVT is a set of three languages that collectively provide a hybrid “language”.

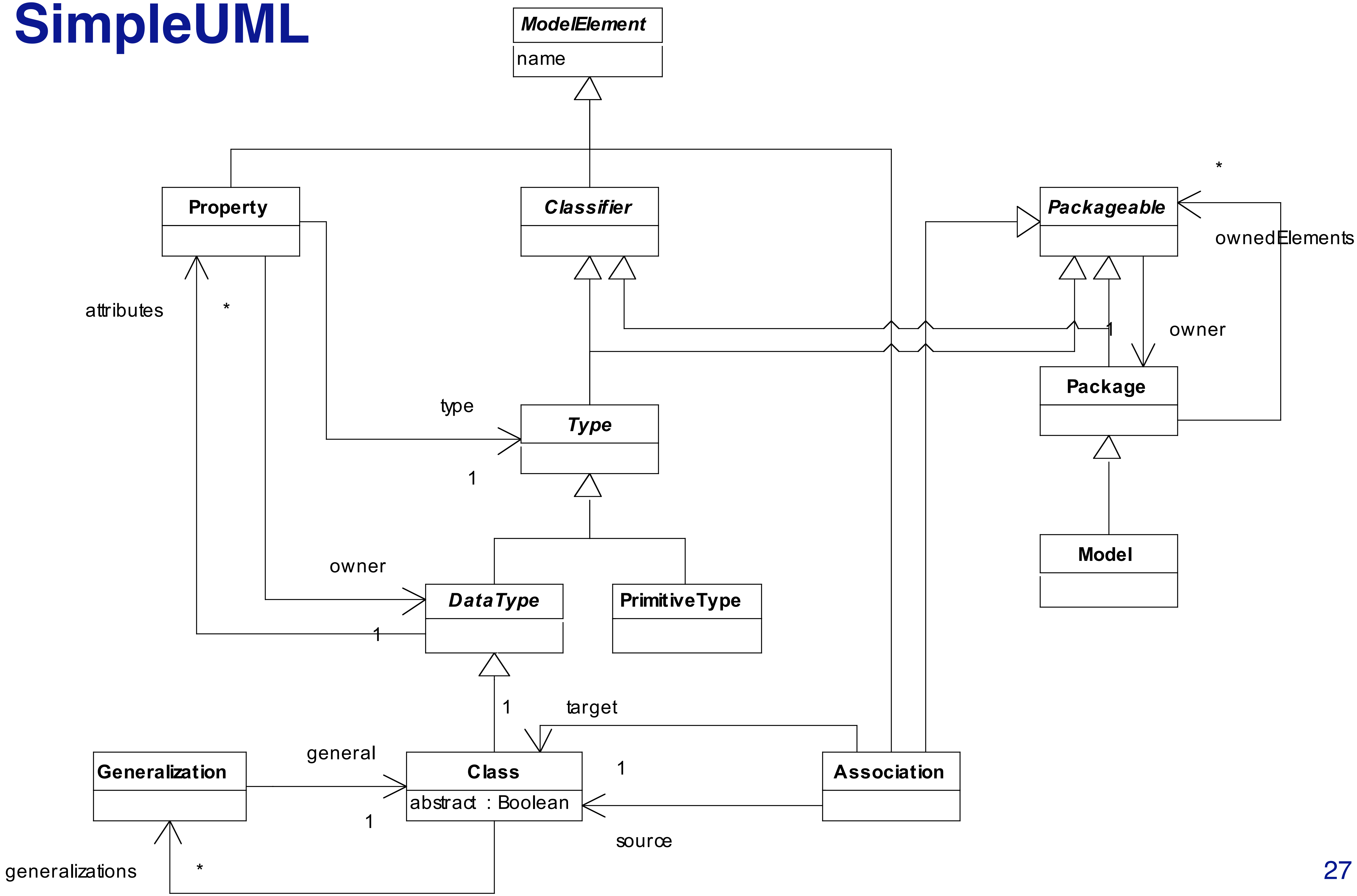
Roadmap

- Introduction to Model Engineering
- Query/Views/Transformations
  - QVT languages: Relations, Core, Operational Mappings
  - **Case study: flattening UML hierarchies**

