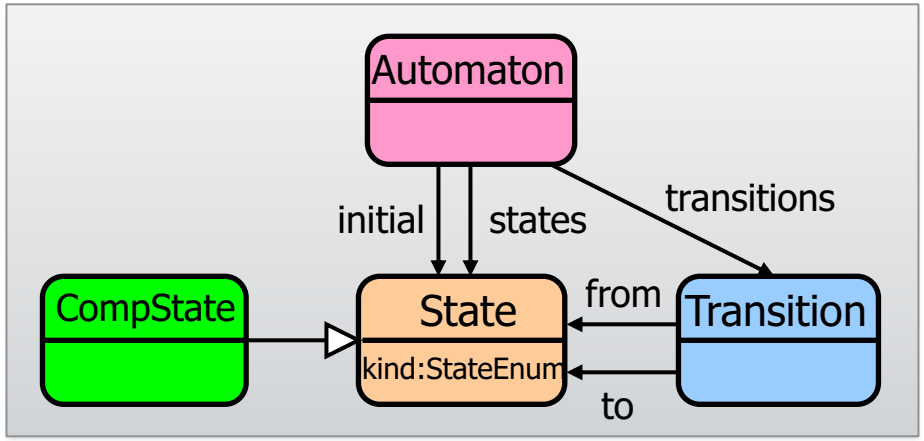
Domain-specific modeling
(and the Eclipse Modeling Framework)

Ákos Horváth
Gábor Bergmann
Dániel Varró
István Ráth

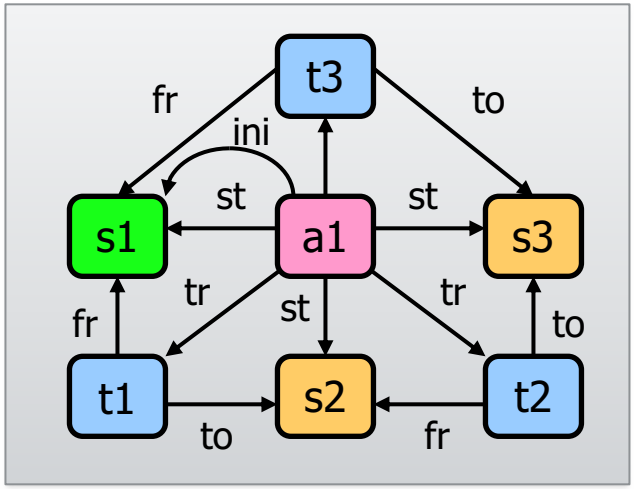
Model Driven Software Development
Lecture 2

METAMODELS, INSTANCE MODELS

Metamodel: Specify Concepts an Appl. Domain



Metamodel



Instance model

- Metamodel:
  - Precise specification of domain concepts of a modeling language
- Goal: to define...
  - Basic concepts
  - Relations between concepts
  - Attributes of concepts
  - Abstraction / refinement (Taxonomy, Ontology) between model elements
  - Aggregation
  - Multiplicity restrictions
  - ...

Metamodels and instance models

Reference / Association

Multiplicity

Generalization

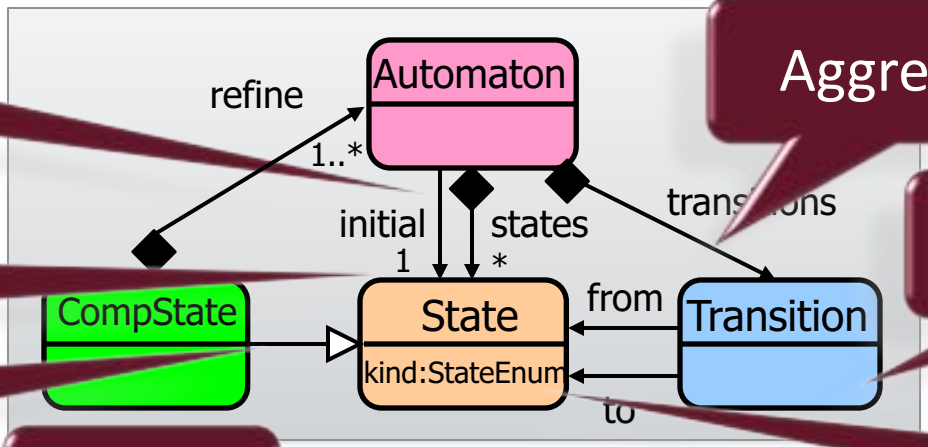
Aggregation

Class

Attribute

Object

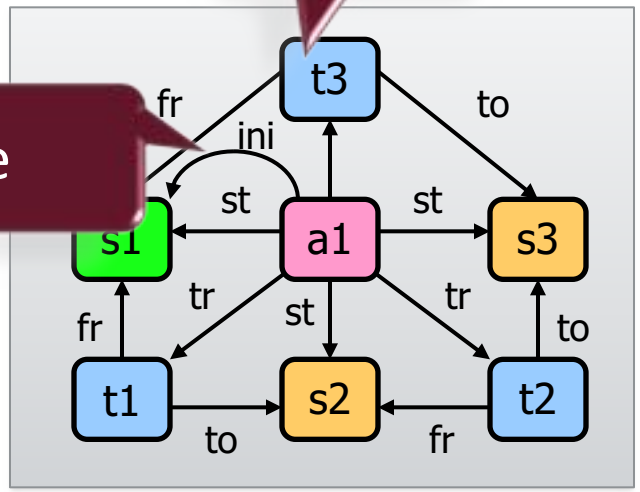
Role



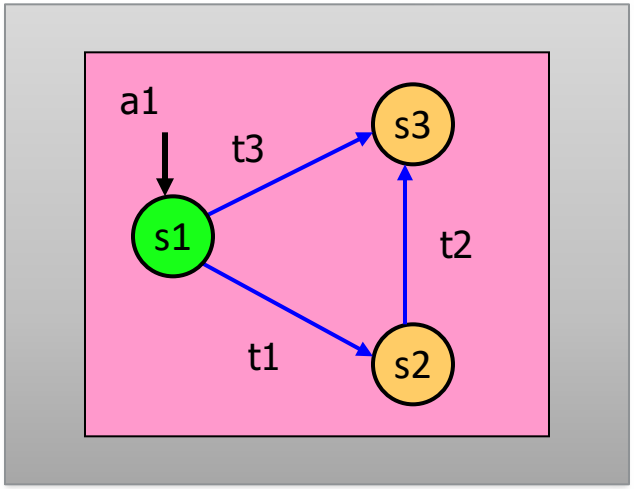
Metamodel level

Metamodel

Model level

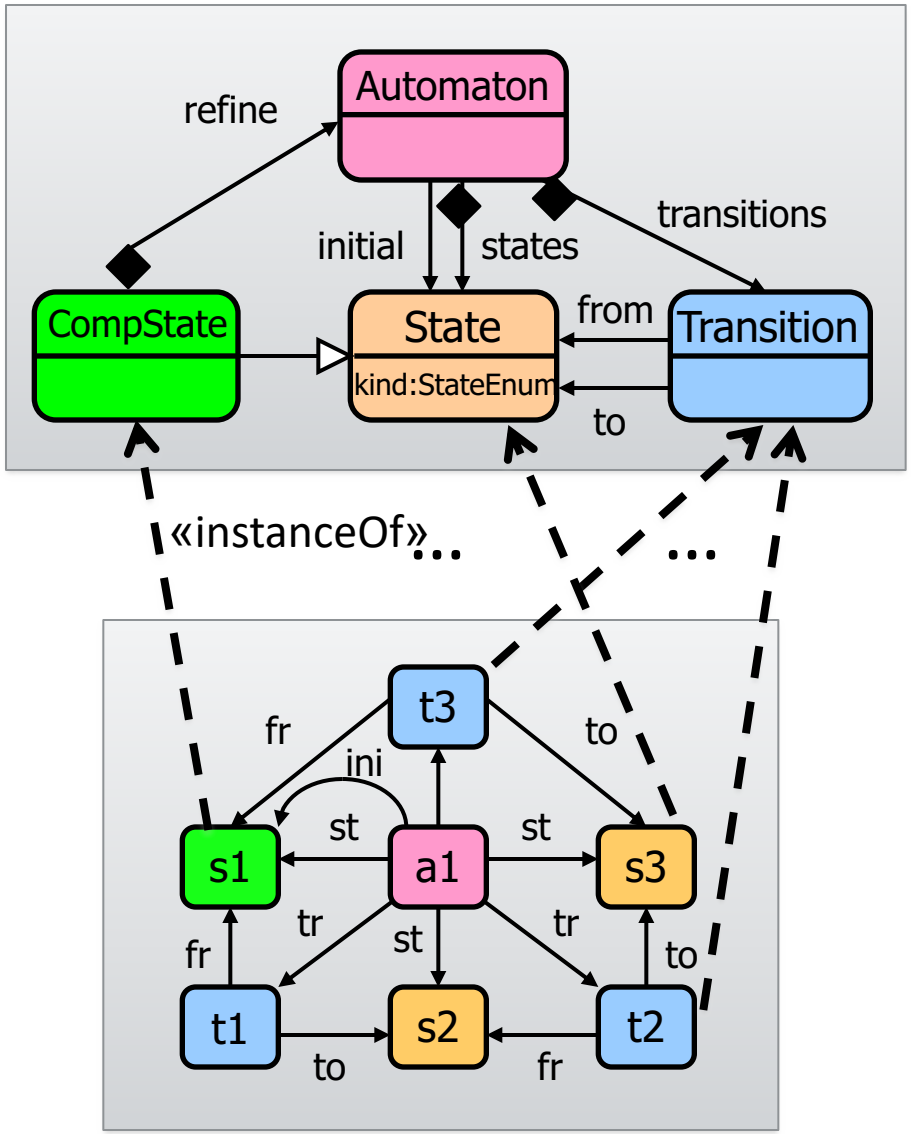


Abstract syntax



Concrete syntax

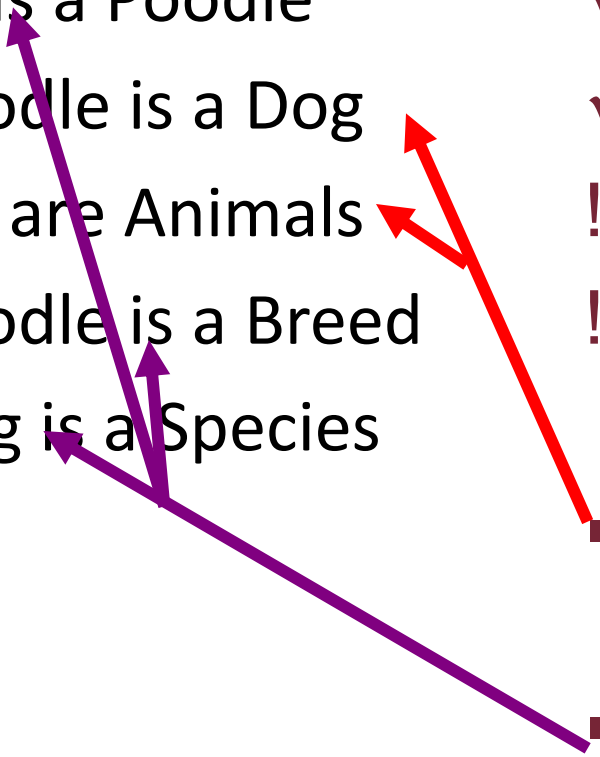
Type conformance /Instantiation /Classification



- Each model element is ***an instance of*** *(conforms to)* a metamodel element
- **Direct type**:
  - No other type exists lower in the type hierarchy
  - s1 → CompState
- **Indirect type**:
  - Superclass of the direct type
  - s1 → State

Classification vs. Generalization

1. Fido is a Poodle
2. A Poodle is a Dog
3. Dogs are Animals
4. A Poodle is a Breed
5. A Dog is a Species



✓ 1+2 = Fido is a Dog

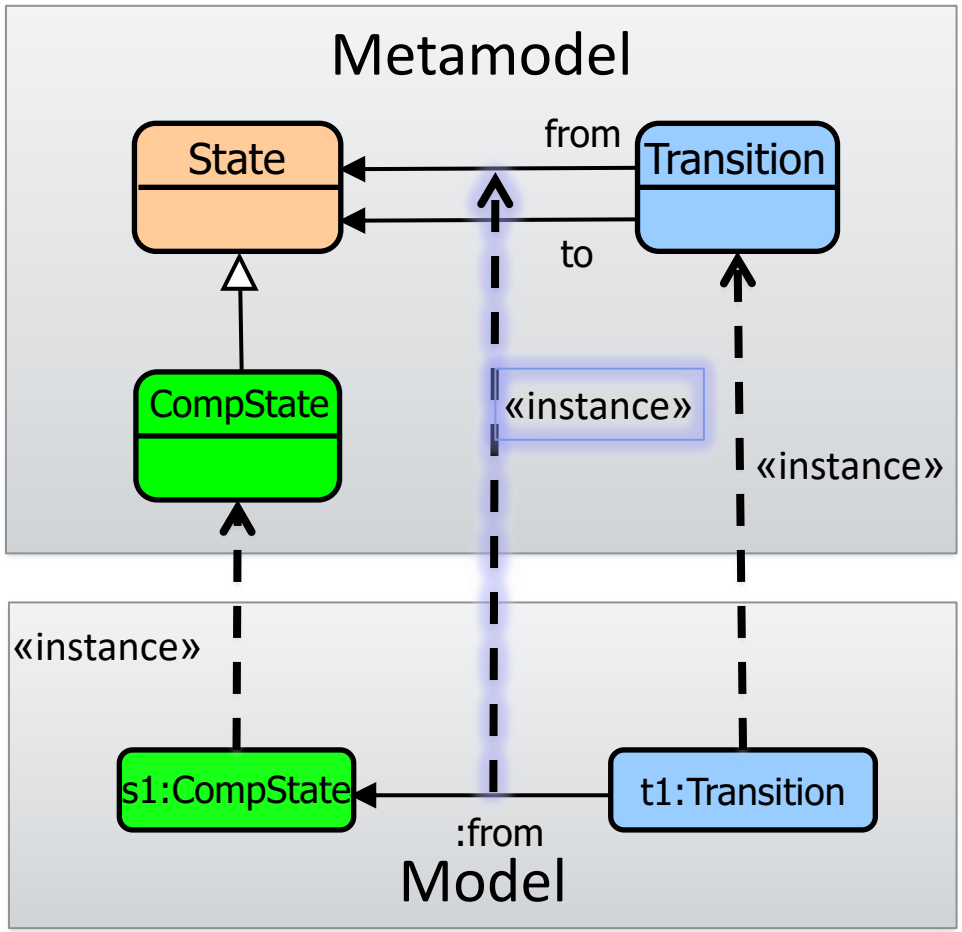
✓ 1+2+3 = Fido is an Animal

! 1+4 = Fido is a Breed

! 2+5 = A Poodle is a Species

- Generalization (SupertypeOf) is transitive
- Classification (InstanceOf) is NOT transitive

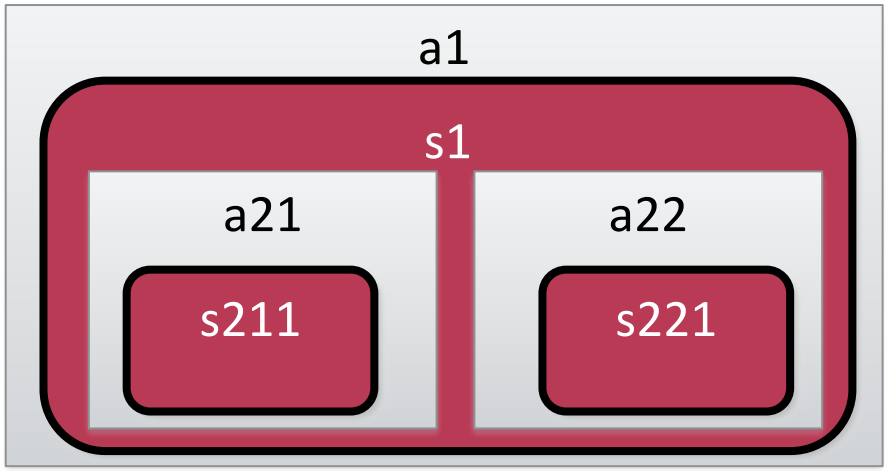
Type conformance of references

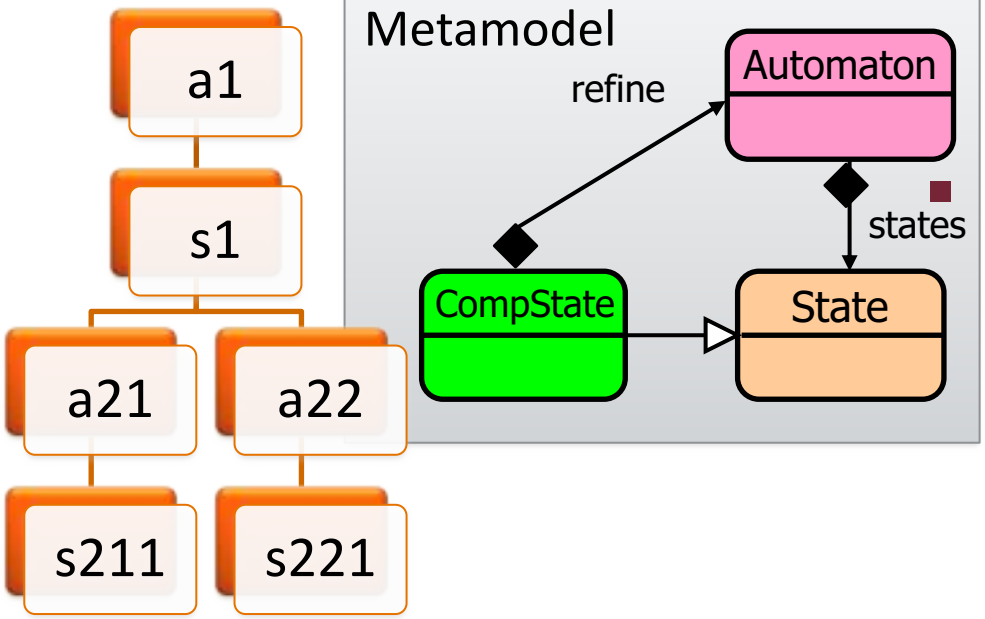


- A link in a model is **type conformant** if
  - ***type(src(link))*** is subtype of ***src(type(link))***
  - ***type(trg(link))*** is subtype of ***trg(type(link))***
  - Informally:
    - The type of the source object is a subtype of the source class of the link's type.
    - The type of the target object is a subtype of the target class of the link's type.

Can you define generalization for references?

Containment hierarchy

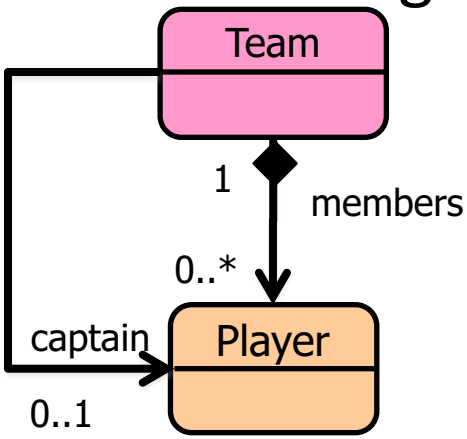




- Each model element has a unique parent
  - N children → 1 parent
  - Single root element
- Aggregation as relationship:
  - Defined in the metamodel along reference edges
  - Provides restriction for instance models
- Circularity
  - No circular containment (in the model)
  - Aggregation relations in the metamodel may be circular (hierarchy)

Multiplicity restrictions

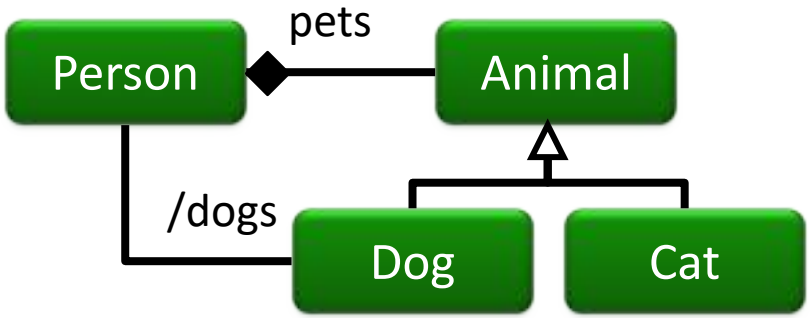
- Definition: Lower bound .. Upper bound
  - Lower bound: 0, 1, (non-negative integer)
  - Upper bound: 1, 2, … * (positive integer + any)
- Scope:
  - References: allowed number of links between objects of specific types
  - Attributes: e.g. arrays of strings (built-in values)

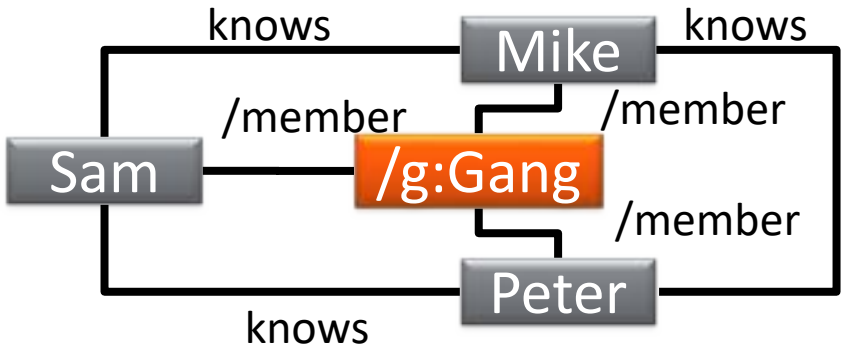


Which are the most common multiplicity definitions in practice?

Derived Features

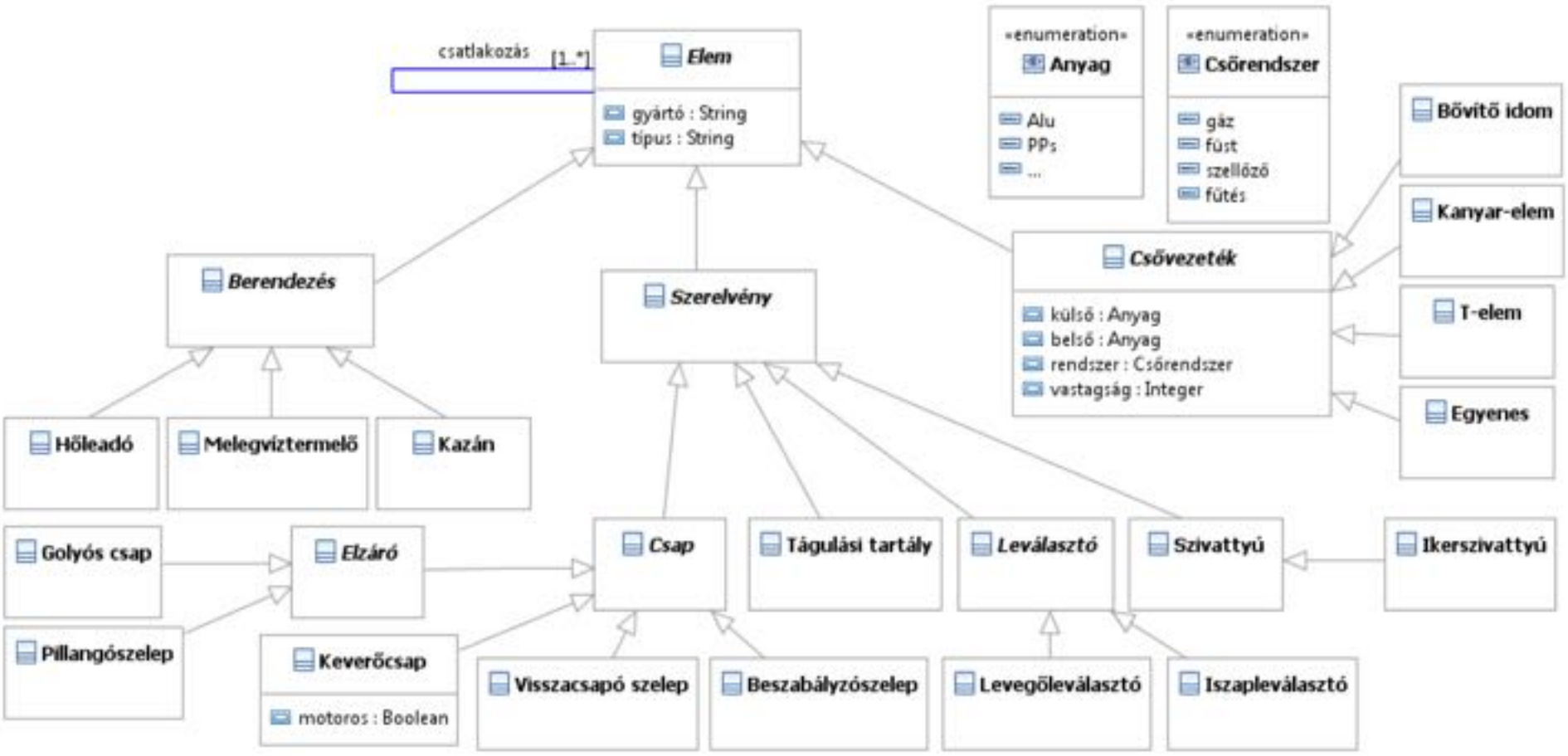
- A derived feature can be calculated from others
  - Usage: helpers for designers / tools
  - It need not be persisted
  - Automatic updates
- Derived attributes:
  `age = currYear – birth`
- Derived references:
  `dogs = -- pets --> Dog`
- Derived objects:
  - „Gang”:
    everyone knows everyone



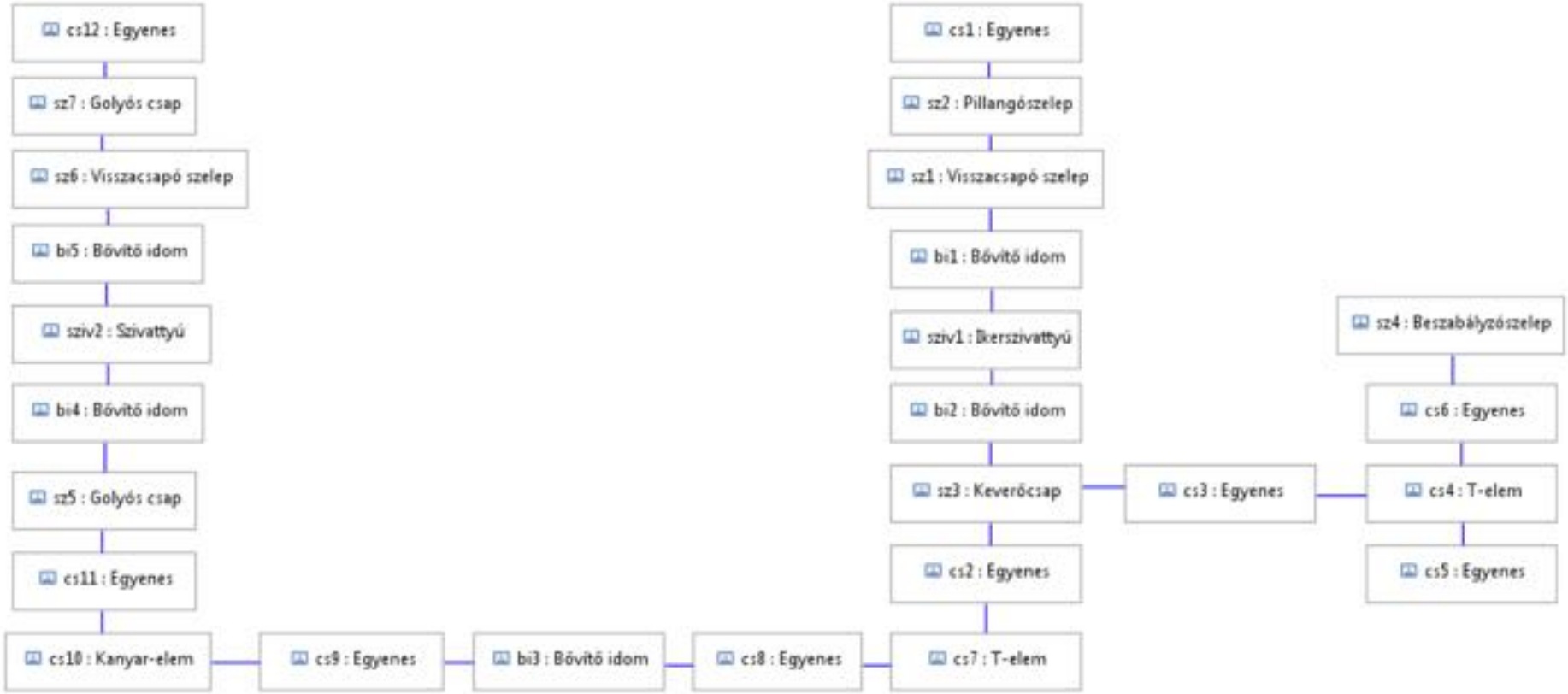


DOMAIN-SPECIFIC MODELING

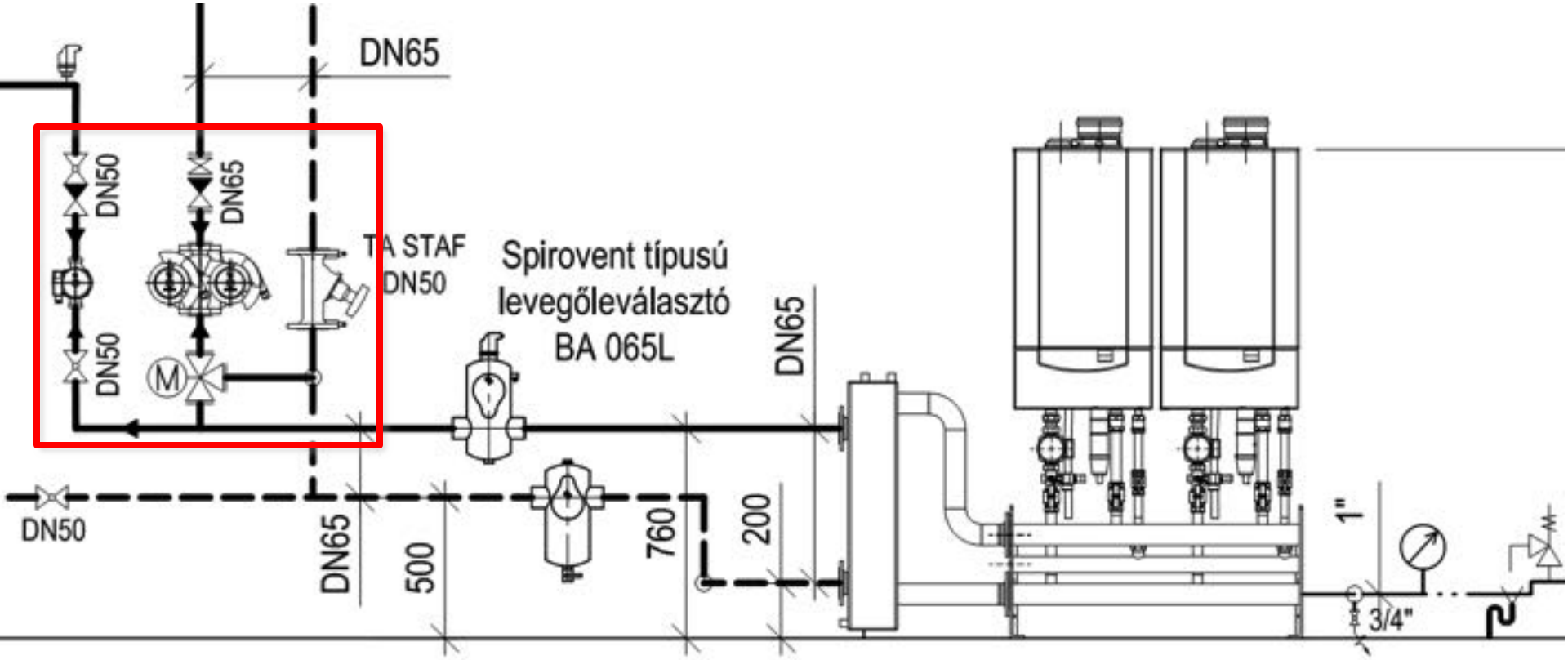
Example metamodel



Instance model, abstract syntax



Instance model, concrete syntax



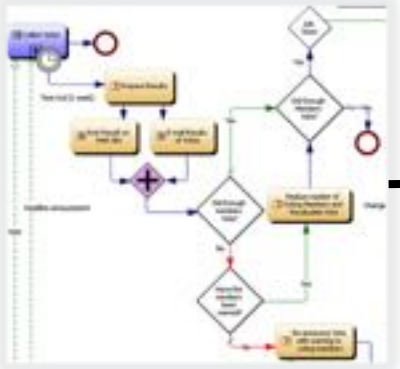
Honeywell keverőcsap DN50 K_{vs} 40

Spirovent típusú iszapleválasztó BE 065L

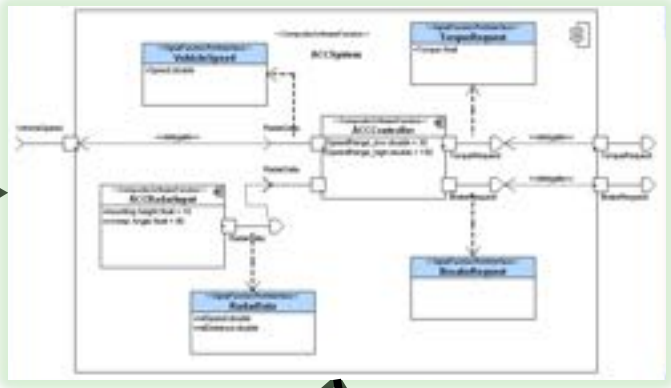
Remeha Quinta kaszkád rendszer hidraulikus váltóval

Domain specific modeling languages

Business analyst

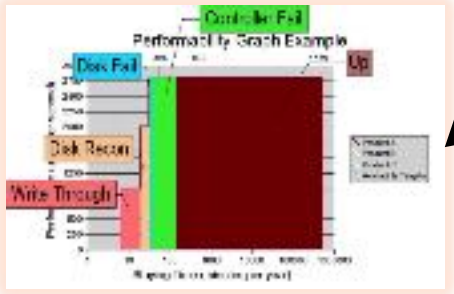


Business process



System designer

Dependability expert



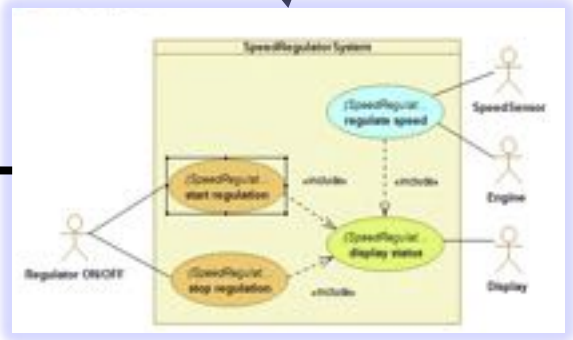
Dependability model

Risk model

Security expert

Software developer

Programming language

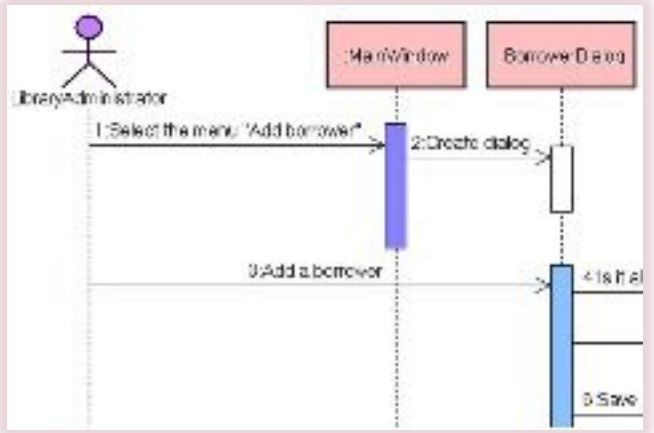


Software model

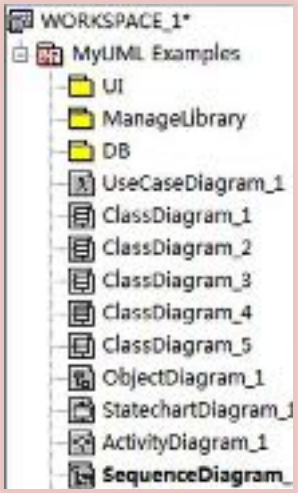
Software architect

Usage example of DSMs

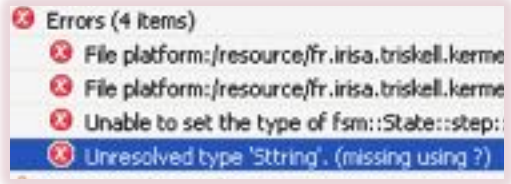
Concrete syntax



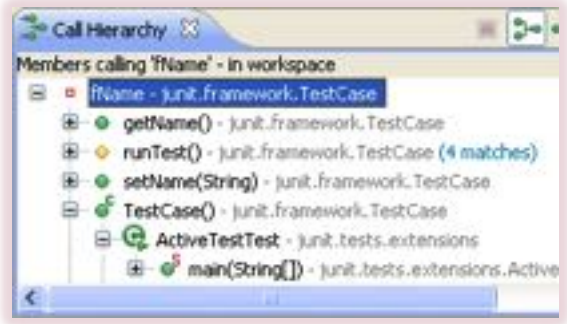
Abstract syntax



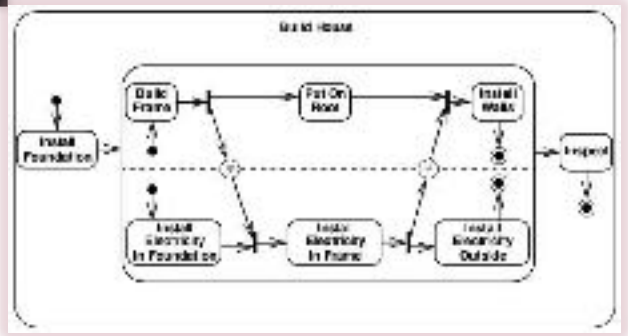
Well-formedness constraints



Behavioural semantics, simulation, refactoring



Call graph (view)



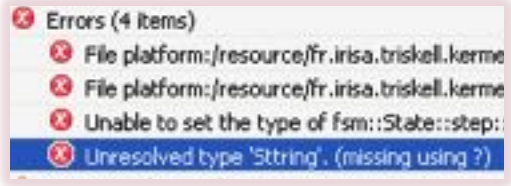
State machines (different DSM)

Structure of DSMs

Graphical syntax

Abstract syntax

Well-formedness constraints

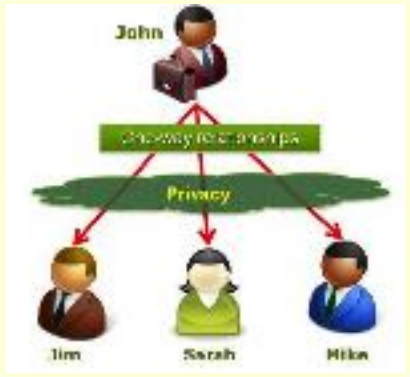


Behavioural semantics, simulation, refactoring

Code generation

Mapping

Textual syntax



View

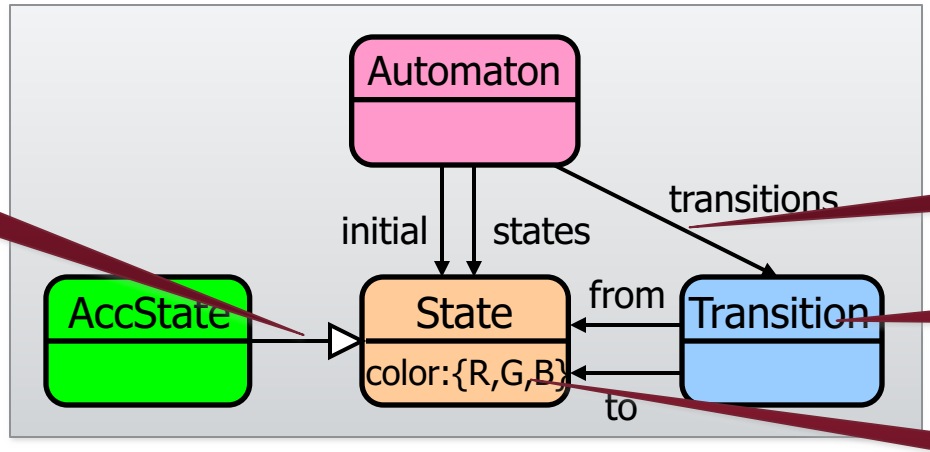
Code (documentation, configuration)

Aspects of Defining DSMLs

Designing modeling languages

- Language design checklist
  - **Abstract syntax** (metamodel)
    - Taxonomy and relationships of model elements
    - Well-formedness rules
  - **Semantics** (does not *strictly* belong to a language)
    - Static
    - Behavioural
  - ???
  - **Concrete syntax**
    - Textual notation
    - Visual notation

Revisiting the example



Generalization

Association

Instantiation

Class

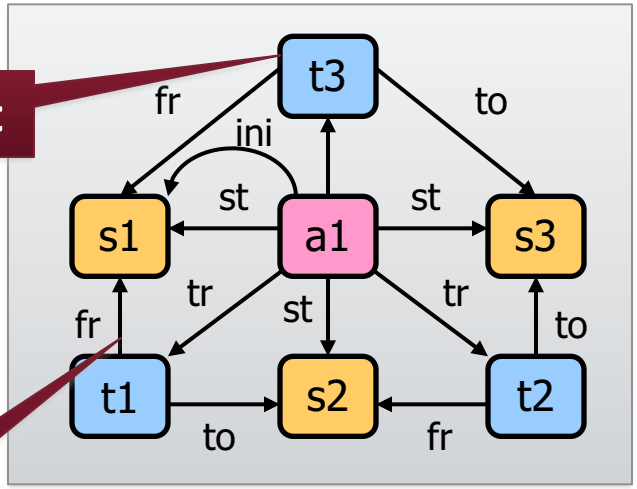
Attribute

Metamodel

Meta (Language) level

(Instance) Model level

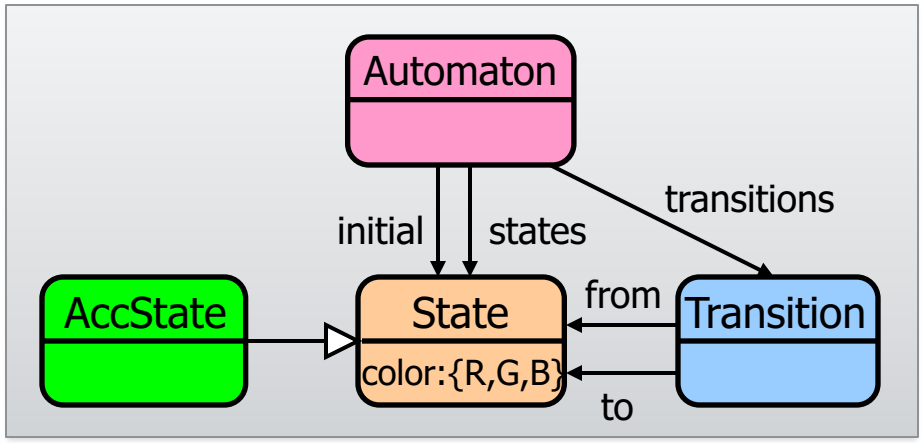
Object



Link

Model in abstract syntax

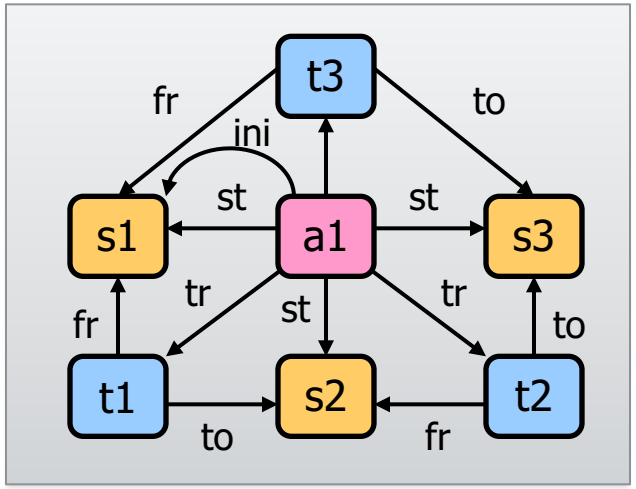
Revisiting the example



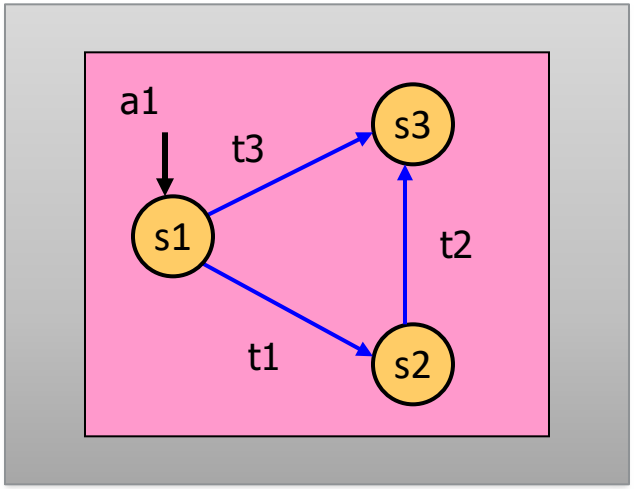
Meta (Language) level

Metamodel

Model level

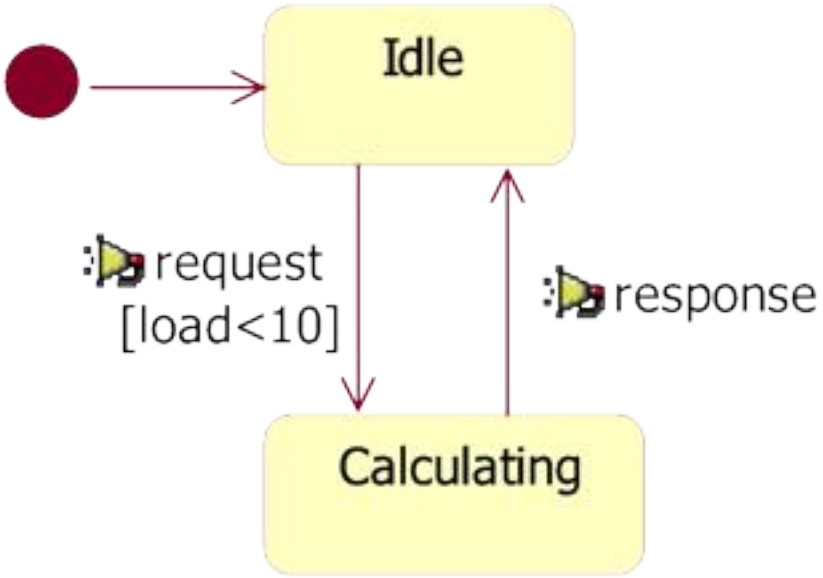


Abstract syntax



Concrete syntax

Example: Concrete Syntax




```
request() {
    if (state == "idle" &&
    this.load<10)
    state = "calculating";
}
response() {
    if (state == "calculating")
    state = "idle"
}
```

