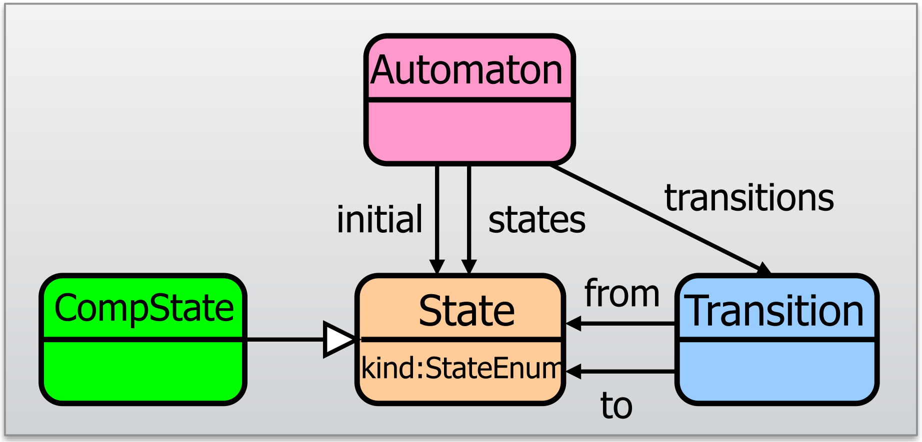
Domain-specific modeling
(and the Eclipse Modeling Framework)

Ákos Horváth
Gábor Bergmann
Dániel Varró
István Ráth

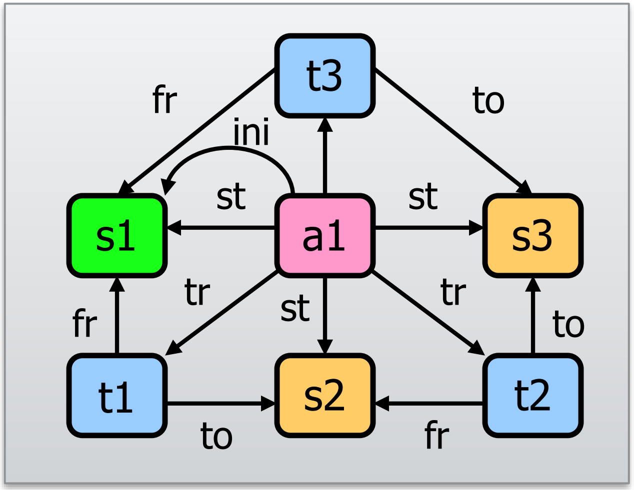
Model Driven Software Development
Lecture 2

