Languages and Software Language Engineering

Lecture 2: Languages

by Prof. Vasco Amaral

2022/2023

In the previous lecture...

We discussed models (and metamodels)

Now we need to have a clear notion of what a valid model is and what does it means

What is a modelling language?

A modelling language defines a set of models that can be used for modelling purposes. It's definition consists of:

- **Syntax, how they are perceived (how the models look like)**
- **Semantics, what each of its models means**
- **Pragmatics, how the models are used according to their purpose**

Concrete Syntax

Describes the concrete representation of the models and is used by humans to uses senses (eyes, ear,...) to read, understand, and create models. The concrete syntax must be sufficiently formal to be procesible by tools.

Abstract Syntax

Contains the essential information of a model, disregarding all details of the concrete syntax that do not contribute to the model's purpose. It is particularly interesting for use by software tools (e.g. interoperability).

Concrete Syntax (CS)

We can have distinct CS for identical languages for different purposes

Visualized (more common):

- Graphical (or diagrammatic)
- Textual
- Variations
  - Graphs,trees,tabular, ASCII or XML textual forms

We can consider other senses like ear (sound) and touch (gestures) although uncommon

The choice of adequate CS for the target users is essential for its adoption (therefore we will study usability issues later on)

Textual Concrete Syntax

- Well established theories and tools, including grammars and parser generators (e.g. ANTLR)
- Relies on underlying alphabet like ASCII or Unicode as basis
- Groups the characters available in two phases:
  - Lexical terms like keywords (e.g. begin, end), operators, numbers or names
  - Full sentences using a grammar
- The editing tool can add (as part of the concrete syntax) highlighting, fonts, tab (etc) to ease reading
- These languages are usually agnostic to white spaces (real spaces, tabulators and line breaks)

Textual Concrete Syntax

Usually described using a EBNF grammar

```
(* a simple program syntax in EBNF − Wikipedia *)
program = 'PROGRAM', white_space, identifier, white_space,
           'BEGIN', white_space,
           { assignment, ";", white_space },
           'END.' ;
identifier = alphabetic_character, { alphabetic_character | digit } ;
number = [ "-" ], digit, { digit } ;
string = '"' , { all_characters - '"' }, '"' ;
assignment = identifier , ":=" , ( number | identifier | string ) ;
alphabetic_character = "A" | "B" | "C" | "D" | "E" | "F" | "G"
                     | "H" | "I" | "J" | "K" | "L" | "M" | "N"
                     | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
                     | "V" | "W" | "X" | "Y" | "Z" ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
white_space = ? white_space characters ? ;
all_characters = ? all visible characters ? ;
```

Example

```
PROGRAM DEMO1
BEGIN
  A:=3;
  B:=45;
  H:=-100023;
  C:=A;
  D123:=B34A;
  BABOON:=GIRAFFE;
  TEXT:="Hello world!";
END.
```

Textual Concrete Syntax

With XML
Grammar is defined using XML Schema Definition (or XSD)

Not considered Human readable for elaborated data structures (significant effort to interpret essential information in between tags…

Usually needs a tree browser to enable the user to view and manipulate

Graphical Concrete Syntax (Box and lines diagrams)

- Typically use both dimensions on paper or screen
- Usually augmented with text fragments to enhance essential information
- Use as elements:
  - Boxes (several possible shapes, including use of icons)
  - Lines (several shapes, styles, colors)
  - Compartments

UML sequence diagrams use special form of lines (lifelines, that have a beginning and no end)

Typically we abstract from the layout and size (as spaces in the textual languages)

Graphical Concrete Syntax (Tabular)

It is a special form of graphical notation for regularly Structured situations where adjacency helps to compact the model

It is often useful to number rows and columns and possibly provide a visualiser with scrolling and zooming functionalities

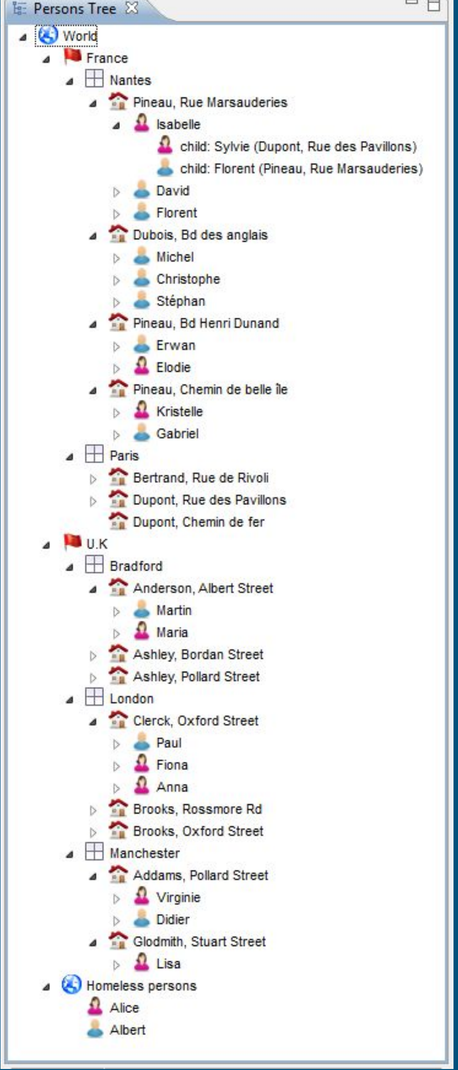
Persons Table

	Nb persons	Father	Mother	Nb children
France	14			
Nantes	10			
Pineau, Rue Marsauderies	3			
David				2
Florent		David	Isabelle	1
Isabelle				2
Dubois, Bd des anglais	3			
Michel		Jean	Dana	1
Christophe		Michel		1
Stéphan		Christophe		0
Pineau, Bd Henri Dunand	2			
Erwan		William		1
Elodie		Erwan		0
Pineau, Chemin de belle île	2			
Gabriel		Florent		1
Kristelle		Gabriel		0
Paris	4			
Bertrand, Rue de Rivoli	3			
Jean				2
Dana				2
Marie		Jean	Dana	1
Dupont, Rue des Pavillons	1			
Sylvie		David	Isabelle	1
Dupont, Chemin de fer	0			
U.K	20			
Bradford	9			
Anderson, Albert Street	2			
Martin				2
Maria				2
Ashley, Bordan Street	3			
Adam		William		1
William		Martin	Maria	2
Nicolas		Adam		0
Ashley, Pollard Street	4			
Edward		Gaby		1
Gaby		Martin	Maria	1
Elisa		Edward		0
Amanda			Anna	0
London	8			
Clerck, Oxford Street	3			
Paul				2
Fiona		Paul	Lisa	2
Anna		Paul	Lisa	1
Brooks, Rossmore Rd	3			
Dave			Fiona	1

Graphical Concrete Syntax: Trees

It is often relatively easy to define a spanning trees over graph structures

XML is usually manipulated this way



Abstract Syntax

Of textual languages

Textual languages typically are defined using grammars and use trees as their internal representation

Sometimes optimizations, rearrangements or extensions are made to allow efficient storage and retrieval of information from the AST (e.g. resolution of names)

But can also use metamodels to describe the possible set of models.

(we will see examples with Xtext)

Abstract Syntax

Of Graphical languages

Usually defined using a metamodeling approach

Metamodels (Class Diagram like) are used to describe the graph structures behind the boxes and lines that are visible in the concrete syntax of the model

As observed in UML, class diagrams allow for structures that might be illegal structures, wrong combinations of attributes, etc.

Well formedness rules(constraints) in OCL like languages are used

Relating Concrete Syntax and Abstract Syntax

In the case of Textual languages the grammar describes both

In the case of graphical languages, first we have the metamodel for the abstract syntax and then we can map the elements to several distinct concrete syntaxes (which is not possible with the textual languages)

Semantics of a Modelling Language

Being sure about the consistent meaning (to avoid misinterpretation)

Captures the essential information of its models in the form of explicitly defined

- Syntactic domain
  - that describes the well-formed models
- Semantic domain
  - that captures all essential information that the model can describe
- Semantic mapping
  - that relates the syntactic constructs of the models to the semantic domain

Semantics of a Modelling Language

Defined using

- Denotational Semantics
  - describes what a model means, typically with mathematical constructs without talking about how the meaning is achieved
- Operational Semantics
  - maps the input model to some executable code. Having this we can, for instance, run a simulation
- Axiomatic Semantics
  - defines the meaning of the language constructs in terms of assertions. Thus there may be aspects of the executions that are ignored.

Denotational Semantics

Defined using

- Set of models - L
  - Let L be the set of models in their syntactic shape
- Semantic Domain - S
  - Let S be the Semantic Domain that precisely defines the set of mathematical entities representing what we want to describe
- Semantic Mapping - M
  - M is a mathematical function that relates one model of our modelling language with its meaning:
  - M: L -> S

Software Language Engineer

One task of a language engineer is to develop languages that make the job of creating software easier.

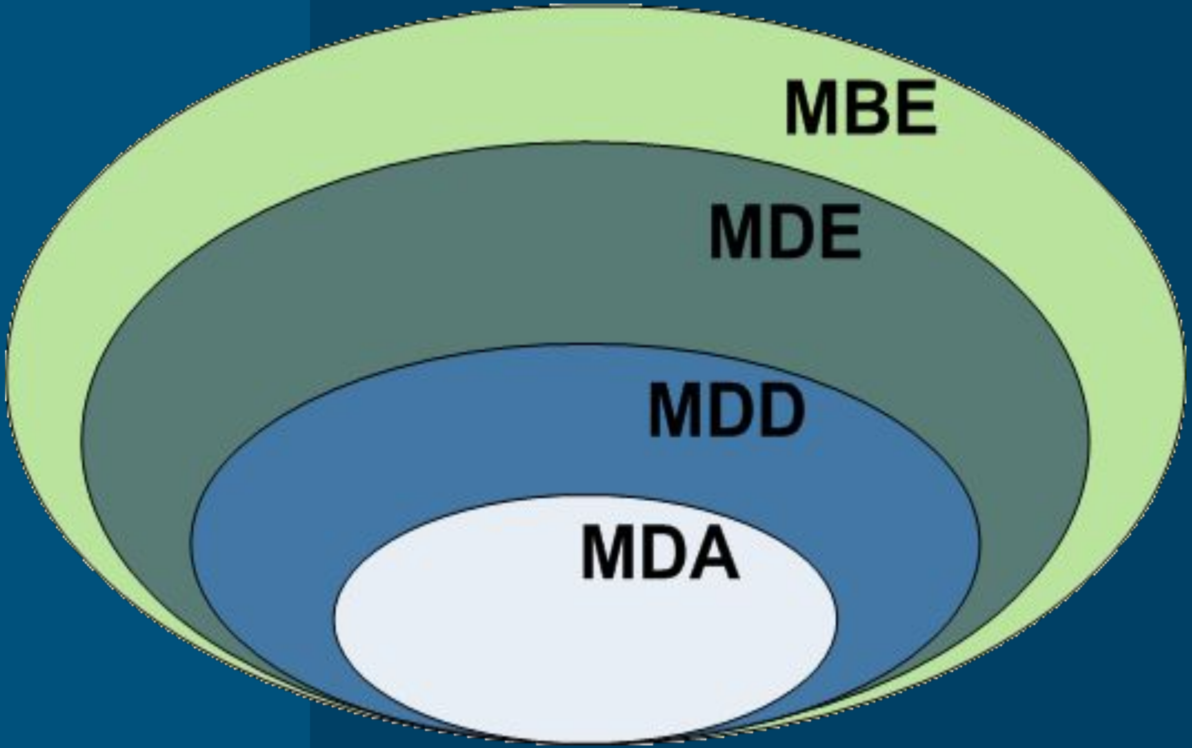
Another task is to create a language that will support the language end-user (also known as domain expert) efficiently and effectively.

Software Language Engineering

SLE is the application of a systematic, disciplined and quantifiable approach to the development, usage, and maintenance of software languages.

The Model-Driven way

MD*



Taken from Master thesis of David Ameller (supervised by Xavier Franch)

MD*

- MBE – Model-Based Engineering
  - Process in which software models play an important role although they are not necessarily the key artifacts of the development (i.e. they do NOT “drive” the process)
- MDE – Model-Driven Engineering
  - Goes beyond of the pure development activities and encompasses other model-based tasks of a complete software engineering process (e.g. the model-based evolution the system or the model-driven reverse engineering of a legacy system).
- MDD – Model-Driven Development
  - Development paradigm that uses models and transformation (which also have models) as the primary artifact of the development process. Usually, in MDD, the implementation is (semi)automatically generated from the models.
- MDA – Model-Driven Architecture
  - OMG’s particular vision of MDD and thus relies on the use of OMG standards.

Taken from Master thesis of David Ameller (supervised by Xavier Franch)

Software

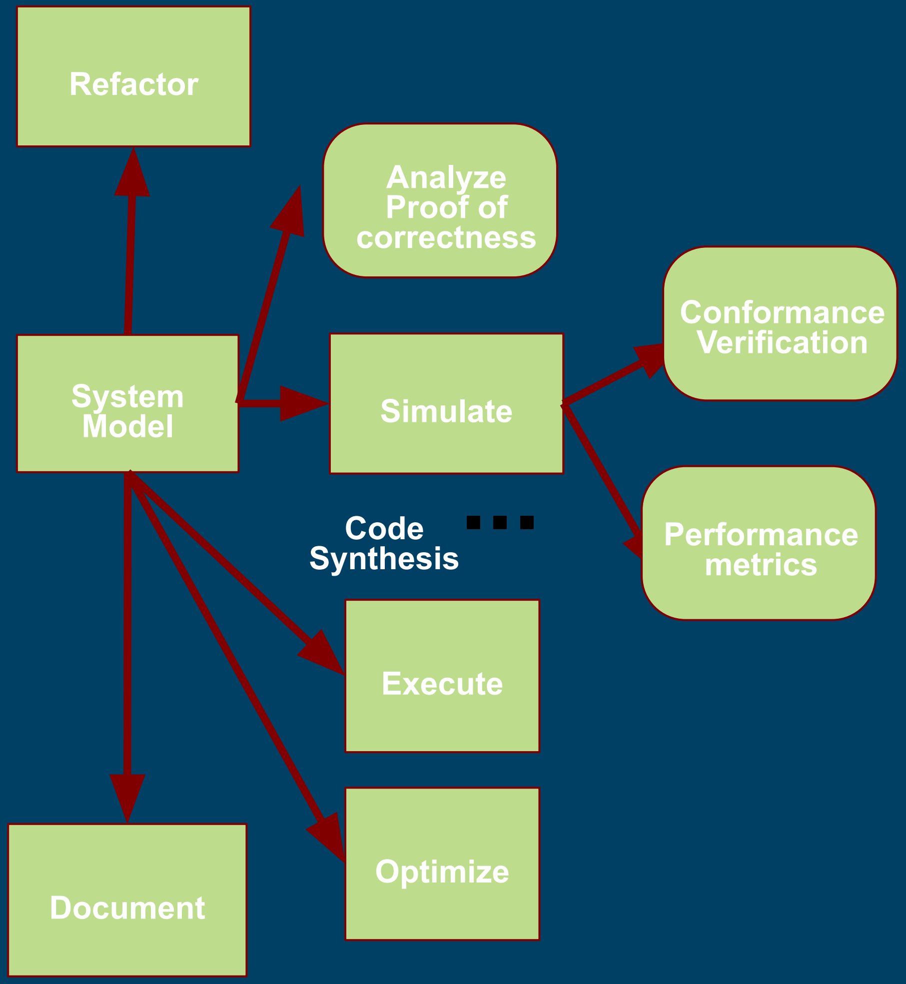
In Programming (Wirth):

Algorithms + Data Structures = Programs

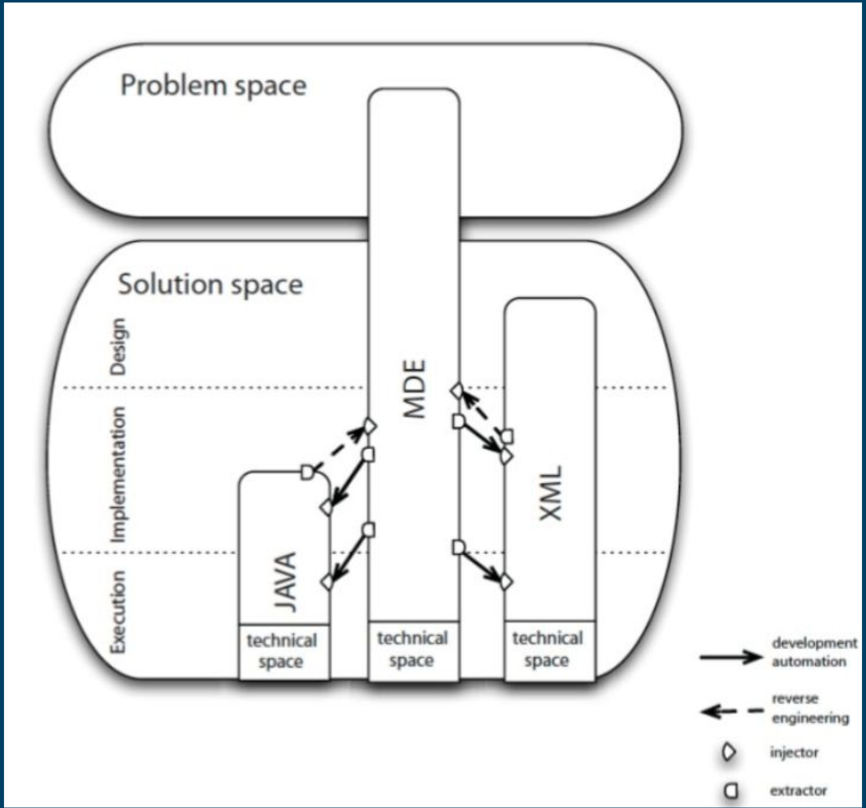
In MDE:

Models + Transformations = Software

Model-Driven Roadmap The model as a central artefact



MDE Coverage



MDE Coverage

Problem Domain - field or area of expertise where to solve a problem

Domain Model - Conceptual problem of the problem domain

Technical spaces- specific working contexts for specification, implementation and deployment of applications

General purpose

General Purpose

Specific Purpose

- **match the user's mental model** of the problem domain
- **maximally constrain** the user (to the problem at hand)
  $\Rightarrow$ easier to learn
  $\Rightarrow$ avoid errors
- **separate** domain-expert's work
  from analysis/transformation expert's work

Anecdotal evidence of 5 to 10 times speedup

Steven Kelly and Juha-Pekka Tolvanen. Domain-Specific Modeling: Enabling Full Code Generation. Wiley, 2008.

