INF5120
”Modellbasert Systemutvikling”
”Modelbased System development”

Lecture 12: 03.04.2017

Arne-Jørgen Berre

arneb@ifi.uio.no or Arne.J.Berre@sintef.no

Content

- Introduction to Metamodels and UML Profiles
- Introduction to Eclipse, EMF and Sirius
- Introduction to Oblig 3 – for May 4th

Course parts (16 lectures) - 2017

- January (1-3) (Introduction to Modeling, Business Architecture and the Smart Building project):
- 1-16/1: Introduction to INF5120
- 2-23/1: Modeling structure and behaviour (UML and UML 2.0 and metamodeling) - (establish Oblig groups)
- 3-30/1: WebRatio for Web Apps/Portals and Mobile Apps – and Entity/Class modeling – (Getting started with WebRatio)

- February (4-7) (Modeling of User Interfaces, Flows and Data model diagrams, Apps/Web Portals - IFML/Client-Side):
- 4-6/2: Business Model Canvas, Value Proposition, Lean Canvas and Essence
- 5-13/2: IFML – Interaction Flow Modeling Language, WebRatio advanced – for Web and Apps
- 6-20/2: BPMN process, UML Activ.Diagrams, Workflow and Orchestration modelling value networks
- 7-27/2: Modeling principles – Quality in Models
- 27/2: Oblig 1: Smart Building – Business Architecture and App/Portal with IFML WebRatio UI for Smart Building

- March (8-11) (Modeling of IoT/CPS/Cloud, Services and Big Data – UML SM/SD/Collab, ThingML Server-Side):
- 8-6/3: Basis for DSL and ThingML -> UML State Machines and Sequence Diagrams
- 9-13/3: ThingML DSL - UML Composite structures, State Machines and Sequence Diagrams II
- 10-20/3: Guest lecture, "Experience with Modelling", Anton Landmark, SINTEF
- 11-27/3: ThingML part 2 and UML Service Modeling, Architectural models, SoaML. Role modeling and UML Collaboration diagrams

- April/May (12-14) (MDE – Creating Your own Domain Specific Language):
- 12-3/4: Model driven engineering – Metamodels, DSL, UML Profiles, EMF, Sirius Editors – intro to Oblig 3

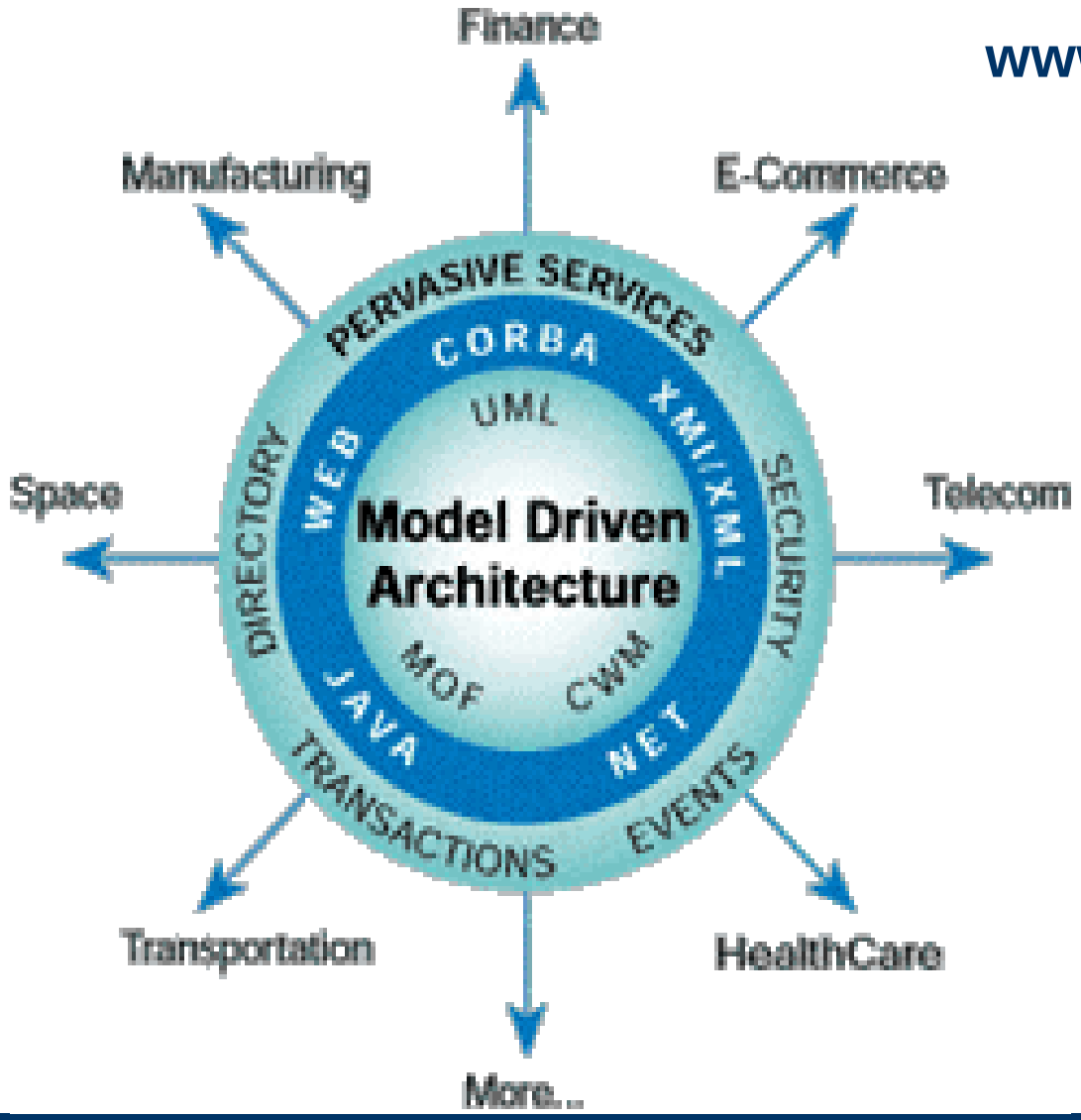
- EASTER – 10/4 og 17/4
- 20/4: Oblig 2: Smart Building – Individual and group delivery - Internet of Things control with ThingML – Raspberry Pi, Wireless sensors (temperature, humidity), actuators (power control)

- 13-24/4: MDE transformations, Non Functional requirements – Discussion of Oblig2 and 3
- 1. Mai – Official holiday
- 4/5: Oblig 3 - Your own Domain Specific Language – (ArchiMate) (Delivery – Thursday May 4th)
- 14-8/5: SmartBuilding – Integrating App with Server side and Architmate editor (Discussion of Oblig 3)

- May (15-17): (Bringing it together)
- 15-15/5: Summary of the course – Final demonstrations
- 16-22/5: Previous exams – group collaborations (No lecture)
- 17-29/5: Conclusions, Preparations for the Exam by old exams
- June (Exam)
- 13/6: Exam (4 hours), June 13th, 0900-1300

OMG Model-Driven Architecture (MDA)

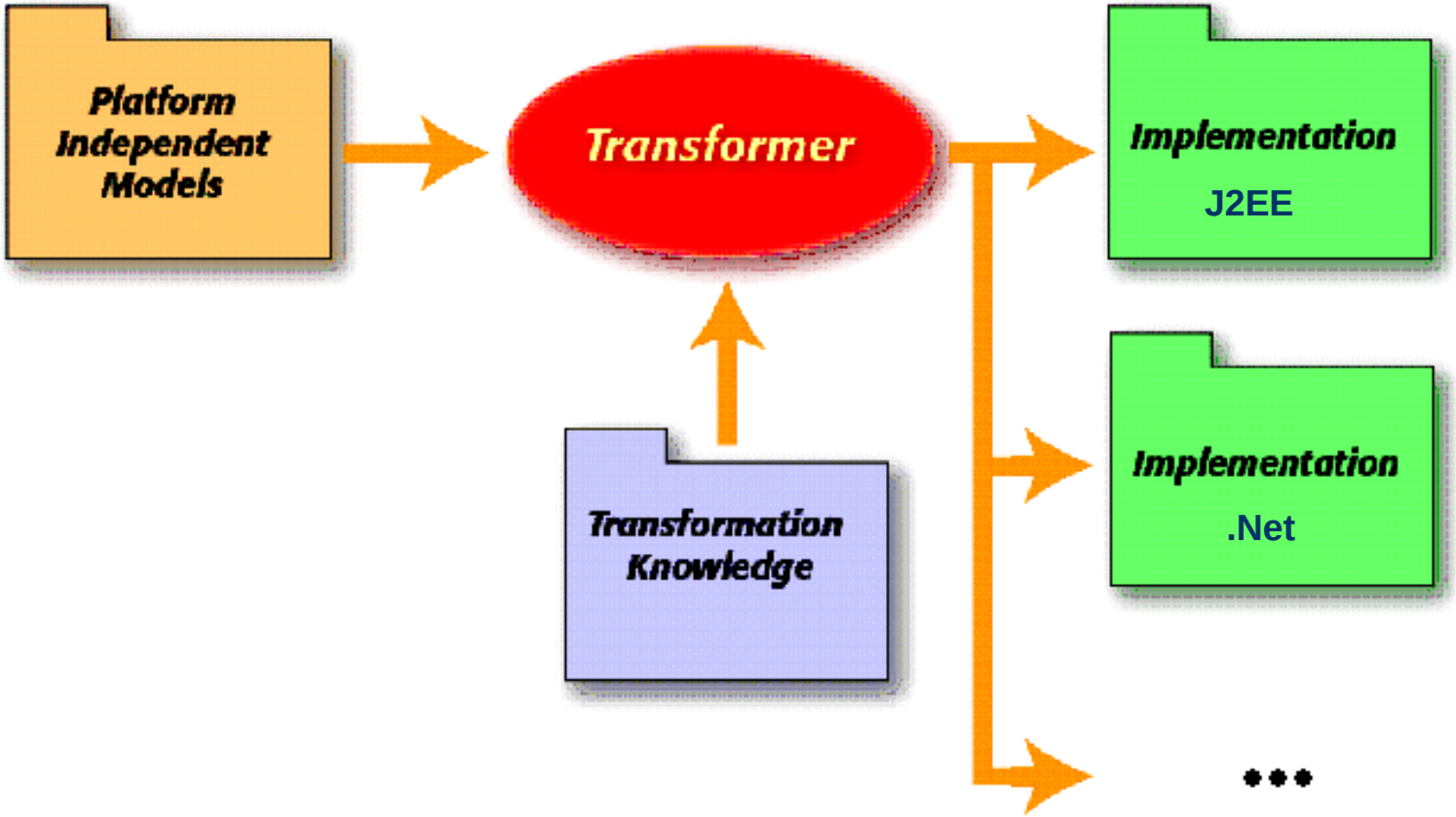
www.omg.org/mda



Model-driven – a definition

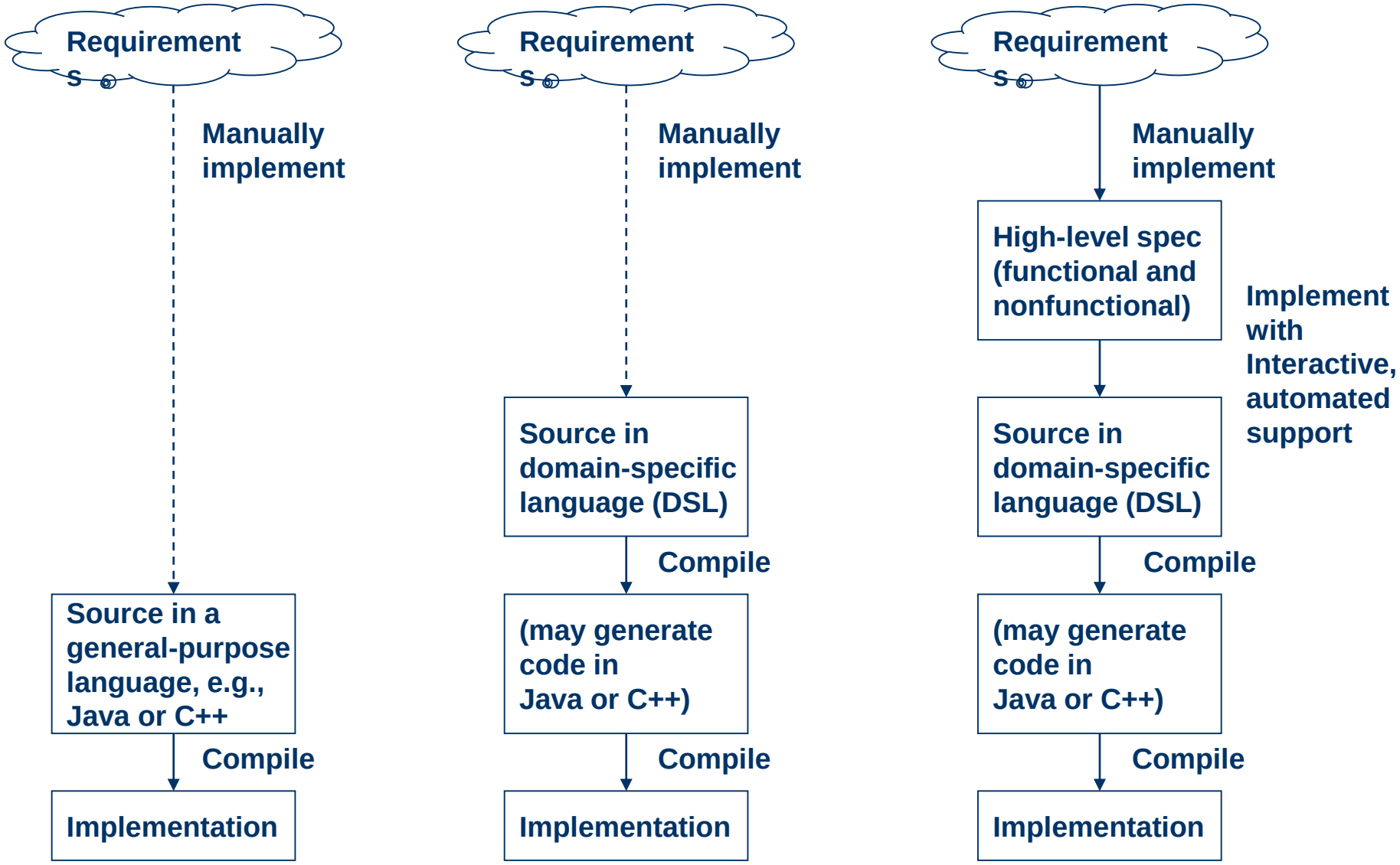
- A system development process is model-driven if
  - the development is mainly carried out using conceptual models at different levels of abstraction and using various viewpoints
  - it distinguishes clearly between platform independent and platform specific models
  - models play a fundamental role, not only in the initial development phase, but also in maintenance, reuse and further development
  - models document the relations between various models, thereby providing a precise foundation for refinement as well as transformation

MDA From 30.000 Feet

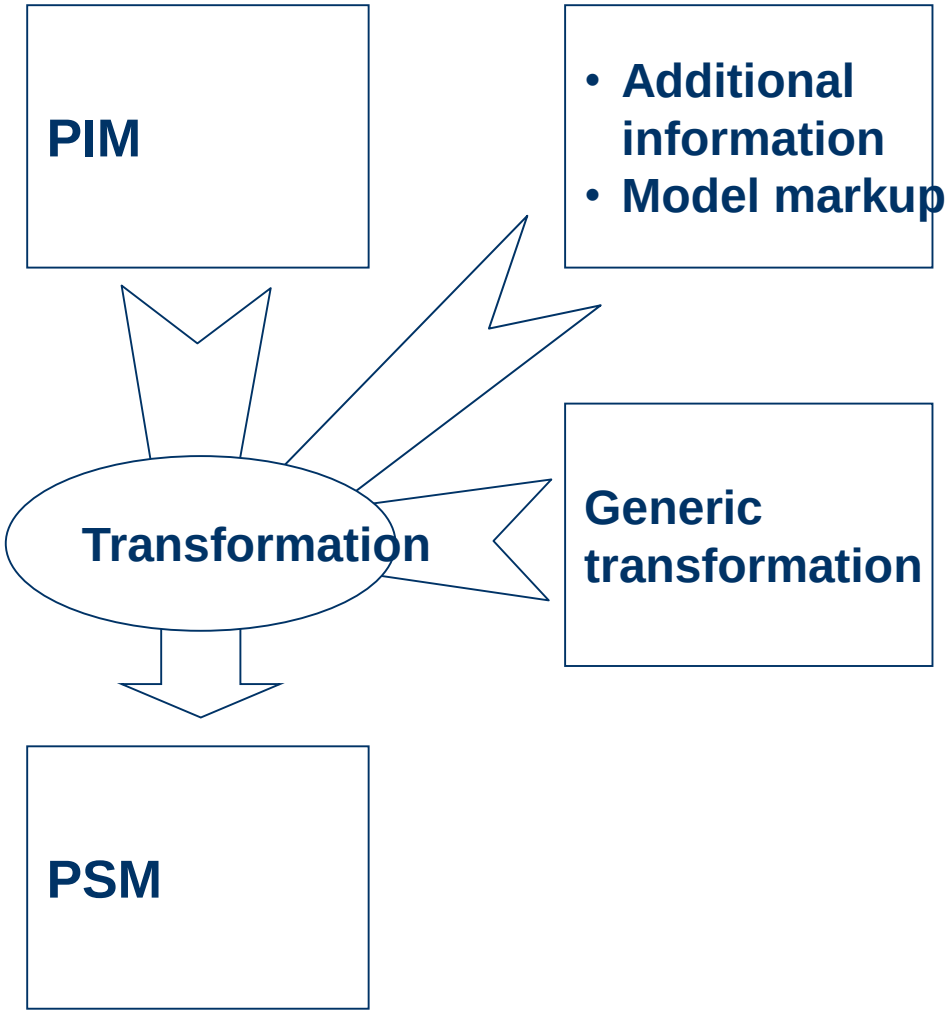


- A PIM can be retargeted to different platforms
- Not the only reason why MDA might be of interest to you…

Automation in Software Development



Basic MDA Pattern



- Generic transformations
  - Implement best practices, architectural and design patterns, technology patterns (e.g., J3EE patterns), optimizations, etc.
- Additional information
  - Adjust the transformation globally
  - Similar to compiler options
- Model markup
  - Direct the transformation of particular model elements
  - Not part of the PIM
  - Different platform mappings may require different markup
  - Similar to compiler pragmas

Goals

- The three primary goals of MDA are portability, interoperability and reusability.
- The MDA starts with the well-known and long established idea of separating the specification of the operation of the system from the details of the way the system uses the capabilities of its software execution platform (e.g. JEE, CORBA, Microsoft .NET and Web services).
- MDA provides an approach for:
  - specifying a system independently of the software execution platform that supports it;
  - specifying software execution platforms;
  - choosing a particular software execution platform for the system;
  - transforming the system specification into one for a particular software execution platform;

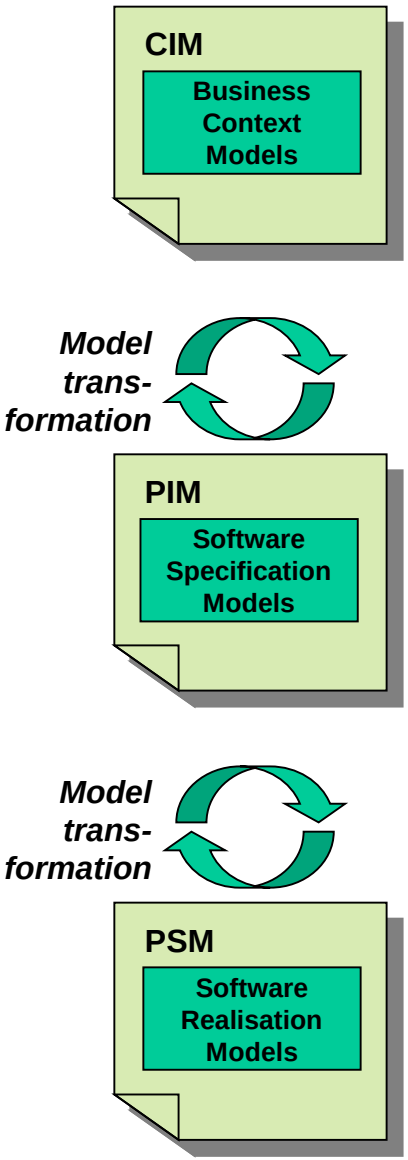
Basic concepts

- System
  - Existing or planned system.
  - System may include anything: a program, a single computer system, some combination of parts of different systems
- Model
  - A model of a system is a description or specification of that system and its environment for some certain purpose.
  - A model is often presented as a combination of drawings and text.
- Architecture
  - The architecture of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors.
  - MDA prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.
- Viewpoint
  - A viewpoint on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system.
- View
  - A viewpoint model or view of a system is a representation of that system from the perspective of a chosen viewpoint.
- Platform
  - A platform is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.