METAMODELS, INSTANCE MODELS

Metamodel: Specify Concepts an Appl. Domain



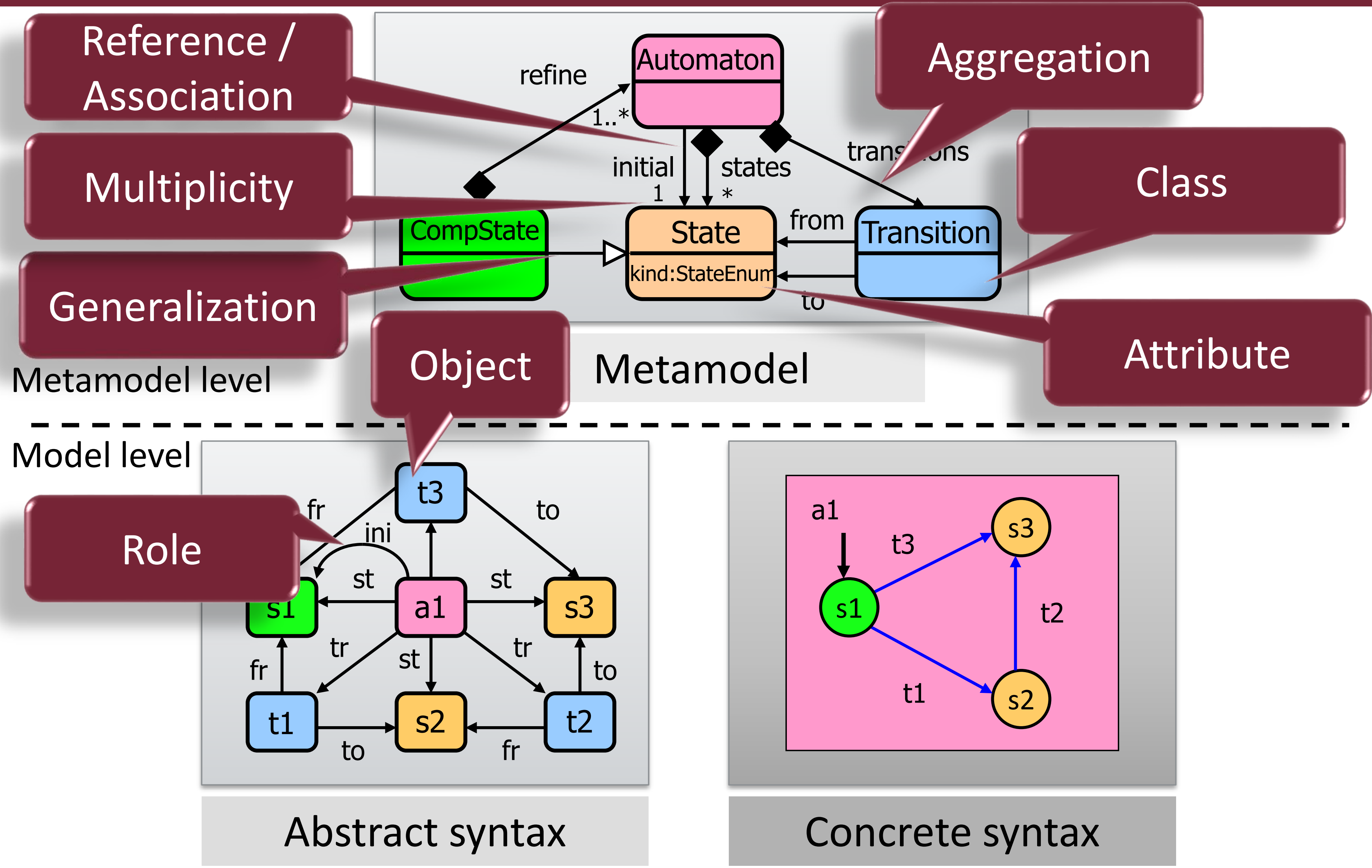
Metamodel



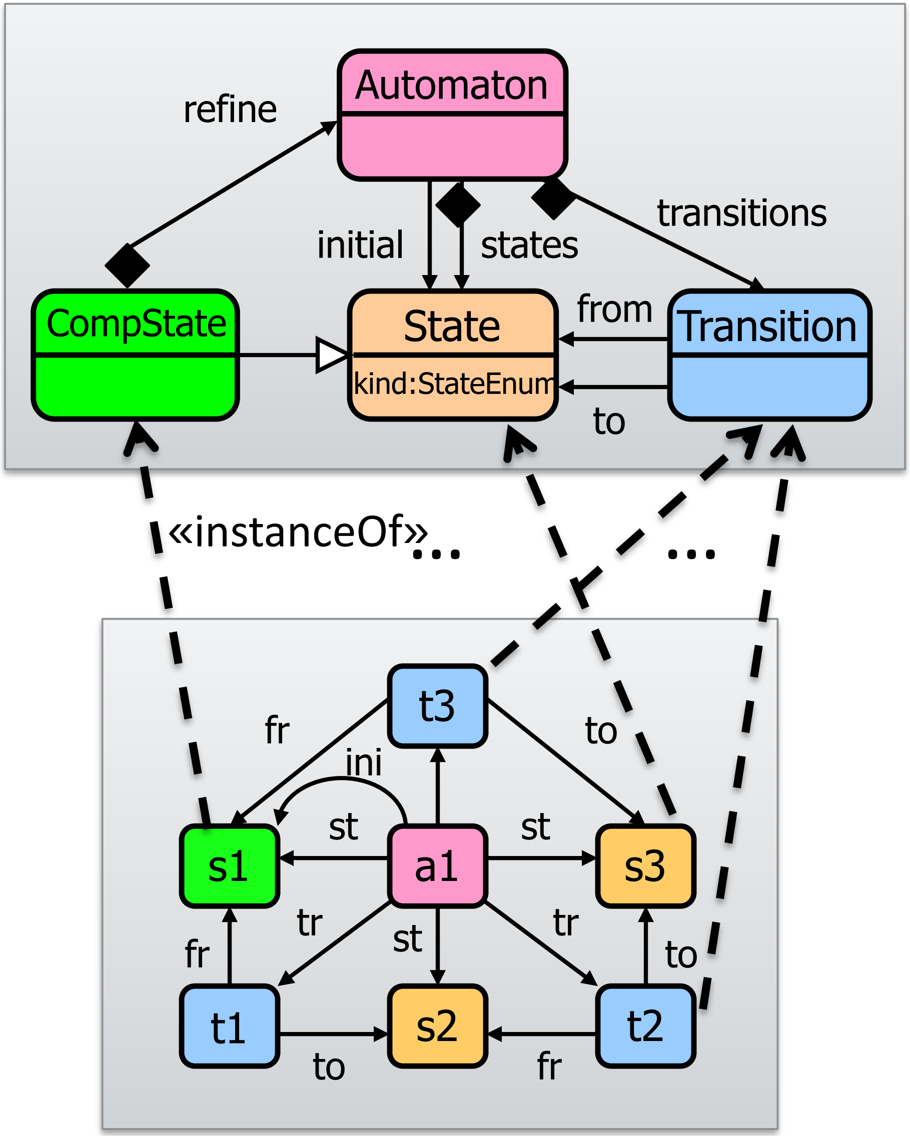
Instance model

- 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



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

