Domain-specific Language engineering

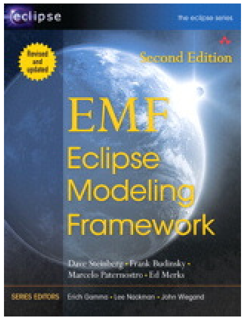
PATRIZIA SCANDURRA – LINGUAGGI E COMPILATORI 2016-17

Outline

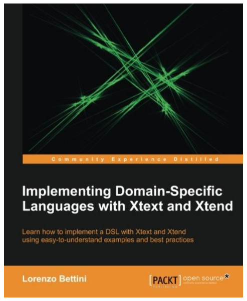
- Introduction on Domain-Specific Languages (DSLs) -- *the 4th generation languages*
- Grammarware vs modelware approaches
- The Eclipse Modeling Framework (EMF) – Modelware approach
- The Xtext framework – Grammarware approach

Books and references

- Slides, documents and examples from my Dropbox repository https://www.dropbox.com/sh/triz42yrp9eb7u1/AADDxORqn0YzN0CYdpqdE9Hza?dl=0


eclipse
the eclipse series
Revised and updated
Second Edition
EMF
Eclipse Modeling Framework
Dave Steinberg • Frank Budinsky • Marcelo Paternostro • Ed Merks
SERIES EDITORS Erich Gamma • Lee Nackman • John Wiegand

- EMF: Eclipse Modeling Framework, 2nd Edition. By Dave Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks. Published Dec 16, 2008 by Addison-Wesley Professional. Part of the Eclipse Series series.


Community Experience Distilled
Implementing Domain-Specific Languages with Xtext and Xtend
Learn how to implement a DSL with Xtext and Xtend using easy-to-understand examples and best practices
Lorenzo Bettini
PACKT open source

- Implementing Domain-Specific Languages with Xtext and Xtend Paperback – August 21, 2013 by Lorenzo Bettini

Introduction

PATRIZIA SCANDURRA – LINGUAGGI E COMPILATORI 2016-17

Domain-specific Languages (DSLs)

- Programming languages or modeling languages that target a specific problem domain
- The goal is to automate development of software applications in a given domain, solving that problem easier and faster by using that **DSL instead of a general purpose language** like Java or C
- A program or model (or ***mogram***) written in a DSL describes essential characteristics of an application
- A mogram can then be interpreted or compiled into a general purpose language
- In other cases, the mogram can represent simply data that will be processed by other systems

Examples of DSLs

- **Main idea**:
  - Use mograms to describe essential characteristics of an application, and
  - use code generation (model compilers/interpreters) to produce the application automatically

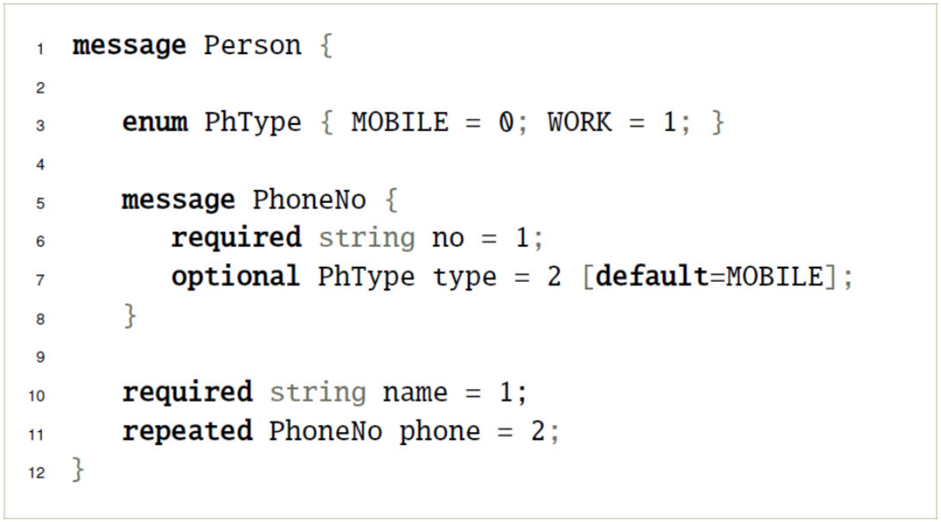
- *Examples of DLSs*:
  - SQL (for querying relational databases)
  - Mathematica (for symbolic mathematics)
  - HTML, and many others

Another DSL example

Google Protocol Buffers data modeling language aimed at flexible and efficient serialization and persistence of structured data across multiple programming languages and platforms

It uses message as a metaphor for a data structure

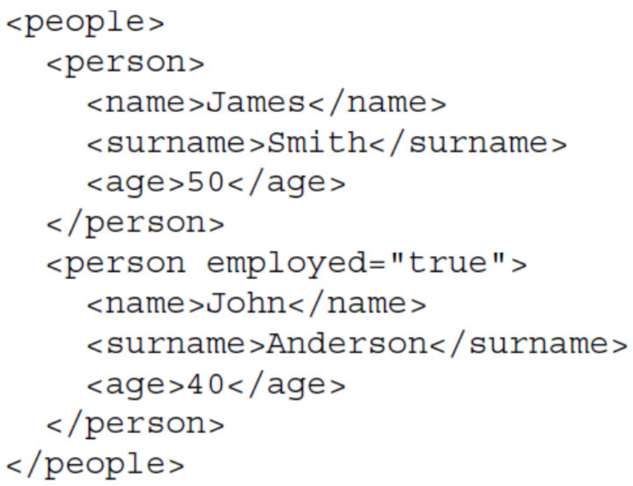
An example of model in the Google Protocol Buffers DSL



```
message Person {

   enum PhType { MOBILE = 0; WORK = 1; }

   message PhoneNo {
      required string no = 1;
      optional PhType type = 2 [default=MOBILE];
   }

   required string name = 1;
   repeated PhoneNo phone = 2;
}
```

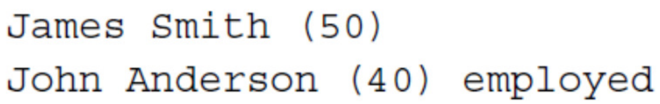
DSLs versus XML

- Also XML allows you to describe data in a machine- and human- readable form
  - But many people find that XML is machine-readable, but not so much human-readable
- Consider a very simple example of an XML file describing people:



```
<people>
  <person>
    <name>James</name>
    <surname>Smith</surname>
    <age>50</age>
  </person>
  <person employed="true">
    <name>John</name>
    <surname>Anderson</surname>
    <age>40</age>
  </person>
</people>
```

XML



```
James Smith (50)
John Anderson (40) employed
```

DSL for people

DSLs engineering

What does it take to design and implement a language?

1. Concrete syntax (textual, graphical, mixed)

2. Abstract syntax

3. Editing/Modeling environment for models

4. Serialization/Deserialization for models

5. Static semantics (mostly type checking)

6. Dynamic semantics (interpreter, model compiler, code generator)

Language engineering approaches

- **Grammaware**: *grammar-dependent* software language engineering
  - "Grammarware comprises grammars and all grammar-dependent software." [Klint, Lämmel and Verhoef, 2005]
- **Modelware**: *model-based* software language engineering
  - Modelware concerns the building of models and associated modeling tools for software systems (e.g., UML)
  - Useful for **graphical DSLs**!

Language engineering frameworks

- Frameworks are necessary for **implementing a DSL together with its IDE functionalities**
- **Common IDE functionalities**
  - syntax highlighting, auto-completion, background parsing, errors markers, suggested quickfixes, automatic build, outline, etc.
- We will assume <u>Eclipse as the underlying IDE</u>

Language engineering: frameworks for textual DSLs

- **Xtext** (approach *grammarware*): open source Eclipse framework
http://www.eclipse.org/Xtext/

- **EMFtext** (approach *modelware*): open source Eclipse plug-in for defining textual DSLs starting by an **EMF Ecore model** (or **metamodel**)
http://www.emftext.org/index.php/EMFText
  - **EMF (Eclipse Modeling Framework) toolkit as standard *de facto* for modelware**
  - with modeling and code generation facility for building tools and other applications based on a structured data model Ecore (the metamodel) of the DSL

Both adopt **ANTLR in the background**! They use the *LL(*)* parser generator of ANTLR, allowing to cover a wide range of syntaxes

Modelware versus Grammarware

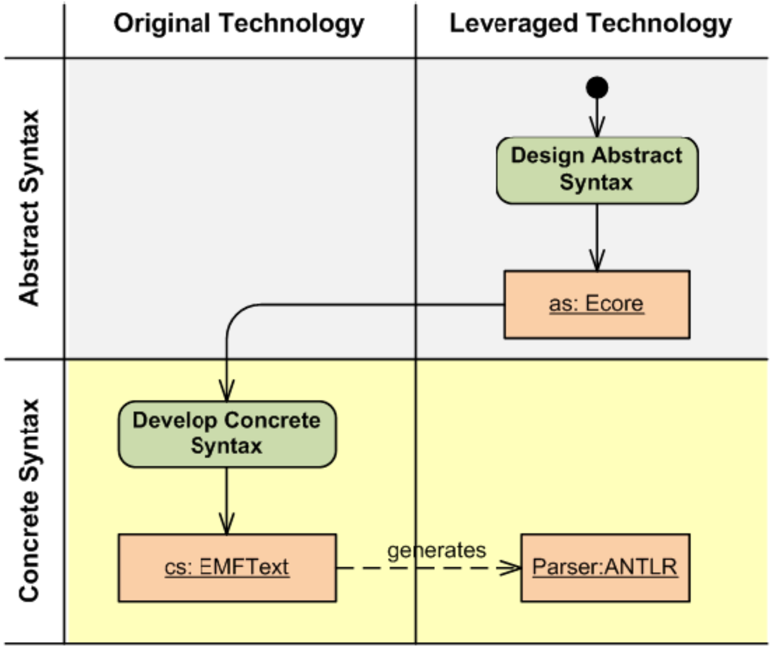
EMFText was built around EMF Ecore models and generate a language to parse these models. You have to define an Ecore model (**abstract syntax** or **metamodel**) first to start creating a DSL