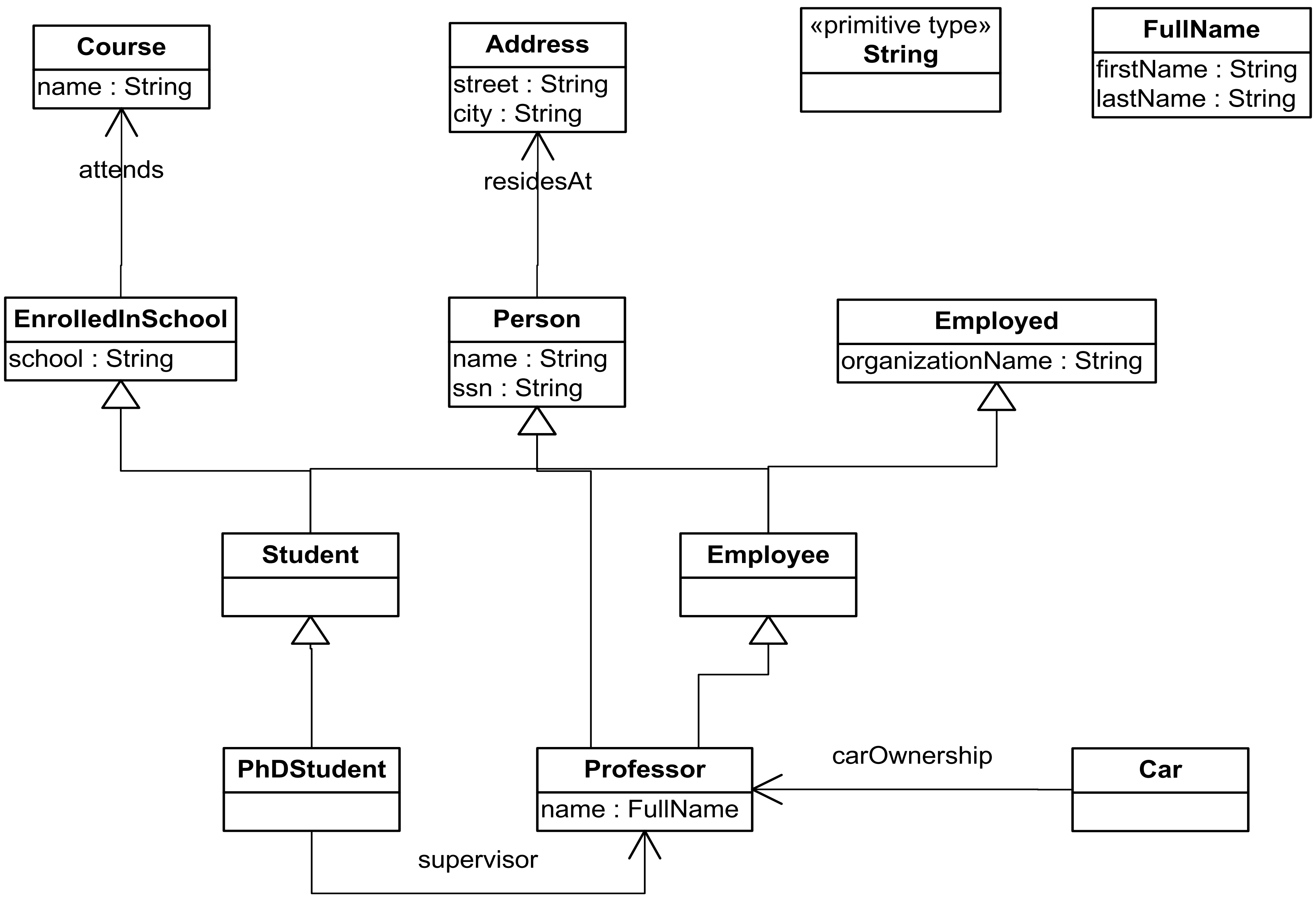
Case Study

- Flattening UML class hierarchies:
  - given a source UML model transform it to another UML model in which only the leaf classes (classes not extended by other classes) in inheritance hierarchies are kept.
- Rules:
  - Transform only the leaf classes in the source model
  - Include the inherited attributes and associations
  - Attributes with the same name override the inherited attributes
  - Copy the primitive types