MDA – Three main abstraction levels

- **Computation independent model (CIM)**
  - The computational independent viewpoint is focused on the environment of the system and on the specific requirements of the system.
  - A CIM represents the computational independent viewpoint.
  - The CIM hides the structural details and, of course, the details related to the targeted platform.
- **Platform independent model (PIM)**
  - A platform independent model is a view of the system from a platform independent viewpoint.
  - The platform independent viewpoint is focused on the operation of the system, hiding the platform specific details.
  - A PIM exhibits platform independence and is suitable for use with a number of different platforms of similar types.
  - The PIM gathers all the information needed to describe the behaviour of the system in a platform independent way.
- **Platform specific model (PSM)**
  - A platform specific model is a view of the system from the platform specific viewpoint.
  - A PSM combines the specifications in the PIM with the details that specify how the system uses a particular type of platform.
  - The PSM represents the PIM taking into account the specific platform characteristics.

Model-driven development (MDD)



Model-driven approach to system engineering where models are used in

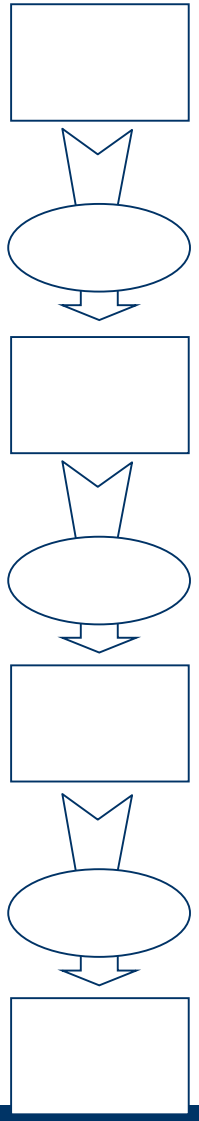
- understanding
- design
- construction
- deployment
- operation
- maintenance
- modification

Model transformation tools and services are used to align the different models.

Business-driven approach to system engineering where models are refined from business needs to software solutions

- Computation independent model (CIM) capturing business context and business requirements
- Platform independent model (PIM) focusing on software services independent of IT technology
- Platform specific model (PSM) focusing on the IT technology realisation of the software services

Basic MDA Pattern

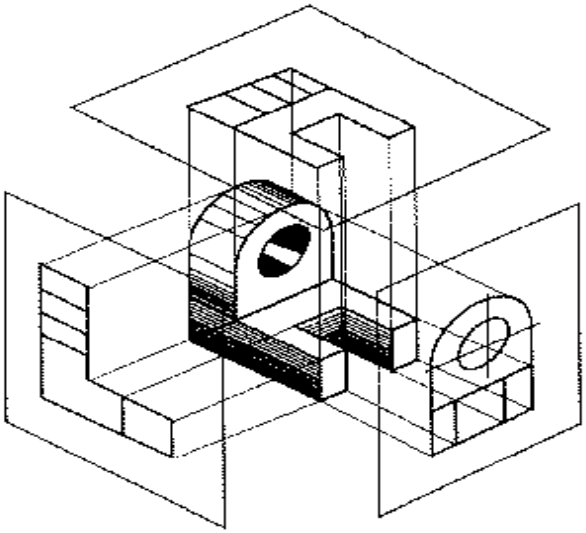


- The basic pattern can be applied multiple times
- PIMs and PSMs are relative notions
  - “Someone’s PIM can be someone else’s PSM”
- Platform independence is relative, too
  - It’s a scoping issue
  - It’s a strategic decision

Role of Models

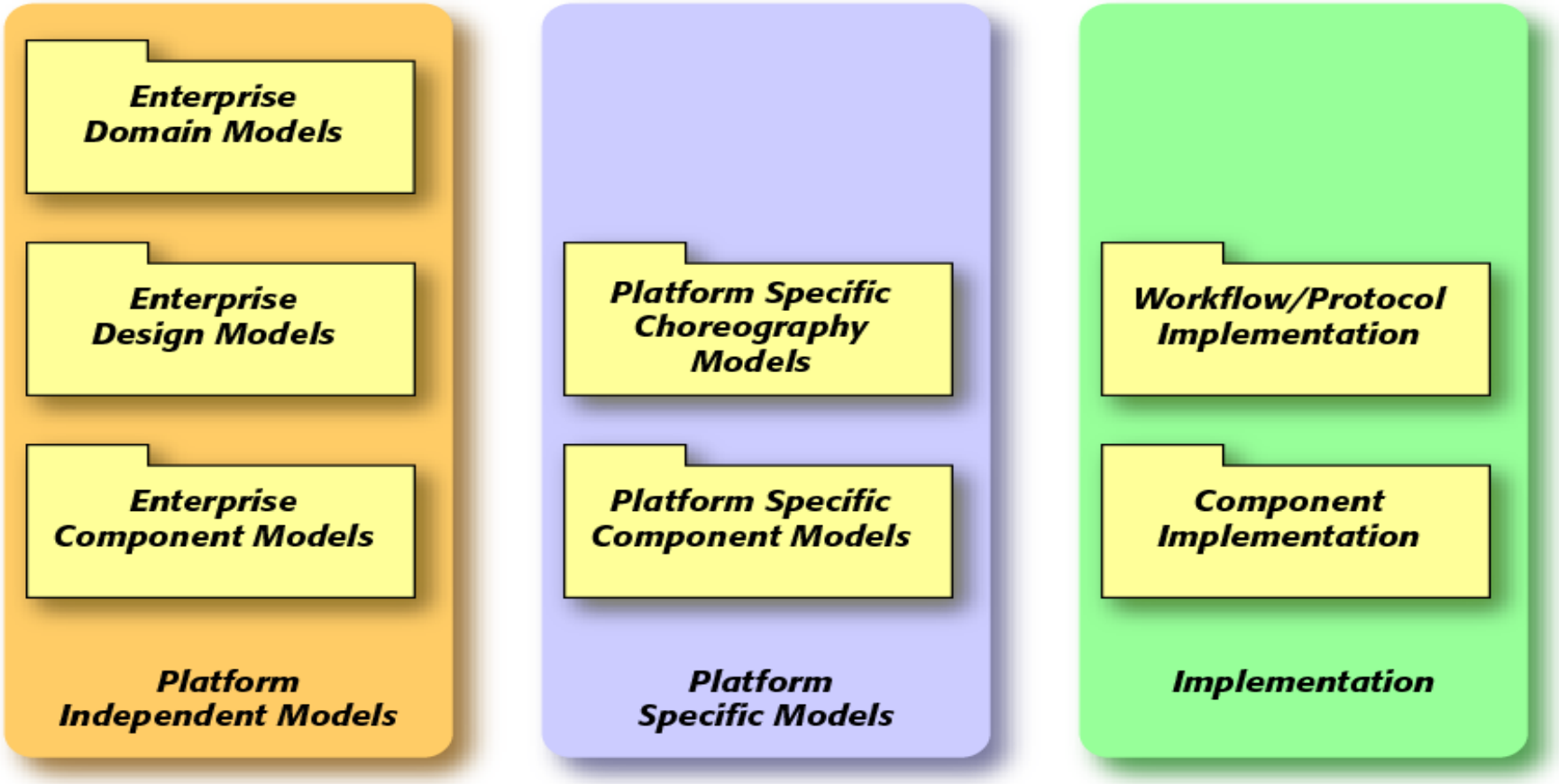
- Capture design information that is usually absent from code and lost during development
- Basis for
  - System generation
  - Analysis
  - Simulation
  - Test generation
  - Documentation generation
  - …
- *Domain-specificity* of a modeling language strengthens its capabilities for generation, optimization, early error detection, etc.

Viewpoints and Views



- System models are organized into multiple views
  - Different abstraction levels
  - Different aspects (e.g., workflow, domain concepts, deployment)
- Each view conforms to some viewpoint that prescribes some appropriate modeling notation
- Each viewpoint is relevant to some stakeholder

Many different views…



MDA-Related Standards

- OMG Standards
  - Modeling – UML
  - Metamodeling – MOF
  - Action semantics
  - Model interchange – XMI
  - Diagram interchange
  - Human-readable textual notation – HUTN
  - Model-based testing and debugging
  - (CWM)
  - …
- Java Community Process (JCP) Standard
  - Java Metadata Interface – JMI

Benefits of MDA

- Preserving the investment in knowledge
  - Independent of implementation platform
  - Tacit knowledge made explicit
- Speed of development
  - Most of the implementation is generated
- Quality of implementation
  - Experts provide transformation templates
- Maintenance and documentation
  - Design and analysis models are not abandoned after writing
  - 100% traceability from specification to implementation

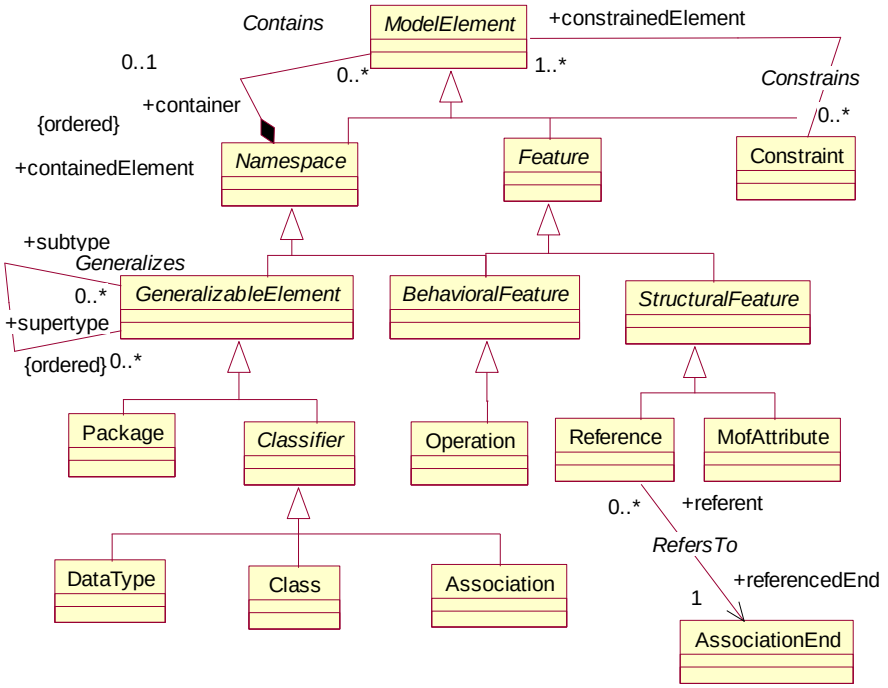
Domain specific modelling languages

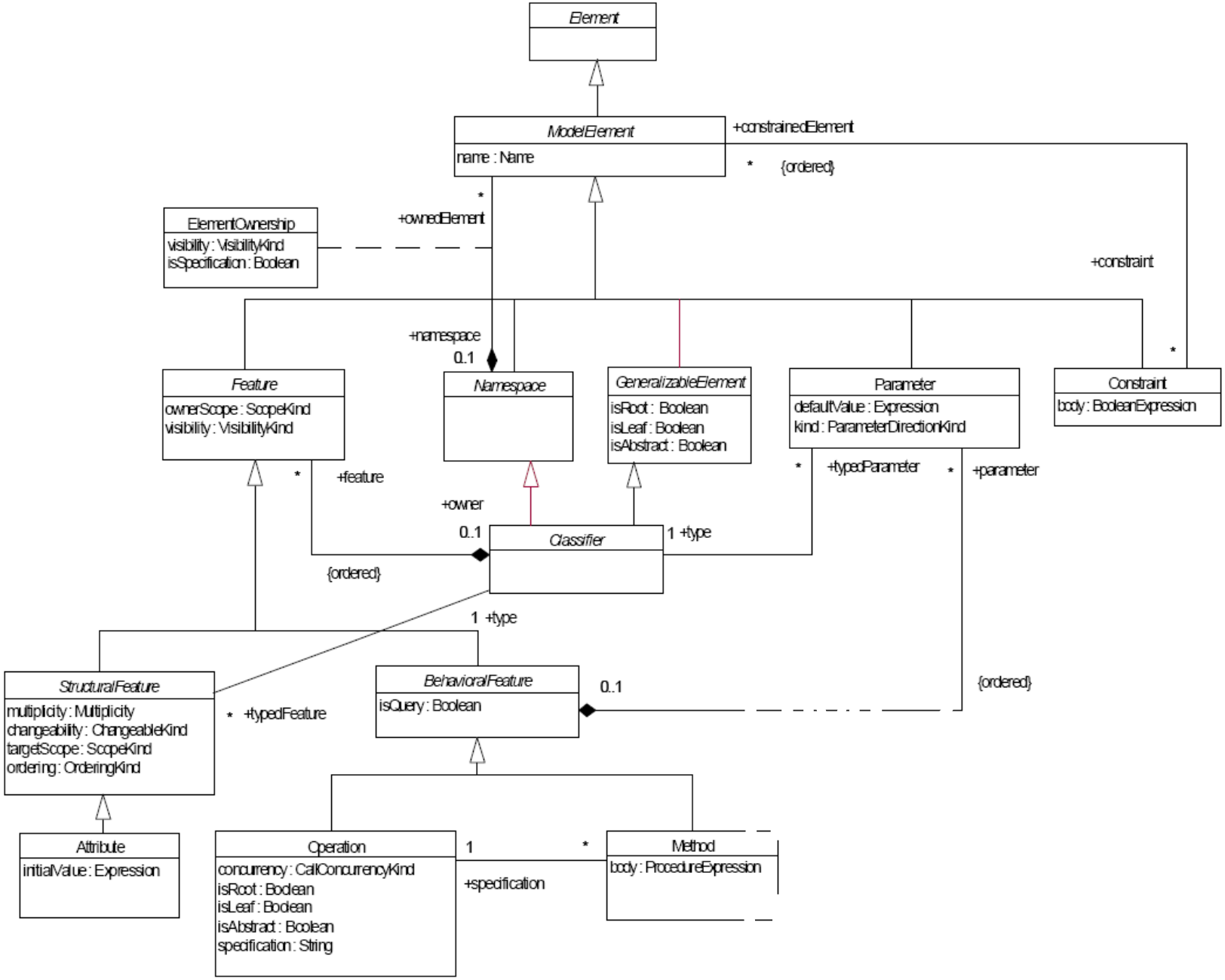
- Specific to a domain
- More focussed purpose
- Usable by the domain experts
- More productive than general purpose
  - If properly designed and tooled!
- UML profiles vs. DSL

Assigning Meaning to Models

- If a model *is no longer* just
  - fancy pictures to decorate your room
  - a graphical syntax for C++/Java/C#/Eiffel...
- Then tools must be able to manipulate models
  - Let's make a model

  of what a model is!
  - => ***meta-modeling***
    - & meta-meta-modeling..
    - Use Meta-Object Facility (MOF) to avoid infinite Meta-recursion




Element
ModelElement
name : Name
+constrainedElement
* {ordered}
ElementOwnership
visibility : VisibilityKind
isSpecification : Boolean
* +ownedElement
+constraint
+namespace
0..1
Feature
ownerScope : ScopeKind
visibility : VisibilityKind
Namespace
GeneralizableElement
isRoot : Boolean
isLeaf : Boolean
isAbstract : Boolean
Parameter
defaultValue : Expression
kind : ParameterDirectionKind
Constraint
body : BooleanExpression
* +feature
{ordered}
+owner
0..1
Classifier
1 +type
* +typedParameter
* +parameter
1 +type
StructuralFeature
multiplicity : Multiplicity
changeability : ChangeableKind
targetScope : ScopeKind
ordering : OrderingKind
* +typedFeature
BehavioralFeature
isQuery : Boolean
0..1
{ordered}
Attribute
initialValue : Expression
Operation
concurrency : CallConcurrencyKind
isRoot : Boolean
isLeaf : Boolean
isAbstract : Boolean
specification : String
1 * +specification
Method
body : ProcedureExpression

Generalizations

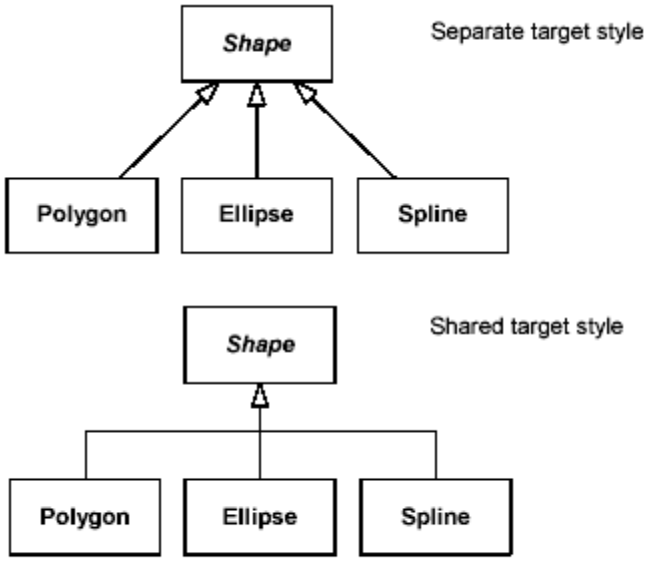


Figure 3-33. Examples of generalizations between classes.