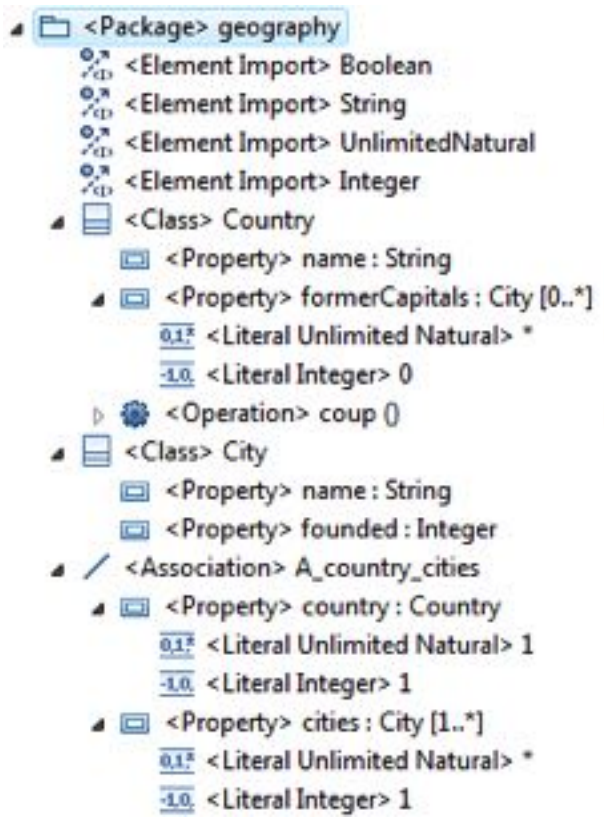
Graphical notation

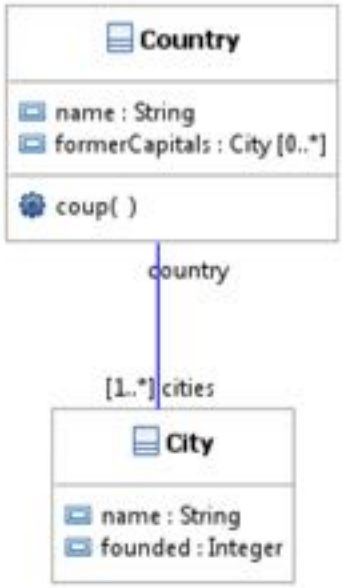
Textual notation

Textual vs. Visual

- Textual notation:
  - \+ Easy to write: Able to capture complex expressions
  - \- Difficult to read: Difficult to comprehend and manage after certain complexity (e.g what refers me?)
- Visual notation:
  - \+ Easy to read: Able to express (selected / subset of) details in an intuitive, understandable form
  - \+ Safe to write: Able to construct syntactically correct models
  - \- Difficult to write: graphical editing is slower

Example: UML model






```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Package name="geography" xmi:version="2.1"
   xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
   xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML"
   xmi:id="_7qi_AS2uEd-VCP9iY9GYHg">
[...]
  <packagedElement xmi:type="uml:Class" name="Country" xmi
    <ownedAttribute name="name" aggregation="composite" xm
      <type xmi:type="uml:PrimitiveType" href="pathmap://U
    </ownedAttribute>
    <ownedAttribute name="formerCapitals" aggregation="com
      <upperValue value="*" xmi:type="uml:LiteralUnlimited
      <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_y
    </ownedAttribute>
    <ownedOperation name="coup" xmi:id="_fHicEC2vEd-VCP9iY
      <ownedParameter direction="return" xmi:id="_1e7b8C2v
    </ownedOperation>
  </packagedElement>
  <packagedElement xmi:type="uml:Class" name="City" xmi:id
    <ownedAttribute name="name" aggregation="composite" xm
      <type xmi:type="uml:PrimitiveType" href="pathmap://U
    </ownedAttribute>
    <ownedAttribute name="founded" aggregation="composite"
      <type xmi:type="uml:PrimitiveType" href="pathmap://U
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Association" xmi:id="_Xq_
    <ownedEnd name="cities" type="__KgpUC2vEd-VCP9iY9GYHg"
      <upperValue value="*" xmi:type="uml:LiteralUnlimited
```

Abstract Syntax

Graphical notation (Class Diagram)

Textual notation (XMI 2.1)

Multiplicity of Notations

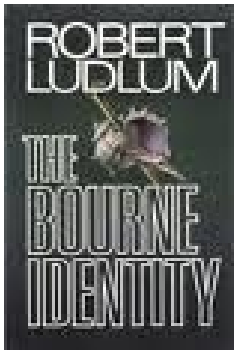
- One-to-many
  - 1 abstract syntax → many textual and visual notations
    - Human-readable-writable textual or visual syntax
    - Textual syntax for exchange or storage (typically XML)
    - In case of UML, each diagram is only a partial view
  - 1 abstract model → many concrete forms in 1 syntax!
    - Whitespace, diagram layout
    - Comments
    - Syntactic sugar
  - 1 semantic interpretation → many abstract models
    - e.g. UML2 Attribute vs. one-way Association

METALEVELS

Exercise Metalevels

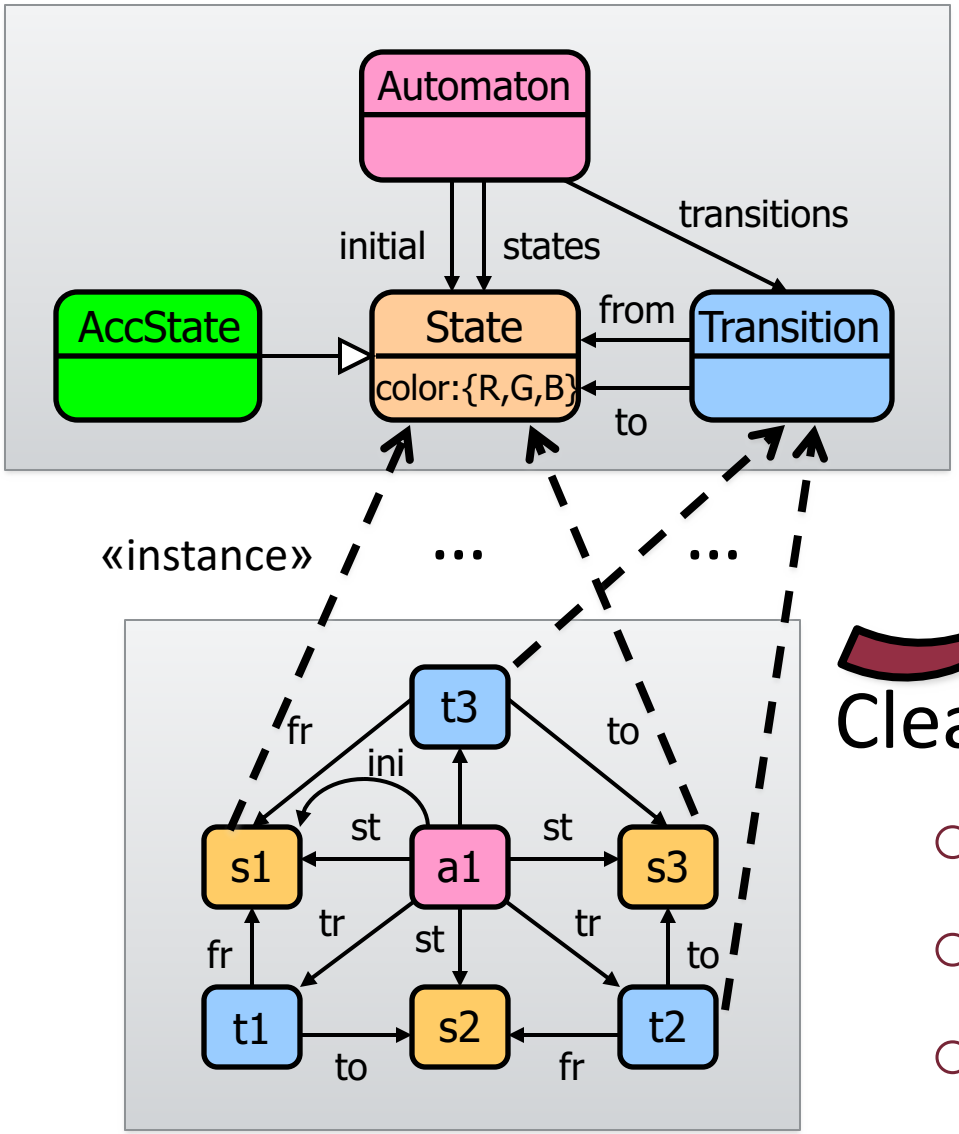
- Nodes
  - Film, Human, Novel, Psycho (film), Book, Man, Thriller, Work of Art, The Bourne Identity (novel), Genre, Robert Ludlum, Sir Alfred Hitchcock, this book here:

Demonstrated by the exercise:
- **Instantiation vs. subtyping**
- **Edge subtyping**
- **Metalevels**
- **Multi-level metamodeling**
- **Deep instantiation**



- Edges
  - written by, directed by, creator, subtype, instance

Metalevels



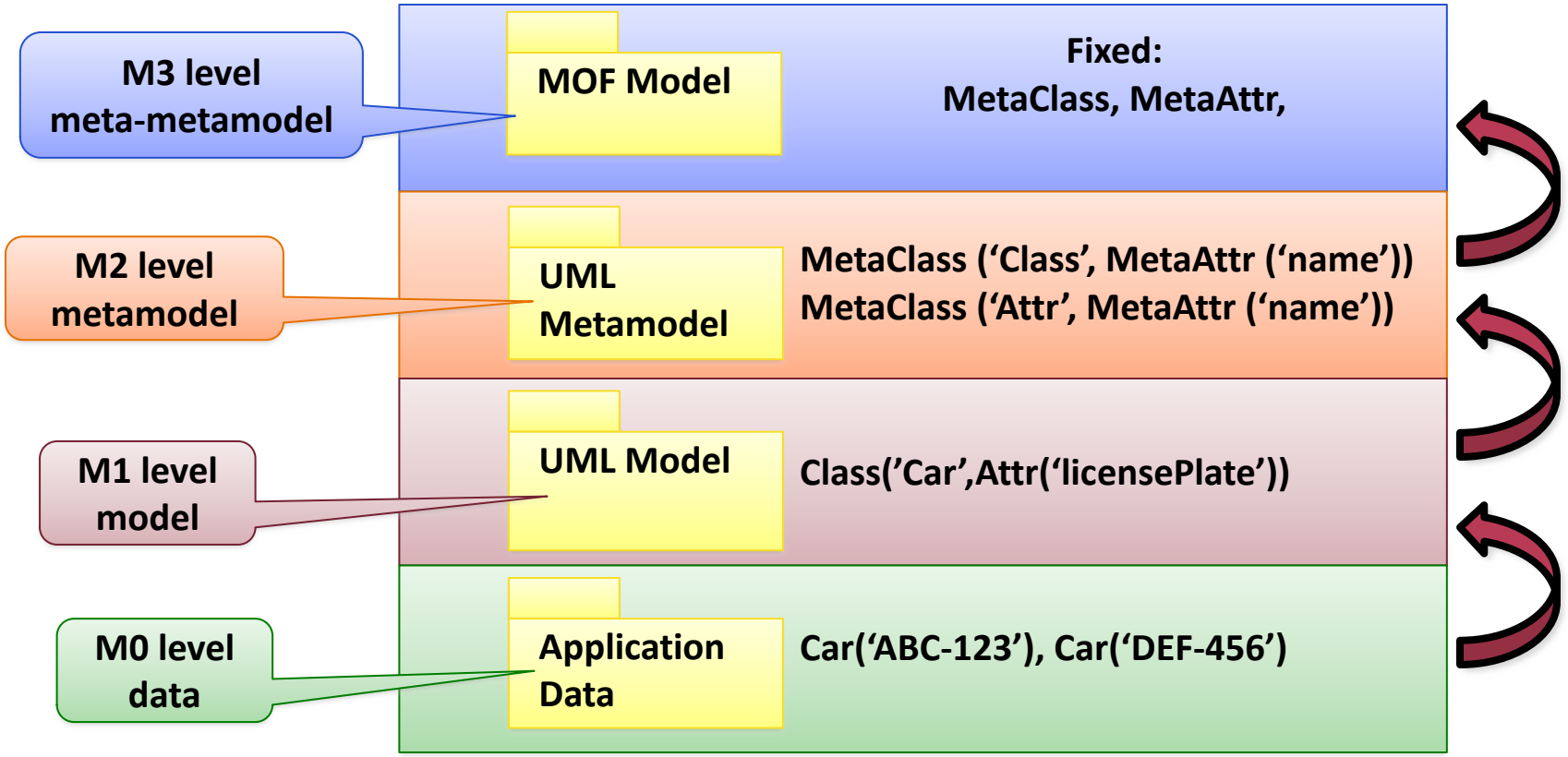
„Meta" relationship between models

Clear level separation:

- Loses some flexibility
- Much easier to understand
- Usually enough to keep two levels in mind at once

Metalevels in MOF

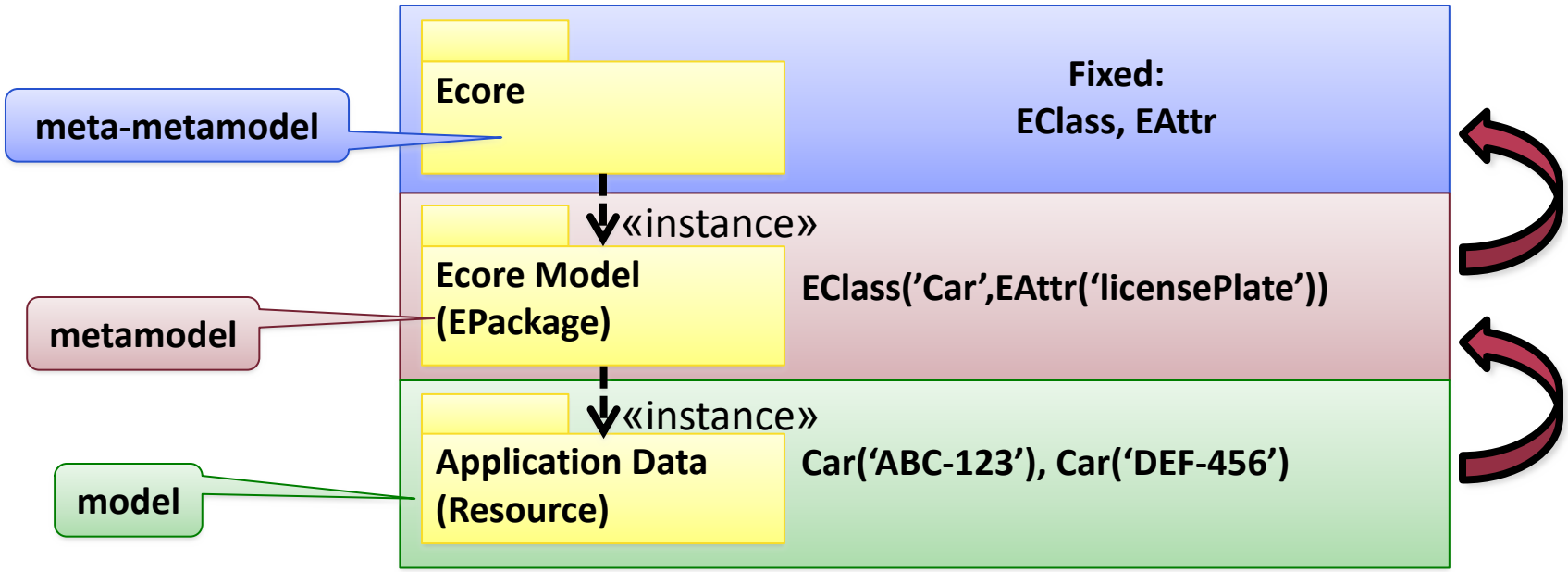
- OMG's MOF (Meta Object Facility)
  - 4-layer approach



  - Why exactly four levels?

Metalevels in other approaches

- EMF (Eclipse Modeling Framework)



- Multi-level metamodeling
  - VPM
  - Ontologies

SEMANTICS

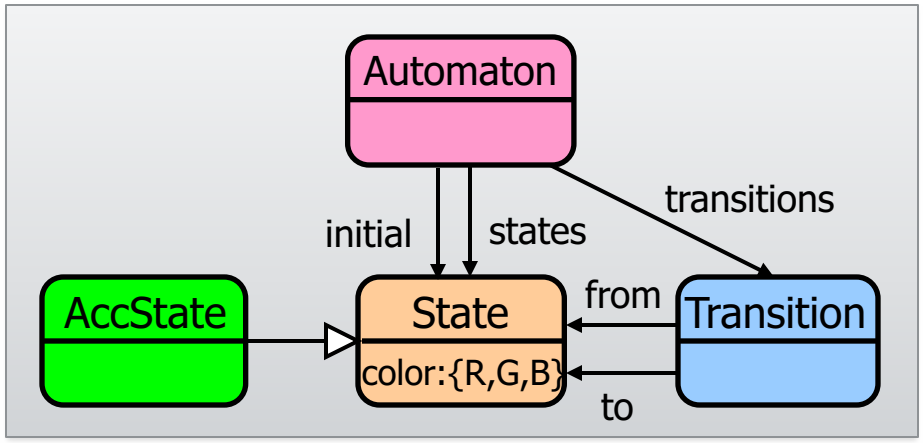
Semantics

- Semantics: the meaning of concepts in a language
  - Static: what does a snapshot of a model mean?
  - Dynamic: how does the model change/evolve/behave?
- Static Semantics
  - Interpretation of metamodel elements
  - Meaning of concepts in the abstract syntax
  - **Formal**: mathematical statements about the interpretation
    - E.g. formally defined semantics of OCL

Dynamic Semantics

- **Operational**
  - Modeling the operational behavior of language concepts
  - „interpreted”
  - e.g. defining how the finite automaton may change state at run-time
  - Sometimes dynamic features are introduced only for formalizing dynamic sematics
- **Denotational** (Translational)
  - translating concepts in one language to another language (called **semantic domain**)
  - „compiled”
  - E.g. explaining state machines as Petri-net

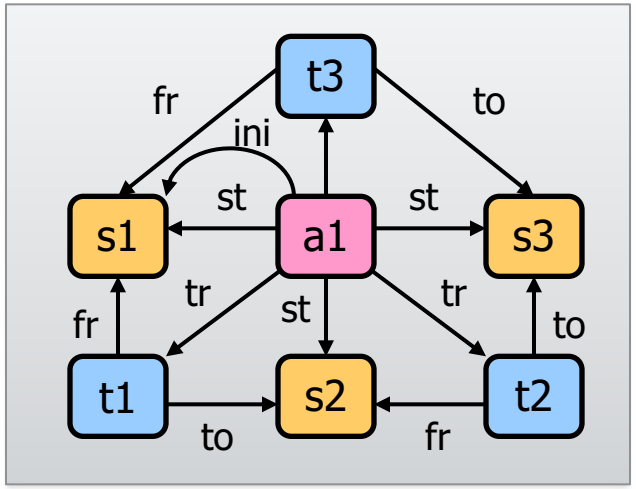
Example: Denotational semantics



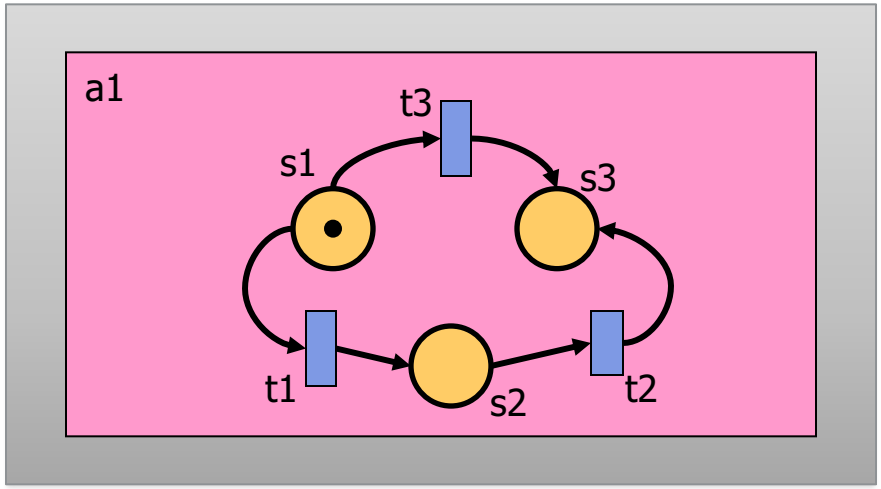
Meta (Language) level

Metamodel

Model level



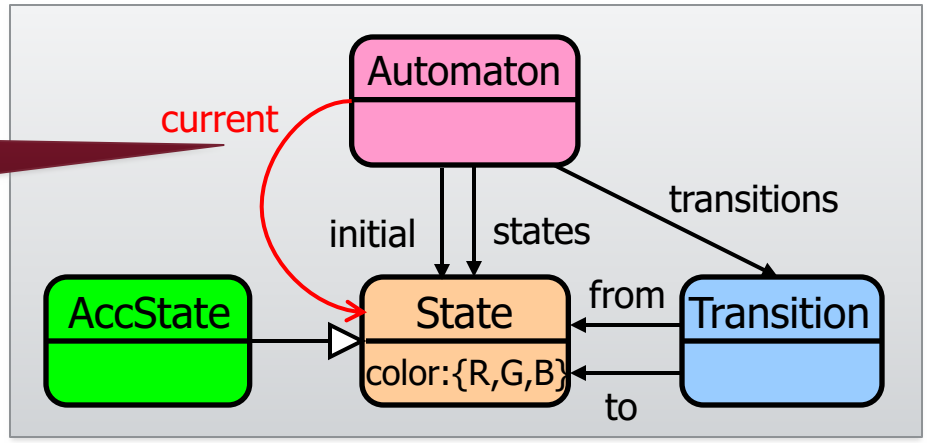
Abstract syntax



Semantic Domain

Example: Operational semantics

Dynamic feature

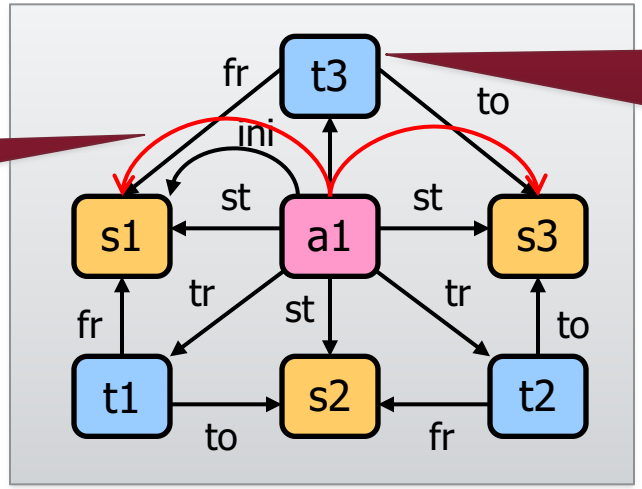


Meta (Language) level

Metamodel

(Instance) Model level

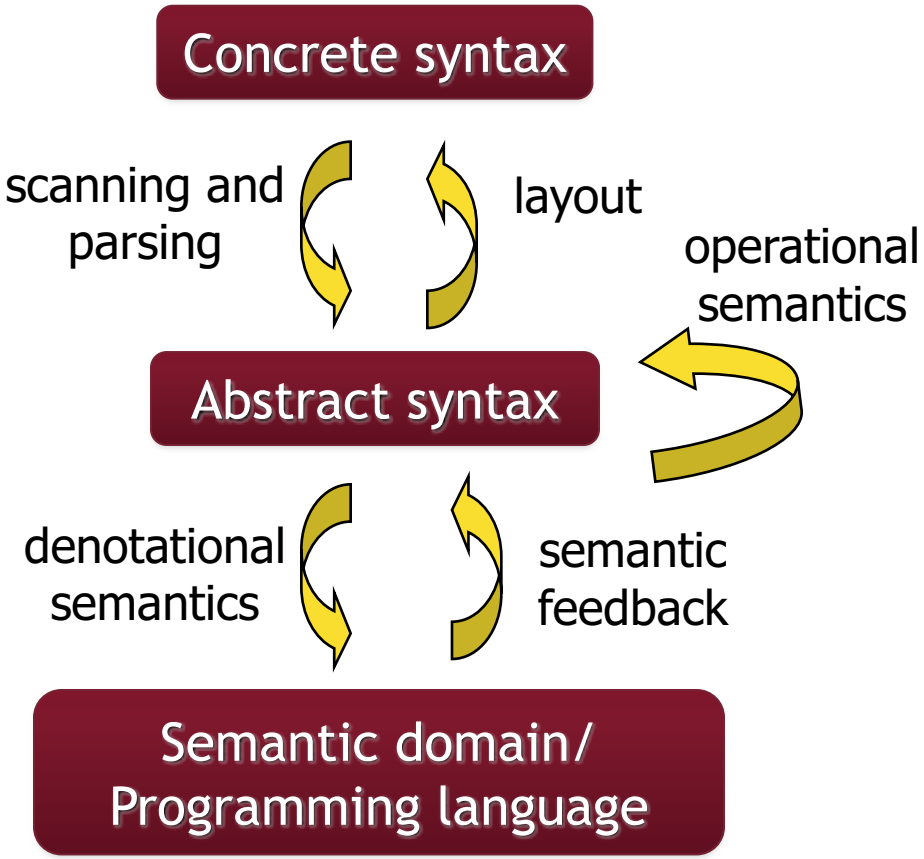
At first, 'current '= 'initial'



Possible evolution: 'current' is redirected along a transition

Model in abstract syntax

Relationship of models



DOMAIN-SPECIFIC MODELING LANGUAGES IN ENGINEERING PRACTICE

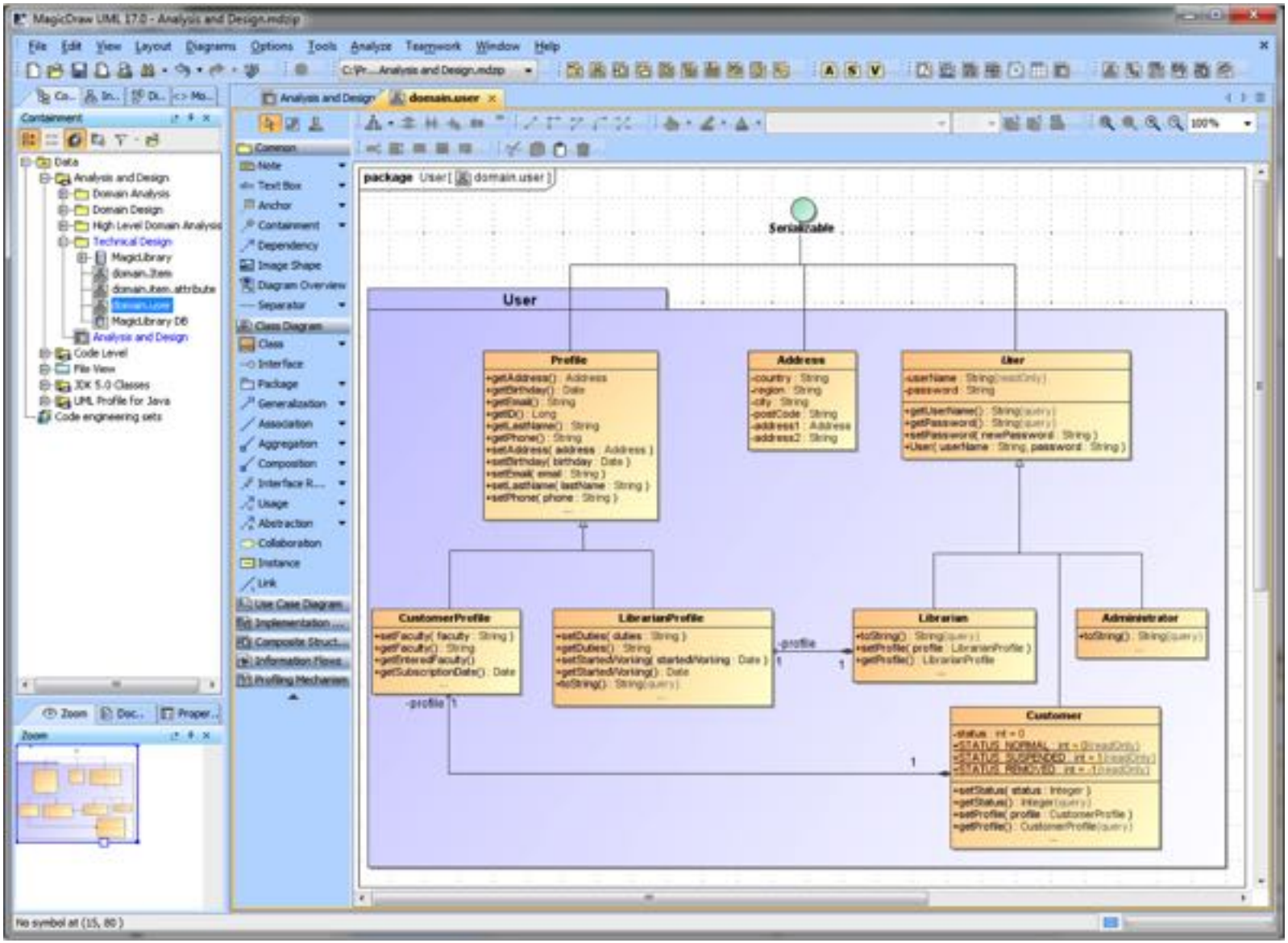
Well known DSLs

- MATLAB, SQL, Erlang, Shell scripts, AWK, Verilog, YACC, R,S, Mathematica, XSLT, XMI, OCL, Template languages, …

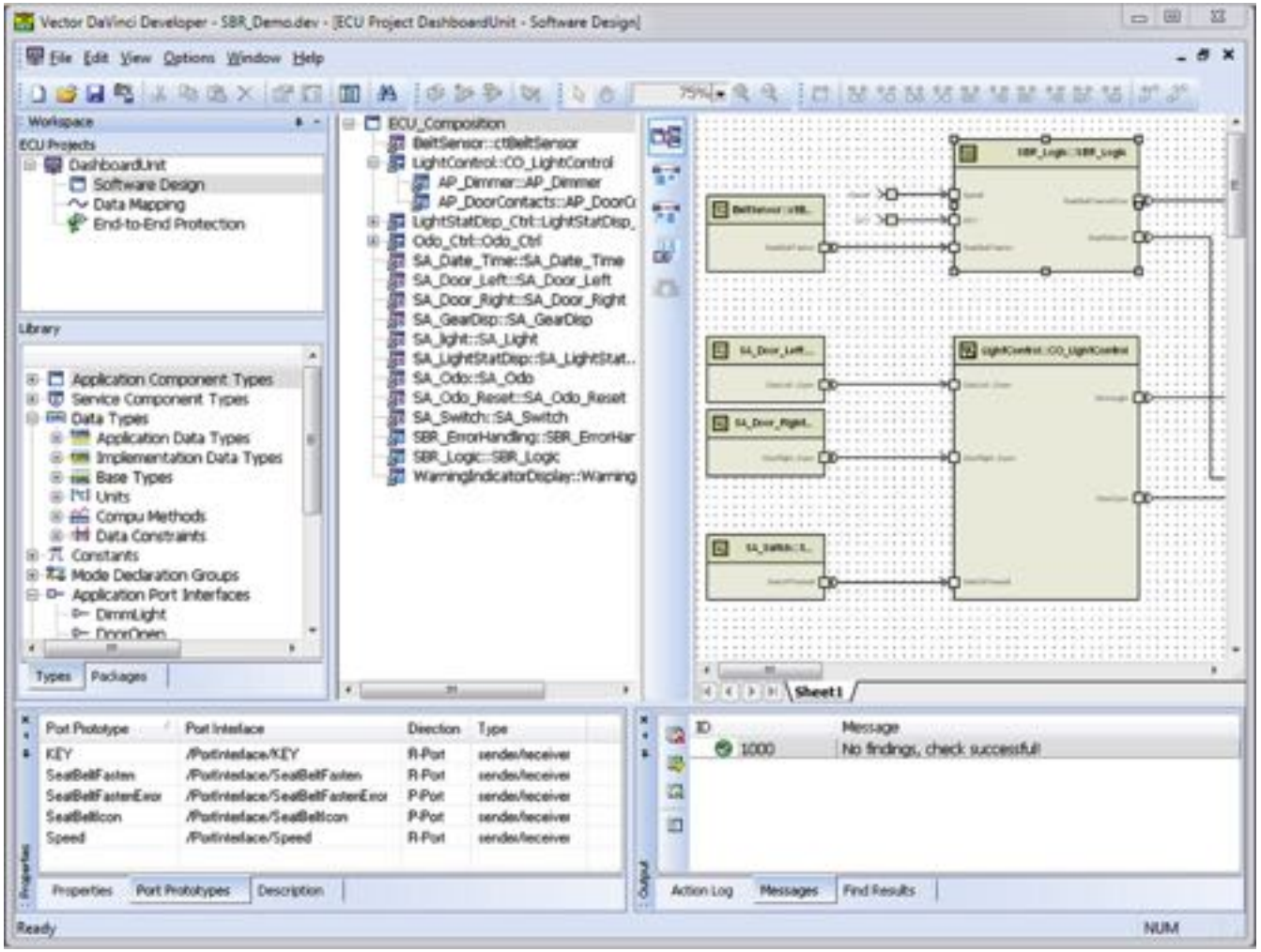
Industry standard DSMLs

- Automotive
  - AUTOSAR, MATLAB StateFlow, EAST-AADL
- Aerospace
  - AADL
- Railways
  - UML-MARTE
- Systems engineering
  - SysML, UML-FT

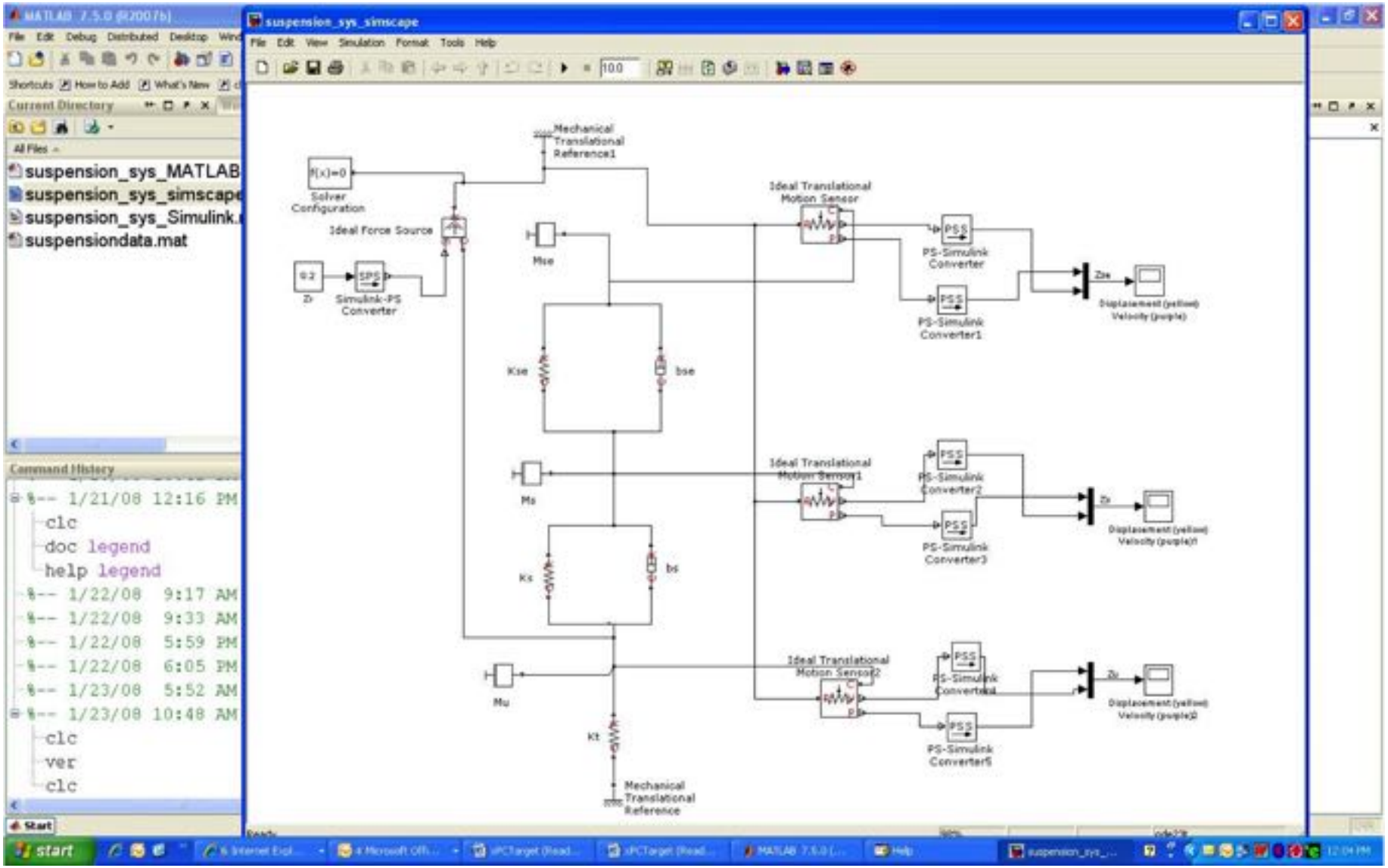
SysML: MagicDraw



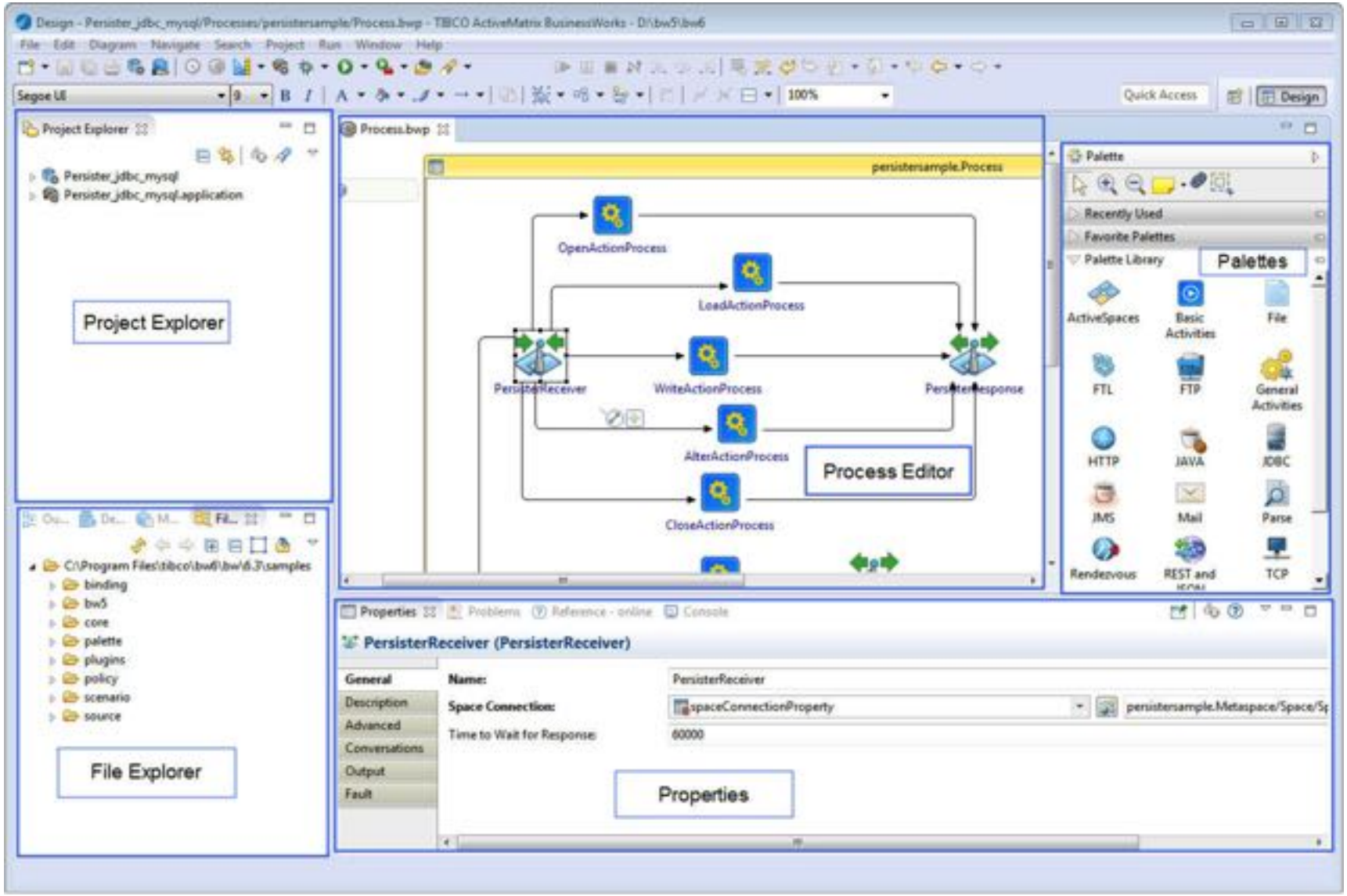
AUTOSAR: Vector DaVinci Developer



Matlab Simulink



TIBCO Business Studio



DSM Technologies

- MATLAB
- Rational Software Architect

COTS

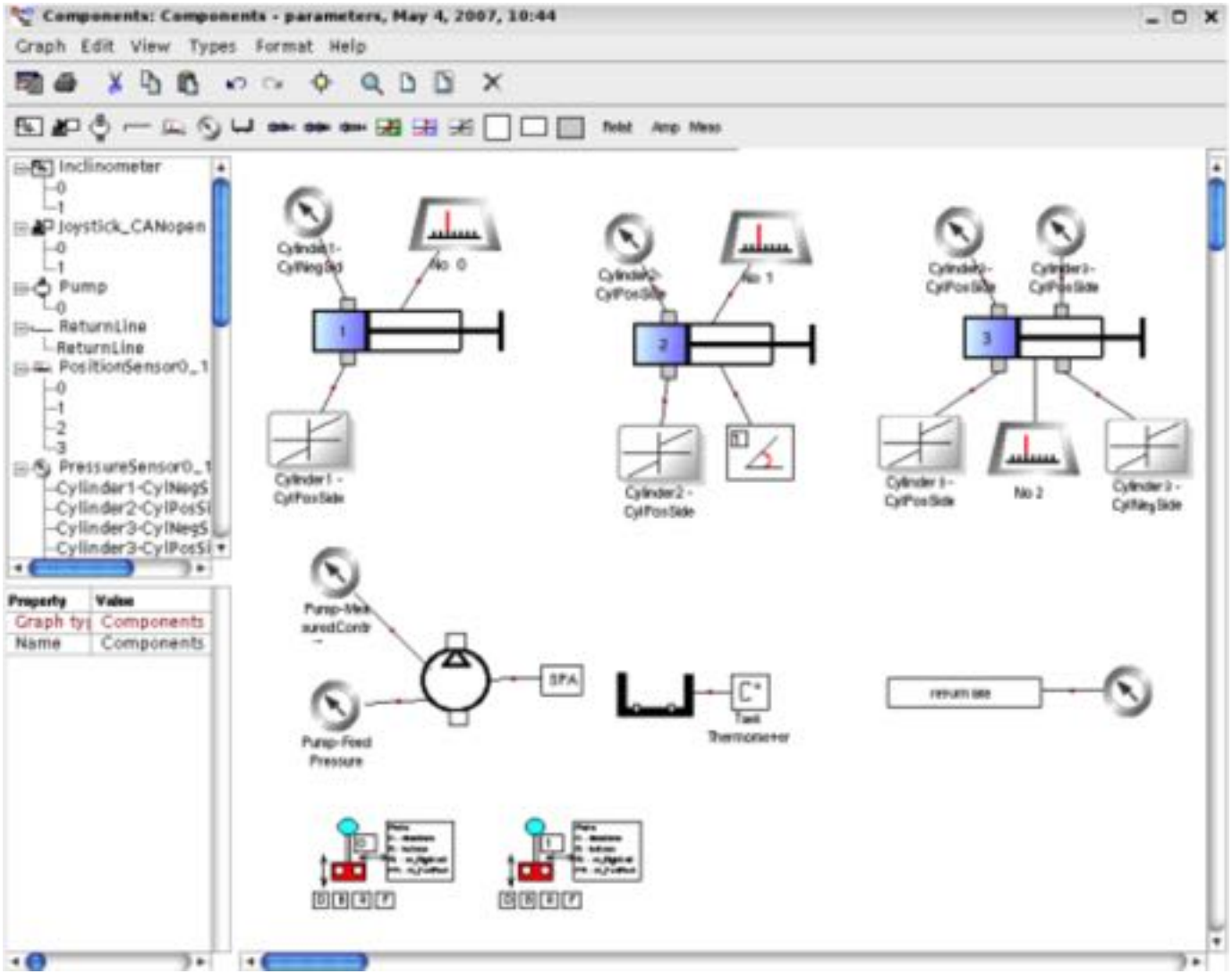
- Eclipse
  - EMF, Sirius
  - Xtext/Xcore/etc.
- Microsoft
  - DSL Tools (Visual Studio) / M / Oslo etc.
- MetaCase
  - MetaEdit+
- JetBrains MPS

Language engineering (industry)