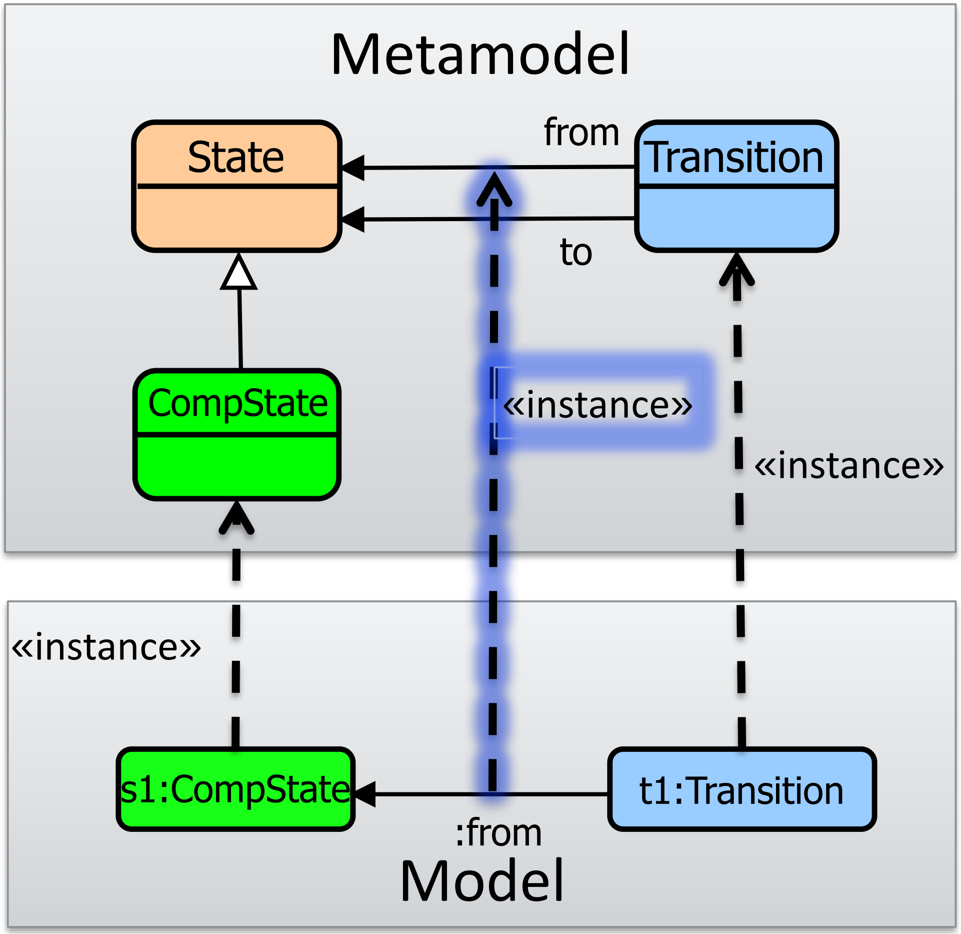
Classification vs. Generalization

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

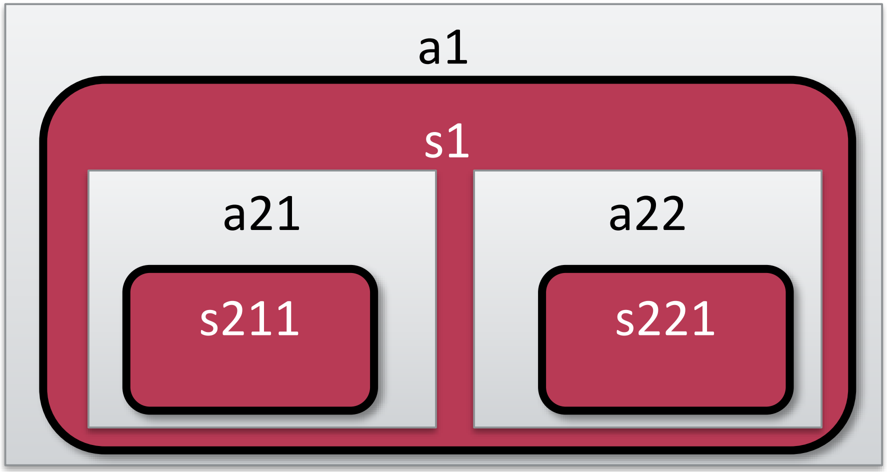
Type conformance of references

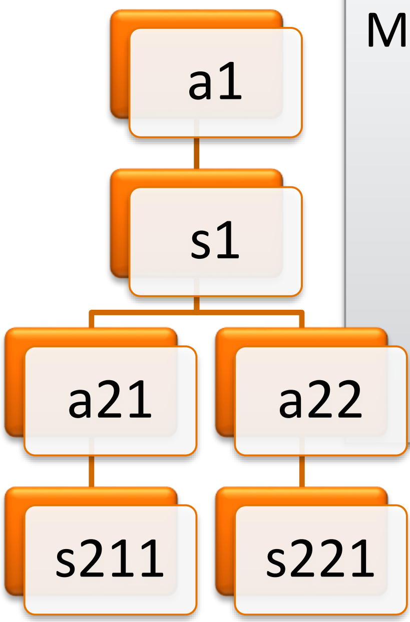


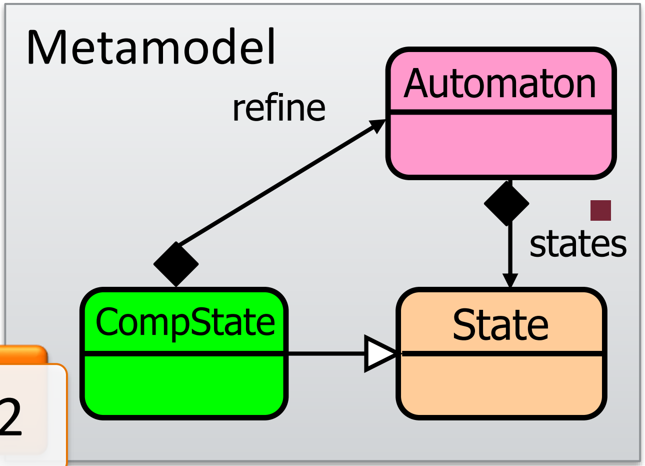
Can you define generalization for references?