Laurent Safa. The practice of deploying DSM, report from a Japanese appliance maker trenches. In Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06), pp. 185-196, 2006.

DSLs and GPLs

(Examples)

DS(M)Ls

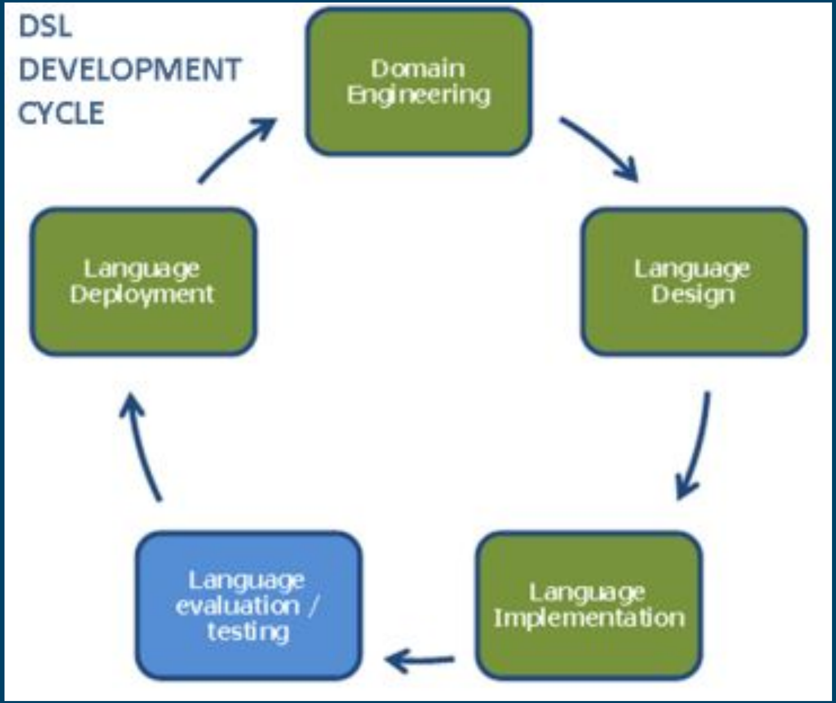
E.g. HTML, Logo, VHDL, Mathematica, SQL

GP(M)Ls

E.g. UML, Petri-nets, Statecharts

SLE Process

(as systematic)

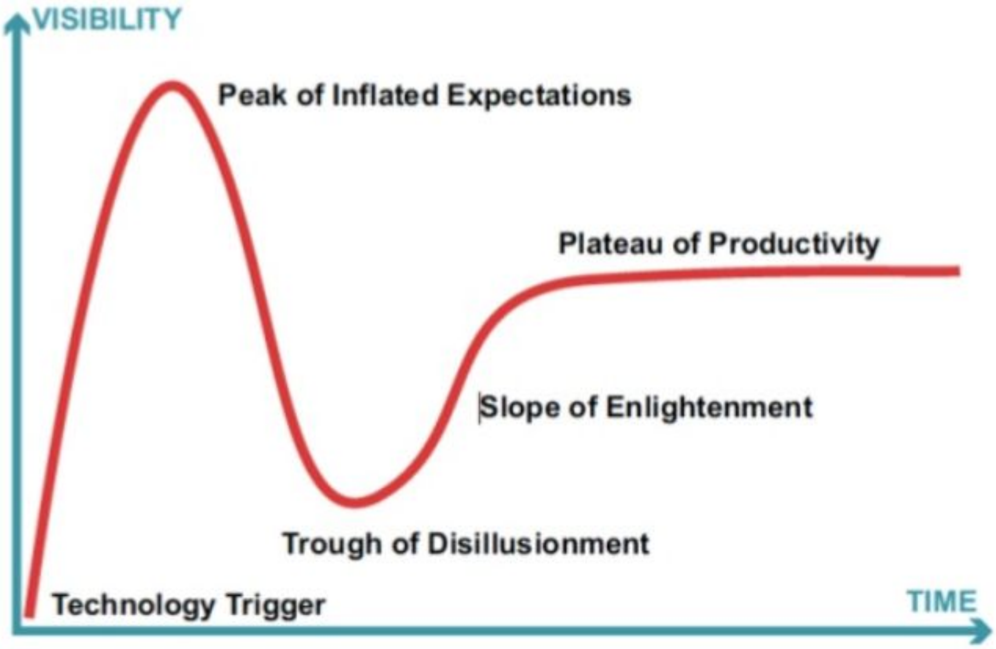


Ankica Barisic, Vasco Amaral, Miguel Goulão:
Usability driven DSL development with USE-ME. Computer Languages, Systems & Structures 51: 118-157 (2018)

MDE

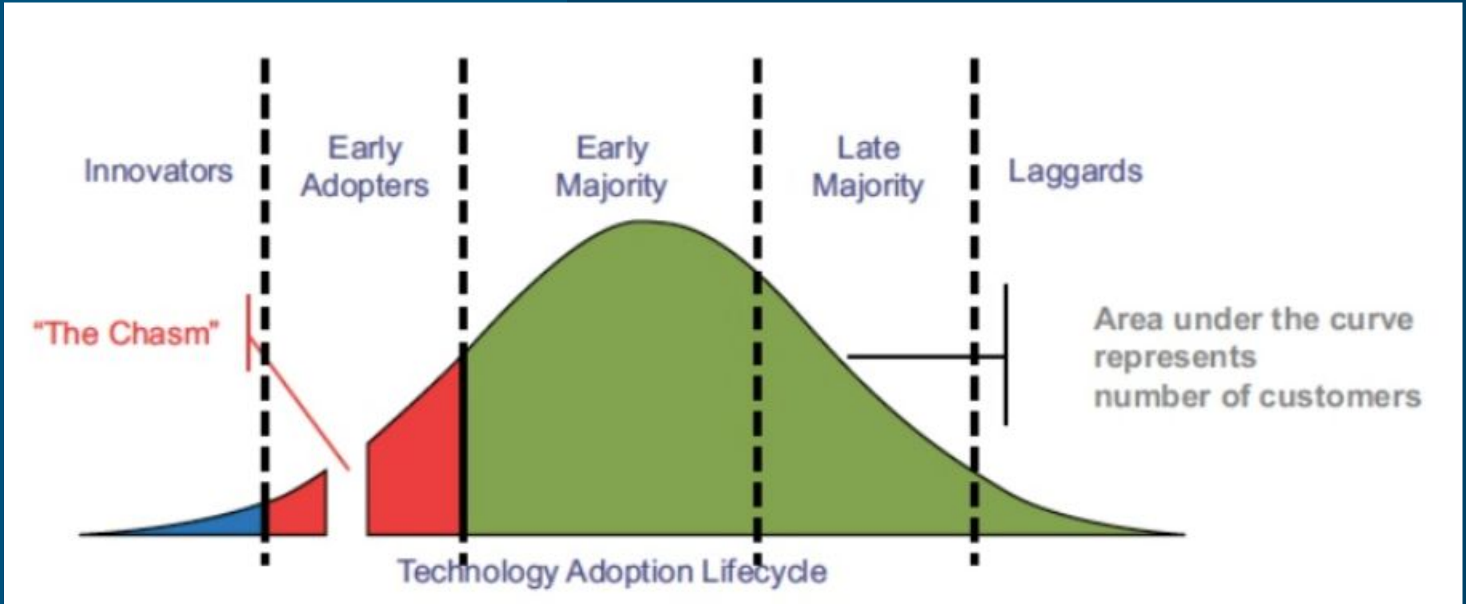
Adoption

- Not yet mainstream in all industries
- Strong in core industry (defense, avionics, …)



MDE

Adoption



Languages and Software Language Engineering

Lecture 3: Relevant definitions and Metamodelling with Eclipse
by Prof. Vasco Amaral
2019/2020

Modelling Languages

DS(M)Ls

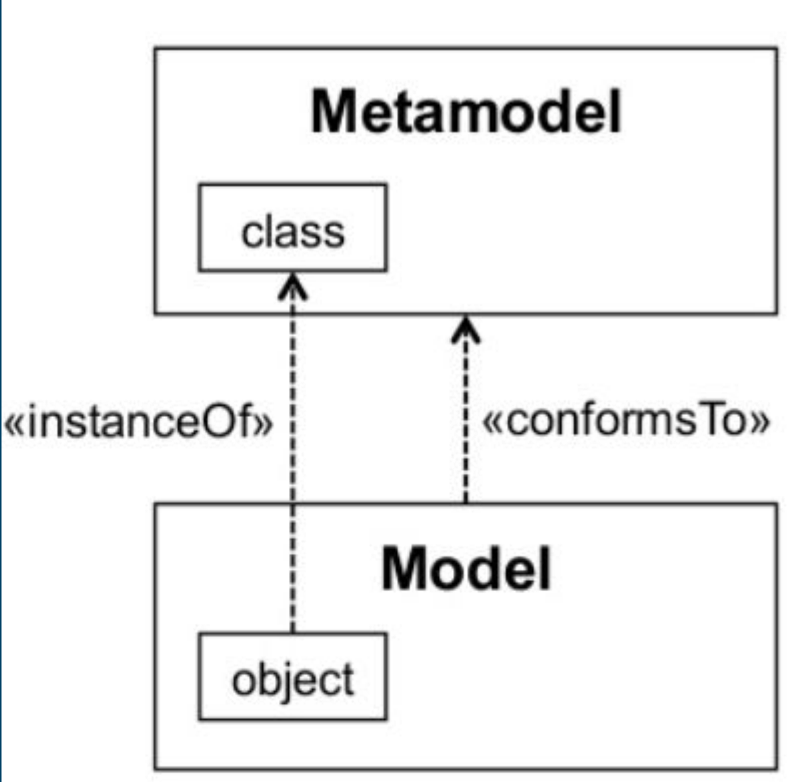
E.g. HTML, Logo, VHDL, Mathematica, SQL

GP(M)Ls

E.g. UML, Petri-nets, Statecharts

InstanceOf vs. ConformsTo

Conformance is between models
Instantiation is between model elements



Model Conformance

A model is valid in a given language if...

A model conforms to a given metamodel if each model element is an instance of a metamodel element. Then a model is valid with respect to the language represented by the metamodel.

Meta-language

A model is valid in a given language if...

Is a language dedicated to language modelling, i.e., for defining metamodels

MDE's meta-languages

MDE approaches leverages the object-oriented paradigm and most of the meta-languages are derivatives of UML's class diagram (we can also find ER like diagrams), often extended with related languages such as Object Constraint Language (OCL)

Modelling workbenches

A language workbench provides a set of tools and meta-languages supporting the development and evolution of a language and its associated tooling, including design, implementation, deployment, evolution, reuse, and maintenance.

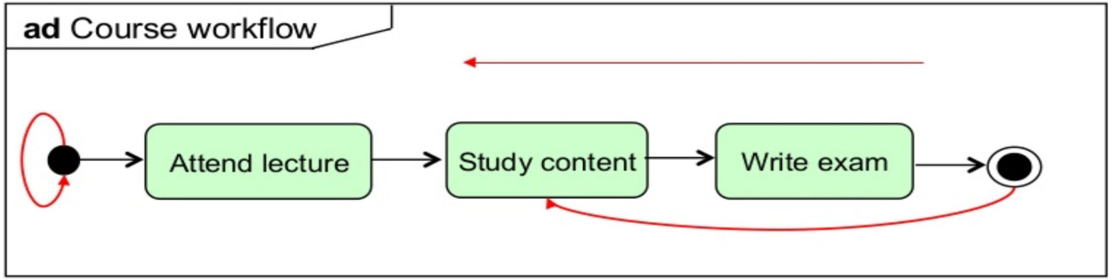
The term was coined in 2005 by Martin Fowler. Examples of workbenches are: JetBrains MPS, Metacase's MetaEdit, EMF, AtomPM, Microsoft Visualization and Modelling SDK

Meta Circularity

Use of a metamodel to model its own shape.All concepts available in a language can be modelled using the language itself.

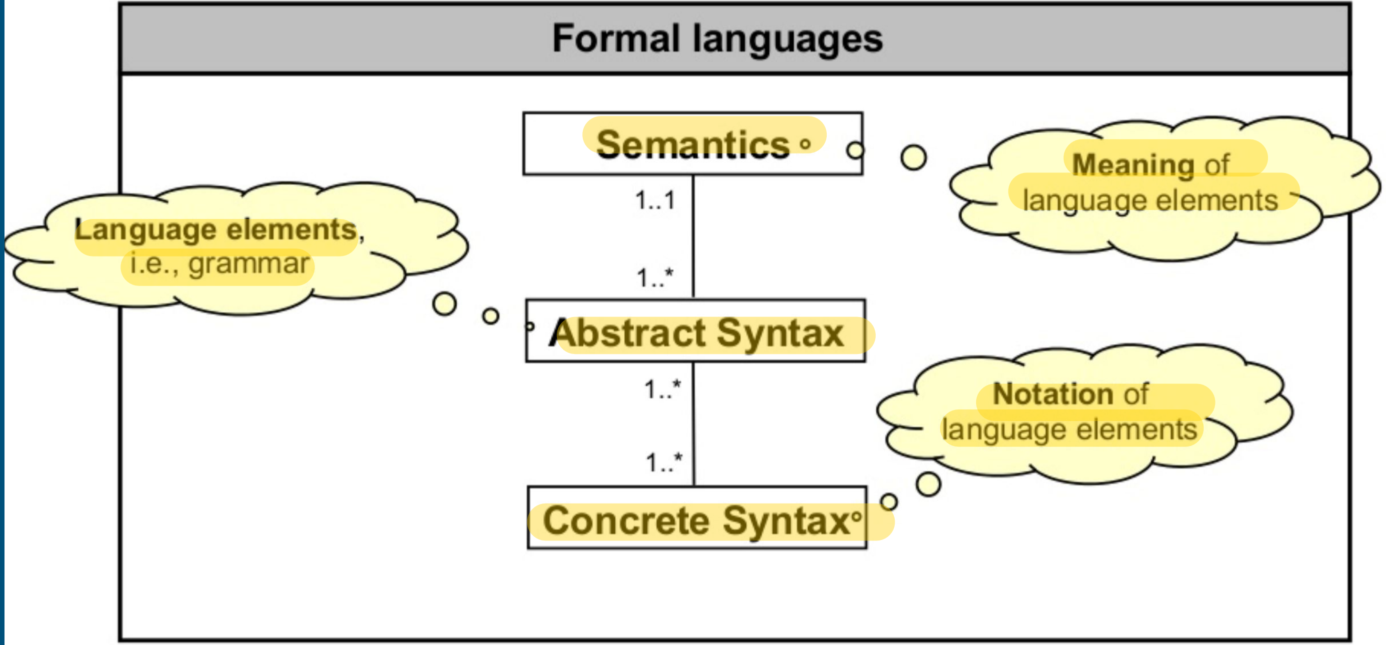
Example EBNF can model any kind of textual language, including itself, not being a threat to EBNF's usability or precision

- **Motivating example**: a simple UML Activity diagram
  - *Activity, Transition, InitialNode, FinalNode*



- **Question**: Is this UML Activity diagram **valid**?
- **Answer**: Check the **UML metamodel**!

- Languages have **divergent goals** and **fields of application, but** still have a **common** definition framework



- **Main components**
  - **Abstract syntax**: Language concepts and how these concepts can be combined (~ grammar)
    - It **does neither define** the **notation nor** the **meaning** of the concepts
  - **Concrete syntax: Notation** to illustrate the language concepts intuitively
    - **Textual**, **graphical** or a mixture of both
  - **Semantics**: **Meaning** of the language concepts
    - How language concepts are actually **interpreted**

- **Additional components**
  - **Extension** of the language by new language concepts
    - Domain or technology specific extensions, e.g., see UML Profiles
  - **Mapping** to other languages, domains
    - Examples: UML2Java, UML2SetTheory, PetriNet2BPEL, …
    - May act as translational semantic definition

- **Formal languages** have a **long tradition** in computer science
- **First attempts**: Transition from machine code instructions to high-level programming languages (Algol60)

- **Major successes**
  - Programming languages such as Java, C++, C#, …
  - Declarative languages such as XML Schema, DTD, RDF, OWL, …

- **Excursus**
  - **How** are **programming languages** and **XML-based languages** defined?
  - **What** can thereof be **learned** for defining modeling languages?