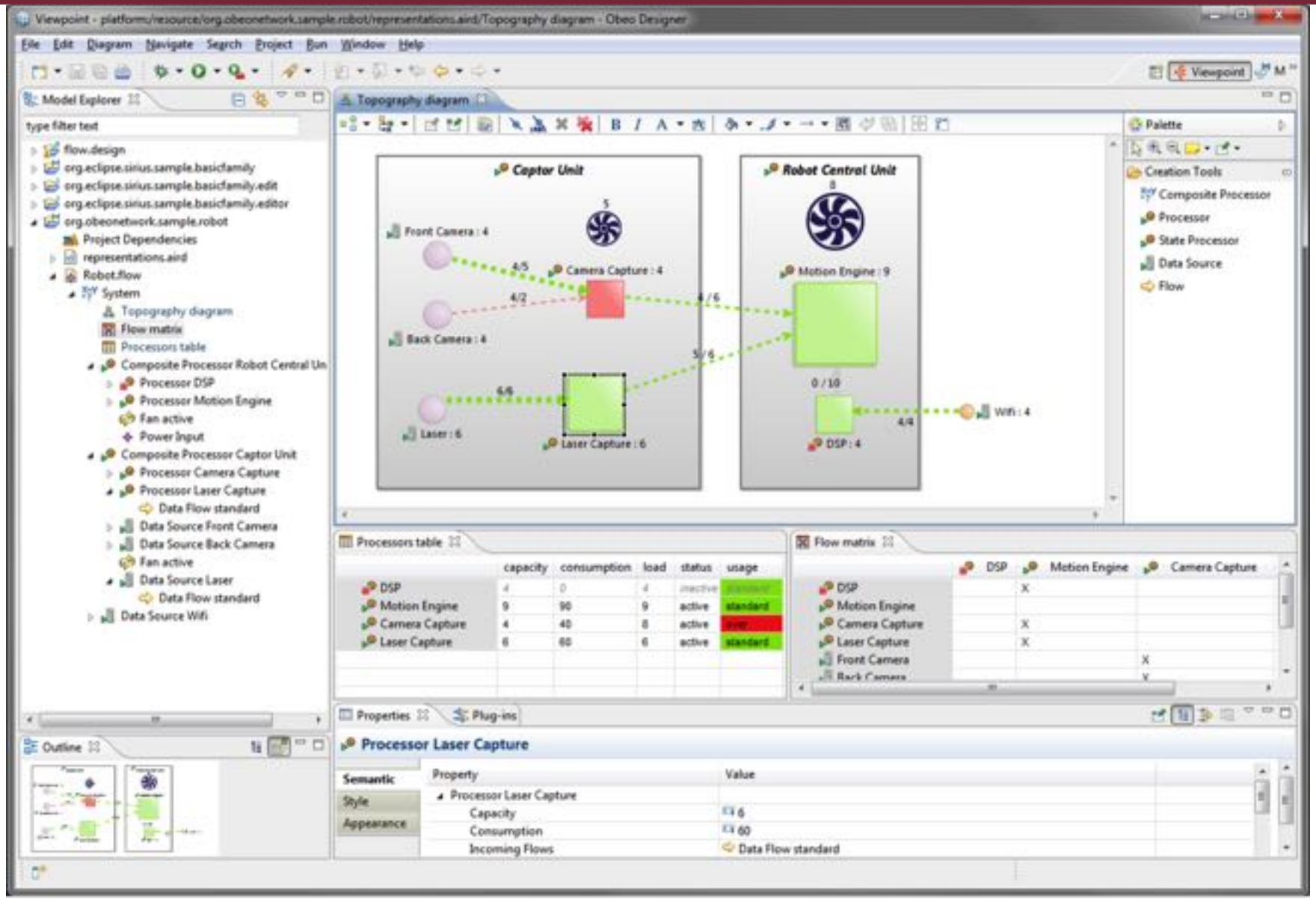
- WebGME, Kermeta

Academia

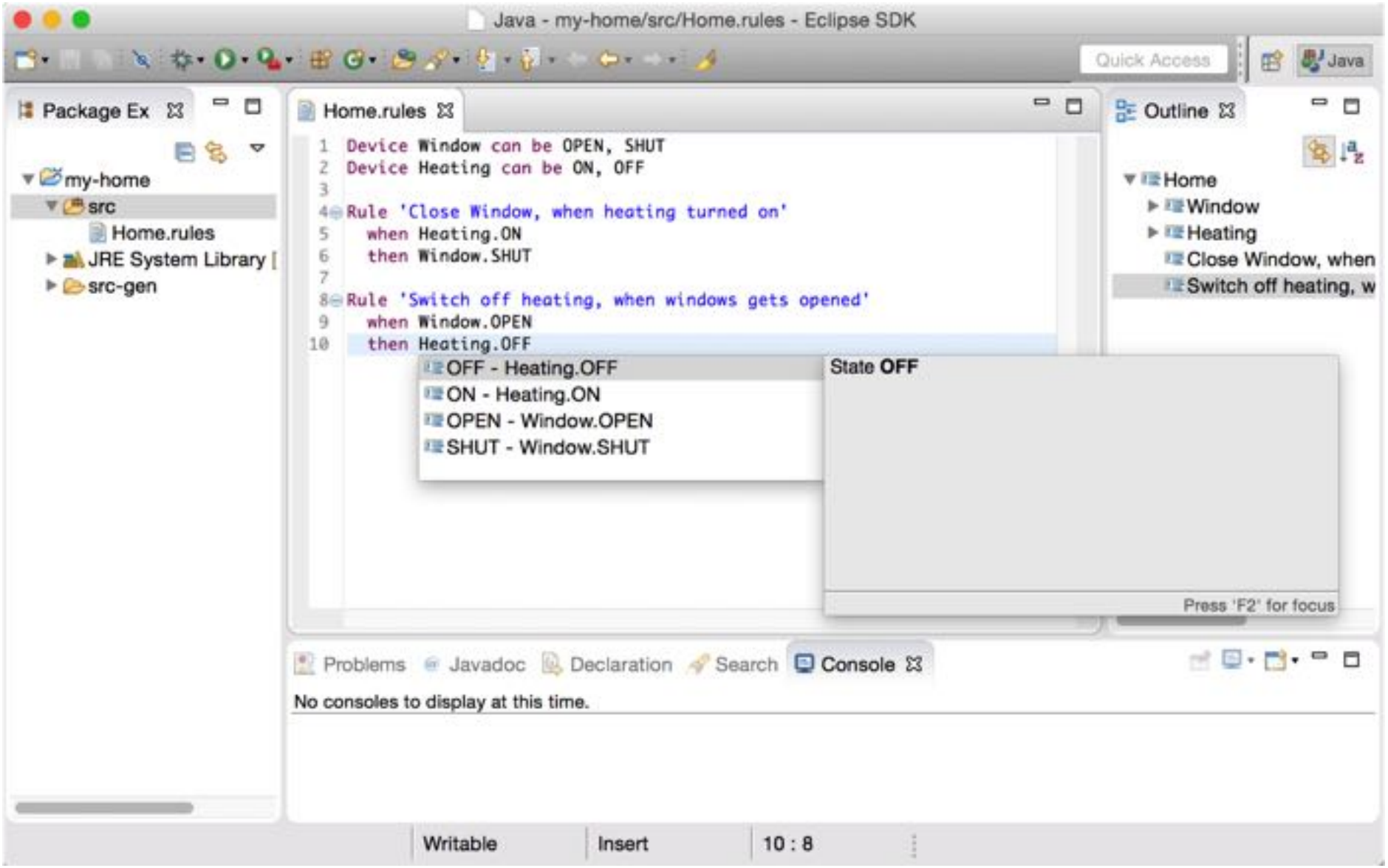
MetaEdit+


Components: Components - parameters, May 4, 2007, 10:44
Graph Edit View Types Format Help
Inclinometer
Joystick_CANopen
Pump
ReturnLine
PositionSensor0_1
PressureSensor0_1
Property
Value
Graph ty
Components
Name
Components
SPA
Pump-Feed Pressure
Tank Thermometer
return line

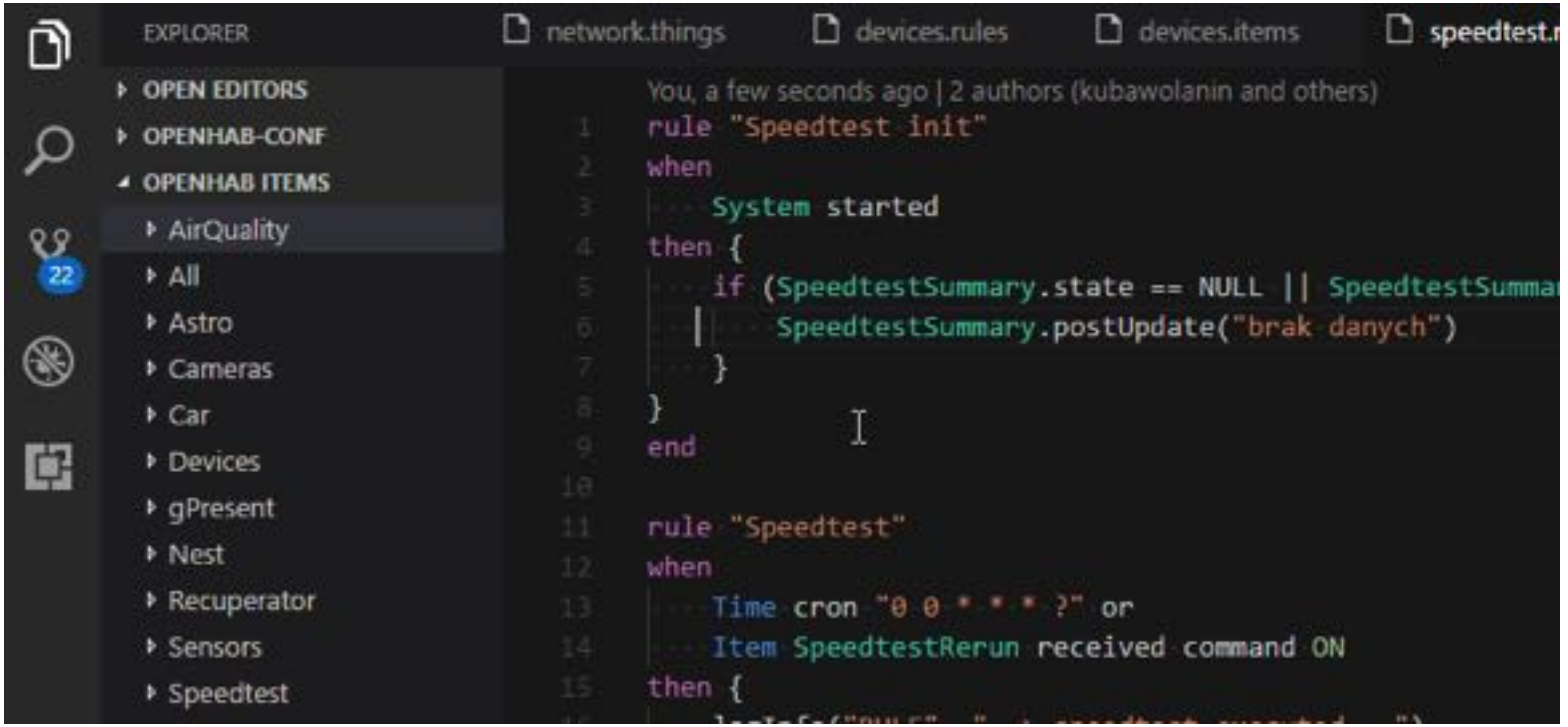
Eclipse Sirius



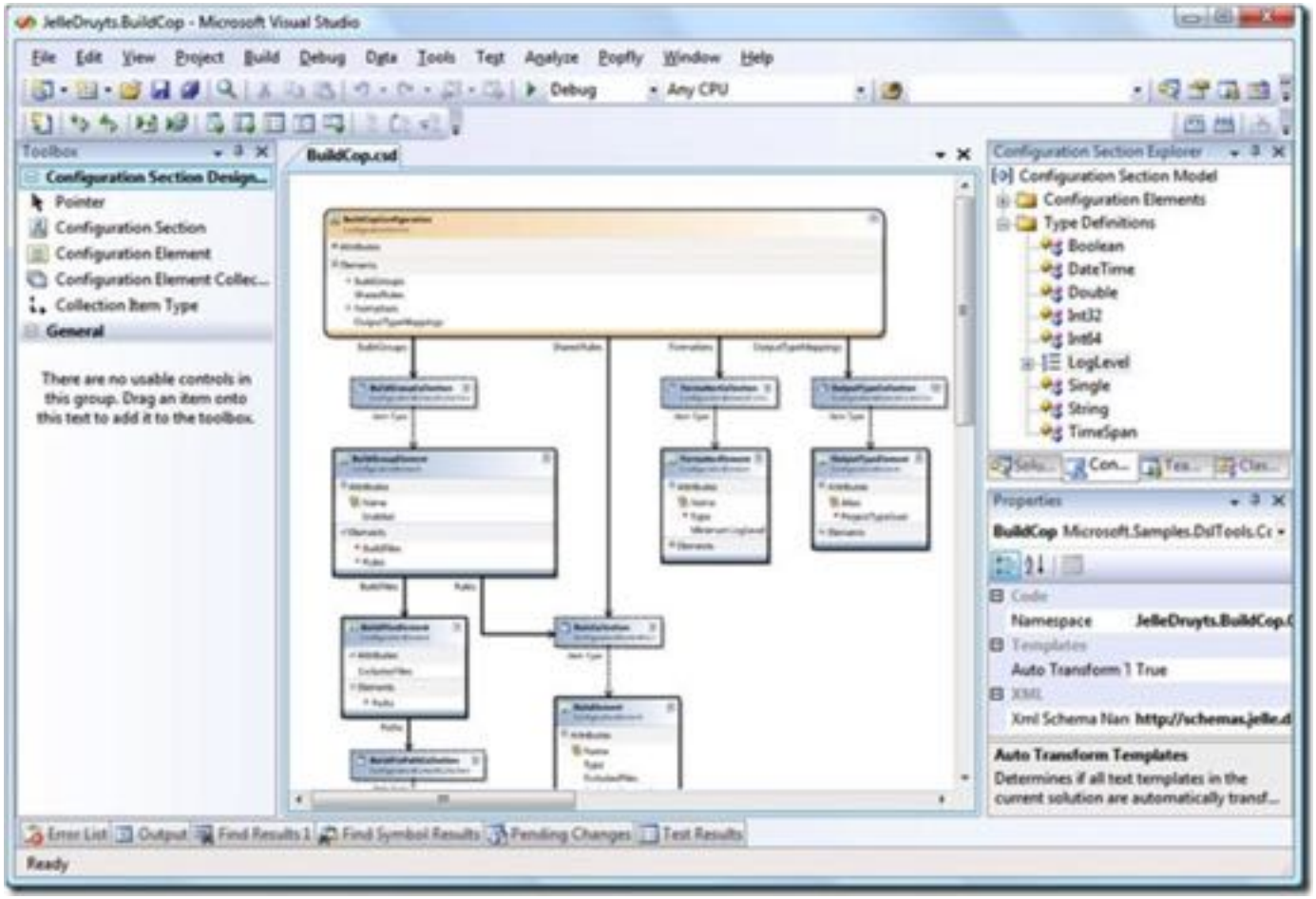
Xtext


Java - my-home/src/Home.rules - Eclipse SDK
Quick Access
Java
Package Ex
my-home
src
Home.rules
JRE System Library
src-gen
Home.rules
1 Device Window can be OPEN, SHUT
2 Device Heating can be ON, OFF
3
4 Rule 'Close Window, when heating turned on'
5 when Heating.ON
6 then Window.SHUT
7
8 Rule 'Switch off heating, when windows gets opened'
9 when Window.OPEN
10 then Heating.OFF
OFF - Heating.OFF
ON - Heating.ON
OPEN - Window.OPEN
SHUT - Window.SHUT
State OFF
Press 'F2' for focus
Outline
Home
Window
Heating
Close Window, when
Switch off heating, w
Problems
Javadoc
Declaration
Search
Console
No consoles to display at this time.
Writable
Insert
10 : 8

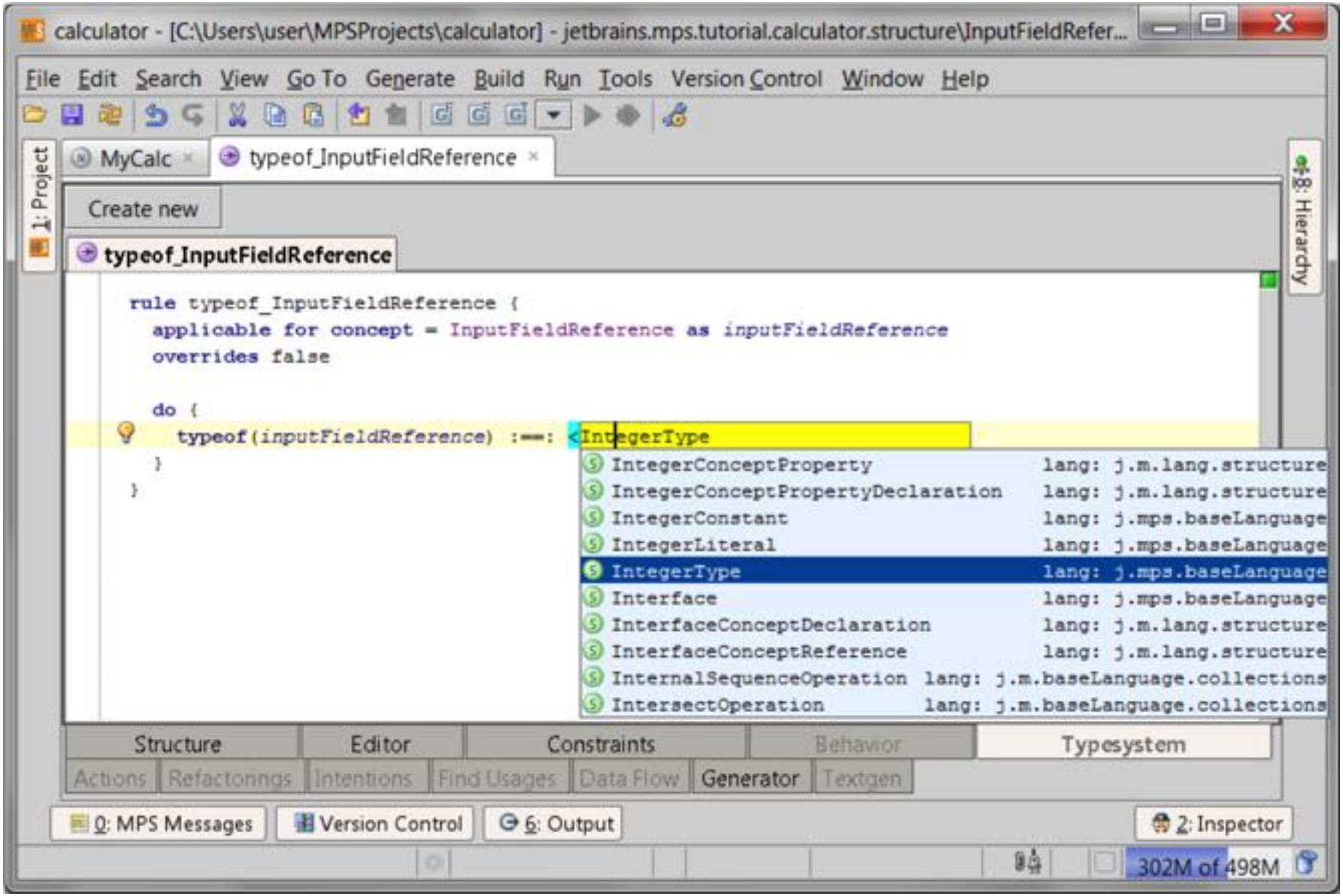
Xtext


EXPLORER
network.things
devices.rules
devices.items
speedtest.r
OPEN EDITORS
OPENHAB-CONF
OPENHAB ITEMS
AirQuality
All
Astro
Cameras
Car
Devices
gPresent
Nest
Recuperator
Sensors
Speedtest
22
You, a few seconds ago | 2 authors (kubawolanin and others)
rule "Speedtest init"
when
System started
then {
if (SpeedtestSummary.state == NULL || SpeedtestSumma
SpeedtestSummary.postUpdate("brak danych")
}
}
end
rule "Speedtest"
when
Time cron "0 0 * * * ?" or
Item SpeedtestRerun received command ON
then {

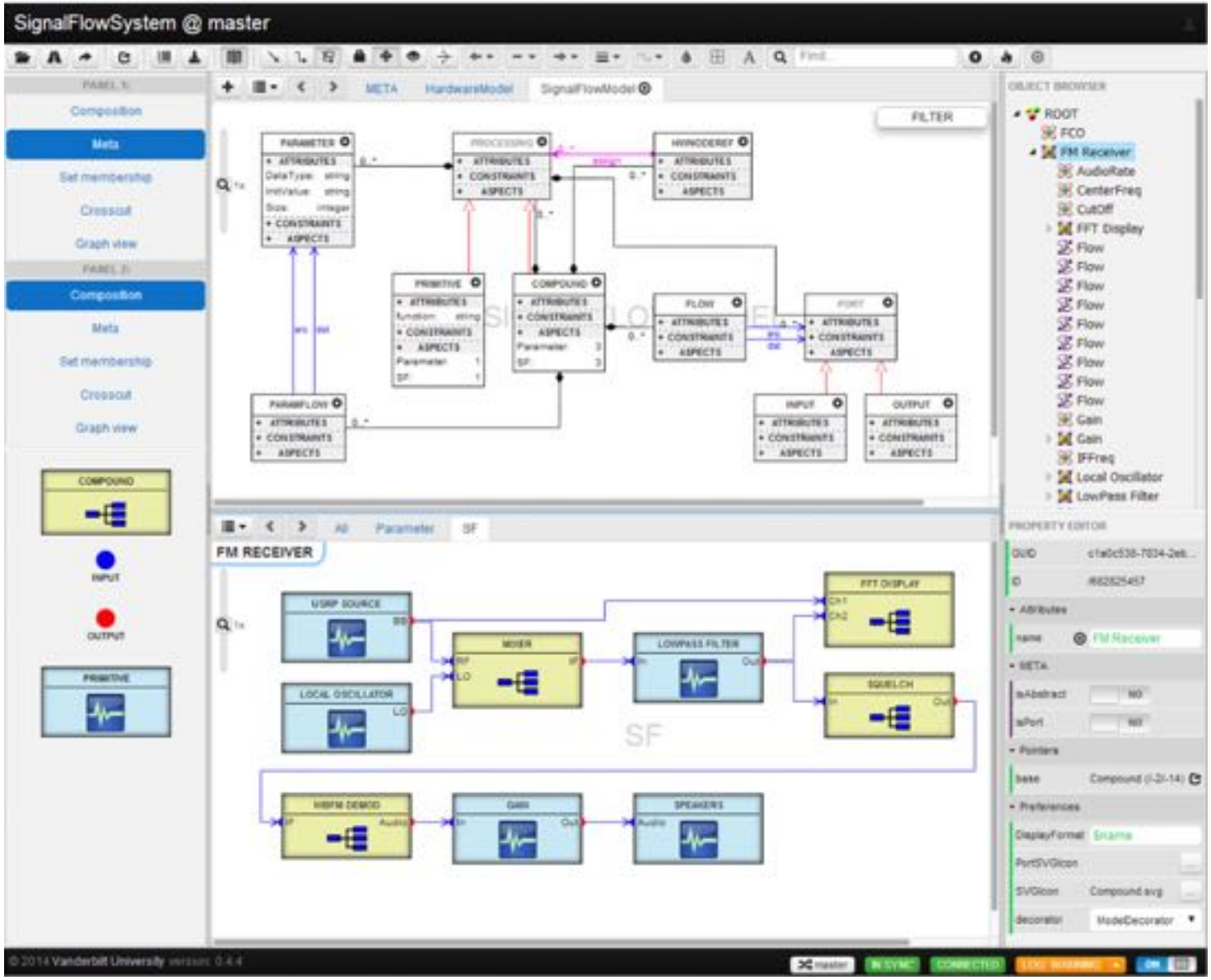
Microsoft DSL Tools



MPS


calculator - [C:\Users\user\MPSProjects\calculator] - jetbrains.mps.tutorial.calculator.structure\InputFieldRefer...
File Edit Search View Go To Generate Build Run Tools Version Control Window Help
MyCalc
typeof_InputFieldReference
Create new
typeof_InputFieldReference
rule typeof_InputFieldReference {
applicable for concept = InputFieldReference as inputFieldReference
overrides false
do {
typeof(inputFieldReference) :==: <IntegerType
}
}
IntegerConceptProperty lang: j.m.lang.structure
IntegerConceptPropertyDeclaration lang: j.m.lang.structure
IntegerConstant lang: j.mps.baseLanguage
IntegerLiteral lang: j.mps.baseLanguage
IntegerType lang: j.mps.baseLanguage
Interface lang: j.mps.baseLanguage
InterfaceConceptDeclaration lang: j.m.lang.structure
InterfaceConceptReference lang: j.m.lang.structure
InternalSequenceOperation lang: j.m.baseLanguage.collections
IntersectOperation lang: j.m.baseLanguage.collections
Structure
Editor
Constraints
Behavior
Typesystem
Actions
Refactorings
Intentions
Find Usages
Data Flow
Generator
Textgen
0: MPS Messages
Version Control
6: Output
2: Inspector
302M of 498M
1: Project
8: Hierarchy

WebGME


SignalFlowSystem @ master
META
HardwareModel
SignalFlowModel
FILTER
OBJECT BROWSER
ROOT
FCO
FM Receiver
AudioRate
CenterFreq
CutOff
FFT Display
Flow
Gain
IFFreq
Local Oscillator
LowPass Filter
Composition
Meta
Set membership
Crosscut
Graph view
COMPOUND
INPUT
OUTPUT
PRIMITIVE
All
Parameter
SF
FM RECEIVER
USRP SOURCE
MIXER
LOWPASS FILTER
FFT DISPLAY
SQUELCH
LOCAL OSCILLATOR
WBFM DEMOD
Gain
SPEAKERS
PROPERTY EDITOR
Attributes
Meta
Pointers
Preferences
© 2014 Vanderbilt University

Summary of DSMs

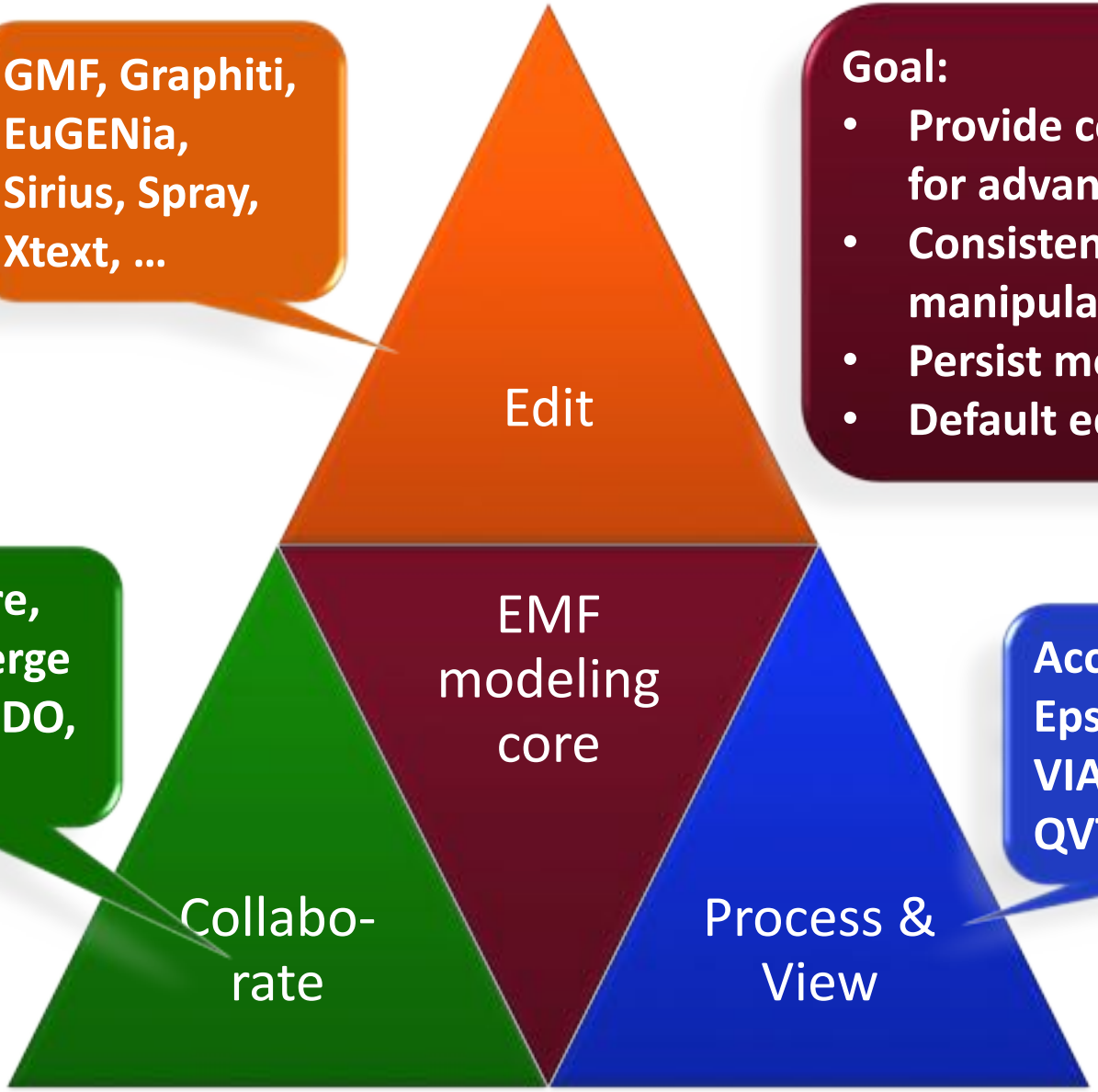
- Metamodeling
  - Structural, formal definition of domains
  - Abstract syntax
- Domain-Specific Modeling
  - Concrete notations
  - Syntax known by experts of the field
- Metalevels
  - Meta-relationship between models
- Semantics
  - Formal dynamic → Denotational / Operational

ECLIPSE MODELING FRAMEWORK

What does EMF provide?

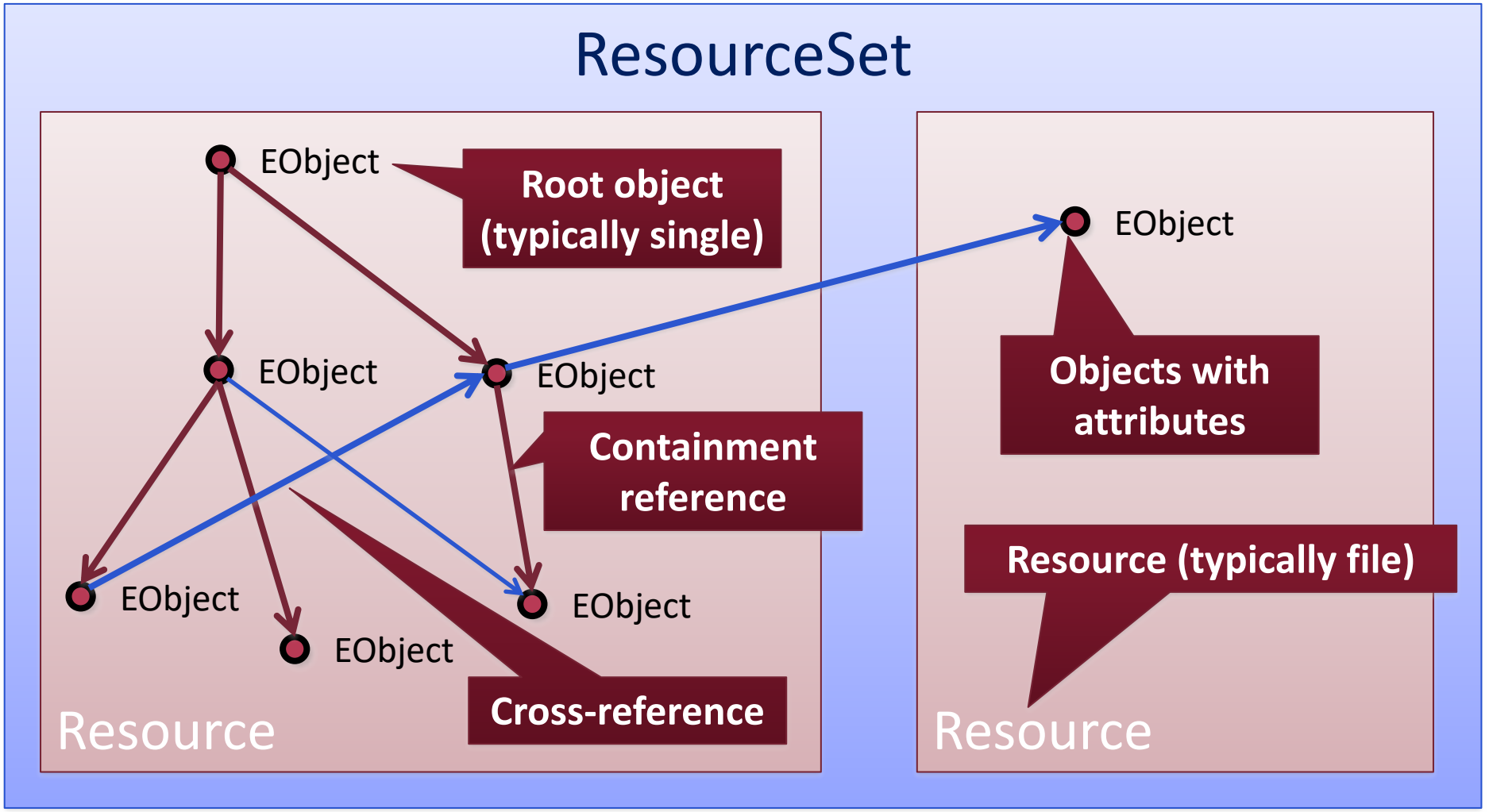
- EMF = Eclipse Modeling Framework
  - Reflective Metamodeling Core (Ecore ➔ MOF 2.0)
  - Support for Domain Specific Languages
  - Editing Support (Notification, Undo, Commands)
  - Basic Editor Support
  - XMI Serialization, DB Persistence
  - Eclipse Integration

Role of EMF/Ecore technology in DSL



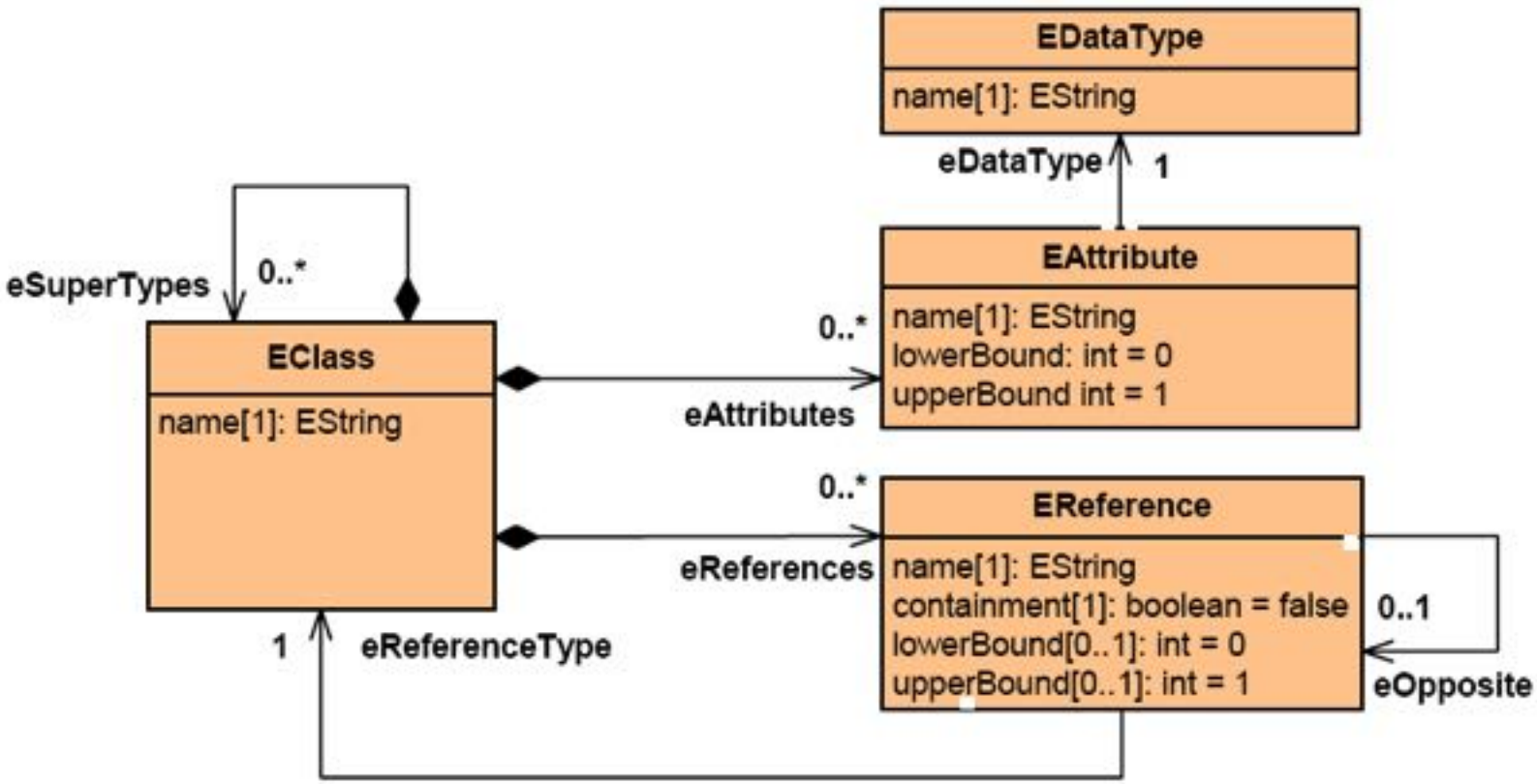
EMF model structure

- Containment hierarchy



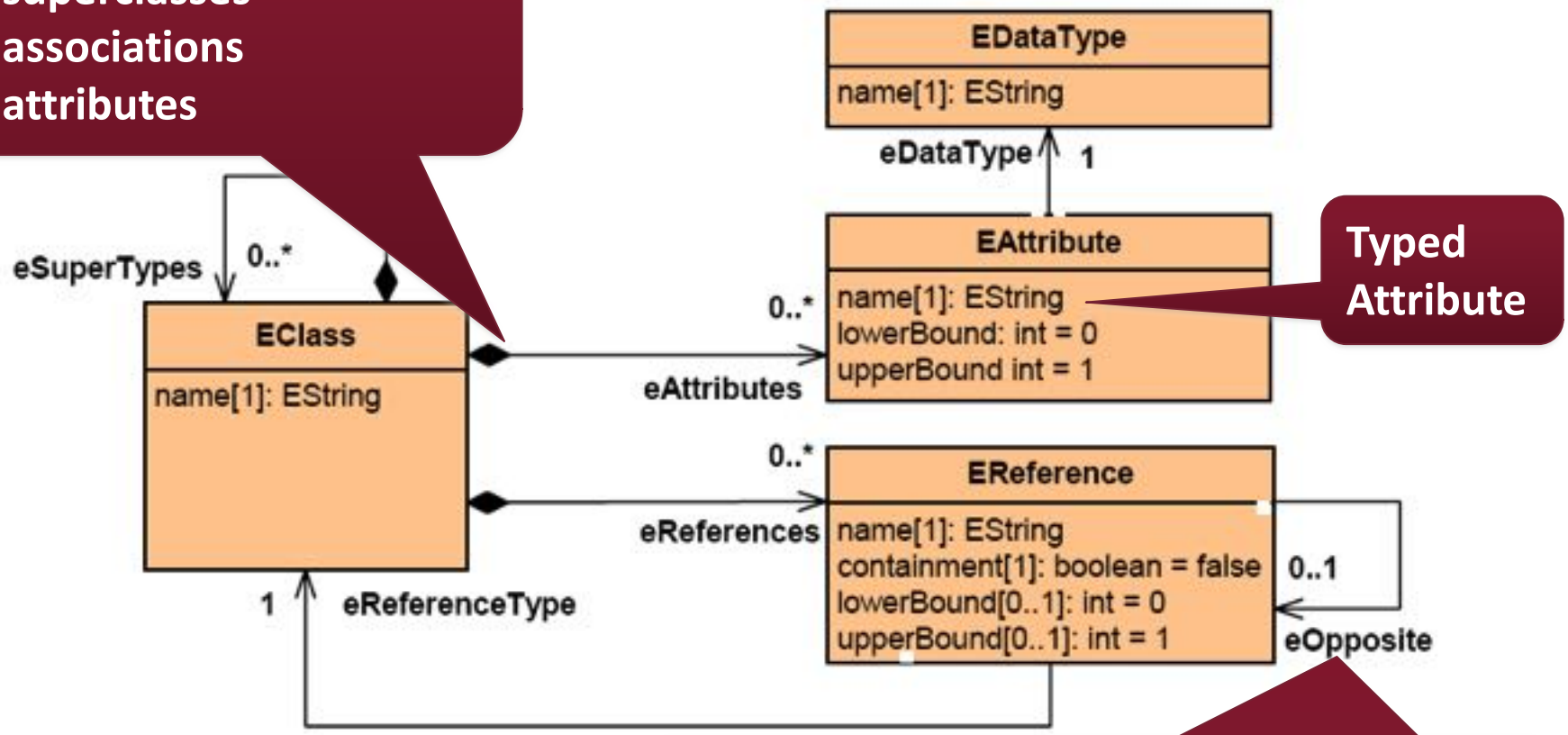
ECORE METAMODELLING

Core Ecore constructs



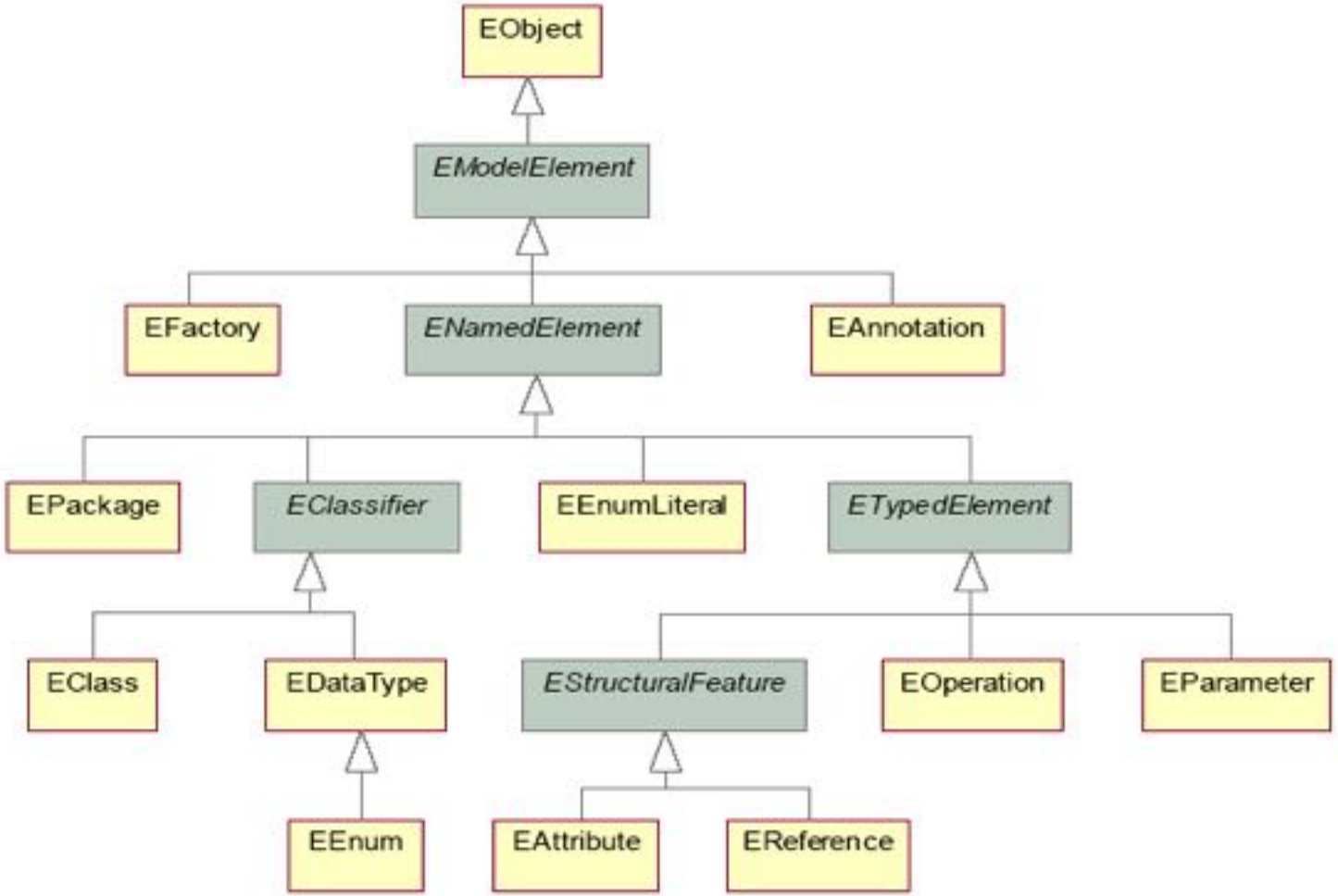
Core Ecore constructs

Class with arbitrary num. of
- **superclasses**
- **associations**
- **attributes**