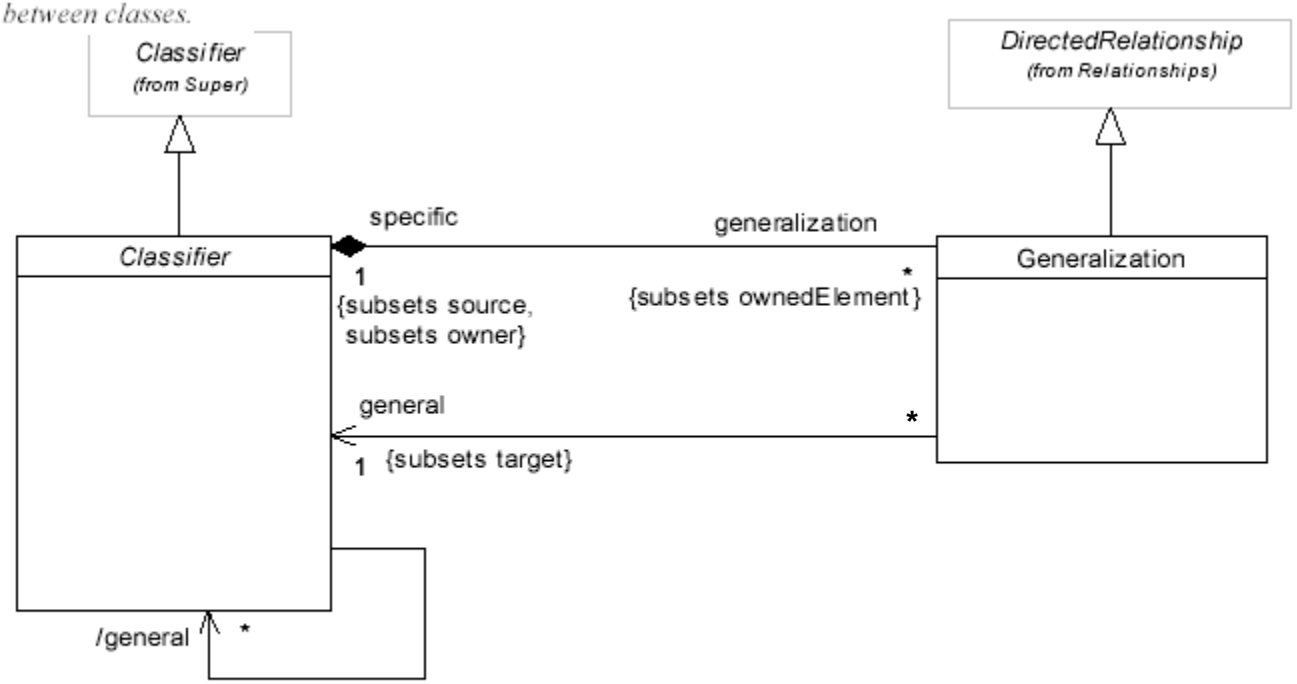
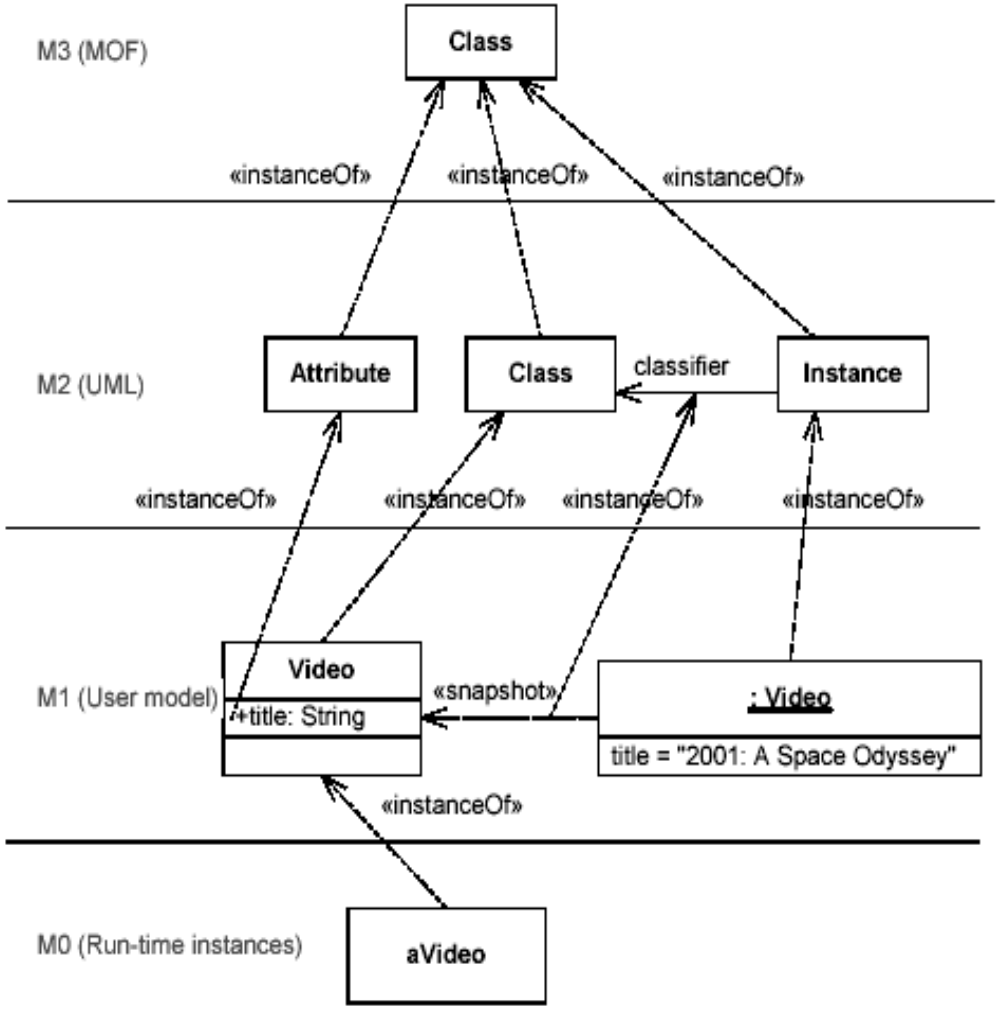
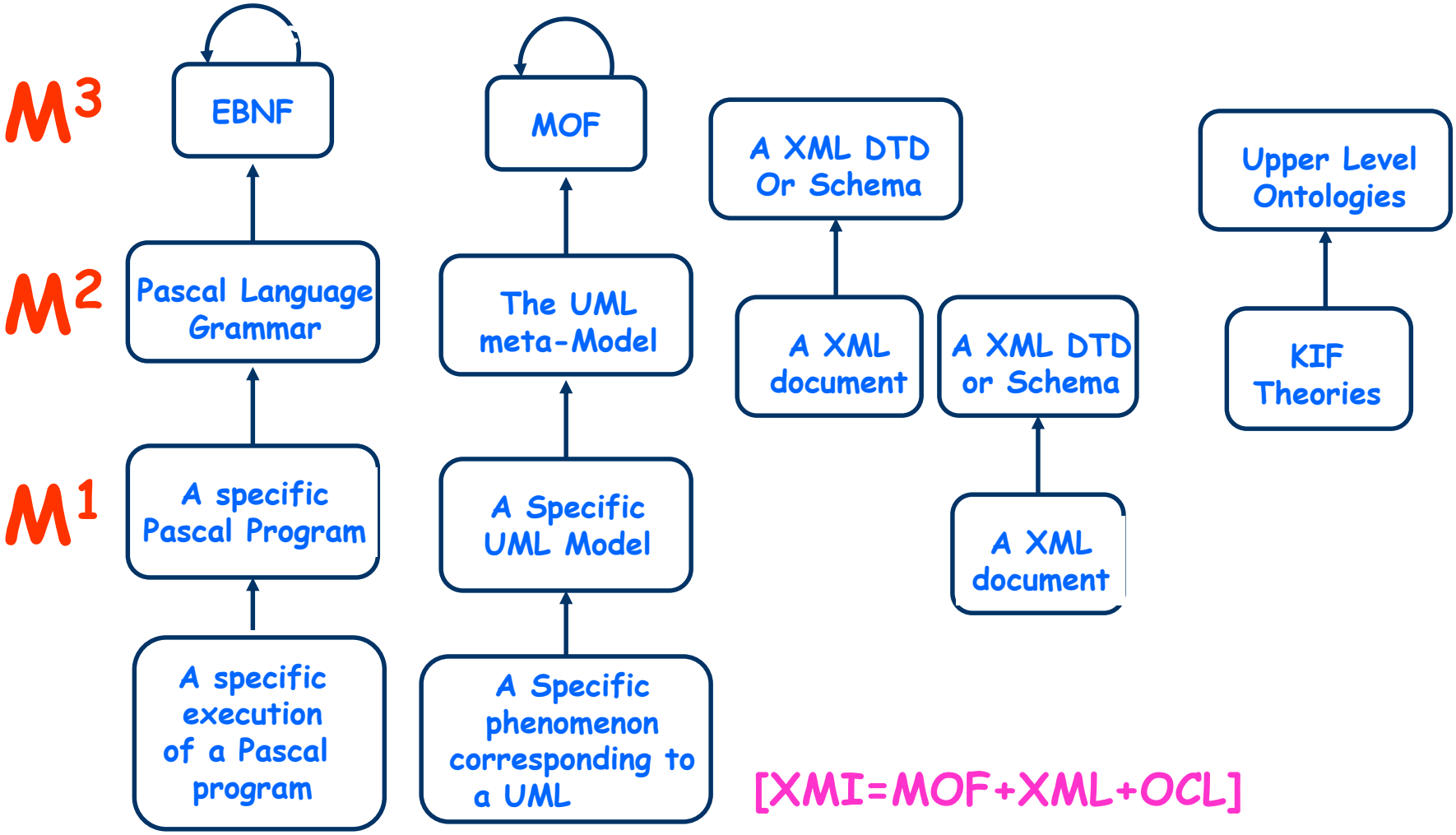


Figure 3-32. The elements defined in the Generalizations package.

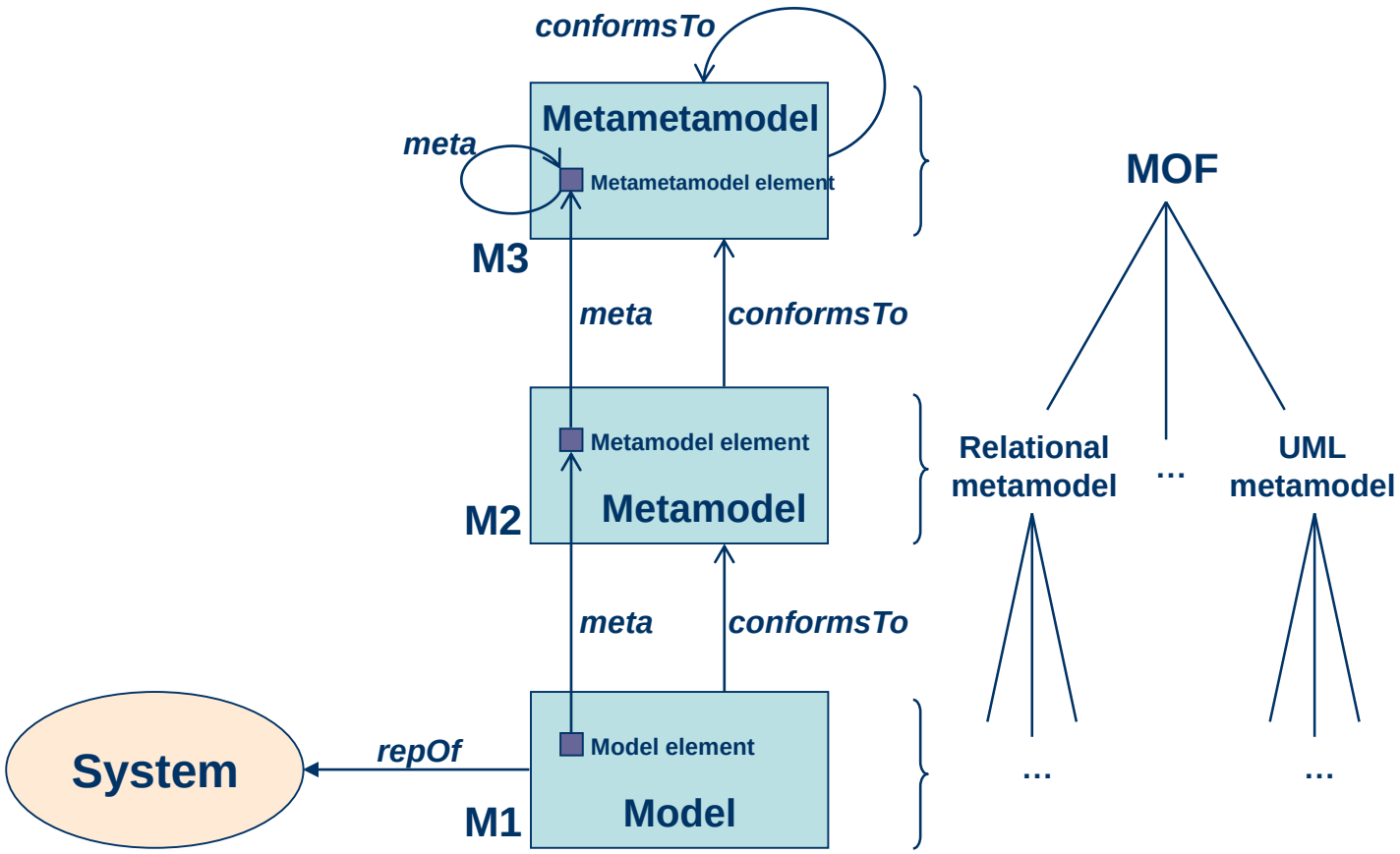
The 4 layers in practice



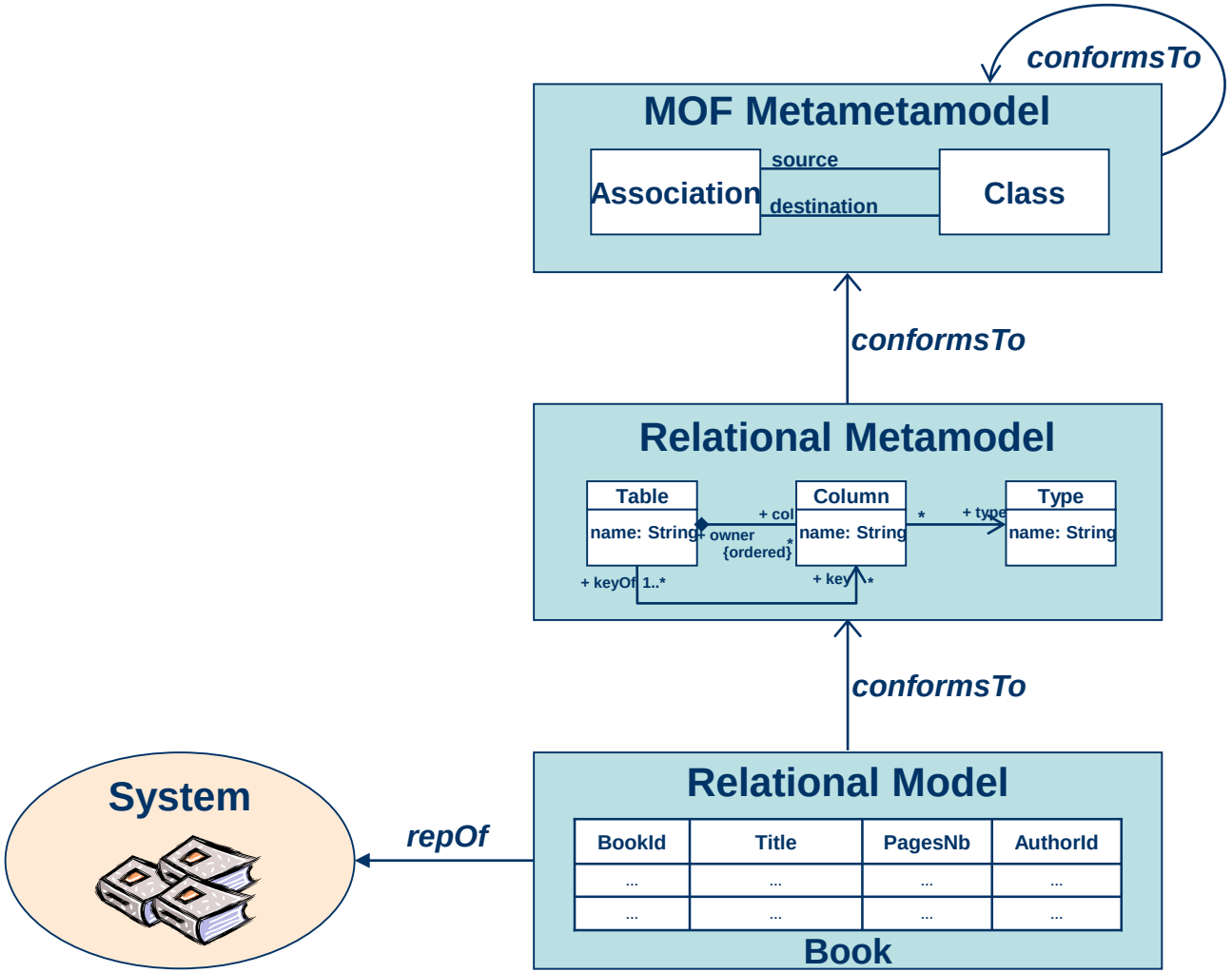
Comparing Abstract Syntax Systems



Model-Driven Architecture



Model-Driven Architecture: Example

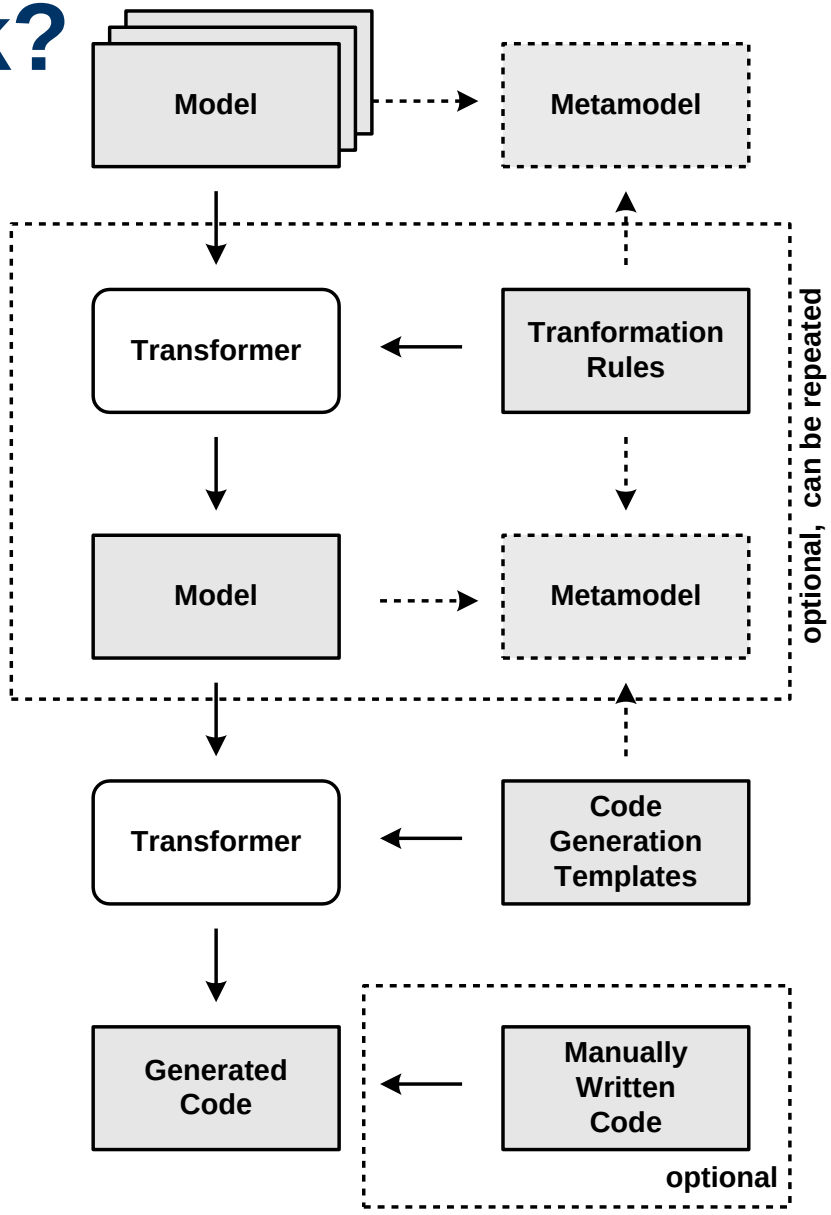


MDA technology standards

- **Unified Modeling Language (UML)**
  - UML is the de-facto standard industry language for specifying and designing software systems.
  - UML addresses the modelling of architecture and design aspects of software systems by providing language constructs for describing, software components, objects, data, interfaces, interactions, activities etc.
- **Meta Object Facility (MOF)**
  - MOF provides the standard modelling and interchange constructs that are used in MDA.
  - These constructs are a subset of the UML modelling constructs.
  - This common foundation provides the basis for model/metadata interchange and interoperability.
- **XML Metadata Interchange (XMI)**
  - XMI is a format to represent models in a structured text form.
  - In this way UML models and MOF metamodels may be interchanged between different modelling tools.
- **Common Warehouse Metamodel (CWM)**
  - CWM is the OMG data warehouse standard.
  - It covers the full life cycle of designing, building and managing data warehouse applications and supports management of the life cycle.
- **MOF Queries/View/Transformations (QVT)**
  - The goals of the QVT are to provide a standard specification of a language suitable for querying and transforming models which are represented according to a MOF metamodel.