- 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.

Containment hierarchy



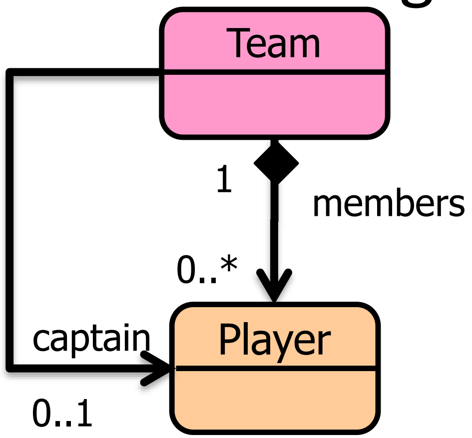




- Each model element has a unique parent
  - N children → 1 parent
  - Single root element
- Aggregation as relationship:
  - Defined in the metamodel along reference edges
  - Provides restriction for instance models
- Circularity
  - No circular containment (in the model)
  - Aggregation relations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

Multiplicity restrictions

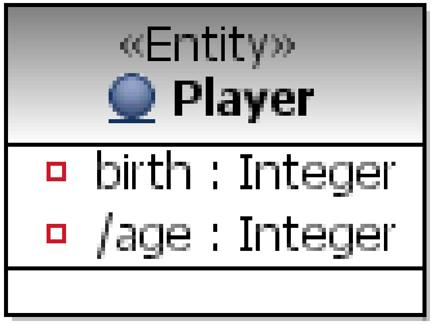
- Definition: Lower bound .. Upper bound
  - Lower bound: 0, 1, (non-negative integer)
  - Upper bound: 1, 2, … * (positive integer + any)
- Scope:
  - References: allowed number of links between objects of specific types
  - Attributes: e.g. arrays of strings (built-in values)

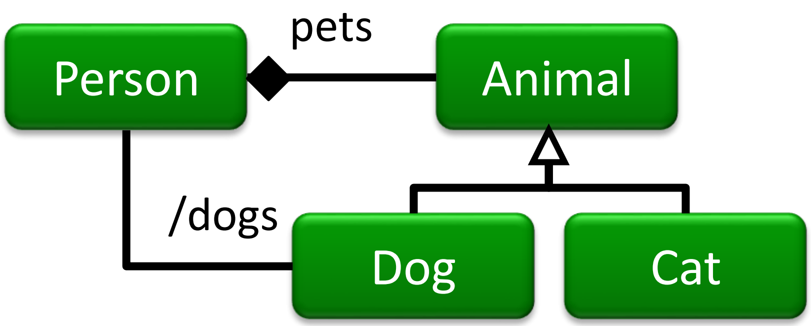


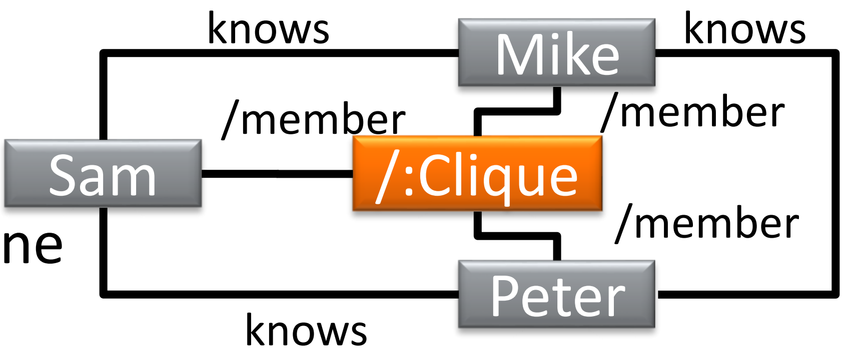
Which are the most common multiplicity definitions in practice?

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`
  - (typical: qualified name)
- Derived references:
  `dogs =  -- pets --> Dog`
  - (typical: inherited features)
- Derived objects:
  - „Clique”: everyone knows everyone

