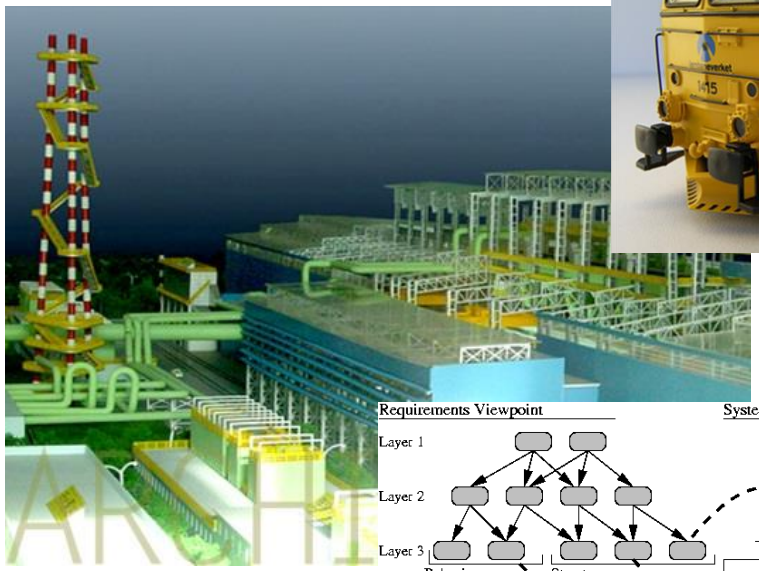
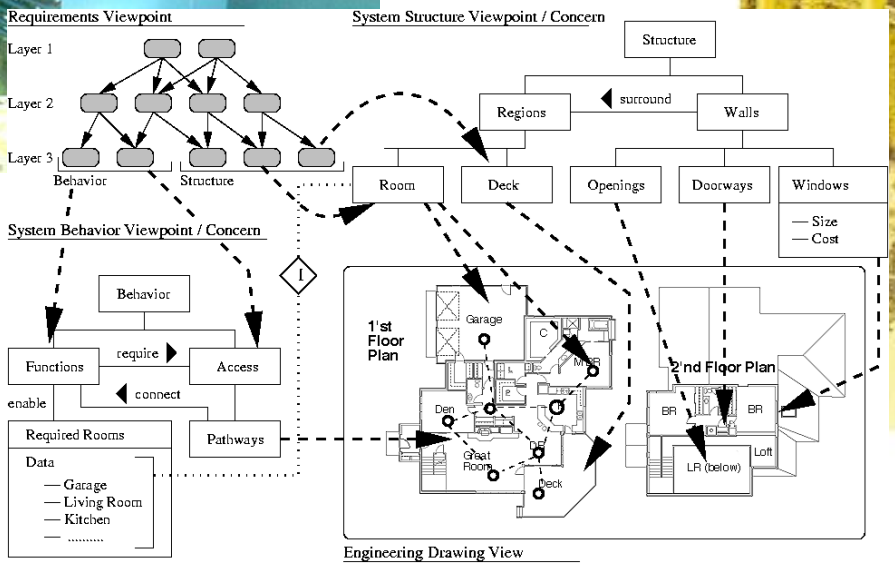
System Modeling and Model-Driven Engineering

MODELS

http://www.flickr.com/photos
/navarzo4/6450276881

http://www.archimod.com





http://www.isr.umd.edu/
~austin/paladinRM/future-work.gif

Bran Selić, SDL Forum 2013 Keynote:
”Model-Based Software Engineering in Industry: Revolution, Evolution, or Smoke?””

What is a Model?

"A model is an abstraction of a (real or language-based) system allowing predictions or inferences to be made."

Thomas Khune, Matters of (Meta-) Modeling, SoSYM, 2006

"no abstraction" → "no model"

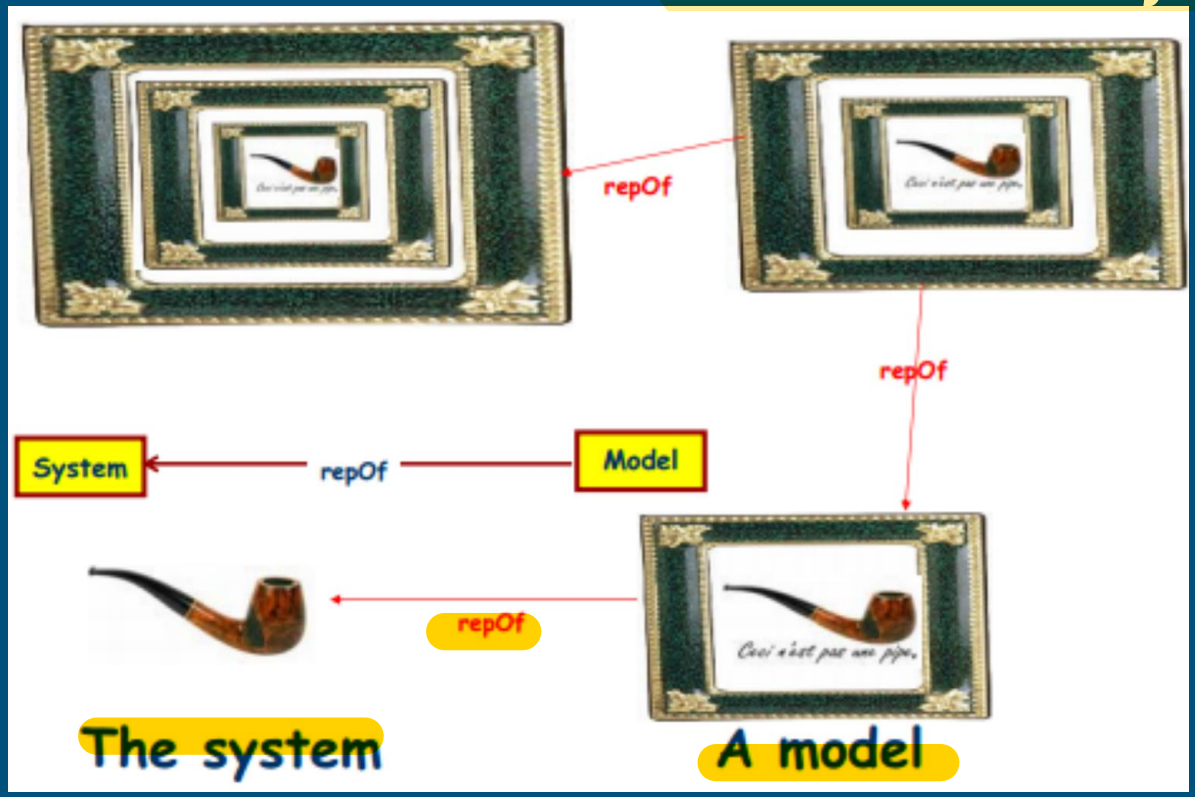
“no abstraction” → “no model”

- A copy is not a model
- Any representation of a real world subject automatically implies reduction and thus can be granted model status.

René Magritte

1928–29

Difference between a model and System



Jean-Marie Favre:
Megamodelling and Etymology. Transformation Techniques in Software Engineering 2005

Why Engineers Build Models

- Engineering model: a selective representation of some system that captures accurately and concisely all of its essential properties of interest for a given set of concerns
- We don't see everything at once; what we do see is adjusted to human needs and understanding
- Reducing complexity to the human scale

http://commons.wikimedia.org

Bran Selić, SDL Forum 2013 Keynote:
"Model-Based Software Engineering in Industry: Revolution, Evolution, or Smoke?"

Purpose of Engineering Models

- Descriptive models:

  To help us understand (i.e., reason about) complex systems

  To communicate understanding and design intent to others

  To predict the interesting characteristics of systems

- Prescriptive models:

  To specify what systems must do

Bran Selić, SDL Forum 2013 Keynote:
”Model-Based Software Engineering in Industry: Revolution, Evolution, or Smoke?”

Modeling vs. Programming Languages

- The primary purpose and focus of programming languages is implementation:
  - The ultimate form of prescription
  - Implementation requires total precision and “full” detail
  - Prescription takes precedence over description
- To be useful for humans, a modeling language must support description as a first-order concern:
  - I.e., communication, prediction, and understanding
  - These generally require omission of “irrelevant” detail such as details of the underlying computing technology used to implement the software

Bran Selić, SDL Forum 2013 Keynote:
”Model-Based Software Engineering in Industry: Revolution, Evolution, or Smoke?”

Desired Characteristics of Models

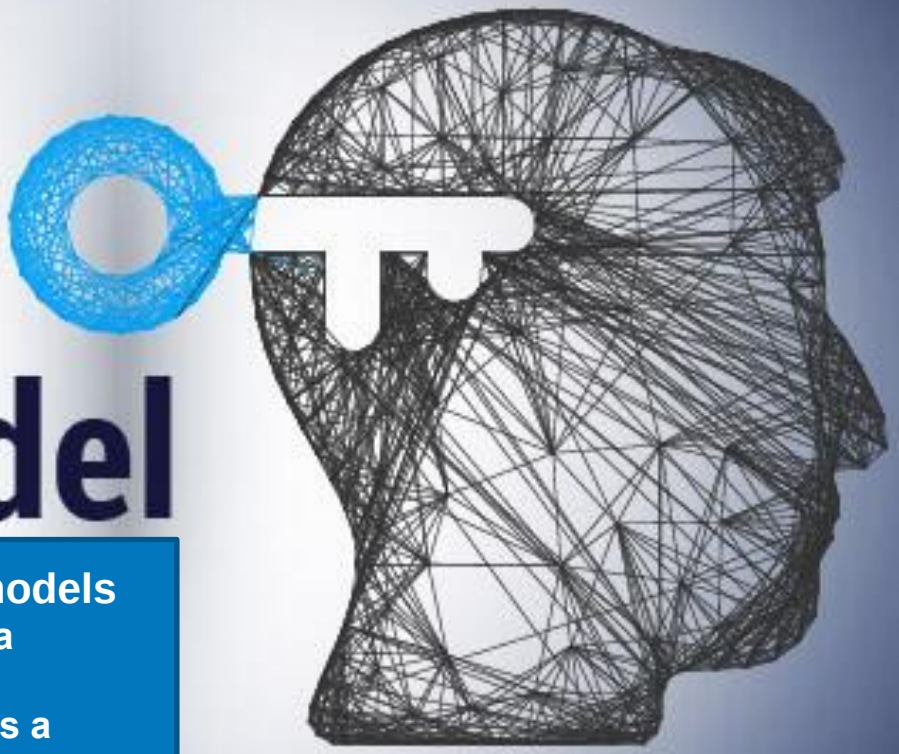
- Abstraction: emphasize important aspects and ignore the irrelevant ones
- Understandability: easy to understand by users
- Accuracy: correctly represent the modeled system for intended purpose
- Prediction: use them to answer questions about the modeled system to detect errors and omissions and determine the most important tradeoffs in complex designs before committing resources for their realization
- Low cost: cheaper to build and study

Usefulness of Models

- Understanding the problem (or reality)
- Communication among stakeholders
  - Customers, users, developers…
- Controlling complexity
  - Abstraction
  - Through analysis (formal) and experimentation
  - Investigate and compare alternative solutions
  - Minimizing risks
- Developing (software) systems
  - Guide implementation
  - Facilitate evolution

But how to define models??