Xtext instead focus on describing a textual **concrete syntax** for a language and derives everything from the syntax itself, including an EMF Ecore model that represents the AST (Abstract Syntax Tree)

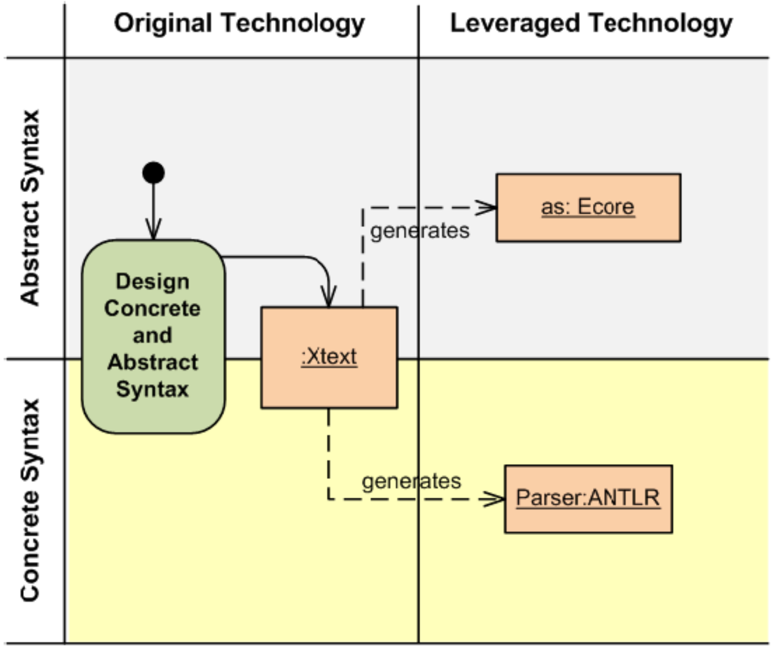
Both tools offer more or less the same functionality and are well integrated into the Eclipse platform, but **each tool has its own workflow** and has to be integrated into the development process differently

Modelware versus Grammarware

A metamodel (abstract syntax) allows **separating syntax design from concept selection** (good thing)



EMFText workflow to create a new DSL



Xtext Workflow to create a new DSL

DSL Design Guidelines

1 Visual vs Textual: subject matter experts vs programmers

2 Do not ignore the users! They need to understand your syntax. Adopt existing notations, which are already understandable

3 Do not introduce new symbols needlessly; omit concepts not contributing to the well defined purpose of the language (KISS)

4 Do not overload symbols with established semantics (use descriptive keywords)

5 Allow comments in your language

6 Keep abstract and concrete syntax close

7 Compose existing languages (REUSE principle!)

- Xtext provides Xbase and terminal grammars for this purpose

DSL engineering with a modelware approach

PATRIZIA SCANDURRA – LINGUAGGI E COMPILATORI 2016-17

Topics

- EMF overview http://www.eclipse.org/modeling/emf/
  - Metamodeling with Ecore: Specifying a DSL metamodel
  - Deriving languages SW artifacts (generate language tooling)

- EMFtext (not in program!)

http://www.emftext.org/EMFTextGuide.php

Modelware steps for developing a DSL

(1) Specifying a Language Metamodel

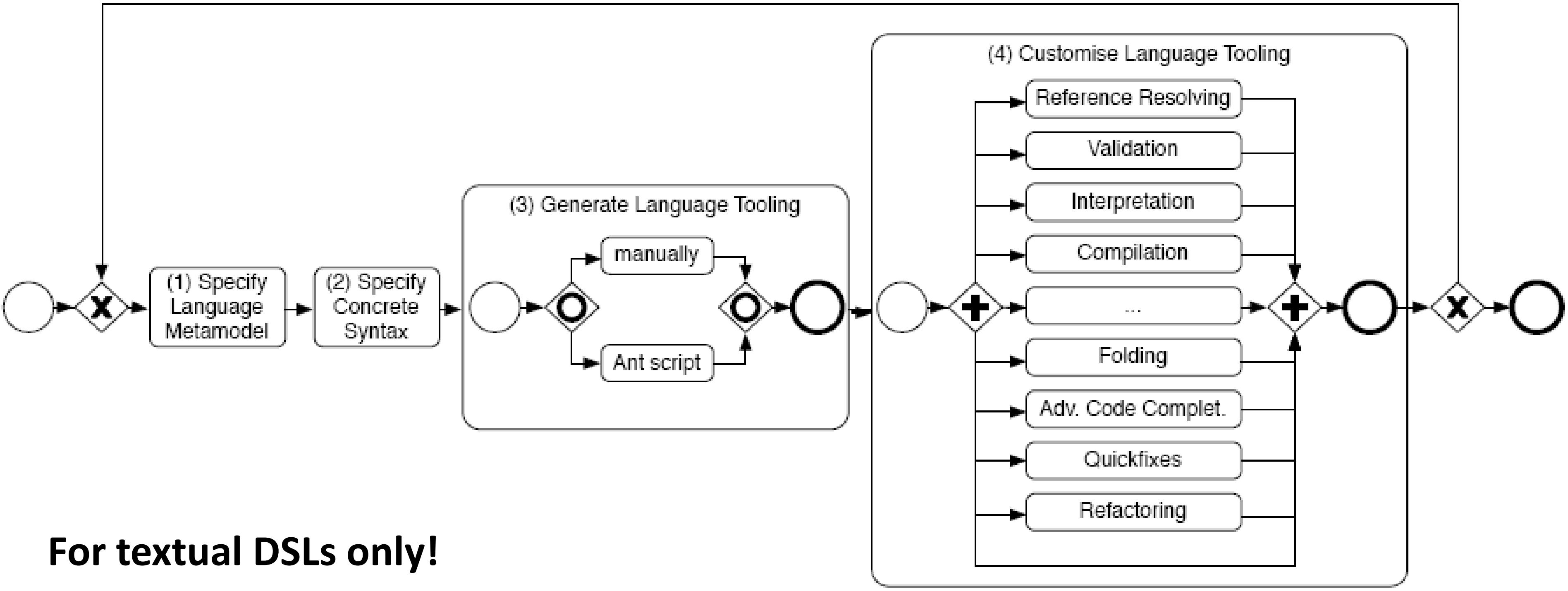
(2) Specifying the Language’s Concrete Syntax (textual or visual or mixed)

(3) Generating the Language Tooling

(4) Optionally Customising the Language Tooling

These steps have to be carried out by a concrete DSL engineering framework, like EMFText

Iterative EMFText language development workflow



For textual DSLs only!

Metamodeling

In order to be able to process languages automatically we need to provide formal definitions of them

Metamodeling *is the practice of modeling (abstract) syntax of languages using UML-like class diagrams*

*So metamodeling is modeling of languages, and **a metamodel is a model of a language***

Model Driven Engineering (MDE)

Engineering approach for software/system development and analysis where **models** play the role of first-class artifacts

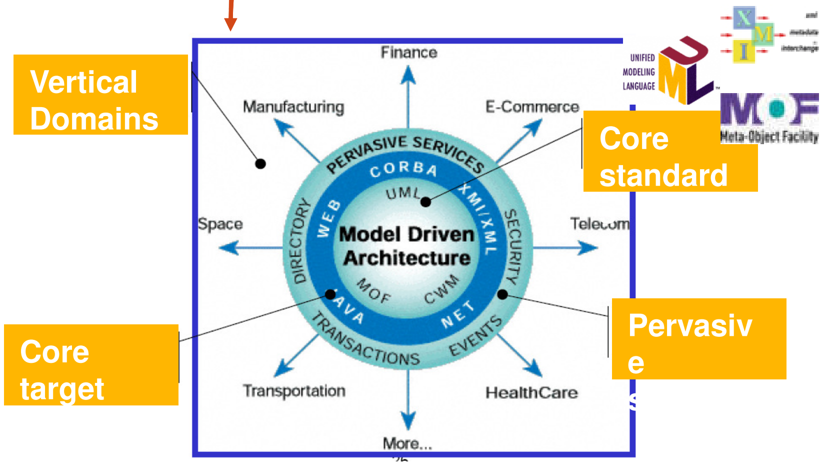
- Beyond their use as documentation
- models can be used to **generate** software artifacts

All Software Engineering **generative techniques** based on the notions of

- **models**, **metamodeling**, and **model transformation**

MDE - standards & tools

Principles	**Model-driven engineering (MDE)**			
Standards	OMG's MDA	MIC	Software Factories	...
Tools	Eclipse EMF, MDR/Netbeans AMMA, etc..	GME	Microsoft support for DSLs	...