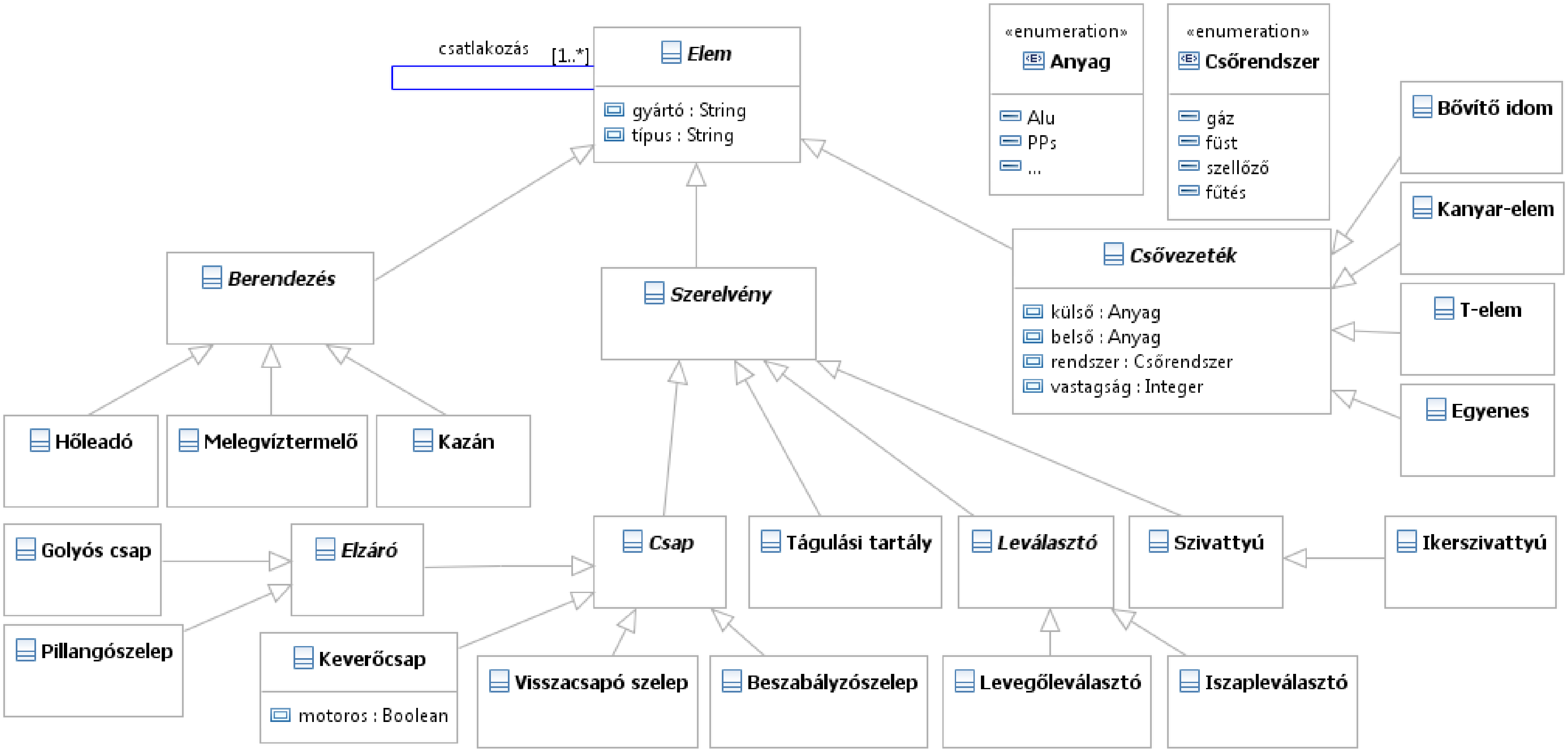




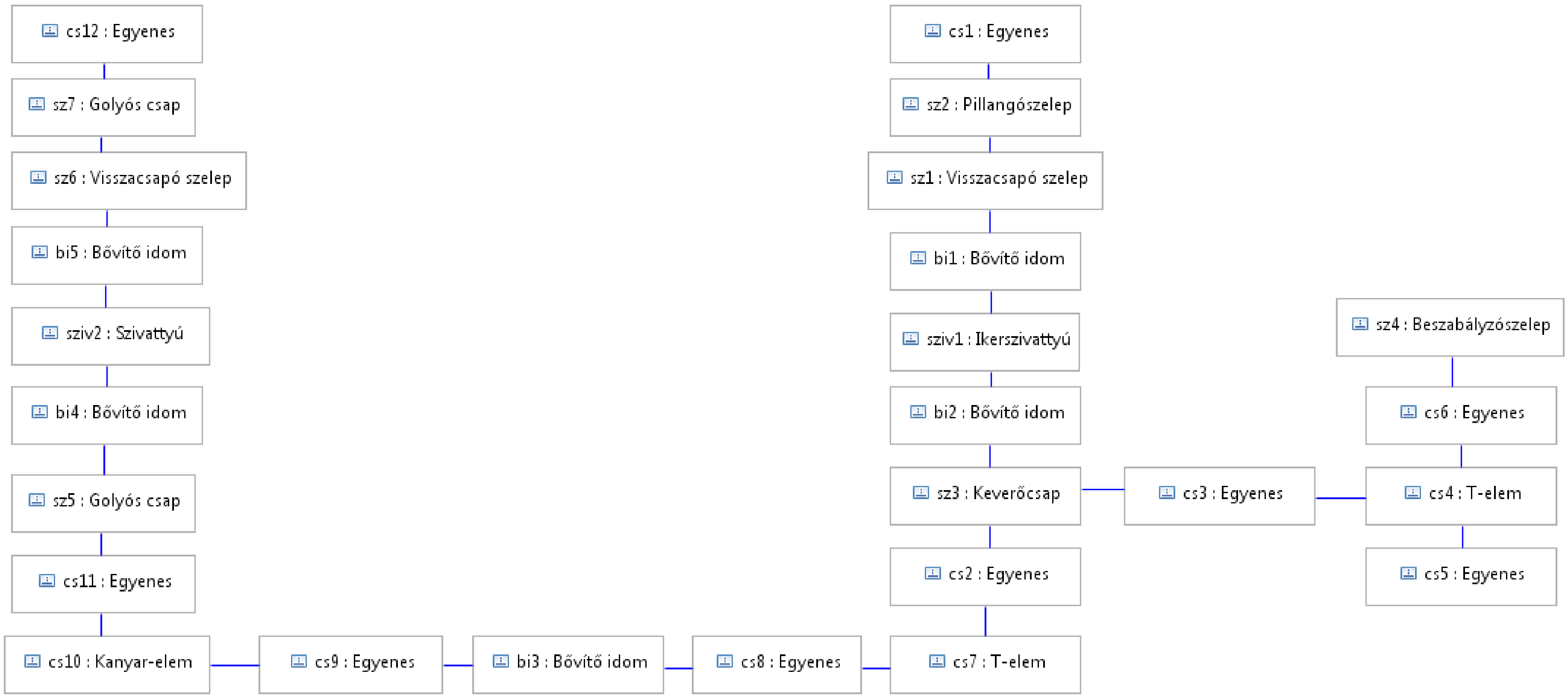


DOMAIN-SPECIFIC MODELING

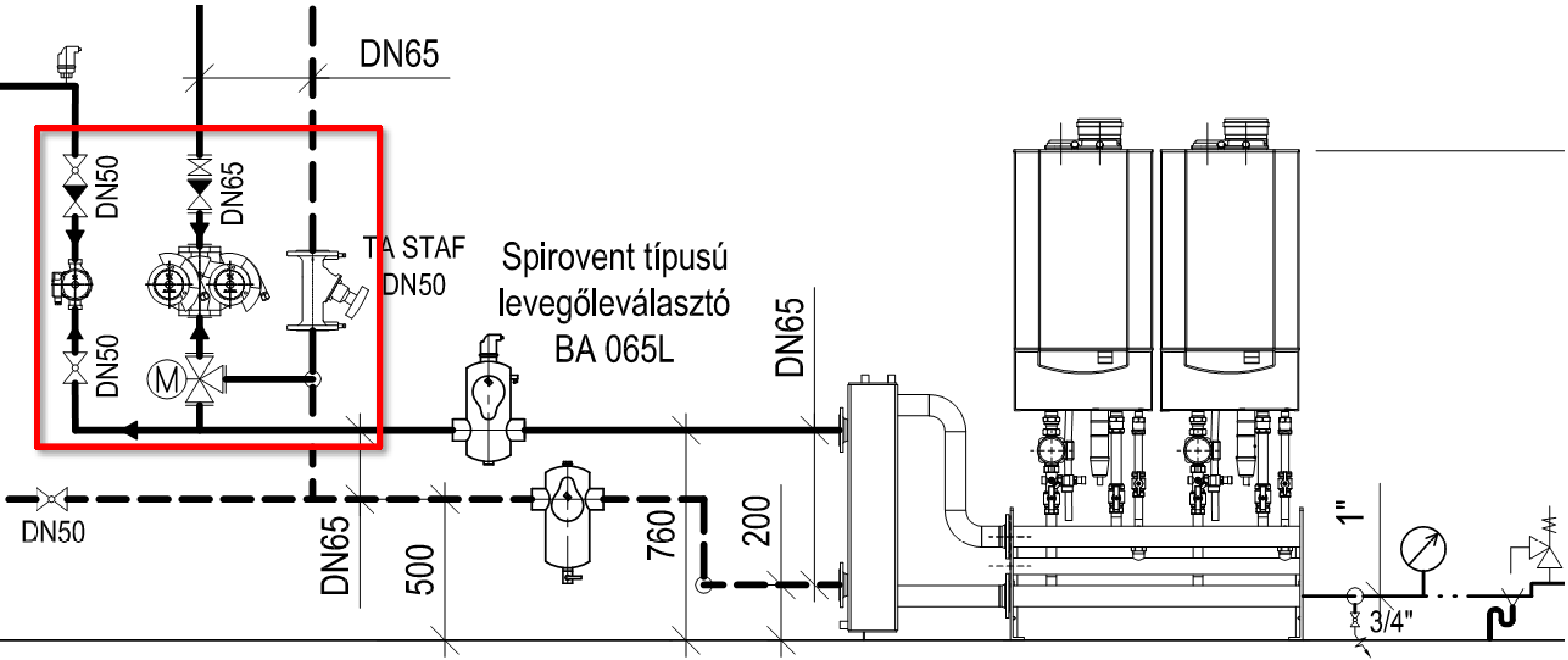