Programming languages

Overview

- John Backus and Peter Naur invented **formal languages** for the **definition of languages** called **meta-languages**
- Examples for meta-languages: BNF, EBNF, …
- Are used since 1960 for the **definition** of the **syntax** of **programming languages**
  - Remark: **abstract** and the **concrete** syntax are both defined
- EBNF Example

option sequence non-terminal

```
Java        := [PackageDec]  {ImportDec}  ClassDec;
PackageDec   := "package"  QualifiedIdentifier;
ImportDec   := "import"  QualifiedIdenfifier;
ClassDec   := Modifier  "class"  Identifier  ["extends" Identifier]
                ["implements" IdentifierList]  ClassBody;
```

production rule terminal

Programming languages

Example: MiniJava

- Grammar

```
Java := [PackageDec] {ImportDec} ClassDec;
PackageDec := "package" QualifiedIdentifier;
ImportDec := "import" QualifiedIdenfifier;
ClassDec := Modifier "class" Identifier ["extends" Identifier]
            ["implements" IdentifierList] ClassBody;
Modifier := "public" | "private" | "protected";
Identifier := {"a"-"z" | "A"-"Z" | "0"-"9"}
```

- Program

```
package mdse.book.example;
import java.util.*;
public class Student extends Person { … }
```

- Validation: *does the program conform to the grammar*?
  - Compiler: javac, gcc, …
  - Interpreter: Ruby, Python, …

- Four-layer architecture

EBNF := {rules}; *rules* := Terminal | Non-Terminal |...	**Definition of EBNF in EBNF – EBNF grammar (reflexive)**	**M3-Layer**
Java := [PackageDec] {ImportDec} ClassDec; *PackageDec* := "**package**" QualifiedIdentifier; ...	**Definition of Java in EBNF – Java grammar**	**M2-Layer**

`package mdse.book.example;`
`public class Student`
`extends Person { … }`
	Program – Sentence conform to the grammar	**M1-Layer**

	**Execution of the program**	**M0-Layer**

XML-based languages

Overview

- XML files require specific structures to allow for a standardized and automated processing
- Examples for XML meta languages
  - DTD, XML-Schema, Schematron
- **Characteristics** of XML files
  - Well-formed (character level) vs. valid (grammar level)

- **DTD Example**

element

contentParticle

1..*

```
<!ELEMENT cookbook (title, meal+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT meal (ingredient+)>
<!ELEMENT ingredient>
<!ATTLIST ingredient name CDATA #REQUIRED
                     amount CDATA #IMPLIED
                     unit CDATA #IMPLIED>
```

1..1

attributes

0..1

XML-based languages

Example: Cookbook DTD

- DTD

```
<!ELEMENT cookbook (title, meal+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT meal (ingredient+)>
<!ELEMENT ingredient>
<!ATTLIST ingredient name CDATA #REQUIRED
                     amount CDATA #IMPLIED
                     unit CDATA #IMPLIED>
```

- XML

```
<cookbook>
     <title>How to cook!</title>
     <meal name= „Spaghetti“ >
        <ingredient name = „Tomato“, amount=„300“ unit=„gramm“>
        <ingredient name = „Meat“, amount=„200“ unit=„gramm“> …
      </meal>
</cookbook>
```

- Validation
  - XML Parser: Xerces, …

XML-based languages

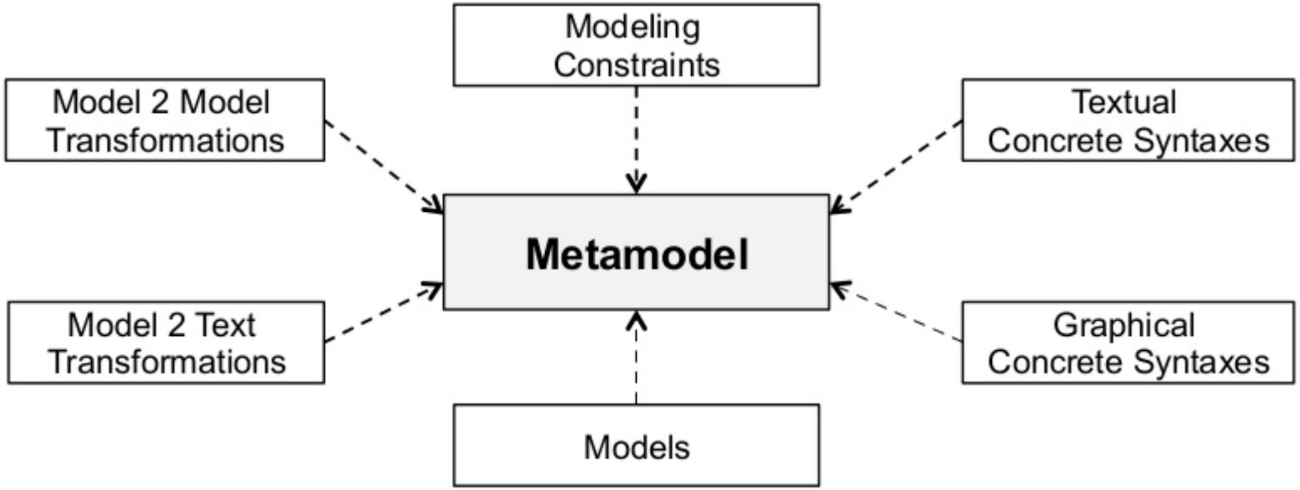
Meta-architecture layers

- Five-layer architecture (was revised with XML-Schema)

EBNF := {rules}; *rules* := Terminal | Non-Terminal |...	**Definition of EBNF in EBNF**	**M4-Layer**
ELEMENT := „<!ELEMENT " Identifier „>" ATTLIST; ATTLIST := „<!ATTLIST " Identifier …	**Definition of DTD in EBNF**	**M3-Layer**
`<!ELEMENT javaProg (packageDec*, importDec*, classDec)>` `<!ELEMENT packageDec (#PCDATA)>`	**Definition of Java in DTD – Grammar**	**M2-Layer**
<javaProg> <packageDec>**mdse.book.example**</packageDec> <classDec name=„**Student**" extends=„**Person**"/> </javaProg>	**XML – conform to the DTD**	**M1-Layer**
Concrete entities (e.g.: Student "Bill Gates")		**M0-Layer**

Abstract Syntax Metamodelling approach

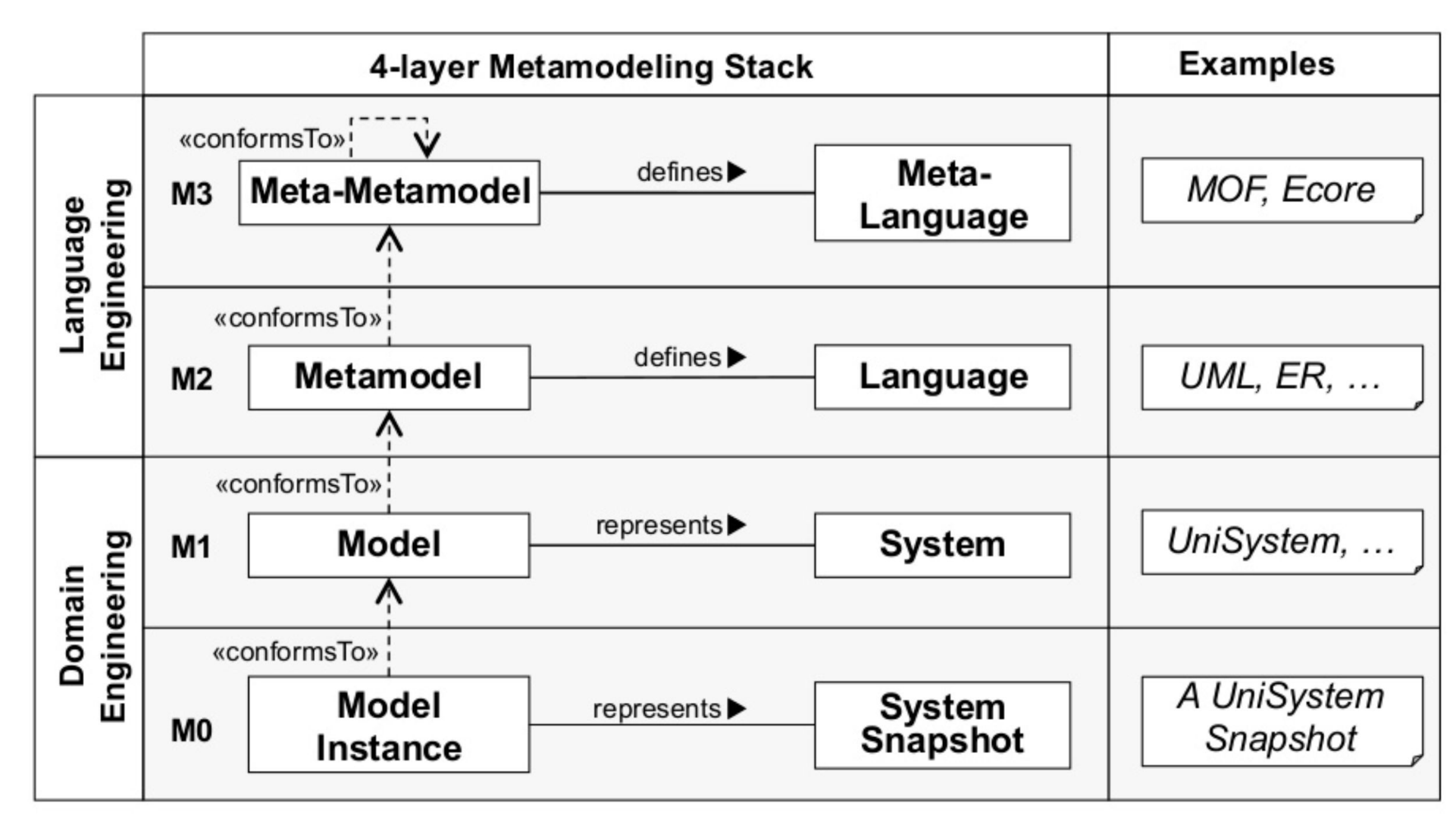
- **Metamodel-centric language design:**
All language aspects base on the abstract syntax of the language defined by its metamodel



- **Advantages** of metamodels
  - Precise, accessible, and evolvable language definition

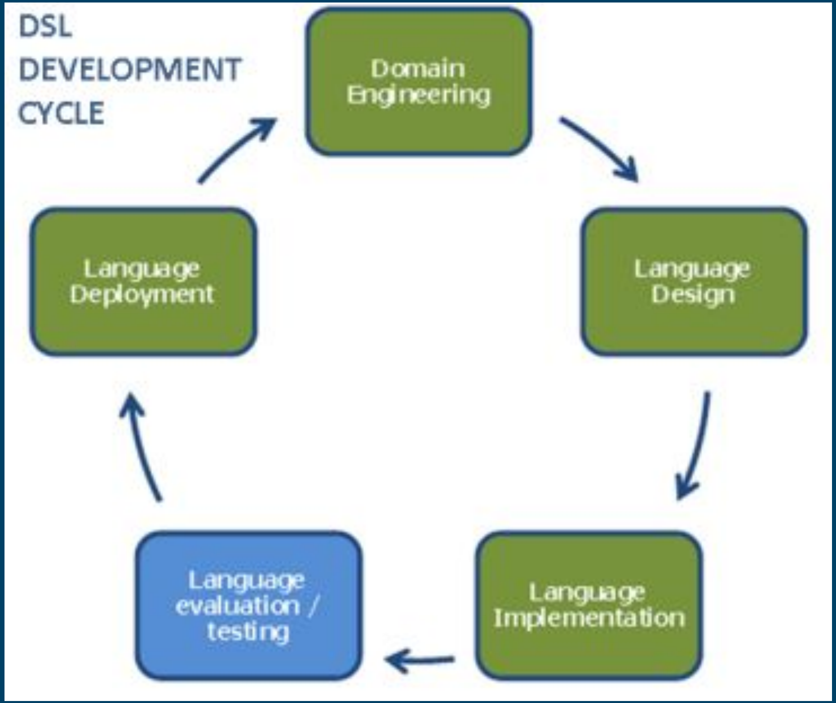
- **Generalization** on a higher level of abstraction by means of the **meta-metamodel**
  - Language concepts for the definition of metamodels
  - MOF, with Ecore as its implementation, is considered as a universally accepted meta-metamodel

- **Metamodel-agnostic** tool support
  - Common exchange format, model repositories, model editors, model validation and transformation frameworks, etc.


4-layer Metamodeling Stack
Examples
Language Engineering
Domain Engineering
«conformsTo»
M3
Meta-Metamodel
defines
Meta-Language
MOF, Ecore
«conformsTo»
M2
Metamodel
defines
Language
UML, ER, …
«conformsTo»
M1
Model
represents
System
UniSystem, …
«conformsTo»
M0
Model Instance
represents
System Snapshot
A UniSystem Snapshot

SLE Process

(as systematic)



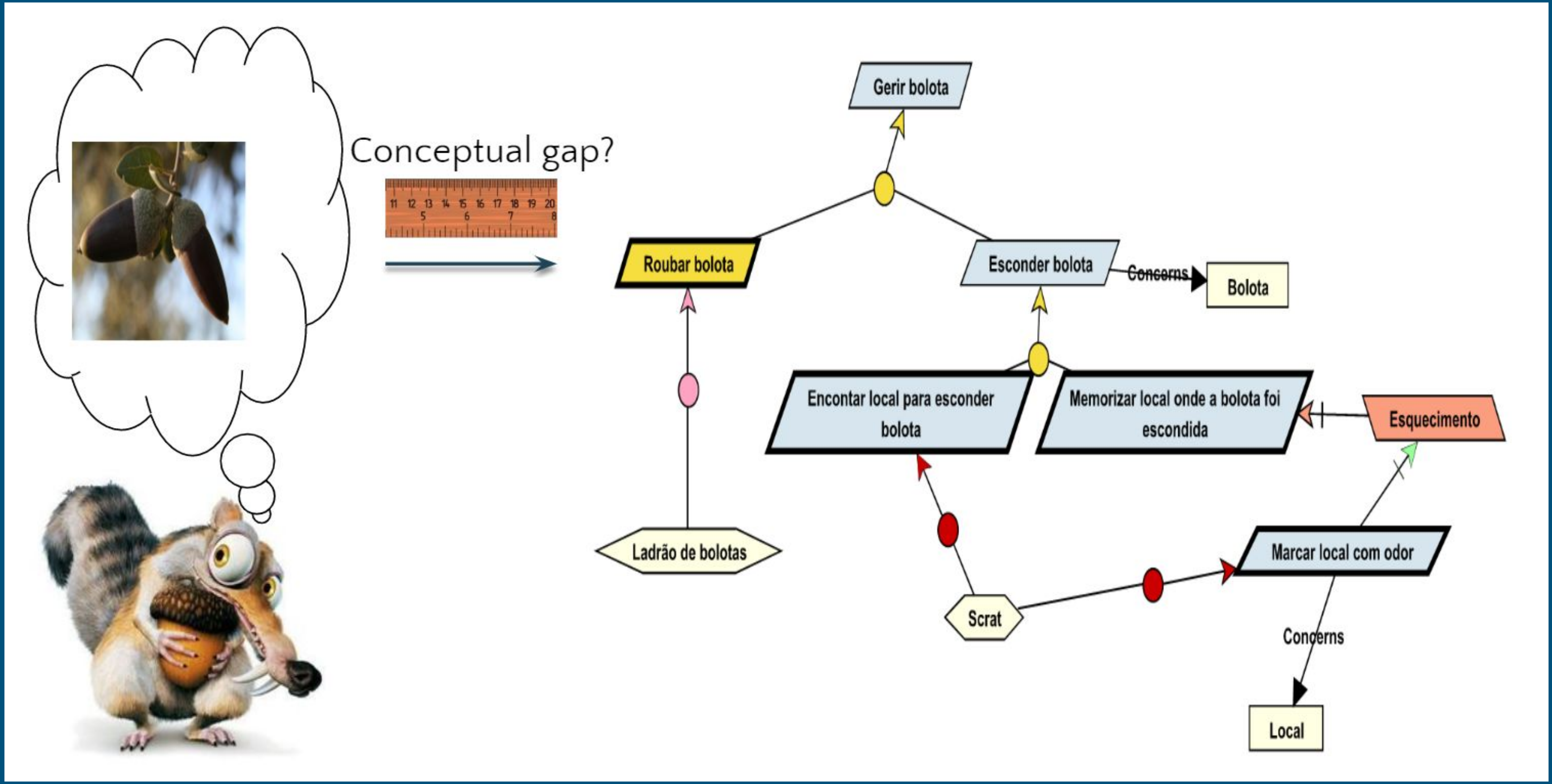
Ankica Barisic, Vasco Amaral, Miguel Goulão:
Usability driven DSL development with USE-ME. Computer Languages, Systems & Structures 51: 118-157 (2018)

Metamodel development process

Incremental and Iterative

Modeling domain analysis	Modeling language design	Modeling language validation
Identify purpose, realiza-tion, and content of the modeling language	**Formalize** modeling language by defining a metamodel	**Instantiate** metamodel by modeling reference models
Sketch reference modeling examples	**Formalize** modeling constraints using OCL	**Collect** feedback for next iteration

The language has to empower its user... or he will end up using something else



What strategies are available to us?

Constructive approaches:
- Our own expertise and common sense
- Usability heuristics such as the "Physics of notations"

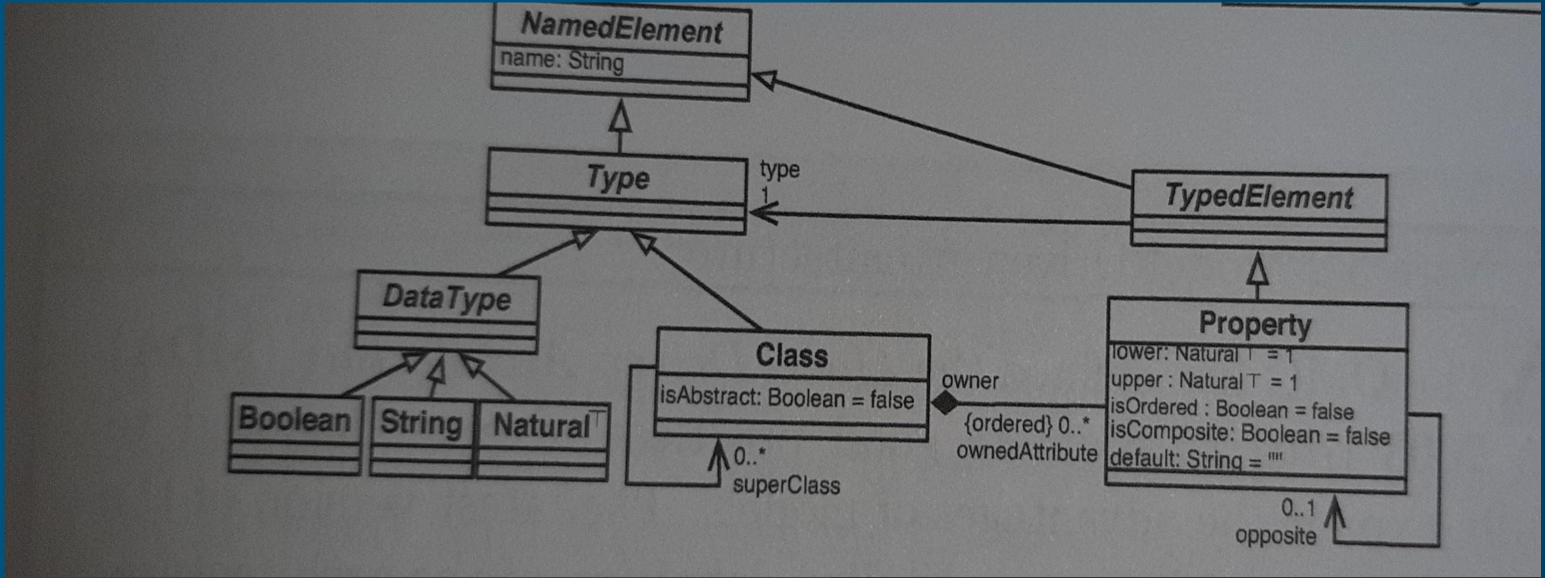
Evaluation-based approaches:
- "Traditional" usability evaluations
- User monitoring while using the DSML