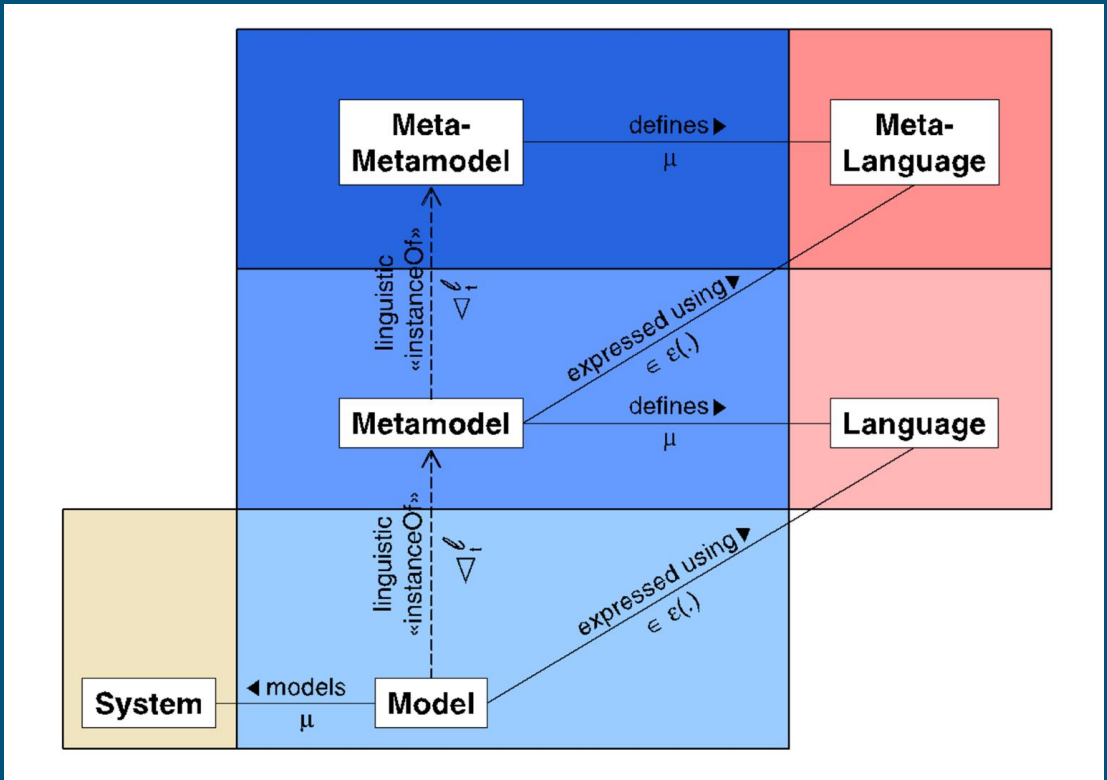
The MetaModel



A metamodel is a model of models

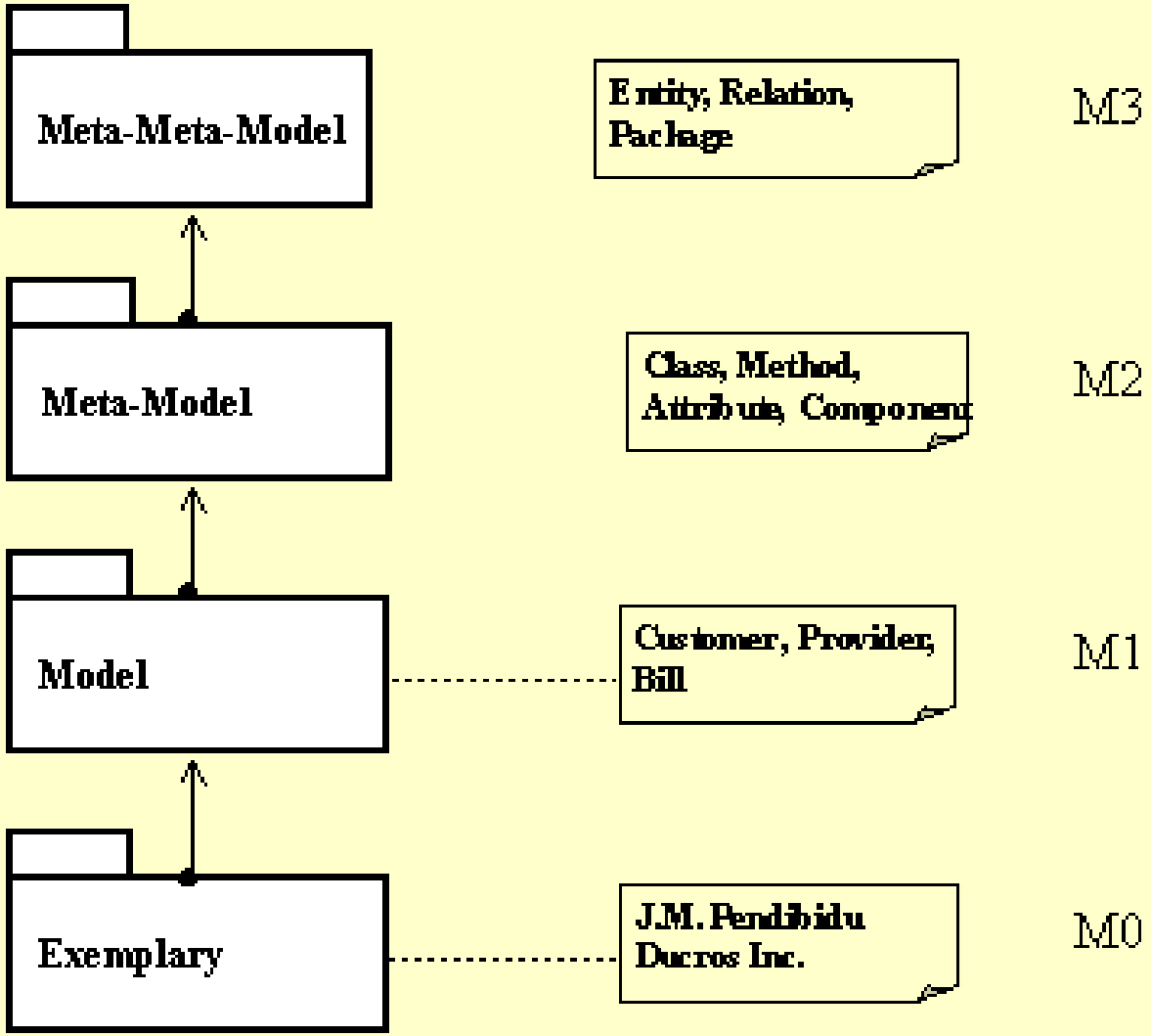
- **A model is an instance of a metamodel**
- **implies that a metamodel is a model of another model.**

Language mechanism stack

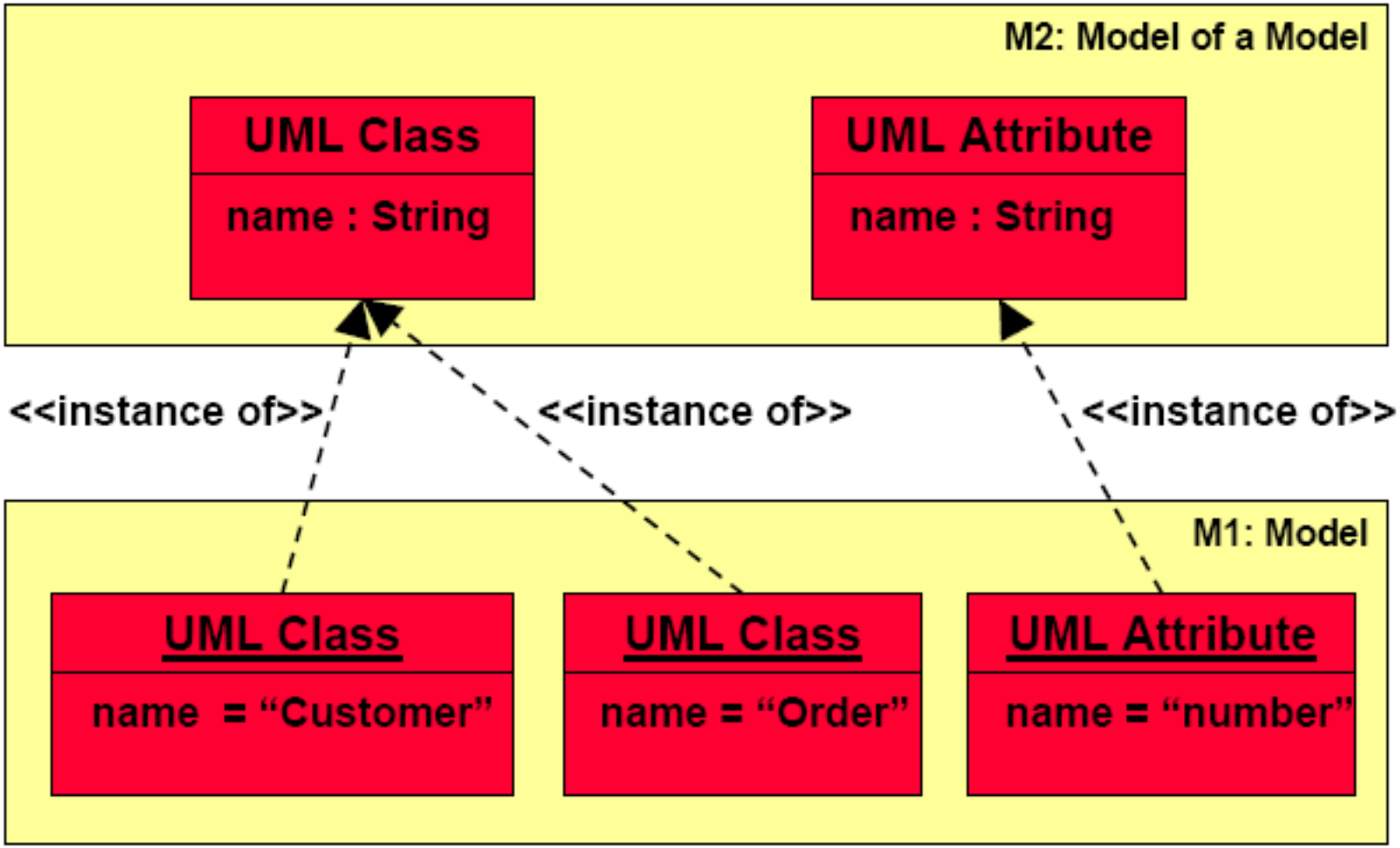


Thomas Kühne:
Matters of (Meta-)Modeling. Software and System Modeling 5(4): 369-385 (2006)

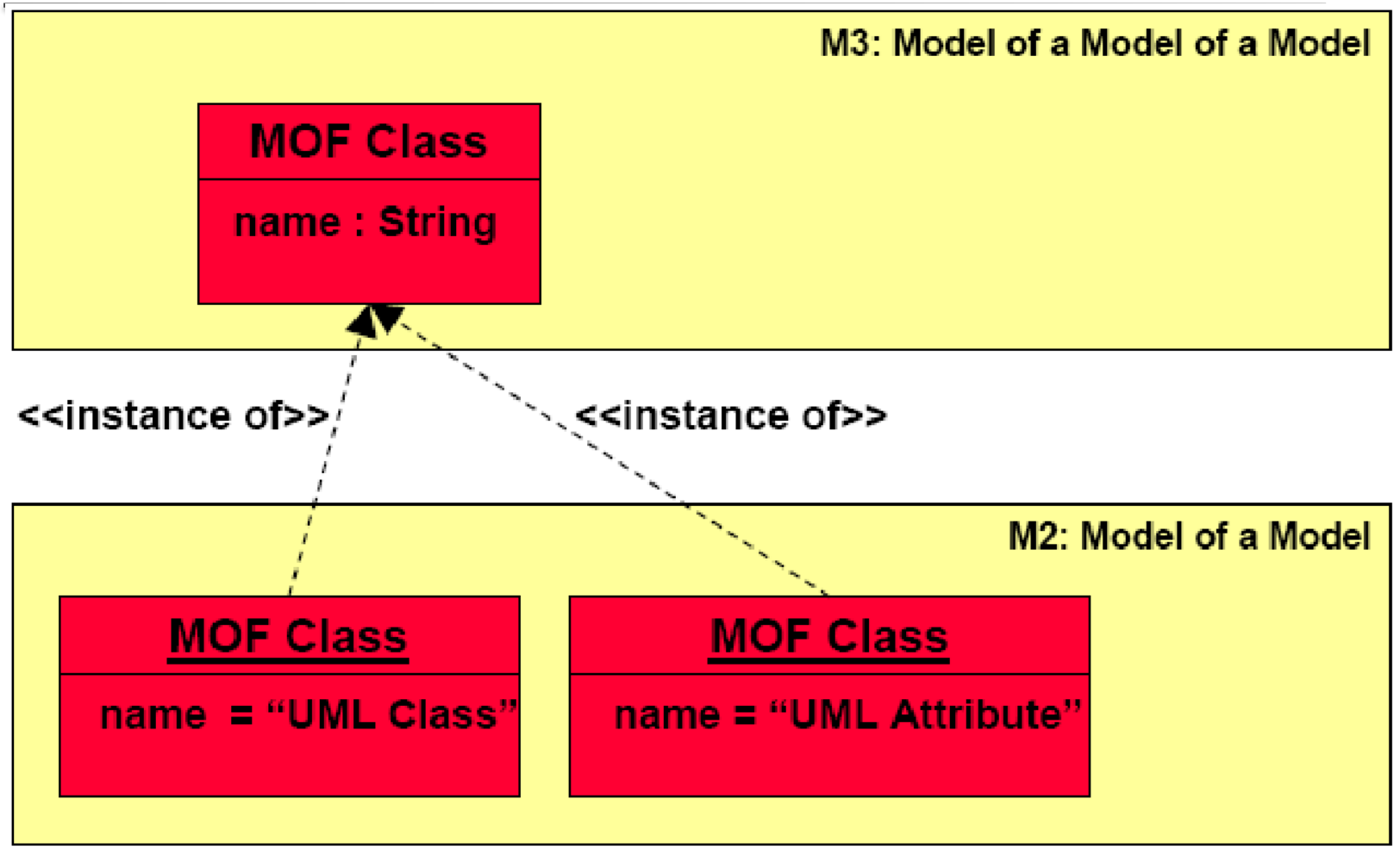
Four-Level Metamodel Hierarchy (OMG)