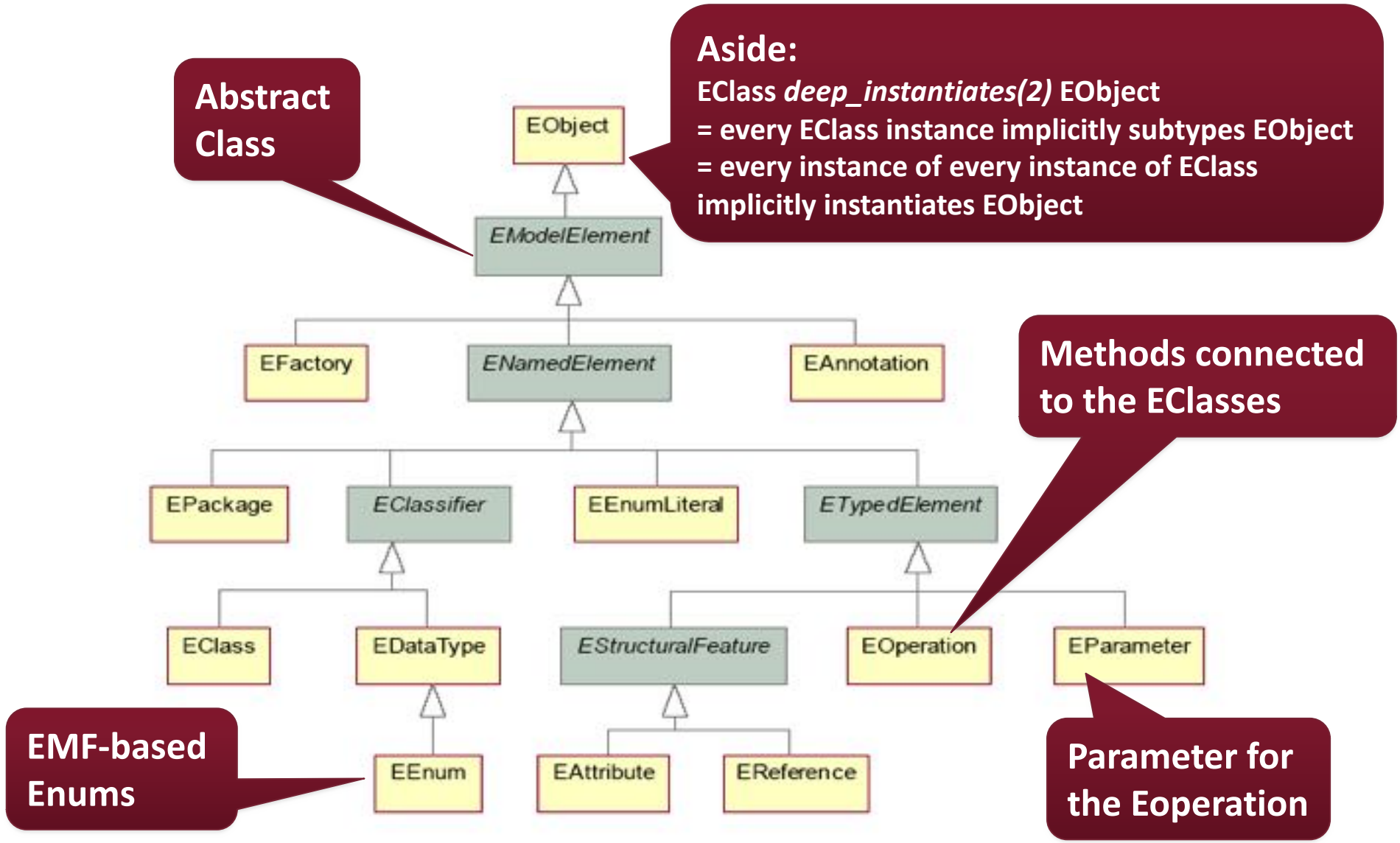


Unidirectional (binary) relation (Association)
- **typed**
- **optional inverse end**
- **multiplicities**

Complete Ecore hiearchy



Complete Ecore hiearchy



DEFINING A DSM
…THE EMF WAY

The Classical EMF/Ecore Waterfall

Design domain metamodel
(Questionnaire.ecore)

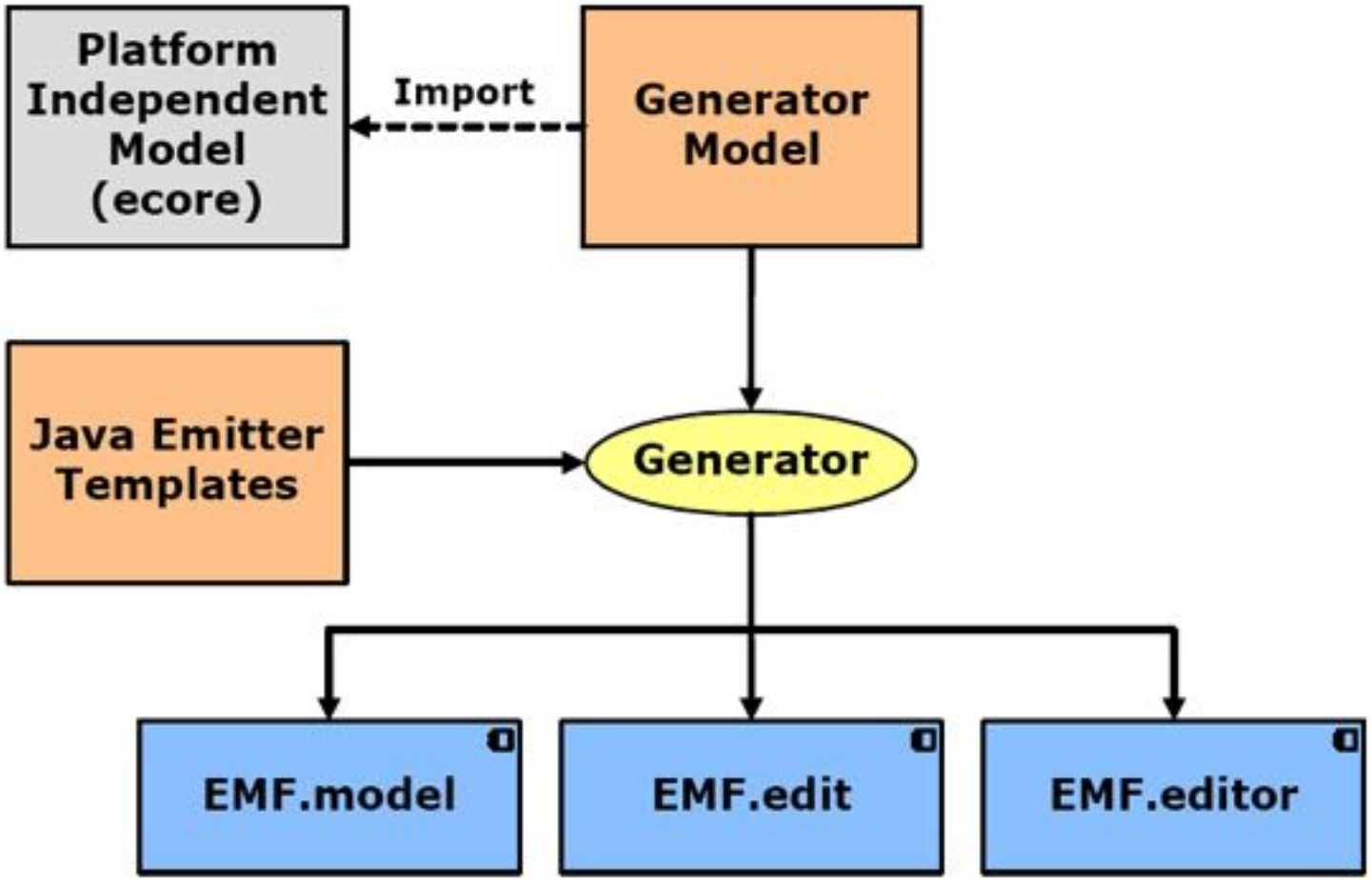
Specify derived features & constraints
(OCL, Epsilon, Viatra Query, Java)

Generate tooling
(Questionnaire.genmodel)

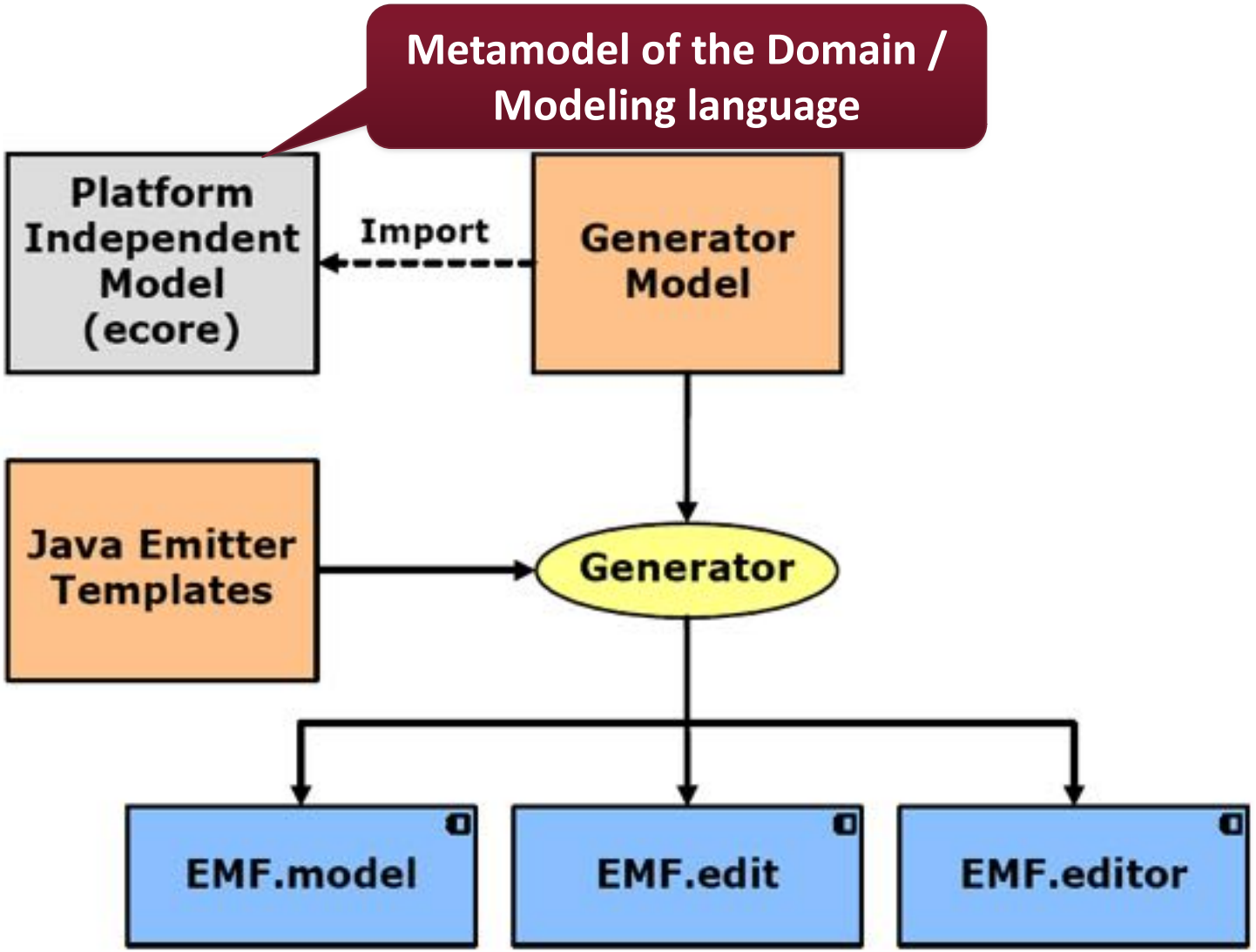
Edit instance models
(Form1.questionnaire)

Validate instance models

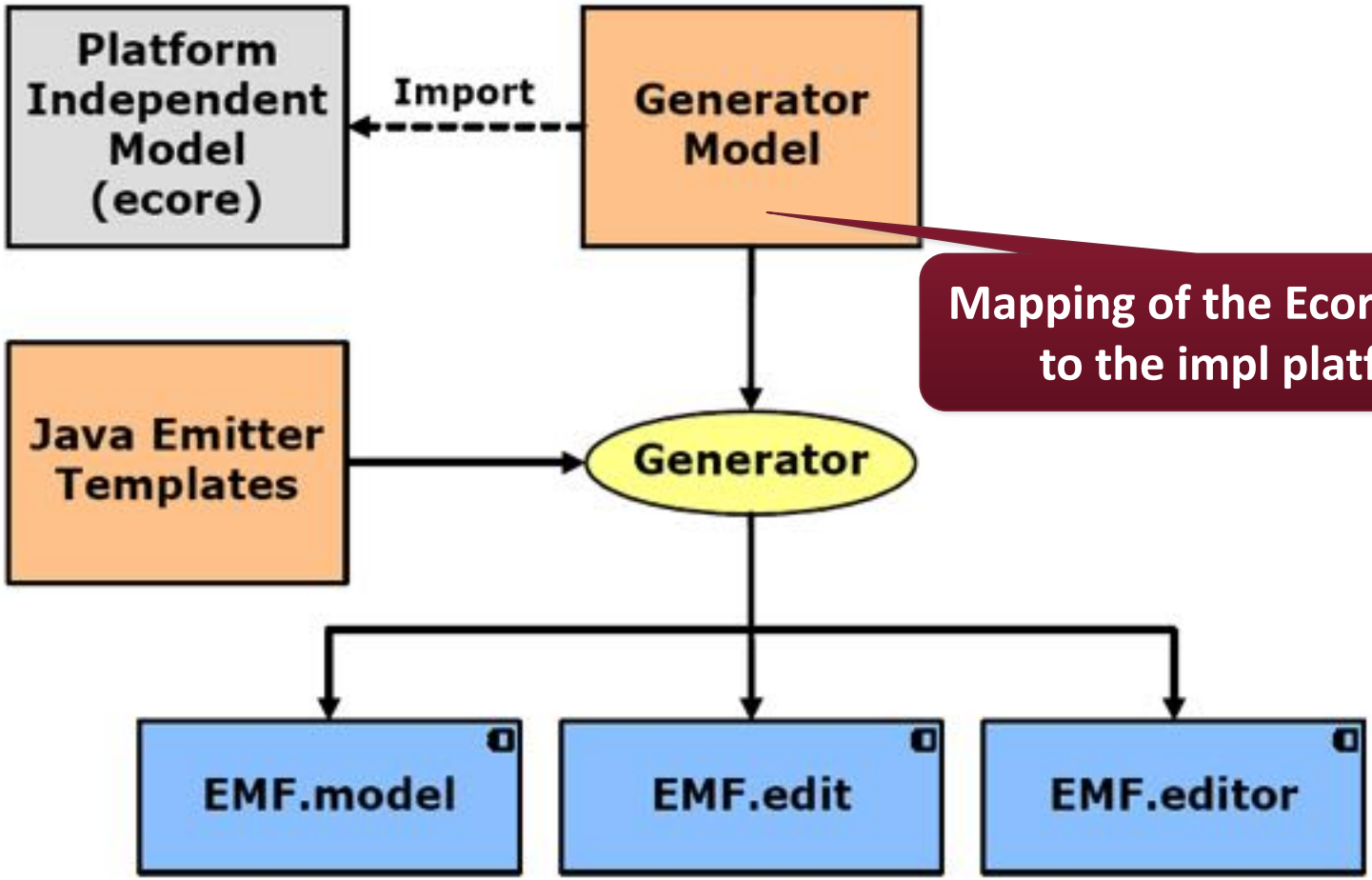
The EMF Toolkit



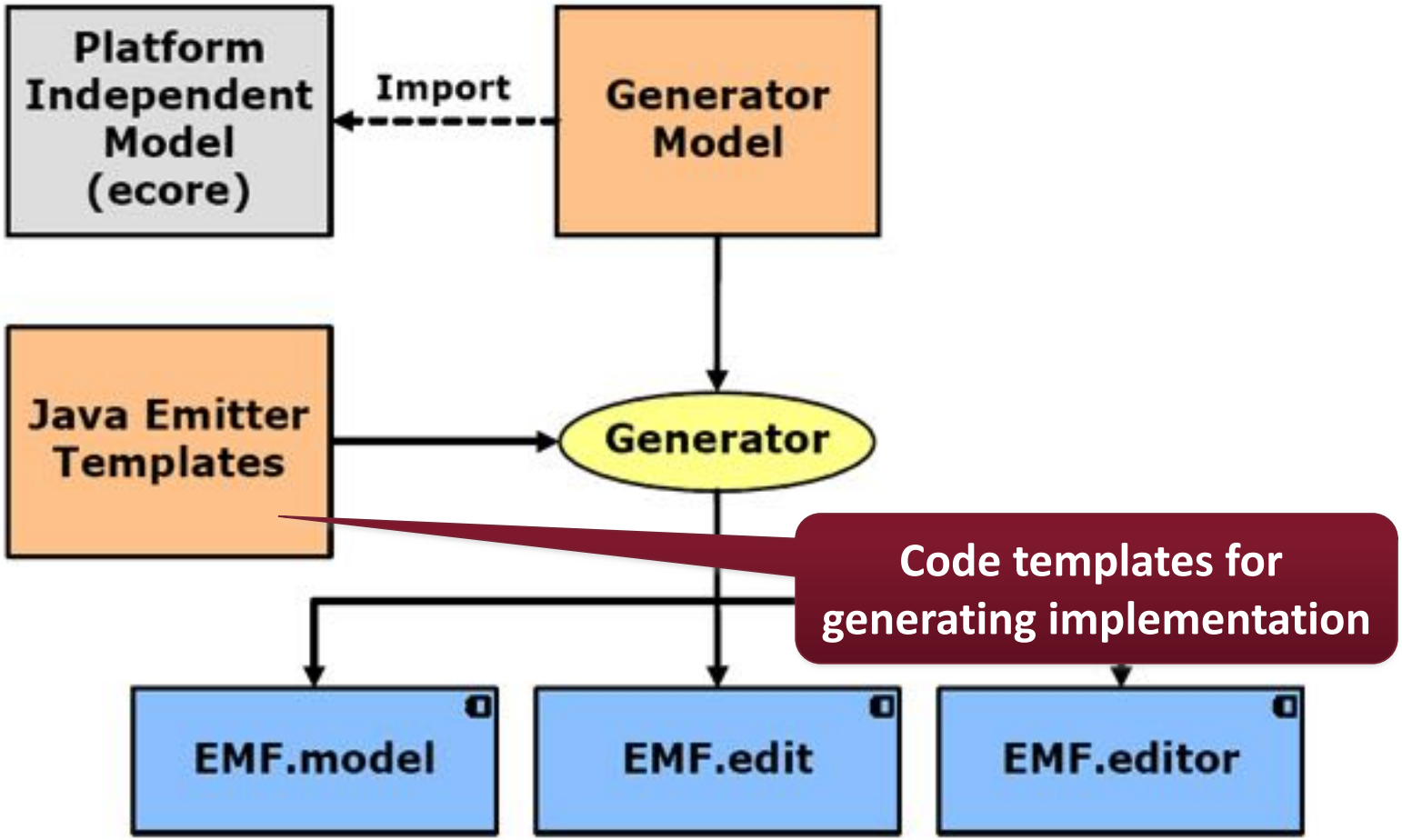
The EMF Toolkit



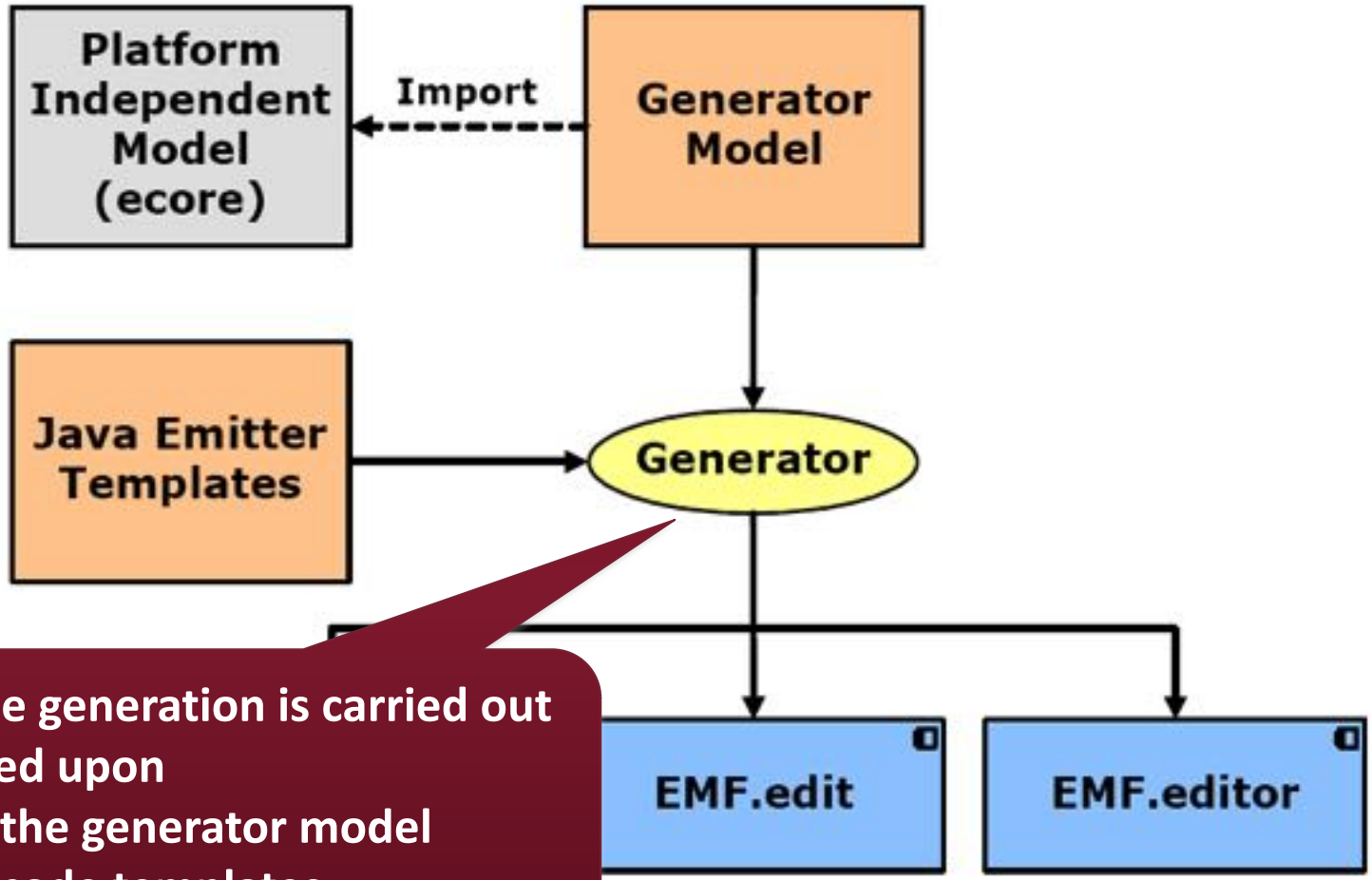
The EMF Toolkit



The EMF Toolkit



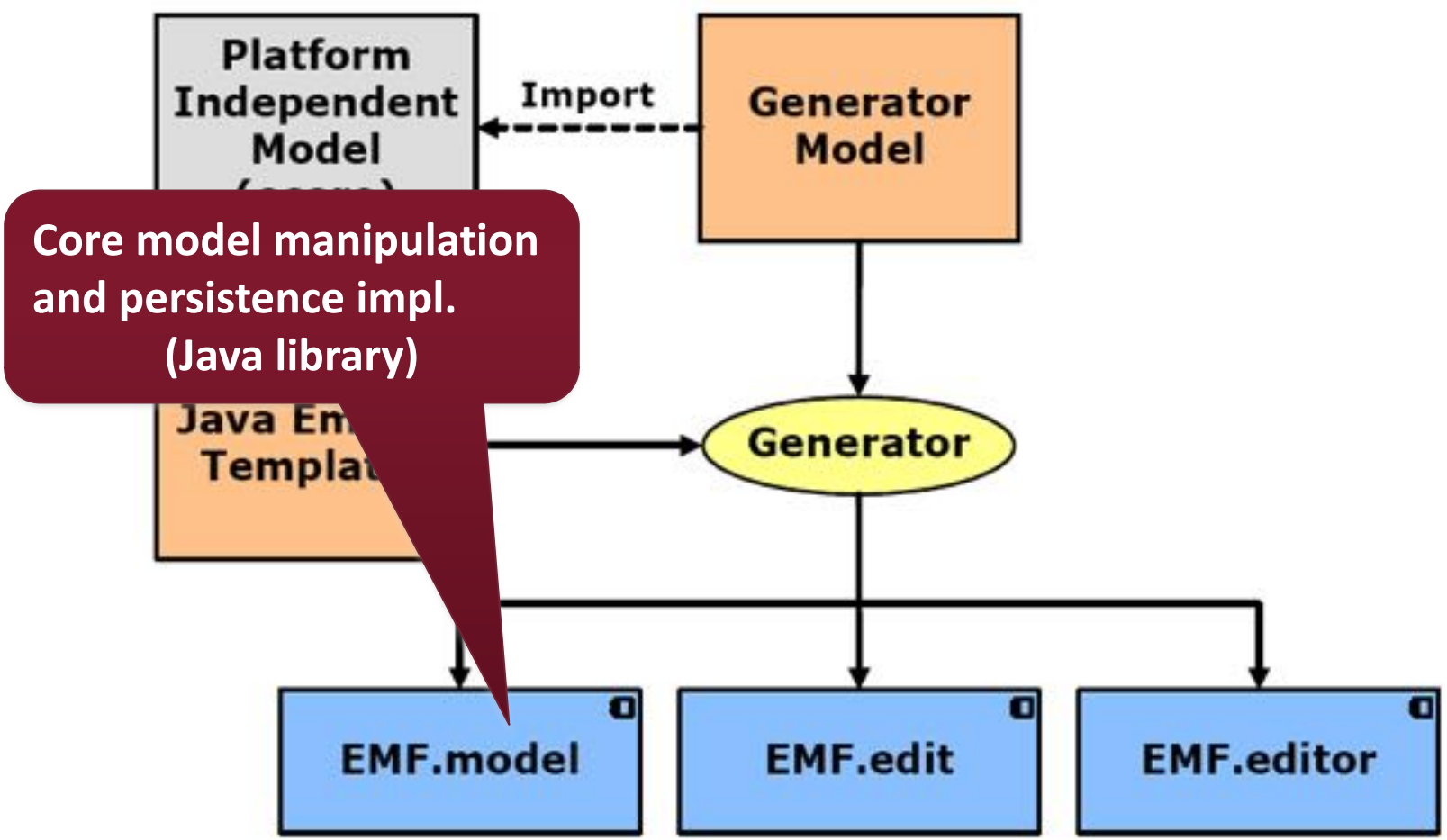
The EMF Toolkit



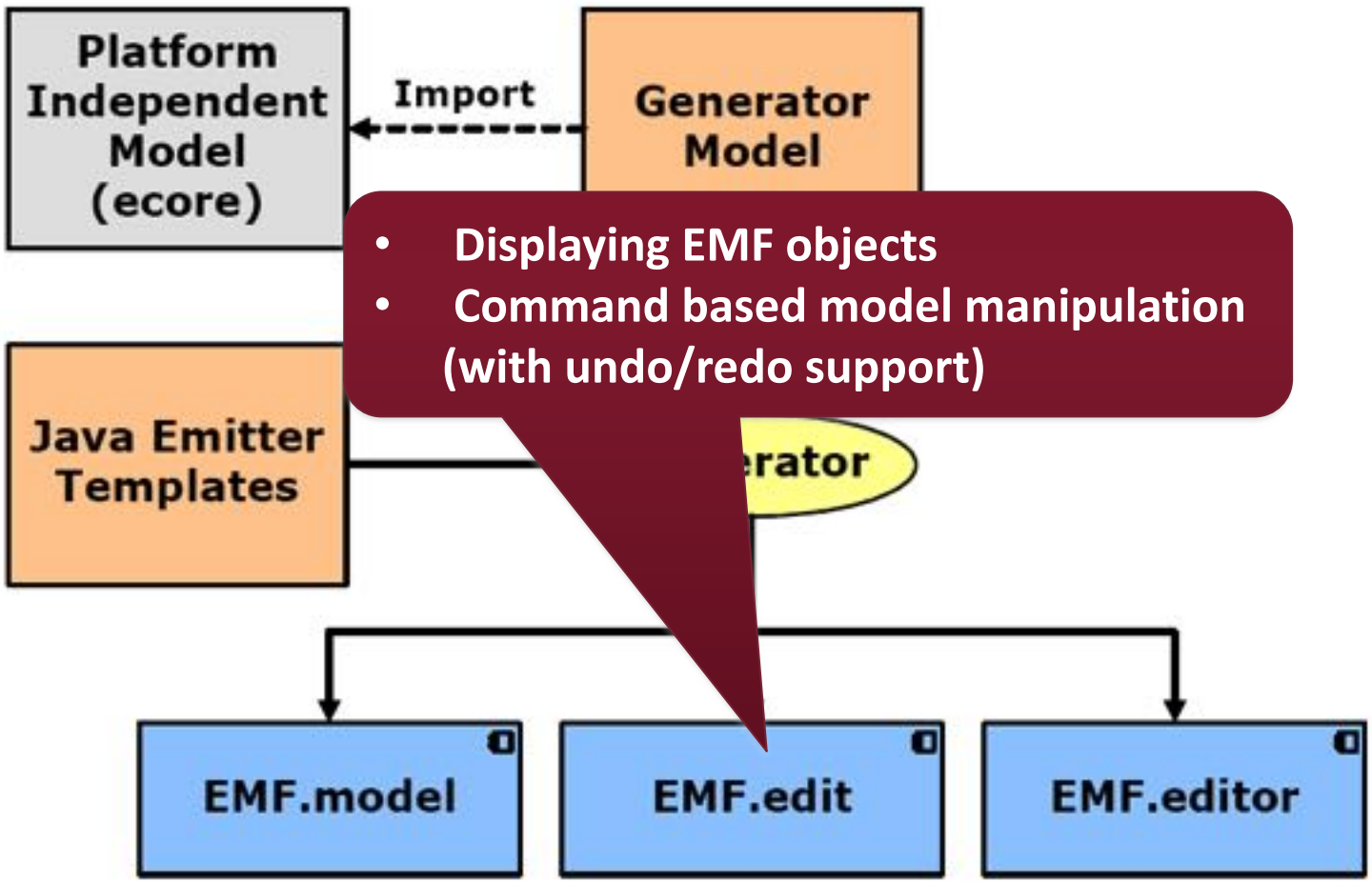
Code generation is carried out based upon

- the generator model
- code templates

The EMF Toolkit

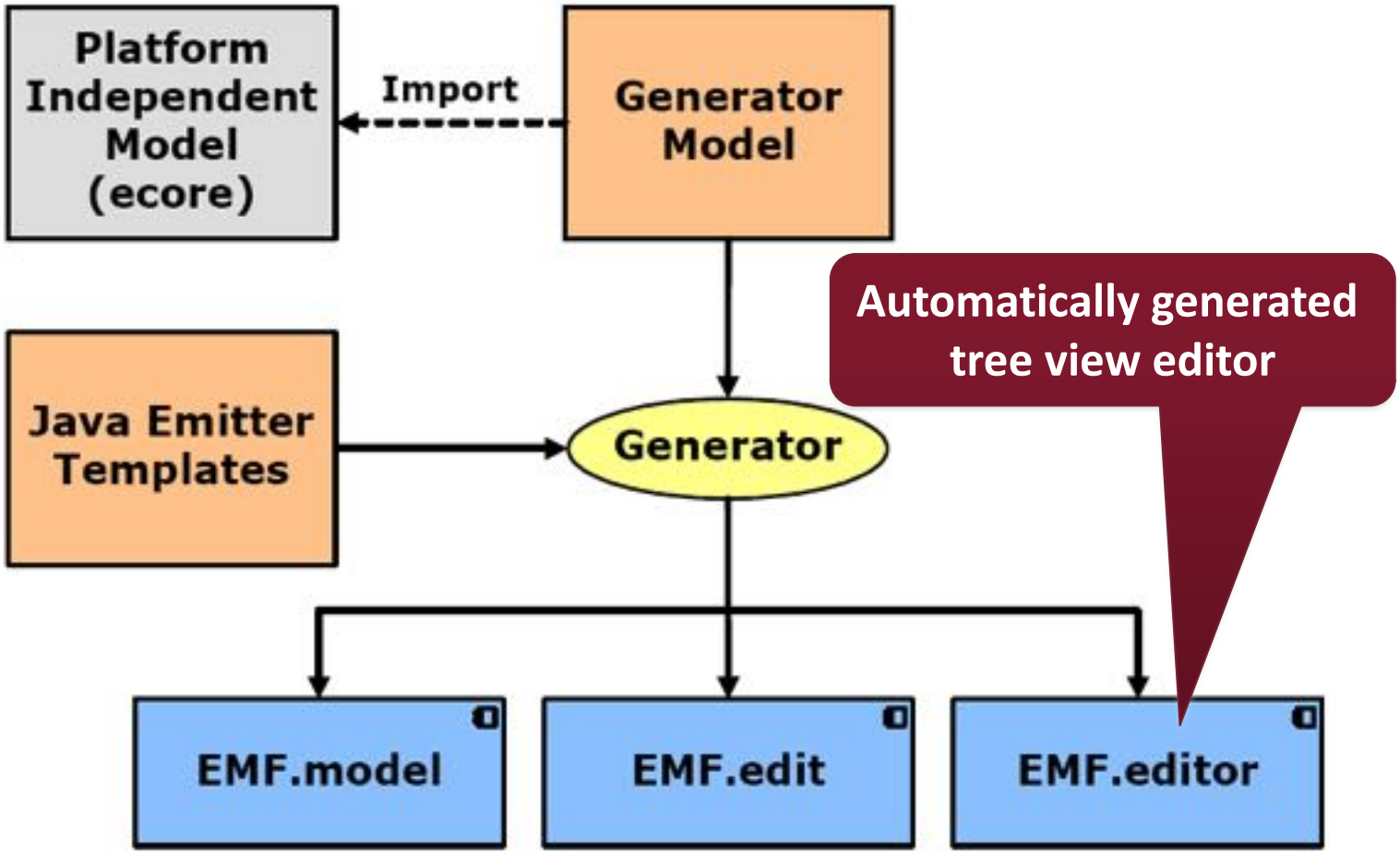


The EMF Toolkit

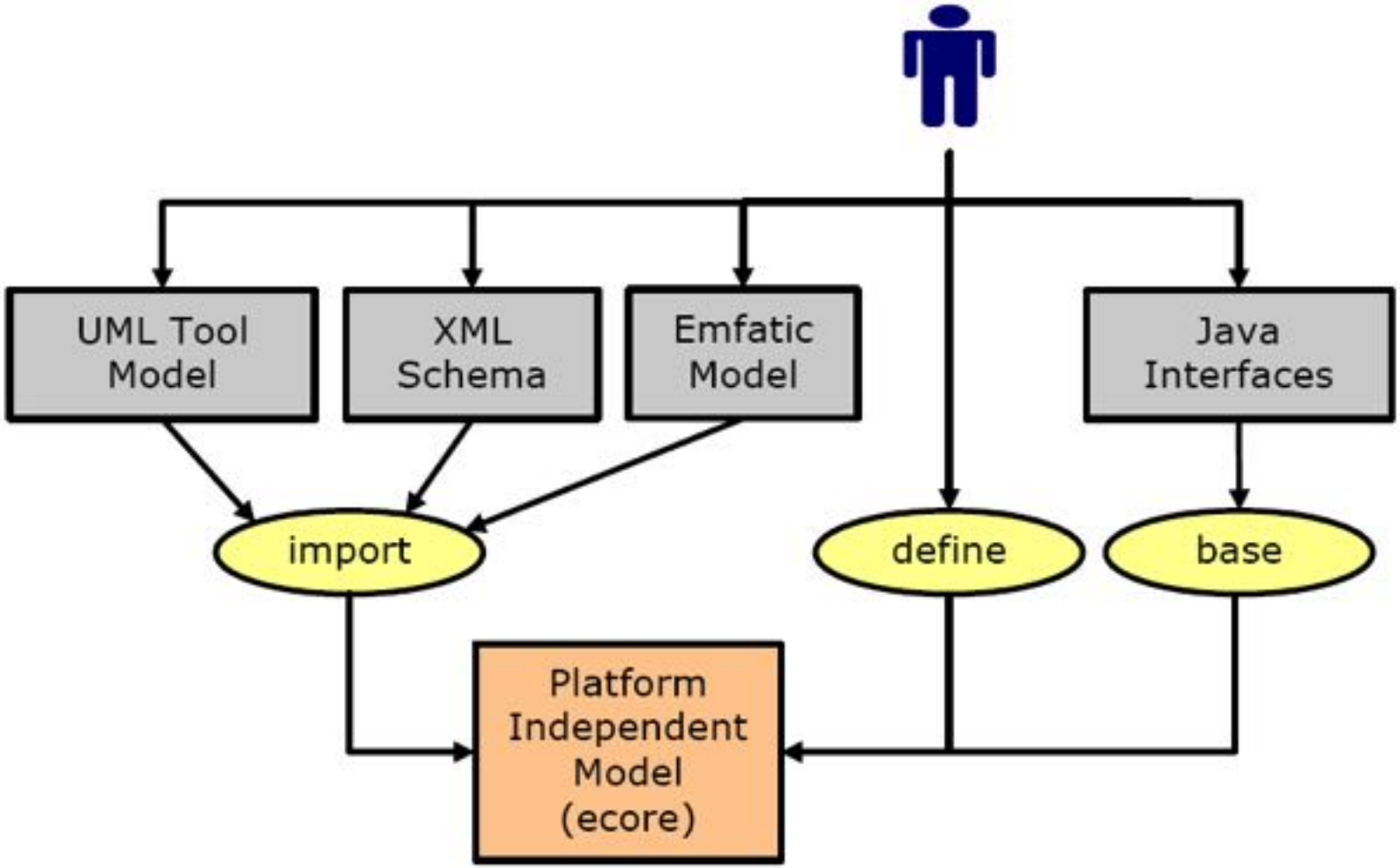


- Displaying EMF objects
- Command based model manipulation (with undo/redo support)

The EMF Toolkit

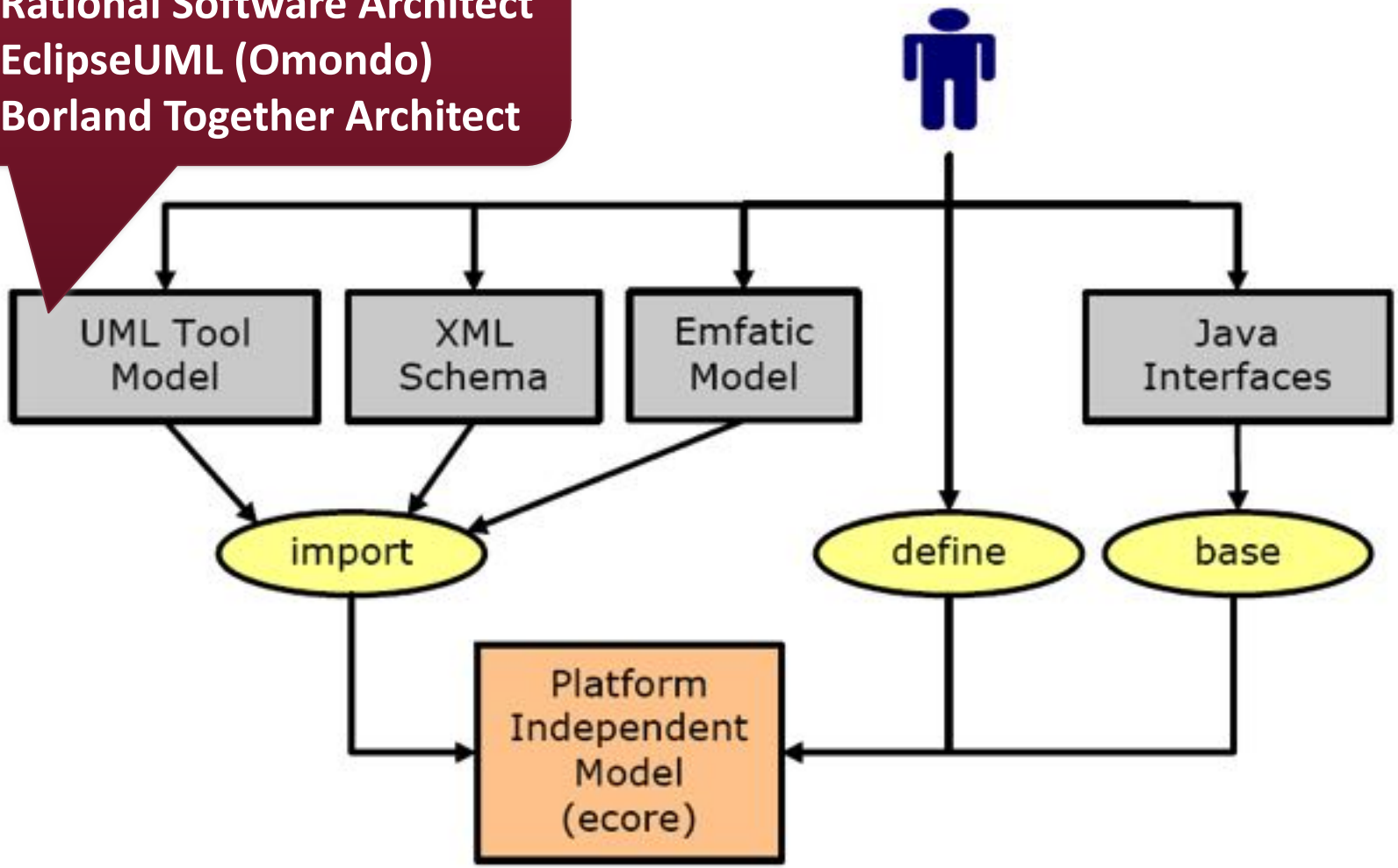


Creation of Ecore metamodels

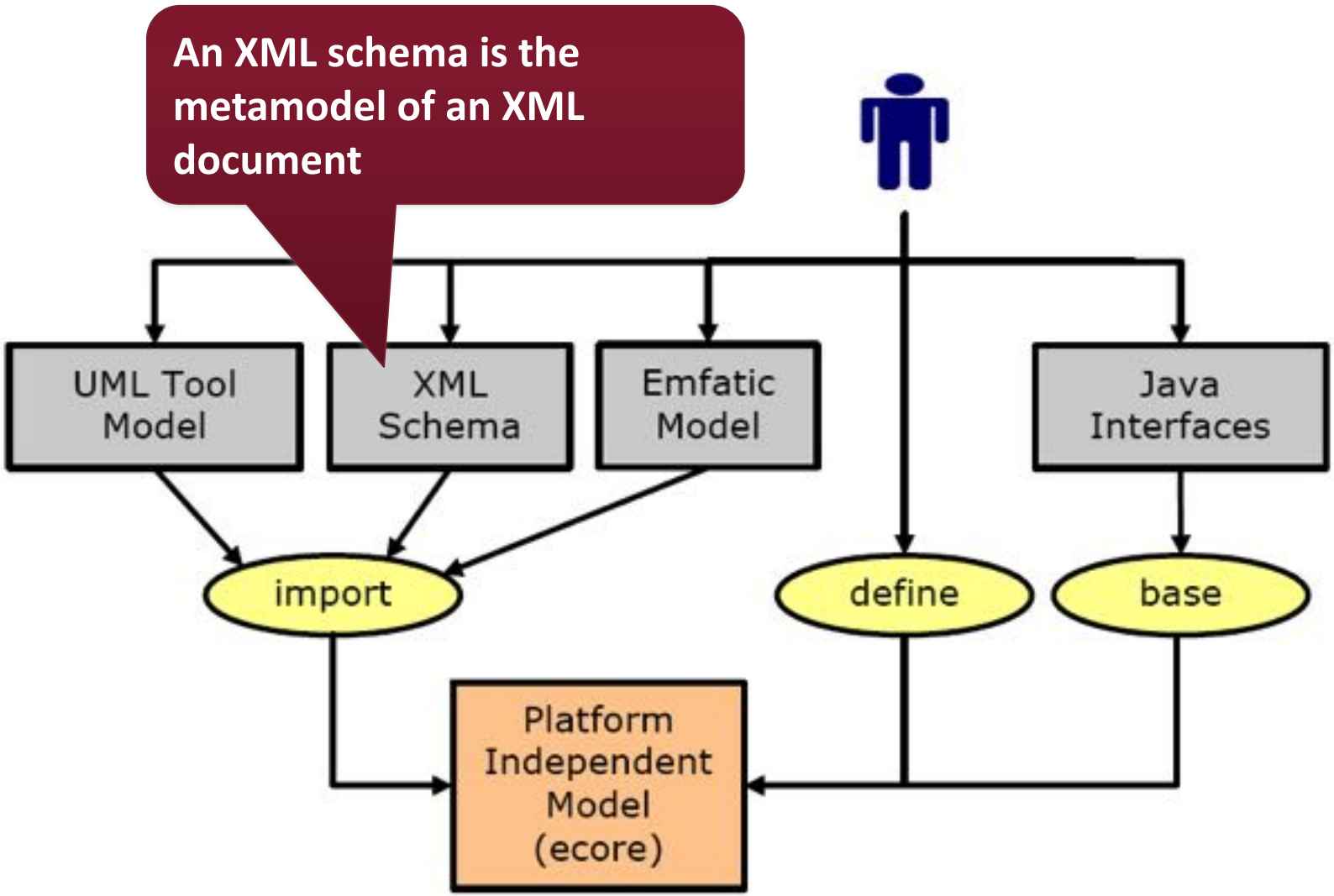


Creation of Ecore metamodels

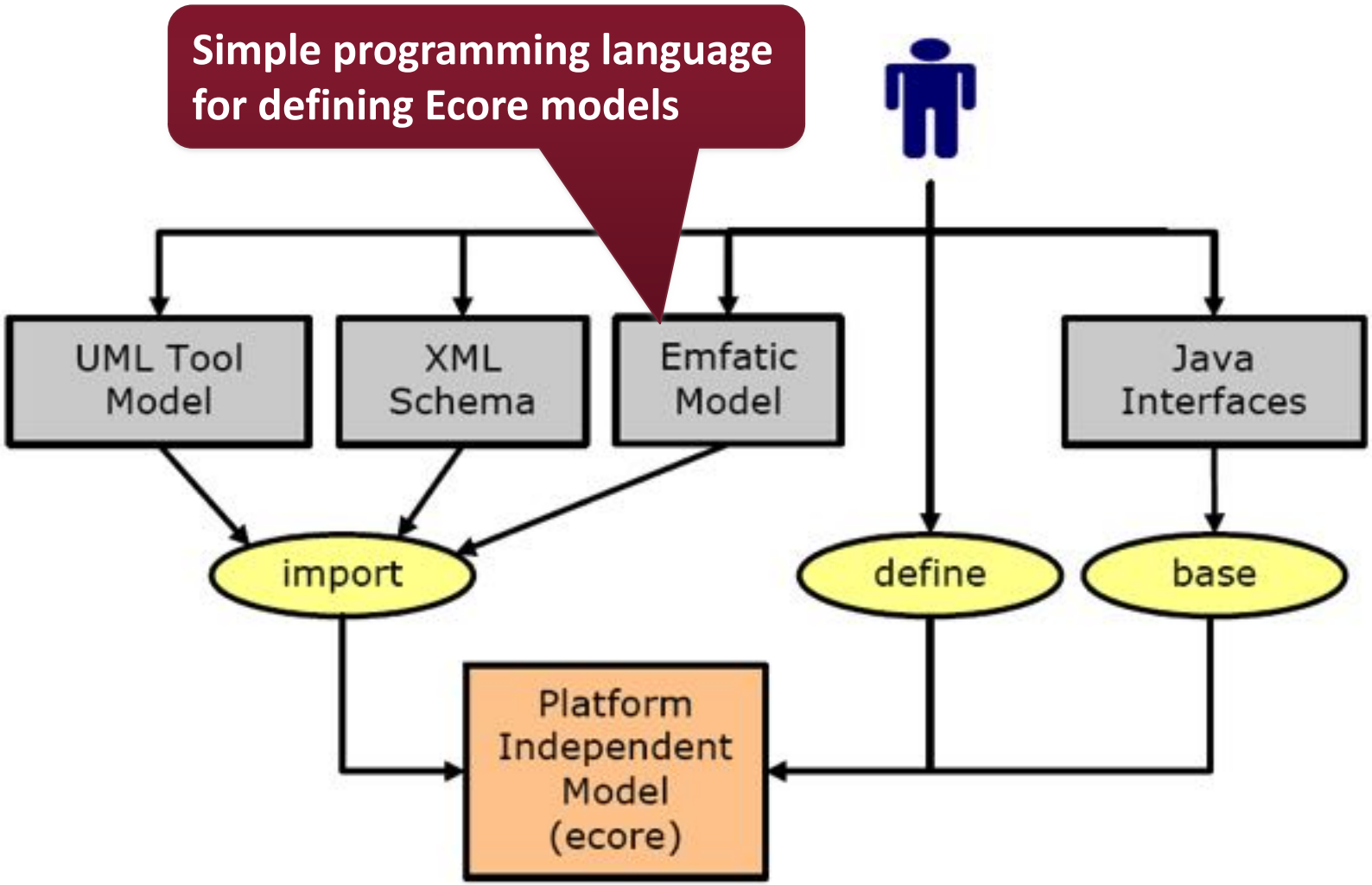
UML class diagram
- **Rational Software Architect**
- **EclipseUML (Omondo)**
- **Borland Together Architect**