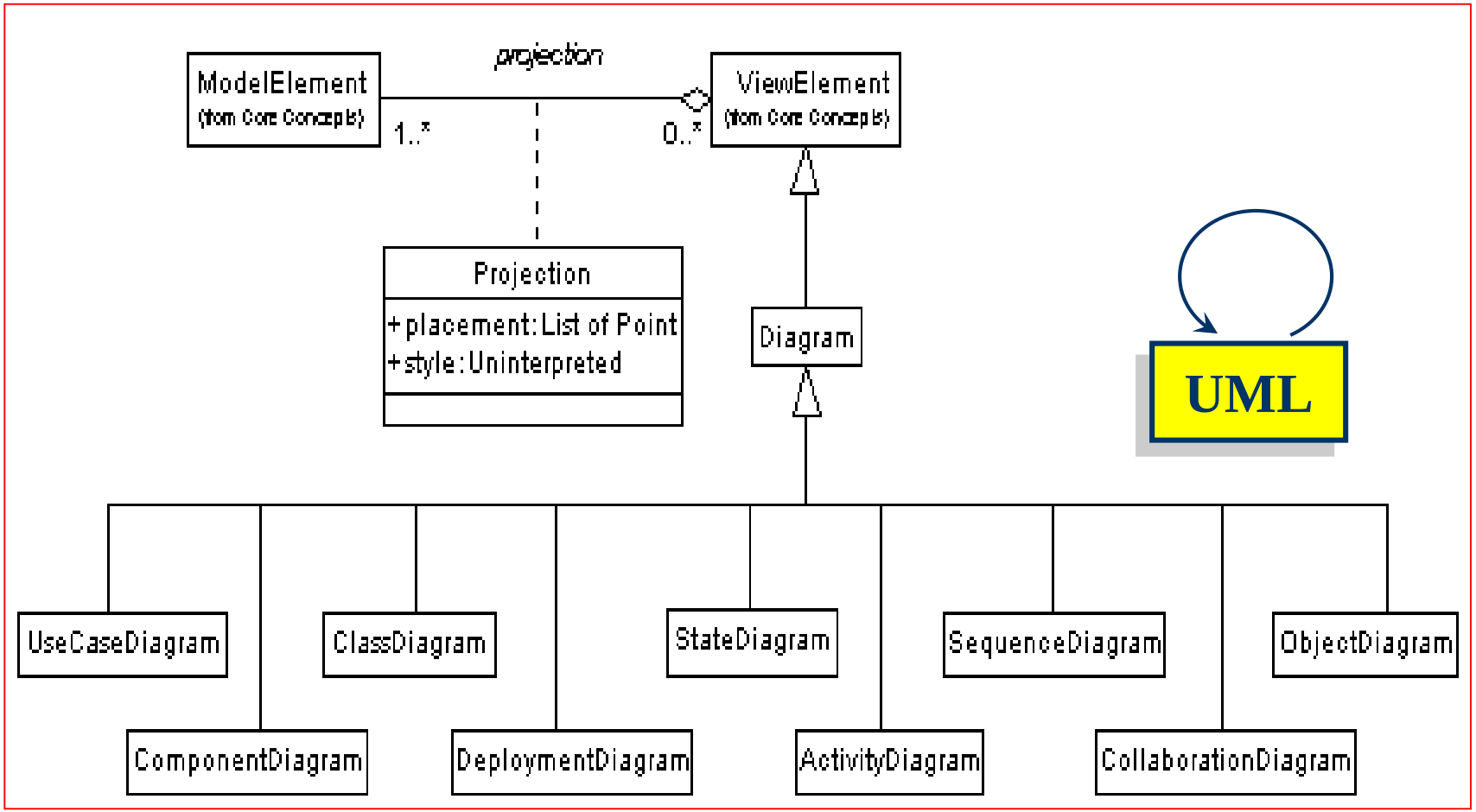
How does MDD work?

- Developer develops **model(s)** based on certain metamodel(s).
- Using **code generation templates**, the model is transformed to executable code.
- Optionally, the **generated code is merged** with manually written code.
- One or more **model-to-model transformation steps** may precede code generation.

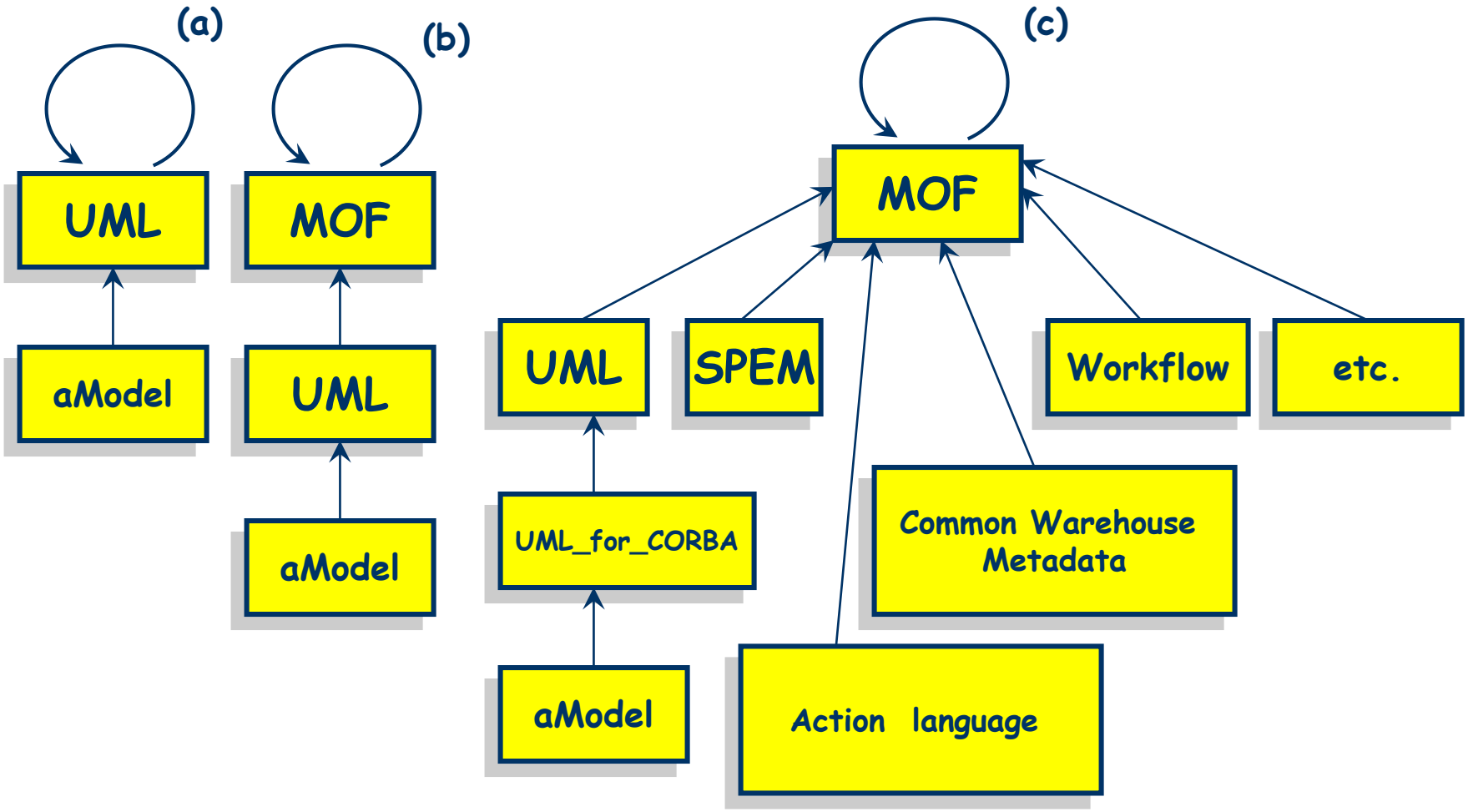


OMG MOF and metamodelling

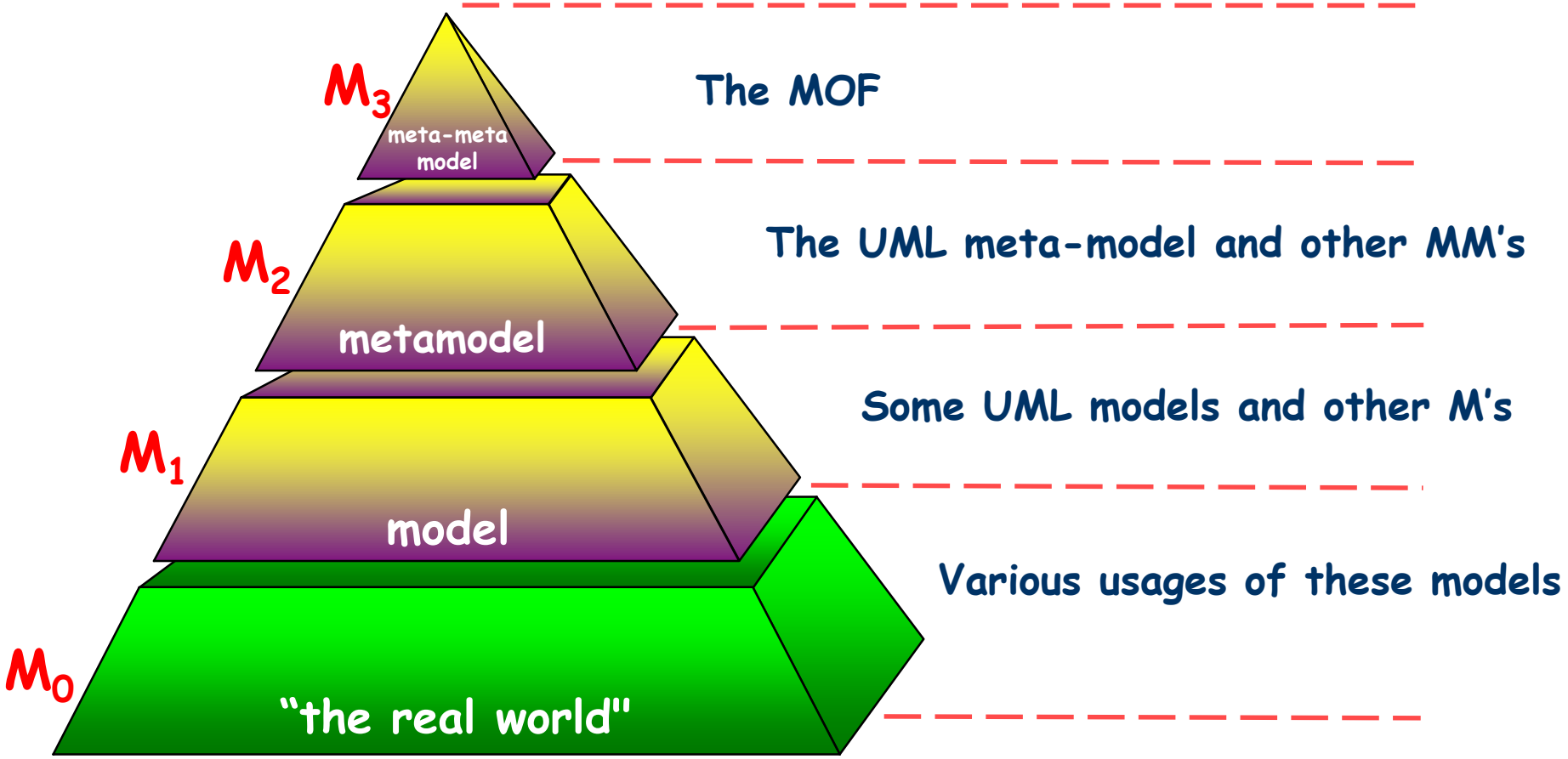
Fragments of a UML metamodel



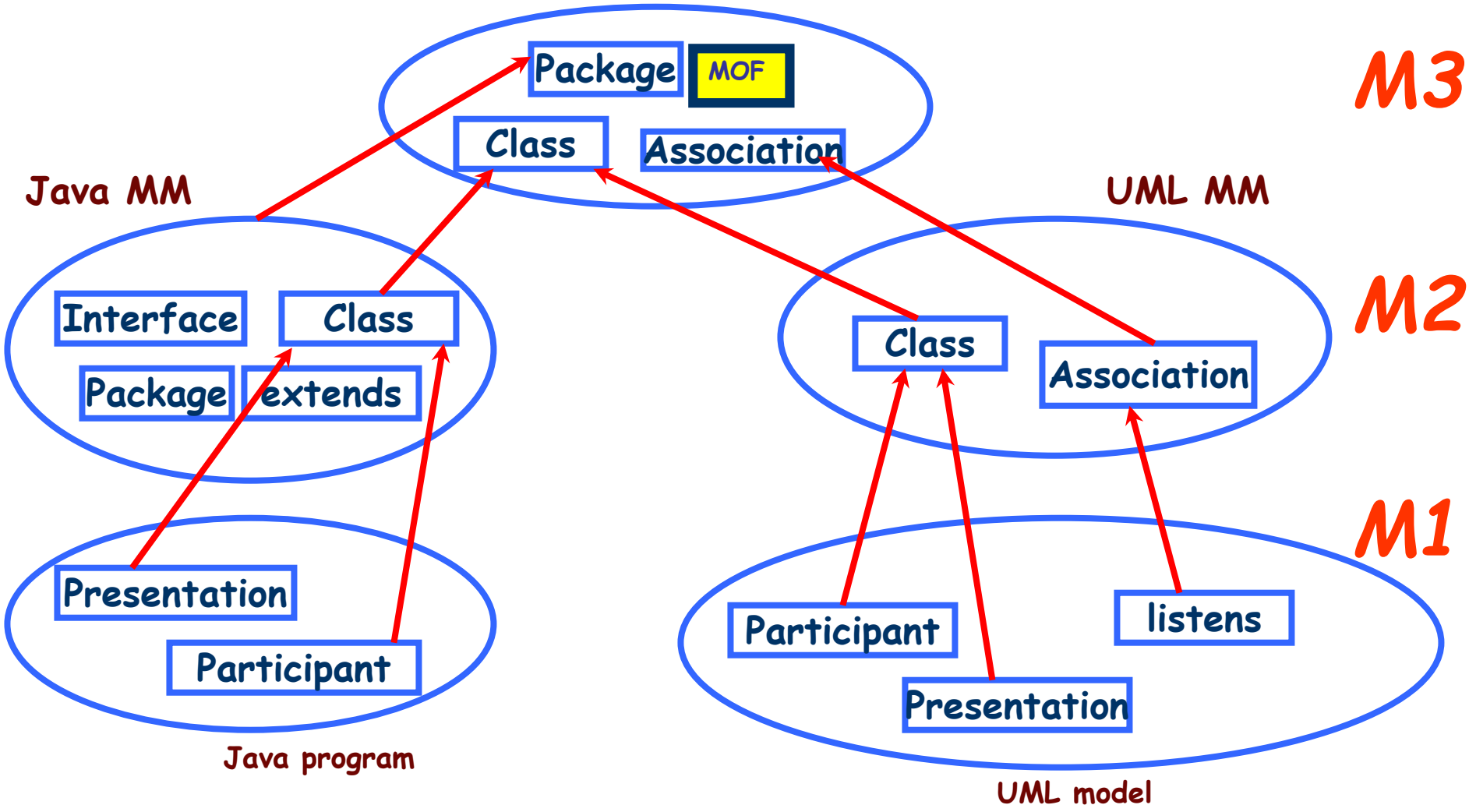
Three stages in the evolution of modelling techniques at the OMG.