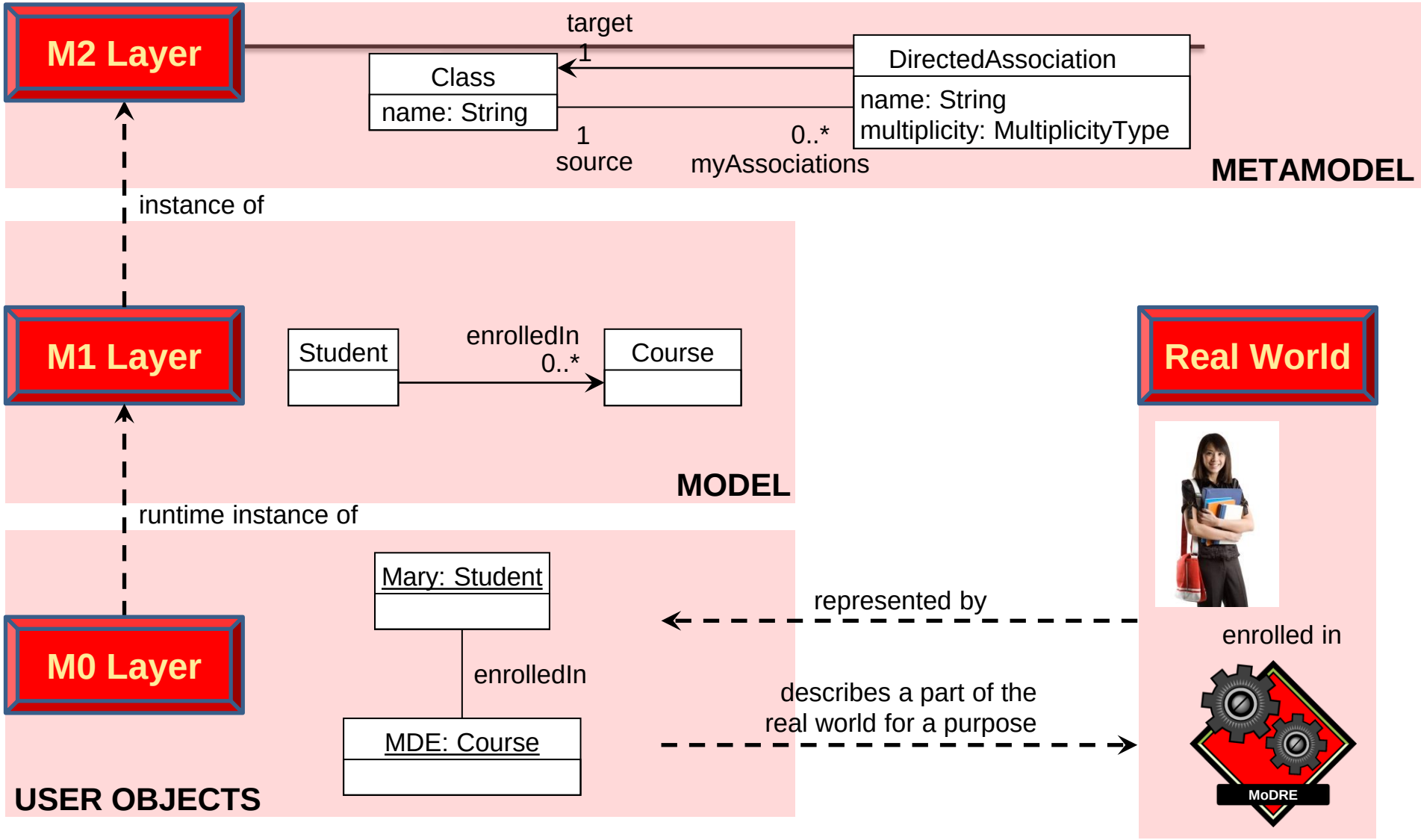
Model & metamodel (from Kleppe et al. “MDA explained”)



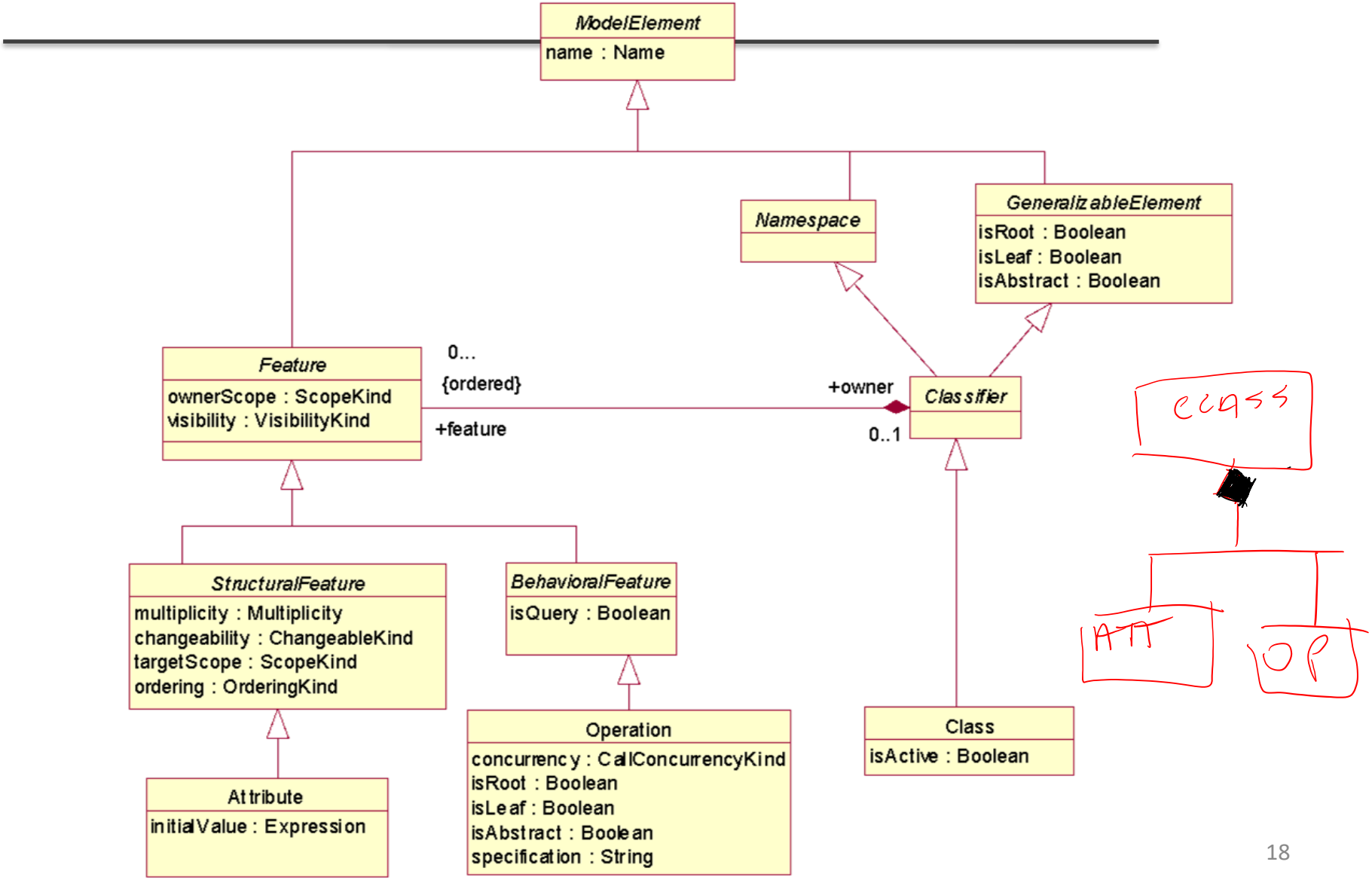
Metamodel & Meta-metamodel (from Kleppe et MDA explained)



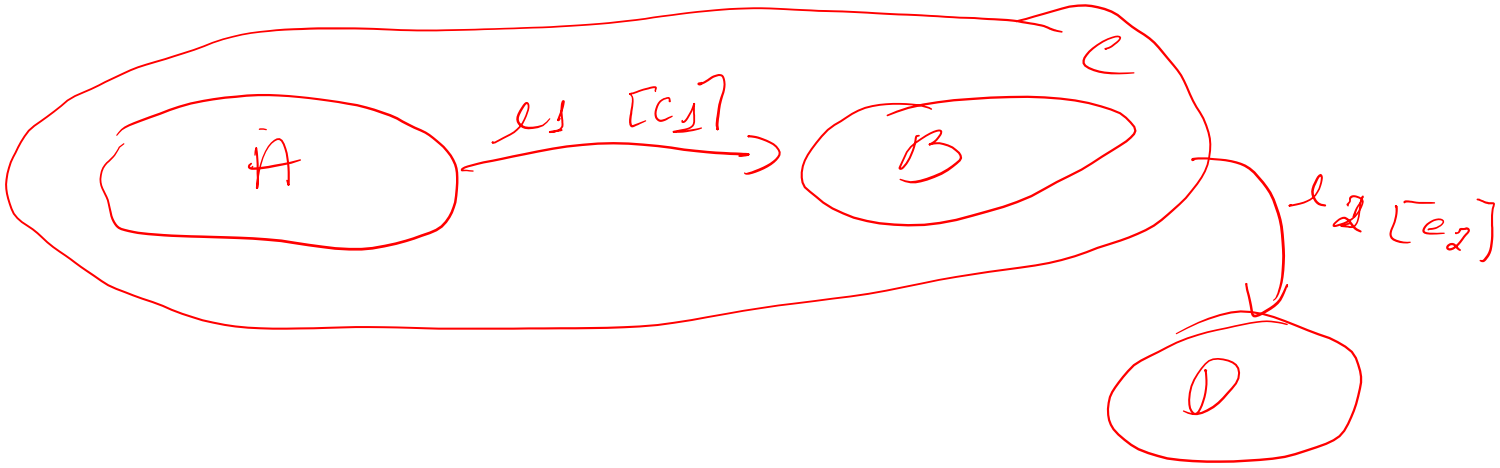
Layered Metamodel Hierarchy



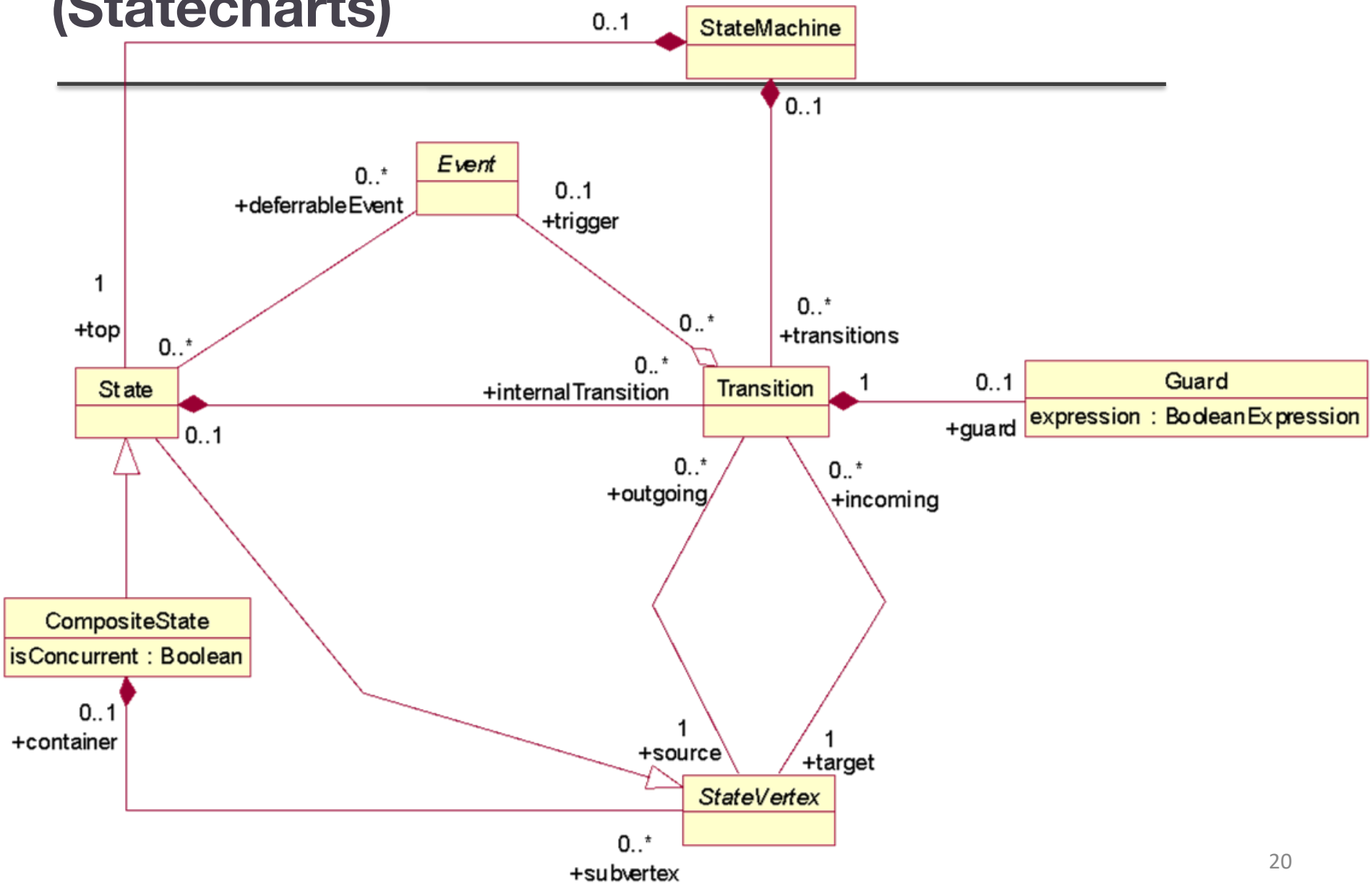
Fragment of UML Metamodel (Classes)