Three ideas:

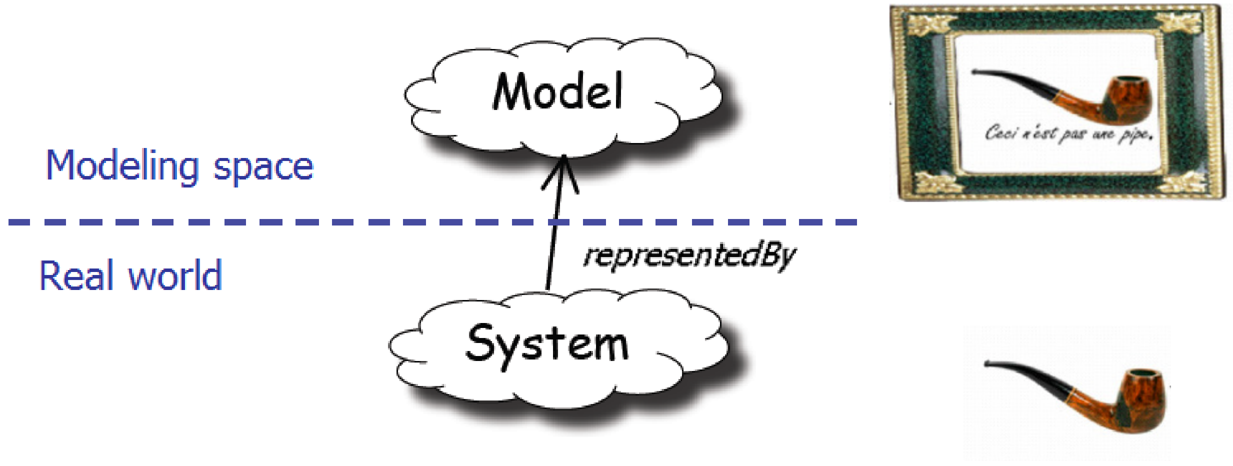
direct representation,

automation, and

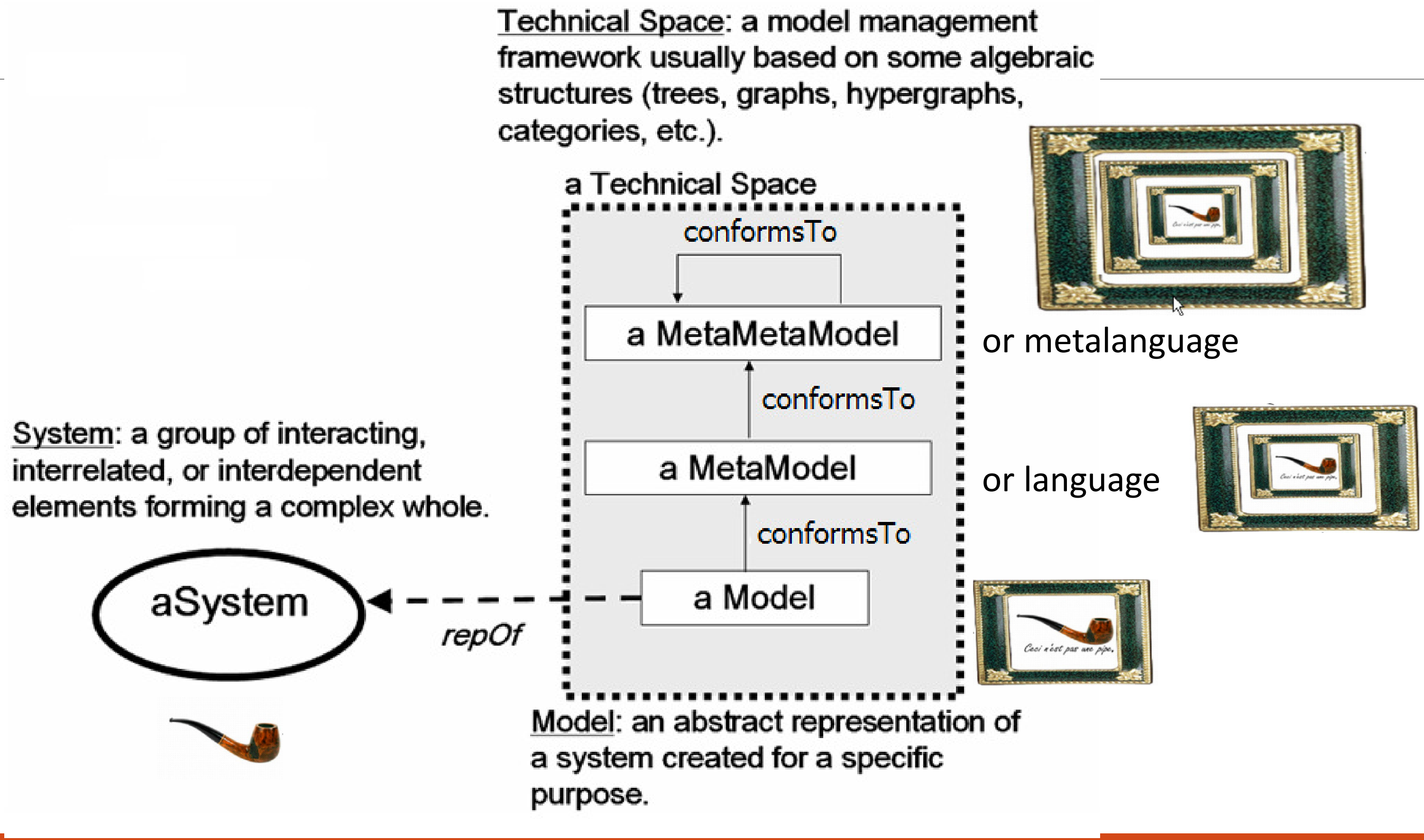
standards
(languages and mappings)

MDE principles: first principle

Unification principle: “Everything is a model”



MDE principles:
the three-level metamodeling stack

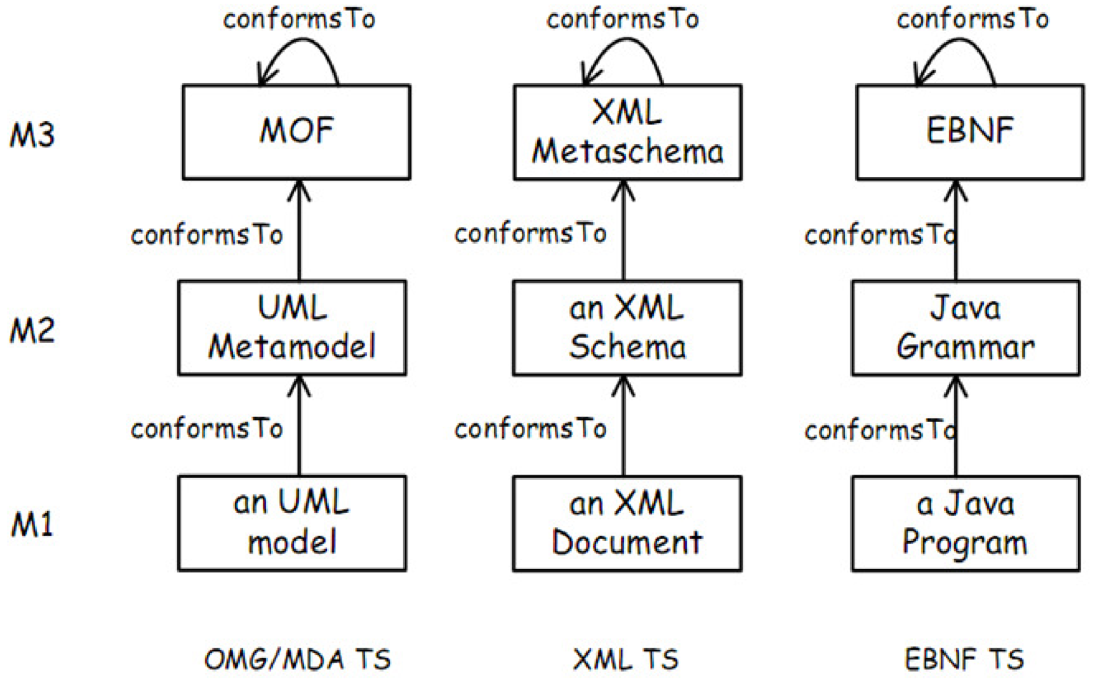


Technical Space: a model management framework usually based on some algebraic structures (trees, graphs, hypergraphs, categories, etc.).