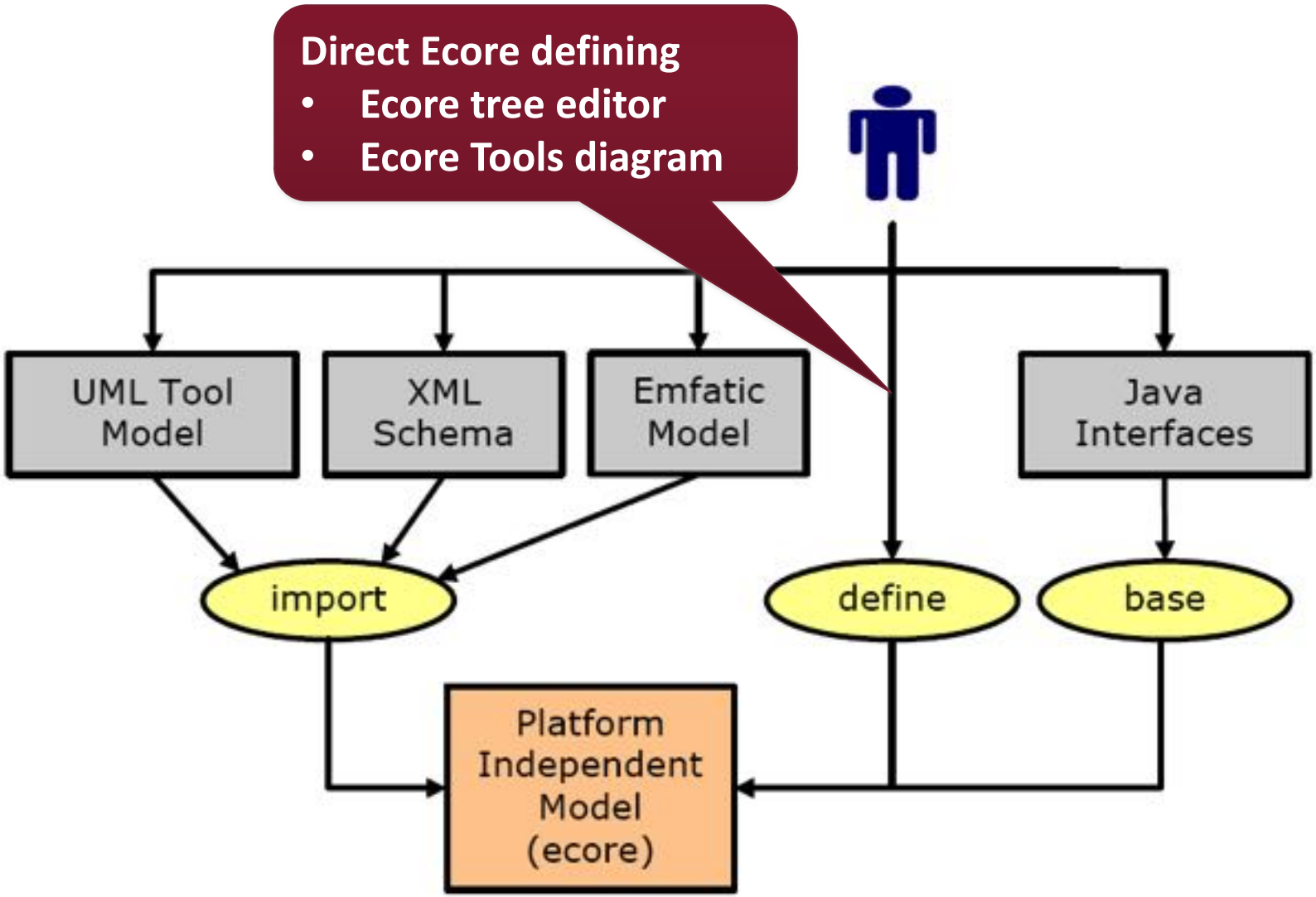
Creation of Ecore metamodels



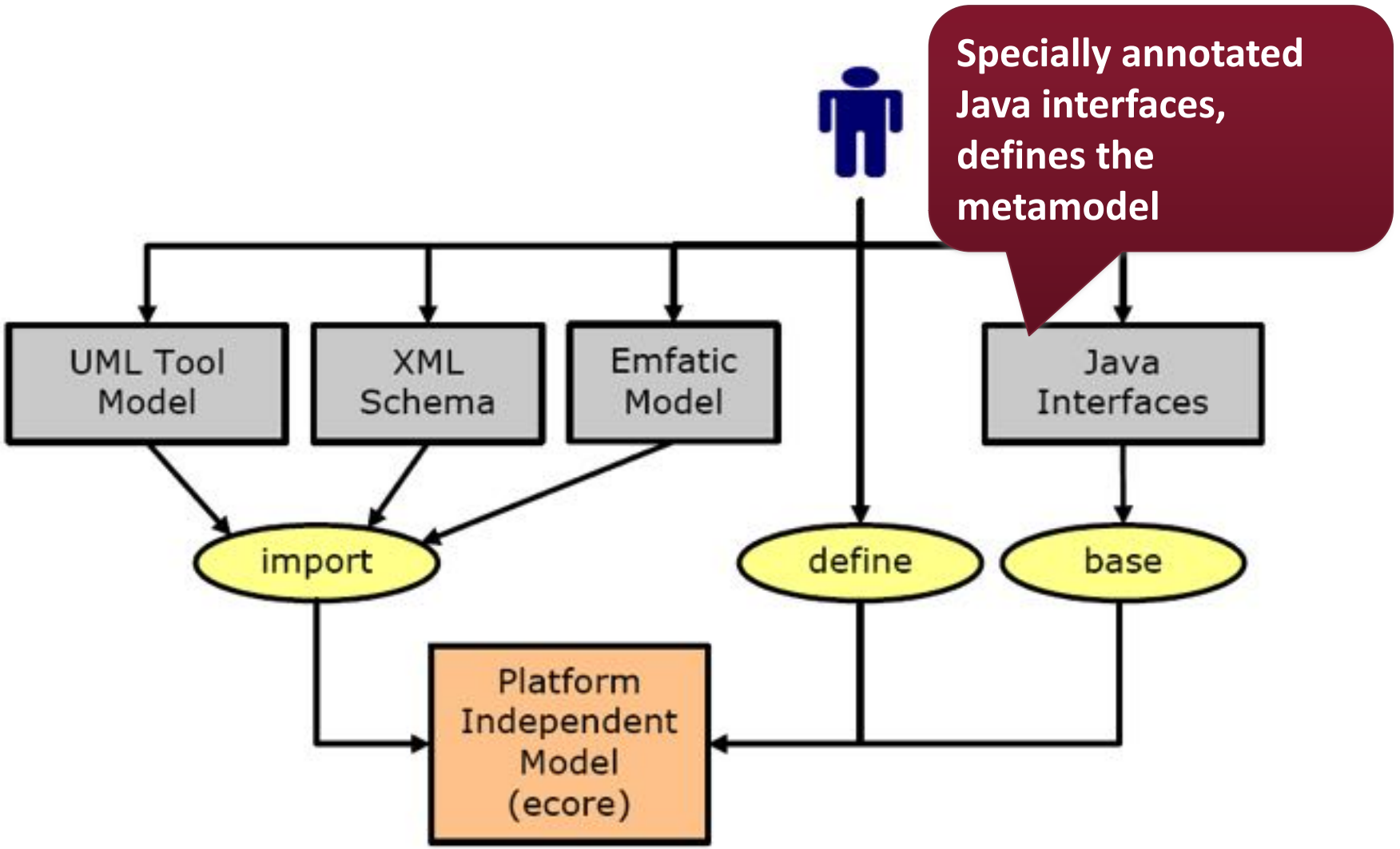
Creation of Ecore metamodels



Creation of Ecore metamodels

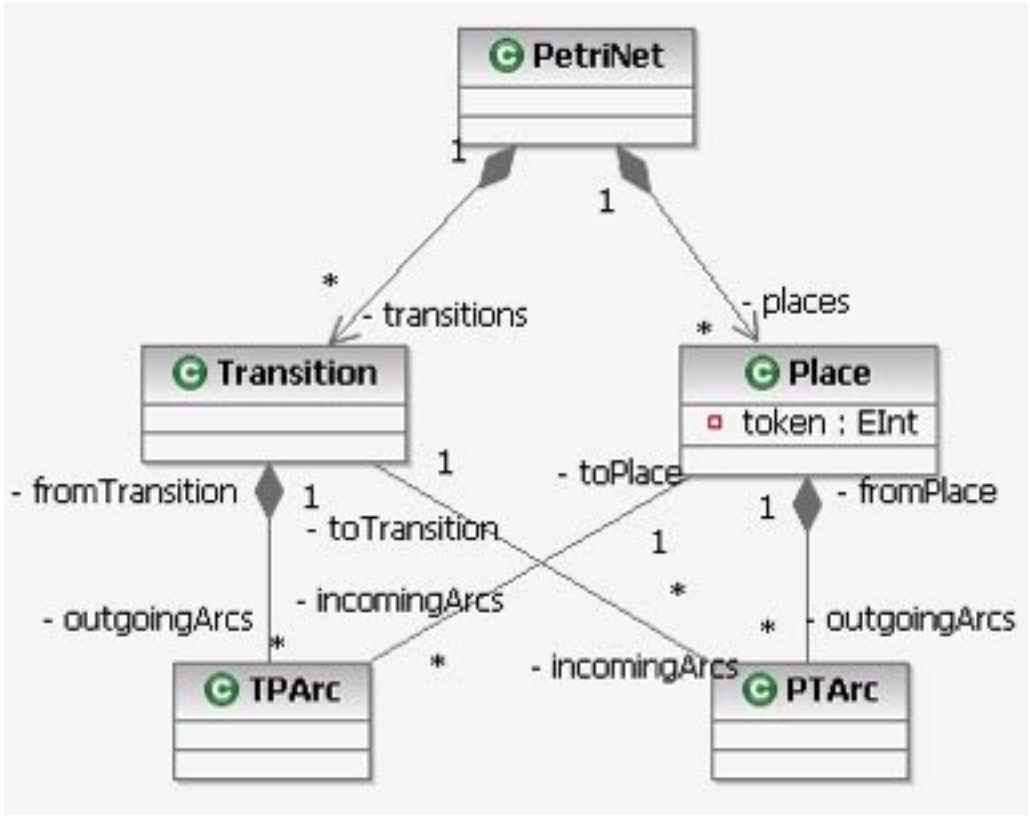


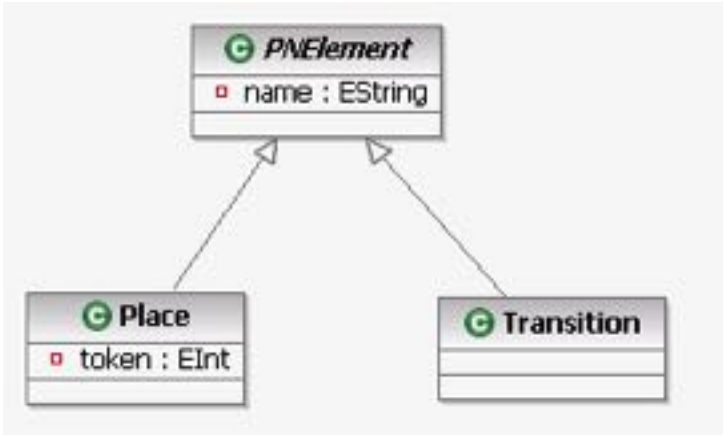
Creation of Ecore metamodels

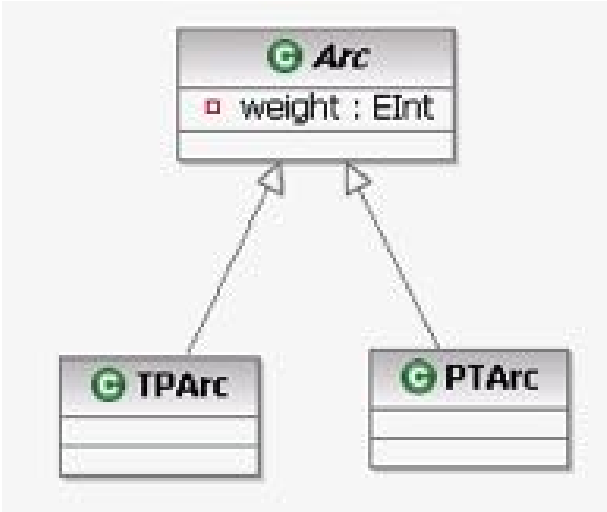


THE PETRI NET EXAMPLE

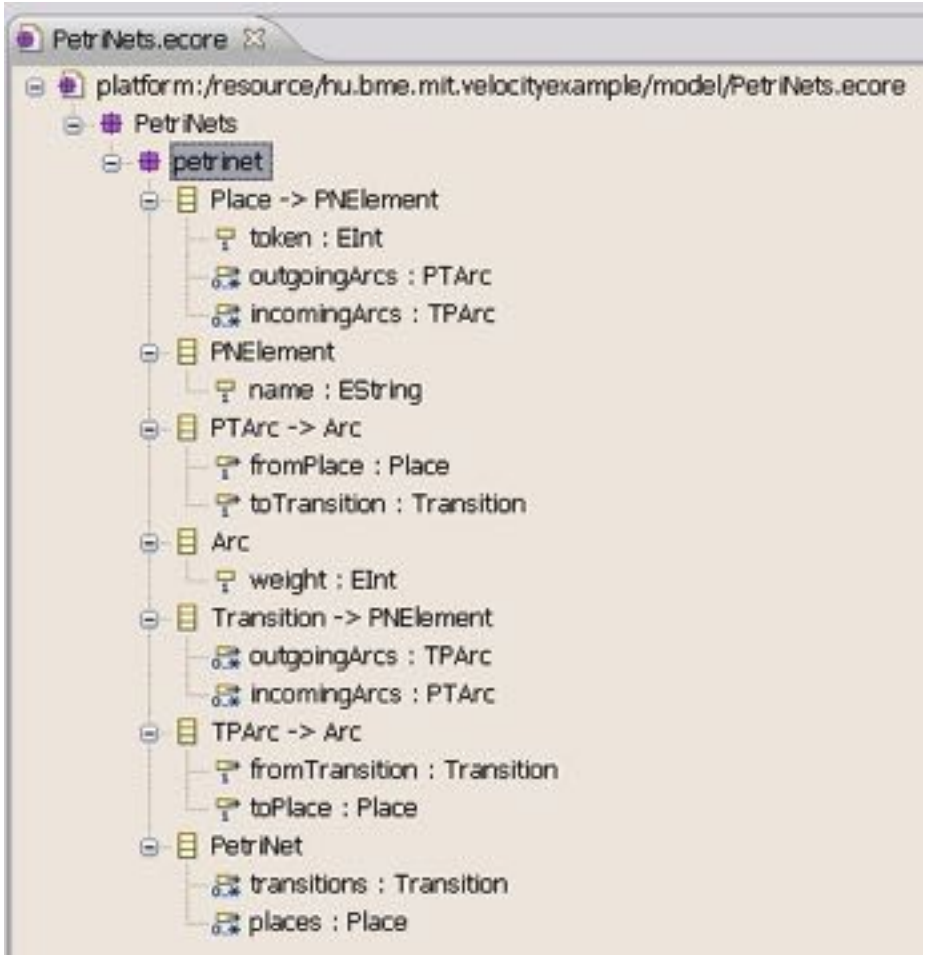
Domain Metamodel: Petri Nets



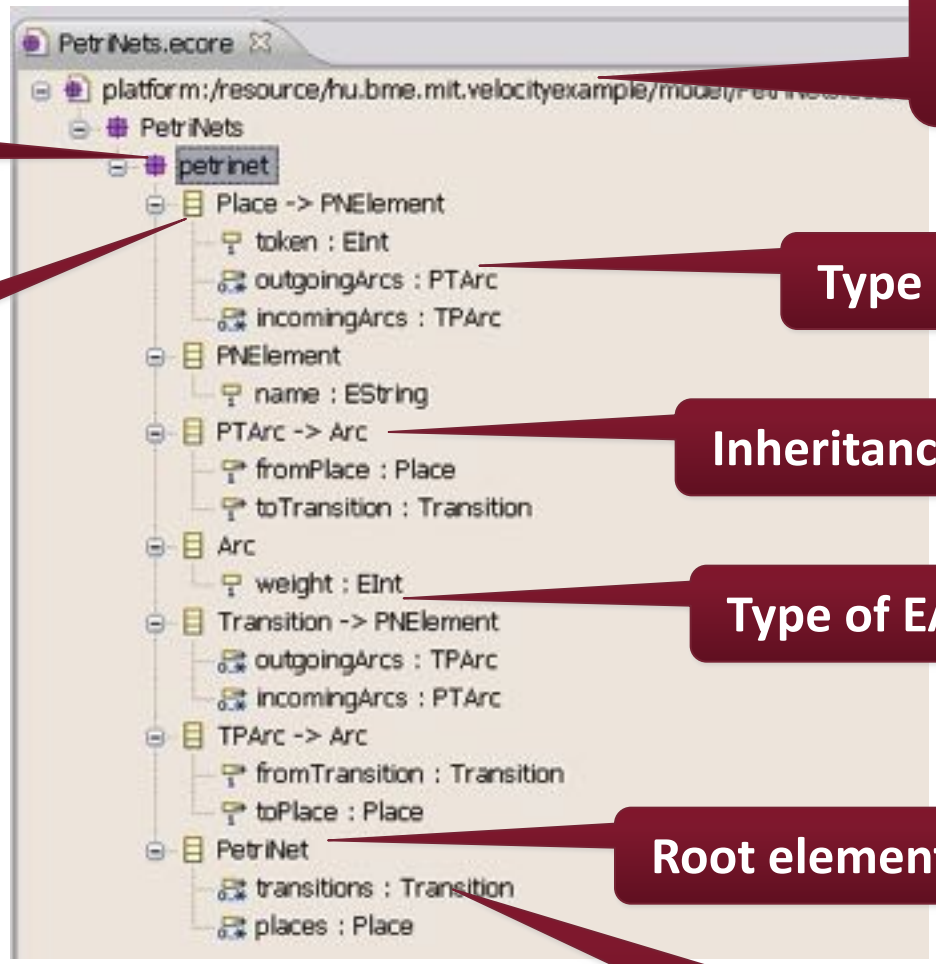




EMF model Ecore representation



EMF model Ecore representation



Path of containing resource

EPackage

EClass

Type of EReference

Inheritance

Type of EAttribute

Root element

Reference to all model elements

Class Definition in PetriNet.ecore

```
<eClassifiers xsi:type="ecore:EClass" name="Place"
    eSuperTypes="#//petrinet/PNElement">

 <eStructuralFeatures xsi:type="ecore:EAttribute" name="token" lowerBound="1"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>

 <eStructuralFeatures xsi:type="ecore:EReference" name="outgoingArcs"
    upperBound="-1"
    eType="#//petrinet/PTArc" containment="true"
    eOpposite="#//petrinet/PTArc/fromPlace"/>

 <eStructuralFeatures xsi:type="ecore:EReference" name="incomingArcs"
    upperBound="-1"
    eType="#//petrinet/TPArc" eOpposite="#//petrinet/TPArc/toPlace"/>
</eClassifiers>
```

Class Definition in PetriNet.ecore

```
<eClassifiers xsi:type="ecore:EClass" name="Place"
    eSuperTypes="#//petrinet/PNElement">

 <eStructuralFeatures xsi:type="ecore:EAttribute" name="token" lowerBound="1"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>

 <eStructuralFeatures xsi:type="ecore:EReference" name="outgoingArcs"
    upperBound="-1"
    eType="#//petrinet/PTArc" containment="true"
    eOpposite="#//petrinet/P

 <eStructuralFeatures xsi:type="ecore:EReference" name="incomingArcs"
    upperBound="-1"
    eType="#//petrinet/TPArc" eOpposite="#//petrinet/TPArc/toPlace"/>
</eClassifiers>
```

Class

Attribute

Reference

Multiplicity

Containment

Type

Opposite End

CODE GENERATION FROM ECORE

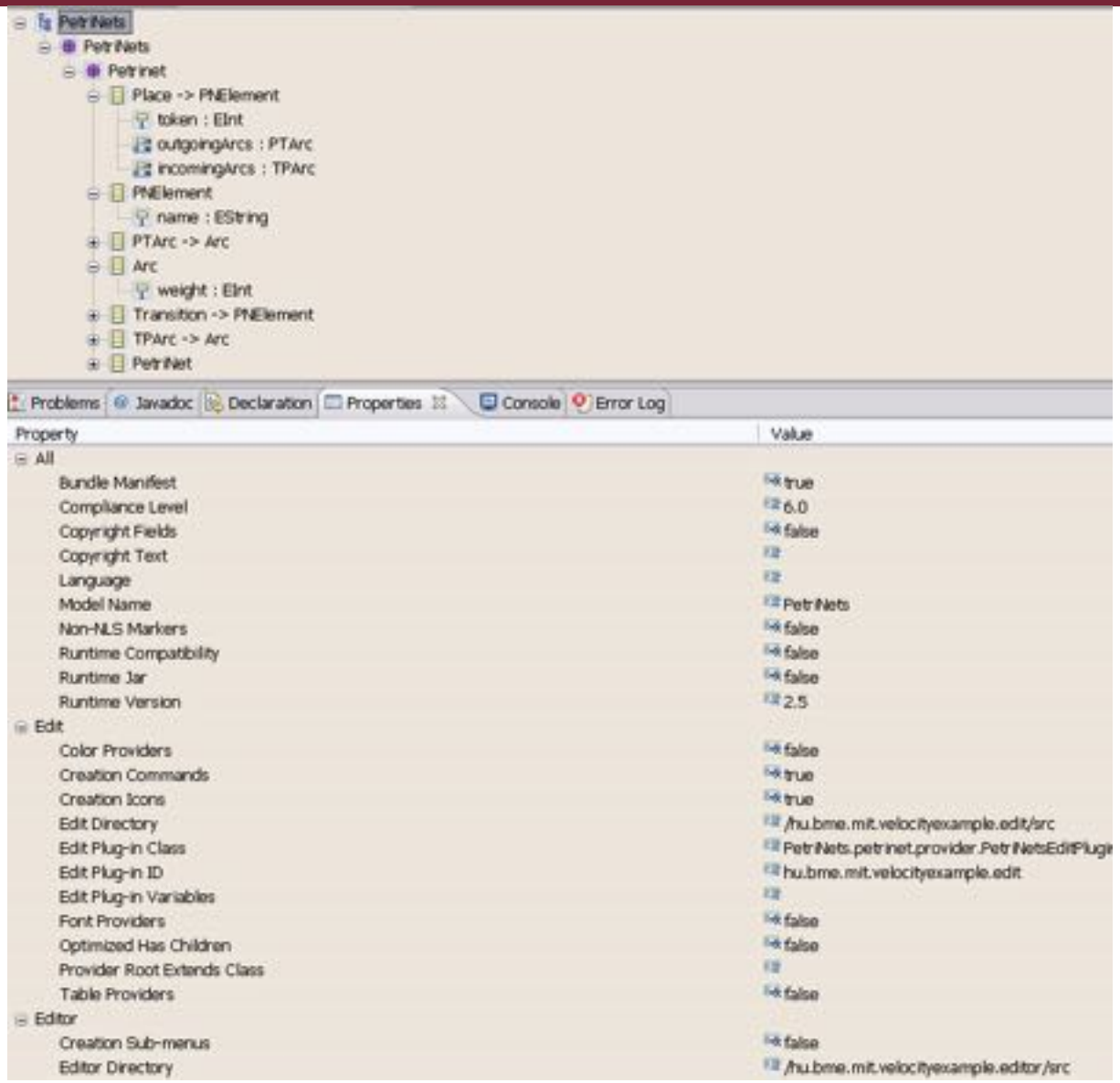
Generator model (.genmodel)

- Goal:
  - Specify the attributes of the code generation
- EMF model
  - Tree Editor
  - Refers to the Ecore model
- Code generation attributes
  - Java version (e.g., use Enums in case of Java 5 and higher)
  - Package/project names
  - …

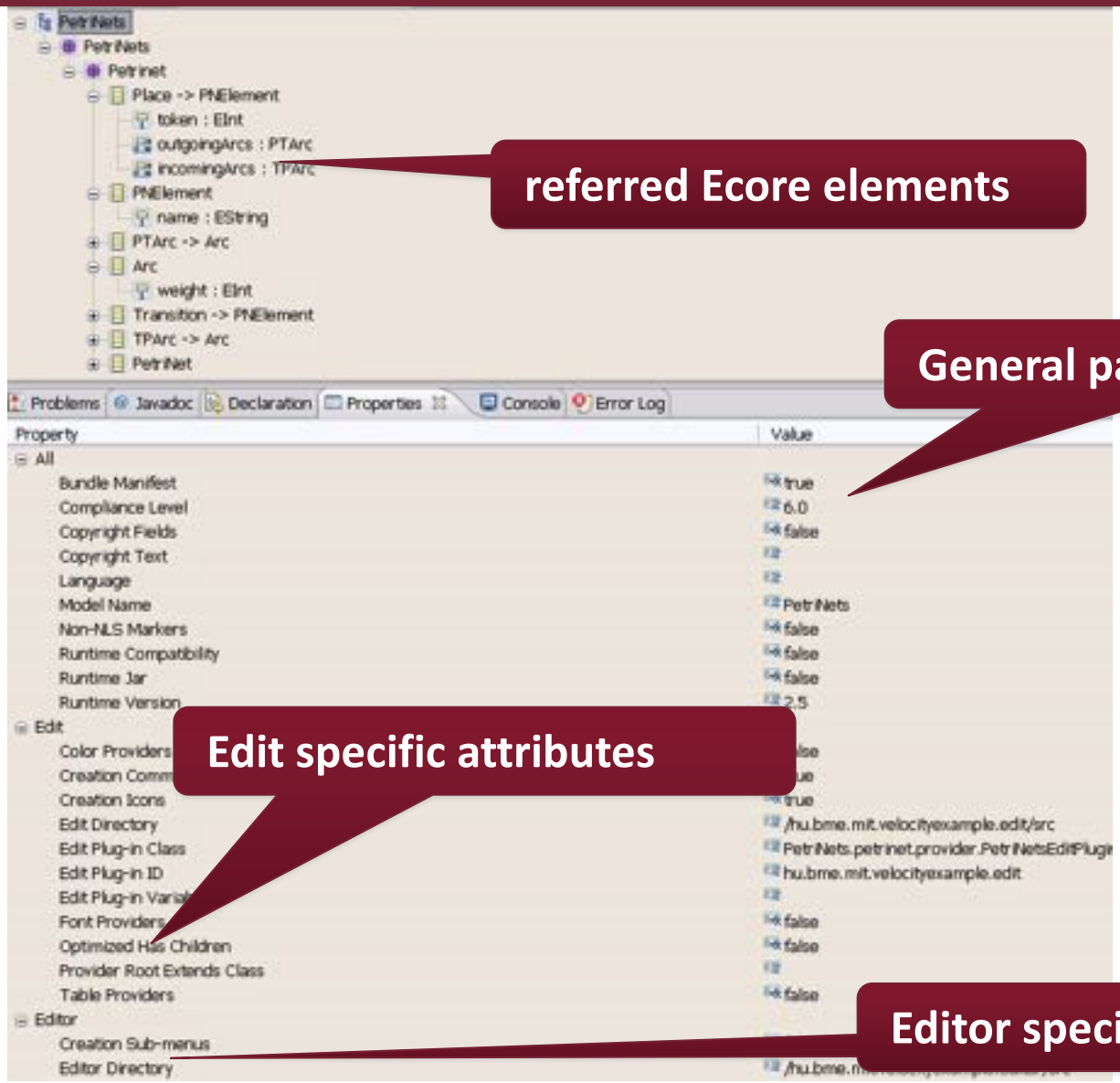
Code Generation from Ecore (.genmodel)

- Ecore model remains pure and independent
- Customizable (wrappers, code formatters, etc.)
- Generated plugins:
  - Model persistency (EMF.model)
  - Model management (EMF.edit)
  - Model editor (EMF.editor)
- Has some limitations
  - What happens when the underlying .ecore changes?

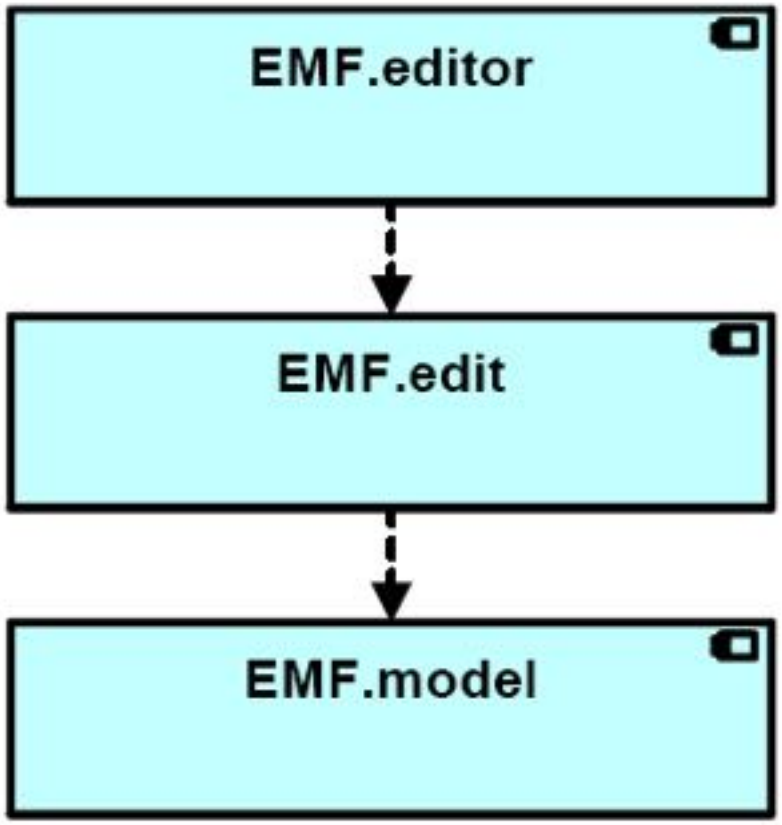
Generator model



Generator model

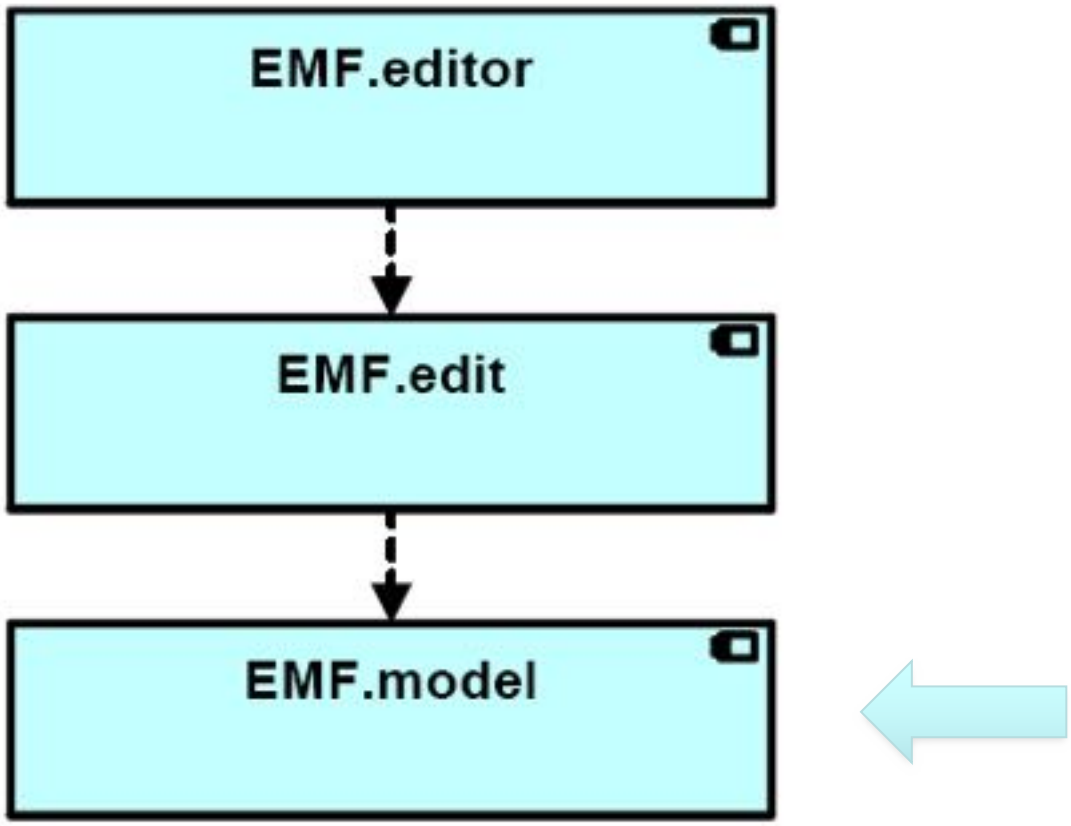


Generated EMF components



- 3. Tree Editor
- 2. Model Manipulation
- 1. Model Persistency

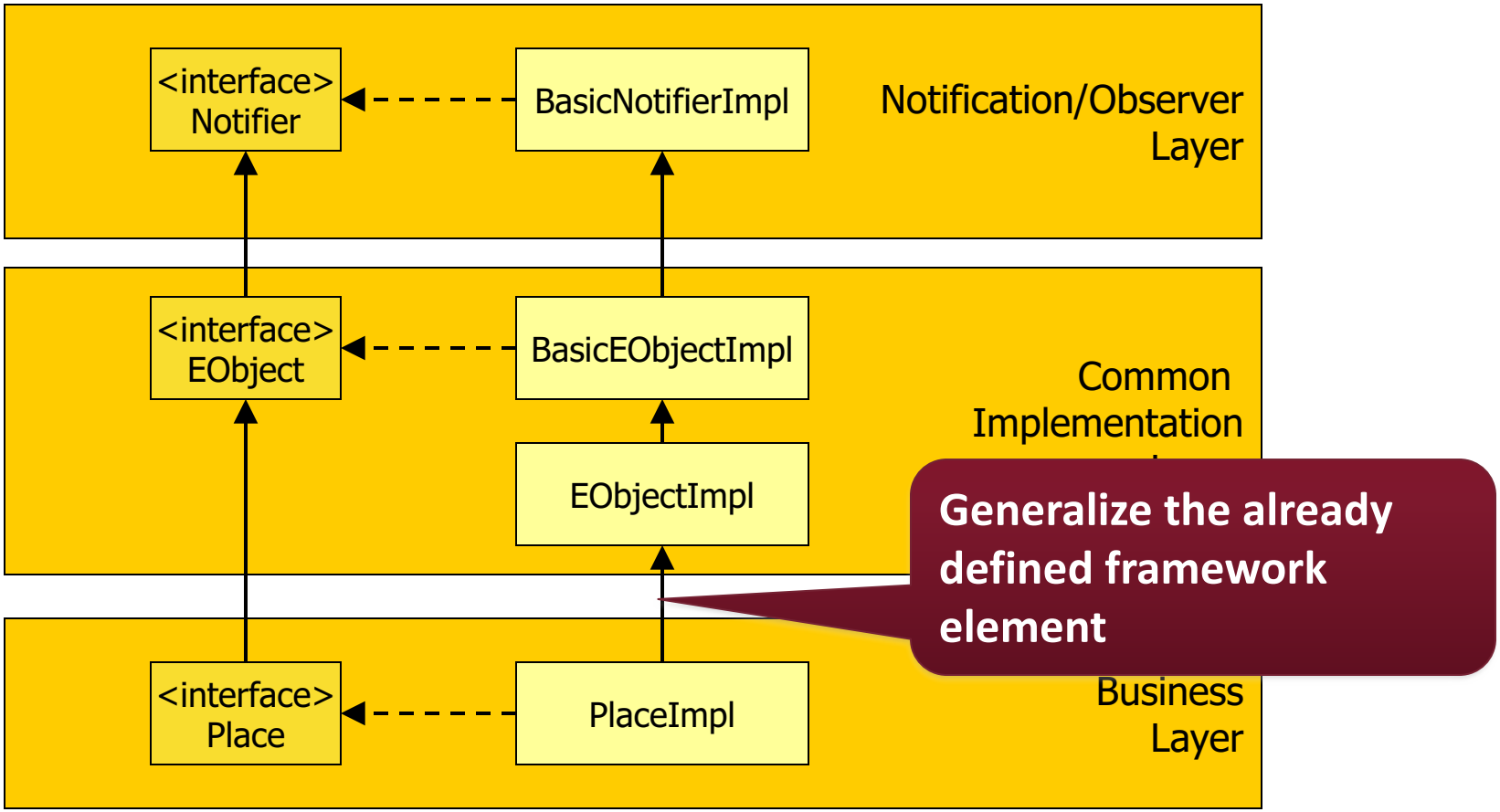
Generated EMF components



EMF.model

- Optimized persistency handling
- Fully featured Java code of the Ecore model
- Specific factories for all packages
- Notification mechanism (observer pattern)
- Possible extension points:
  - Advanced editor
  - Own file format with parser

EClass implementation



Auto-Generated Interface

```
 * @model
 * @generated
 */
public interface Place extends PNElement {
    /**
     * @model required="true"
     * @generated
     */
    int getToken();

    /**
     * @see #getToken()
     * @generated
     */
    void setToken(int value);

    /**
     * @model opposite="fromPlace" containment="true"
     * @generated
     */
    EList<PTArc> getOutgoingArcs();

    /**
     * @model opposite="toPlace"
     * @generated
     */
    EList<TPArc> getIncomingArcs();

} // Place
```

ESuperClass

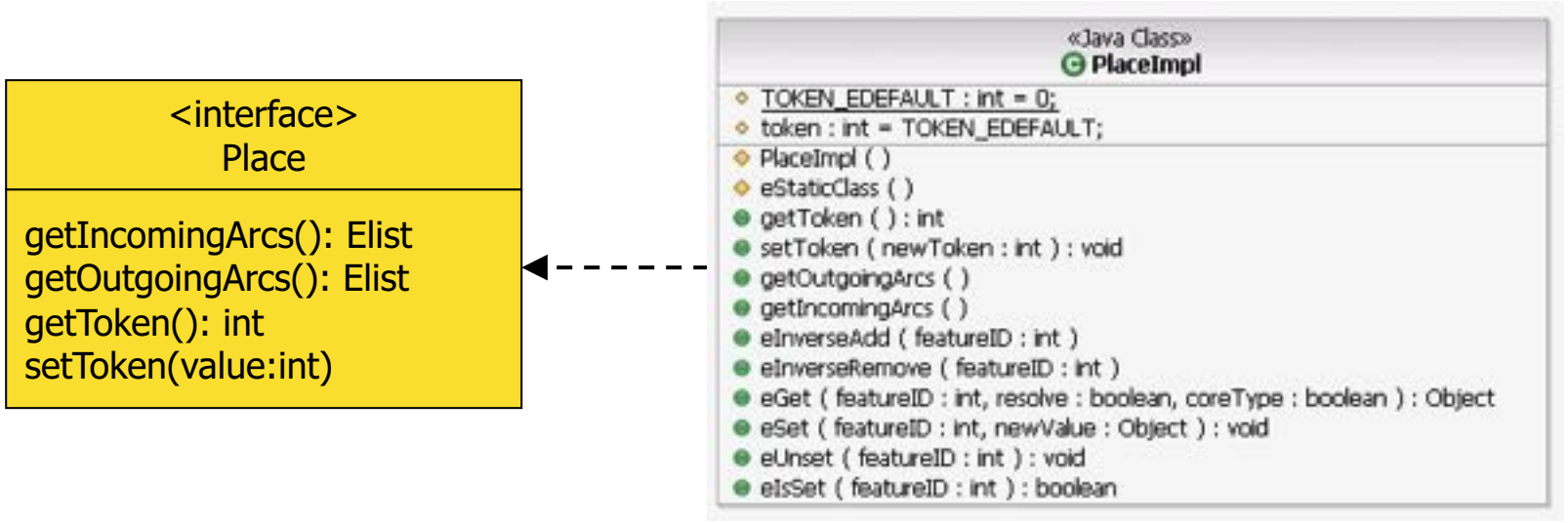
EMF specific „annotations”

Getters/Setters for attributes

No setter when multiplicity > 1 (use add/remove instead)

EList: EMF list interface (~10 implementations

EObject API



- Every class contains framework-specific methods:
  - Reflective get/set (eGet, eSet)
  - Consistent manipulation (eInverseRemove)
  - Notifications for feature changes (very useful e.g. in GUI!)
- Inherited from common supertype EObject
  - see deep instantiation earlier

EOperation Implementation

```
public class XImpl extends EObjectImpl implements X {

   /**
    * @generated NOT
    */
   void f() {
     // Provide the implementation
   }
}
```

- Represents the frame of a Java method
- Present in both the interface and implementing class
- Important:
  - Have to change the generated annotation to NOT
    - …so that next code generation phase does not overwrite it
  - Have to implement the method manually

Client Programming with EMF

```
Place p1 = PetrinetFactory.eINSTANCE.createPlace();
p1.setName("p1");
Place p2 = PetrinetFactory.eINSTANCE.createPlace();
p2.setName("p2");
Transition t1 = PetrinetFactory.eINSTANCE.createTransition();
t1.setName("t1");

// Inverse direction (p1.outgoingArcs) is set automatically
PTArc a0 = PetrinetFactory.eINSTANCE.createPTArc();
a0.setFromPlace(p1);
a0.setToTransition(t1);
TPArc a1 = PetrinetFactory.eINSTANCE.createTPArc();
a1.setToPlace(p2);
a1.setFromTransition(t1);
```

Create a place

Create a transition

Create a PT arc

Set source of PT arc

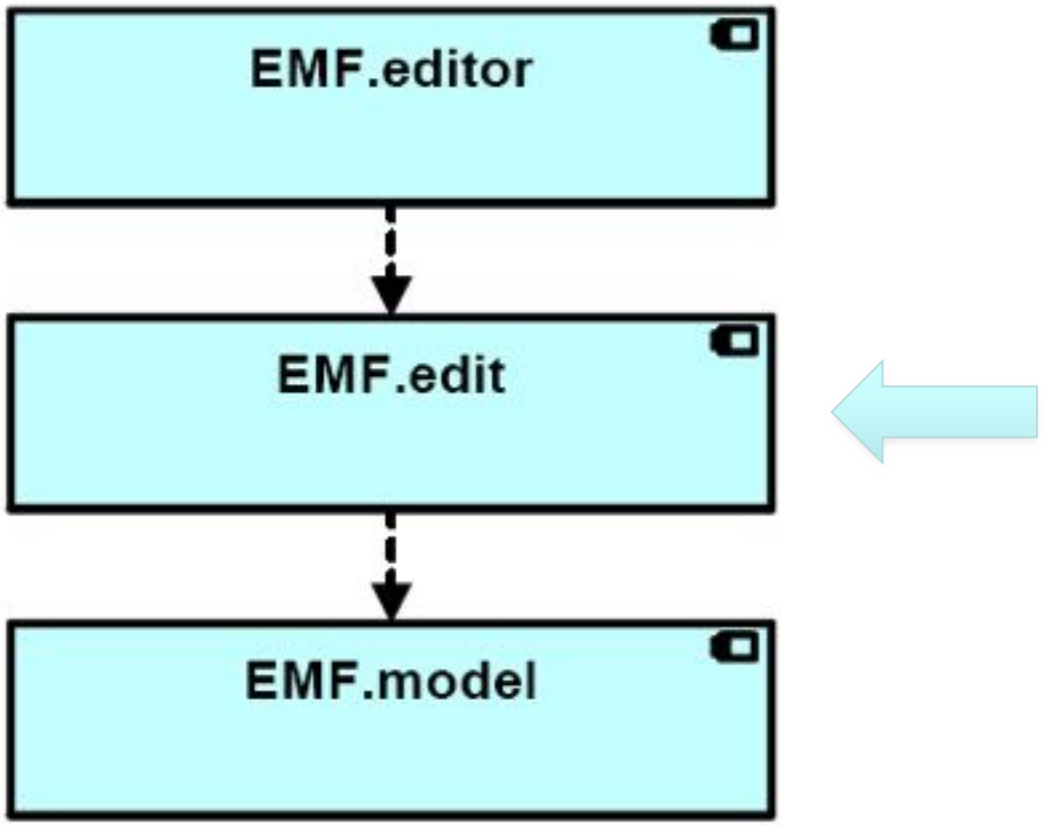
Set target of PT arc

Advanced client programming: Reflective Ecore API

The org.eclipse.emf.ecore.util Package

- Contains utility classes and interfaces:
  - *ECoreEContentAdapter*: maintains itself as a notification adapter for a whole containment (sub)tree
  - *UsageCrossReferencer*: finds each ModelElement pointing to the corresponding EObject
  - *ContentTreeIterator*: An iterator over the tree contents of a collection of EObjects
  - *Copier*: deep copy of EObject Elements and EReferences
  - Etc.
- (This is not generated but a generic component)