Exercise: Build a metamodel for statecharts, which includes states, transitions, events, Guards and composite states

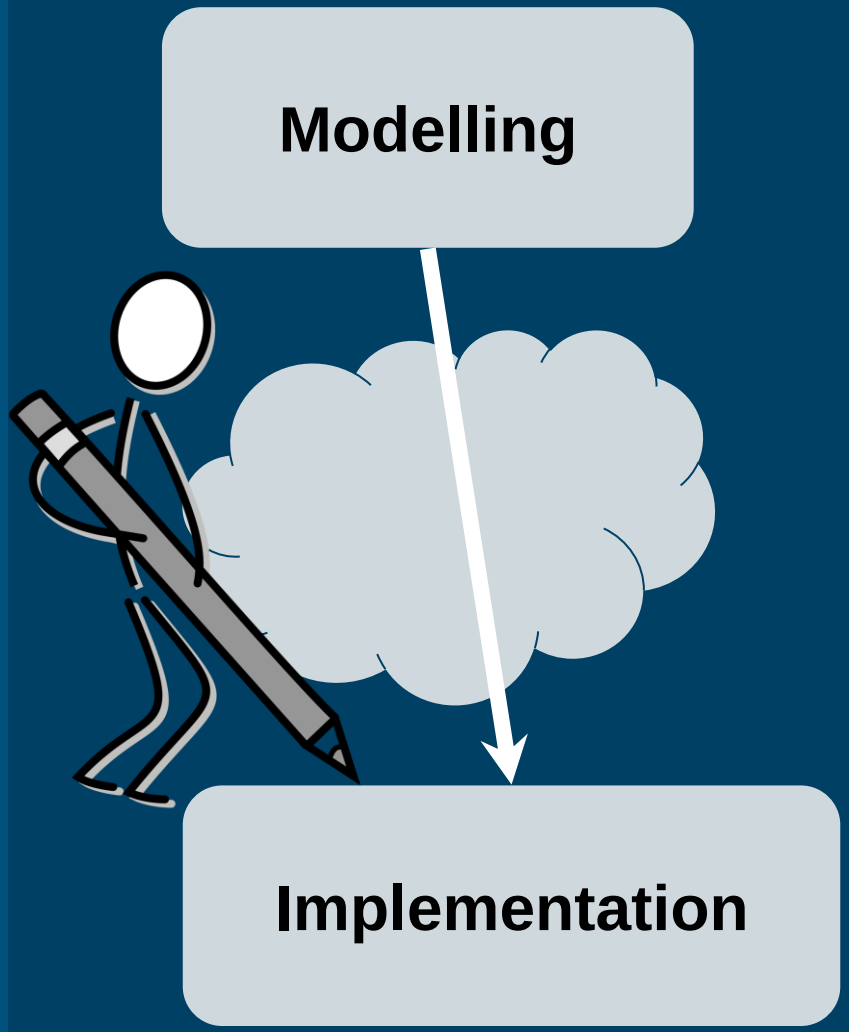


Fragment of the UML or SysML Metamodel (Statecharts)



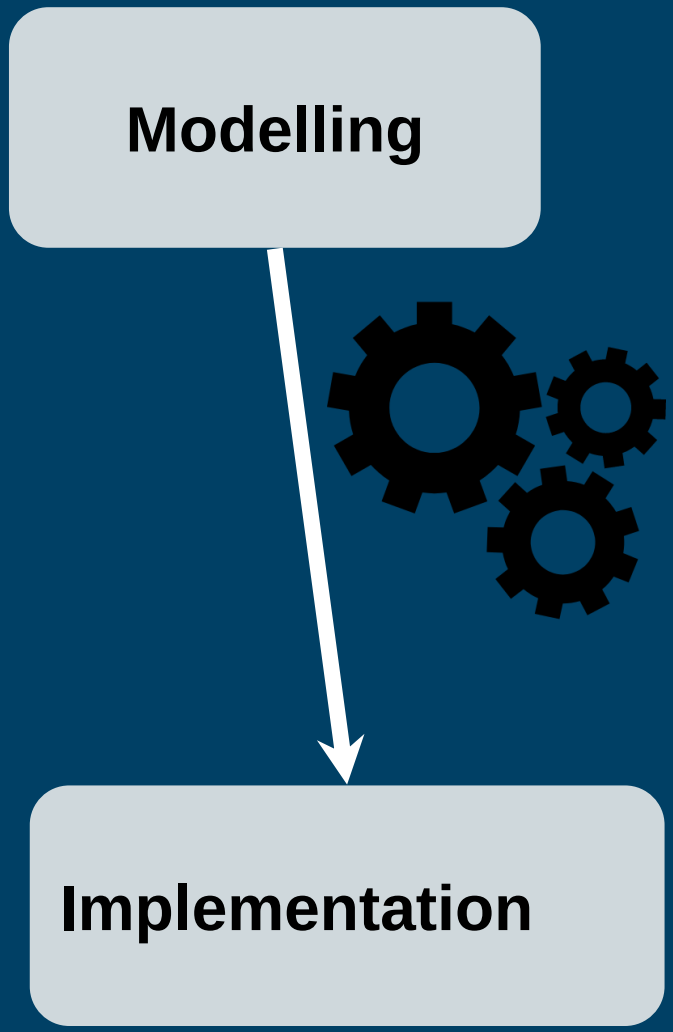
The Modelling Gap

The tradition...

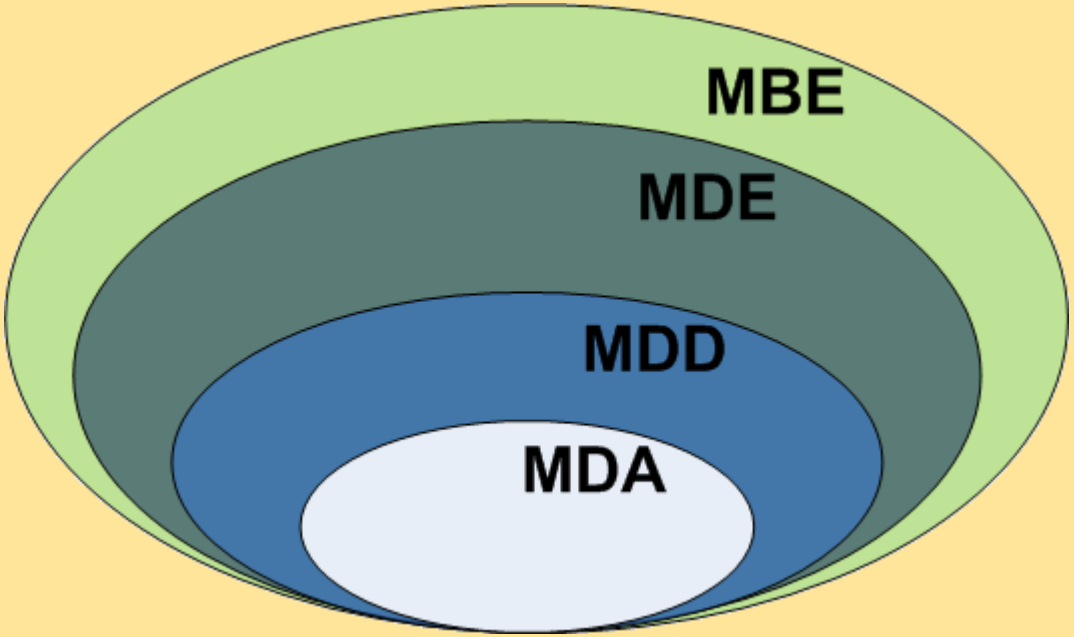


Modelling Gap

We need automation



Model Based Engineering, Model-Driven Engineering, Model-Driven Development, Model-Driven Architecture



Master thesis of David Ameller (supervised by Xavier Franch)

Model Driven Engineering

With MDE models are no longer simple mediums for describing systems or only facilitating inter-team communication…

A software system is obtained through the definition of different models at different abstraction layers.

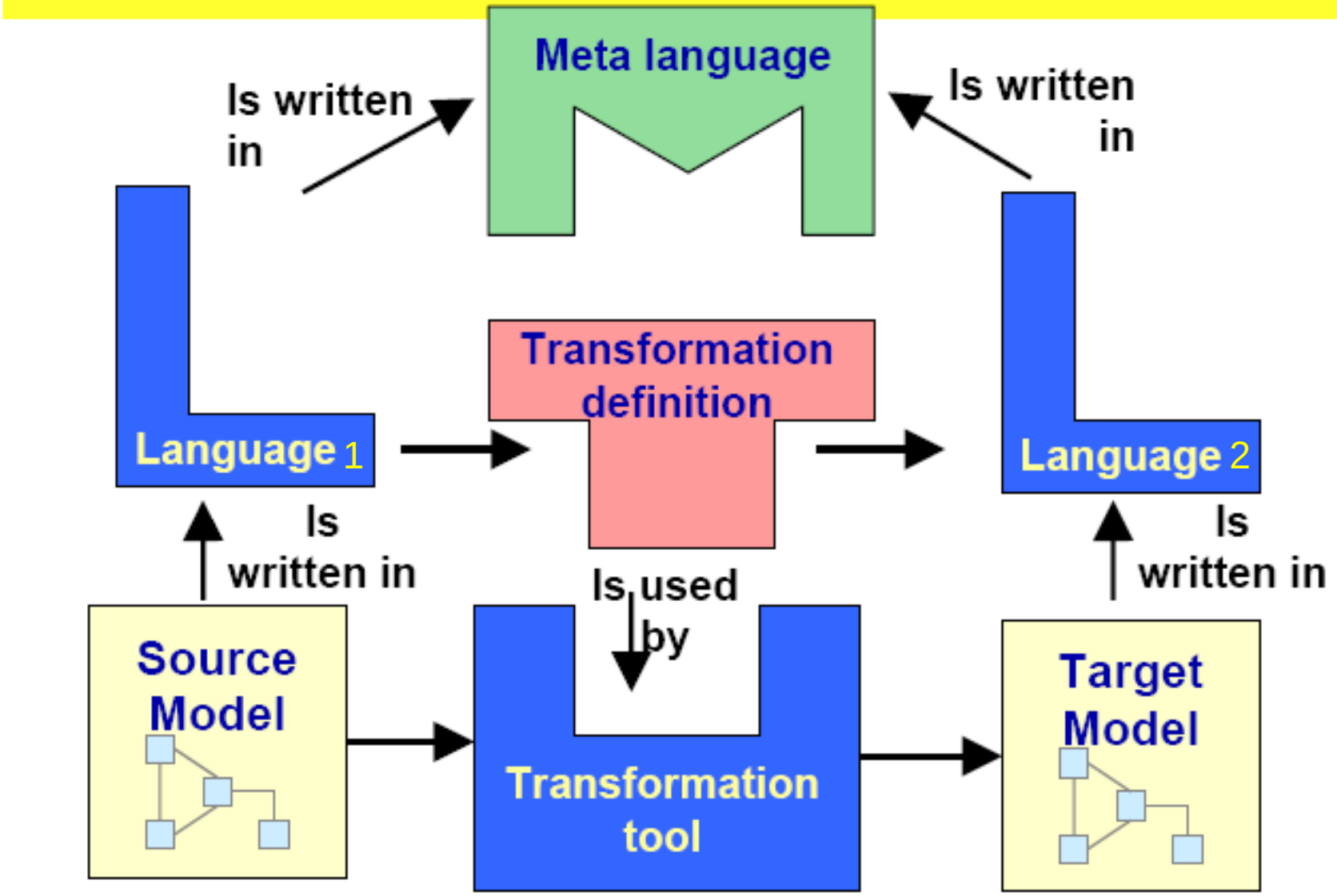
MDE proposes the systematic use of models as first-class software artifacts and their subsequent transformations throughout its life cycle.

MDE increases the level of abstraction and automates the development process

- This provides faster and more reliable results.