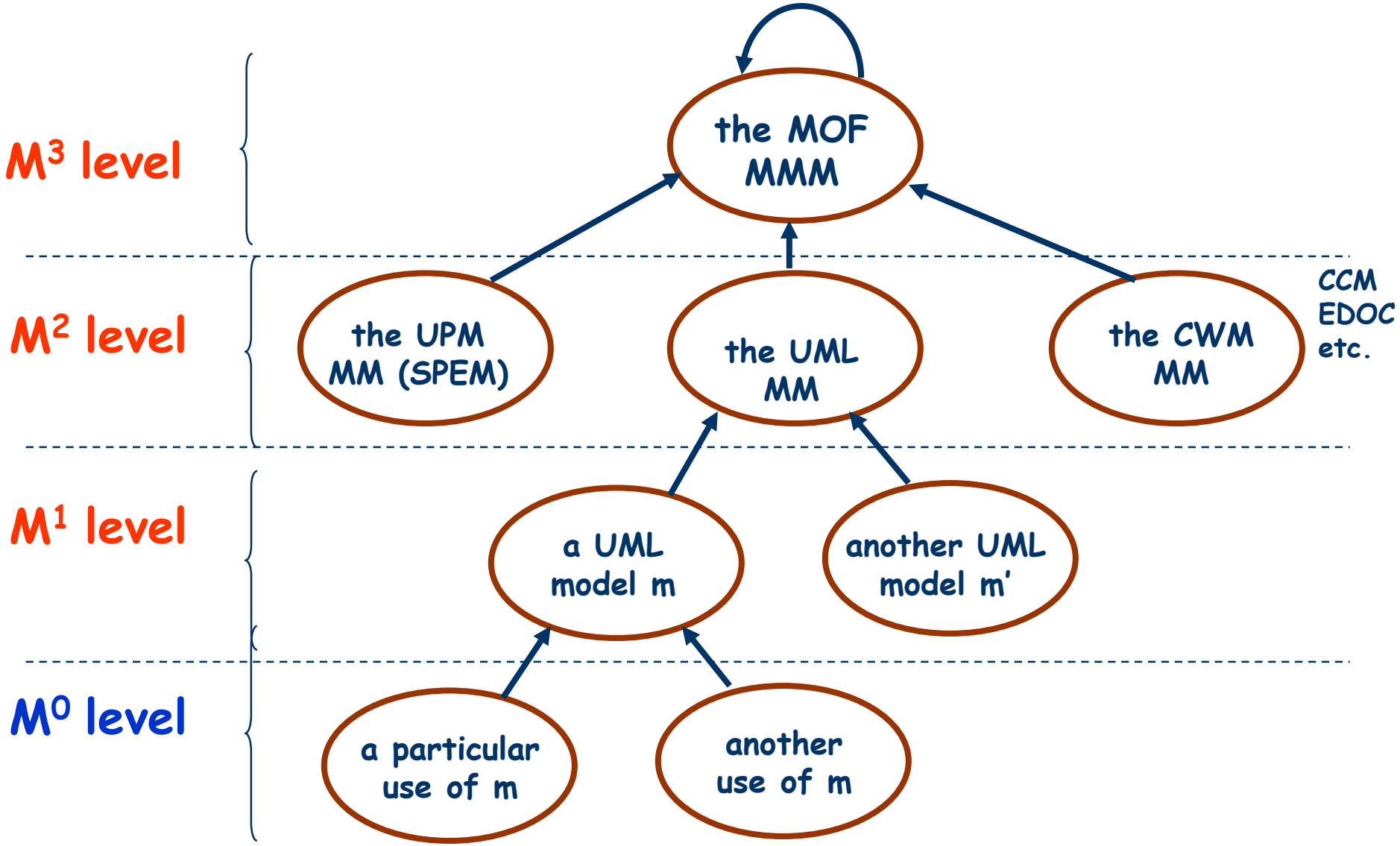
Egyptian architecture



Illustration



The three modelling levels



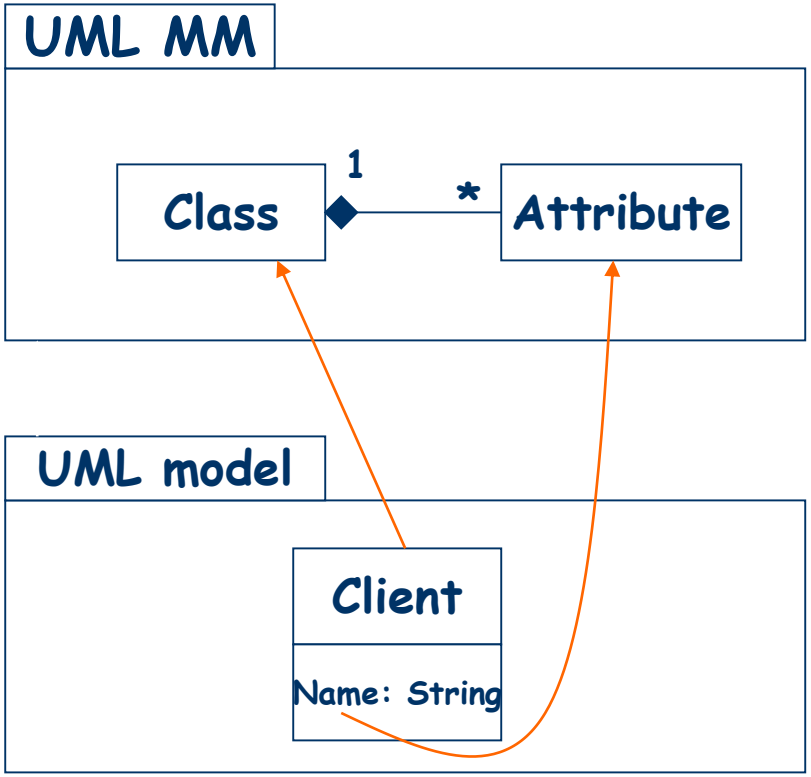
Model -> Metamodel

entity → meta-entity

relationship

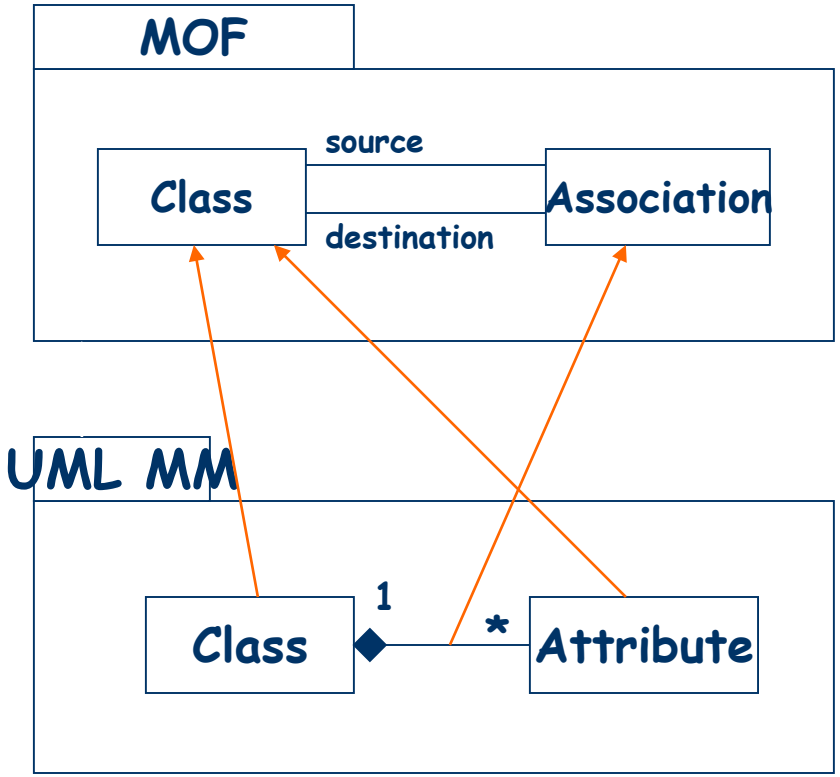
model → meta-model

relationship

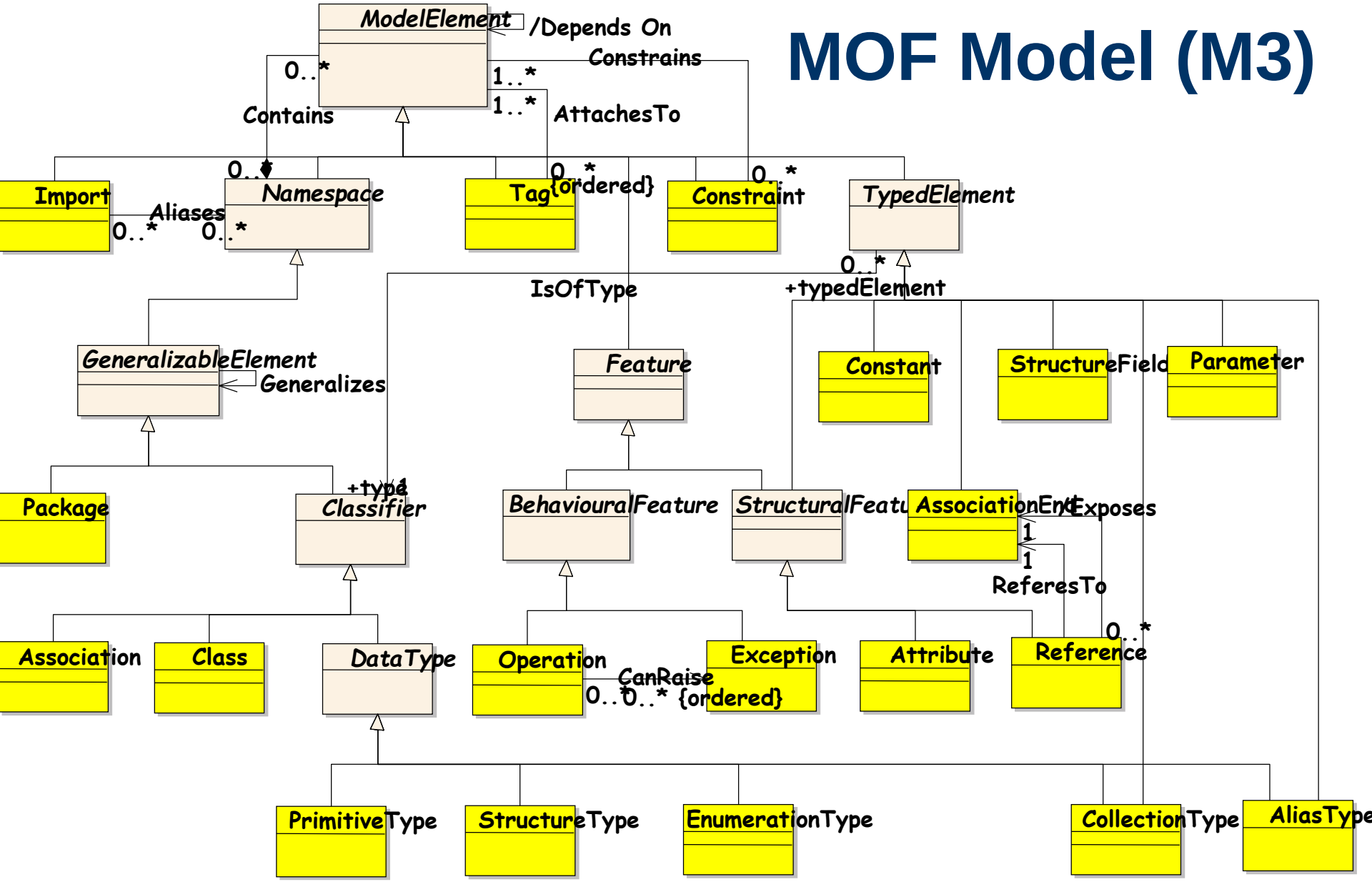


Metamodel -> Meta-metamodel

- entity → meta-entity relationship
- model → meta-model relationship



MOF Model (M3)


ModelElement
/Depends On
Constrains
0..*
1..*
1..*
Contains
AttachesTo
Import
Aliases
0..*
0..*
Namespace
0..*
Tag
{ordered}
0..*
Constraint
TypedElement
0..*
+typedElement
IsOfType
GeneralizableElement
Generalizes
Feature
Constant
StructureField
Parameter
Package
+type
1
Classifier
BehaviouralFeature
StructuralFeatu
AssociationEnd
Exposes
1
1
ReferesTo
0..*
Association
Class
DataType
Operation
CanRaise
0..*
0..*
{ordered}
Exception
Attribute
Reference
PrimitiveType
StructureType
EnumerationType
CollectionType
AliasType

Goals & Challenges

- **Goals:**
  - We need an **end-to-end tool chain** that allows us to build models, verify them and generate various artefacts from them.
  - All of this should happen in a homogeneous environment, namely Eclipse.
- **Challenges:**
  - **Good Editors** for your models
  - **Verifying** the models as you build them
  - **Transforming/Modifying** models
  - **Generating** Code
  - **Integrating** generated and non-generated code

Applied metamodelling

Metamodels

- A metamodel is just another model (e.g. written in UML)
  - Model of a set of models
- Metamodels are specifications
  - Models are valid if no false statements according to metamodel (e.g. well-formed)
  - Metamodels typically represents domain-specific models (real-time systems, safety critical systems, e-business)
- The domain of metamodelling is language definition
  - A metamodel is a model of some part of a language
  - Which part depends on how the metamodel is to be used
  - Parts: syntax, semantics, views/diagrams, ...
- Meta-metamodel
  - Model of metamodels
  - Reflexive metamodel, i.e., expressed using itself
  - Minimal reflexive metamodel

What is a metamodel?

- In its broadest sense, a metamodel is a model of a modelling language.
- The term ”meta” means transcending or above, emphasising the fact that a metamodel describes a modelling language at a higher level of abstraction than the modelling language itself.
- In order to understand what a metamodel is, it is useful to understand the difference between a metamodel and a model.
- Whilst a metamodel is also a model, a metamodel has two main distinguishing characteristics.
  - Firstly, it must capture the essential features and properties of the language that is being modelled.
    - Thus, a metamodel should be capable of describing a language’s concrete syntax, abstract syntax and semantics.
  - Secondly, a metamodel must be part of a metamodel architecture.
    - Just as we can use metamodels to describe the valid models or programs permitted by a language, a metamodel architecture enables a metamodel to be viewed as a model, which itself is described by another metamodel.
    - This allows all metamodels to be described by a single metamodel.
    - This single metamodel, sometimes known as a meta-metamodel, is the key to metamodelling as it enables all modelling languages to be described in a unified way.

Why metamodel?

- System development is fundamentally based on the use of languages to capture and relate different aspects of the problem domain.
- The benefit of metamodelling is its ability to describe these languages in a unified way.
  - This means that the languages can be uniformly managed and manipulated thus tackling the problem of language diversity.
  - For instance, mappings can be constructed between any number of languages provided that they are described in the same metamodelling language.
- Another benefit is the ability to define semantically rich languages that abstract from implementation specific technologies and focus on the problem domain at hand.
  - Using metamodels, many different abstractions can be defined and combined to create new languages that are specifically tailored for a particular application domain.
  - Productivity is greatly improved as a result.

Uses for a metamodel

- Define the syntax and semantics of a language.
- Explain the language.
- Compare languages rigorously.
- Specify requirements for a tool for the language.
- Specify a language to be used in a meta-tool.
- Enable interchange between tools.
- Enable mapping between models.

The metamodelling process

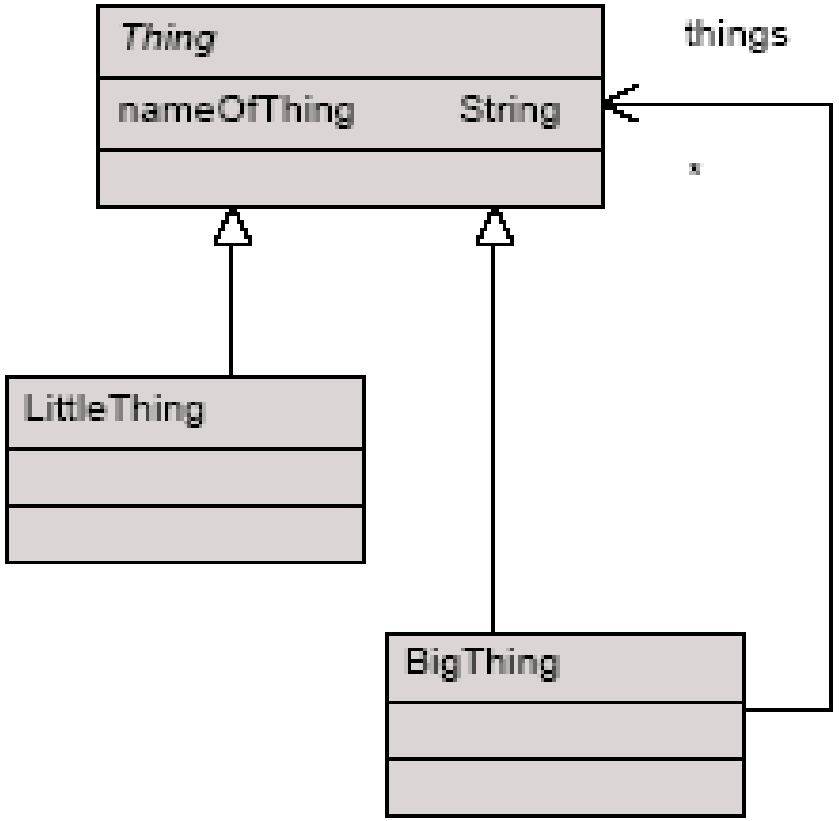
- There is a clearly defined process to constructing metamodels, which does at least make the task a well-defined, if iterative, process.
- The process has the following basic steps:
  - defining abstract syntax
  - defining well-formedness rules and meta-operations
  - defining concrete syntax
  - defining semantics
  - constructing mappings to other languages

Abstract syntax

- The metamodel describes the abstract syntax of a language.
- The abstract syntax of a language describes the vocabulary of concepts provided by the language and how they may be combined to create models.
- It consists of a definition of the concepts, the relationships that exist between concepts and well-formedness rules that state how the concepts may be legally combined.

Concrete syntax – visual

- A visual syntax presents a model or program in a diagrammatical form.
- A visual syntax consists of a number of graphical icons that represent views on an underlying model.
- A good example of a visual syntax is a class diagram, which provides graphical icons for class models.
- The visual syntax shown in the figure (left) is particularly good at presenting an overview of the relationships and concepts in a model.