Meta-Object Facility (MOF)

Modelling formalism standardized by OMG to specify concepts and relationships between these concepts for a particular domain. MOF can be used for Domain Modelling and to describe the abstract syntax of a corresponding DSML

Meta-Object Facility (MOF)

Allows specifying concepts of a given domain in a package.
Package contains Classes, Properties, and relationships
Property can be an attribute or a reference to other class
Attribute is typed by enumeration or primitive type such as Boolean, String, integer, Real or Unlimited Natural



MOF - Meta Object Facility

Introduction 1/3

- **OMG standard** for the **definition of metamodels**

- MOF is an **Object-Oriented** modeling language
  - **Objects** are described by **classes**
  - **Intrinsic properties** of objects are defined as **attributes**
  - **Extrinsic properties** (links) between objects are defined as **associations**
  - **Packages** group classes

- MOF itself is defined by MOF (reflexive) and divided into
  - **eMOF** (essential MOF)
    - Simple language for the definition of metamodels
    - Target audience: **metamodelers**
  - **cMOF** (complete MOF)
    - Extends eMOF
    - Supports management of meta-data via enhanced services (e.g. reflection)
    - Target audience: **tool manufacturers**

MOF - Meta Object Facility

Introduction 2/3

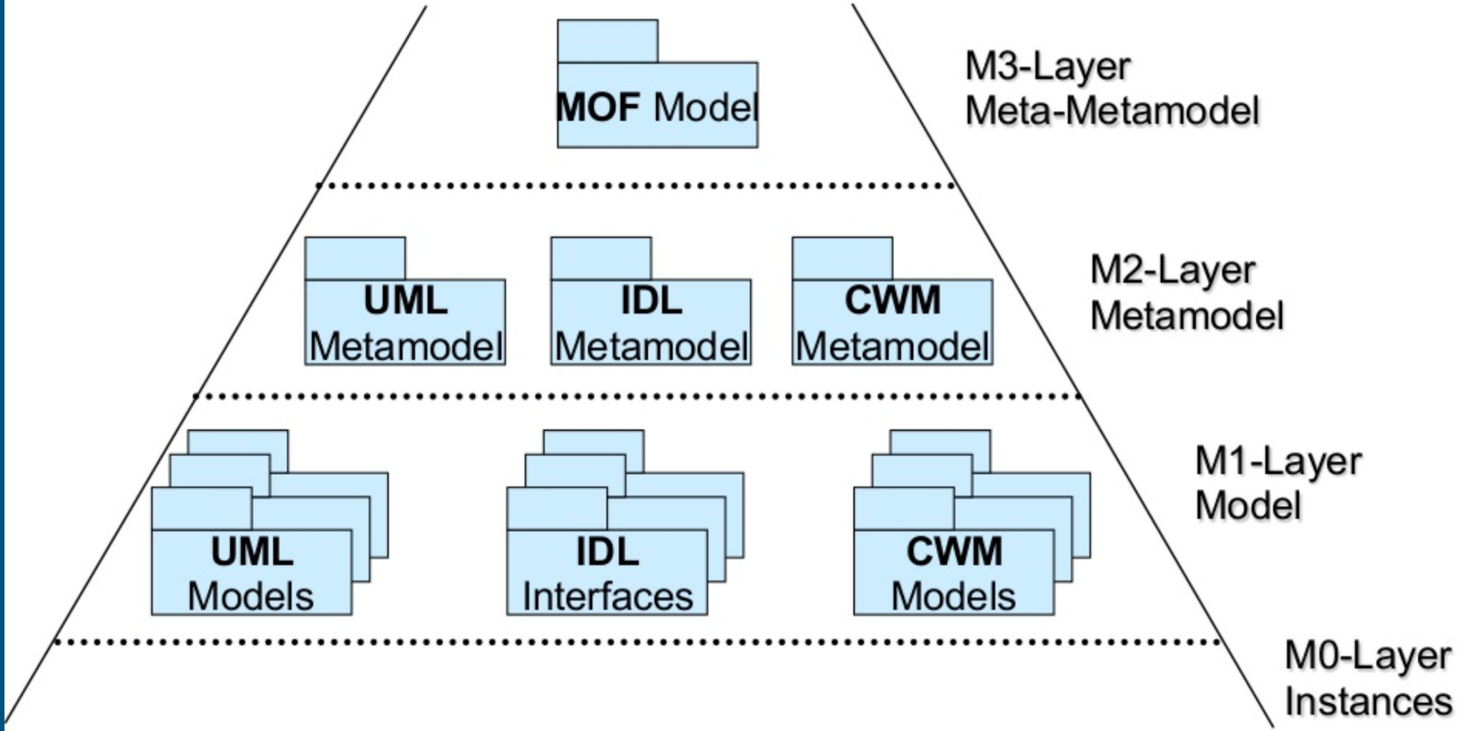
- Offers **modeling infrastructure** not only for MDA, but for MDE in general
  - MDA dictates MOF as meta-metamodel
  - UML, CWM and further OMG standards are conform to MOF

- **Mapping rules** for various **technical platforms** defined for MOF
  - XML: XML Metadata Interchange (XMI)
  - Java: Java Metadata Interfaces (JMI)
  - CORBA: Interface Definition Language (IDL)

MOF - Meta Object Facility

Introduction 3/3

- OMG language definition stack



Why an additional language for M3

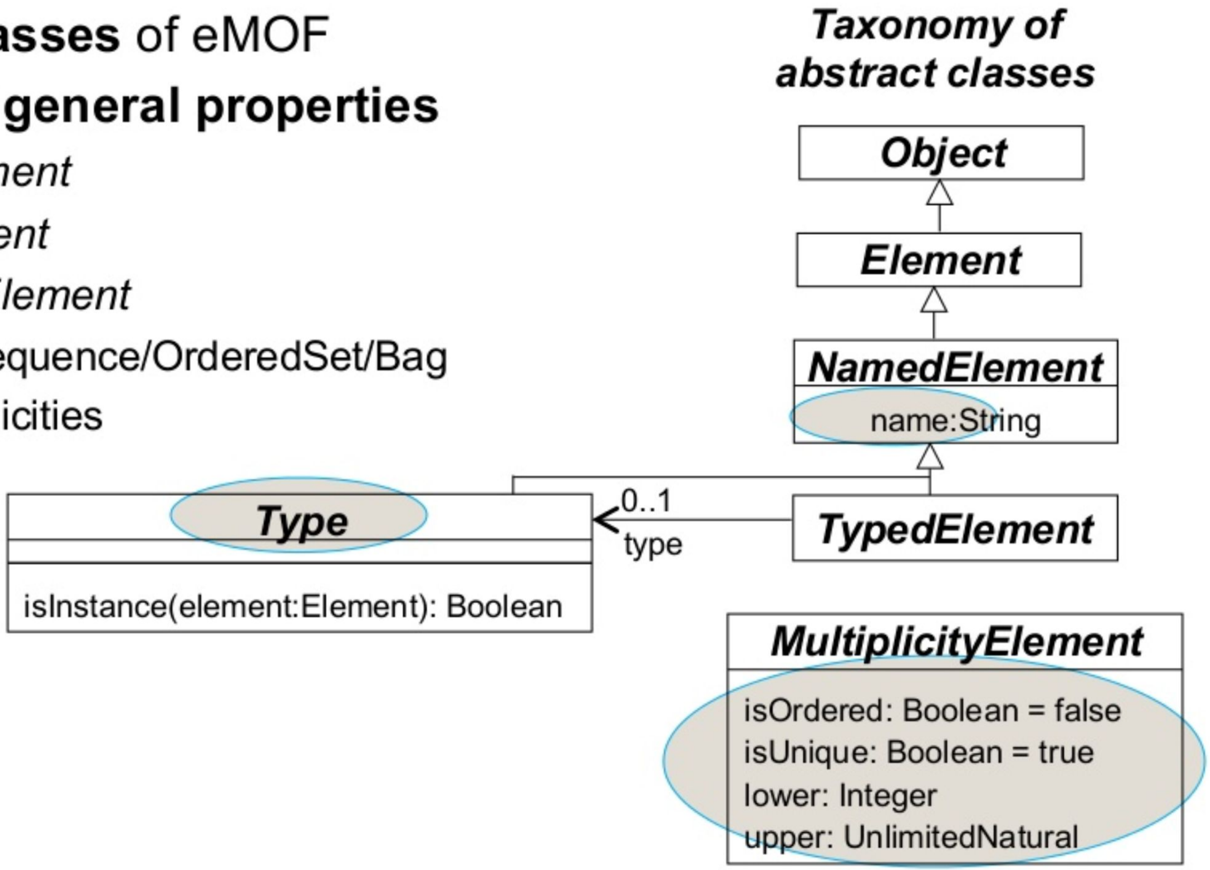
... isn‘t UML enough?

- **MOF** only a **subset** of **UML**
  - MOF is **similar** to the UML class diagram, but much more limited
  - No n-ary associations, no association classes, …
  - No overlapping inheritance, interfaces, dependencies, …
- Main differences result from the **field of application**
  - UML
    - Domain: **object-oriented modeling**
    - Comprehensive modeling language for various software systems
    - **Structural** and **behavioral modeling**
    - **Conceptual** and **implementation modeling**
  - MOF
    - Domain: **metamodeling**
    - Simple **conceptual structural modeling language**
- **Conclusion**
  - MOF is a highly **specialized DSML** for metamodeling
  - **Core** of UML and MOF (almost) **identical**

MOF – Meta Object Facility

Language architecture of MOF 2.0

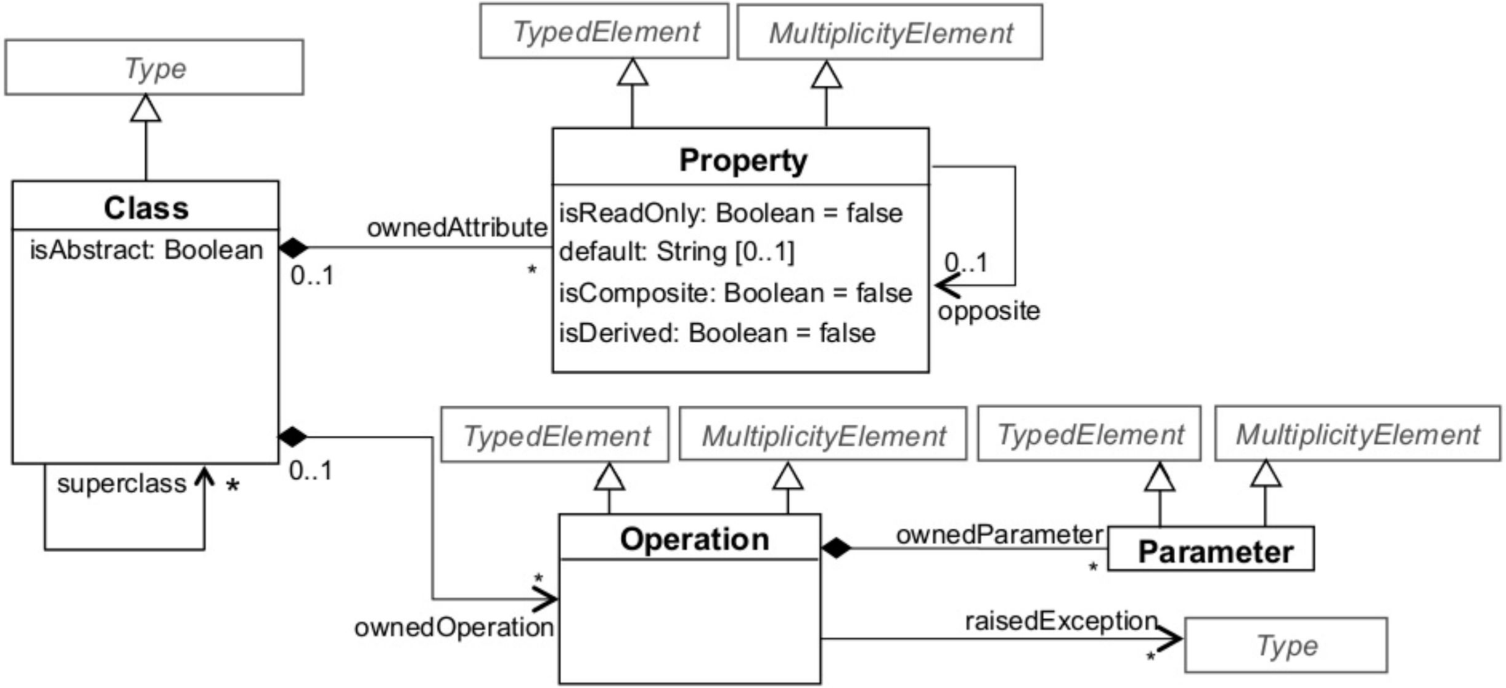
- **Abstract classes** of eMOF
- Definition of **general properties**
  - *NamedElement*
  - *TypedElement*
  - *MultiplicityElement*
    - Set/Sequence/OrderedSet/Bag
    - Multiplicities



MOF – Meta Object Facility

Language architecture of MOF 2.0

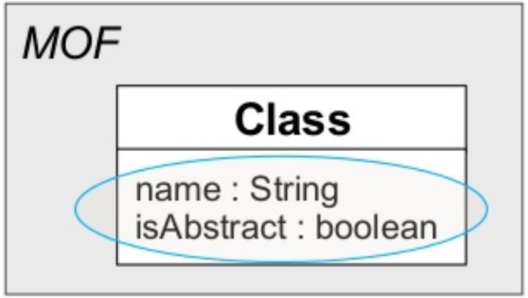
- **Core** of eMOF
  - Based on object-orientation
  - Classes, properties, operations, and parameters

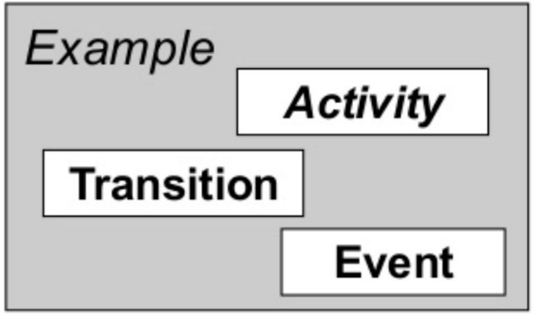


MOF – Meta Object Facility

Classes

- A class specifies ***structure*** and ***behavior*** of a **set of objects**
  - **Intentional** definition
  - An unlimited number of instances (objects) of a class may be created
- A class has an **unique name** in its namespace
- Abstract classes cannot be instantiated!
  - **Only useful in inheritance hierarchies**
  - Used for **»highlighting«** of **common features** of a set of subclasses
- Concrete classes can be instantiated!

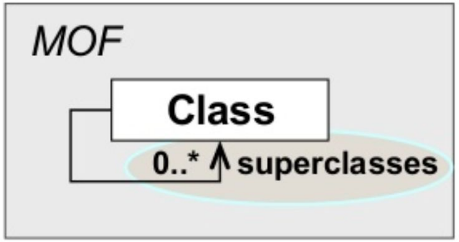


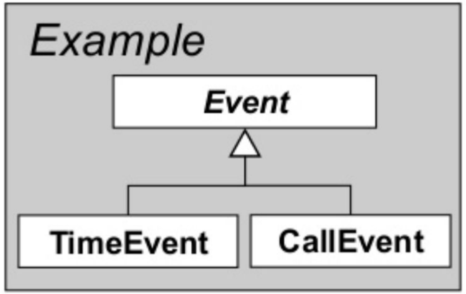


MOF – Meta Object Facility

Generalization

- **Generalization**: relationship between
  - a **specialized class** (*subclass*) and
  - a **general class** (*superclass*)
- Subclasses **inherit** properties of their superclasses and may add further properties
- Discriminator: „virtual“ attribute used for the **classification**
- **Disjoint** (non-overlapping) generalization
- **Multiple inheritance**

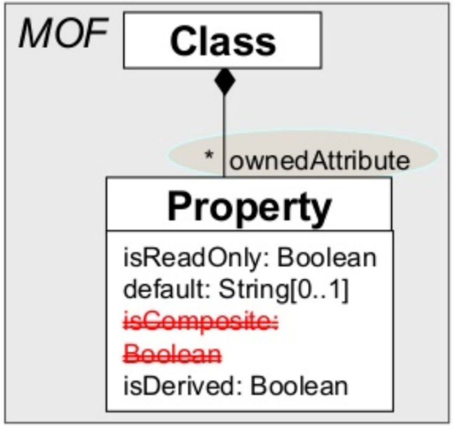


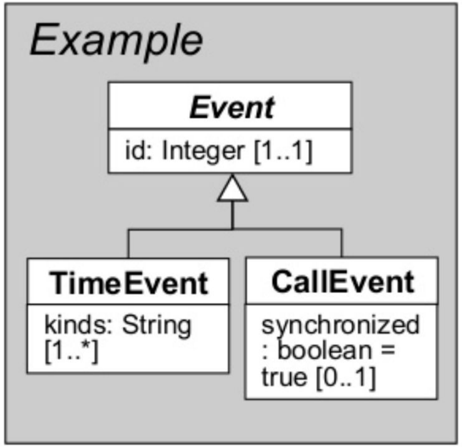


MOF – Meta Object Facility

Attributes

- **Attributes** describe *inherent* characteristics of *classes*
- Consist of a **name** and a **type** (obligatory)
- **Multiplicity**: how many values can be stored in an attribute slot (obligatory)
  - Interval: **upper** and **lower limit** are natural numbers
  - * asterisk - also possible for upper limit (Semantics: *unlimited number*)
  - 0..x means optional: null values are allowed
- **Optional**
  - **Default** value
  - **Derived** (calculated) attributes
  - **Changeable**: isReadOnly = false
  - isComposite is always true for attributes