Model Transformations

MDA Framework (from Kleppe et al. MDA explained)



Usage of model-driven engineering

- Studies regarding the adoption of MDE by the industry:
- Good
  - Many success stories
  - Increment of productivity
  - Adequate for software reuse and product lines
  - Product quality increment with reduced:
    - testing, maintenance, integration, dev. time
- Bad
  - Lack of education on MDE
  - Upfront investment
- Ugly
  - Supporting tooling need to be improved

Usage of model-driven engineering

- Model-driven engineering in the industry examples of MDE Success stories:

Quidgest

outsystems

ARTiCA

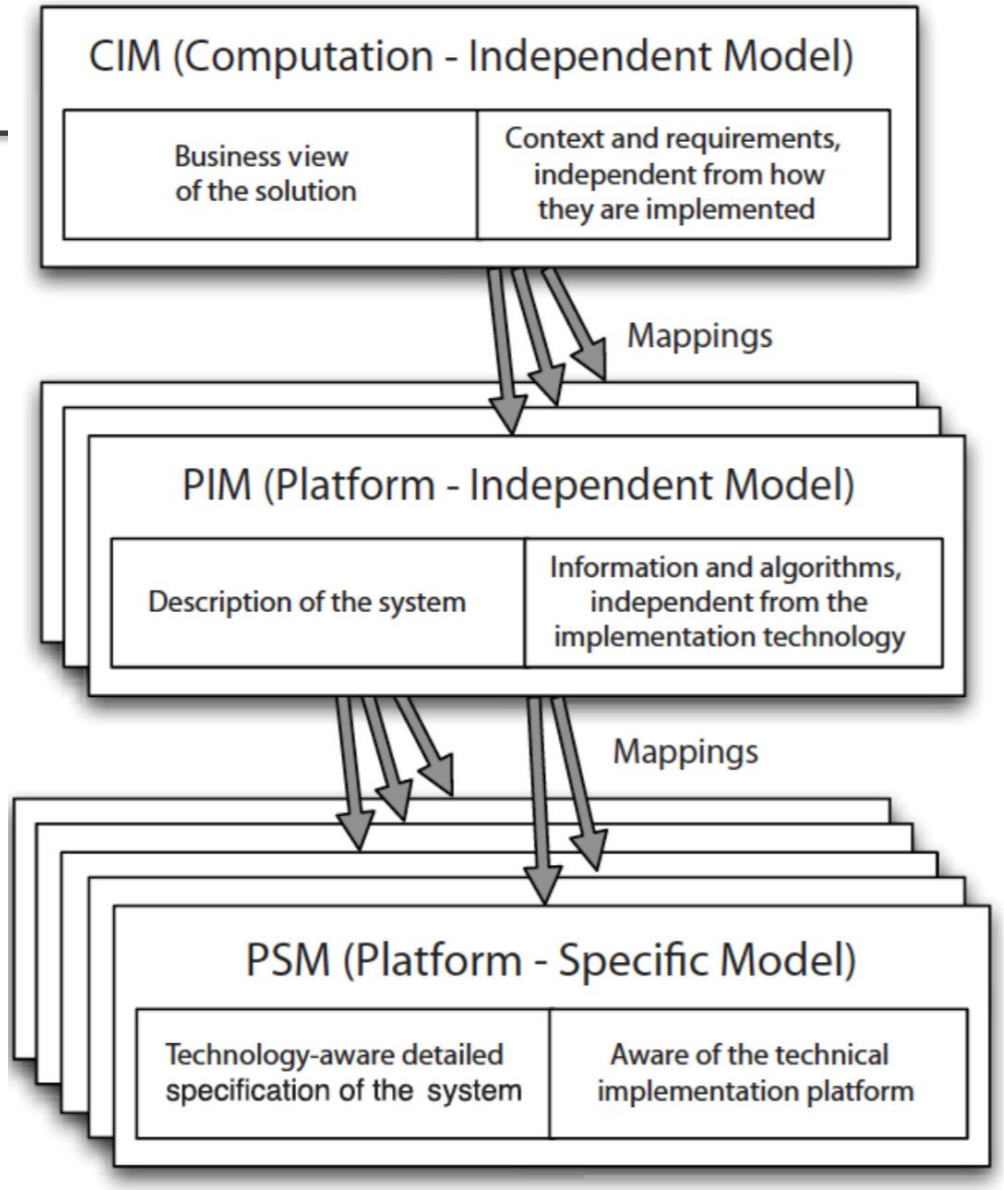
gmv INNOVATING SOLUTIONS

Model-driven architecture

- Model-driven architecture (MDA) was the precursor of more general model-driven engineering
- MDA is a model-focused approach to software design and implementation that uses OMG standards, **not only a subset of UML models to describe a system but also QVT as language transformations.**
- Models at different levels of abstraction are created. From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention.

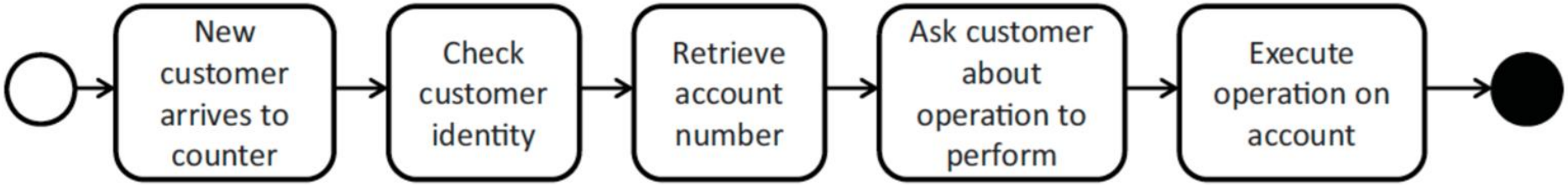
Types of model

- A computation independent model (CIM)
  - These model the important domain abstractions used in a system. CIMs are sometimes called domain models.
- A platform independent model (PIM)
  - These model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.
- *Platform specific models (PSM)*
  - These are transformations of the platform-independent model with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.


CIM (Computation - Independent Model)
Business view of the solution
Context and requirements, independent from how they are implemented
Mappings
PIM (Platform - Independent Model)
Description of the system
Information and algorithms, independent from the implementation technology
Mappings
PSM (Platform - Specific Model)
Technology-aware detailed specification of the system
Aware of the technical implementation platform

CIM

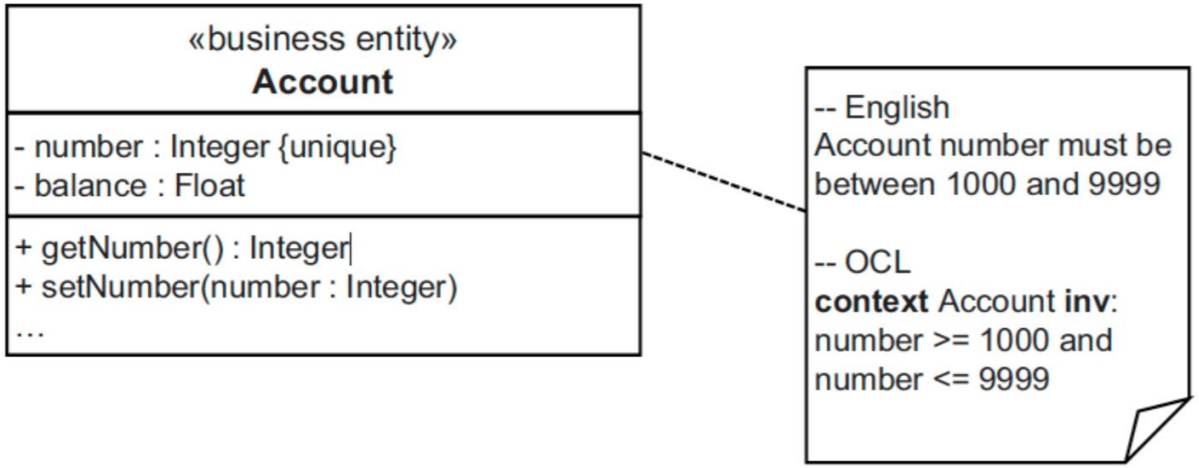
- E.g., business process , Use cases



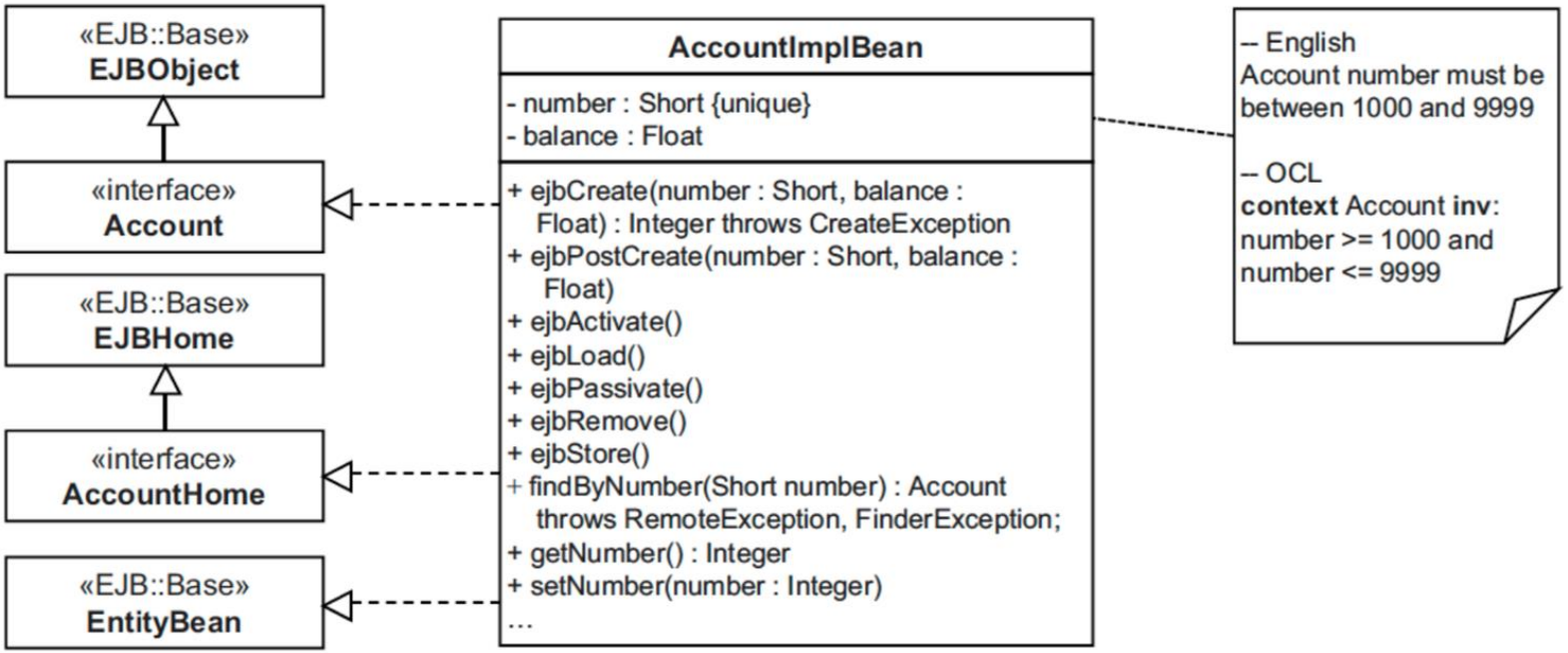
PIM

Specification of structure and behaviour of a system, abstracted from technological details:

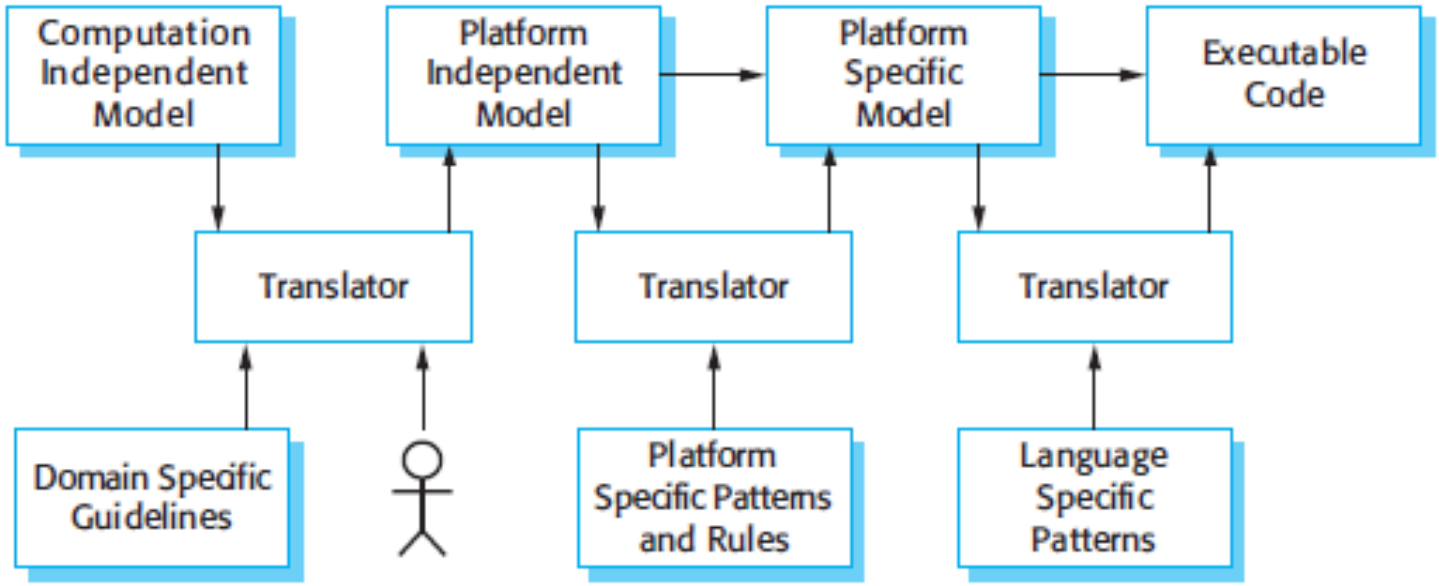
- ✧ Validation for correctness of the model
- ✧ Create implementations on different platforms
- ✧ Tool support during implementation