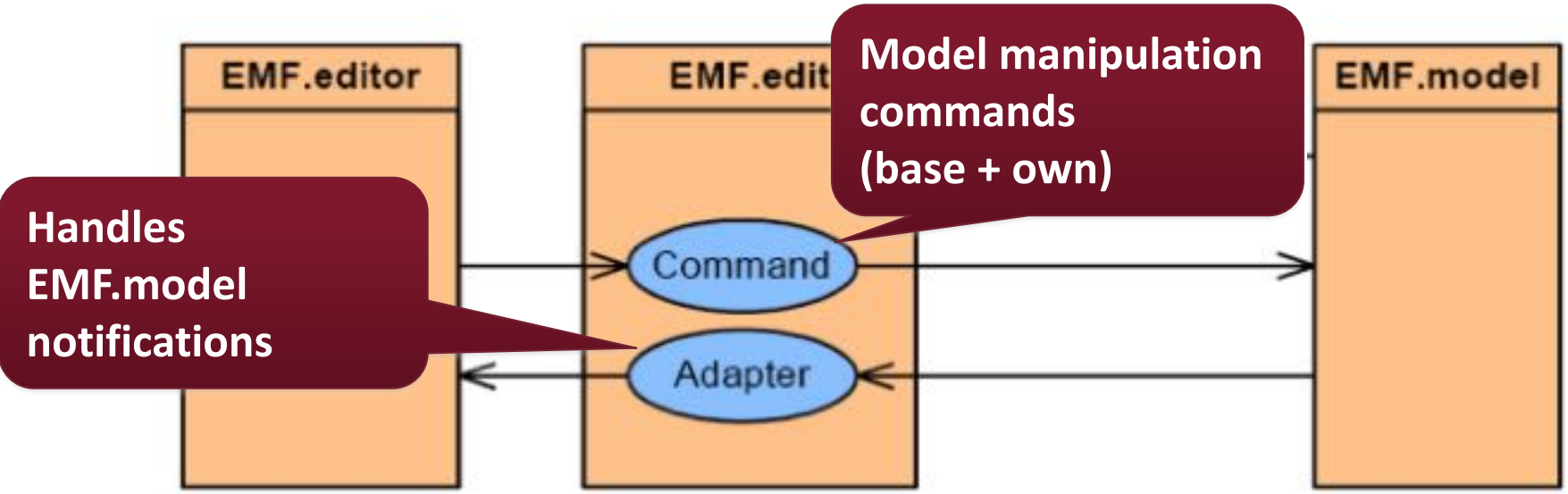
Generated EMF components



EMF.Edit

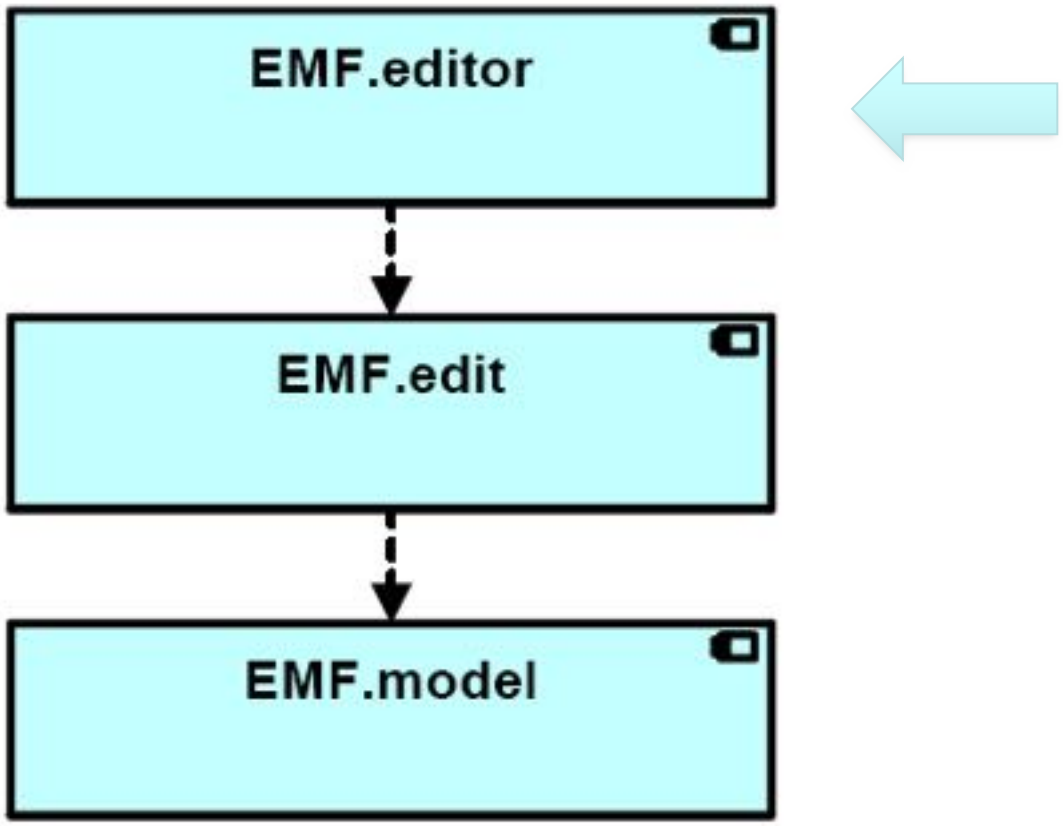
- Separates the GUI and the model
- Generator pattern:
  - Provider class for each model element
  - Base class: ItemProvider
  - Forward EMF model change notifications to the viewer
- Provides:
  - Element text
  - Icon
  - Description of features in EClass

EMF.Edit

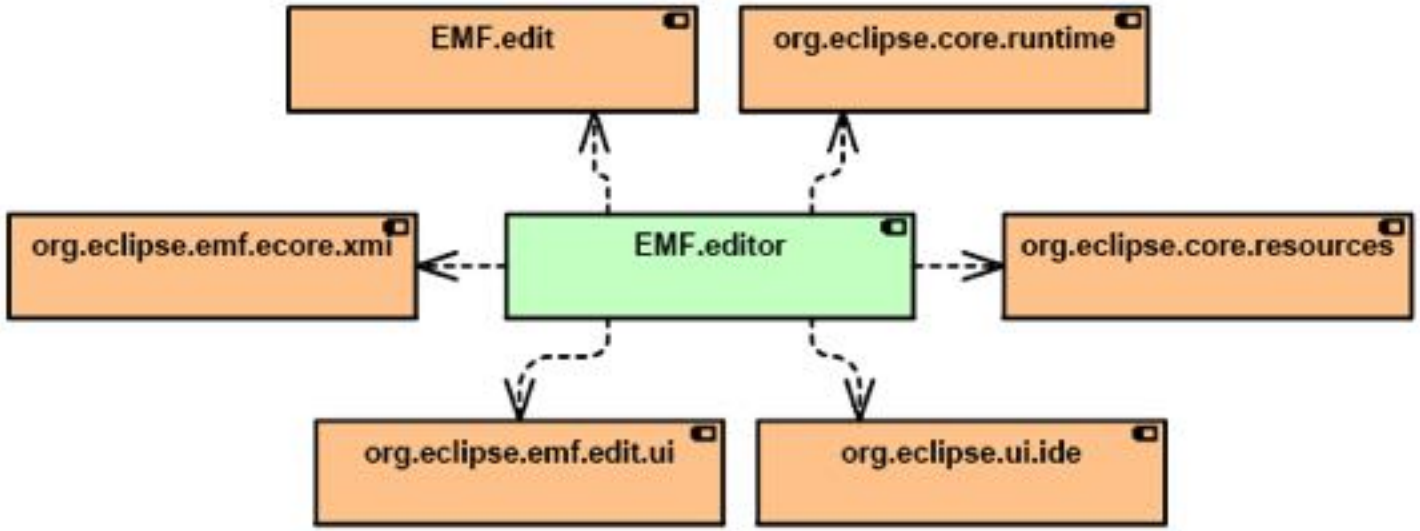


- Converts model notifications to GUI notifications
- Model manipulation through commands
  - Possible alternative to direct setters
  - Undoable, redoable
  - ItemProvider.create*Add*Command(…) etc.

Generated EMF components

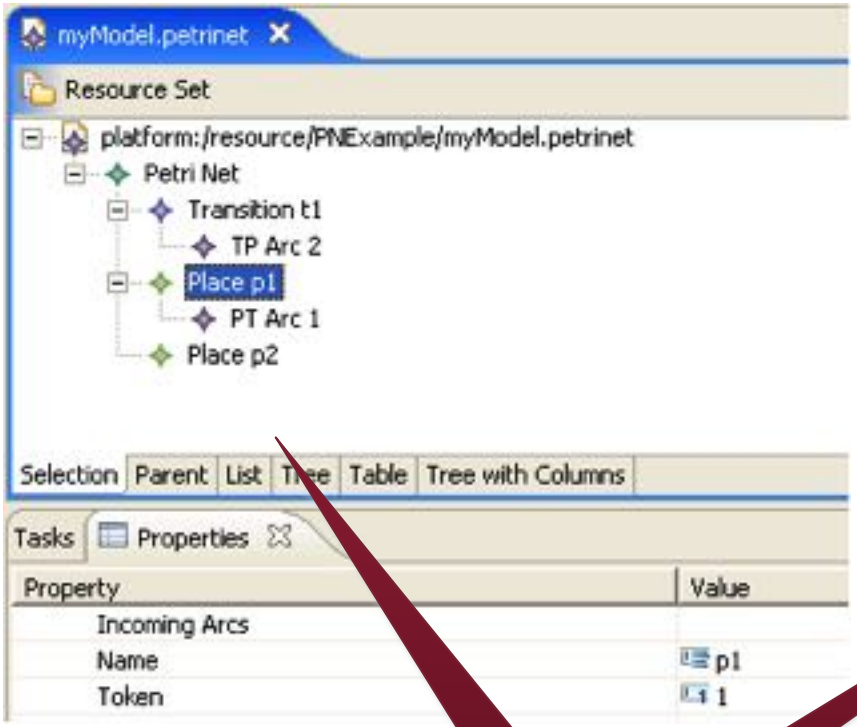


EMF.Editor



- EMF.Editor generates the SWT/JFace for the graphical editor
- Generates:
  - Tree editor
  - Wizards
  - Menus
  - plugins

The editor of Petri Net models




```
<?xml version="1.0" encoding="UTF-8"?>
<PetriNets.petrinet:PetriNet xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:PetriNets.petrinet=
    "http:///PetriNets/petrinet.ecore">
 <transitions name="t1" incomingArcs=
    "//@places.0/@outgoingArcs.0">
  <outgoingArcs weight="2"
    toPlace="//@places.1"/>
 </transitions>
 <places name="p1" token="1">
  <outgoingArcs weight="1"
    toTransition="//@transitions.0"/>
 </places>
 <places name="p2" incomingArcs=
    "//@transitions.0/@outgoingArcs.0"/>
</PetriNets.petrinet:PetriNet>
```

Place p1

Reference: URI (or XMI.id)

Tree View

XMI 2.0 View

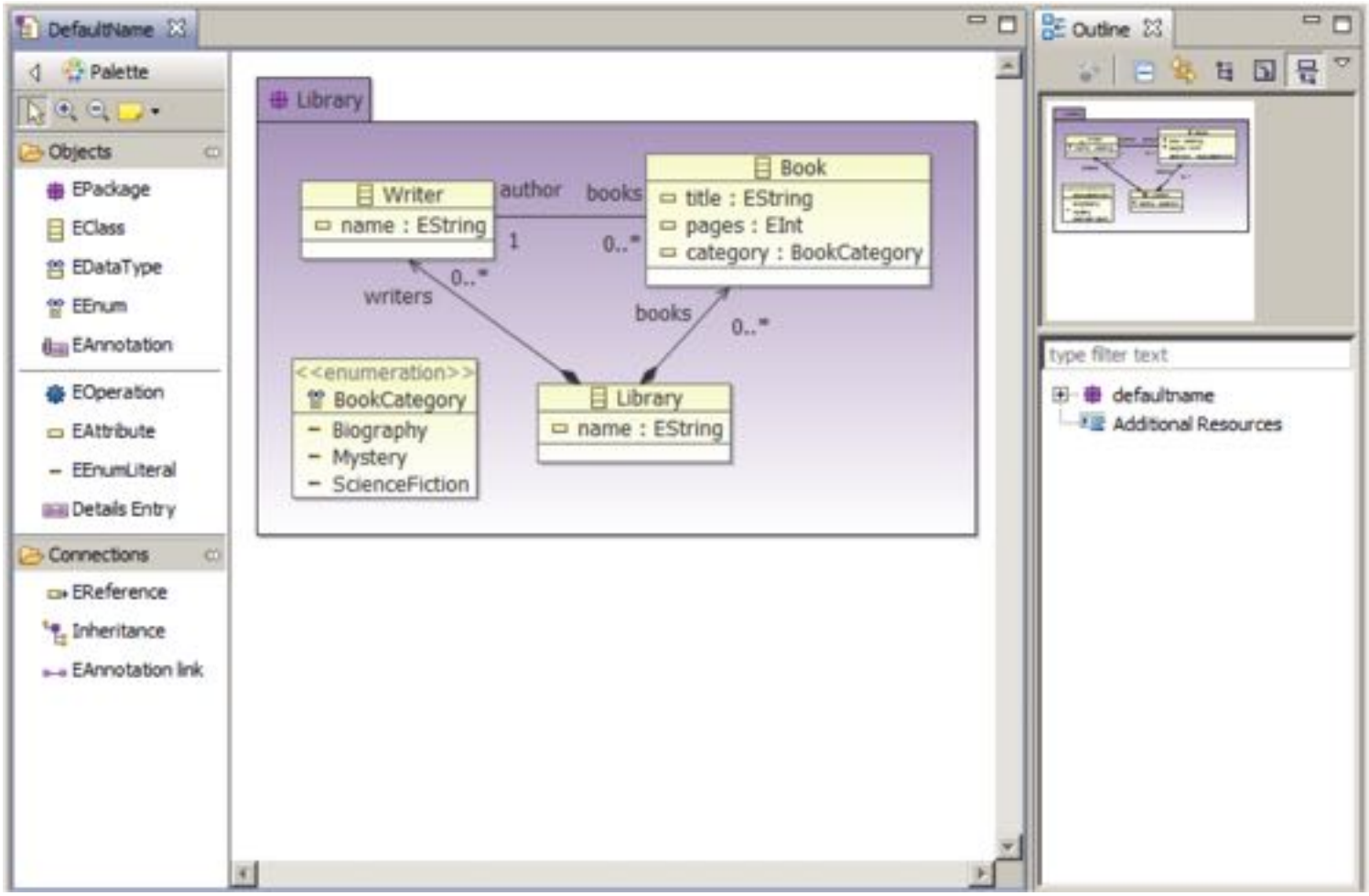
TOOLS, API AND UTILITIES

Basic EMF tools

- Validation
  - Validate constraints over EMF models
- Query
  - High-level query language for EMF
  - See also: Viatra Query ☺
- Compare
  - To structurally compare EMF models (e.g., versioning)
- Teneo
  - Persistency layer over relation databases
- SDO
  - Service Oriented Architecture based on EMF
- CDO
  - distributed, client-server EMF models

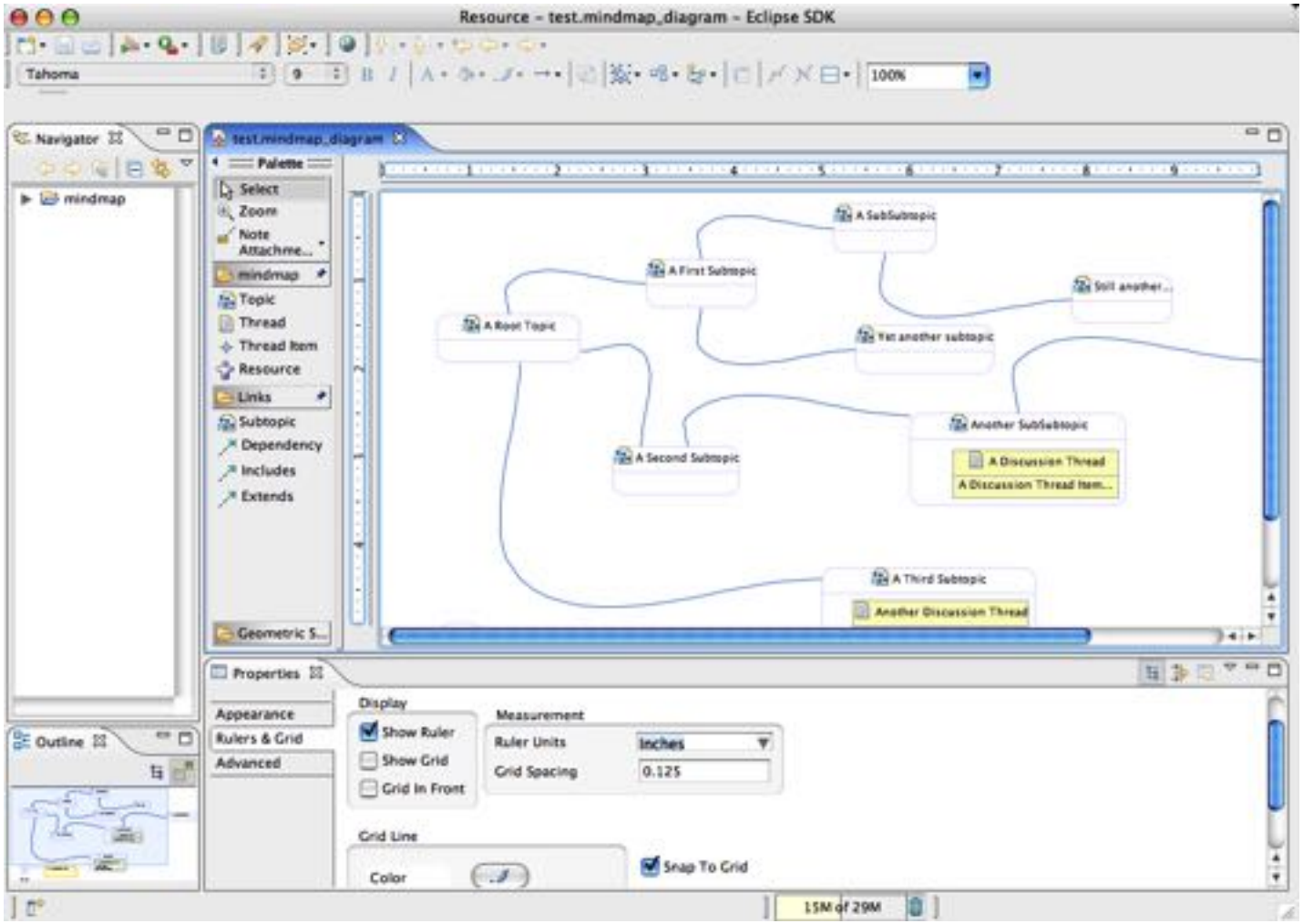
Ecore Tools: Ecore Diagram Editor

- Graphical DSL to define EMF metamodels
  - Based on GMF



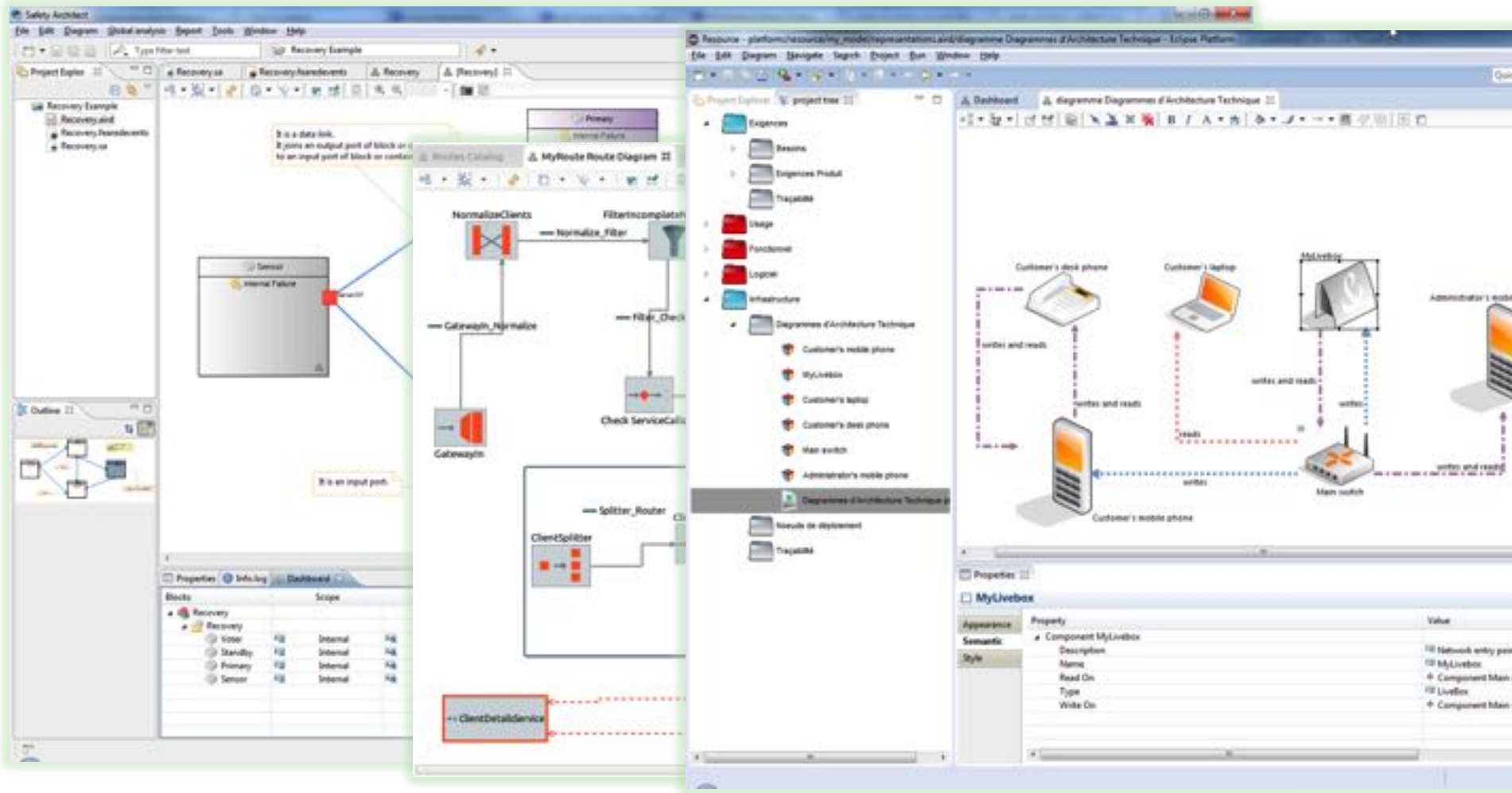
GMF

- DSL to define graphical concrete syntax



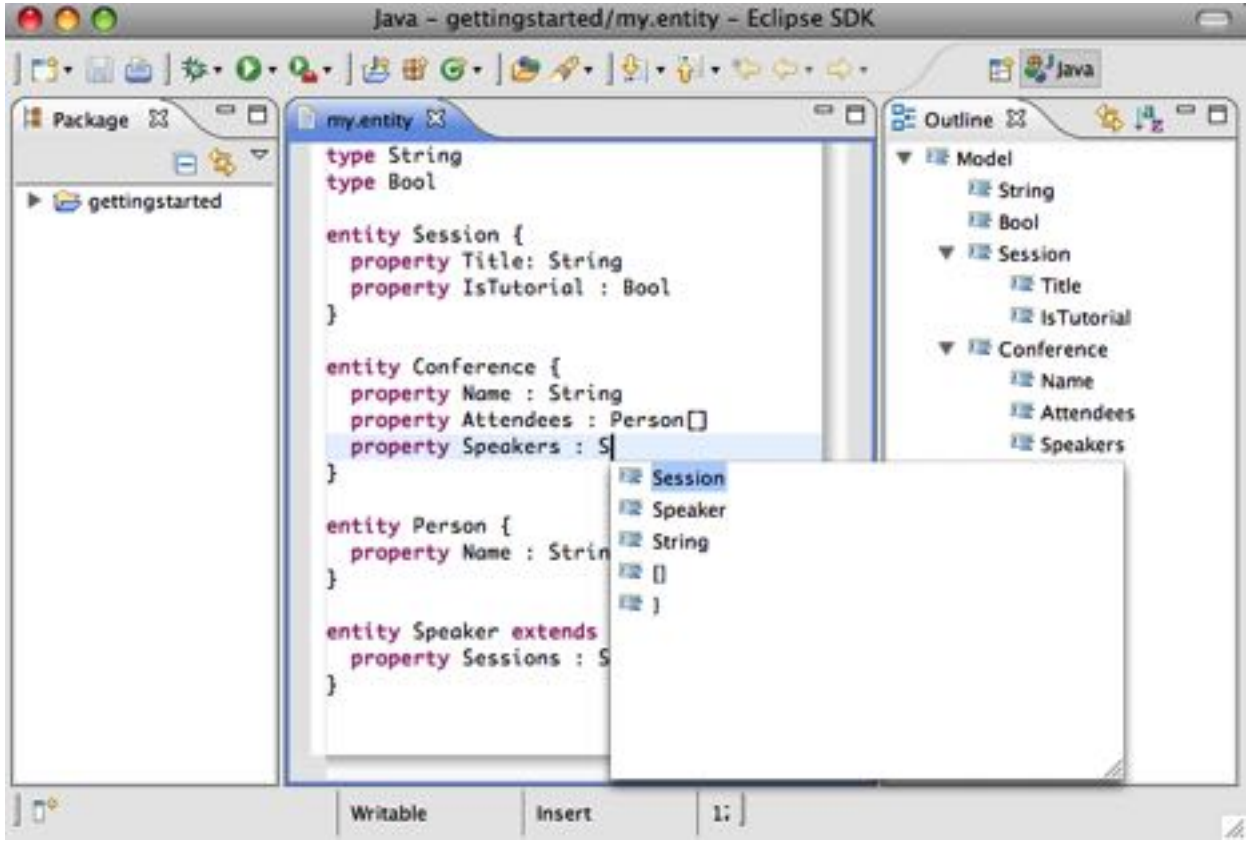
Sirius

- DSL to define workbench incl. graphical concrete syntax



Xtext

- Textual DSL for defining metamodel + textual syntax
- Context-free grammar!
- Generates:
  - Metamodel
  - Parser
  - Editor features

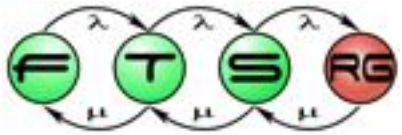


OCL – The Object Constraint Language

Gábor Bergmann, Ákos Horváth, Dániel Varró, **István Ráth**, István Majzik and Gergely Pintér

Model Driven Software Development

Lecture 3



OCL Motivation

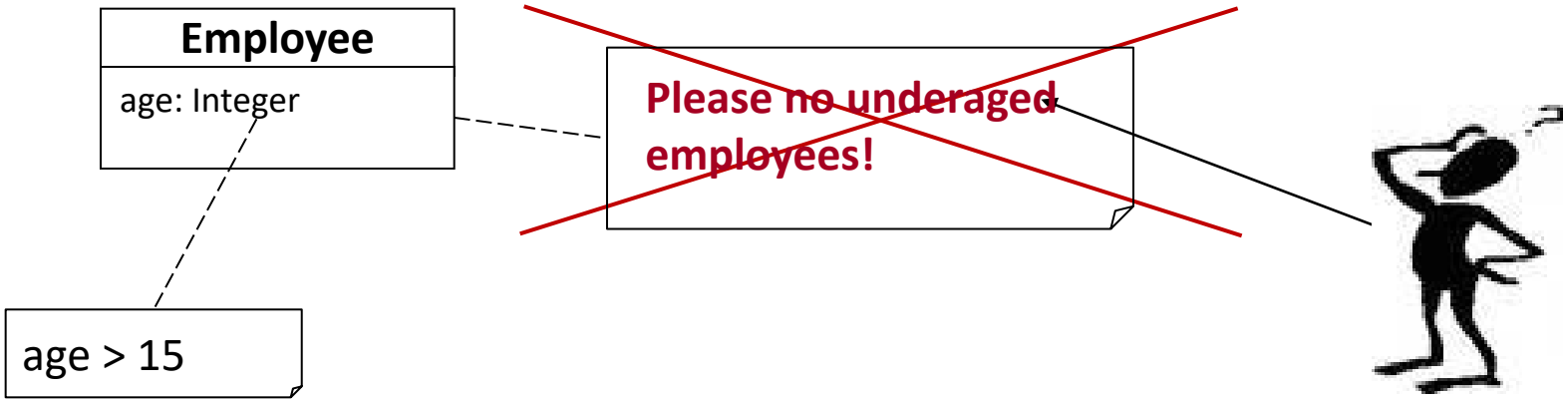
How to capture restrictions / constraints of domain classes?

Motivation

- Graphical modeling languages are generally not able to describe all facets of a problem description
  - *MOF, UML, ER, …*
- Special **constraints** are often (if at all) added to the diagrams in **natural language**
  - Often **ambiguous**
  - Cannot be validated **automatically**
  - No **automatic** code generation
- Constraint definition also crucial in the definition of new modeling languages (DSLs).

Motivation

- Example 1



Additional question: How do I get all Employees younger than 30 years old?

Motivation

- **Formal specification languages** are the solution
  - Mostly based on **set theory** or **predicate logic**
  - Requires good mathematical understanding
  - Mostly used in the academic area, but hardly used in the industry
  - Hard to learn and hard to apply
  - Problems when to be used in big systems

- ***Object Constraint Language (OCL)*: Combination of modeling language and formal specification language
  - Formal, precise, unique
  - Intuitive syntax is key to **large group of users**
  - No programming language (no algorithms, no technological APIs, …)
  - Tool support: *parser, constraint checker, codegeneration,…*

OCL usage

- Constraints in UML-models
  - Invariants for classes, interfaces, stereotypes, …
  - Pre- and postconditions for operations
  - Guards for messages and state transition
  - Specification of messages and signals
  - Calculation of derived attributes and association ends
- Constraints in meta models
  - Invariants for Meta model classes
  - Rules for the definition of well-formedness of meta model
- Query language for models
  - In analogy to SQL for DBMS, XPath and XQuery for XML
  - Used in transformation languages

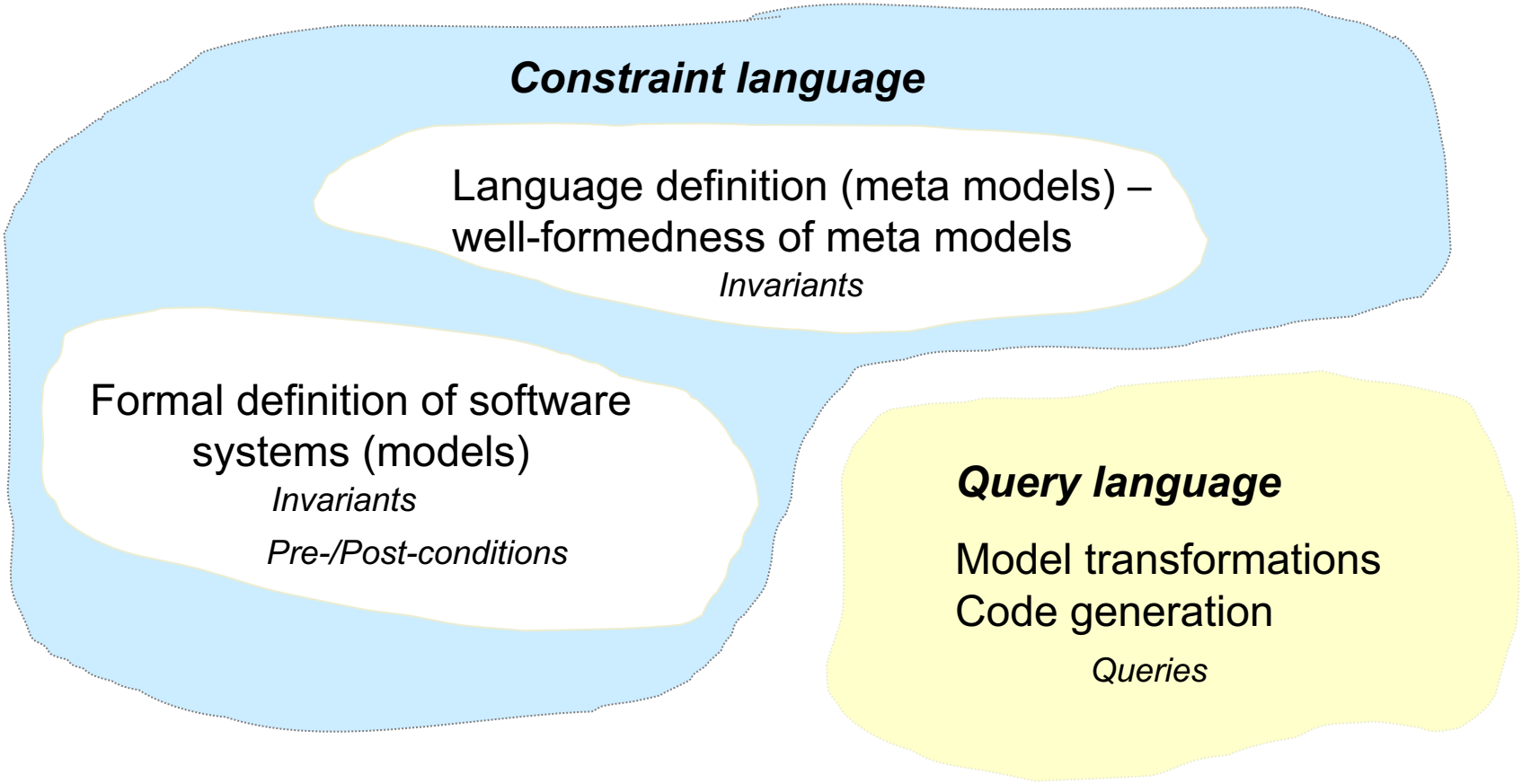
OCL usage

- OCL field of application
  - Invariants — **context** *C* **inv:** *I*
  - Pre-/Postconditions — **context** *C::op() : T* **pre:** *P* **post:** *Q*
  - Query operations — **context** *C::op() : T* **body:** *e*
  - Initial values — **context** *C::p : T* **init:** *e*
  - Derived attributes — **context** *C::p : T* **derive:** *e*
  - Attribute/operation definition — **context** *C* **def:** *p : T = e*

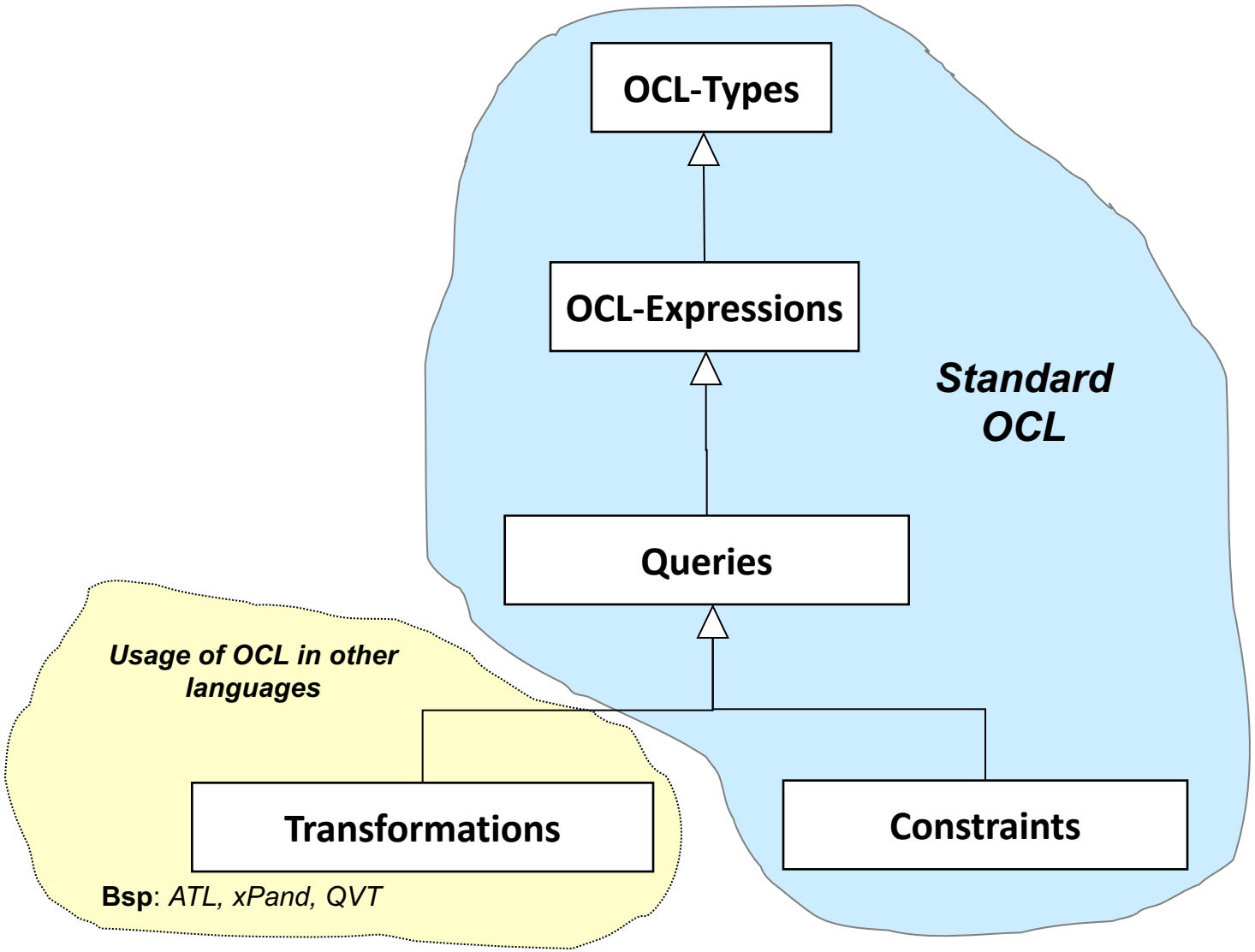
- Caution: Side effects are not allowed!
  - Operation `C::getAtt : String body: att` allowed in OCL
  - Operation `C::setAtt(arg) : T body: att = arg` **not** allowed in OCL

OCL usage

- **Field of application** of OCL in model driven engineering



OCL usage

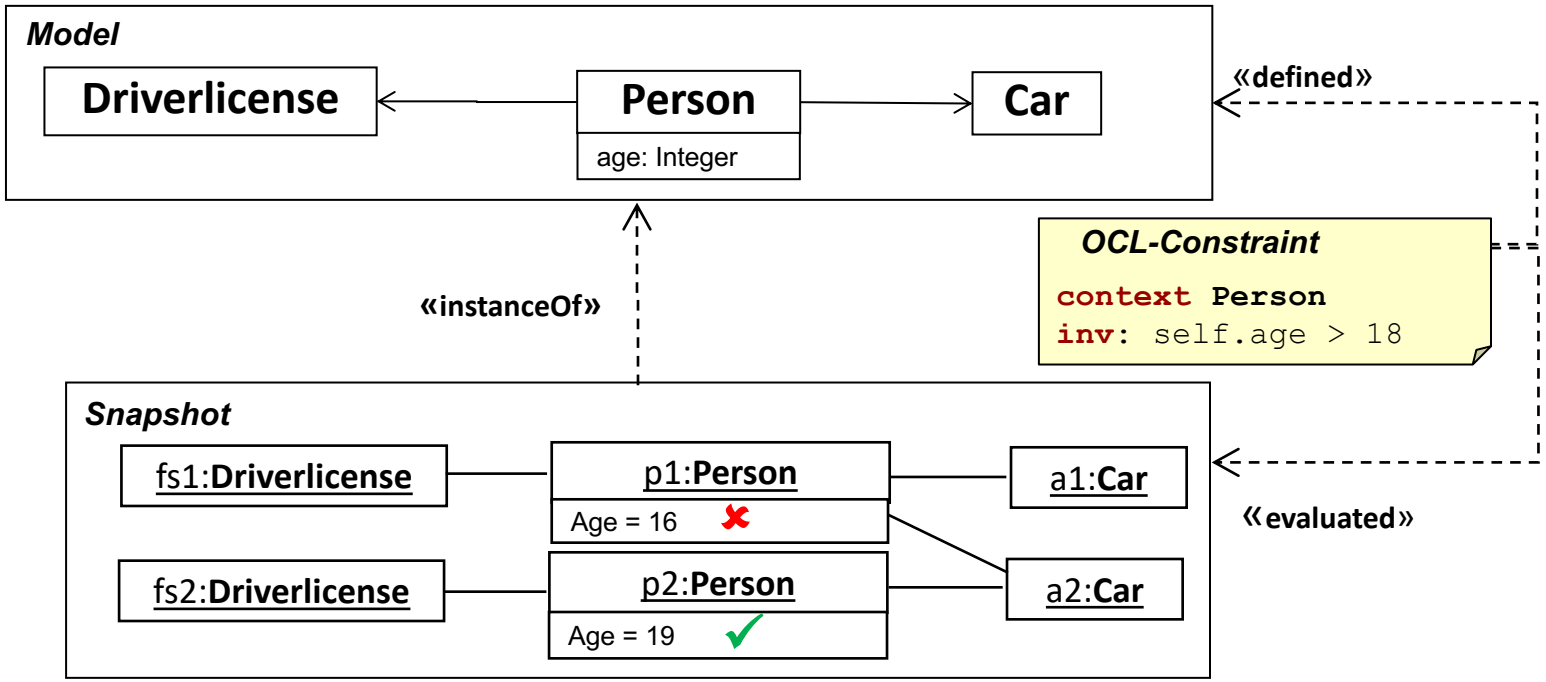

OCL-Types
OCL-Expressions
Standard OCL
Queries
Usage of OCL in other languages
Transformations
Constraints
Bsp: ATL, xPand, QVT

OCL usage

How does OCL work?