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

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

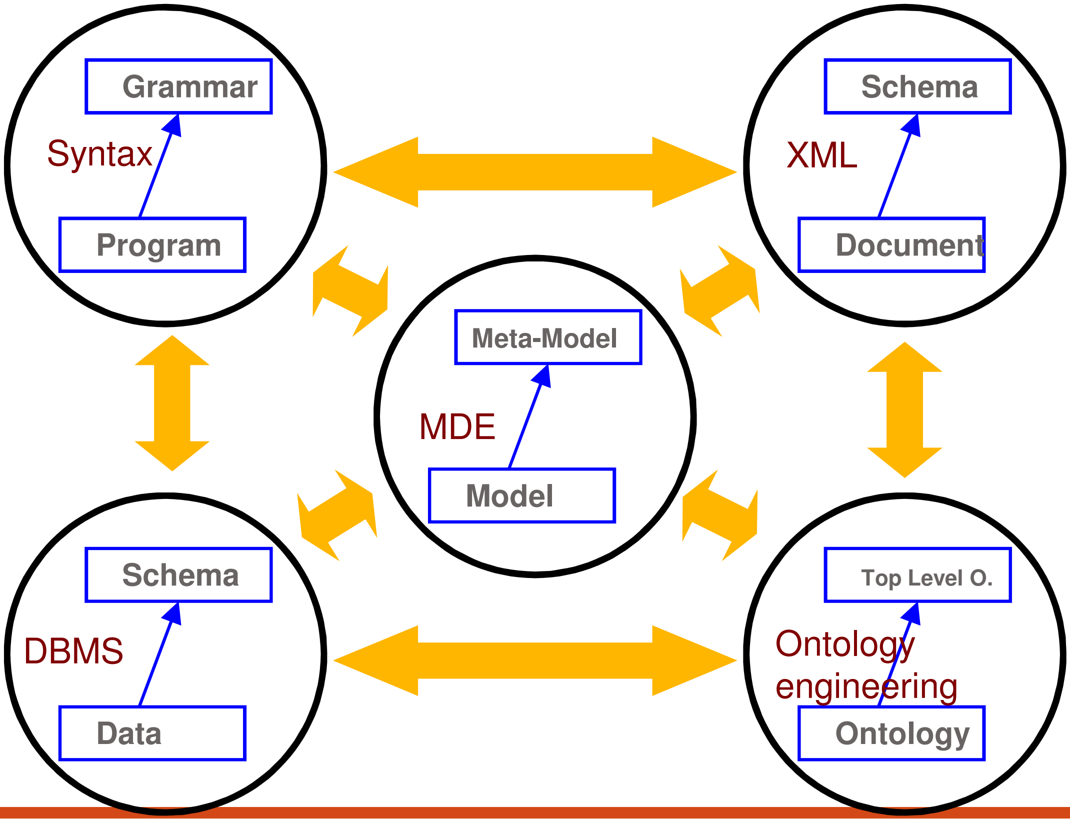
The three-level model organization stack in various technical spaces



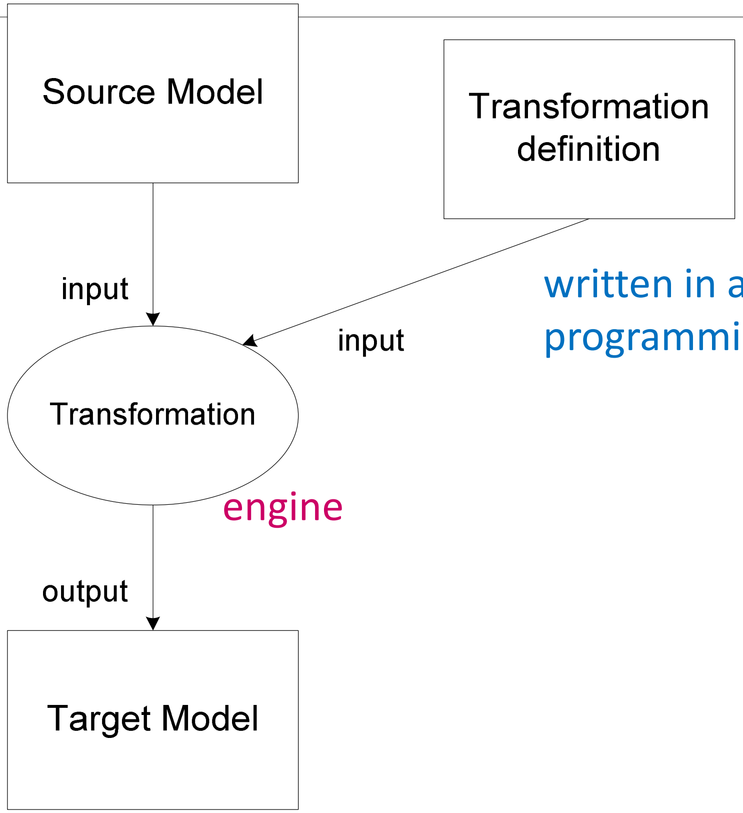
Usually, every TS provides tools that check for the presence of the conformance relation

Model transformation and Technical Spaces

- **Technical Spaces** are similarly organized around a set of concepts
- Spaces may be connected via ***transformation bridges*** (model compilers!)



Common Transformation Pattern

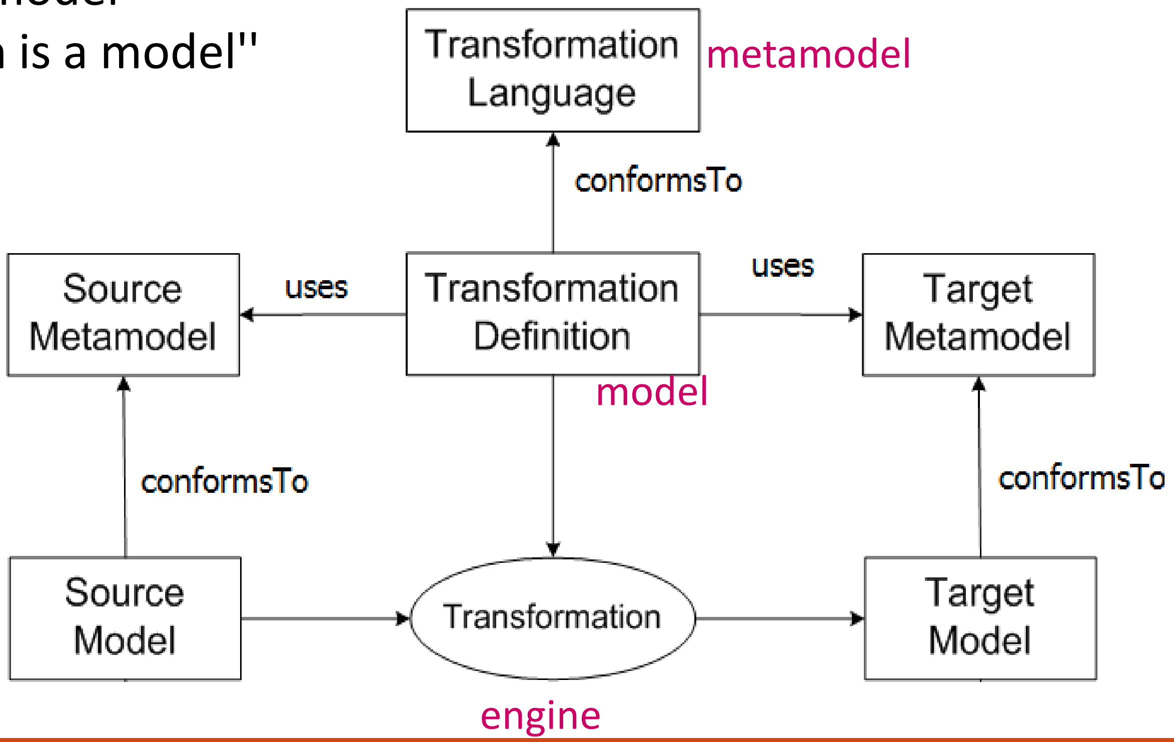


written in a general-purpose programming language

engine

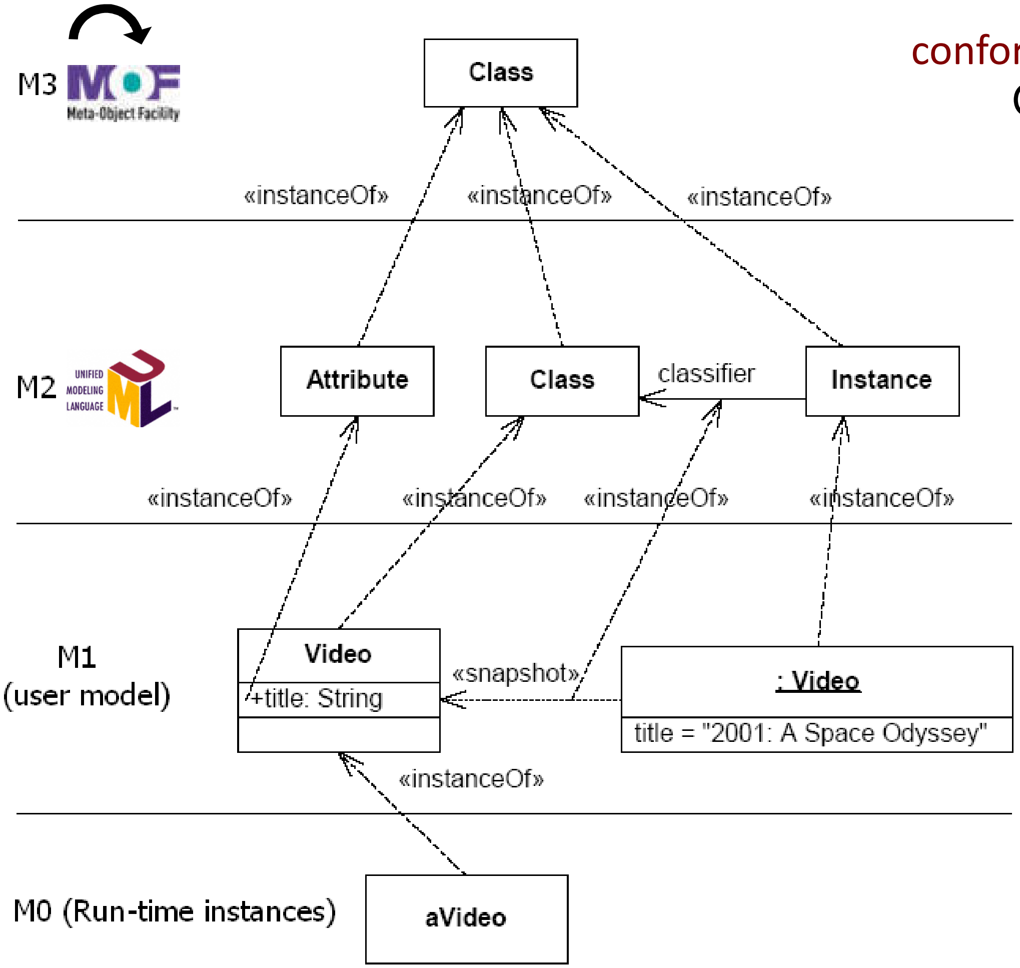
MDE Transformation Pattern

Corollary: ``A model transformation is a model''



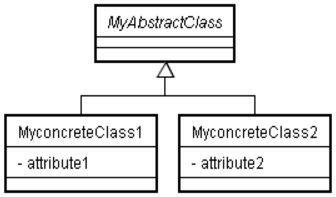
The OMG’s Metamodeling Framework

conformsTo Ξ instanceOf
OT-based!