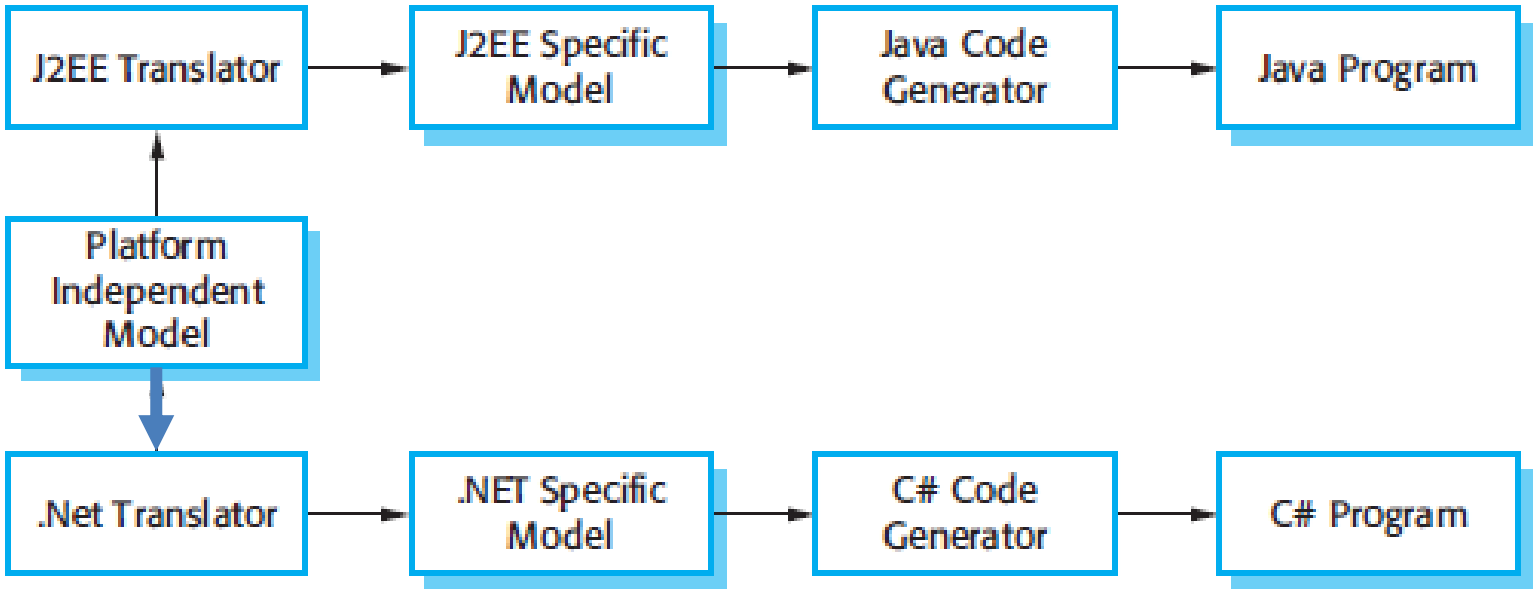
PSM


«EJB::Base»
EJBObject
«interface»
Account
«EJB::Base»
EJBHome
«interface»
AccountHome
«EJB::Base»
EntityBean
AccountImplBean
- number : Short {unique}
- balance : Float
+ ejbCreate(number : Short, balance : Float) : Integer throws CreateException
+ ejbPostCreate(number : Short, balance : Float)
+ ejbActivate()
+ ejbLoad()
+ ejbPassivate()
+ ejbRemove()
+ ejbStore()
+ findByNumber(Short number) : Account throws RemoteException, FinderException;
+ getNumber() : Integer
+ setNumber(number : Integer)
...
-- English
Account number must be between 1000 and 9999
-- OCL
context Account inv:
number >= 1000 and number <= 9999

MDA transformations



Multiple platform-specific models

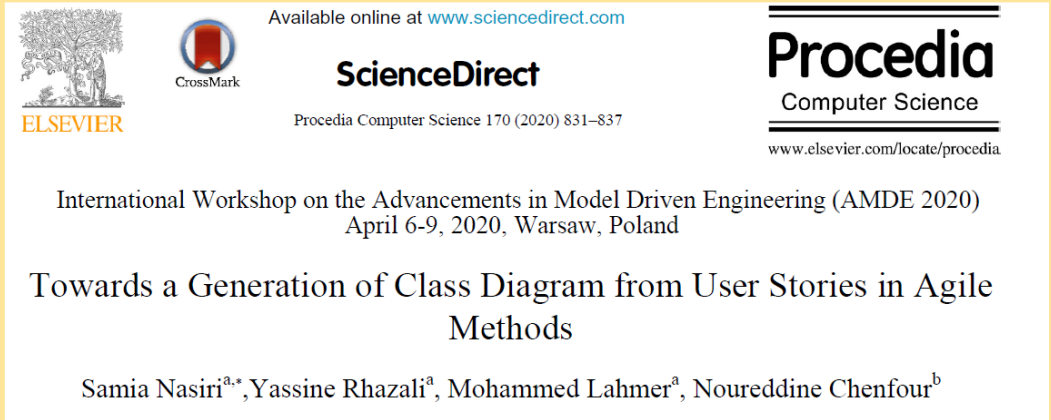


MDE + RE!

- RE also benefits from MDE
- MDE may be used to:
  - ensure **consistency between different kinds of requirements** analysis models (e.g., goal, scenario or domain models)
  - to **automatically build architectural models from requirements**
  - to **increase separation of concerns** and their **composability**
  - Creation of domain specific requirements modeling languages

Agile methods and MDA

- The developers of MDA claim that it is intended to support an iterative approach to development and **so can be used within agile methods**.
- **If transformations can be completely automated** and a complete program generated from a **PIM, then, MDA could be used in an agile development** process as no separate coding would be required.
- Agile MDA:
  - Models are linked together, rather than transformed, and they are then all mapped to a single combined model that is then translated into code according to a single system architecture.



Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 170 (2020) 831–837

Procedia Computer Science

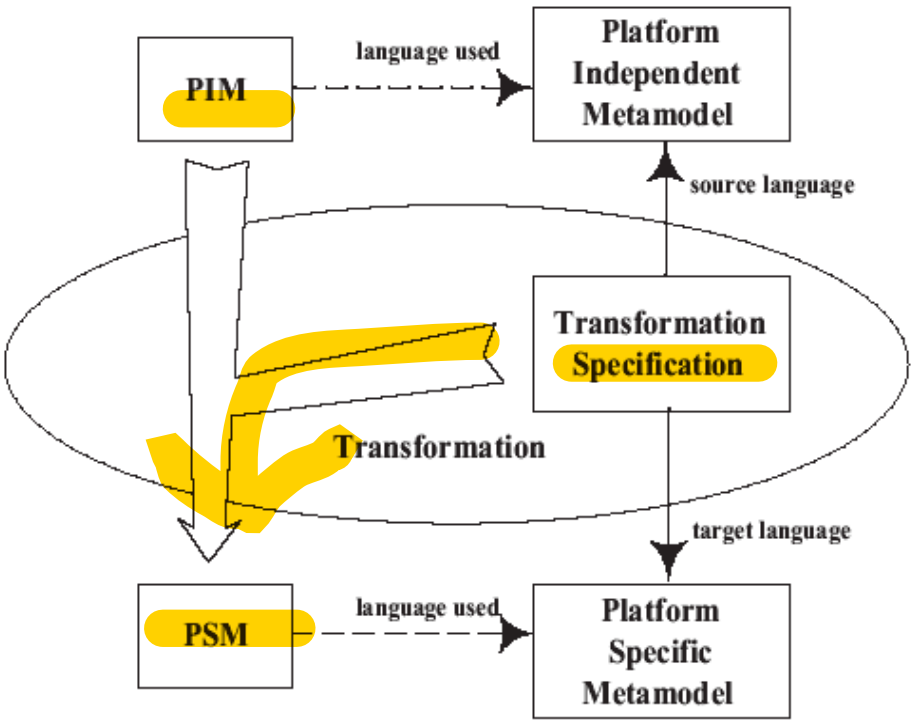
www.elsevier.com/locate/procedia

International Workshop on the Advancements in Model Driven Engineering (AMDE 2020)
April 6-9, 2020, Warsaw, Poland

Towards a Generation of Class Diagram from User Stories in Agile Methods

Samia Nasiri[a,*], Yassine Rhazali[a], Mohammed Lahmer[a], Noureddine Chenfour[b]

Transformation with metamodels



A model is prepared using the platform-independent language specified by the meta-model.

A particular platform is chosen.

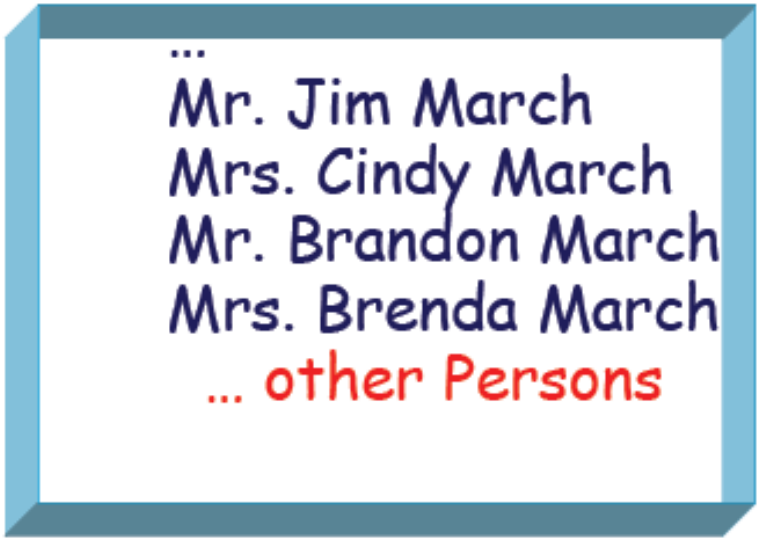
The specification is described in terms of the mapping between the meta-models.

Model Transformation Languages: using ATL and QVT

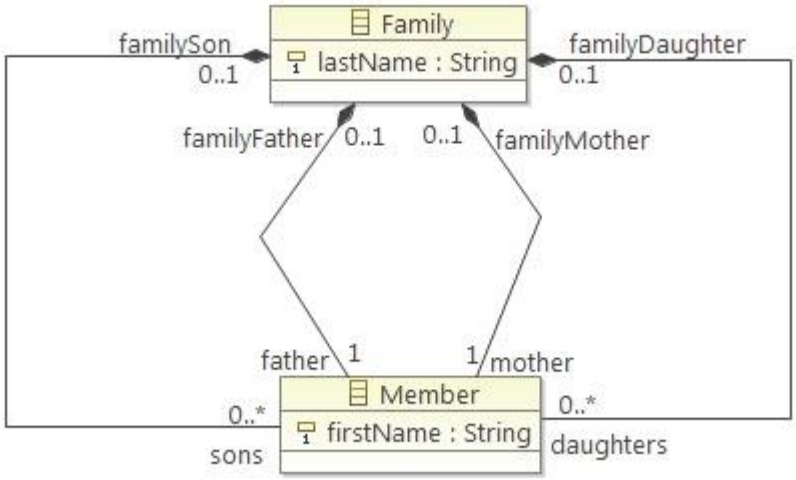
- A language to describe families and other for persons

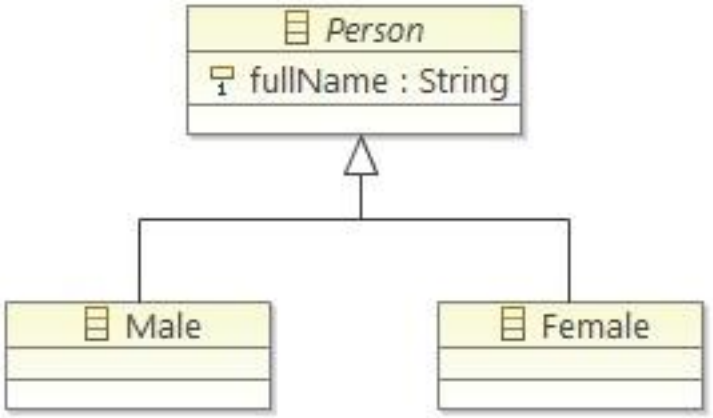
Transforming this ...

... into this.