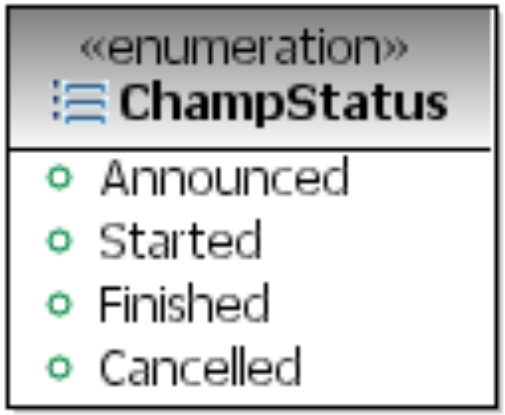
- **Constraints** are defined on the modeling level
  - Basis: Classes and their properties
- Information of the **object graph** are **queried**
  - Represents system status, also called ***snapshot***
- **Anaology** to XML query languages
  - XPath/XQuery query XML-documents
  - Scripts are based on XML-schema information

- Examples



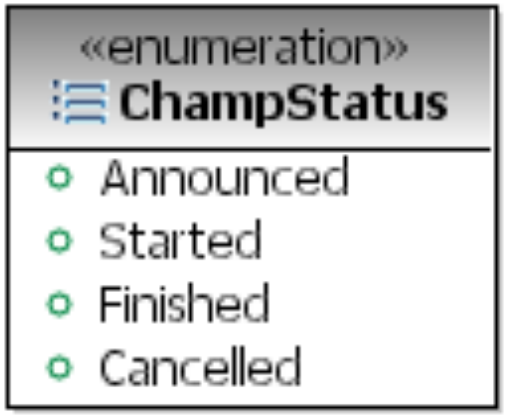
First OCL Examples

Informal Constraints on Championship



- What are the restrictions?
  - `name` is not empty
  - `minParticipants ≤ maxParticipants`
  - `minParticipants ≥ 0`
  - `maxParticipants > 0`

First OCL constraints



- Name is not empty

Context

Invariant

```
context Championship inv:
  self.name <> ''
```

- Constraints on participants

```
context Championship inv:
  self.minParticipants >=
  0
context Championship inv:
  self.maxParticipants >=
  1
context Championship inv:
  self.maxParticipants >=
  self.minParticipants
```

Instance of the class

Navigation along attributes

Informal Constraints on Player

- What are the restrictions?
  - `userName` is not empty
  - `userName` is unique
  - `1800 ≤ birth ≤ 3000`
  - `password` is not empty
  - `age = current_year - birth`

Informal Constraints on Player

- 1800 ≤ birth ≤ 3000

```
context Player inv:
  self.birth >= 1800 and
  self.birth <= 3000
```

Get all instances into a collection

Logical AND

- Name is unique

```
context Player inv:
  Player.allInstances()->
  forAll(p1, p2 | p1<>p2 implies
  p1.userName <> p2.userName)
```

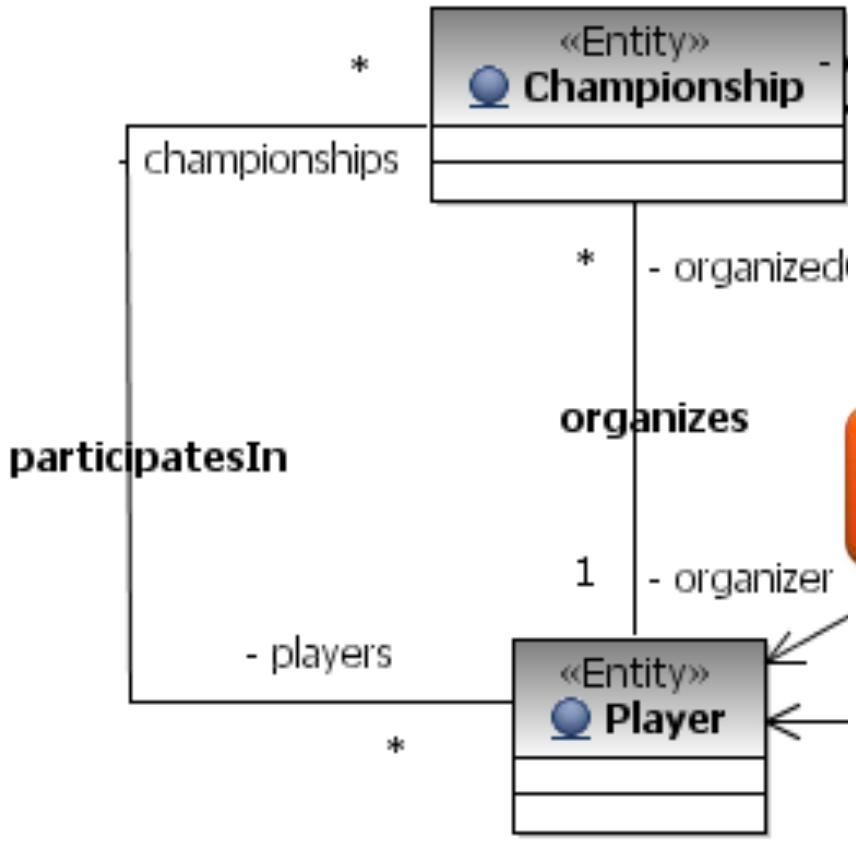
Logical implication

If p1 ≠ p2

Then p1.userName ≠ p2.userName

Universal quantification: For all objects in the collection

Navigation along roles



Only attributes of an **object** can be compared with a value

- Multiplicity 0..1

```
context Championship inv:
  self.organizer.birth > 1976
```

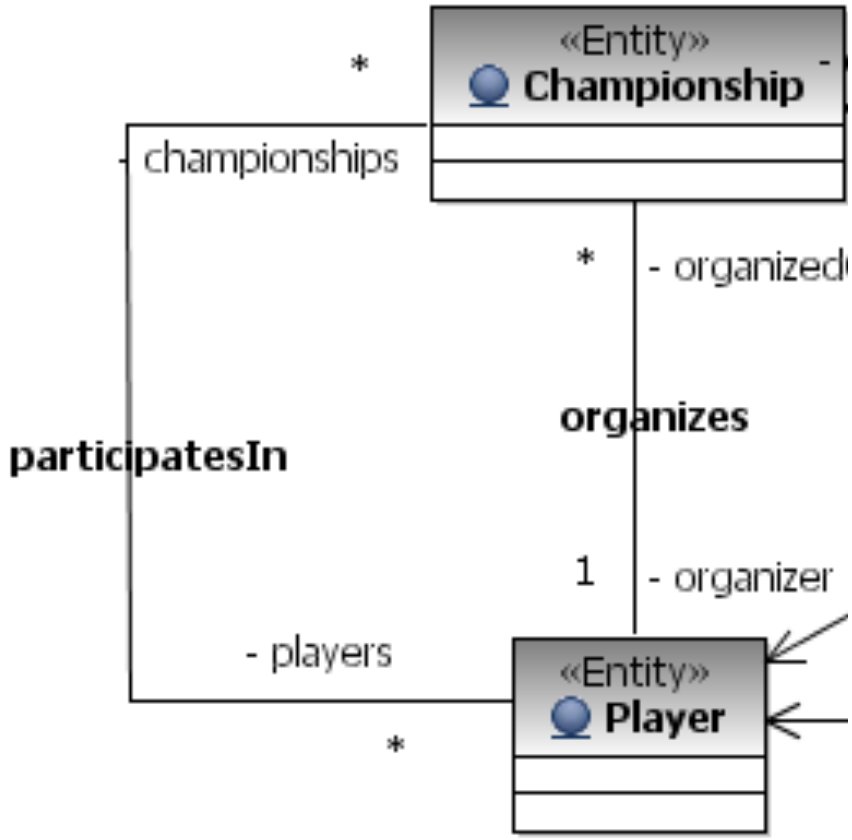
- Multiplicity * (many)

```
context Championship inv:
  self.players.birth > 1976
```

`self.players` results in a **collection**
`self.players.birth`: the coll. of birth years

```
context Championship inv:
  self.players-> …
  (operations on collections)
```

Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

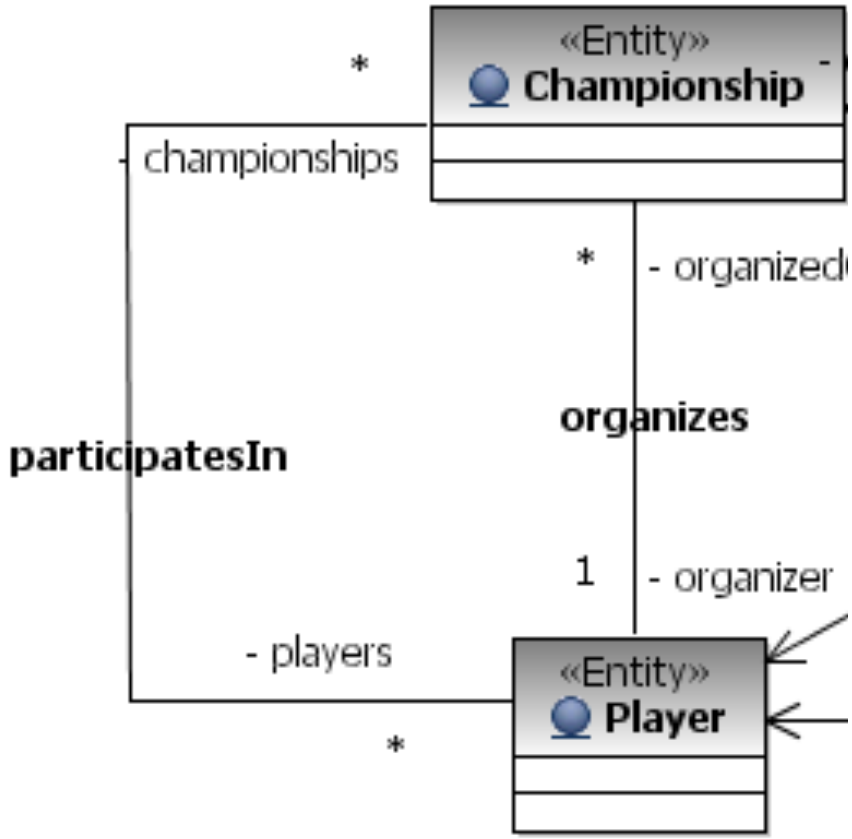
~~context Championship inv:~~
~~self.organizer.organized=self~~

Collection = Single object
Such an equality is invalid

```
context Championship inv:
  self.organizer.organized
  -> includes(self)
```

Coll->includes(e):
Tests collection membership: **e ∈Coll**

Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

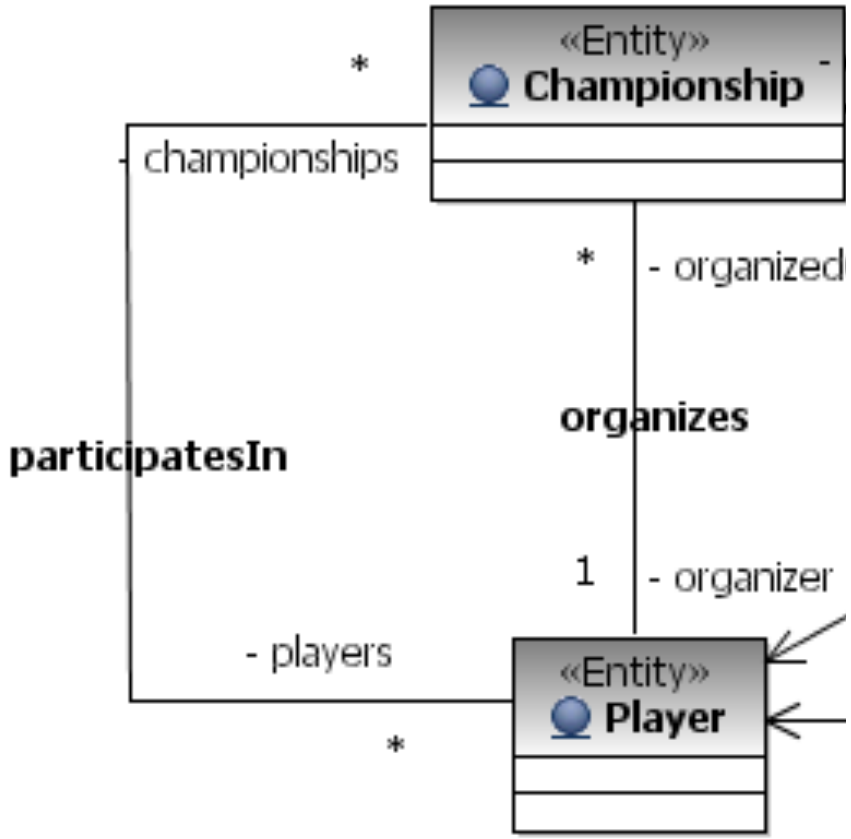
```
context Player inv:
  self.organized->exists(
  c | c.organizer = se
```

Incorrect: constraint is prescribed **for all** champs

```
context Player inv:
  self.organized->forAll(
  c | c.organizer = self)
```

`Coll->forAll(e|cond(e))`
Quantifiers can only be applied to collections

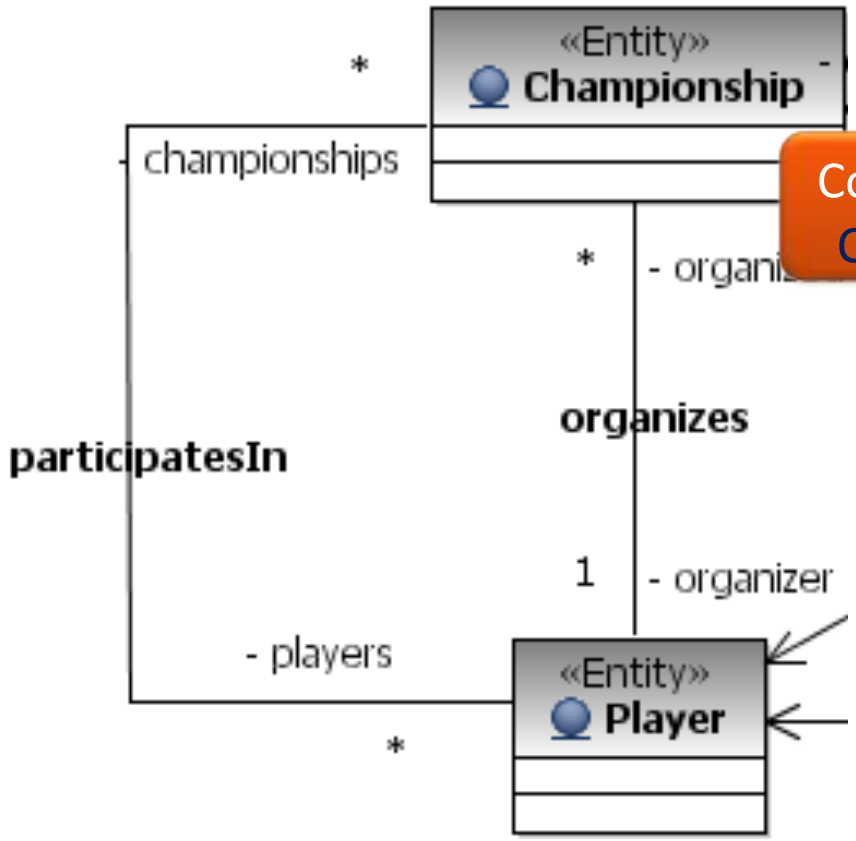
Consistency of bidirectional associations



- If a bidirectional association exists between two objects then it is navigable from both directions

```
context Championship inv:
  self.players->forall(
  p | p.championships->
  includes(self))
context Player inv:
  self.championships->forall(
  c | c.players ->
  includes(self))
```

Consistency of bidirectional associations



- The organizer of the championship organizes at least one championship

```
context Player inv:
  self.organized->size() > 0
```

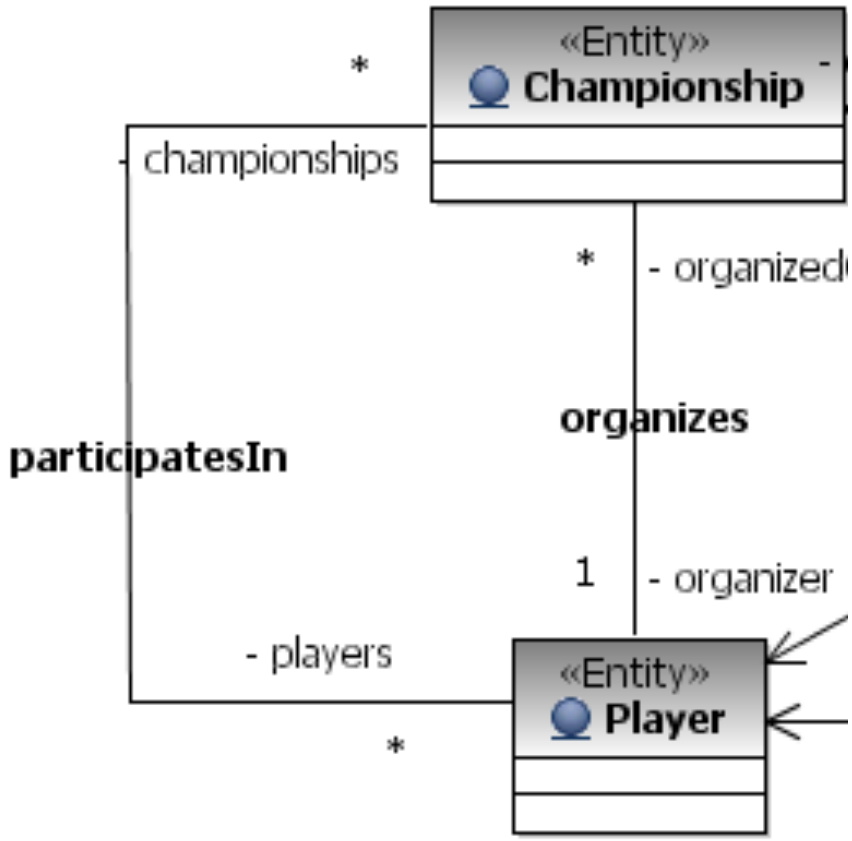
Context should be Championship

No player is forced to organize a champs

```
context Championship inv:
  self.organizer.organized->
  size() > 0
```

```
context Championship inv:
  self.organizer.organized->
  notEmpty()
```

Application specific constraints

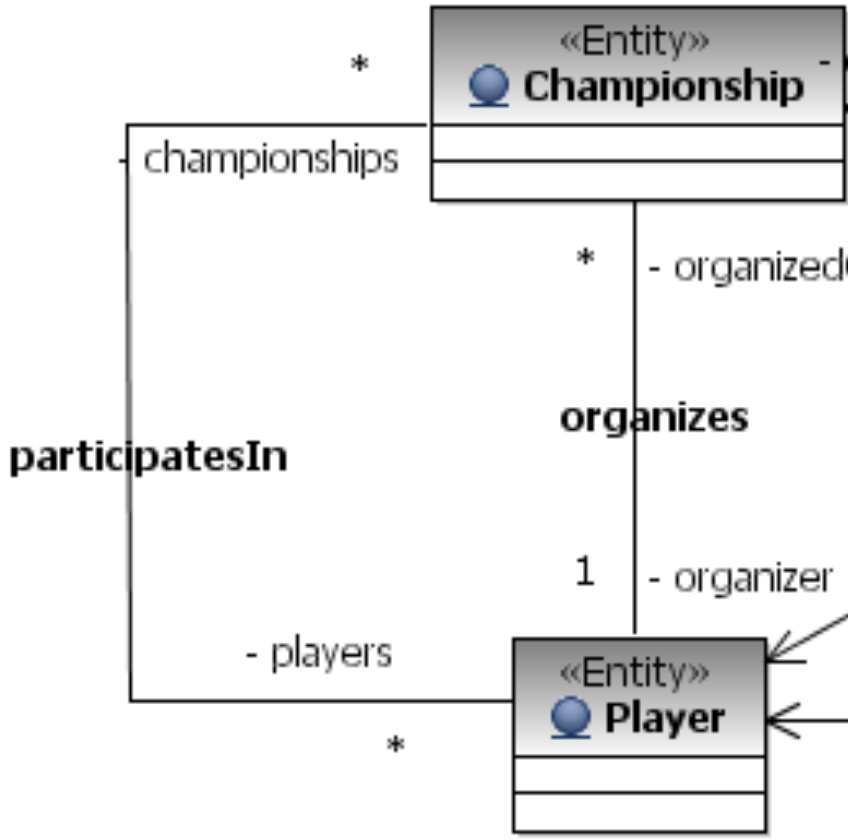


- A player is allowed to organize a single active championship at a time

```
context Player inv:
  self.organized->
  forall(c1, c2 | c1<>c2 implies
  (c1.status = ChS::closed or
   c1.status = ChS::cancelled)
  or
  (c2.status = ChS::closed or
   c2.status = ChS::cancelled))
context Player inv:
  self.organized->select(c |
  c.status = ChS::announced or
  c.status = ChS::started)->
  size() <=1
```

Values of an enumeration

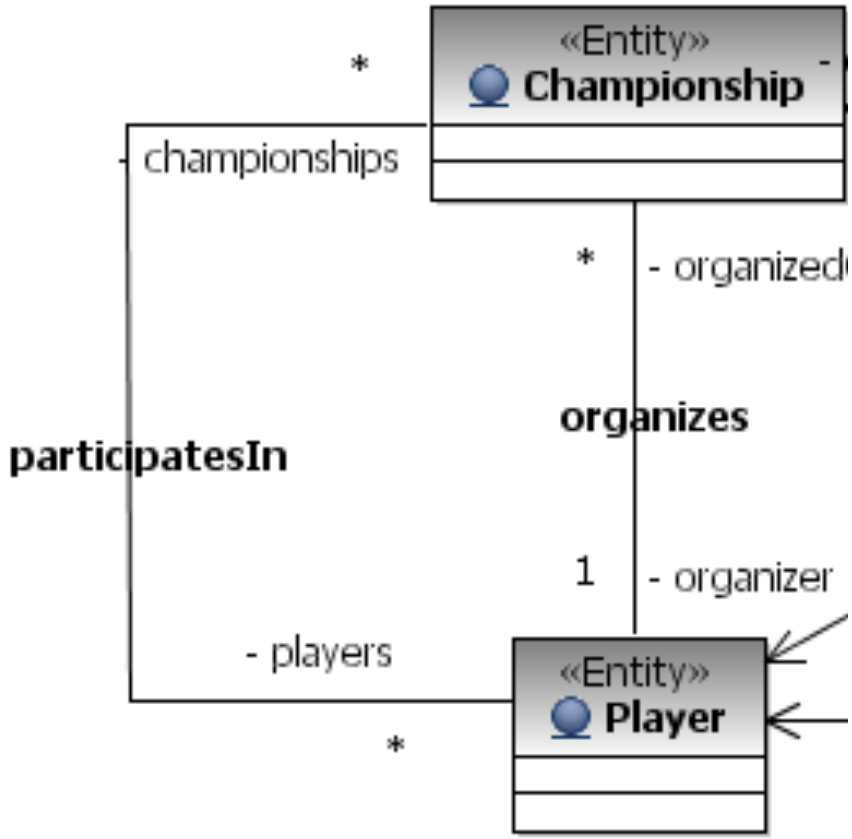
Application specific constraints



- A championship can only be started when the sufficient number of participants are present.

```
context Championship inv:
  (self.status =
  ChampStatus::started or
  self.status =
  ChampStatus::finished)
  implies
  (self.players->size() >=
   self.minParticipants and
   self.players->size() <=
   self.maxParticipants)
```

Application specific constraints



- Youth championship: the average age of participants is below 21.

`players.age` is the collection of the age attributes of players

```
context Championship inv:
  self.players.age->sum() /
  self.players->size() < 21
```

`sum()` can only be applied to a collection that contains numbers

An Overview of OCL Constructs

Types and Boole algebra in OCL

- All OCL expressions are typed
  - `OclAny`:
    The type that includes all others. E.g. `x, y : OclAny`
  - `x = y`
    x and y are the same object.
  - `x <> y`
    `not (x = y)`.
  - `x.oclType()`
    The type of x.
  - `x.isKindOf ( T )`
    True if T is a supertype (transitive) of the type of x.
  - `T.allInstances()` : Collection
    All the instances of type T.
- Boolean operators:
  - `b and b2, b or b2, b xor b2, not b`
    If any part of a Boolean expression fully determines the result, then it does not matter if some other parts of that expression have unknown or undefined results.
  - `b implies b2`
    True if b is false or if b is true and b2 is true.
  - `if b then e1 else e2 endif`
    If b is true the result is the value of e1; otherwise, the result is the value of e2.

Overview of Collection Valued Terms

- Size / aggregation:
  - `c->size()`: Integer
    Number of elements in the collection; for a bag or sequence, duplicates are counted as separate items.
  - `c->sum()`: Integer
    Sum of elements in the collection. Elements must be numbers
  - `c->count(e)`: Integer
    The number of times that e is in c.
  - `c->isEmpty()`: Boolean
    Same as `c->size() = 0`.
  - `c->notEmpty()`: Boolean
    Same as `not c->isEmpty()`.
- Equality
  - `c = c2` : Boolean
- Collection membership
  - `c->includes(e)`: Boolean;
    `c->exists ( x | x = e )`.
  - `c->excludes(e)`: Boolean;
    `not c->includes( e )`.
  - `c->includesAll(c2)`:
    Boolean;
    c includes all the elements in c2.
  - `c->including(e)`: Collection
    The collection that includes all of c as well as e.
  - `c->excluding(e)`: Collection
    The collection that includes all of c except e.

Overview of Collection Valued Terms

- Existential quantifier:
  - `c->exists(x | P )`: Boolean;
    there is at least one element in c, named x, for which predicate P is true.
  - Equivalent notation is:
    `c->exists(P)`,
    `c->exists(x:Type | P(x))`
- Universal quantifier:
  - `c->forAll(x | P)`: Boolean; for every element in c, named x, predicate P is true.
  - Equivalent notation is:
    `c->forAll(P)`
    `c->forAll(x:Type | P)`
- Selection:
  - `c->select(x | P)`: Collection
    The collection of elements in c for which P is true.
  - Equivalent is: `c->select(P)`
- Filtering:
  - `c->reject(x | P)`: Collection
    `c->select(x | not P)`.
  - Equivalent is: `c->reject(P)`
- Collection:
  - `c->collect(x | E)` : Bag
    The bag obtained by applying E to each element of c, named x.
  - `c.attribute` : Collection
    The collection(of type of c) consisting of the attribute of each element of c.

Sets, Bags, Sequences

Literals:

```
Set{ 1, 2, 5, 88 }
Set{ 'apple', 'orange',
   'strawberry'}
Sequence{ 1, 3, 45, 2, 3 }
Sequence{ 'ape', 'nut' }
Bag{1, 3, 4, 3, 5 }
Sequence{ 1..(5+4) } =
Sequence{ 1.. 9 } =
Sequence{ 1, 2, 3, 4, 5, 6,
   7, 8, 9 }
```

Traditional operations are defined (union, intersection, etc.)

- Conversion from Collection:
  - `c->asSet()`: Set
    A set corresponding to the collection (duplicates are dropped, sequencing is lost).
  - `c->asSequence()`: Sequence
    A sequence corresponding to the collection.
  - `c->asBag()`: Bag
    A bag corresponding to the collection.
- Comments:
  - `--`

OCL – OBJECT CONSTRAINT LANGUAGE

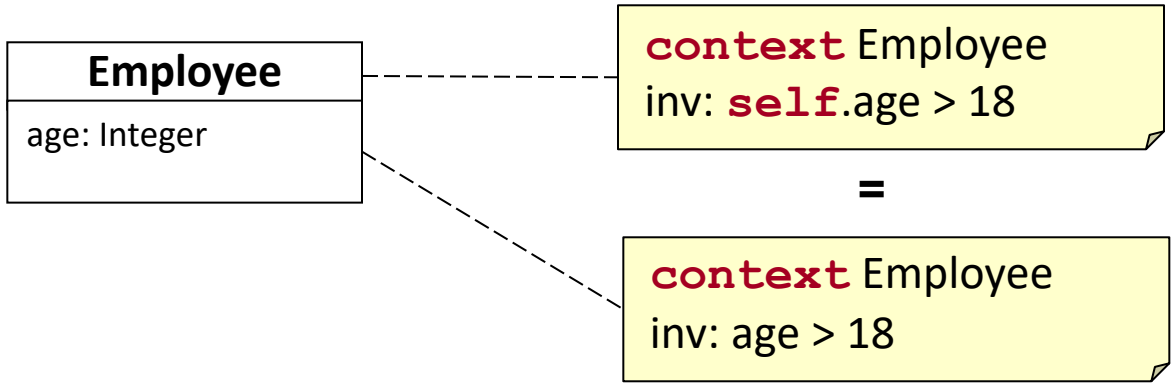
OCL Topics

- Introduction
- OCL Core Language
- OCL Standard Library
- Tool Support
- Examples

Design of OCL

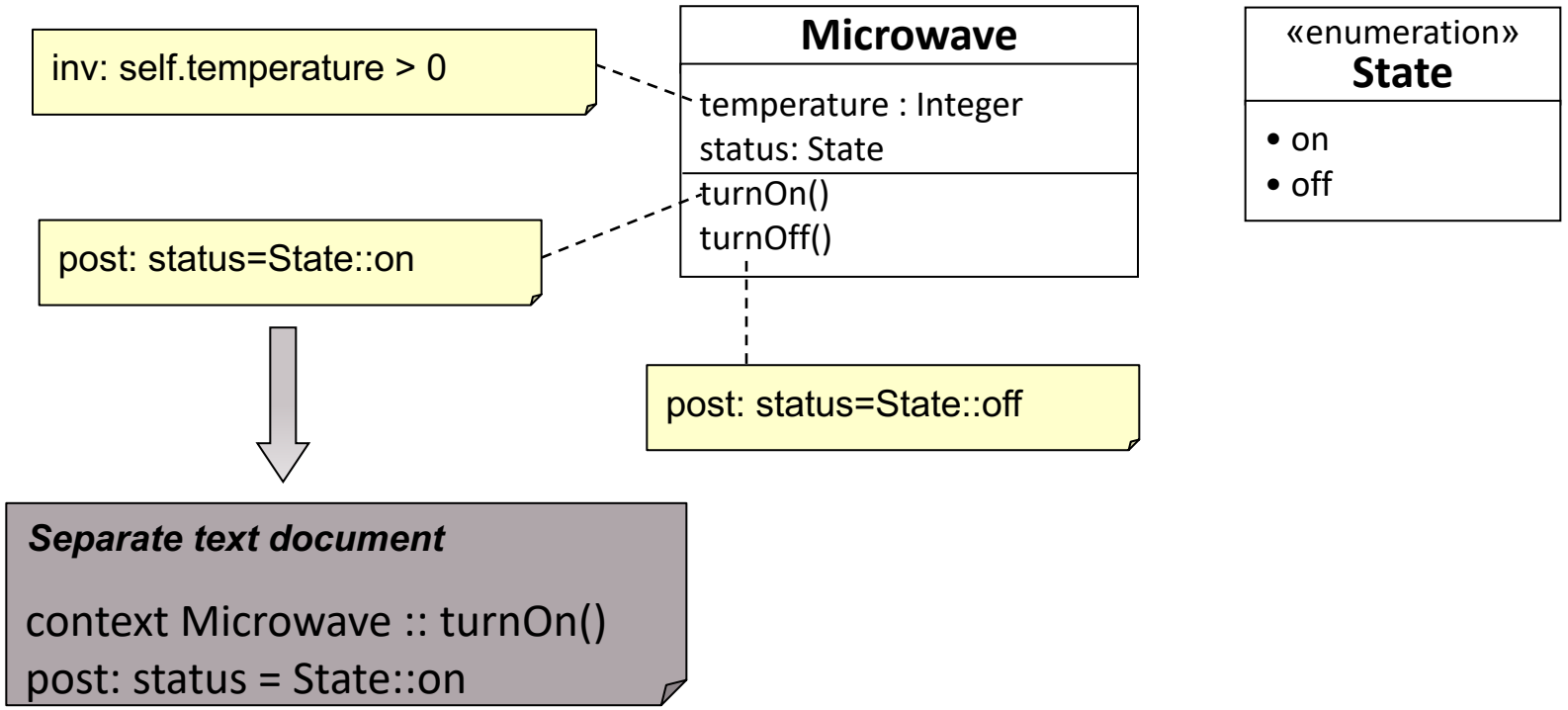
- A context has to be assigned to each OCL-statement
  - **Starting address –** which model element is the OCL-statement defined for
  - Specifies which model elements can be reached using path expressions
- The context is specified by the keyword **`context`** followed by the name of the model element (mostly class names)
- The keyword **`self`** specifies the current instance, which will be evaluated by the invariant (context instance).
  - **`self`** can be omitted if the context instance is unique

- Example:



Design of OCL

- OCL can be specified in **two** different ways
  - As a comment **directly** in the class diagram (context described by connection)
  - Separate document file



Types

- **OCL** is a typed language
  - Each **object, attribute,** and **result** of an operation or navigation is assigned to a **range of values** (type)

- **Predefined types**
  - **Basic types**
    - Simple types: *Integer, Real, Boolean, String*
    - OCL-specific types: *AnyType, TupleType, InvalidType, …*
  - **Set-valued, parameterized Types**
    - Abstract supertyp: Collection(T)
    - Set(T) – no duplicates
    - Bag(T) – duplicates allowed
    - Sequence(T) – Bag with ordered elements, association ends {*ordered*}
    - OrderedSet(T) – Set with ordered elements, association ends {*ordered, unique*}

- **Userdefined Types**
  - Instances of *Class* in MOF and indirect instances of *Classifier* in UML are types
  - *EnumerationType* – user defined set of values for defining constants

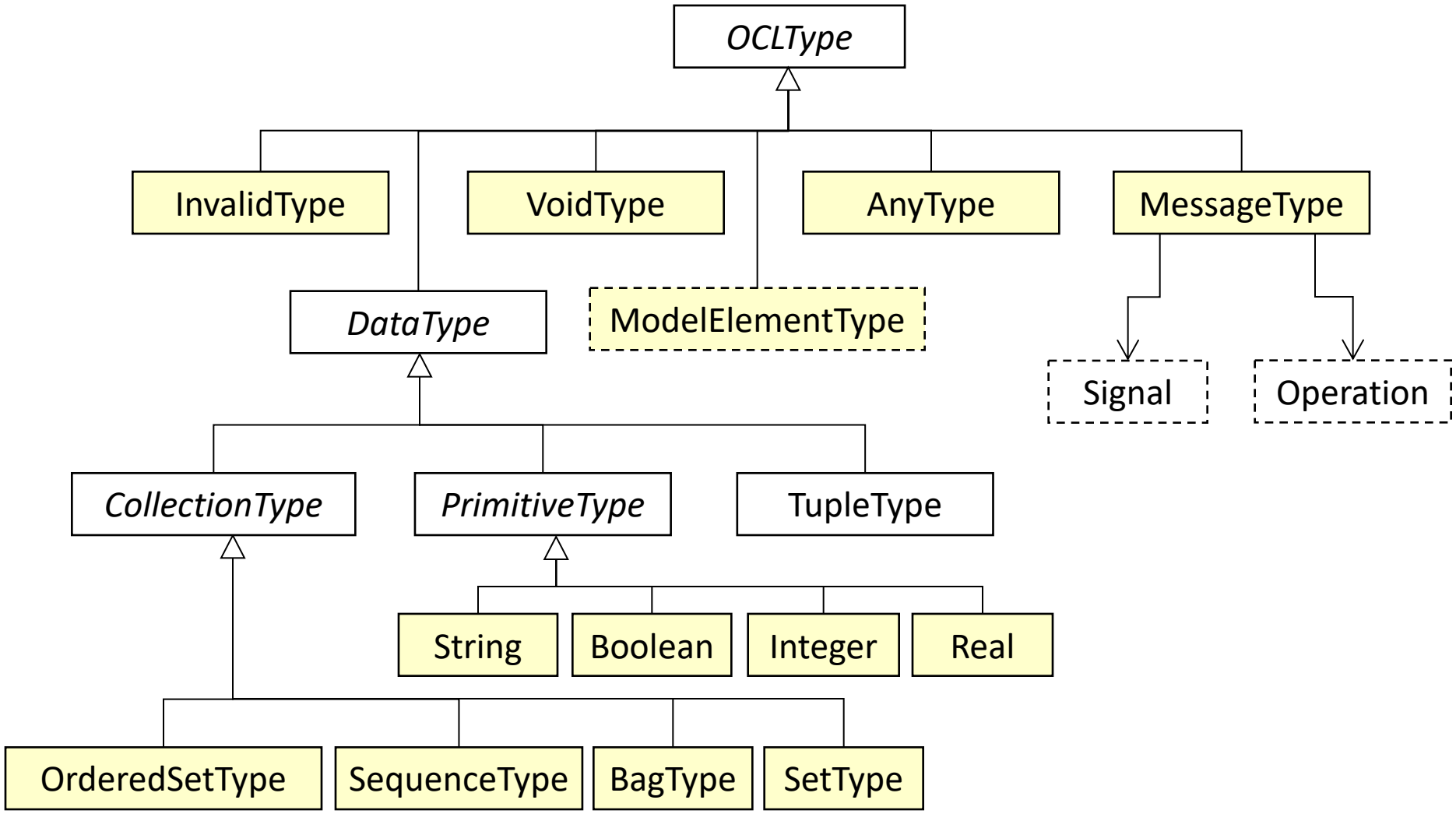
Types

Examples

- **Basic types**
  - true, false : *Boolean*
  - -17, 0, 1, 2 : *Integer*
  - -17.89, 0.01, 3.14 : *Real*
  - “Hello World” : *String*
- **Set-valued, parameterized types**
  - Set{ Set{1}, Set{2, 3} } : *Set(Set(Integer))*
  - Bag{ 1, 2.0, 2, 3.0, 3.0, 3 } : *Bag(Real)*
  - Tuple{ x = 5, y = false } : *Tuple{x: Integer, y : Boolean}*
- **Userdefined types**
  - Passenger : *Class*, Flight : *Class*, Provider : *Interface*
  - Status::started - enum Status {started, landed}

Types

OCL meta model (extract)



Expressions

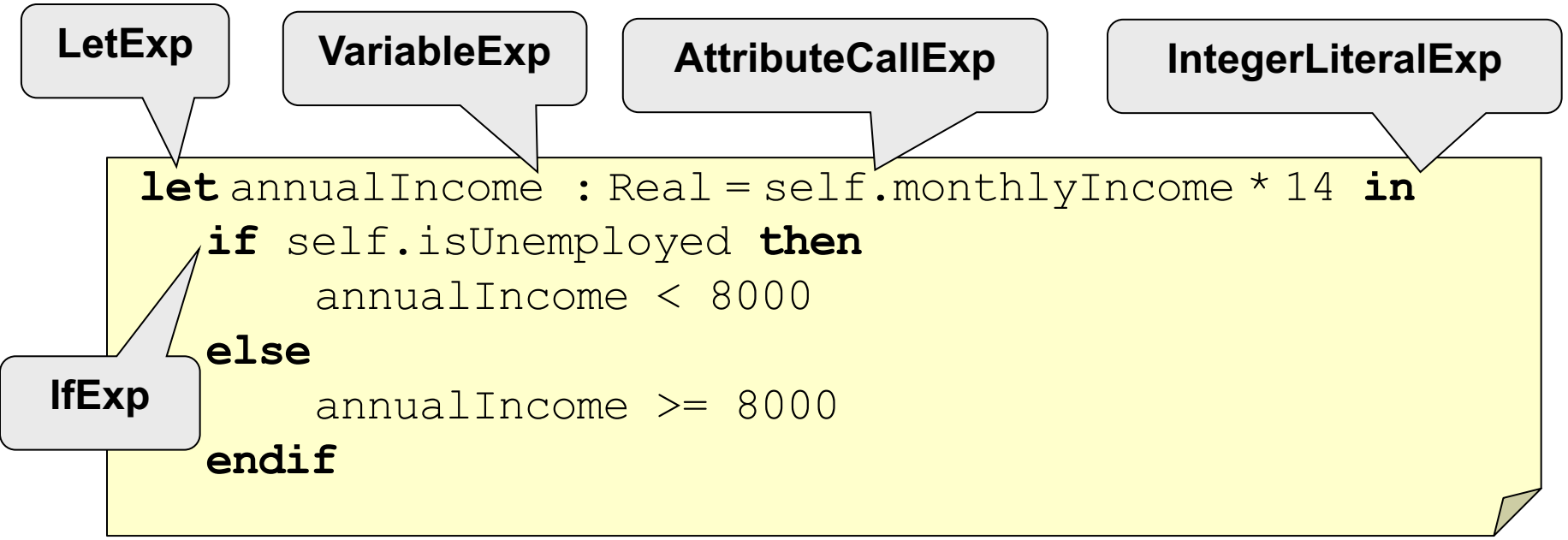
- Each OCL expression is an indirect instance of *OCLExpression*
  - Calculated in certain environment – cf. context
  - Each OCL expression has a **typed return value**
  - **OCL Constraint is an OCL expression with return value Boolean**

- **Simple OCL expressions**
  - *LiteralExp, IfExp, LetExp, VariableExp, LoopExp*

- **OCL expressions for querying model information**
  - *FeatureCallExp* – abstract superclass
  - *AttributeCallExp* – querying attributes
  - *AssociationEndCallExp* – querying association ends
    - Using role names; if no role names are specified, lowercase class names have to be used (if unique)
  - *AssociationClassCallExp* – querying association class (only in UML)
  - *OperationCallExp* – Call of query operations
    - Calculate a value, but do **not** change the system state!

Expressions

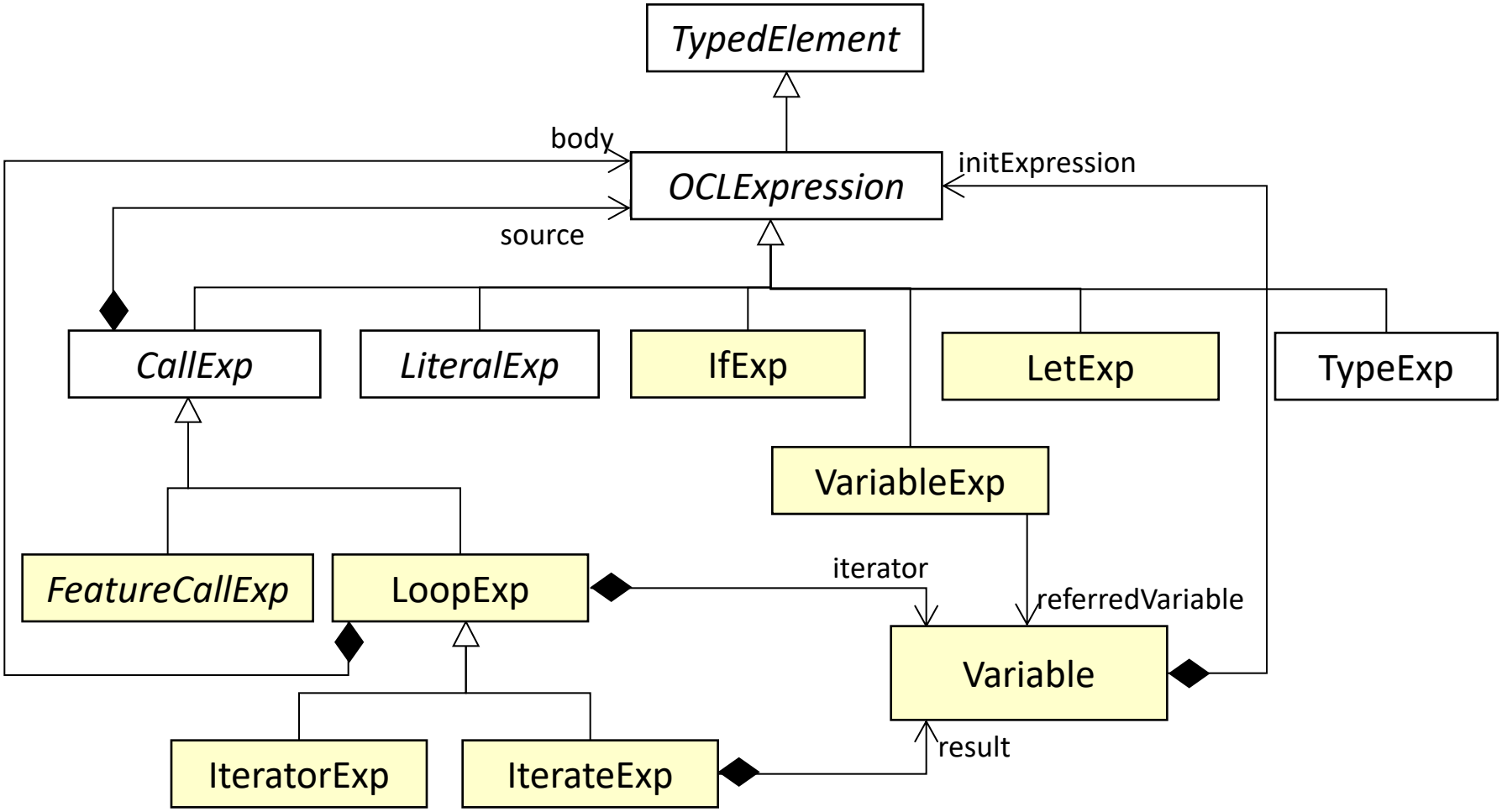
- Examples for *LiteralExp, IfExp, VariableExp, AttributeCallExp*



- **Abstract syntax** of OCL is described as **meta model**
- **Mapping from abstract syntax to concrete syntax**
  - *IfExp -> **if** Expression **then** Expression **else** Expression **endif***

Expressions

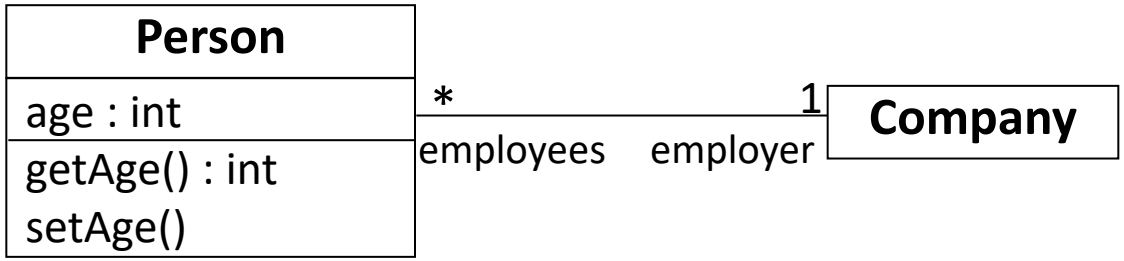
OCL meta model (extract)



LiteralExp: *CollectionLiteralExp, PrimitiveLiteralExp, TupleLiteralExp, EnumLiteralExp*

Query of model information

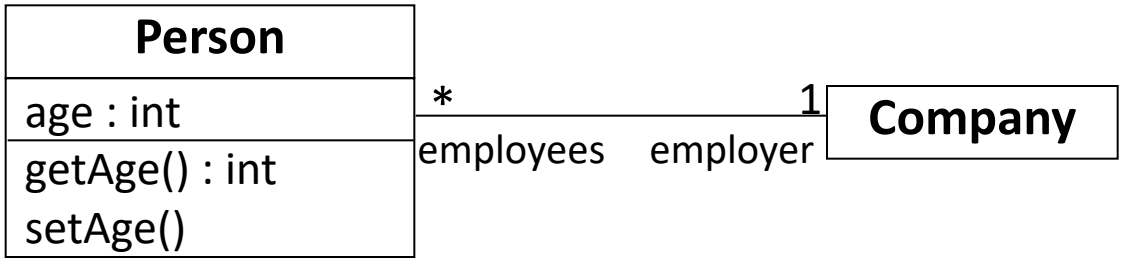
- Context instance
  - `context` **`Person`**
- AttributeCallExp
  - `self.age : int`

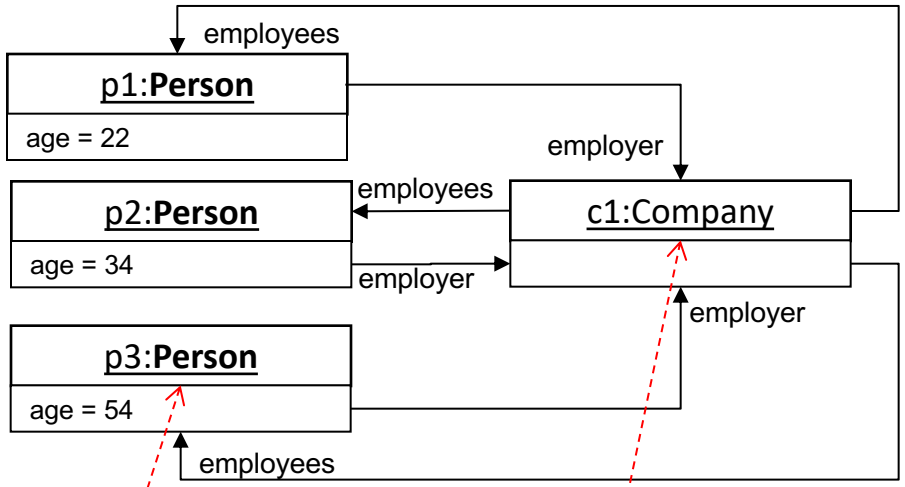


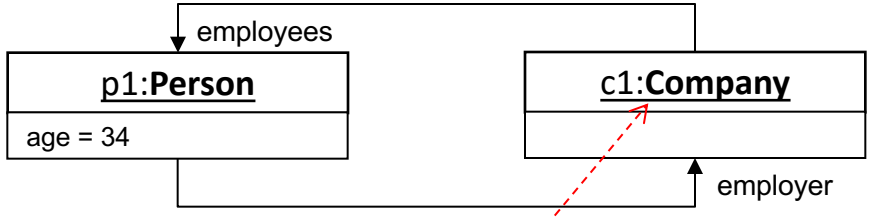
- OperationCallExp
  - Operations must not have **side effects**
  - Allowed: `self.getAge() : int`
  - **Not allowed**: `self.setAge()`
- AssociationEndCallExp
  - Navigate to the opposite association end using role names
    `self.employer` – Return value is of type ***Company***
  - Navigation often results into a set of objects – Example
    `context` **`Company`**
    `self.employees` – Return value is of type **Set (Person)**