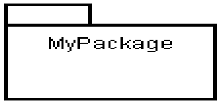
The MOF meta-language

- **Classes** with attributes and operations, possibly inherited from other classes by **Generalization**



- **Associations** (simple, composite) between classes, with cardinality and uniqueness

- **Packages** to group elements for modularizing



- **Data types** whose values do not have object identity
  - ⇨ primitive types: Boolean, Integer, and String
  - ⇨ data type constructors: Enumeration, Collection, etc.
- **Constraints** (*well-formedness rules)* in the ***Object Constraint Language*** (OCL)

  a mix of Predicate Logic and set theory

- **APIs for model manipulation/implementation**
  - ⇨ Java Interfaces
  - ⇨ CMI (CORBA Metadata Interface), etc.
- **Model serialization**
  - ⇨ XMI (XML Metadata Interchange), HUTN (Human Usable Textual Notation), etc.

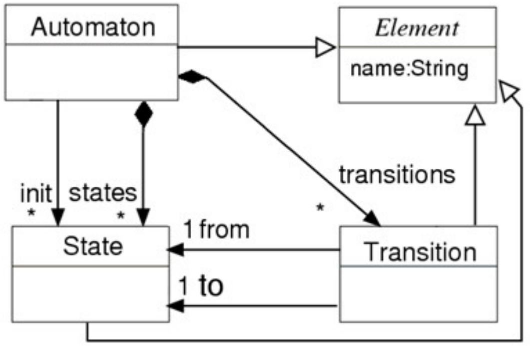
Standard MOF Projections to handle models

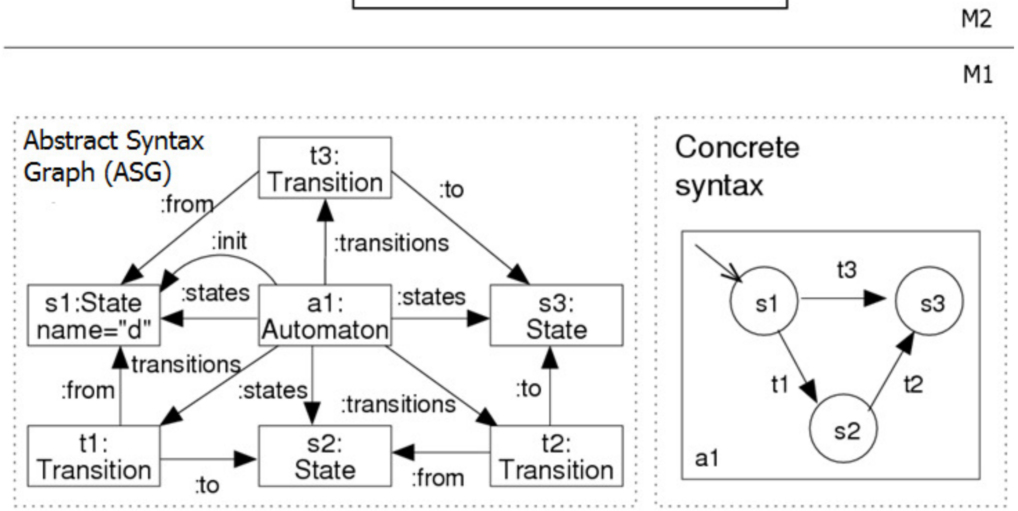
The MOF meta-language: a metamodel example

The “automaton” metamodel

Abstract Syntax (AS)
expressed by a metamodel

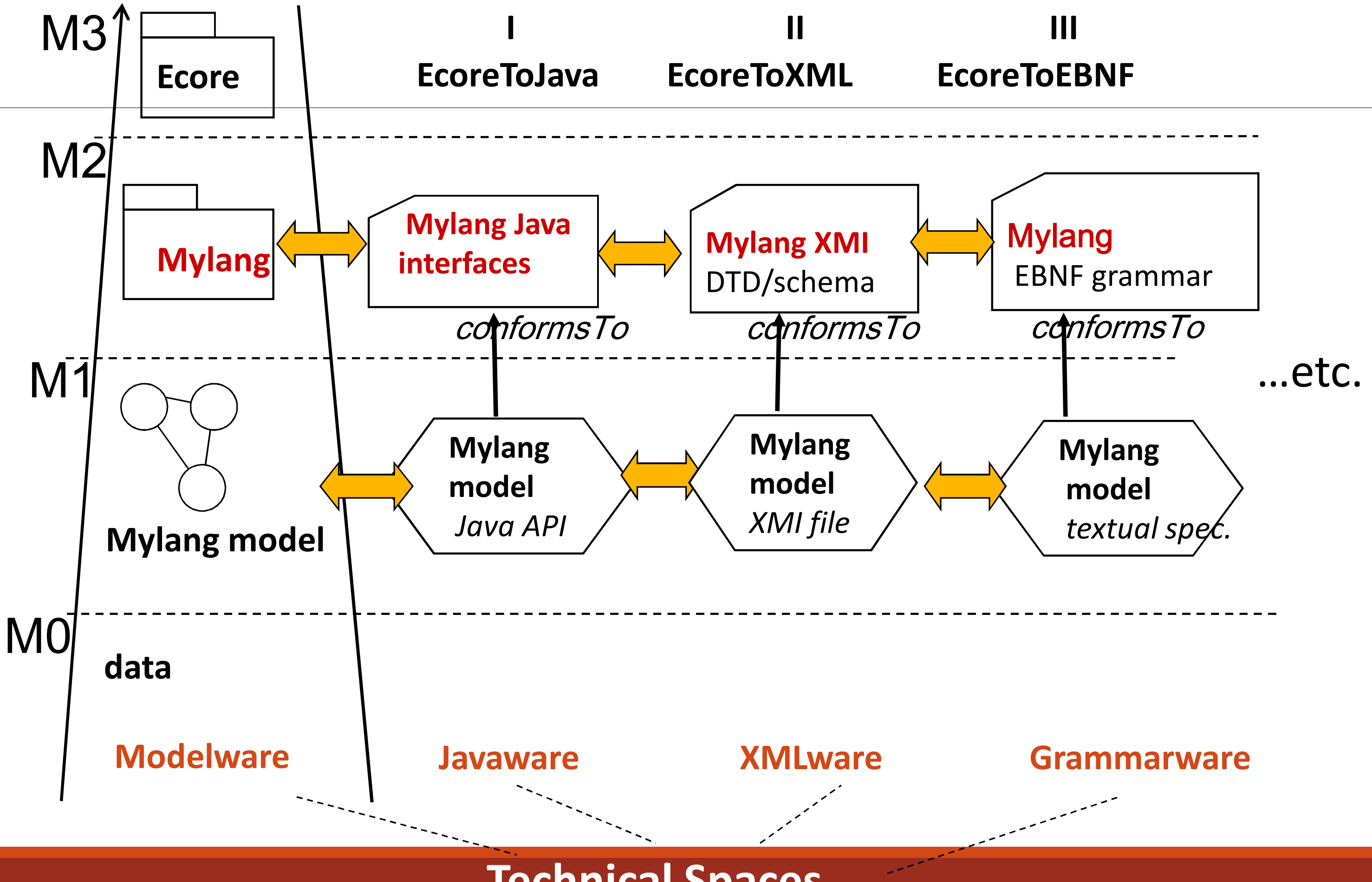
"metalevel" boundary





EMF: a modelig framework for MDE

EMF framework (with Ecore the metalanguage as MOF)