MOF – Meta Object Facility

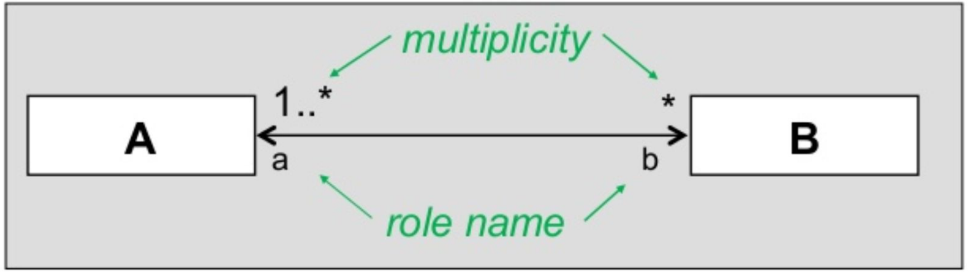
Associations

- An **association** describes the common structure of a set of relationships between objects
- MOF only allows ***unary*** and ***binary*** associations, i.e., defined between **two** classes
- **Binary associations** consist of **two roles** whereas each role has
  - **Role name**
  - **Multiplicity** limits **the number of** partner objects of an object
- **Composition**
  - „part-whole" relationship (also "part-of" relationship)
  - One part can be **at most** part of **one composed object** at one time
  - Asymmetric and transitive
  - Impact on multiplicity: 1 or 0..1

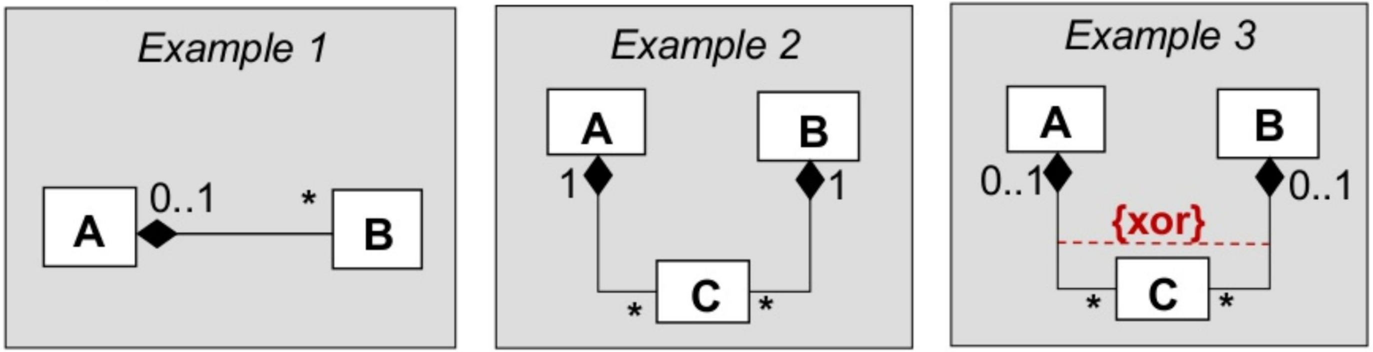
MOF – Meta Object Facility

Associations - Examples

- **Association**



- **Composition**



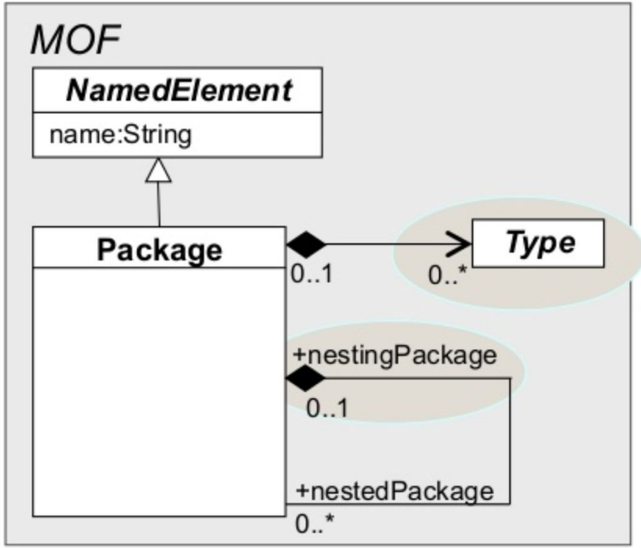
Example 2: *Syntax* ✓ *Semantics* ✗

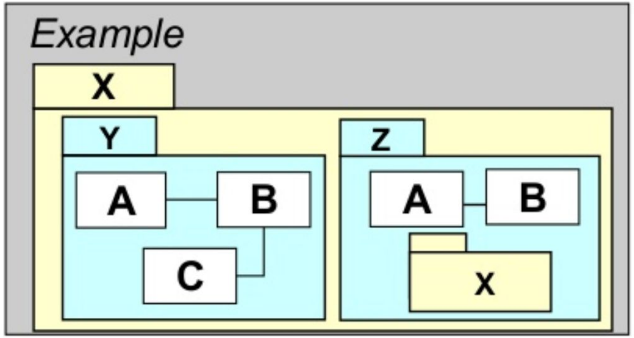
Example 3: *Syntax* ✓ *Semantics* ✓

MOF – Meta Object Facility

Packages

- Packages serve as a **grouping mechanism**
  - Grouping of related types, i.e., classes, enumerations, and primitive types.
- Partitioning criteria
  - Functional or information cohesion
- Packages form **own namespace**
  - Usage of identical names in different parts of a metamodel
- Packages may be **nested**
  - *Hierarchical grouping*
- Model elements are contained in **one** package

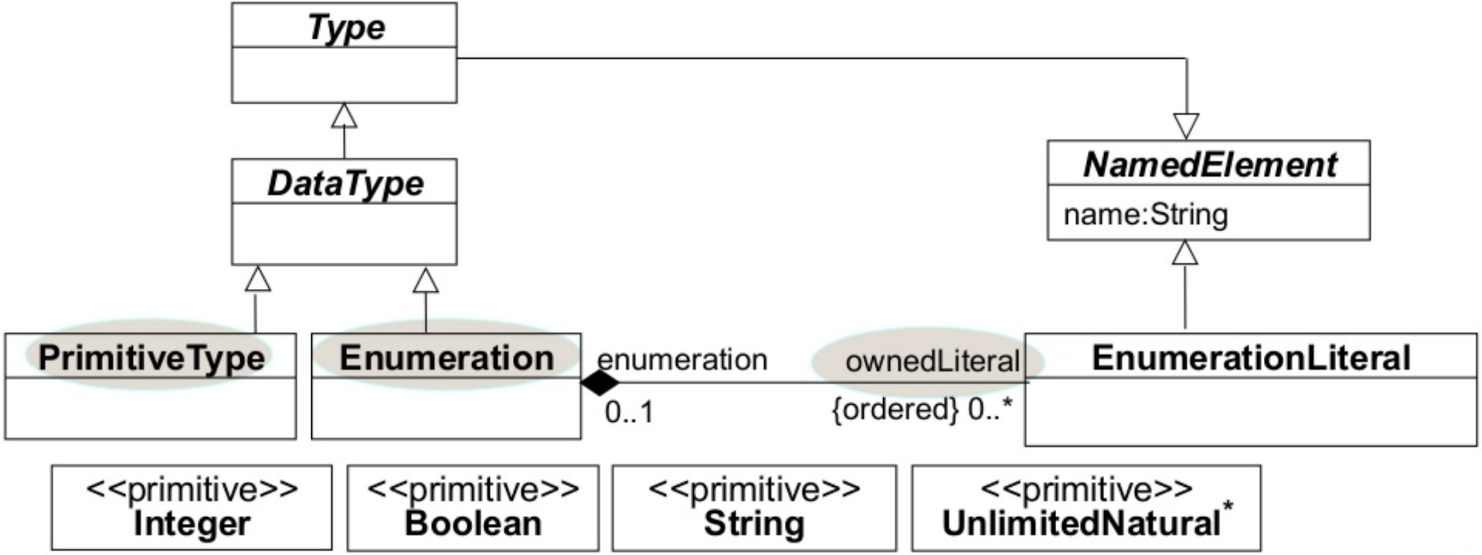




MOF – Meta Object Facility

Types 1/2

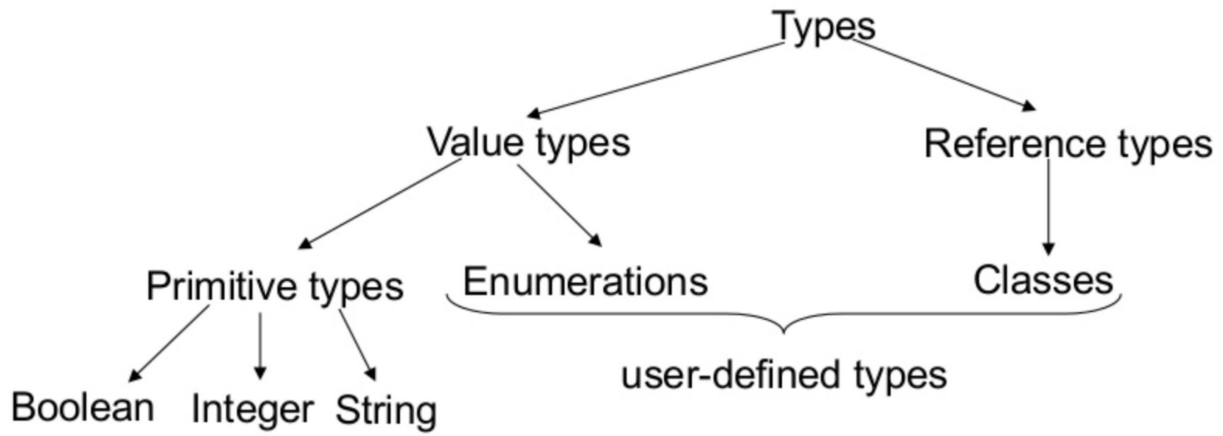
- **Primitive data types**: Predefined types for integers, character strings and Boolean values
- **Enumerations**: Enumeration types consisting of named constants
  - Allowed values are defined in the course of the declaration
    - Example: `enum Color {red, blue, green}`
  - Enumeration types can be used as data types for attributes



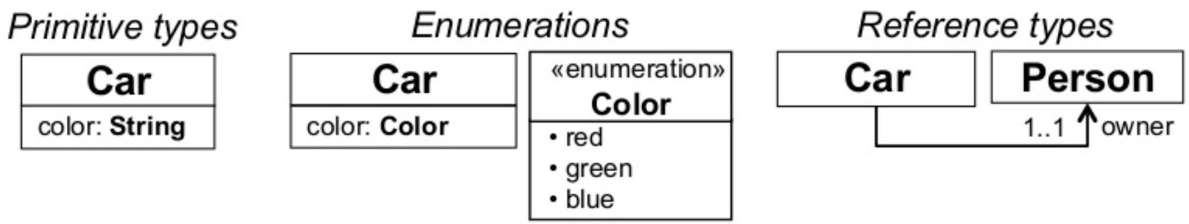
MOF – Meta Object Facility

Types 2/2

- Differentiation between ***value types* and *reference types***
  - Value types: contain a direct value (e.g., 123 or ‘x‘)
  - Reference types: contain a reference to an object

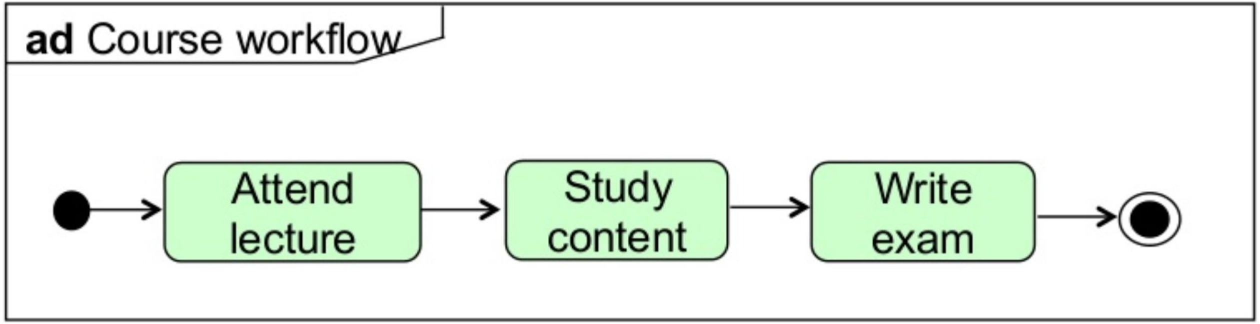


- **Examples**



Example 1/9

- **Activity diagram example**
  - Concepts: *Activity, Transition, InitialNode, FinalNode*
  - Domain: Sequential linear processes



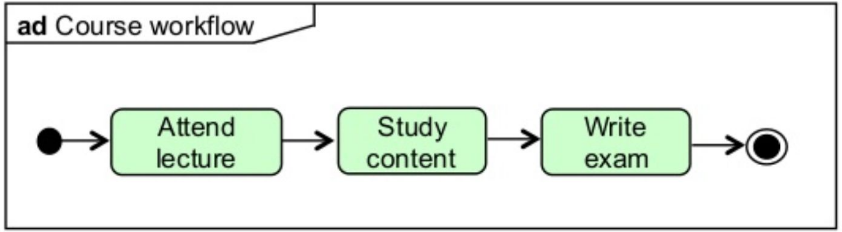
- Question: How does a possible metamodel to this language look like?

- Answer: apply metamodel development process!

Example 2/9

Identification of the modeling concepts

Example model = Reference Model

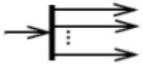
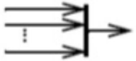


Notation table

Syntax	*Concept*
ad name	ActivityDiagram
◉	FinalNode
●	InitialNode
name	Activity
→	Transition

- Several languages have **no formalized definition** of their **concrete syntax**

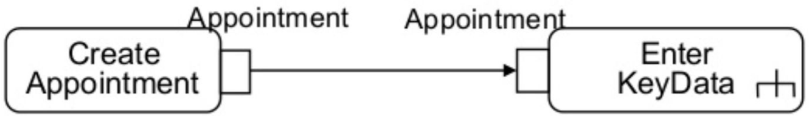
- Example – Excerpt from the UML-Standard

NODE TYPE	*NOTATION*	*REFERENCE*
ForkNode		See ForkNode (from IntermediateActivities) on page -404.
InitialNode		See InitialNode (from BasicActivities) on page -406.
JoinNode		See "JoinNode (from CompleteActivities, IntermediateActivities)" on page 411.
MergeNode		See "MergeNode (from IntermediateActivities)" on page 416.

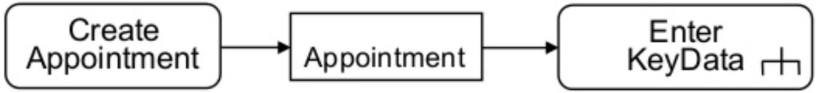
- Concrete syntax **improves** the **readability** of models
  - Abstract syntax not intended for humans!
- **One** abstract syntax may have **multiple** concrete ones
  - Including textual and/or graphical
  - Mixing textual and graphical notations still a challenge!

- **Example** – Notation alternatives for the creation of an appointment

Notation alternative 1:



Notation alternative 2:



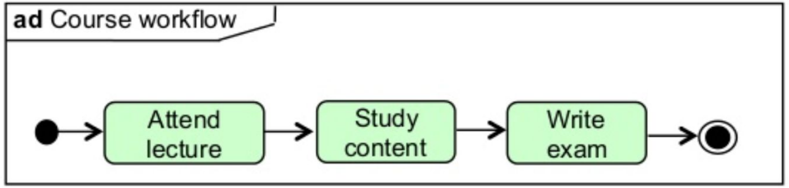
Notation alternative 3:

```
Appointment a;
a = new Appointment;
EnterKeyData (a);
```

Example 3/9

Determining the properties of the modeling concepts

Example model

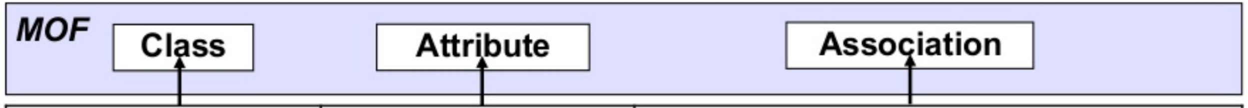


Modeling concept table

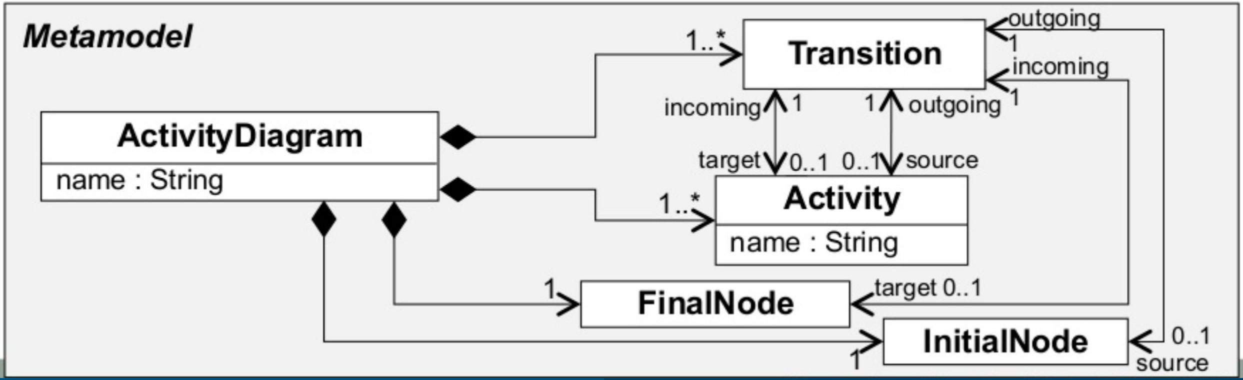
Concept	*Intrinsic properties*	*Extrinsic properties*
ActivityDiagram	Name	1 *InitialNode* 1 *FinalNode* Unlimited number of *Activities* and *Transitions*
FinalNode	-	Incoming *Transitions*
InitialNode	-	Outgoing *Transitions*
Activity	Name	Incoming and outgoing *Transitions*
Transition	-	Source node and target node Nodes: *InitialNode*, *FinalNode*, *Activity*