Metamodels






```
package Families {

	class Family {
		attribute lastName : String;
		reference father container : Member oppositeOf familyFather;
		reference mother container : Member oppositeOf familyMother;
		reference sons[*] container : Member oppositeOf familySon;
		reference daughters[*] container : Member oppositeOf familyDaughter;
	}

	class Member {
		attribute firstName : String;
		reference familyFather[0-1] : Family oppositeOf father;
		reference familyMother[0-1] : Family oppositeOf mother;
		reference familySon[0-1] : Family oppositeOf sons;
		reference familyDaughter[0-1] : Family oppositeOf daughters;
	}

}

package PrimitiveTypes {
	datatype String;
}
```

```
package Persons {

	abstract class Person {
		attribute fullName : String;
	}

	class Male extends Person { }

	class Female extends Person { }

}

package PrimitiveTypes {
	datatype String;
}
```

Regras de transformação

- Member to Male

```
rule Member2Male {
        from
                s : Families!Member (not s.isFemale())
        to
                t : Persons!Male (
                        fullName <- s.firstName + ' ' + s.familyName
                )
```

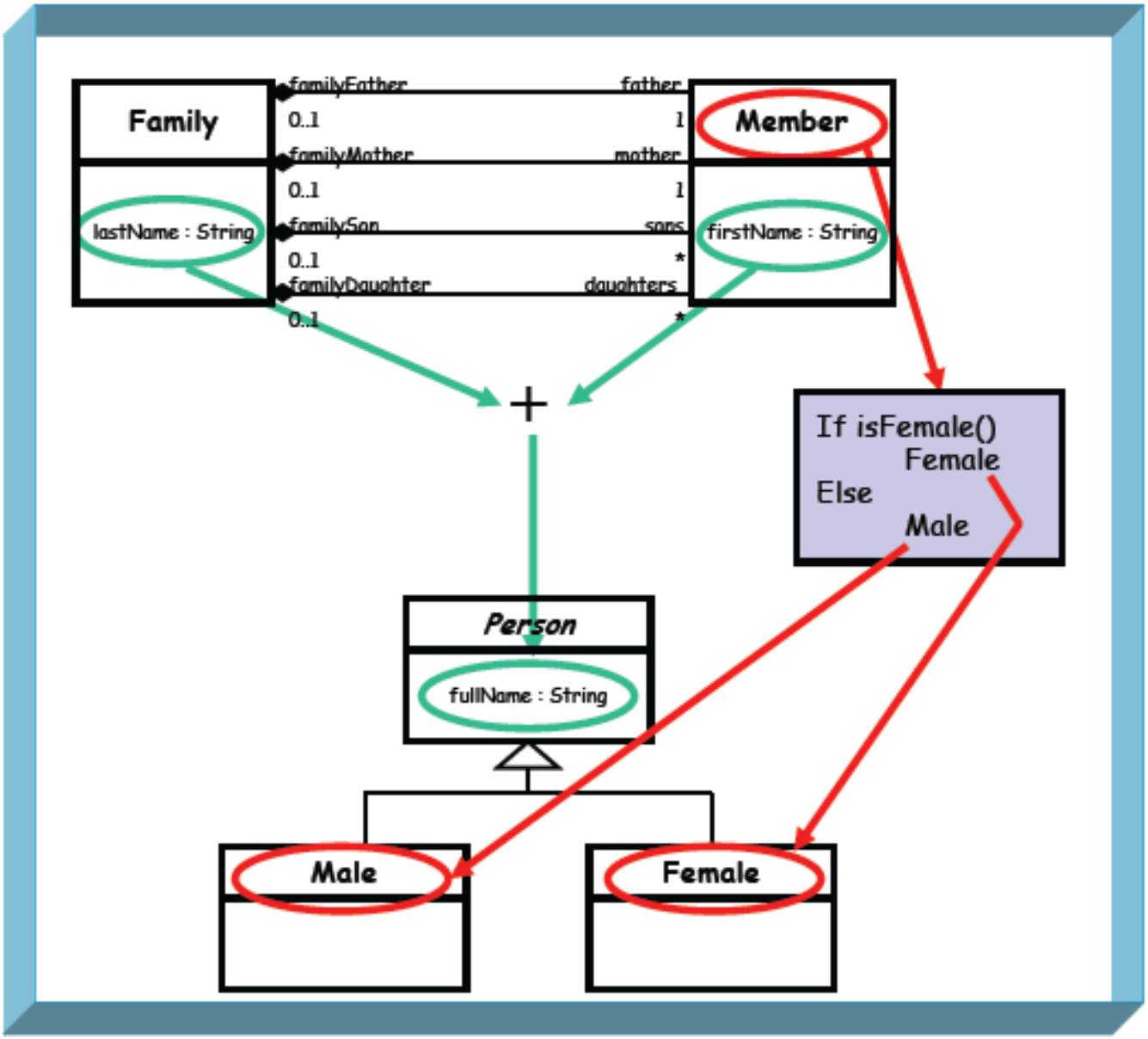
- Member to Female

```
rule Member2Female {
        from
                s : Families!Member (s.isFemale())
        to
                t : Persons!Female (
                        fullName <- s.firstName + ' ' + s.familyName
                )
}
```

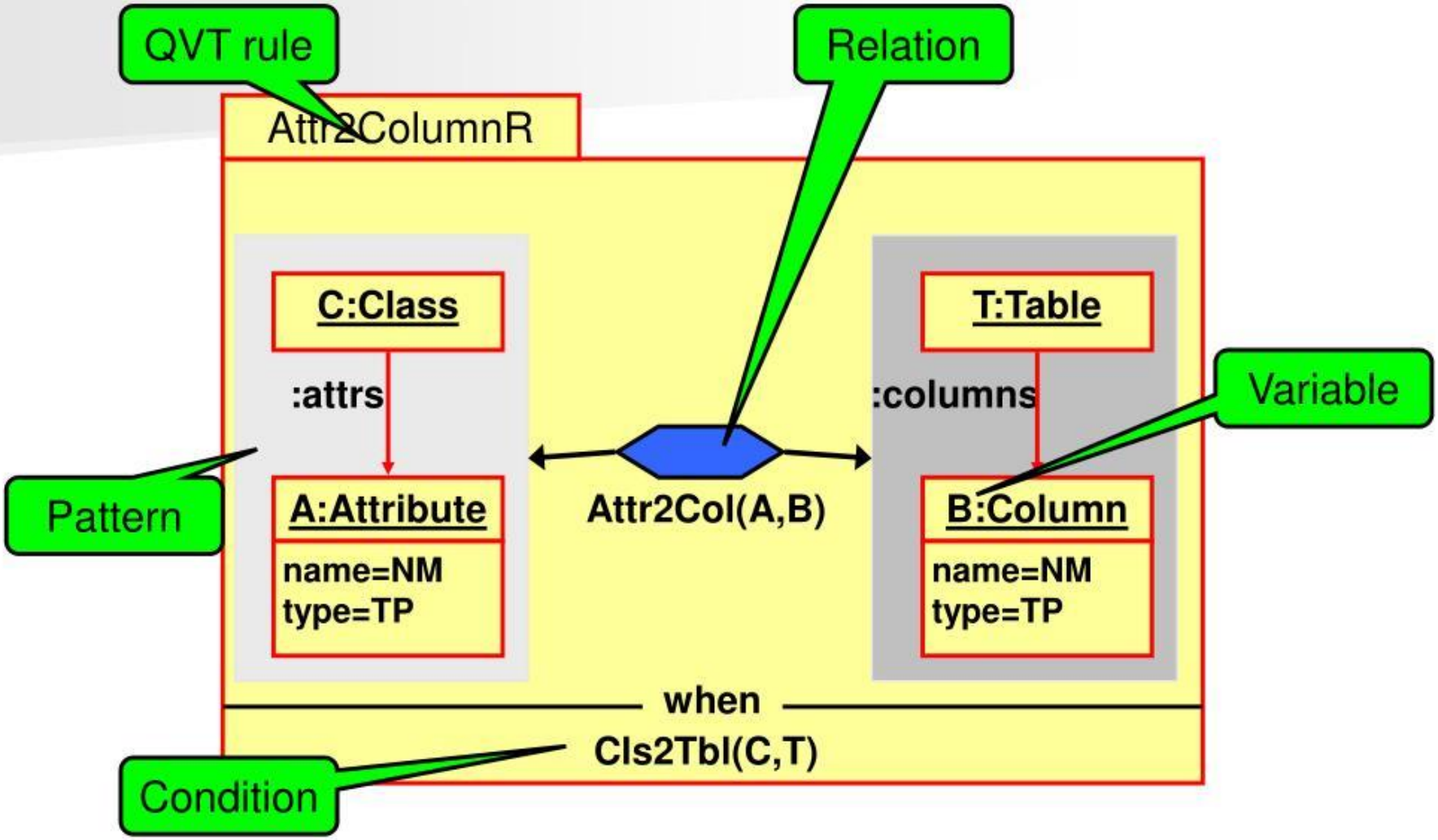
isFemale()

```
helper context Families!Member def: isFemale(): Boolean =
        if not self.familyMother.oclIsUndefined() then
                true
        else
                if not self.familyDaughter.oclIsUndefined() then
                        true
                else
                        false
                endif
        endif
```

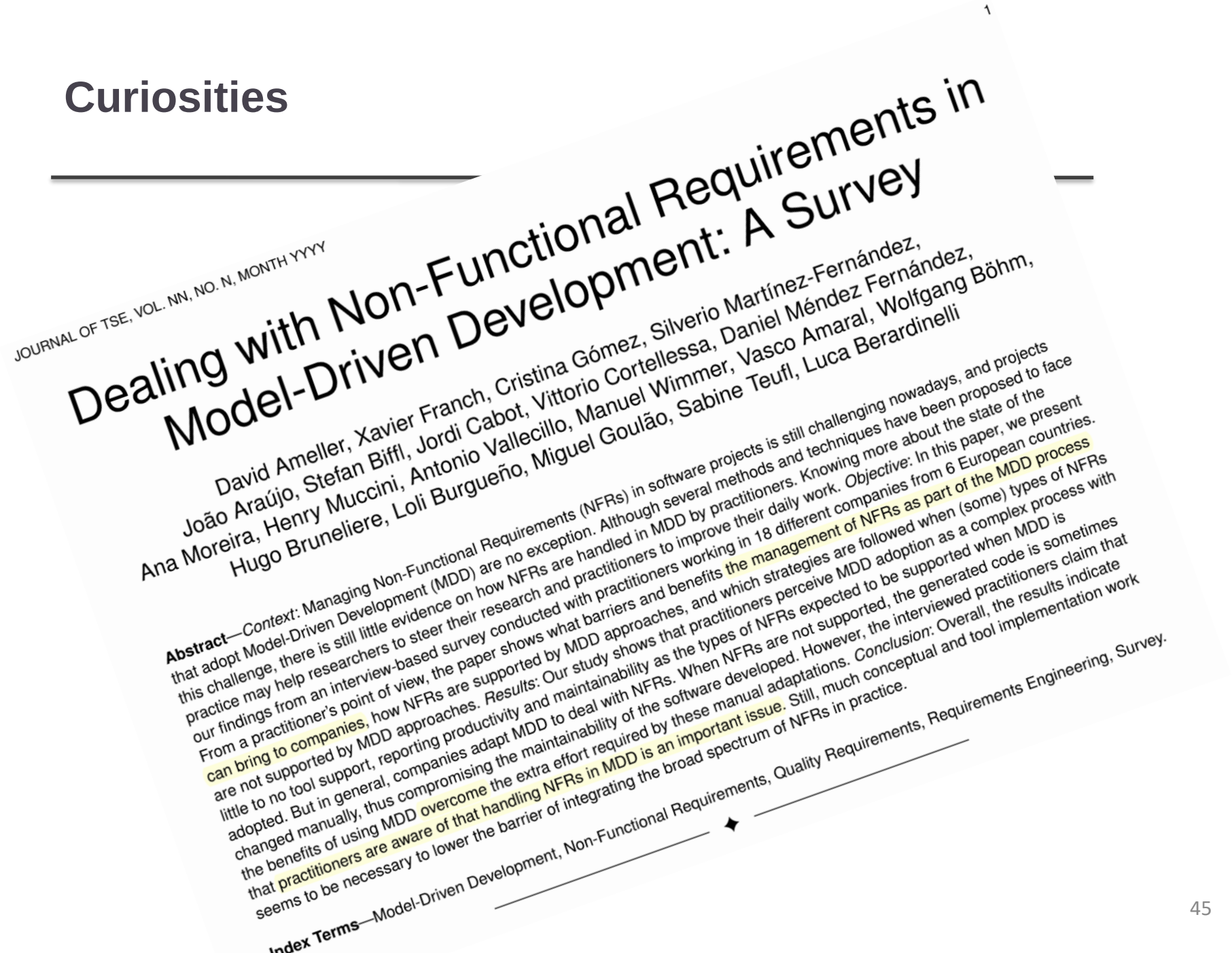
Summary of the transformation



QVT example


QVT rule
Relation
Attr2ColumnR
C:Class
:attrs
T:Table
:columns
Variable
Pattern
A:Attribute
name=NM
type=TP
Attr2Col(A,B)
B:Column
name=NM
type=TP
when
Cls2Tbl(C,T)
Condition