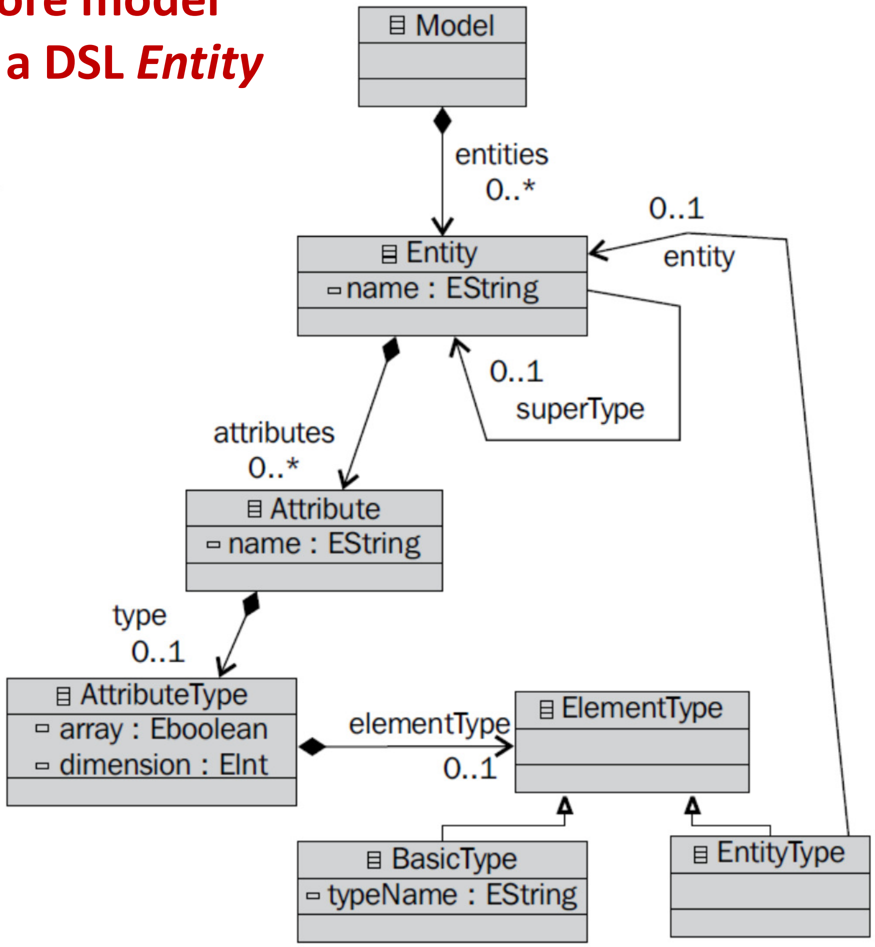
Technical Spaces

EMF Ecore

Ecore is a meta-language to define the abstract syntax (metamodel) of a DSL in terms of **an object model**

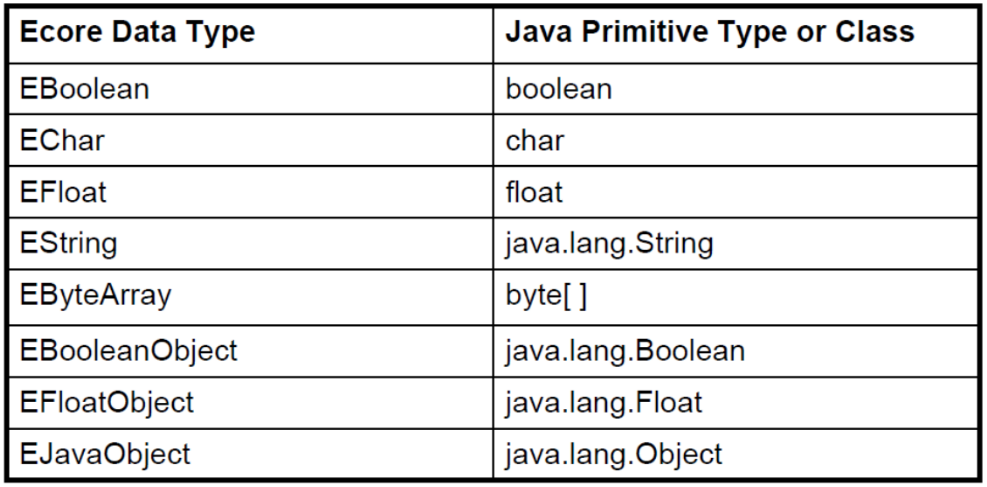
- **EClass**: represents a class, with zero or more attributes and zero or more references
- **EAttribute**: represents an attribute which has a name and a type
- **EReference**: represents one end of an association between two classes. It has flags to indicate if it represents a containment and a reference class to which it points
- **EDataType**: represents the type of an attribute, e.g., int, float, Estring, etc.

Ecore model of a DSL *Entity*



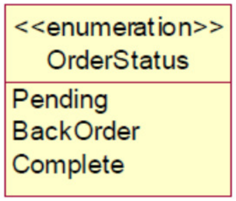
Partial list of Ecore datatypes

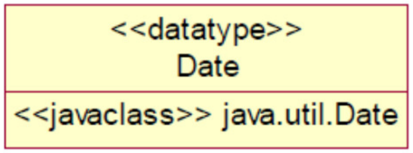
Ecore datatypes are serializable



Ecore Data Type	Java Primitive Type or Class
EBoolean	boolean
EChar	char
EFloat	float
EString	java.lang.String
EByteArray	byte[ ]
EBooleanObject	java.lang.Boolean
EFloatObject	java.lang.Float
EJavaObject	java.lang.Object

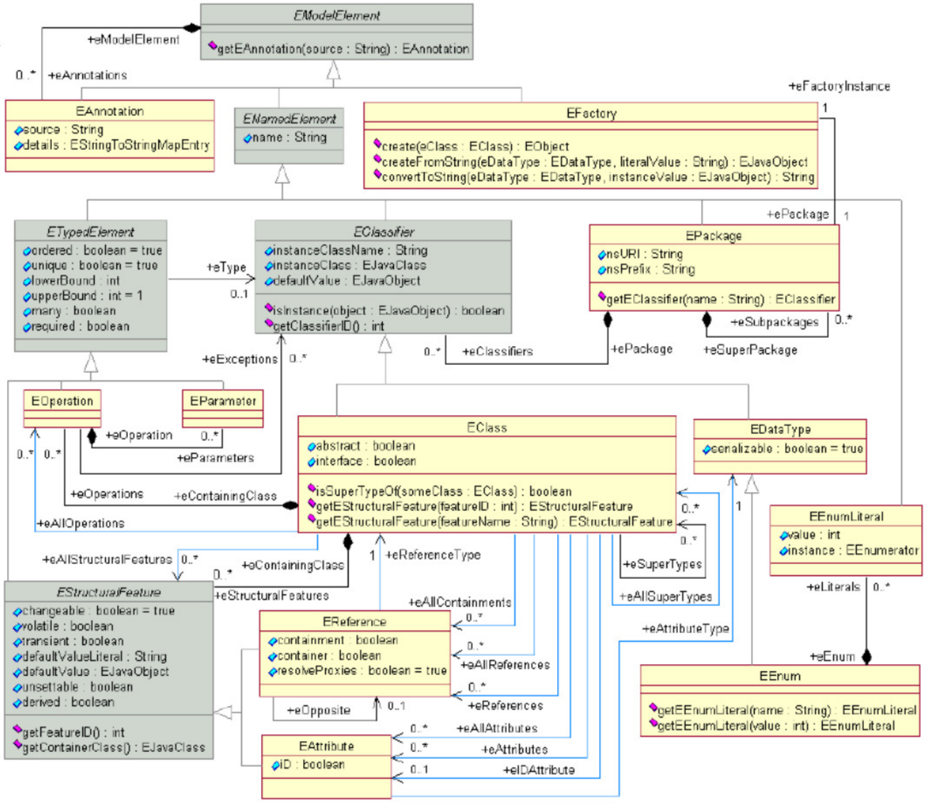
Support for custom datatypes



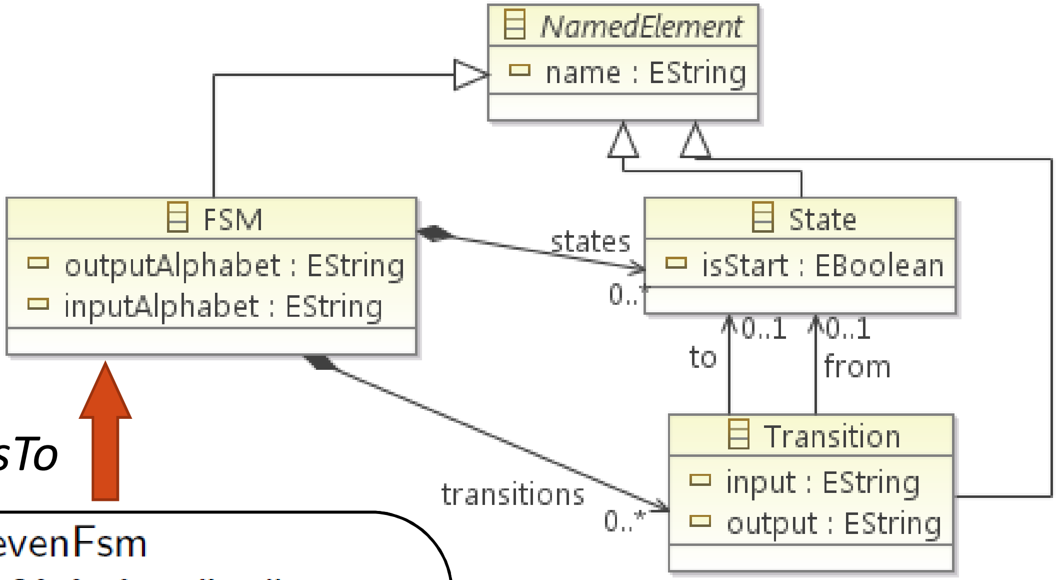


http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

An ecore model of Ecore

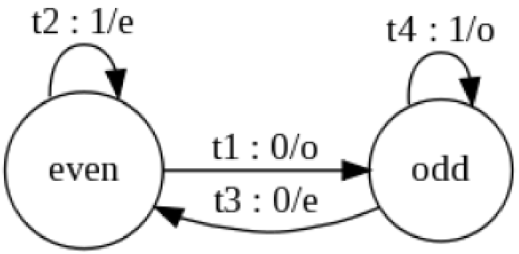


Example: The FSM metamodel (abstract syntax)