Example 4/9

Object-oriented design of the language

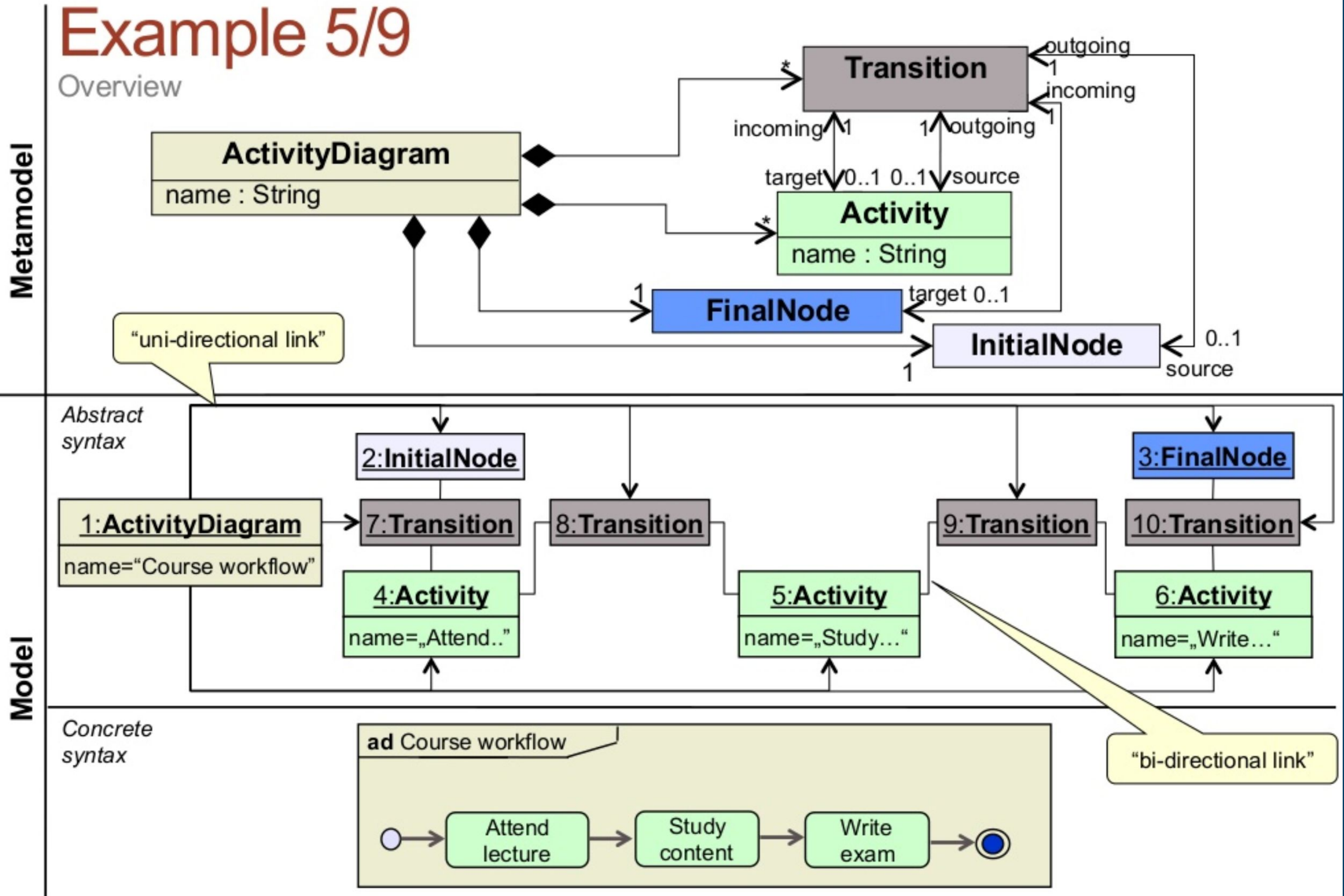


Concept	Intrinsic properties	Extrinsic properties
ActivityDiagram	Name	1 *InitialNode* 1 *FinalNode* Unlimited number of Activities and Transitions
FinalNode	-	Incoming *Transition*
InitialNode	-	Outgoing *Transition*
Activity	Name	Incoming and outgoing *Transition*
Transition	-	Source node and target node Nodes: *InitialNode*, *FinalNode*, *Activity*



Example 5/9

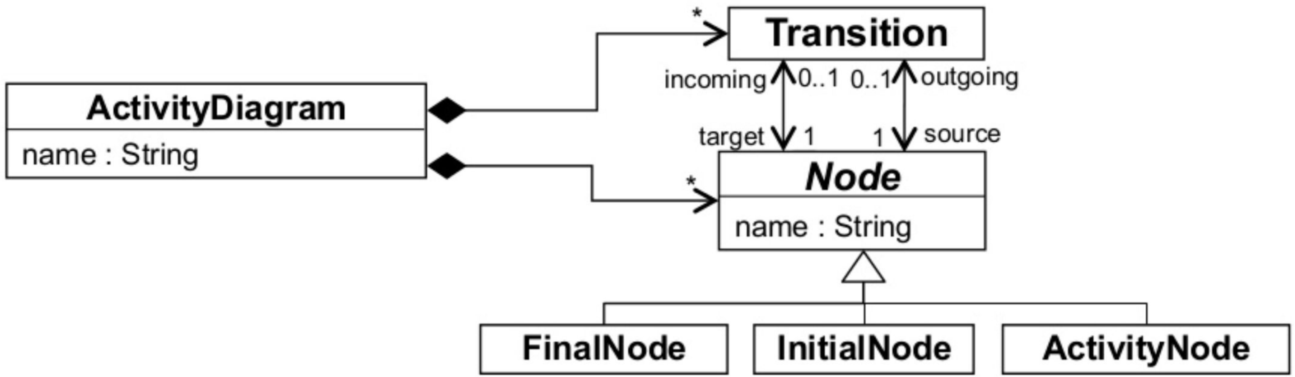
Overview



Example 6/9

Applying refactorings to metamodels

Metamodel



OCL Constraints

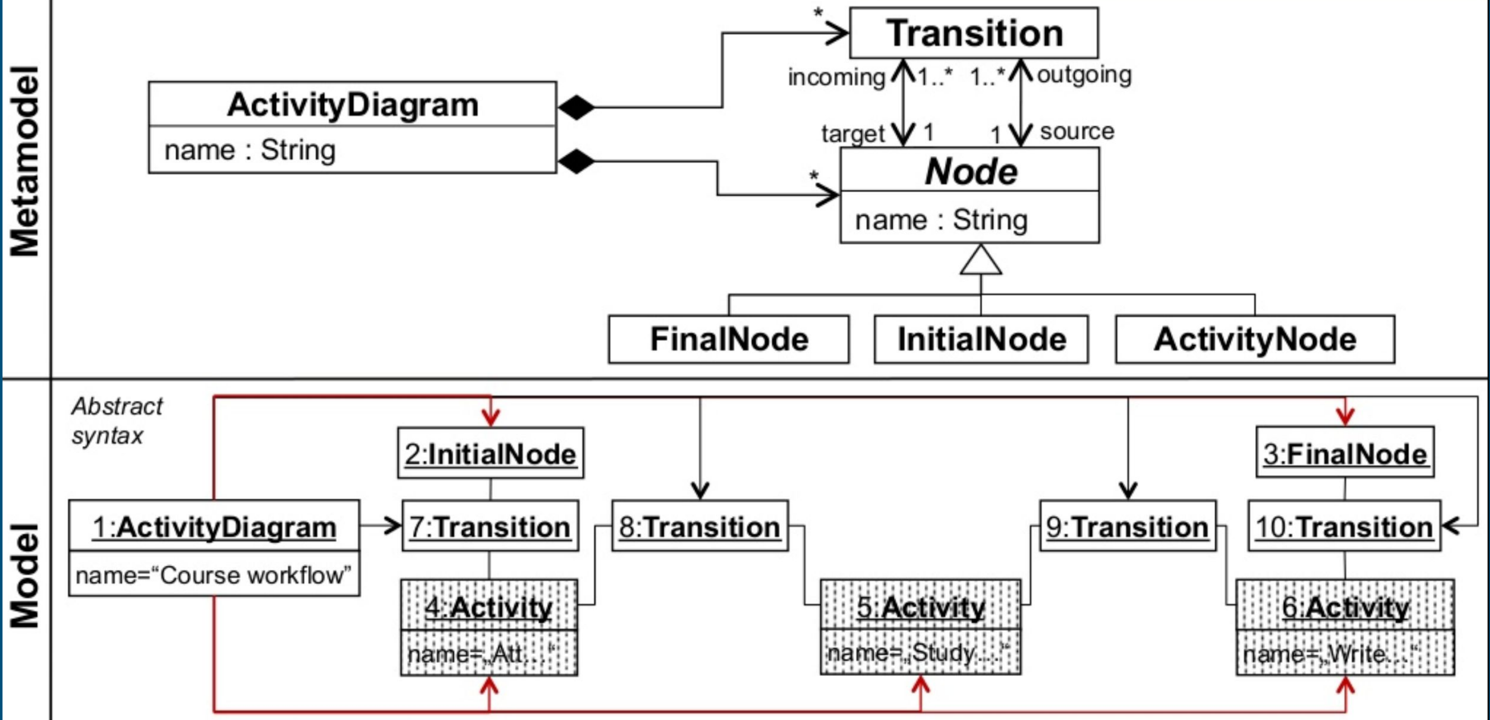
```
context ActivityDiagram
inv: self.nodes -> exists(n|n.isTypeOf(FinalNode))
inv: self.nodes -> exists(n|n.isTypeOf(InitialNode))

context FinalNode
inv: self.outgoing.oclIsUndefined()

context InitialNode
inv: self.incoming.oclIsUndefined()

context ActivityDiagram
inv: self.name <> '' and self.name <> OclUndefined …
```

Example 7/9

Impact on existing models

Changes:

- **Deletion** of class Activity
- **Addition** of class ActivityNode
- **Deletion** of redundant references



Validation errors:

- Class Activity is unknown,
- Reference finalNode, initialNode, activity are unknown

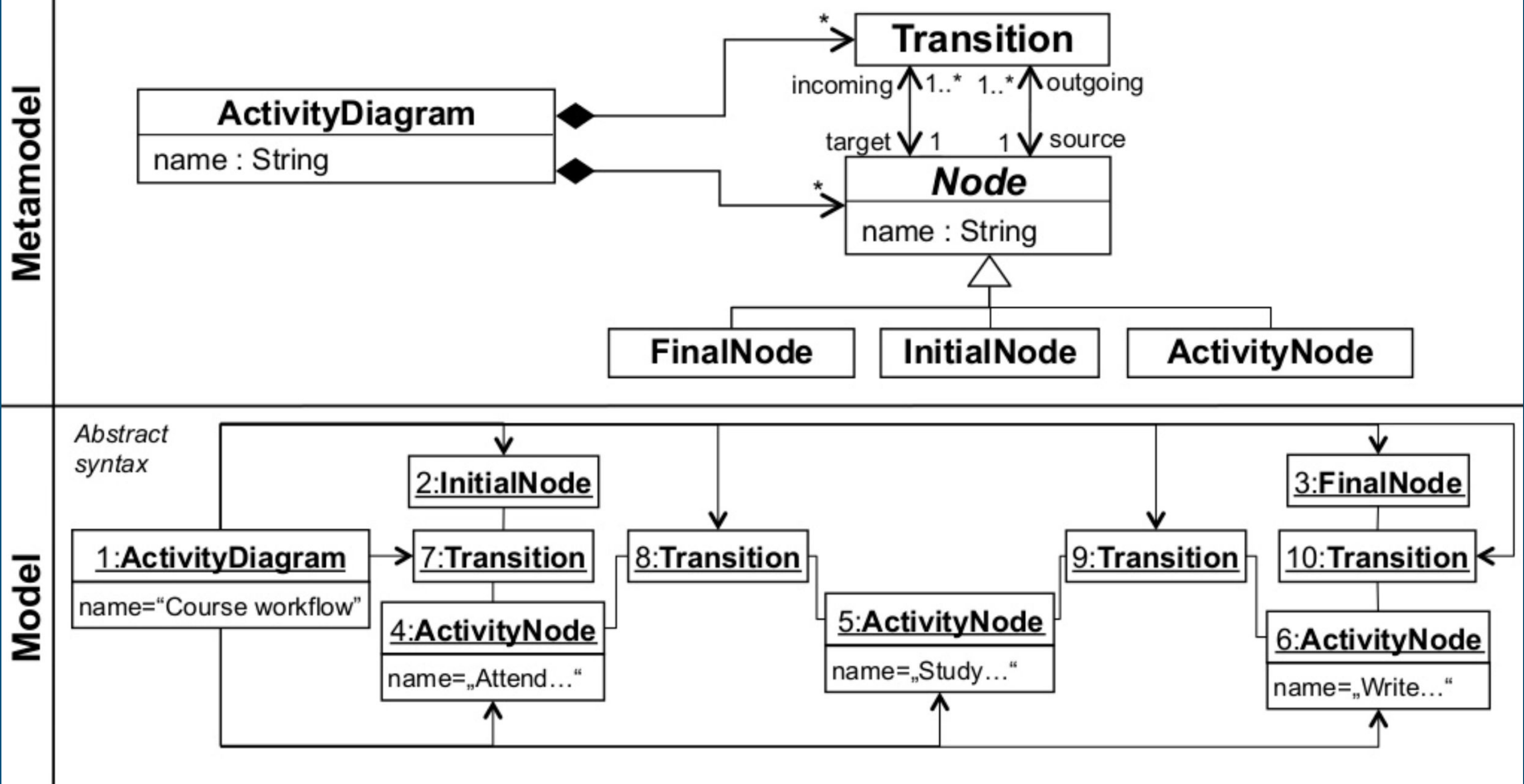
Example 8/9

How to keep metamodels evolvable when models already exist

- **Model/metamodel co-evolution problem**
  - Metamodel is changed
  - Existing models eventually become invalid
- **Changes** may **break** conformance relationships
  - Deletions and renamings of metamodel elements
- **Solution**: **Co-evolution rules** for models **coupled** to metamodel **changes**
  - Example 1: Cast all *Activity* elements to *ActivityNode* elements
  - Example 2: Cast all *initialNode*, *finalNode*, and *activity* links to *node* links

Example 9/9

Adapted model for new metamodel version



Eclipse Modelling Framework (EMF)

EMF is a Modelling framework in the Eclipse workbench.Has tools such as reflective editors, XML serialization of models, uniform way to access models from Java

ECORE

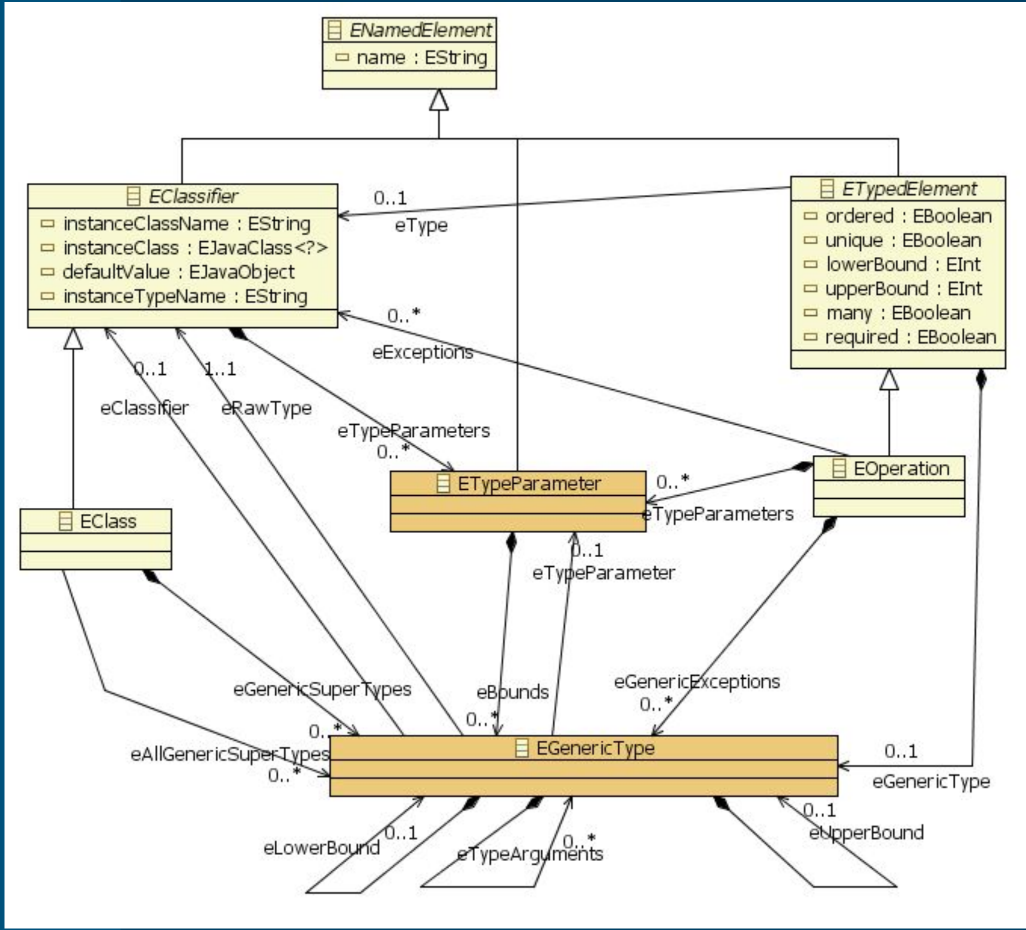
EMF meta-language (implementation of MOF)

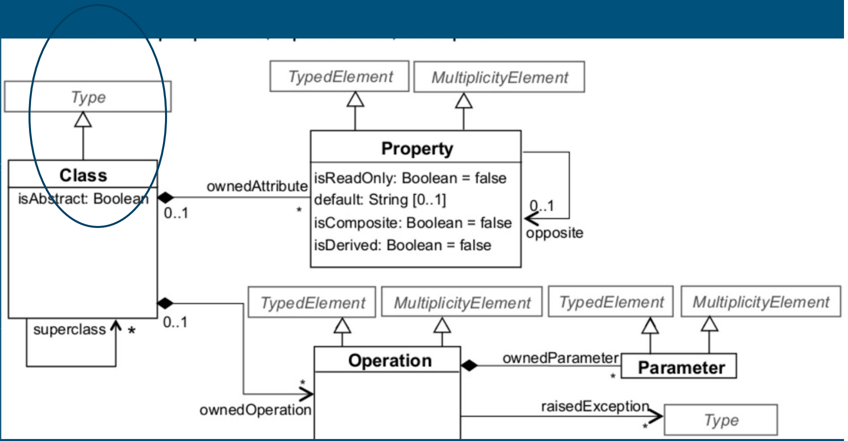
Some remarks:

To avoid confusion in eclipse (for instance with the underlying Java elements) Ecore has prefixed all concepts with an **E**.