Curiosities



JOURNAL OF TSE, VOL. NN, NO. N, MONTH YYYY 1

Dealing with Non-Functional Requirements in Model-Driven Development: A Survey

David Ameller, Xavier Franch, Cristina Gómez, Silverio Martínez-Fernández, João Araújo, Stefan Biffl, Jordi Cabot, Vittorio Cortellessa, Daniel Méndez Fernández, Ana Moreira, Henry Muccini, Antonio Vallecillo, Manuel Wimmer, Vasco Amaral, Wolfgang Böhm, Hugo Bruneliere, Loli Burgueño, Miguel Goulão, Sabine Teufl, Luca Berardinelli

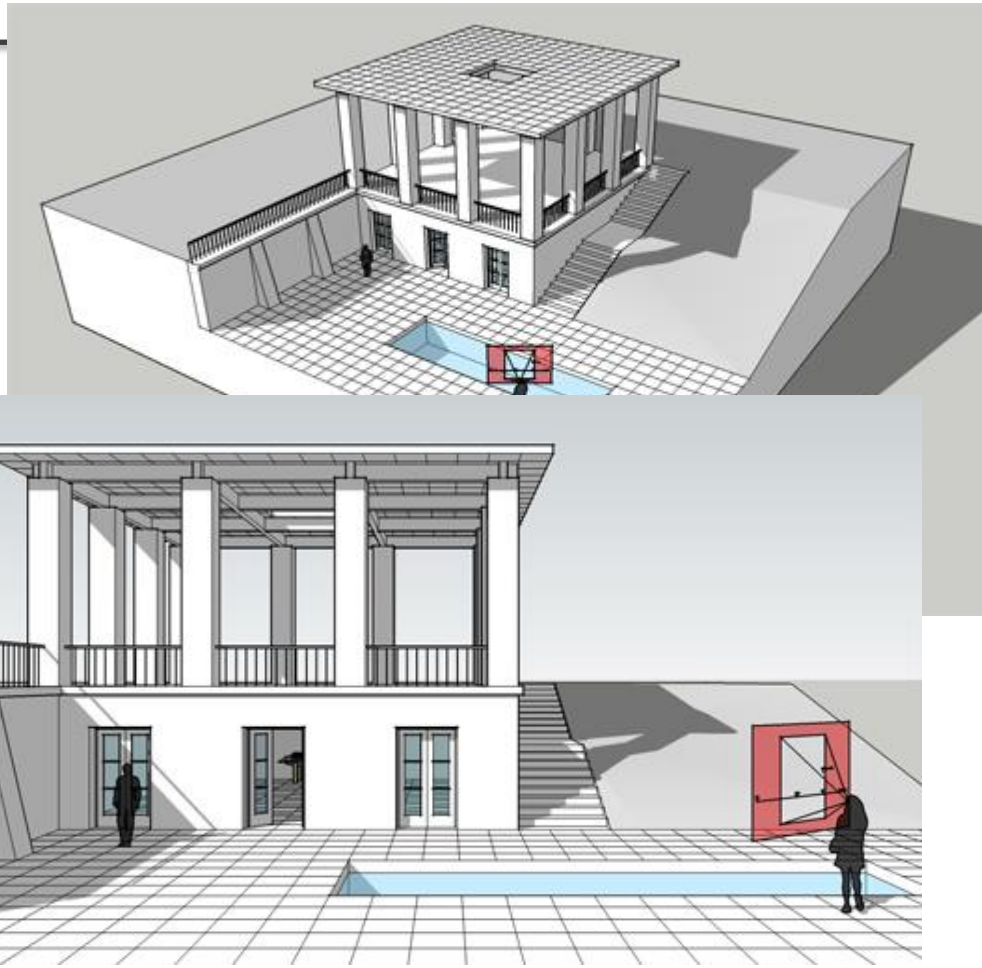
Abstract—*Context*: Managing Non-Functional Requirements (NFRs) in software projects is still challenging nowadays, and projects that adopt Model-Driven Development (MDD) are no exception. Although several methods and techniques have been proposed to face this challenge, there is still little evidence on how NFRs are handled in MDD by practitioners. Knowing more about the state of the practice may help researchers to steer their research and practitioners to improve their daily work. *Objective*: In this paper, we present our findings from an interview-based survey conducted with practitioners working in 18 different companies from 6 European countries. From a practitioner's point of view, the paper shows what barriers and benefits the management of NFRs as part of the MDD process can bring to companies, how NFRs are supported by MDD approaches, and which strategies are followed when (some) types of NFRs are not supported by MDD approaches. *Results*: Our study shows that practitioners perceive MDD adoption as a complex process with little to no tool support, reporting productivity and maintainability as the types of NFRs expected to be supported when MDD is adopted. But in general, companies adapt MDD to deal with NFRs. When NFRs are not supported, the generated code is sometimes changed manually, thus compromising the maintainability of the software developed. However, the interviewed practitioners claim that the benefits of using MDD overcome the extra effort required by these manual adaptations. *Conclusion*: Overall, the results indicate that practitioners are aware of that handling NFRs in MDD is an important issue. Still, much conceptual and tool implementation work seems to be necessary to lower the barrier of integrating the broad spectrum of NFRs in practice.

Index Terms—Model-Driven Development, Non-Functional Requirements, Quality Requirements, Requirements Engineering, Survey.

Models perspectives

(Software) Systems modeling perspectives

- System modeling is the process of developing abstract models of a system
  - each model is a **different system view or perspective**
- System modeling represents a system using graphical notation (e.g., SysML).
- System modelling **helps the analyst to understand the functionalities of the system**
- Models are used to communicate with customers



Existing and planned system models

- Models of the existing system are **used during RE**.
  - They help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses.
  - These then lead to requirements for the new system.
- Models of the new system are used during RE to **help explain the proposed requirements** to other system stakeholders.
  - Engineers use these models to discuss design proposals and to document the system for implementation.
- In a **MDE** process, **it is possible to generate a complete or partial system implementation** from the system model.

System perspectives

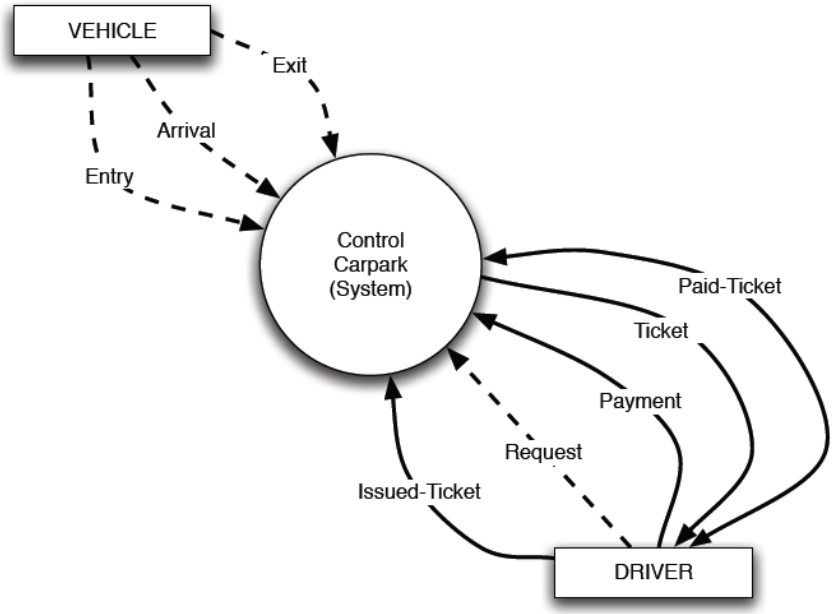
- An **external perspective**, where you model the context or environment of the system.
- An **interaction perspective**, where you model the interactions **between a system and its environment**, or **between the components** of a system.
- A **structural perspective**, where you model the **organization** of a system or the **structure of the data** that is processed by the system.
- A **behavioral perspective**, where you model the **dynamic behavior** of the system and how it responds to events.

Use of graphical models

- As a means of **facilitating discussion** about an existing or proposed system
  - Incomplete and incorrect models are OK as their role is to support discussion.
- As a way of **documenting** an existing system
  - Models should be an accurate representation of the system but need not be complete.
- As a **detailed system description** that can be used to generate a system implementation
  - Models have to be **both correct and complete**.

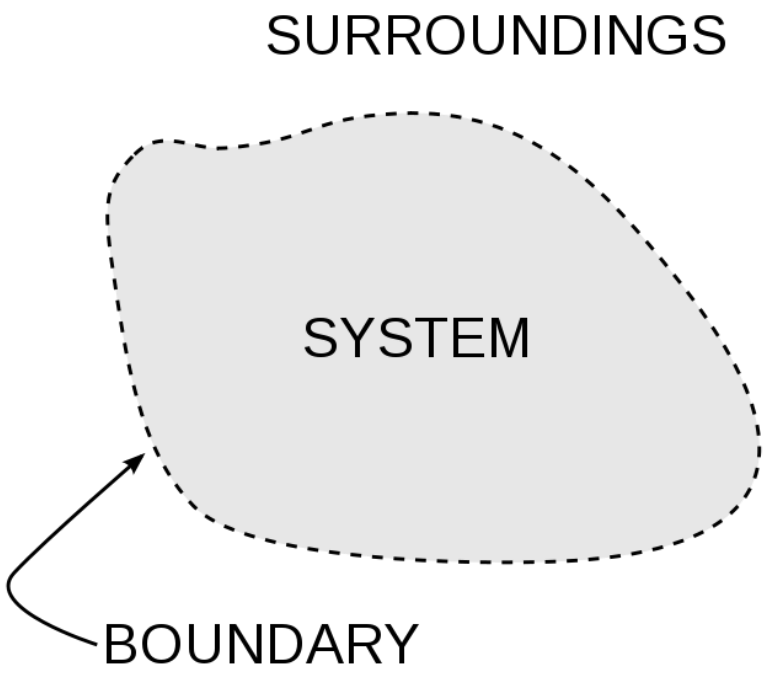
Context models

- Context models are used to **illustrate the operational context of a system** - they show what lies outside the system boundaries.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- **Architectural models show the system and its relationship with other systems**.



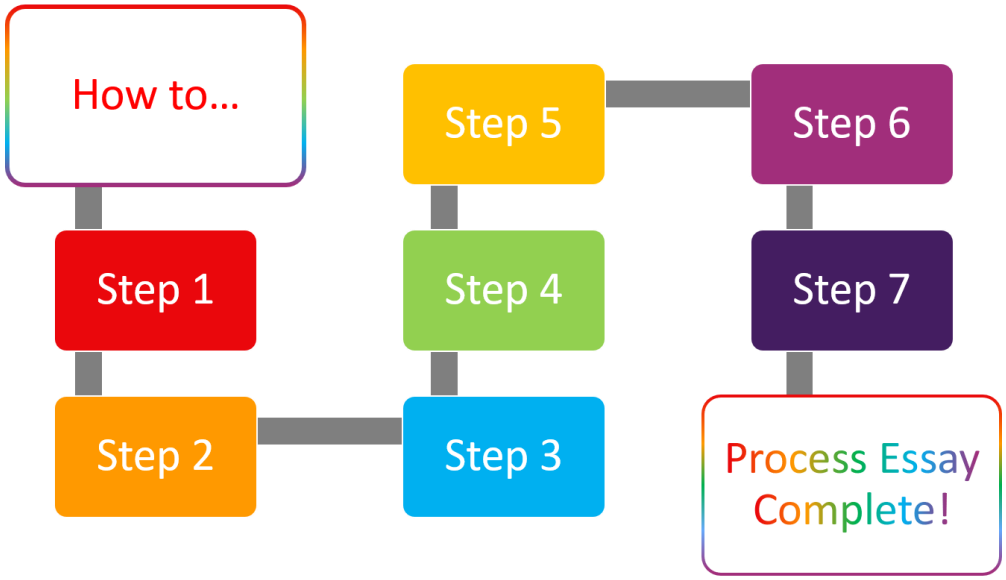
System boundaries

- System boundaries are established to define **what is inside and what is outside the system**.
  - They show other systems that are used or depend on the system being developed.
- The position of the system boundary has a profound effect on the system requirements.
- Defining a system boundary is a political judgement
  - There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.



Process perspective

- Process models reveal how the system being developed is used in broader business processes.
- SysML activity diagrams or BPMN models may be used to define business process models.



Structural and Interaction models

- Structural models:
  - Display the organization of a system in terms of the components that make up that system and their relationships.
  - You create **structural models of a system when you are discussing and designing the system architecture**.
- Interaction models
  - Modeling user interaction is important as it **helps to identify user requirements**.
  - Modeling system-to-system interaction **highlights the communication problems** that may arise.
  - **Use case diagrams and sequence diagrams may be used for interaction modeling**.

UML Models → SysML Models