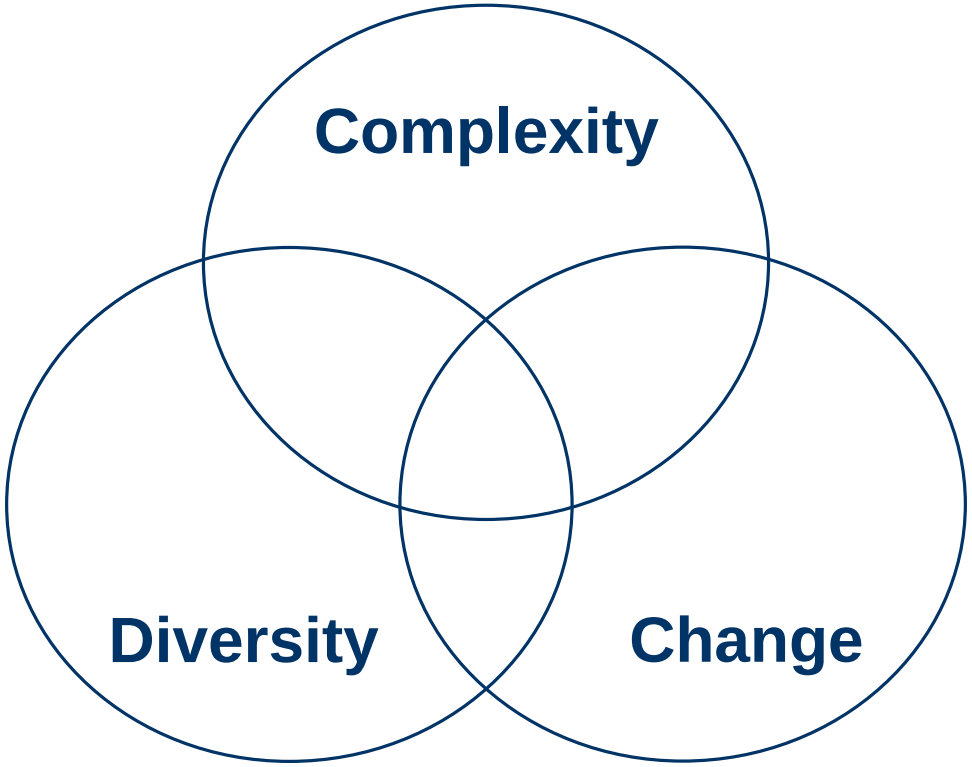


Concrete syntax – textual

- A textual syntax enables models or programs to be described in a structured textual form.
- A textual syntax can take many forms, but typically consists of a mixture of declarations, which declare specific objects and variables to be available, and expressions, which state properties relating to the declared objects and variables.
- The following Java code illustrates a textual syntax that includes a class with a local attribute declaration and a method with a return expression:

```
public abstract class Thing
  {
  private String nameOfThing;
  public String getName()
    {return nameOfThing;}
  }
```

Challenges facing developers



Language-driven development – Providing the solution

- Execution
  - allows the model or program to be tested, run and deployed
- Analysis
  - provides information of the properties of models and programs
- Testing
  - support for both generating test cases and validating them must be provided
- Visualisation
  - many languages have a graphical syntax, and support must be provided for this via the user interface to the language
- Parsing
  - if a language has a textual syntax, a means must be provided for reading in expressions written in the language
- Translation
  - languages don't exist in isolation. They are typically connected together whether it is done informally or automatically through code generation or compilation
- Integration
  - it is often useful to be able to integrate features from one model or program into another, e.g. through the use of configuration management.

Language engineering and metamodelling

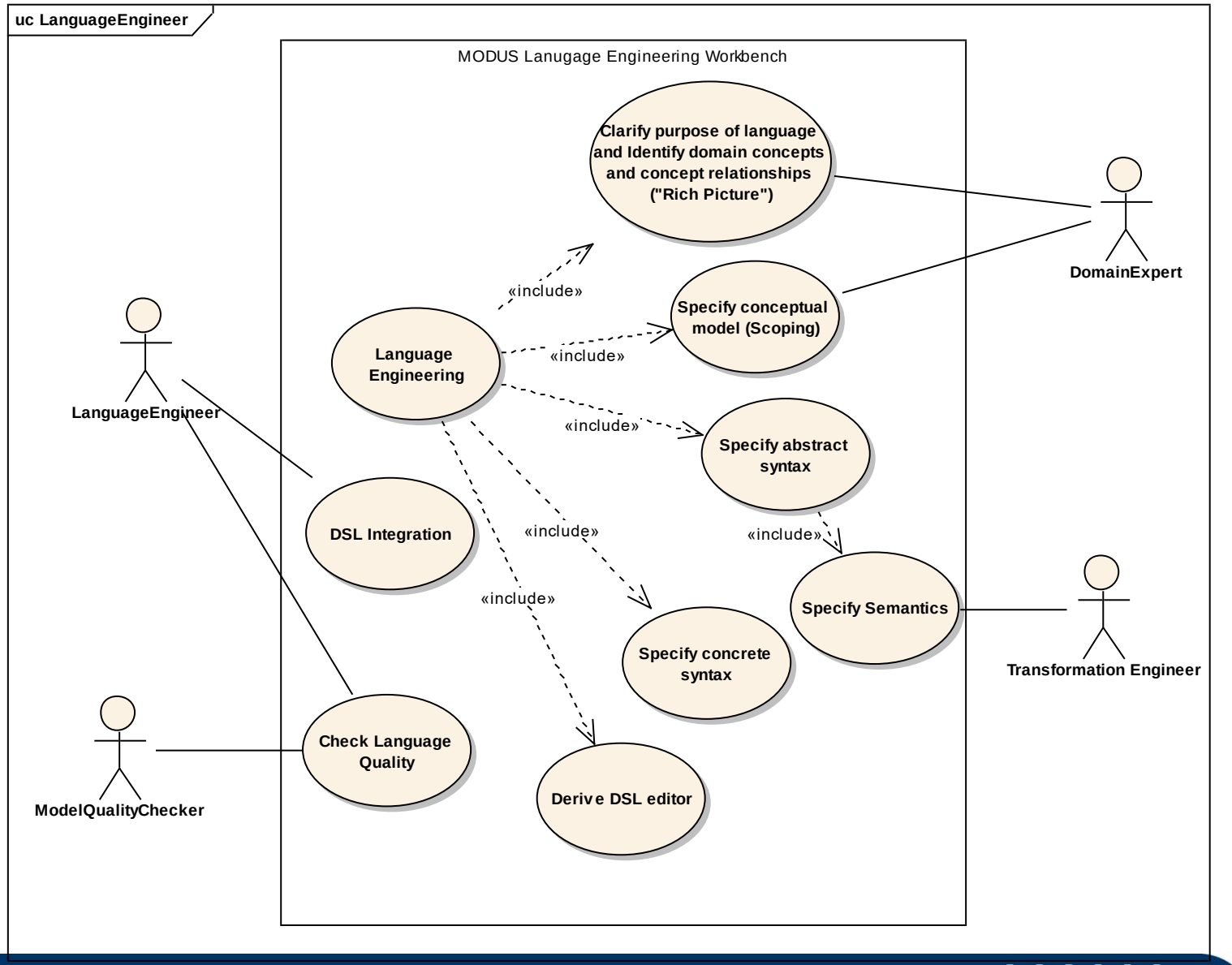
- In order to be able to engineer languages, we need a language for capturing, describing and manipulating all aspects of languages in a unified and semantically rich way.
- This language is called a metamodelling language.
- Metamodels (models of languages) are the primary means by which language engineering artefacts are expressed, and are therefore the foundation for language-driven development.

Semantics

- An abstract syntax conveys little information about what the concepts in a language actually mean.
- Therefore, additional information is needed in order to capture the semantics of a language.
- Defining a semantics for a language is important in order to be clear about what the language represents and means.
- Otherwise, assumptions may be made about the language that lead to its incorrect use.
- For instance, although we may have an intuitive understanding of what is meant by a state machine, it is likely that the detailed semantics of the language will be open to misinterpretation if they are not defined precisely.
  - What exactly is a state?
  - What does it mean for transition to occur?
  - What happens if two transitions leave the same state.
  - Which will be chosen?
- All these questions should be captured by the semantics of the language.

Language Engineering with Eclipse Modeling Framework (EMF)

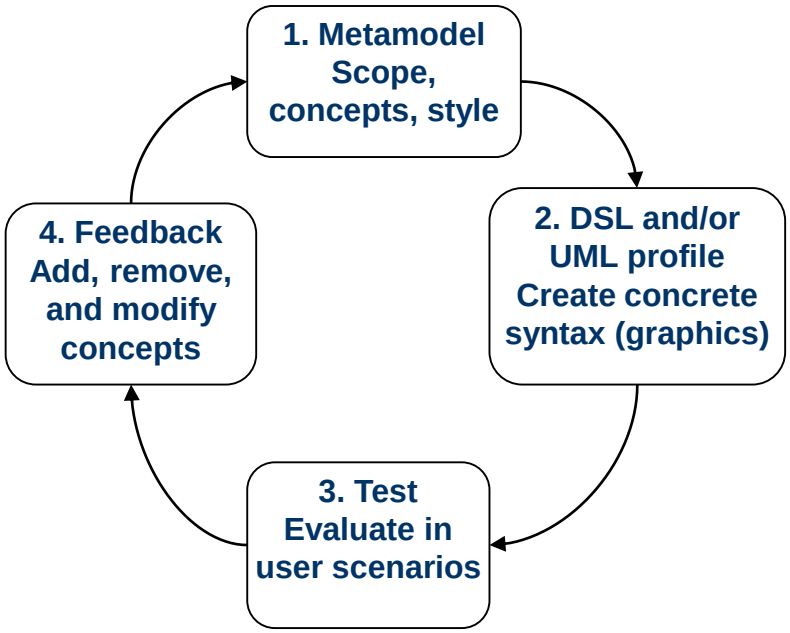
The Language Engineer responsibilities



Metamodel development

Understanding of concepts and requirements

- **Initial requirements**
- **Concepts**
- **Partitioning of the metamodel into structures**
- **Architectural style**
- **Document the metamodel and develop it in EMF (.ecore)**



Characteristics for metamodel

- Suited for target roles
  - Support domain concepts and scenarios of target roles
  - Ease-of-use and understandable for business modeller (use terms)
  - Support precise details and correctness for solution architect
- Avoid unnecessary complexity
  - Keep it simple stupid (KISS)
  - Number of elements and associations
  - Type and navigation of associations
- Make it modular
  - Provide core with extensions
  - Define and illustrate possible subsets (”dialects”) that support scenarios
  - Consider integration and extension points
- Suited for implementation
  - EMF representation
  - Transformation from/to UML profile
  - Transformation to PSM

Technology overview

OMG MDA specification	Eclipse technology	Comments
MOF	EMF	
UML	UML	
UML profile/DSL	GEF GMF	
QVT	ATL, MOFScript	
MOF2TExt	MOFScript	
SPEM	EPF	
XMI	EMF	

MDA-compliant Eclipse technologies

- Sirius https://eclipse.org/sirius
- Eclipse Modeling Tools: http://www.eclipse.org/downloads/
- Eclipse Modeling Framework (EMF)
  - http://www.eclipse.org/emf/
  - EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model.
- Eclipse Graphical Editing Framework (GEF)
  - http://www.eclipse.org/gef/
  - The Graphical Editing Framework (GEF) allows developers to take an existing application model and quickly create a rich graphical editor.
- Eclipse Graphical Modeling Framework (GMF)
  - http://www.eclipse.org/gmf/
  - The Eclipse Graphical Modeling Framework (GMF) provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF.
- Xtext
  - https://eclipse.org/Xtext/
  - Preferred tool for textual domain specific languages (used for ThingML for example)
- Atlas Transformation Language
  - http://www.eclipse.org/gmt/atl/
  - The ATL project aims at providing a set of transformation tools for GMT. These include some sample ATL transformations, an ATL transformation engine, and an IDE for ATL (ADT: ATL Development Tools).
- Eclipse Process Framework (EPF)
  - http://www.eclipse.org/epf/
  - To provide an extensible framework and exemplary tools for software process engineering - method and process authoring, library management, configuring and publishing a process.

EMF – Eclipse Modeling Framework

- Unifying Java, XML and (almost) UML
- EMF models are essentially simplified UML Class Diagrams
- EMF generates Java code based on these models
- Standard serialization is in the form of XMI
- “EMF is MDA on training wheels”

EMF Models and Ecore

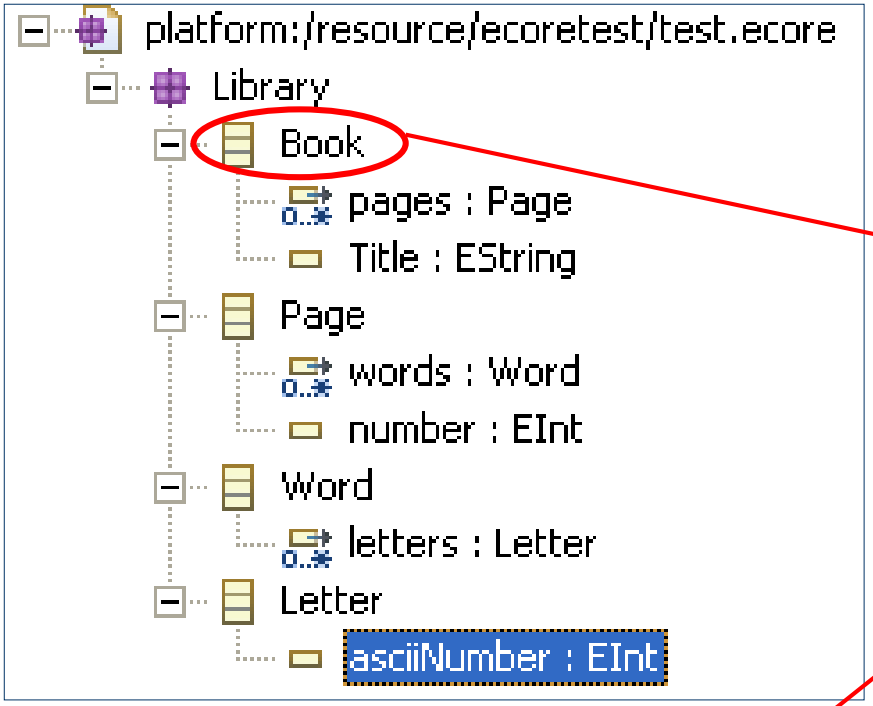
- Ecore is the model used to represent EMF models
- Ecore is also an EMF model and therefore its own metamodel
  - And its own meta-meta-….-model, but never mind
- Available elements are:
  - EClass
  - EAttributes
  - EReference
  - EDataType
  - EEnum, EEnum Literal
  - EPackage
  - EOperation, EParameter
- Conceptually equal to OMGs Essential MOF (EMOF)

Creating your model

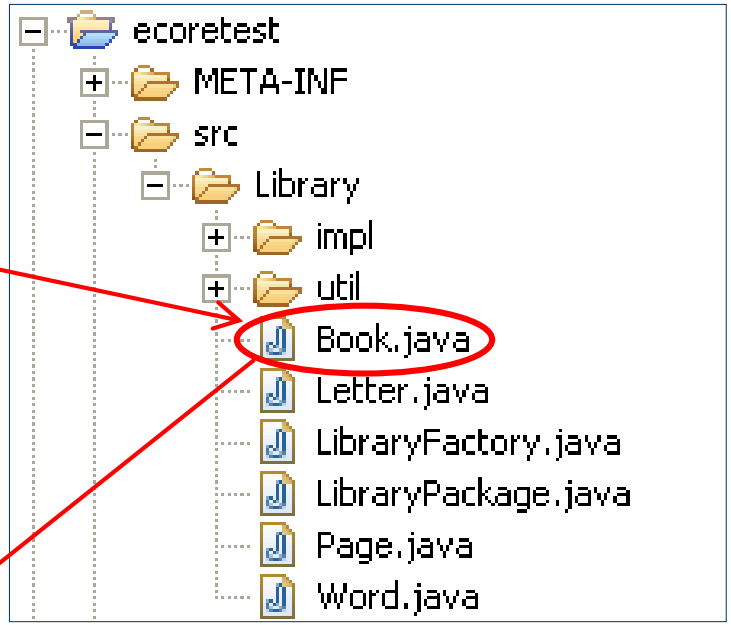
- Can be defined in three ways
  - Java
  - XML Schema
  - Directly manipulate the model (the almost UML way)
- Both Java and XML Schema approach builds an EMF model
- Editing the model can be done with the EMF tree editor or the GMF graphical editor
  - It is also possible to import Rational Rose (.mdl) files

3 shades of EMF

Ecore model



Generated Java files



Creation of an instance

```
Book book = LibraryFactory.eINSTANCE.createBook();
book.setTitle("How to be a meta role model");
```

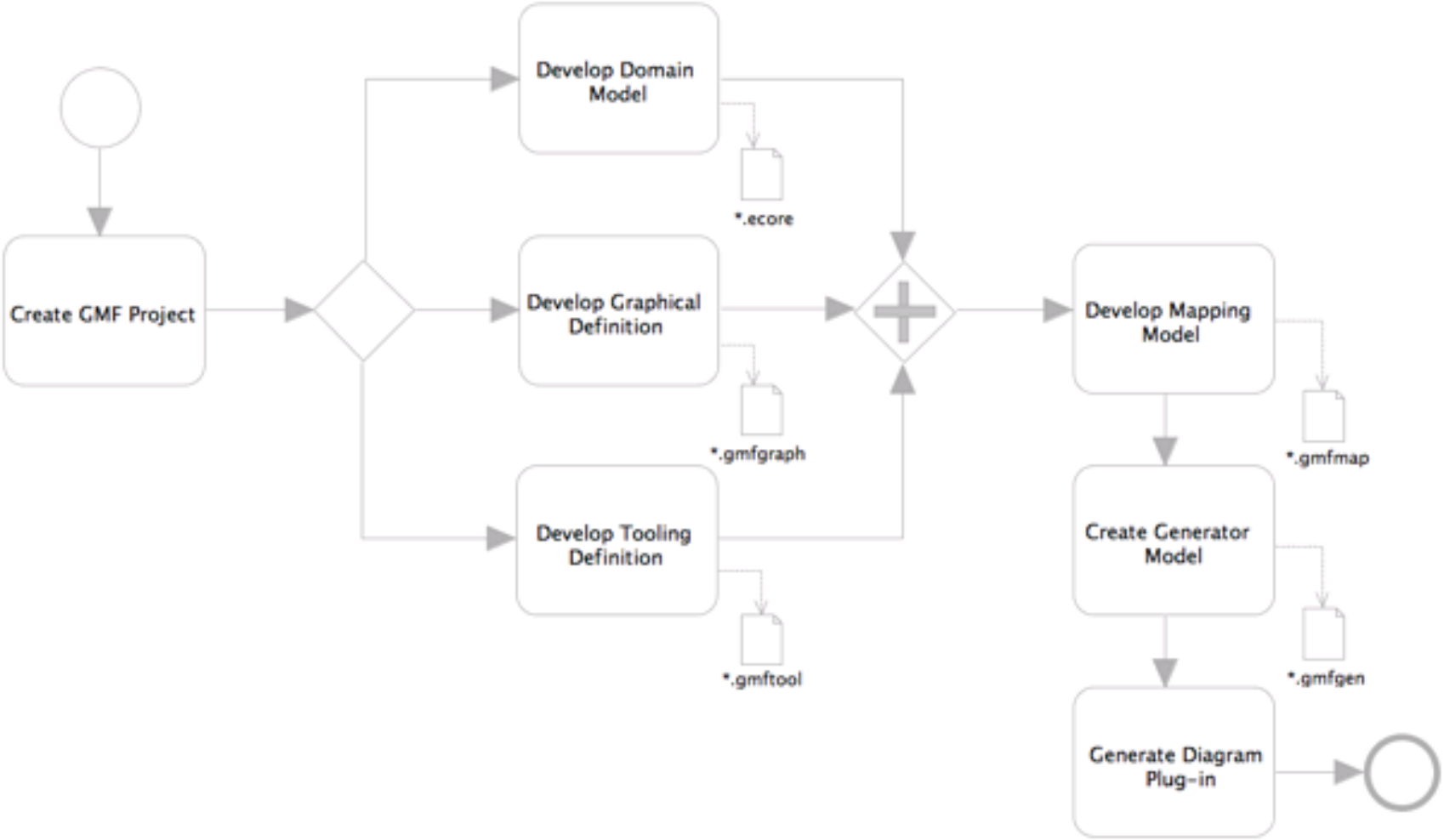
GMF

- The Eclipse Graphical Modeling Framework (**GMF**) project is an open source project under the Eclipse Technology Project
- Infrastructure and components for developing visual design and modeling surfaces in Eclipse
  - UML editors
  - Business process editors
  - Etc..
- **GMF** forms a generative bridge between **EMF** and **GEF**
- A diagram definition (**GEF**) will be linked to a domain model (**EMF**) as input to the generation of a visual editor