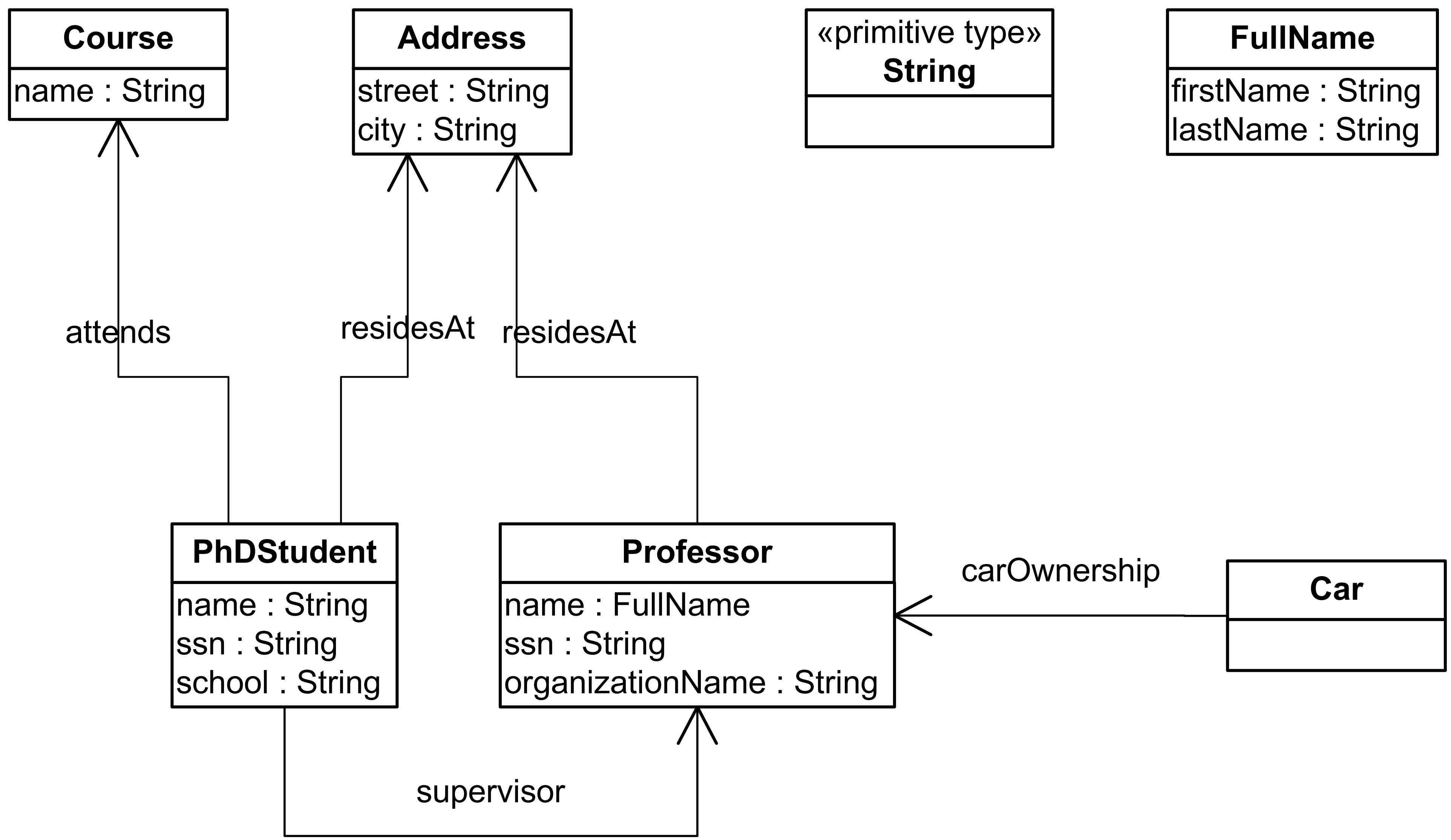
Source and Target Metamodel: SimpleUML



Example Input Model



Example Output Model



Model Transformation expressed in Operational Mappings Language

Overall structure of a transformation program:

```
transformation SimpleUML2FlattenSimpleUML(in source : SimpleUML
                                          out target : SimpleUML);
…………………………………………………………………

main() {}

…………………………………………………………………
…helpers………………………………………………
…mapping operations………………
```