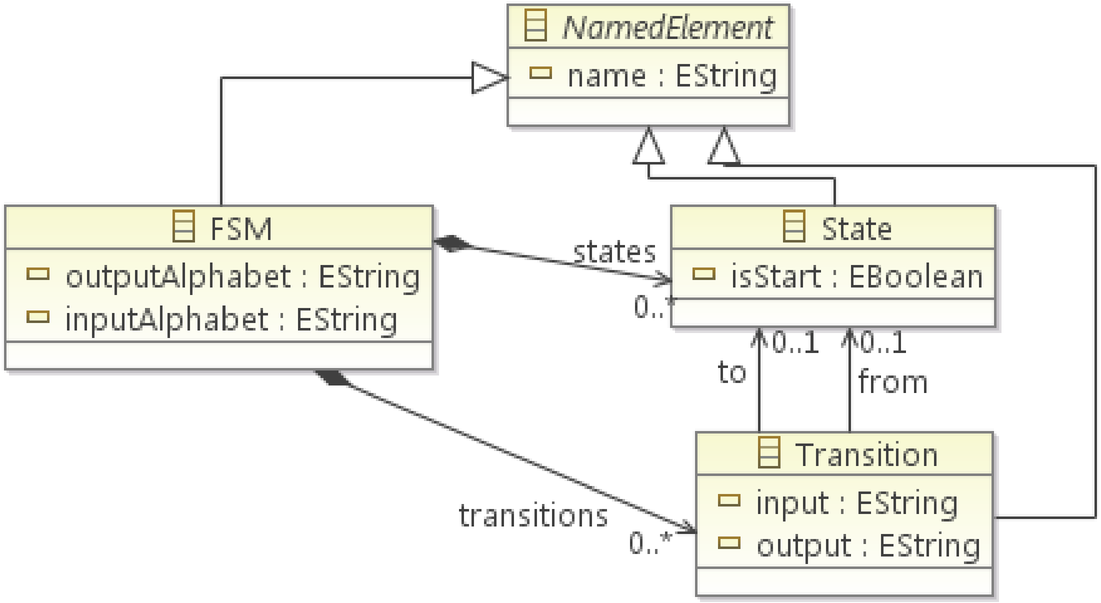
conformsTo

fsm evenFsm
inputAlphabet "01"
outputAlphabet "eo"



an FSM model

The FSM metamodel (abstract syntax)



Example of OCL constraint for *well-formedness of models:*

```
context  FSM
-- The following invariant checks that if there are no
states, there are no transitions
inv I0: self.states->isEmpty() implies
        self.transitions->isEmpty()
```

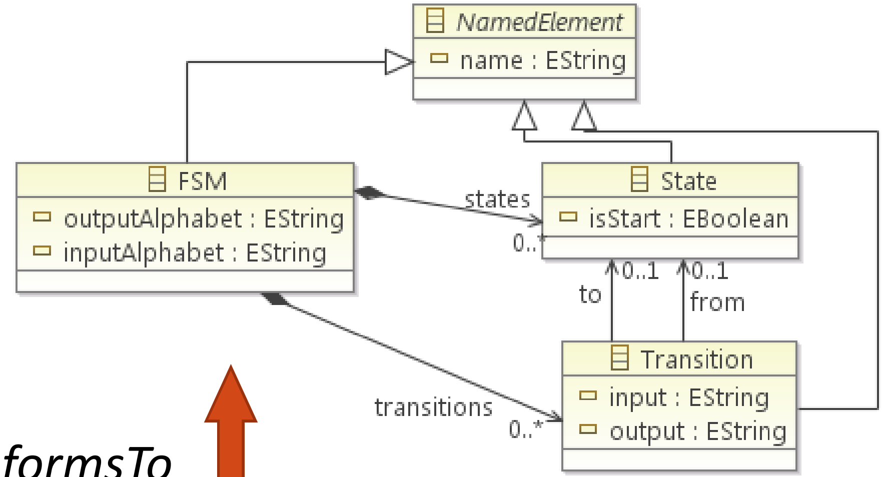
Concrete syntaxes (or notations) for FSM

They can be: textual, graphical or both

Moreover they can be:

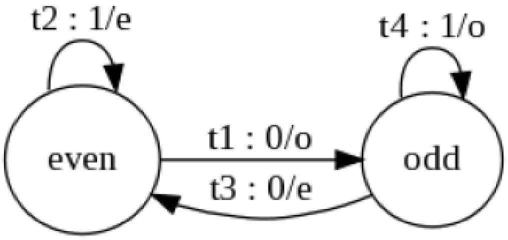
- Human-comprehensible for human use to edit models conforming to the metamodel, and as
- Machine-comprehensible for model handling by software applications
  - Examples: an XMI (XML Metadata Interchange) format and Java APIs

Human-comprehensible notations for FSM



conformsTo

fsm evenFsm
inputAlphabet "01"
outputAlphabet "eo"

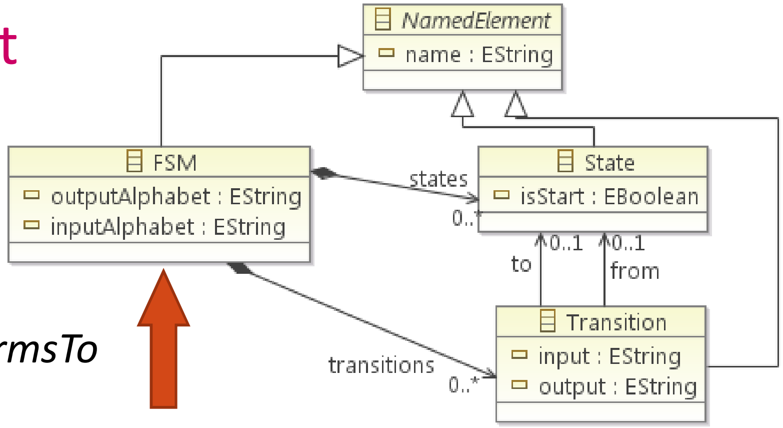


conformsTo

```
fsm evenFsm
inputAlphabet "01"
// Char 'e' for even and 'o' for odd
outputAlphabet "eo"
states
start even odd
transitions
t1: even – 0 / o –> odd ;
t2: even – 1 / e –> even ;
t3: odd – 0 / e –> even ;
t4: odd – 1 / o –> odd ;
```

Machine-comprehensible notations for FSM

XMI format

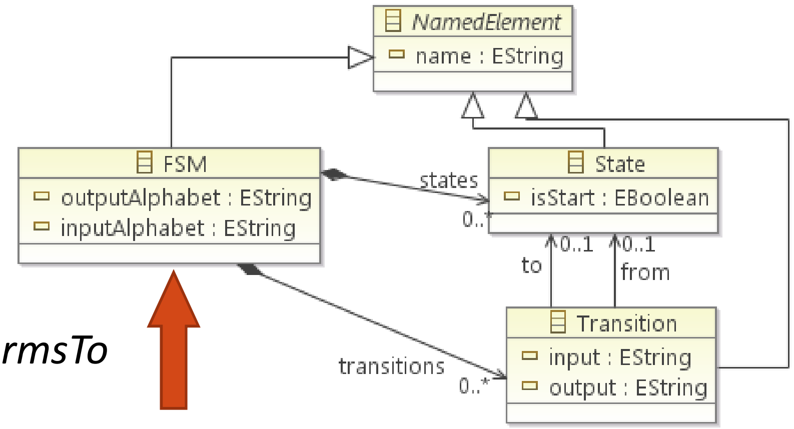


conformsTo

```
<?xml version="1.0" encoding="UTF-8"?>
<fsm:FSM xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
               xmlns:fsm="http://www.openarchitectureware.org/xtext/dsl/fsm"
               name="evenFsm" outputAlphabet="eo" inputAlphabet="01">
  <states name="even" isStart="true"/>
  <states name="odd" isStart="true"/>
  <transitions name="t1" input="0" to="//@states.1" from="//@states.0"
                    output="o"/>
  <transitions name="t2" input="1" to="//@states.0" from="//@states.0"
                    output="e"/>
…
</fsm:FSM>
```

Machine-comprehensible notations for FSM

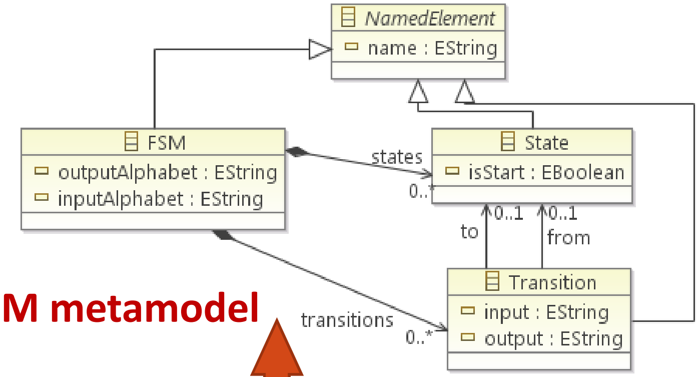
Java API



```
/**
 * @model
 * @generated
 */
public interface State extends NamedElement {
/**
 * @model unique="false" ordered="false"
 * @generated
 */
boolean isIsStart();
/**
 * @param value the new value of the '<em>Is Start</em>' attribute.
 * @see #isIsStart()
 * @generated
 */
void setIsStart(boolean value);
}
```

Java annotated code

A human-comprehensible textual notation for FSM



FSM metamodel

conformsTo

```
fsm evenFsm
inputAlphabet "01"
// Char 'e' for even and 'o' for odd
outputAlphabet "eo"
states
start even odd
transitions
t1: even – 0 / o –> odd ;
t2: even – 1 / e –> even ;
t3: odd – 0 / e –> even ;
t4: odd – 1 / o –> odd ;
```

bridge?