GMF – Graphical Modeling Framework

- Utilizes EMF and GEF to support generation of graphical editors
  - GEF – Graphical Editing Framework
- Basic idea:
  - Bring your own model
  - Define diagram notation
  - Define your tools
  - Map model elements to diagram elements
  - Generate editor
- Metamodel = Abstract Syntax
- Diagram notation = Concrete Syntax

Simplified workflow



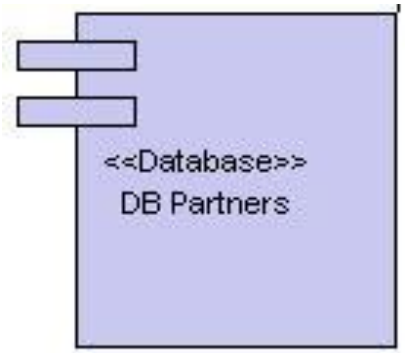
UML profiles

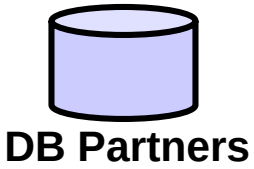
UML profiles

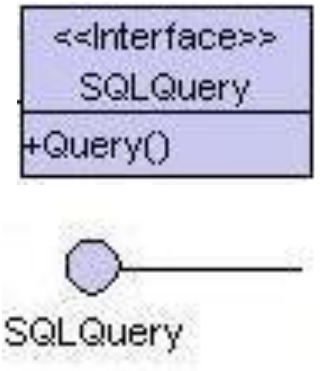
- They allow us to adapt the UML language to the needs of the analysts or the application domain
- Allows designers to model using application domain concepts.
- There are three extension mechanisms:
  - Stereotypes
  - Restrictions
  - Tagged values

Stereotype

- **Extends** the vocabulary of **UML** with **new construction elements** derived from **existing** UML but specific to a problem domain
- Can have associated **restrictions and tagged values**
- Possibility of assigning an **icon** for a better graphical representation



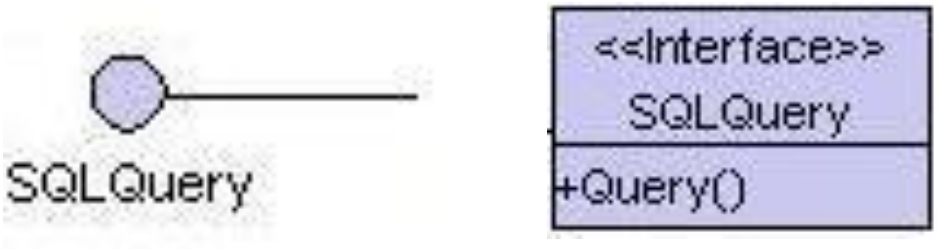




Restriction

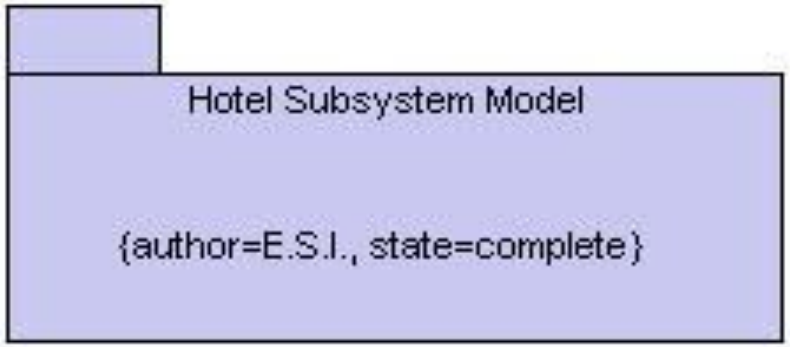
- Is a semantical **condition** represented by a **textual expression**
- Imposes some kind of condition or **requisite on the element** to which it is applied
- OCL – Object Constraint Language

{An interface does not have attributes, only operations}

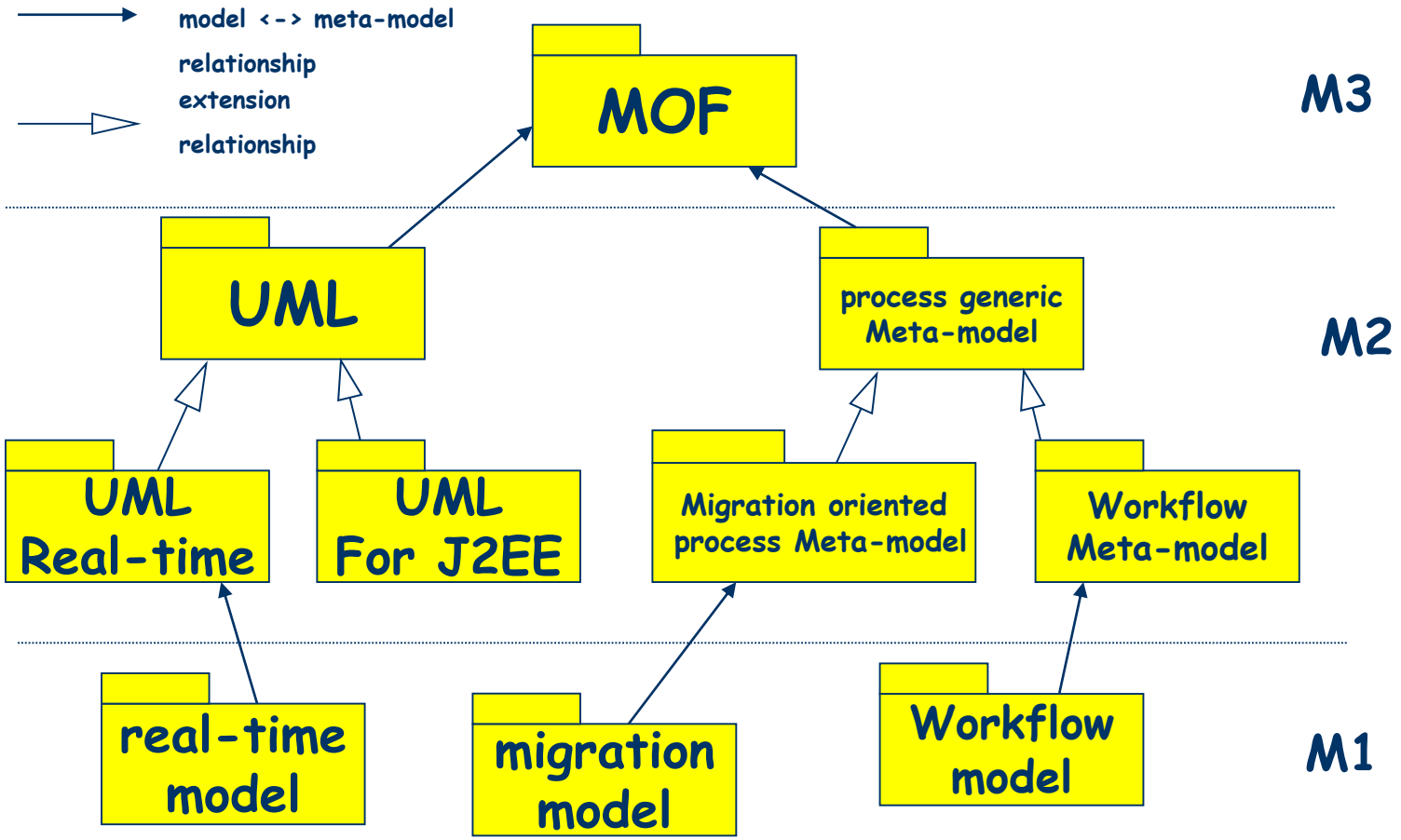


Tagged value

- Is a **property** associated to a **model element**
- Used to store **information** about the element
  - Management information, documentation, coding parameters, ...
- Generally, the **tools** store this information but **it is not shown in the diagrams**



Metamodels and profiles



Domain-specific languages (DSLs)

UML – one size fits all?

- While the OMG MDA promotes UML as the visual “universal” glue suitable for modelling everything, we are also seeing a trend towards development and co-existence of several domain-specific modelling languages, e.g. supported by the Microsoft Domain-Specific Language (DSL) tools (http://lab.msdn.microsoft.com/teamsystem/workshop/dsltools/default.aspx).
- Such approaches are now also being discussed in various OMG forums.
- UML is seen as a “general-purpose” language while DSLs may be more expressive for most purposes.
- A model-driven framework needs to acknowledge the existence of different models and views expressed in different modelling languages.
- The MDA technologies can help us to align these models through a common metamodelling language on which model transformations and model mappings can be defined.

Software factory

- The Software Factories Web site (http://www.softwarefactories.com/) defines the term Software Factory in the following way:
- *"A Software Factory is a software product line that configures extensible development tools like Visual Studio Team System with packaged content like DSLs, patterns, frameworks and guidance, based on recipes for building specific kinds of applications. For example, we might set up a Software Factory for thin client Customer Relationship Management (CRM) applications using the .NET framework, C#, the Microsoft Business Framework, Microsoft SQL Server, and the Microsoft Host Integration Server. Equipped with this factory, we could rapidly punch out an endless variety of CRM applications, each containing unique features based on the unique requirements of specific customers. Better yet, we could use this factory to create an ecosystem, by making it available to third parties, who could extend it to rapidly build CRM applications incorporating their value added extensions."*

UML and DSLs

- The issue of the role of UML is often stated in overly simplistic terms: MDD advocates the use of UML for all domain modelling while the Software Factories approach advocates that UML never used.
- This is an incorrect statement of the positions of both camps.
  - While the MDD approach treats UML, with customization, as the modelling language of choice for most application modelling, it also acknowledges the value of custom languages in certain specialized circumstances.
  - This is the purpose of the OMG Meta-Object Facility (MOF) standard that plays an important role in MDD. UML itself is defined using MOF and there are MOF definitions of many other languages.
  - The MDD approach acknowledges the value of non-UML DSLs as a technique to be applied judiciously.
  - Further, the Software Factories approach does not reject UML entirely. It suggests that you use UML for developing sketches and documentation, where DSLs should be used for developing models from which code is generated.

Advantages of using UML profiles

- UML is an open standard modelling language for which there are many available books and training courses.
- UML profiles provide a lightweight approach that is easily implemented using readily available UML tooling.
- Models with UML profiles applied can be read by all UML tools even if they do not have any knowledge of the profile.
- Basing all DSLs on UML creates a set of related languages that share common concepts.
- UML can be used for high-level architectural models as well as detailed models from which code can be generated.

Disadvantages of using UML profiles

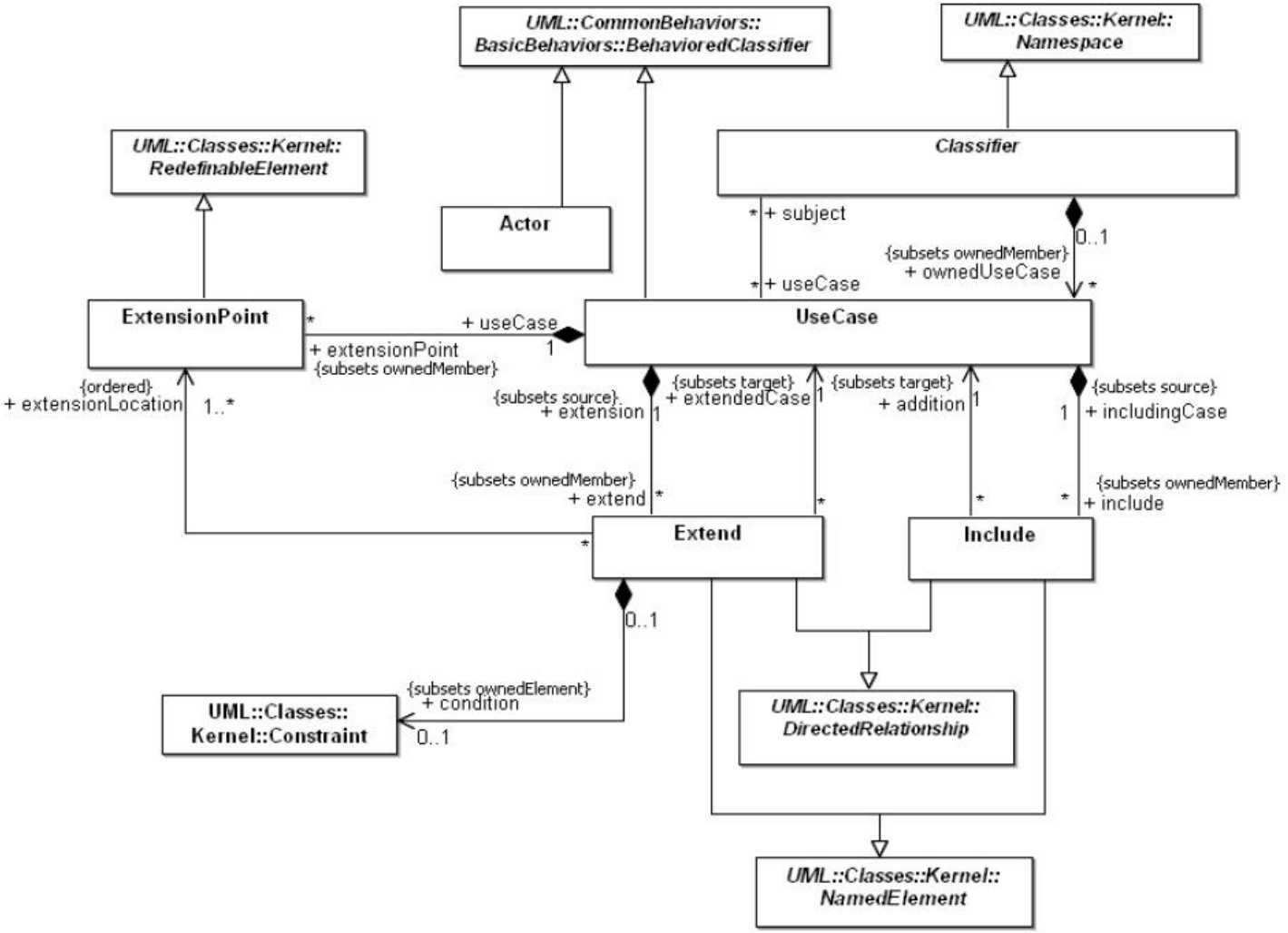
- UML profiles only permit a limited amount of customization.
  - It is not possible to introduce new modelling concepts that cannot be expressed by extending existing UML elements.
- The use of UML does require familiarity with modelling concepts.

Example Metamodels

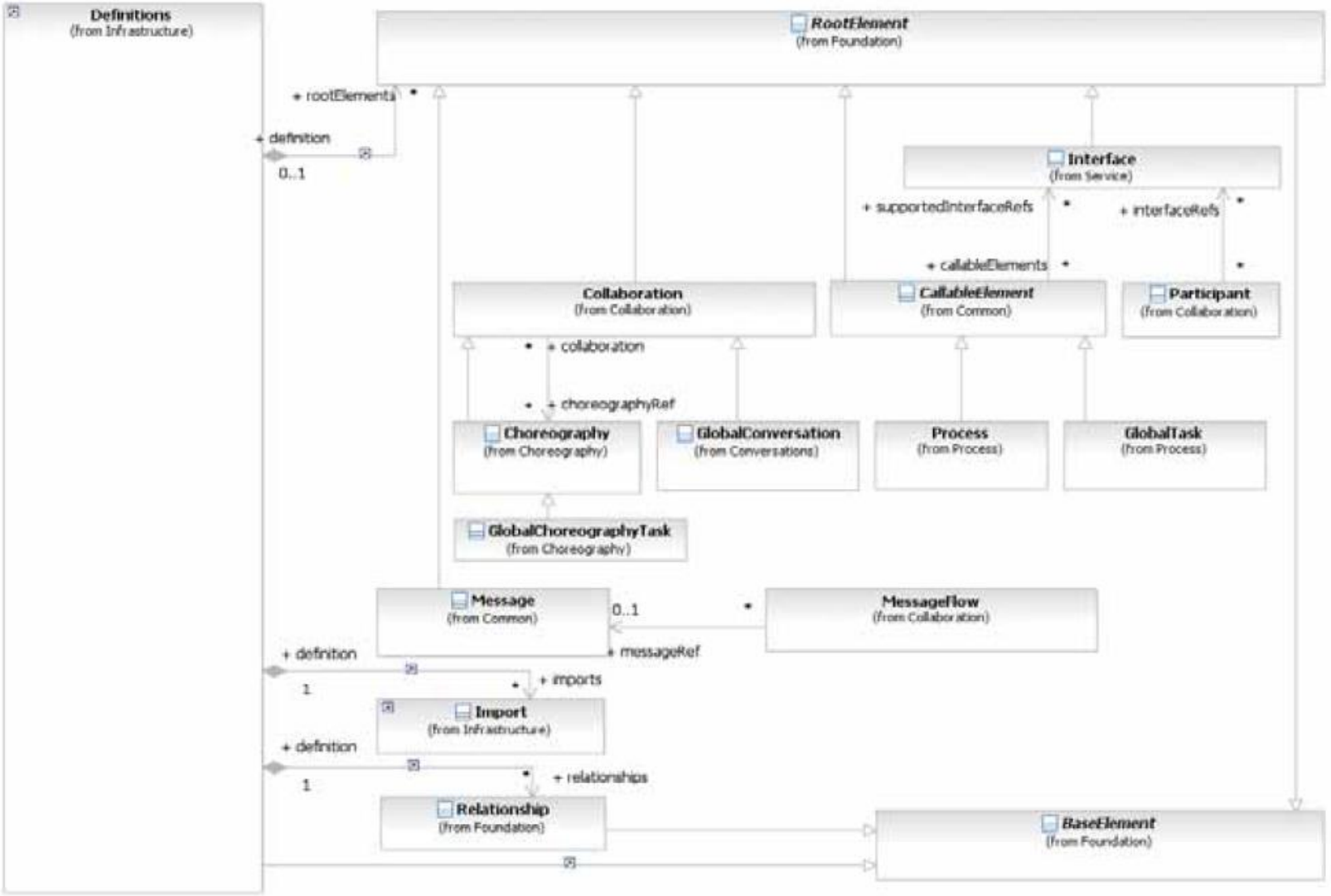
- UML Use case Metamodel
- BPMN Metamodel
- IFML Metamodel