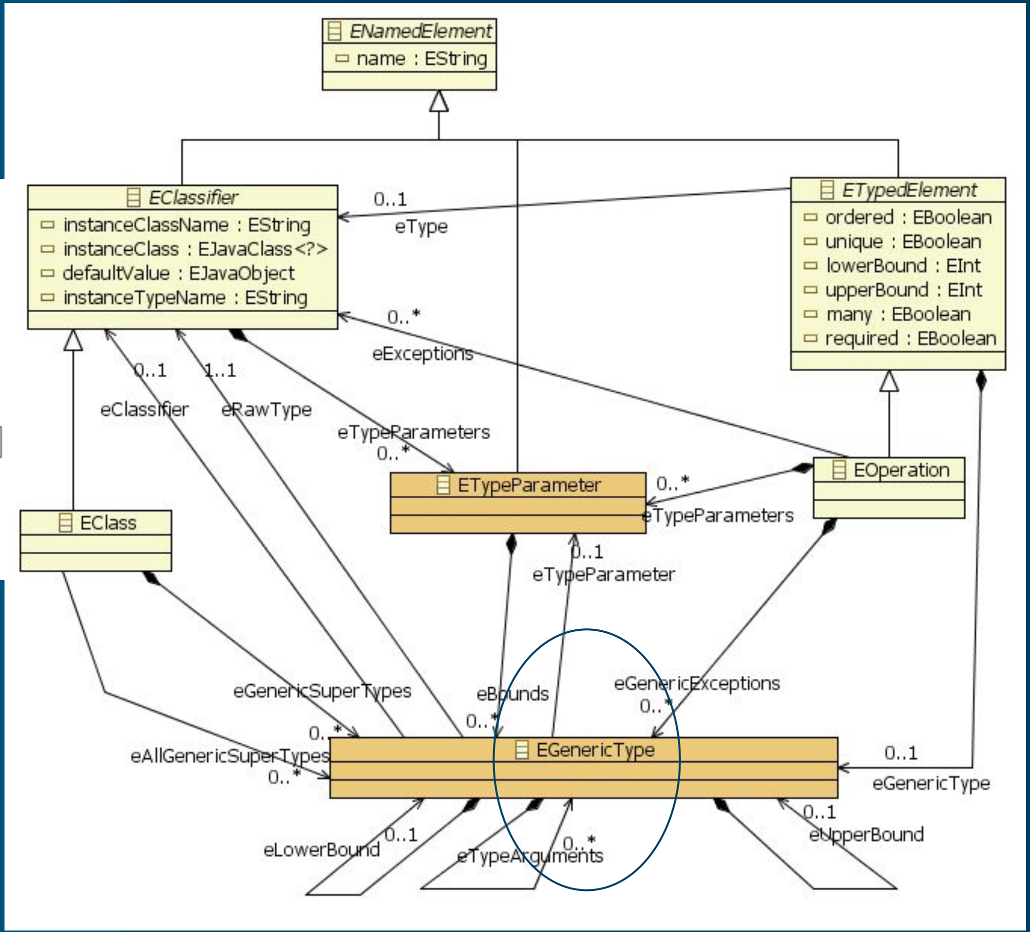
EMF is not tied with Eclipse as any java application with the EMF runtime jars in its classpath can use the project to manipulate models

Generic Ecore Metamodel

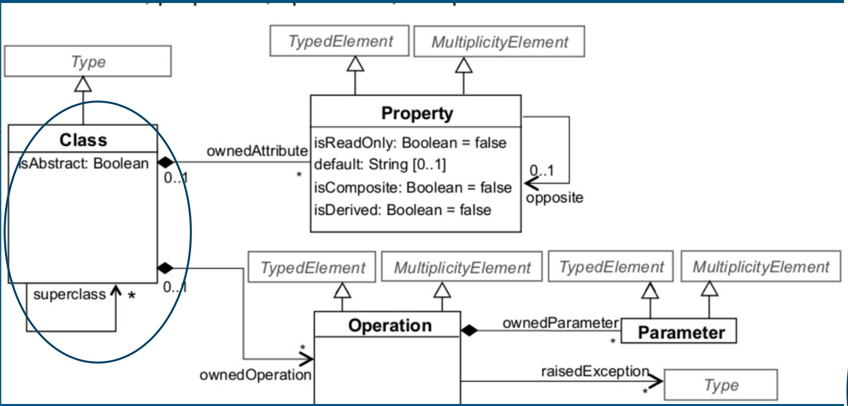




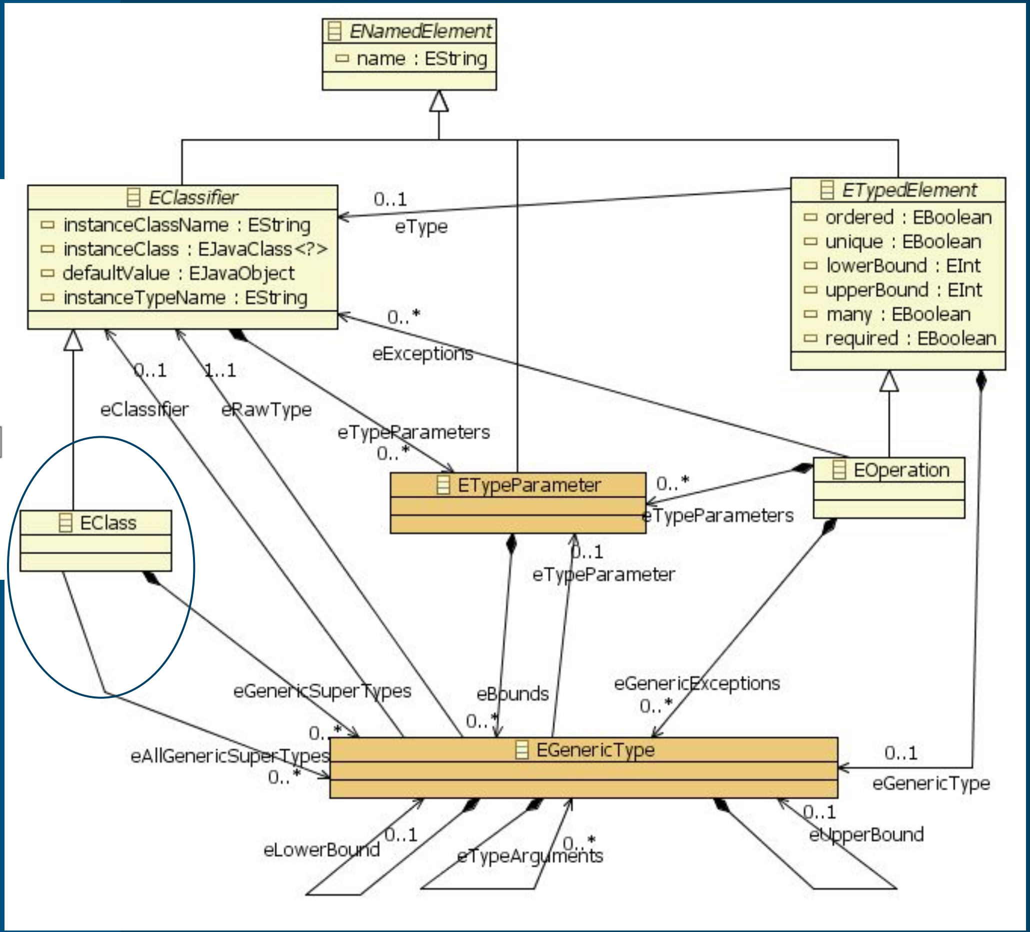
Type
TypedElement
MultiplicityElement
Class
isAbstract: Boolean
ownedAttribute
0..1
*
Property
isReadOnly: Boolean = false
default: String [0..1]
isComposite: Boolean = false
isDerived: Boolean = false
0..1
opposite
superclass
*
0..1
TypedElement
MultiplicityElement
TypedElement
MultiplicityElement
Operation
ownedParameter
*
Parameter
*
ownedOperation
raisedException
*
Type



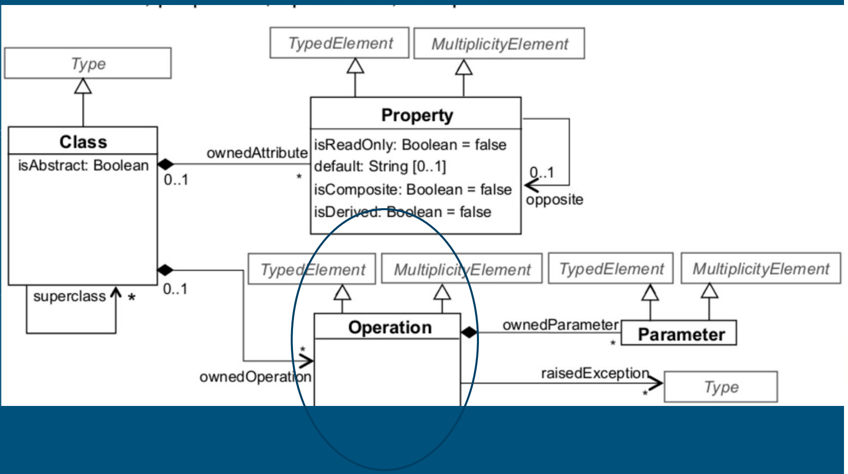
ENamedElement
name : EString
EClassifier
instanceClassName : EString
instanceClass : EJavaClass<?>
defaultValue : EJavaObject
instanceTypeName : EString
0..1
eType
ETypedElement
ordered : EBoolean
unique : EBoolean
lowerBound : EInt
upperBound : EInt
many : EBoolean
required : EBoolean
0..*
eExceptions
0..1
1..1
eClassifier
eRawType
eTypeParameters
0..*
ETypeParameter
0..*
eTypeParameters
EOperation
EClass
0..1
eTypeParameter
eGenericSuperTypes
eBounds
eGenericExceptions
0..*
0..*
0..*
eAllGenericSuperTypes
0..*
EGenericType
0..1
eGenericType
eLowerBound
0..1
eTypeArguments
0..*
0..1
eUpperBound

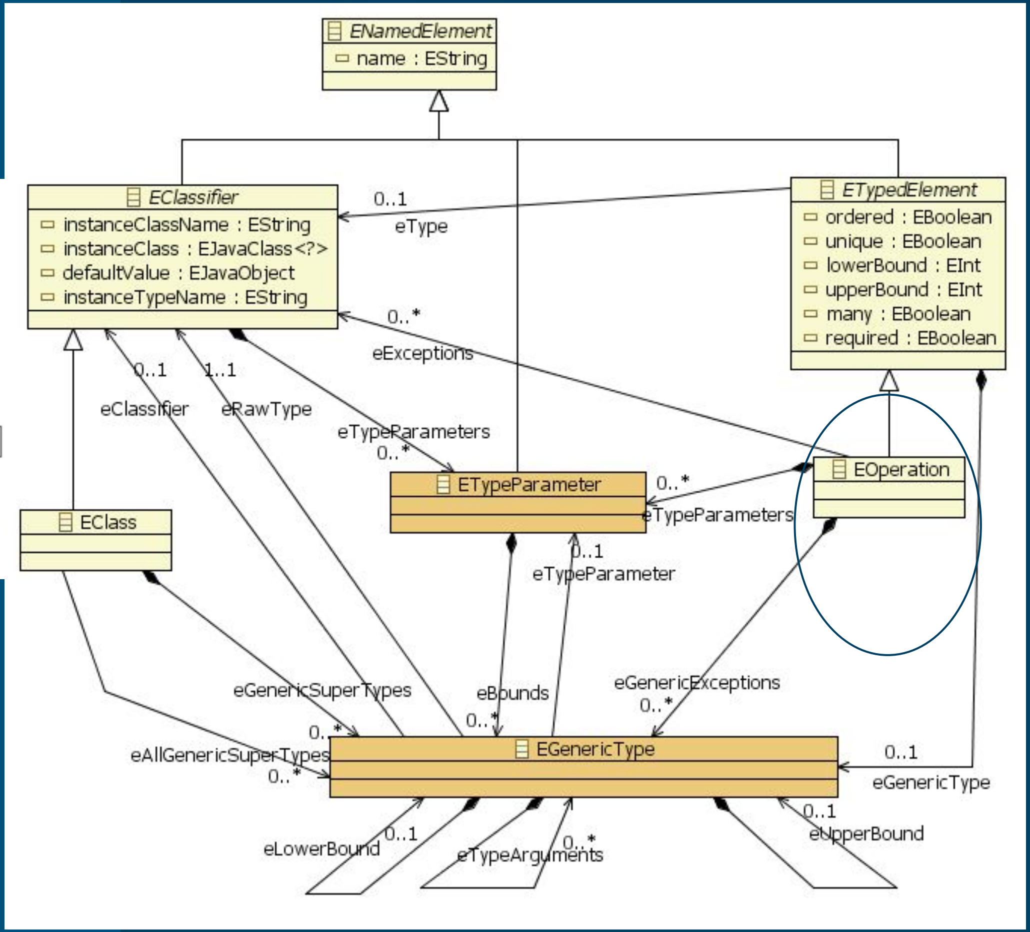


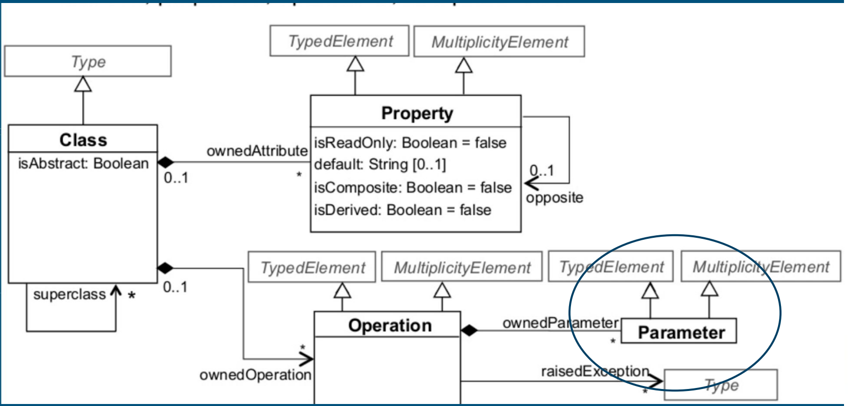
Type
TypedElement
MultiplicityElement
Class
isAbstract: Boolean
Property
isReadOnly: Boolean = false
default: String [0..1]
isComposite: Boolean = false
isDerived: Boolean = false
ownedAttribute
0..1
*
0..1
opposite
superclass
*
0..1
TypedElement
MultiplicityElement
TypedElement
MultiplicityElement
Operation
ownedParameter
Parameter
*
*
ownedOperation
raisedException
*
Type