Fsm = "**fsm**" id "**inputAlphabet**" string
"**outputAlphabet**" string
"**states**" (State)* "**transitions**" (Transition)*
State = ["**start**"] id
Transition = id ":" id "–" char ["/" char] "–>" id ";"

FSM EBNF grammar

conformsTo

Transformation Languages

Two approaches for writing transformation definitions:

In general-purpose programming language

In domain-specific transformation language

OMG approach:

Domain-specific transformation Language

- Declarative, imperative or hybrid

MOF 2.0 **Query/Views/Transformation** (QVT) standard

- Not only transformation, but also for model query and view

Example: DSTC/IBM Proposal

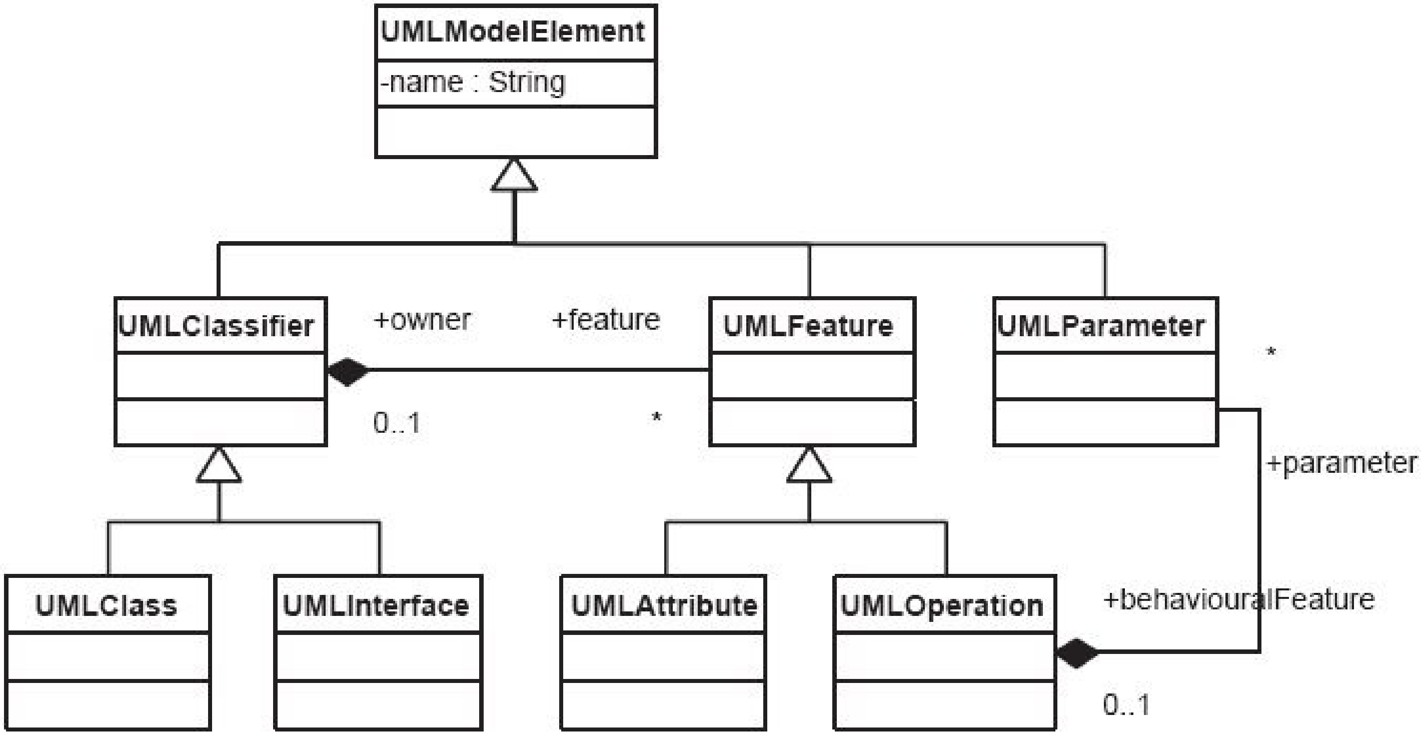
Declarative language

Example: **UML-to-Java transformation**

- Example is taken from DSTC/IBM/CBOP QVT Submission

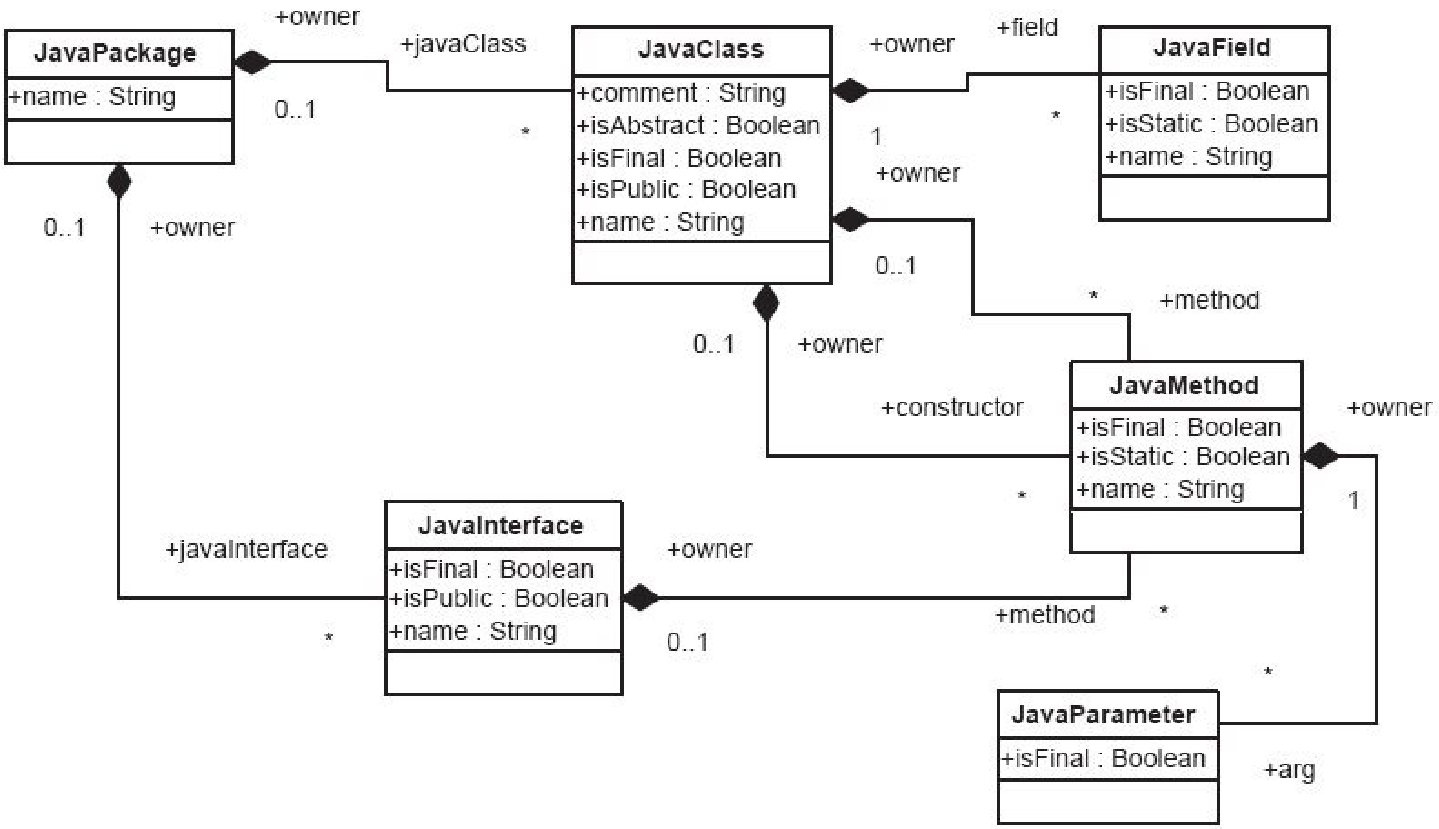
Source Meta-model

Simplified UML meta-model



Target Meta-model

Simplified Java meta-model



JavaPackage
+name : String

JavaClass
+comment : String
+isAbstract : Boolean
+isFinal : Boolean
+isPublic : Boolean
+name : String

JavaField
+isFinal : Boolean
+isStatic : Boolean
+name : String

JavaMethod
+isFinal : Boolean
+isStatic : Boolean
+name : String

JavaInterface
+isFinal : Boolean
+isPublic : Boolean
+name : String

JavaParameter
+isFinal : Boolean

Transformation Rules

Example of transformation declaration and a transformation rule:

```
TRANSFORMATION uml2java(SOURCE UML, TARGET
   Java)
   TRACKING TModel;

   RULE umlClassifierToJavaClass(X, Y)
        FORALL UMLClassifier X
        WHERE X.name = N
        MAKE JavaClass Y,
             Y.name = N
        LINKING X, Y BY JavaClassFromUMLClassifier;
        ...
```

EMF installation

- Download EclipseModeling Tools (Release: Neon) corresponding to your operating system from

  http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/neon1a

- Unzip it and run eclipse

- To test if everything works as expected, import the **project Quiz-Model.zip**
  - Unzip it and use *File -> Import*. Select *General* and *Existing Projects intoWorkspace*.
  - Browse for the folder where you unzipped the project, and then press *Finish*

See more details on the file **EclipseModelingTools_installation.pdf** (but do not consider the Luna version; download the latest NEON)