Example metamodel



Instance model, abstract syntax



Instance model, concrete syntax

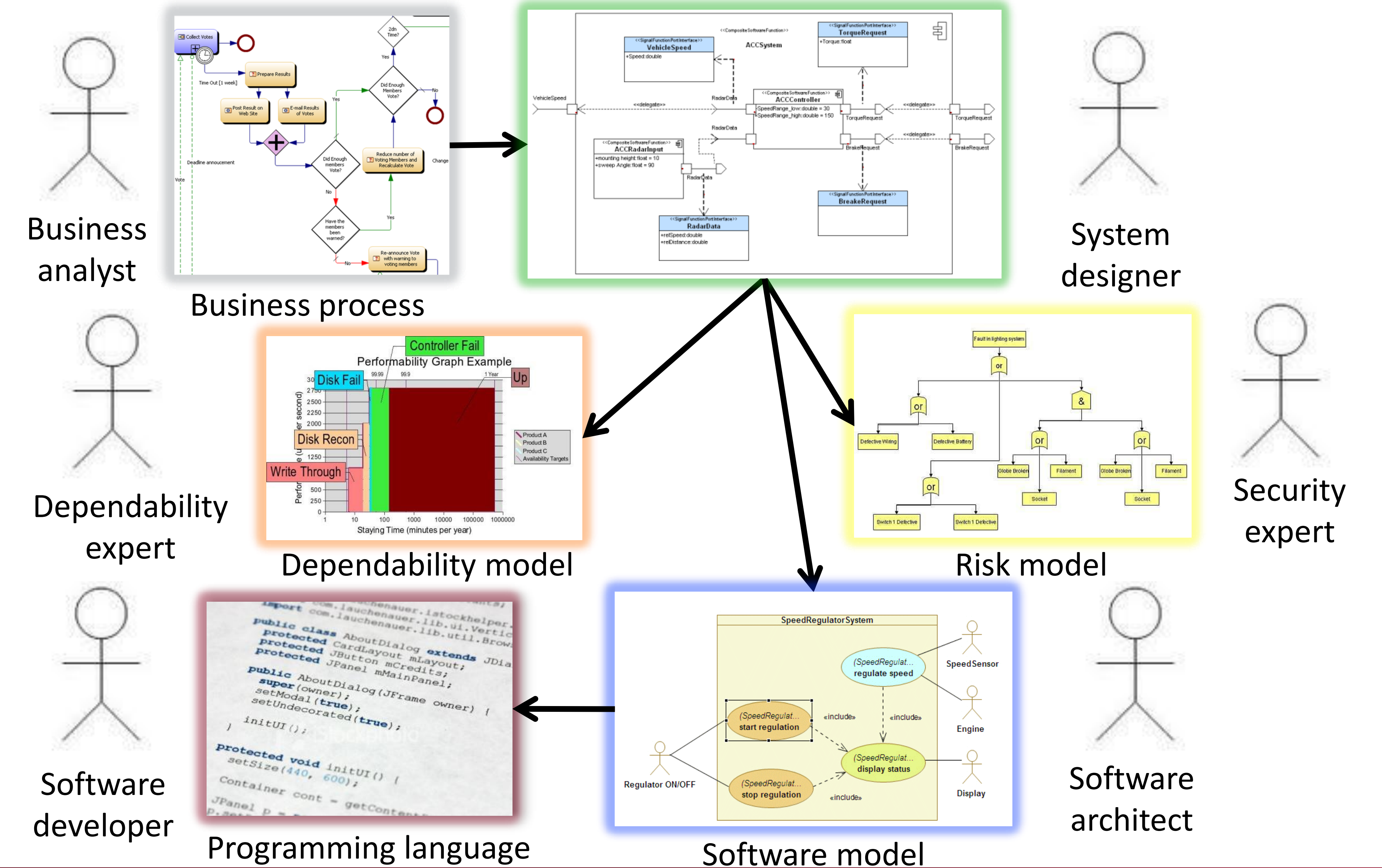


Honeywell keverőcsap DN50 Kvs 40

Spirovent típusú iszapleválasztó BE 065L

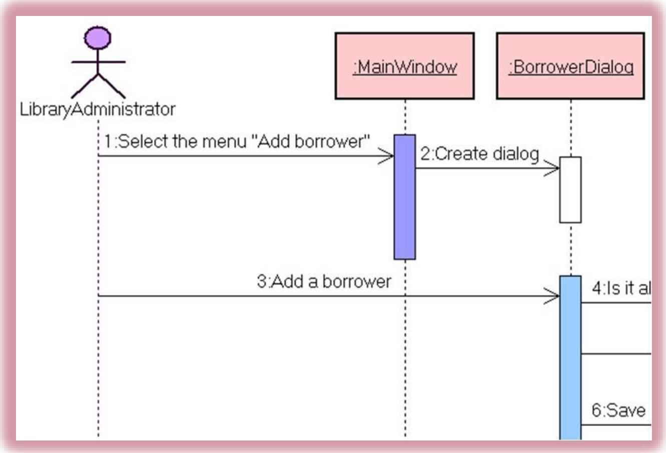
Remeha Quinta kaszkád rendszer hidraulikus váltóval