Signature:
Declares the transformation name and the source and target metamodels.

in and **out** keywords indicate source and target model variables.

Entry point:
The execution of the transformation starts here by executing the operations in the body of **main**

Transformation elements:
Transformation consists of *mapping operations* and *helpers*. They form the transformation logic.

Mapping Operations

- A mapping operation maps one or more source elements into one or more target elements
  - Always unidirectional
  - Selects source elements on the base of a type and a Boolean condition (guard)
  - Executes operations in its body to create target elements
  - May invoke other mapping operations and may be invoked
  - Mapping operations may be related by inheritance

Mapping Operations: Example (1)

> Consider the rule that transforms only leaf classes
  - Selects only classes without subclasses
  - Collects all the inherited properties
  - Creates new class in the target model

Signature and guard

```
mapping Class::leafClass2Class(in model : Model) : Class
         when {not model.allInstances(Generalization)->exists(g |
         g.general = self)}
{
  name:= self.name;
  abstract:= self.abstract;
  attributes:=
        self.derivedAttributes()->map property2property(self)->asOrderedSet();
}
```

Operation body

Mapping Operations: Example (2)

Operation Signature and Guard

Source element type

Operation name

An optional sequence of input/output parameters

Target element type

```
mapping Class::leafClass2Class(in model : Model) : Class
        when {not model.allInstances(Generalization)->exists(g |
        g.general = self)}
```

Guard

The Guard is an OCL expression used to filter source elements of a given type. The mapping operation is executed only on elements for which the guard expression is evaluated to true.

Mapping Operations: Example (3)

Operation Body

The predefined variable ***self*** refers to the source element on which the operation is executed

The keyword ***map*** is used to invoke another mapping operation named *property2property* over the elements returned by the helper *derivedAttributes*

```
name:= self.name;
abstract:= self.abstract;
attributes:=
    self.derivedAttributes()->map property2property(self)->asOrderedSet();
```

The left-hand side of the assignments denotes properties of the target element

Invocation of helper *derivedAttributes*

The mapping operation body contains initialization expressions for the properties of the target element. When an operation is executed over a source element the *self* variable is bound to it and an instance of the target type is created. Then the operation body is executed.

Conclusions (1)

- QVT: Query/Views/Transformations – the OMG standard language for model transformations in MDA/MDE
- The issue of Views over models is not addressed
- Query language based on OCL
- A family of three transformation languages:
  - Relations: declarative language
  - Core: declarative language, simplification of Relations
  - Operational Mappings: imperative transformation language that extends relations
- Collectively QVT languages form a hybrid language

Conclusions (2)

- Tool support is still insufficient (at the time of preparing of this lecture – June 2006) [still true in 2011!]
- QVT is not proved yet in non-trivial industrial like scenarios
- Many issues need further exploration:
  - Performance
  - Testing
  - Scalability of transformations
  - Ease of use
  - Handling change propagation
  - Incremental transformations
  - Adequacy of the reuse mechanisms

License

- The present courseware has been elaborated in the context of the MODELWARE European IST FP6 project (http://www.modelware-ist.org/).
- Co-funded by the European Commission, the MODELWARE project involves 19 partners from 8 European countries. MODELWARE aims to improve software productivity by capitalizing on techniques known as Model-Driven Development (MDD).
- To achieve the goal of large-scale adoption of these MDD techniques, MODELWARE promotes the idea of a collaborative development of courseware dedicated to this domain.
- The MDD courseware provided here with the status of open source software is produced under the EPL 1.0 license.

http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/ecesis-home/downloads/index.html