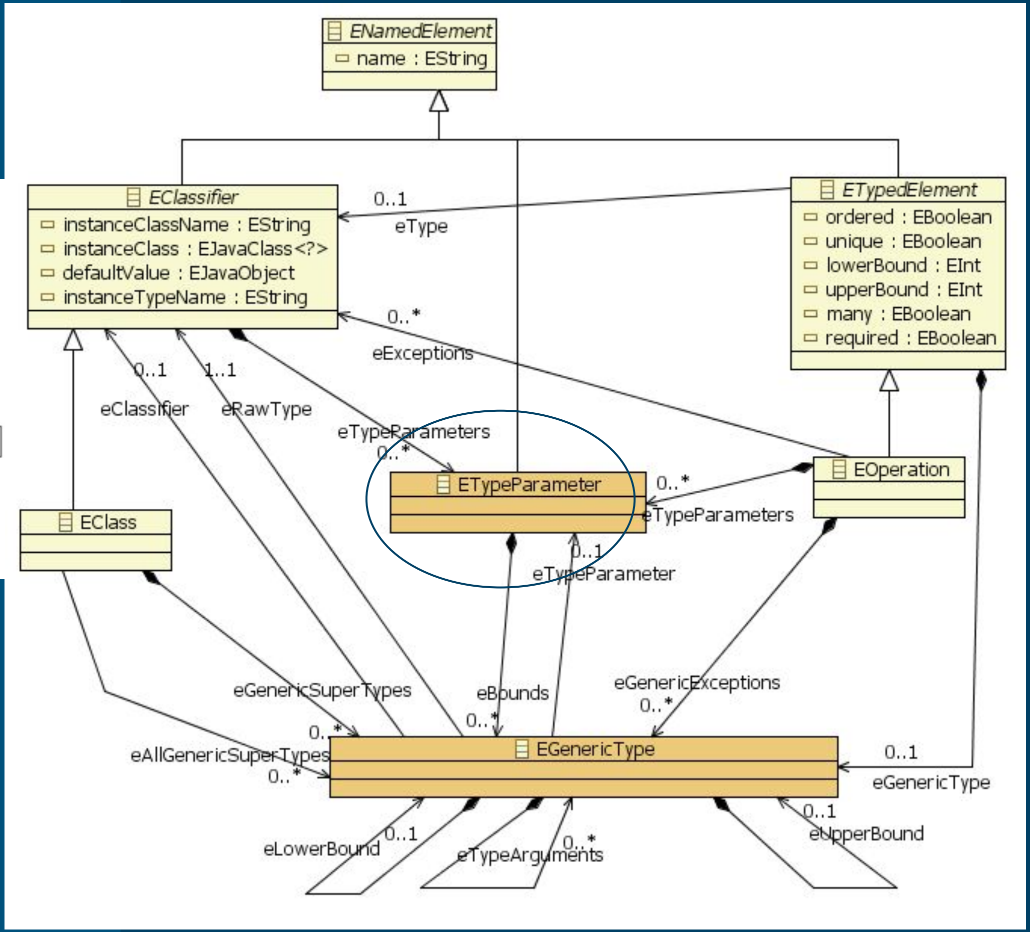


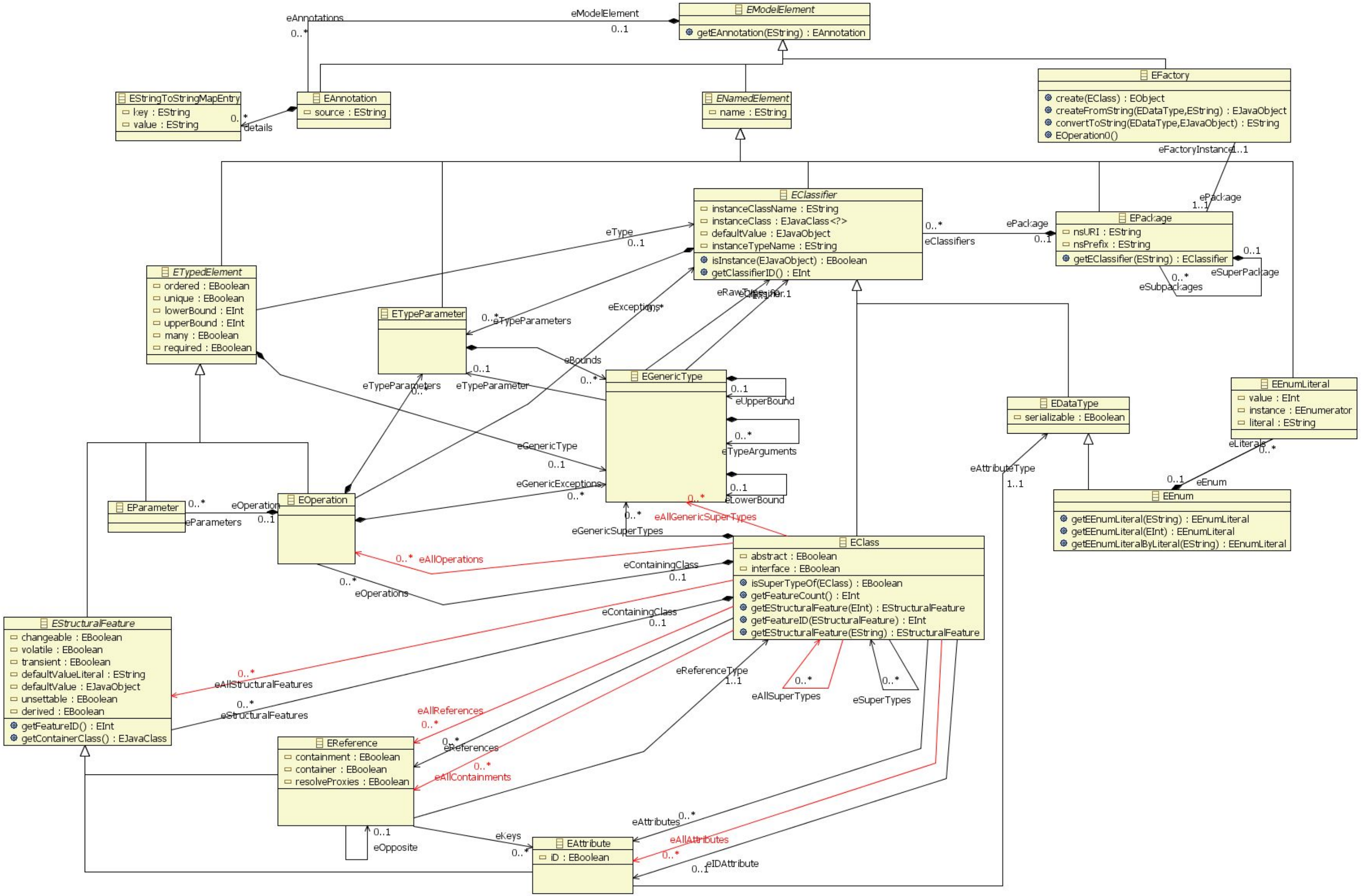
ENamedElement
name : EString
EClassifier
instanceClassName : EString
instanceClass : EJavaClass<?>
defaultValue : EJavaObject
instanceTypeName : EString
ETypedElement
ordered : EBoolean
unique : EBoolean
lowerBound : EInt
upperBound : EInt
many : EBoolean
required : EBoolean
0..1
eType
0..*
eExceptions
0..1
eClassifier
1..1
eRawType
eTypeParameters
0..*
ETypeParameter
EOperation
0..*
eTypeParameters
EClass
0..1
eTypeParameter
eGenericSuperTypes
eBounds
eGenericExceptions
0..*
0..*
0..*
eAllGenericSuperTypes
0..*
EGenericType
0..1
eGenericType
0..1
eLowerBound
0..1
eTypeArguments
0..*
eUpperBound


Type
TypedElement
MultiplicityElement
Property
isReadOnly: Boolean = false
default: String [0..1]
isComposite: Boolean = false
isDerived: Boolean = false
0..1
opposite
Class
isAbstract: Boolean
ownedAttribute
0..1
*
superclass
*
0..1
TypedElement
MultiplicityElement
TypedElement
MultiplicityElement
Operation
ownedParameter
*
Parameter
ownedOperation
*
raisedException
*
Type


ENamedElement
name : EString
EClassifier
instanceClassName : EString
instanceClass : EJavaClass<?>
defaultValue : EJavaObject
instanceTypeName : EString
0..1
eType
ETypedElement
ordered : EBoolean
unique : EBoolean
lowerBound : EInt
upperBound : EInt
many : EBoolean
required : EBoolean
0..*
eExceptions
0..1
1..1
eClassifier
eRawType
eTypeParameters
0..*
EClass
ETypeParameter
0..*
eTypeParameters
EOperation
0..1
eTypeParameter
eGenericSuperTypes
0..*
eBounds
0..*
eGenericExceptions
0..*
eAllGenericSuperTypes
0..*
EGenericType
0..1
eGenericType
eLowerBound
0..1
eTypeArguments
0..*
0..1
eUpperBound


Type
TypedElement
MultiplicityElement
Property
Class
isAbstract: Boolean
ownedAttribute
0..1
*
isReadOnly: Boolean = false
default: String [0..1]
isComposite: Boolean = false
isDerived: Boolean = false
0..1
opposite
superclass
*
0..1
TypedElement
MultiplicityElement
TypedElement
MultiplicityElement
Operation
ownedParameter
*
Parameter
*
ownedOperation
raisedException
*
Type


ENamedElement
name : EString
EClassifier
instanceClassName : EString
instanceClass : EJavaClass<?>
defaultValue : EJavaObject
instanceTypeName : EString
0..1
eType
ETypedElement
ordered : EBoolean
unique : EBoolean
lowerBound : EInt
upperBound : EInt
many : EBoolean
required : EBoolean
0..*
eExceptions
0..1
eClassifier
1..1
eRawType
eTypeParameters
0..*
ETypeParameter
0..*
eTypeParameters
EOperation
EClass
0..1
eTypeParameter
eGenericSuperTypes
eBounds
eGenericExceptions
0..*
0..*
0..*
eAllGenericSuperTypes
0..*
EGenericType
0..1
eGenericType
eLowerBound
0..1
eTypeArguments
0..*
0..1
eUpperBound


EModelElement
getEAnnotation(EString) : EAnnotation
eAnnotations
0..*
eModelElement
0..1
EStringToStringMapEntry
key : EString
value : EString
0..*
details
EAnnotation
source : EString
ENamedElement
name : EString
EFactory
create(EClass) : EObject
createFromString(EDataType,EString) : EJavaObject
convertToString(EDataType,EJavaObject) : EString
EOperation0()
eFactoryInstance
1..1
ePackage
1..1
EClassifier
instanceClassName : EString
instanceClass : EJavaClass<?>
defaultValue : EJavaObject
instanceTypeName : EString
isInstance(EJavaObject) : EBoolean
getClassifierID() : EInt
eType
0..1
0..*
eClassifiers
ePackage
0..1
EPackage
nsURI : EString
nsPrefix : EString
getEClassifier(EString) : EClassifier
0..1
eSuperPackage
0..*
eSubpackages
ETypedElement
ordered : EBoolean
unique : EBoolean
lowerBound : EInt
upperBound : EInt
many : EBoolean
required : EBoolean
ETypeParameter
0..*
eTypeParameters
eExceptions
0..*
eRawType
eClassifier
1..1
eBounds
0..*
0..1
eTypeParameters
0..*
eTypeParameter
EGenericType
0..1
eUpperBound
0..*
eTypeArguments
0..1
eLowerBound
eGenericType
0..1
eGenericExceptions
0..*
EDataType
serializable : EBoolean
EEnumLiteral
value : EInt
instance : EEnumerator
literal : EString
eLiterals
0..*
eAttributeType
1..1
0..1
eEnum
EEnum
getEEnumLiteral(EString) : EEnumLiteral
getEEnumLiteral(EInt) : EEnumLiteral
getEEnumLiteralByLiteral(EString) : EEnumLiteral
EParameter
0..*
eOperation
0..1
eParameters
EOperation
0..*
eAllGenericSuperTypes
0..*
eGenericSuperTypes
EClass
abstract : EBoolean
interface : EBoolean
isSuperTypeOf(EClass) : EBoolean
getFeatureCount() : EInt
getEStructuralFeature(EInt) : EStructuralFeature
getFeatureID(EStructuralFeature) : EInt
getEStructuralFeature(EString) : EStructuralFeature
0..* eAllOperations
eContainingClass
0..1
0..*
eOperations
eContainingClass
0..1
EStructuralFeature
changeable : EBoolean
volatile : EBoolean
transient : EBoolean
defaultValueLiteral : EString
defaultValue : EJavaObject
unsettable : EBoolean
derived : EBoolean
getFeatureID() : EInt
getContainerClass() : EJavaClass
0..*
eAllStructuralFeatures
0..*
eStructuralFeatures
eReferenceType
1..1
0..*
eAllSuperTypes
0..*
eSuperTypes
eAllReferences
0..*
0..*
eReferences
0..*
eAllContainments
EReference
containment : EBoolean
container : EBoolean
resolveProxies : EBoolean
0..1
eOpposite
eKeys
0..*
EAttribute
iD : EBoolean
eAttributes
0..*
eAllAttributes
0..*
0..1
eIDAttribute

Eclipse

- Eclipse Modeling Framework
- Full support for metamodeling and language design
- Fully MD (vs. programming-based tools)
- Used in this course!

emf

ECLIPSE MODELING FRAMEWORK

EMF

EMF at Eclipse.org

- Foundation for the Eclipse Modeling Project
  - EMF project incorporates core and additional mature components: Query, Transaction, Validation
  - EMF Technology project incubates complementary components: CDO, Teneo, Compare, Search, Temporality, Ecore Tools…
  - Other projects build on EMF: Graphical Modeling Framework (GMF), Model Development Tools (MDT), Model to Model Transformation (M2M), Model to Text Transformation (M2T)…
- Other uses: Web Tools Platform (WTP), Data Tools Platform (DTP), Business Intelligence and Reporting Tools (BIRT), SOA Tools Platform (STP)…
- Large open source user community

EMF

- EMF models can be defined in (at least) three ways:
  1. Java Interfaces
  2. UML Class Diagram
  3. XML Schema
- Choose the one matching your perspective or skills and EMF can create the others, as well as the implementation code

EMF

- EMF models can be defined in (at least) three ways:
  1. Java Interfaces
  2. UML Class Diagram
  3. XML Schema
- Choose the one matching your perspective or skills and EMF can create the others, as well as the implementation code

EMF

Three Ecore Model Perspectives: Java API

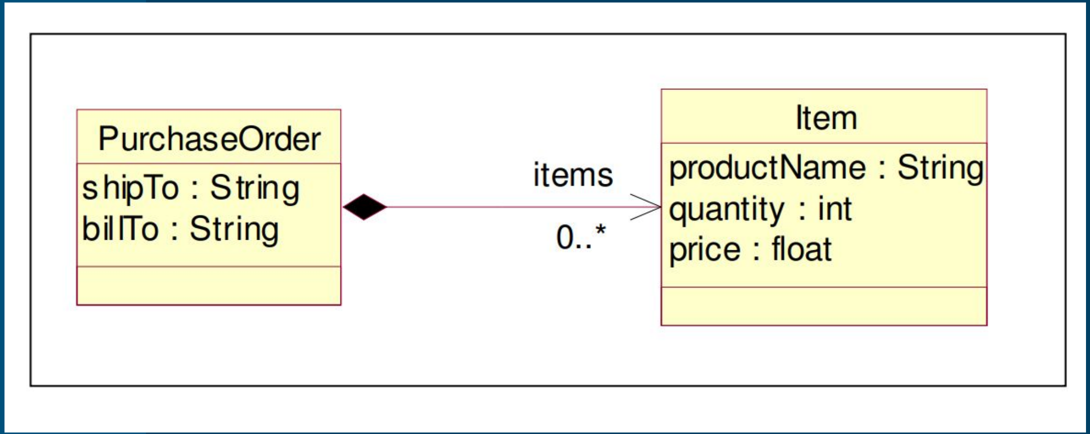
Java Interfaces

```
public interface PurchaseOrder
{
  String getShipTo();
  void setShipTo(String value);
  String getBillTo();
  void setBillTo(String value);
  List<Item> getItems();  // containment
}
```

```
public interface Item
{
  String getProductName();
  void setProductName(String value);
  int getQuantity();
  void setQuantity(int value)
  float getPrice();
  void setPrice(float value);
}
```

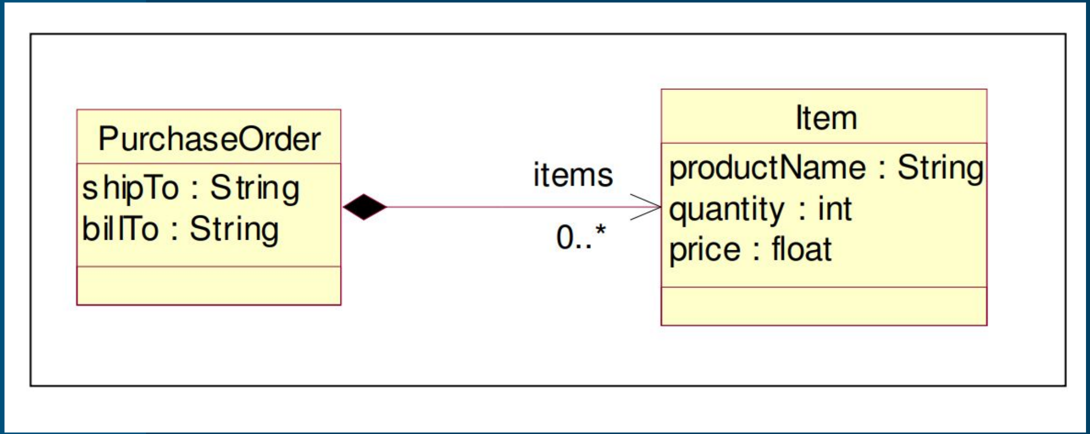
EMF

Three Ecore Model Perspectives: Diagram



EMF

Three Ecore Model Perspectives: Diagram

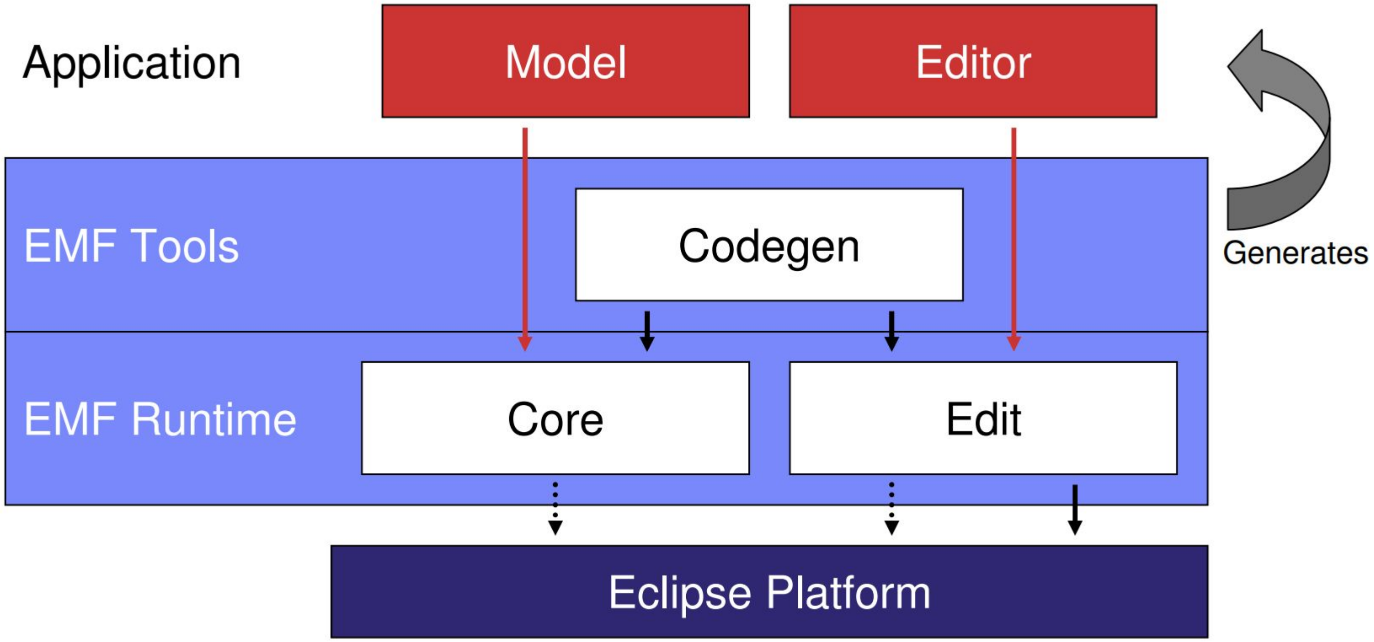


EMF

Three Ecore Model Perspectives: XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.example.com/SimplePO"
            xmlns:po="http://www.example.com/SimplePO">
  <xsd:complexType name="PurchaseOrder">
    <xsd:sequence>
      <xsd:element name="shipTo" type="xsd:string"/>
      <xsd:element name="billTo" type="xsd:string"/>
      <xsd:element name="items"  type="po:Item"
                   minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Item">
    <xsd:sequence>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="quantity" type="xsd:int"/>
      <xsd:element name="price" type="xsd:float"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

EMF Architecture



EMF Components

- Core Runtime
  - Notification framework
  - Ecore metamodel
  - Persistence (XML/XMI), validation, change model
- EMF.Edit
  - Support for model-based editors and viewers
  - Default reflective editor
- Codegen
  - Code generator for application models and editors
  - Extensible model importer/exporter framework

Ecore

- Persistent format is XMI (.ecore file)

```
<eClassifiers xsi:type="ecore:EClass"
  name="PurchaseOrder">
 <eStructuralFeatures xsi:type="ecore:EReference"
   name="items" eType="#//Item"
   upperBound="-1" containment="true"/>
 <eStructuralFeatures xsi:type="ecore:EAttribute"
   name="shipTo"
   eType="ecore:EDataType http:...Ecore#//EString"/>
 ...
</eClassifiers>
```

- Alternate format is Essential MOF XMI (.emof file)

Thank you!

Contact: vma@fct.unl.pt