Query of model information

Example







context Person
self.employer

`c1 : Company`

context Company
self.employees

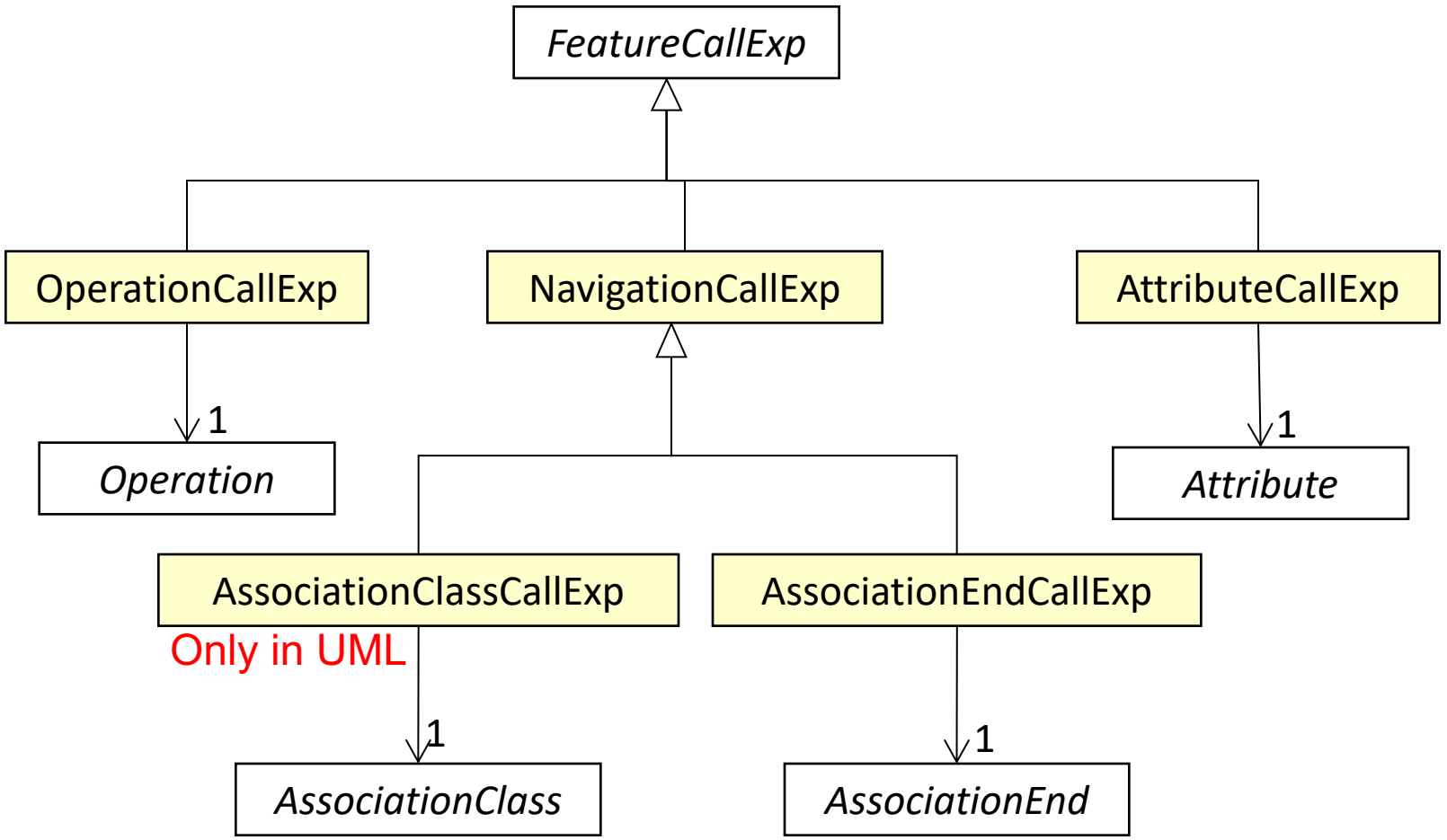
`Set{p1,p2,p3} : Set(Person)`

context Company
self.employees

`Set{p1} : Set(Person)`

Query of model information

OCL meta model (extract)



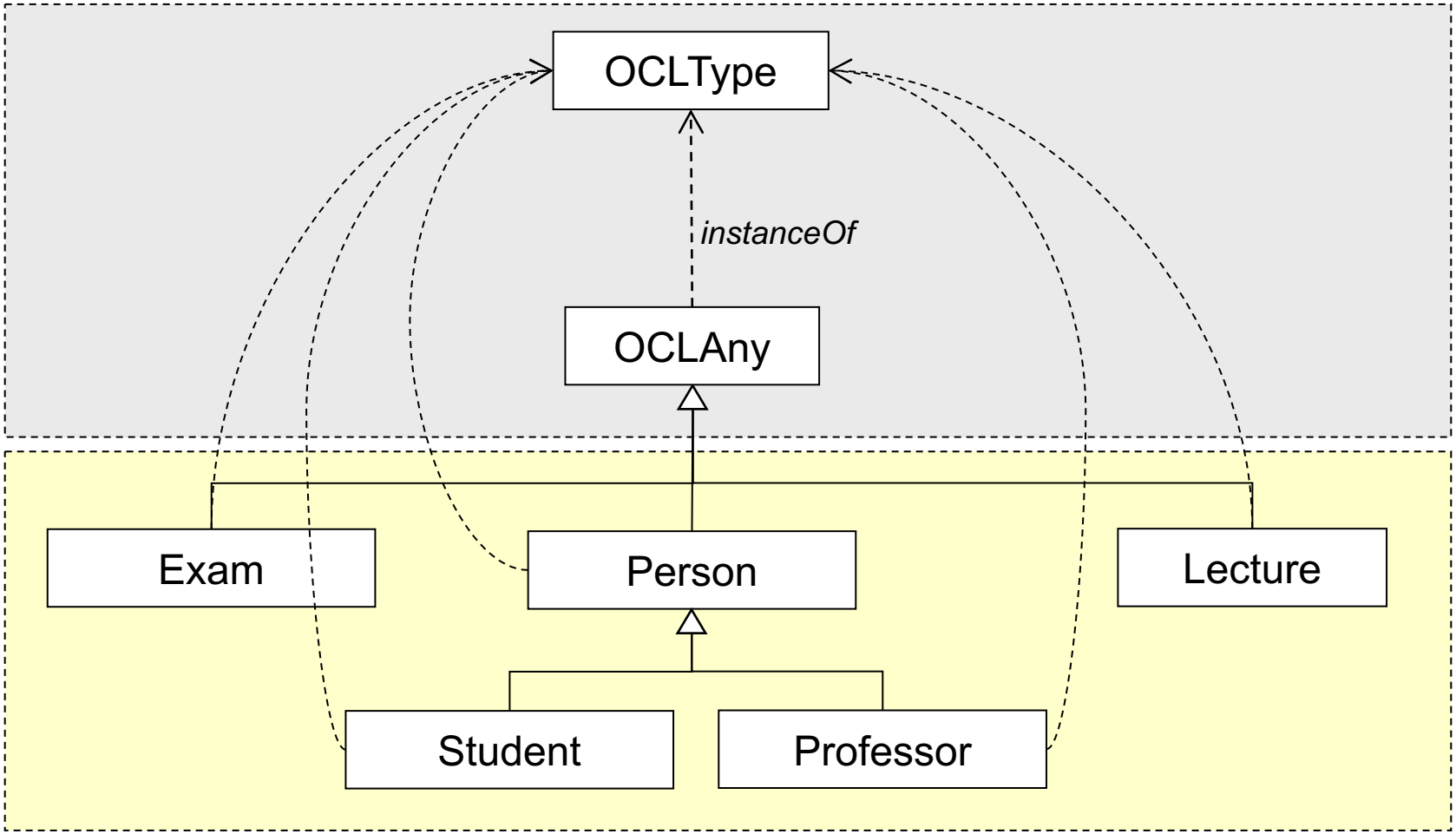
OCL Library: Operations for OclAny

- *OclAny* - **Supertype** of all other types in OCL
  - **Operations** are **inherited** by all other types.
- **Operations** of *OclAny* (extract)
  - Receiving object is denoted by *obj*

Operation	**Explanation of result**
=(obj2:OclAny):Boolean	True, if *obj2* and *obj* reference the same object
oclIsTypeOf(type:OclType):Boolean	True, if *type* is the type of *obj*
oclIsKindOf(type:OclType): Boolean	True, if *type* is a direct or indirect supertype or the type of *obj*
oclAsType(type:Ocltype): Type	The result is *obj* of type *type*, or *undefined*, if the current type of *obj* is not *type* or a direct or indirect subtype of it (casting)

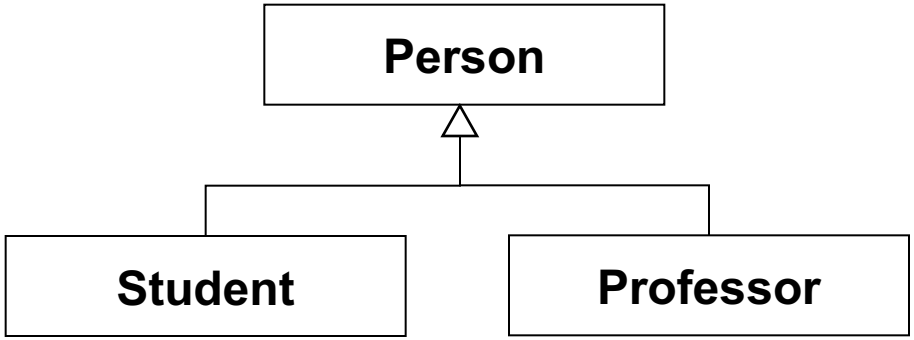
Operations for OclAny

Predefined environment for model types



Operations for OclAny

- ***oclIsKindOf vs. oclIsTypeOf***



context **Person**
self.oclIsKindOf(Person) : *true*
self.oclIsTypeOf(Person) : *true*
self.oclIsKindOf(Student) : *false*
self.oclIsTypeOf(Student) : *false*

context **Student**
self.oclIsKindOf(Person) : *true*
self.oclIsTypeOf(Person) : *false*
self.oclIsKindOf(Student) : *true*
self.oclIsTypeOf(Student) : *true*
self.oclIsKindOf(Professor) : *false*
self.oclIsTypeOf(Professor) : *false*

Operations for simple types

- **Predefined** simple types
  - Integer {Z}
  - Real {R}
  - Boolean {true, false}
  - String {ASCII, Unicode}

- Each simple type has predefined operations

Simple type	Predefined operations
Integer	*, +, -, /, abs(), …
Real	*, +, -, /, floor(), …
Boolean	and, or, xor, not, implies
String	concat(), size(), substring(), …

Operations for simple types

- Syntax
  - v.operation(para1, para2, …)
    - Example: “bla”.**concat**(“bla”)
  - Operations without brackets (Infix notation)
    - Example: 1 **+** 2, true **and** false

Signature	**Operation**
Integer **X** *Integer* → *Integer*	{+, -, *}
t1 **X** *t2* → *Boolean*	{<,>,≤,≥}, *t1, t2* typeOf {Integer or Real}
Boolean **X** *Boolean* → *Boolean*	{and, or, xor, implies}

Operations for simple types

Boolean operations - semantic

- OCL is based on a **three-valued (trivalent) logic**
  - Expressions are mapped to the three values {true, false, undefined}
- Semantic of the operations
  - $\mathcal{M}$(I, *exp*)= *I(exp*), if exp not further resolvable
  - $\mathcal{M}$(I, **not** *exp*)= ¬$\mathcal{M}$ (I, *exp*)
  - $\mathcal{M}$(I,(*exp1* **and** *exp2*)) = $\mathcal{M}$(I, *exp1*) $\wedge$ $\mathcal{M}$(I, *exp2*)
  - $\mathcal{M}$(I,(*exp1* **or** *exp2*)) = $\mathcal{M}$(I, *exp1*) $\vee$ $\mathcal{M}$(I, *exp2*)
  - $\mathcal{M}$(I,(*exp1* **implies** *exp2*)) = $\mathcal{M}$(I, *exp1*) $\rightarrow$ $\mathcal{M}$(I, *exp2)*
- Truth table: true(1), false (0),undefined (?)

Undefined: Return value if an expression fails

1. Access on the first element of an empty set
2. Error during *Type Casting*
3. …

¬	
0	1
1	0
?	?

$\wedge$	0	1	?
0	0	0	0
1	0	1	?
?	0	?	?

$\vee$	0	1	?
0	0	1	?
1	1	1	1
?	?	1	?

$\rightarrow$	0	1	?
0	1	1	1
1	0	1	?
?	?	1	?

Operations for simple types

Boolean operations - semantic

- Simple example for an **undefined** OCL expression
  - *1/0*
- **Query** if undefined– OCLAny.oclIsUndefined()
  - *(1 / 0).oclIsUndefined() : true*
- Examples for the evaluation of Boolean operations
  - *(1/0 = 0.0)* ***and*** *false : false*
  - *(1/0 = 0.0)* ***or*** *true : true*
  - *false* ***implies*** *(1.0 = 0.0) : true*
  - *(1/0 = 0.0) implies true : true*

Operations for collections

- Collection is an **abstract supertype** for all set types
  - Specification of the **mutual** operations
  - *Set, Bag, Sequence, OrderedSet* inherit these operations
- **Caution:** Operations with a return value of a set-valued type create a new collection (no side effects)
- Syntax: v -> op(…) – Example: {1, 2, 3} -> size()

- Operations of collections (extract)
  - Receiving object is denoted by *coll*

Operation	**Explanation of result**
size():Integer	Number of elements in *coll*
includes(obj:OclAny):Boolean	True, if *obj* exists in *coll*
isEmpty:Boolean	True, if *coll* contains no elements
sum:T	Sum of all elements in *coll* Elements have to be of type Integer or Real

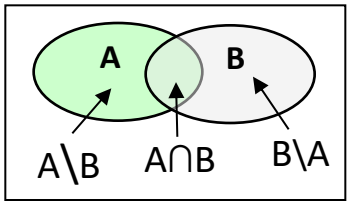
Operations for collections

- **Model operations vs. OCL operations**

OCL-Constraint	*Semantic*
context Container inv: self.content -> first().isEmpty()	Operation *isEmpty()* always has to return true
context Container inv: self.content -> isEmpty()	Container instances must not contain bottles

Operationen for Set/Bag

- *Set* and *Bag* define additional operations
  - Generally based on **theory of set concepts**
- **Operations** of **Set** (extract)
  - Receiving object is denoted by set



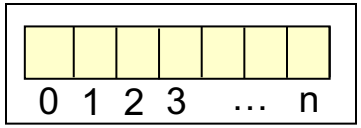
Operation	Explanation of result
union(set2:Set(T)):Set(T)	Union of *set* and *set2*
intersection(set2:Set(T)):Set(T)	Intersection of *set* and *set2*
difference(set2:Set(T)):Set()	Difference set; elements of *set*, which do not consist in *set2*
symmetricDifference(set2:Set(T)):Set(T)	Set of all elements, which are either in *set* or in *set2*, but do not exist in both sets at the same time

- **Operations** of **Bag** (extract)
  - Receiving object is denoted by bag

Operation	Explanation of result
union(bag2:Bag(T)):Bag(T)	Union of *bag* and *bag2*
intersection(bag2:Bag(T)): Bag(T)	Intersection of *bag* and *bag2*

Operations for OrderedSet/Sequence

- *OrderedSet* and *Sequences* define additional operations
  - Allow access or modification through an **Index**



- **Operations** of **OrderedSet** (extract)
  - Receiving object is denoted by *orderedSet*

Operation	**Explanation of result**
first:T	First element of *orderedSet*
last:T	Last element of *orderedSet*
at(i:Integer):T	Element on index i of *orderedSet*
subOrderedSet(lower:Integer, upper:Integer):OrderedSet(T)	Subset of *orderedSet,* all elements of *orderedSet* including the element on position *lower* and the element on position *upper*
insertAt(index:Integer,object:T) :OrderedSet(T)	Result is a copy of the *orderedSet*, including the element *object* at the position *index*

- **Operations** of **Sequence**
  - Analogous to the operations of OrderedSet

Iterator-based operations

- OCL defines operations for *Collections* using *Iterators*
  - Expression Package: LoopExp
  - **Projection** of new *Collections* out of existing ones
  - Compact **declarative specification** instead of imperative algorithms

- Predefined Operations
  - select(exp) : *Collection*
  - reject(exp) : *Collection*
  - collect(exp) : *Collection*
  - forAll(exp) : *Boolean*
  - exists(exp) : *Boolean*
  - isUnique(exp) : *Boolean*

- iterate(…) – Iterate over all elements of a *Collection*
  - Generic operation
  - Predefined operations are defined with iterate(…)

Iterator-based operations

Select-/Reject-Operation

- **Select** and **Reject** return subsets of collections
  - Iterate over the complete collection and collect elements
- Select
  - **Result**: Subset of collection, including elements where *booleanExpr* is **true**

```
collection -> select( v : Type | booleanExp(v) )
collection -> select( v | booleanExp(v) )
collection -> select( booleanExp )
```

- Reject
  - **Result**: Subset of collection, including elements where *booleanExpr* is **false**
  - Just *Syntactic* Sugar, because each *reject-Operation* can be defined as a *select-Operation* with a negated expression

```
collection-> reject(v : Type | booleanExp(v))
```

=

```
collection-> select(v : Type | not (booleanExp(v))
```


Iterator-based operations

Select-/Reject-Operation

- Semantic of the *Select-Operation*

OCL

```
context Company inv:
  self.employee -> select(e : Employee | e.age>50)  -> notEmpty()
```

Java

```
List persons<Person> = new List();
for ( Iterator<Person> iter = comp.getEmployee();
iter.hasNext() ){
    Person p = iter.next();
    if ( p.age > 50 ){
        persons.add(p);
    }
}
```


Iterator-based operations

Collect-Operation

- *Collect-Operation* returns a new collection from an existing one. It collects the **Properties** of the objects and not the objects itself.
  - Result of *collect* always **Bag<T>.T** defines the type of the property to be collected

```
collection -> collect( v : Type | exp(v) )
collection -> collect( v | exp(v) )
collection -> collect( exp )
```

- Example
  - *self.employees -> collect(age)* – Return type: Bag(Integer)
- Short notation for collect
  - *self.employees.age*

Iterator-based operations

Collect-Operation

- Semantic of the *Collect-Operator*

OCL

```
context Company inv:
  self.employee -> collect(birthdate)   -> size() > 3
```

Java

```
List birthdate<Integer> = new List();
for ( Iterator<Person> iter = comp.getEmployee();
iter.hasNext() ){
    birthdate.add(iter.next().getBirthdate()); }
```

- Use of *asSet()* to eliminate duplicates

OCL

```
context Company inv:
  self.employee -> collect(birthdate)   -> asSet()
```

Set (without duplicates)

Bag (with duplicates)

Iterator-based operations

ForAll-/Exists-Operation

- **ForAll** checks, if all elements of a collection evaluate to true

```
collection -> forAll( v : Type | booleanExp(v) )
collection -> forAll( v | booleanExp(v) )
collection -> forAll( booleanExp )
```

- **Example:** self.employees -> forAll(age > 18)

- **Nesting** of forAll-Calls (*Cartesian Product*)

```
context Company inv:
self.employee->forAll (e1 | self.employee -> forAll (e2 |
        e1 <> e2 implies e1.svnr <> e2.svnr))
```

- ***Alternative***: Use of multiple iterators

```
context Company inv:
self.employee -> forAll (e1, e2 | e1 <> e2 implies e1.svnr <> e2.svnr))
```

- **Exists** checks, if at least one element evaluates to true
  - Beispiel: employees -> exists(e: Employee | e.isManager = true)

Iterator-based operations

Iterate-Operation

- **Iterate** is the generic form of all iterator-based operations

- **Syntax**

  collection -> iterate(**elem** : Typ; **acc** : Typ =
  <initExp> | **exp(elem**, **acc**))

  - Variable **elem** is a typed ***Iterator***
  - Variable **acc** is a typed ***Accumulator***
  - Gets assigned initial value initExp
  - **exp(elem, acc)** is a function to calculate **acc**

- **Example**

  collection -> collect(x : T | x.property)

  -- semantically equivalent to:

  collection -> iterate(x : T; acc : T2 = Bag{} | acc -> including(x.property))

Iterator-based operations

Iterate-Operator

- Semantic of the *Iterate-Operator*

OCL

```
collection -> iterate(x : T; acc : T2 = value | acc -> u(acc, x)
```

Java

```
iterate (coll : T, acc : T2 = value){
      acc=value;
      for( Iterator<T> iter =
coll.getElements(); iter.hasNext(); ){
            T elem = iter.next();
            acc = u(elem, acc);
      }
}
```

- Example
  - Set{1, 2, 3} -> iterate(i:Integer, a:Integer=0 | a+i)
  - Result: 6

Tool Support

- **Wishlist**
  - Syntactic analysis: Editor support
  - Validation of logical consistency (Unambiguous)
  - Dynamic validation of invariants
  - Dynamic validation of Pre-/Post-conditions
  - Code generation and test automation

- **Today**
  - UML-tools provide OCL-editors
  - MDA-tools provide code generation of OCL-expressions
  - Meta modeling platforms provide the opportunity to define OCL Constraints for meta models.
    - The editor should dynamically check constraints or restrict modeling, respectively.

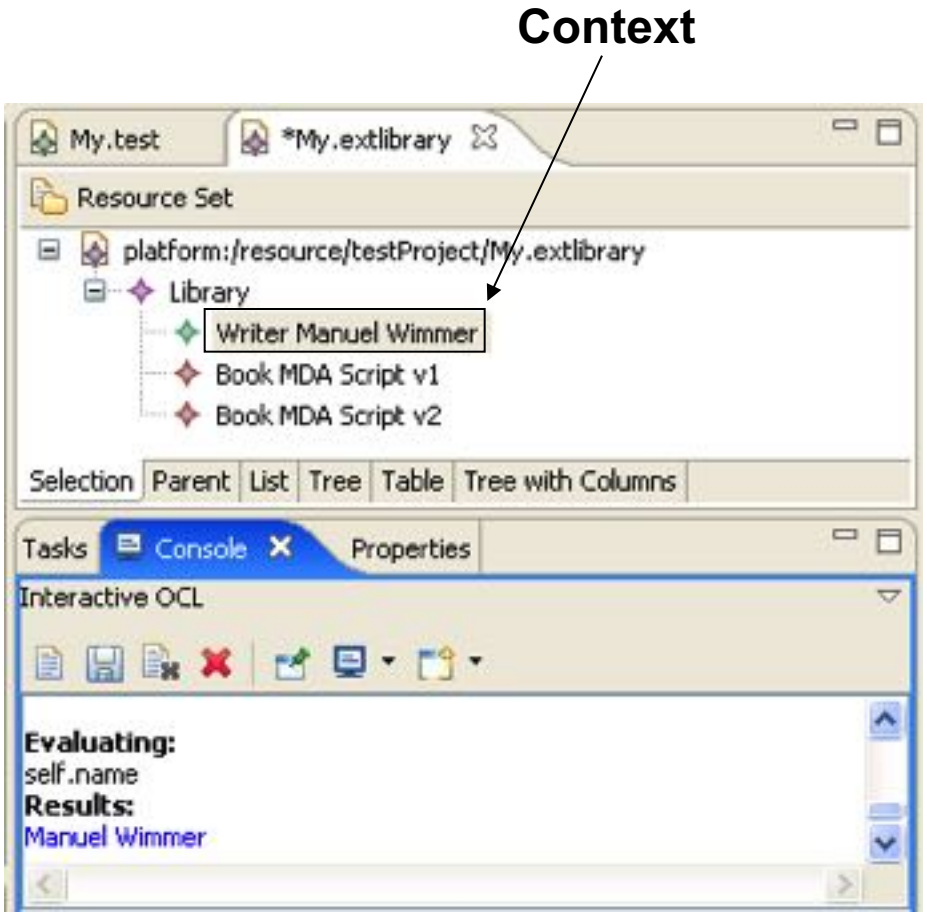
OCL Tools

- Some OCL-parsers, which check the syntax of OCL-constraints and apply them to the models, are for free.
  - IBM Parser
- Dresden OCL Toolkit 2.0
  - Generation of Java code out of OCL-constraints
  - Possible integration with ArgoUML
- OCL-frameworks are originated in the areas of EMF and the UML2 project of Eclipse
  - Octopus
  - Frauenhofer Toolkit
  - OSLO
  - EMFT OCL-Framework/Query-Framework

OCL-Tools

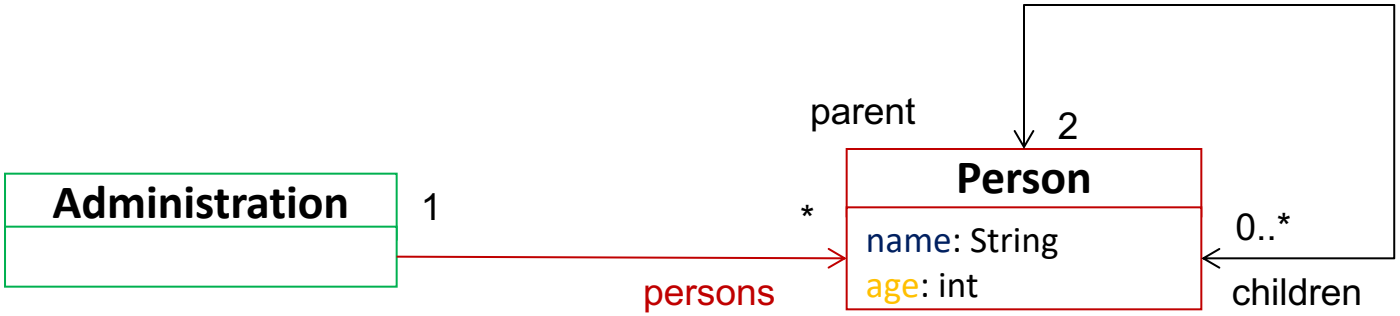
- **EMFT OCL-Framework**
  - Based on EMF
  - *OCL-API* – Enables the use of OCL in Java programs
  - *Interactive OCL Console* – Enables the definition and evaluation of OCL-constraints

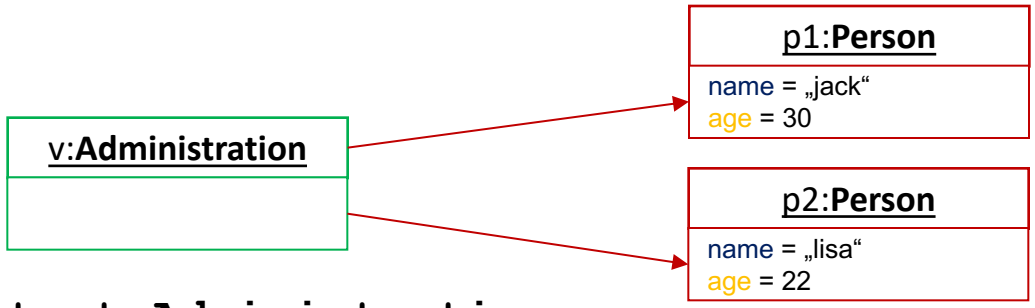
- **EMFT Query-Framework**
  - **Goal**: SQL-like query of model information
  - **select** exp **from** exp **where** *oclExp*



TUWEL: Interactive OCL Console Screencast

Example 1: Navigation (1)

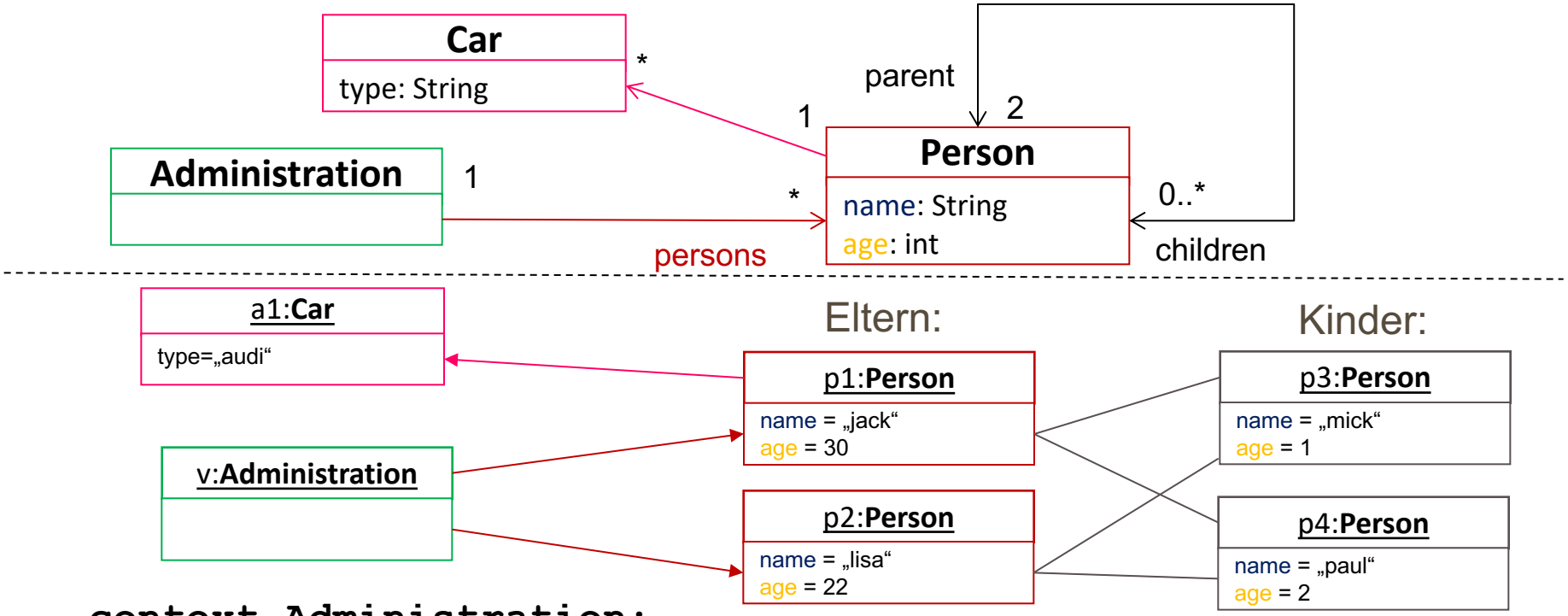





```
context Administration:
```

- `self.persons` → {Person p1, Person p2}
- `self.persons.name` → {jack, lisa}
- `self.persons.alter` → {30, 22}

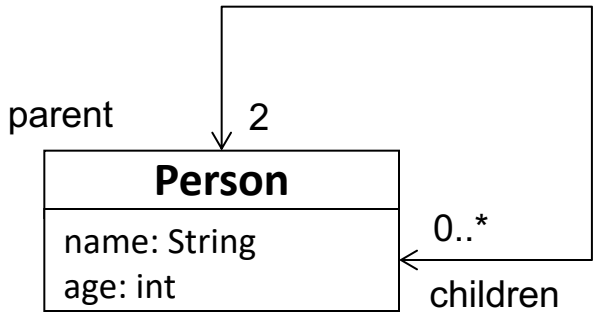
Example 1: Navigation (2)



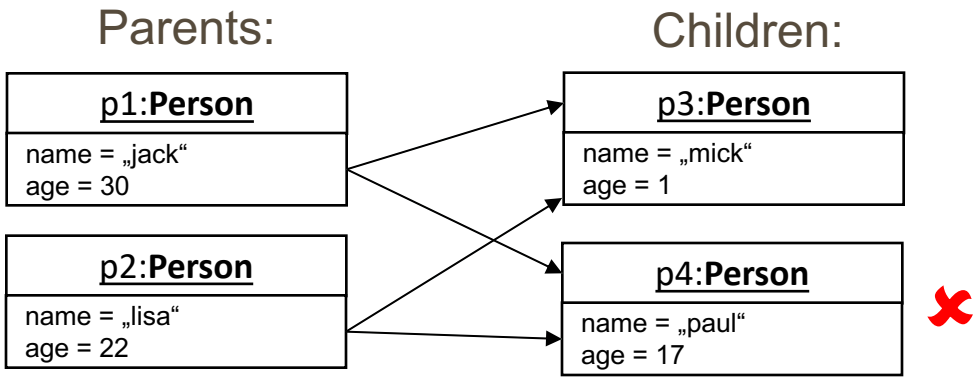

```
context Administration:
```

- `self.persons.children` → {{p3, p4}, {p3, p4}}
- `self.persons.children.parent` → {{{p1, p2}, {p1, p2}}, ...}
- `self.persons.car.type` → {{"audi"}}

Example 2: Invariant (1)

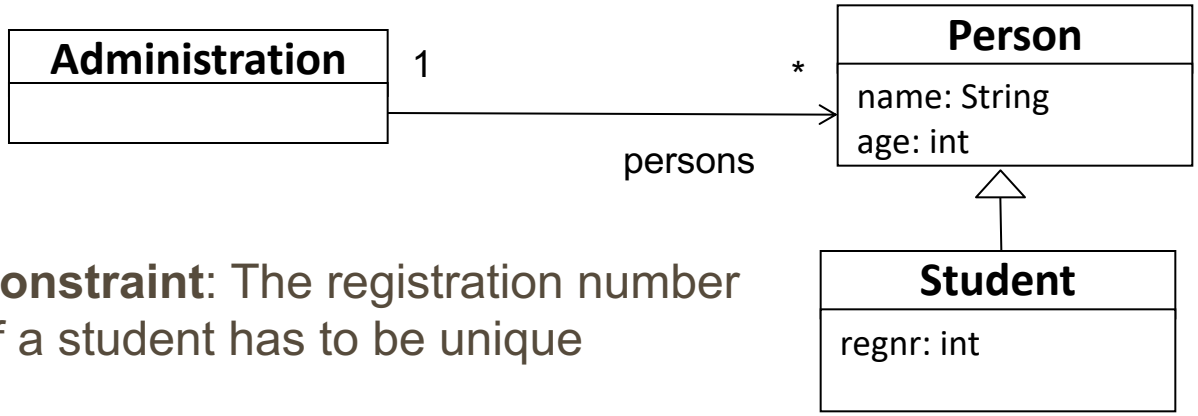


Constraint: A child is at least 15 years younger than his parents.




```
context Person
inv: self.children->forAll(k : Person | k.age
 < self.age-15)
```

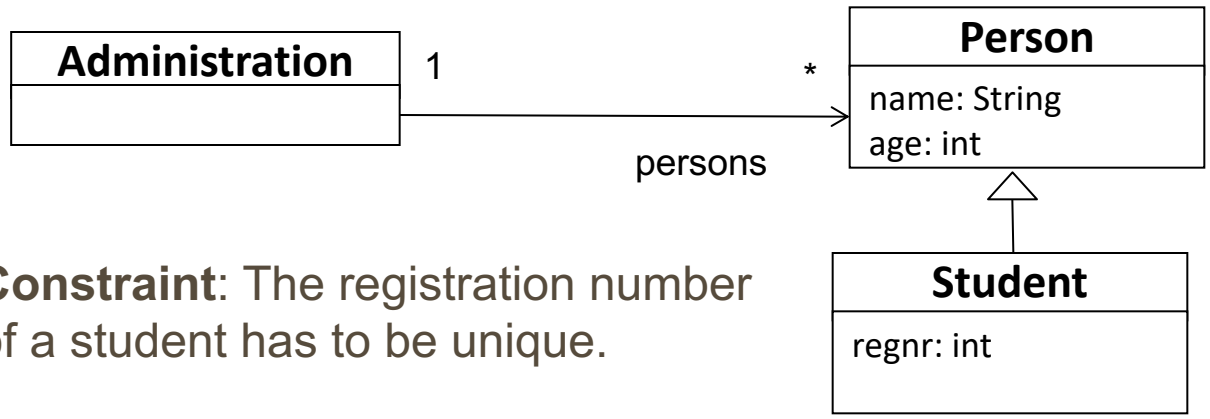
Example 2: Invariant (2)



Constraint: The registration number of a student has to be unique

```
context Administration
inv uniqueRegnr :
  self.persons -> select(e : Person| e.oclIsTypeOf(Student))
                       -> forAll(e1 |
   self.persons -> select(e : Person | e.oclIsTypeOf(Student))
                       -> forAll(e2 |
  e1 <> e2 implies   e1.oclAsType(Student).regnr       <>
                           e2.oclAsType(Student).regnr))
```


Example 2: Invariant (2) cont.

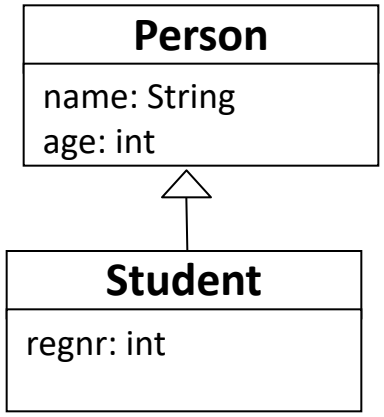


Constraint: The registration number of a student has to be unique.

```
context Administration
inv uniqueRegnr :
  self.persons -> select(e : Person| e.oclIsTypeOf(Student))
       -> forAll(e1, e1 | e1 <> e2 implies
       e1.oclAsType(Student).regnr <>
       e2.oclAsType(Student).regnr)
       )
```

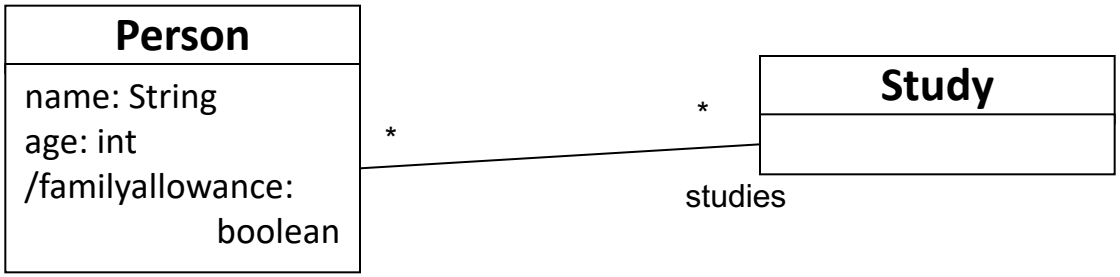

Example 2: Invariant (2) cont.

Constraint: The registration number of a student has to be unique.




```
context Student
inv uniqueRegnr :
  Student.allInstances() -> forAll(e1, e1 | e1 <> e2 implies
        e1.oclAsType(Student).regnr <>
        e2.oclAsType(Student).regnr))
```

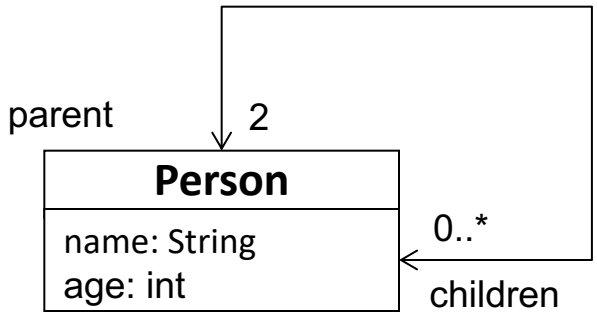

Example 3: Inherited attribute



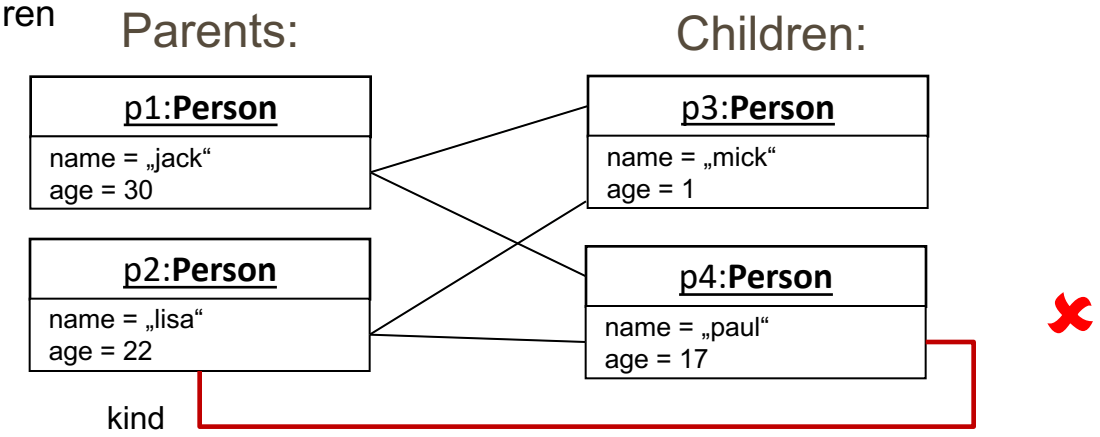
A Person obtains family allowance, if he/she is younger than 18 years, or if he/she is studying and younger than 27 years old.

```
context Person::familyallowance
derive: self.age < 18 or
        (self.age < 27 and self.studies -> size() > 0)
```


Example 4: Definitions



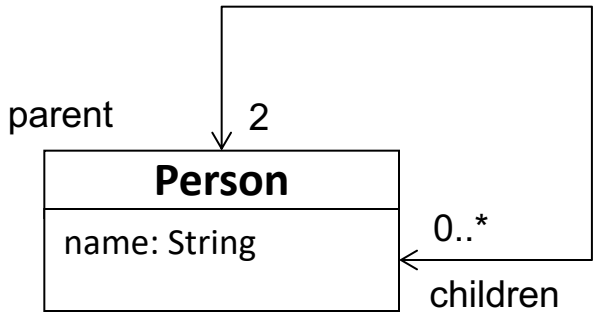
Constraint: A Person is not a relative of itself




```
context Person
def: relative: Set(Person) = children-> union(relative)
inv: self.relative -> excludes(self)
```

Assumption: Fixed-point semantic, otherwise if then else required

Example 5: equivalent OCL-formulations (1)



Constrain: A person is not its own child

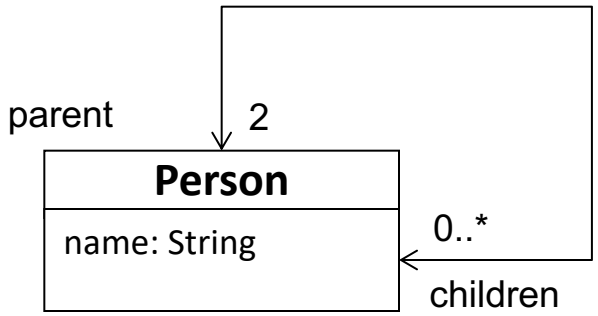
- `(self.children->select(k | k = self))->size() = 0`

  The Number of children for each person „self“, where the children are the person „self“, have to be 0.

- `(self.children->select(k | k = self))->isEmpty()`

  The set of children for each person „self, where the children are the person „self“, has to be empty.

Example 5: equivalent OCL-formulations (2)



Constrain: A person is not its own child

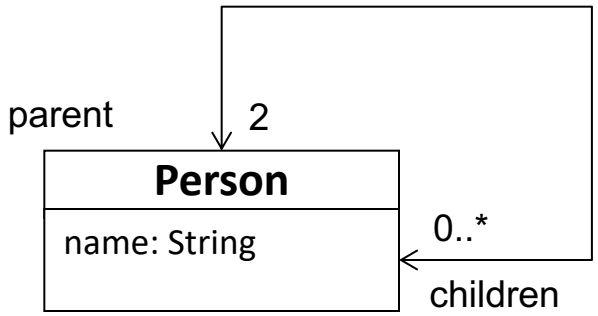
- `not self.children->includes(self)`

  It is not possible, that the set of children of each person „self“ contains the person „self“.

- `self.children->excludes(self)`

  `The set of children of each person „self“ cannot contain „self“.`

Example 5: equivalent OCL-formulations (3)



Constrain: A person is not its own child

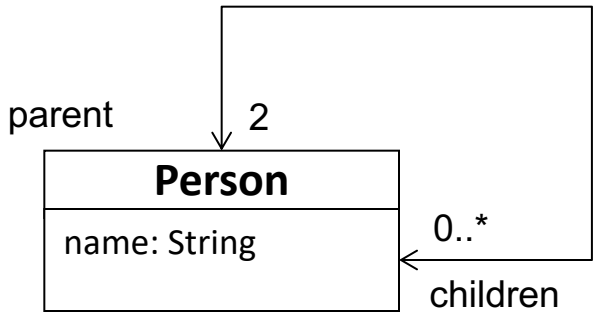
- `Set{self}->intersection(self.children)->isEmpty()`

  The intersection between the one element set, which only includes one person „self“ and the set of the children of „self“ has to be empty.

- `(self.children->reject(k | k <> self))->isEmpty()`

  The set of children for each person „self“, for whome does not apply, that they are not equal to the person „self“, has to be empty.

Example 5: equivalent OCL-formulations (4)



Constrain: A person is not its own child

- `self.children->forAll(k | k <> self)`

  Each child of the person „self“ is not the person „self“.

- `not self.children->exists(k | k = self)`

  There is no child for each person „self“, which is the person „self“

References on OCL

- **Literature**
  - Object Constraint Language Specification, Version 2.0
    - http://www.omg.org/technology/documents/formal/ocl.htm
  - Jos Warmer, Anneke Kleppe: The Object Constraint Language - Second Edition, Addison Wesley (2003)
  - Martin Hitz et al: UML@Work, d.punkt, 2. Auflage (2003)

- **Tools**
  - OSLO - http://oslo-project.berlios.de
  - Octopus - http://octopus.sourceforge.net
  - Dresden OCL Toolkit - http://dresden-ocl.sourceforge.net
  - EMF OCL - http://www.eclipse.org/modeling/mdt/?project=ocl