- Oblig 3 – Visual Service Journey language

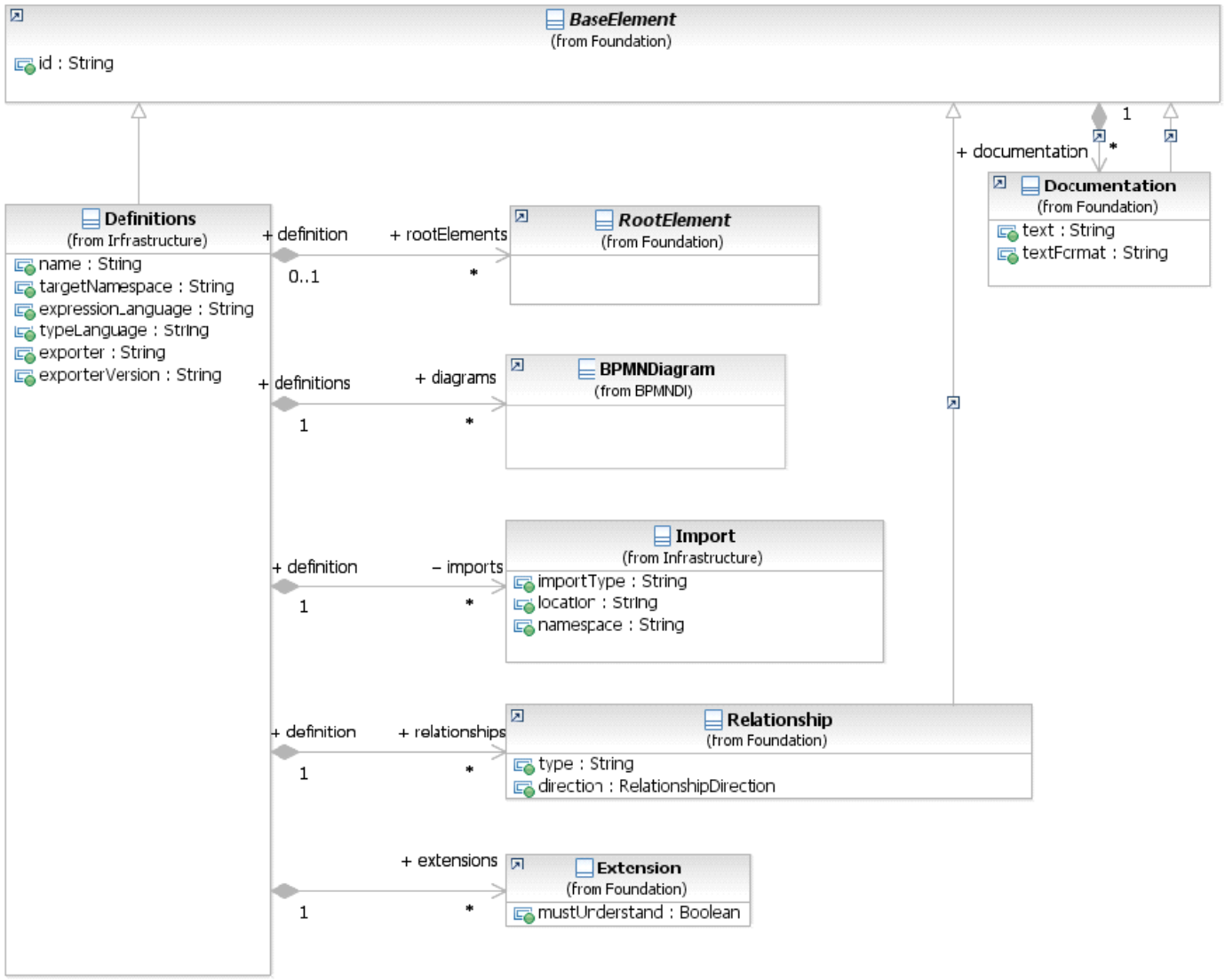
UML Use Case Metamodel



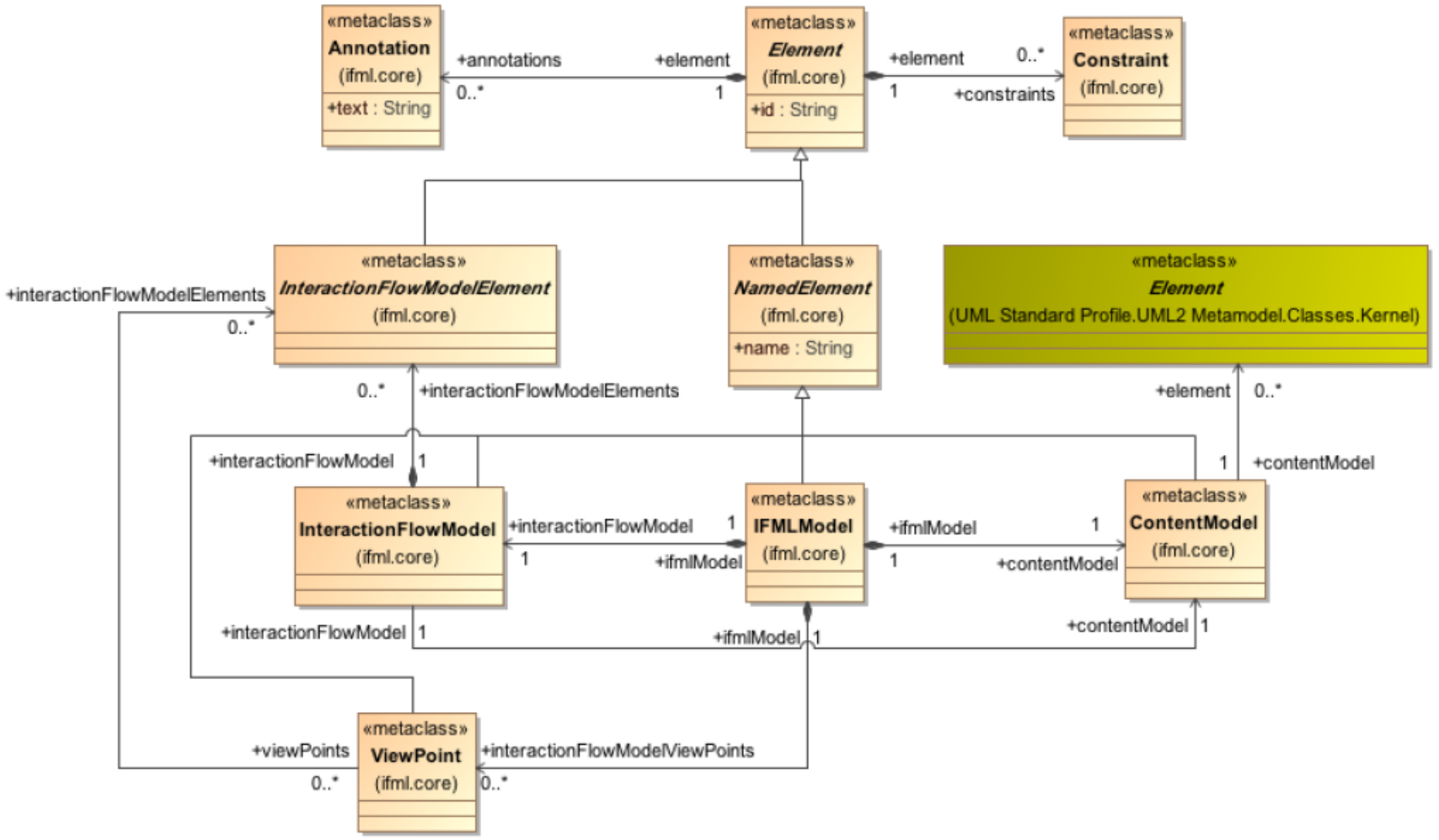
BPMN Metamodel



BPMN Definitions

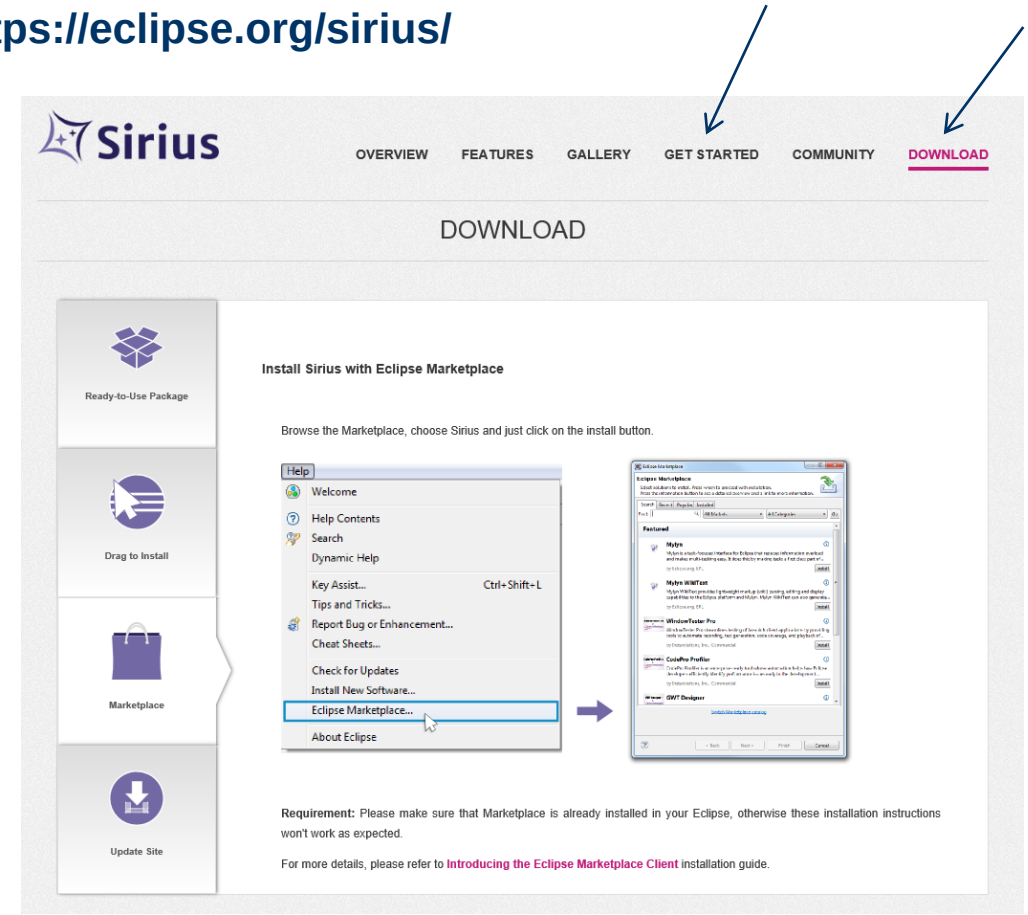


IFML Metamodel

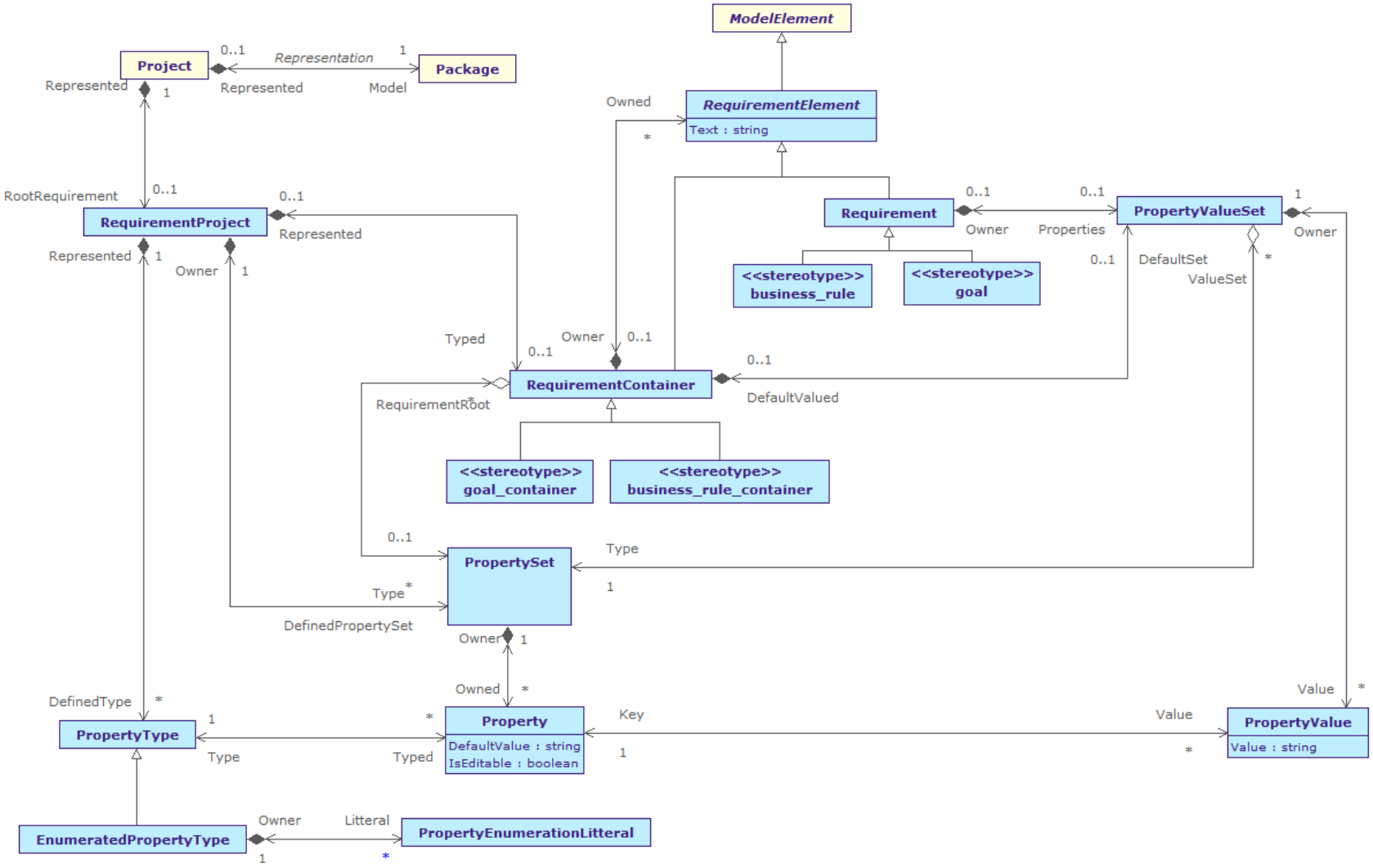


Oblig 3 – Use of Eclipse EMF and SIRIUS for the creation of an Archimate graphical editor

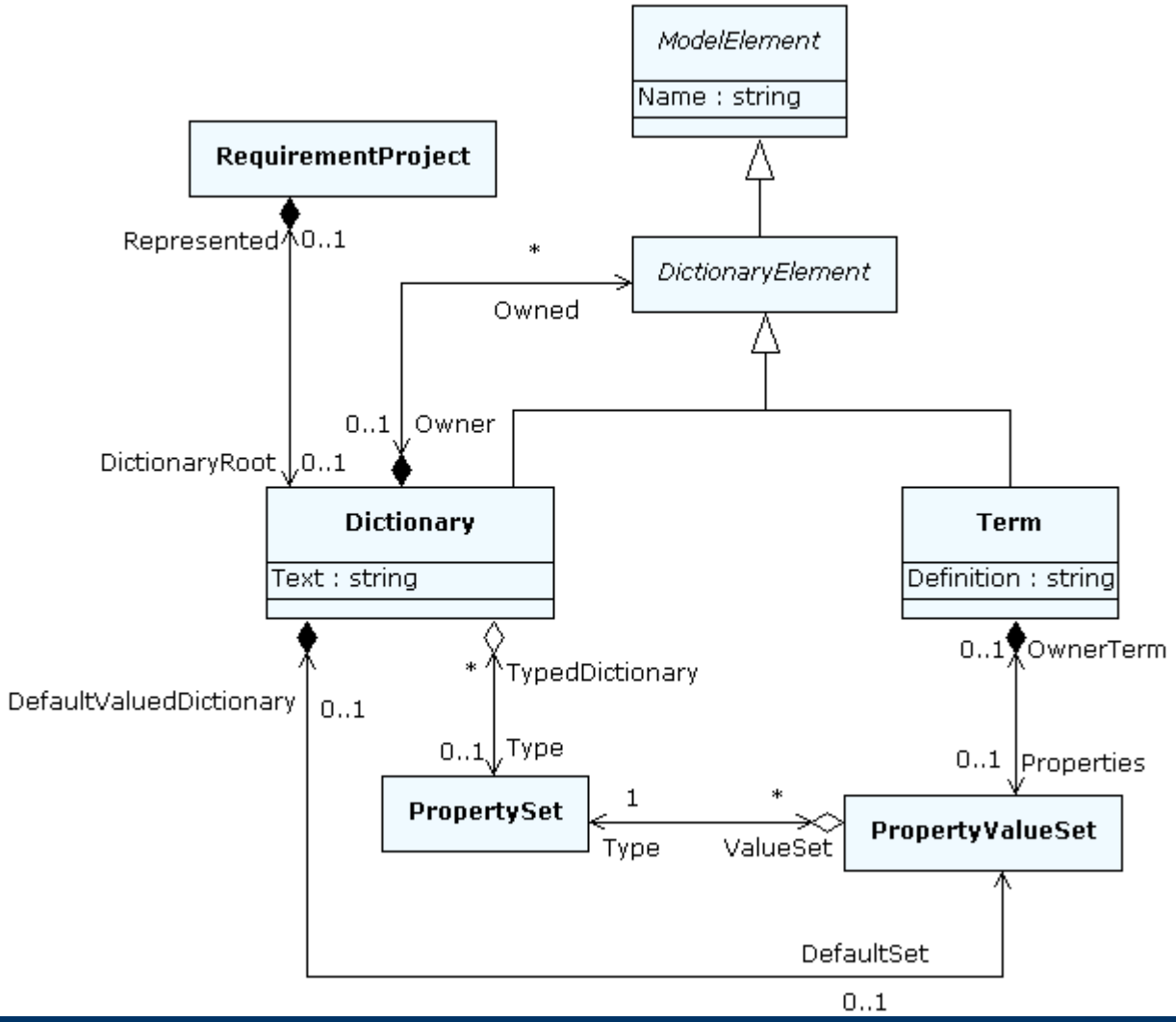
https://eclipse.org/sirius/



Scope Manager metamodel



Dictionary metamodel



Some historic references

[Atkinson and Kühne 2003] C. Atkinson and T. Kühne, "Model-Driven Development: A Metamodeling Foundation", IEEE Software, vol. 20, no. 5, pp. 36-41, 2003. http://www.mm.informatik.tu-darmstadt.de/staff/kuehne/publications/papers/mda-foundation.pdf

[Clark, et al. 2004] T. Clark, A. Evans, P. Sammut, and J. Willans, "Applied Metamodelling - A Foundation for Language Driven Development, Version 0.1", 2004. http://albini.xactium.com/web/index.php?option=com_remository&Itemid=54&func=select&id=1

[Seidewitz 2003] E. Seidewitz, "What Models Mean", IEEE Software, vol. 20, no. 5, pp. 26-32, 2003.

[Swithinbank, et al. 2005] P. Swithinbank, M. Chessell, T. Gardner, C. Griffin, J. Man, H. Wylie, and L. Yusuf, "Patterns: Model-Driven Development Using IBM Rational Software Architect", IBM, Redbooks, December 2005. http://www.redbooks.ibm.com/redbooks/pdfs/sg247105.pdf

Next lecture - Monday April 24th, 2017

- Concluding on Model Driven Engineering
- Presentation of Oblig 2
- Further discussions for Oblig 3 – with Eclipse, EMF and Sirius – for May 4th