Domain specific modeling languages

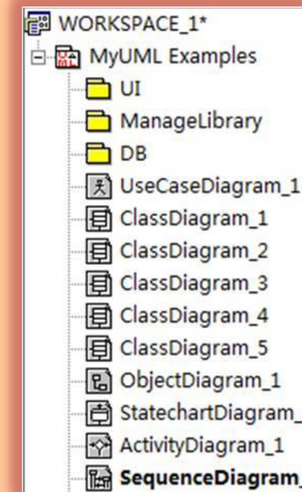


Usage example of DSMs

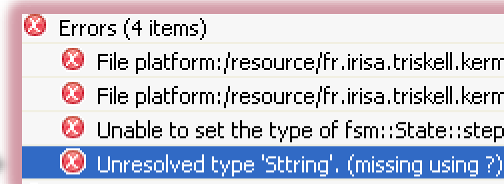
Concrete syntax



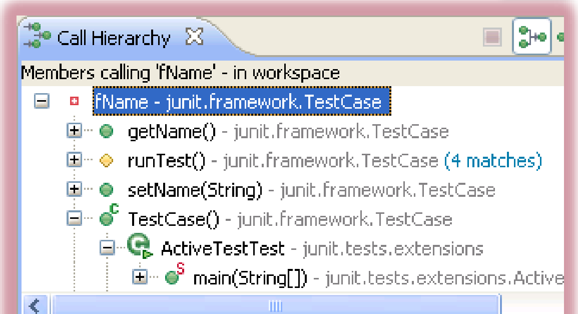
Abstract syntax



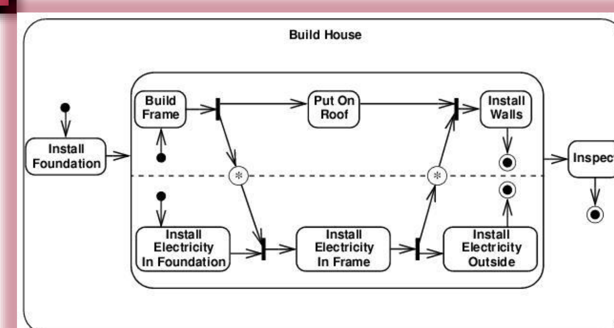
Well-formedness constraints



Behavioural semantics, simulation, refactoring

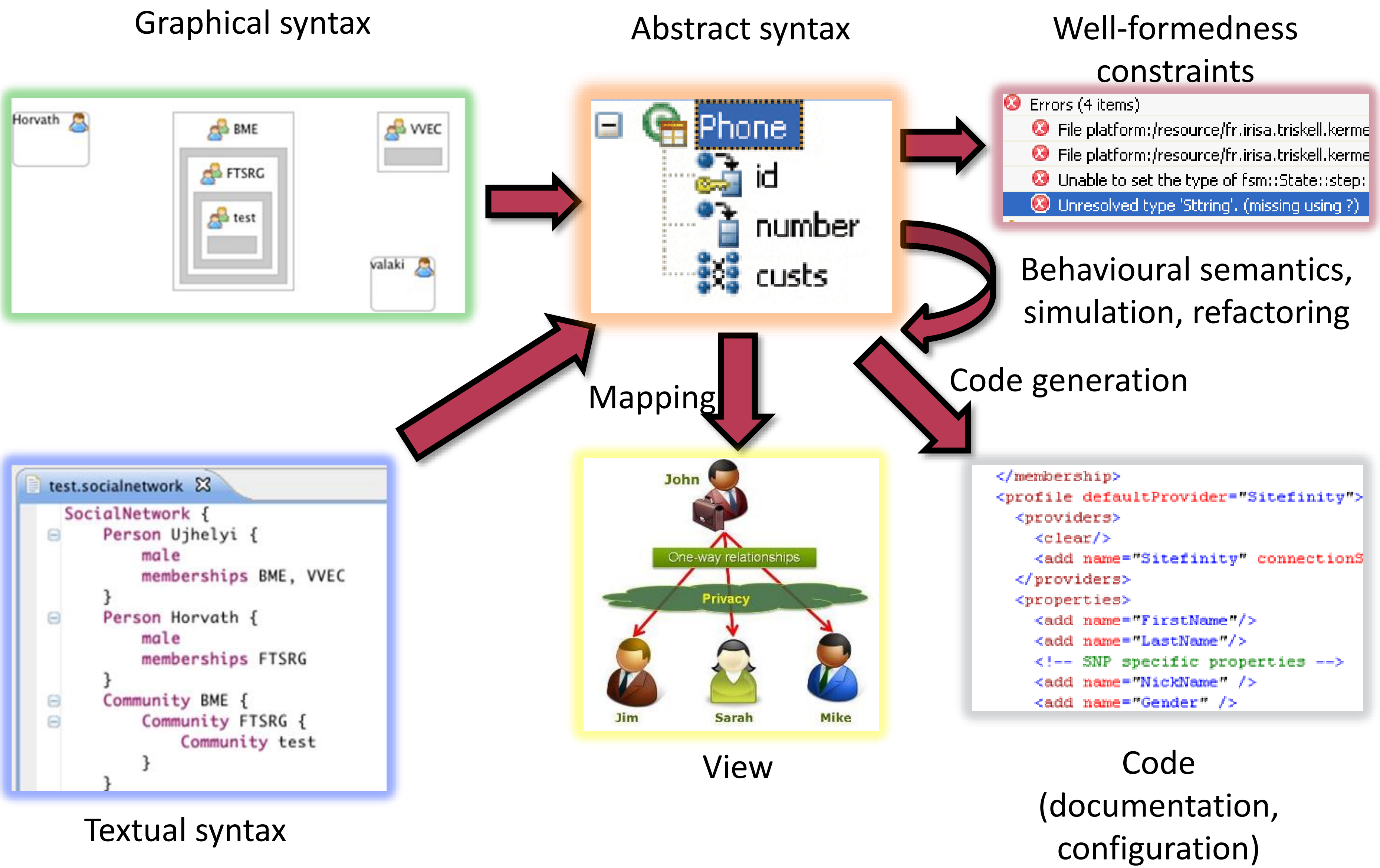


Call graph (view)



State machines (different DSM)

Structure of DSMs

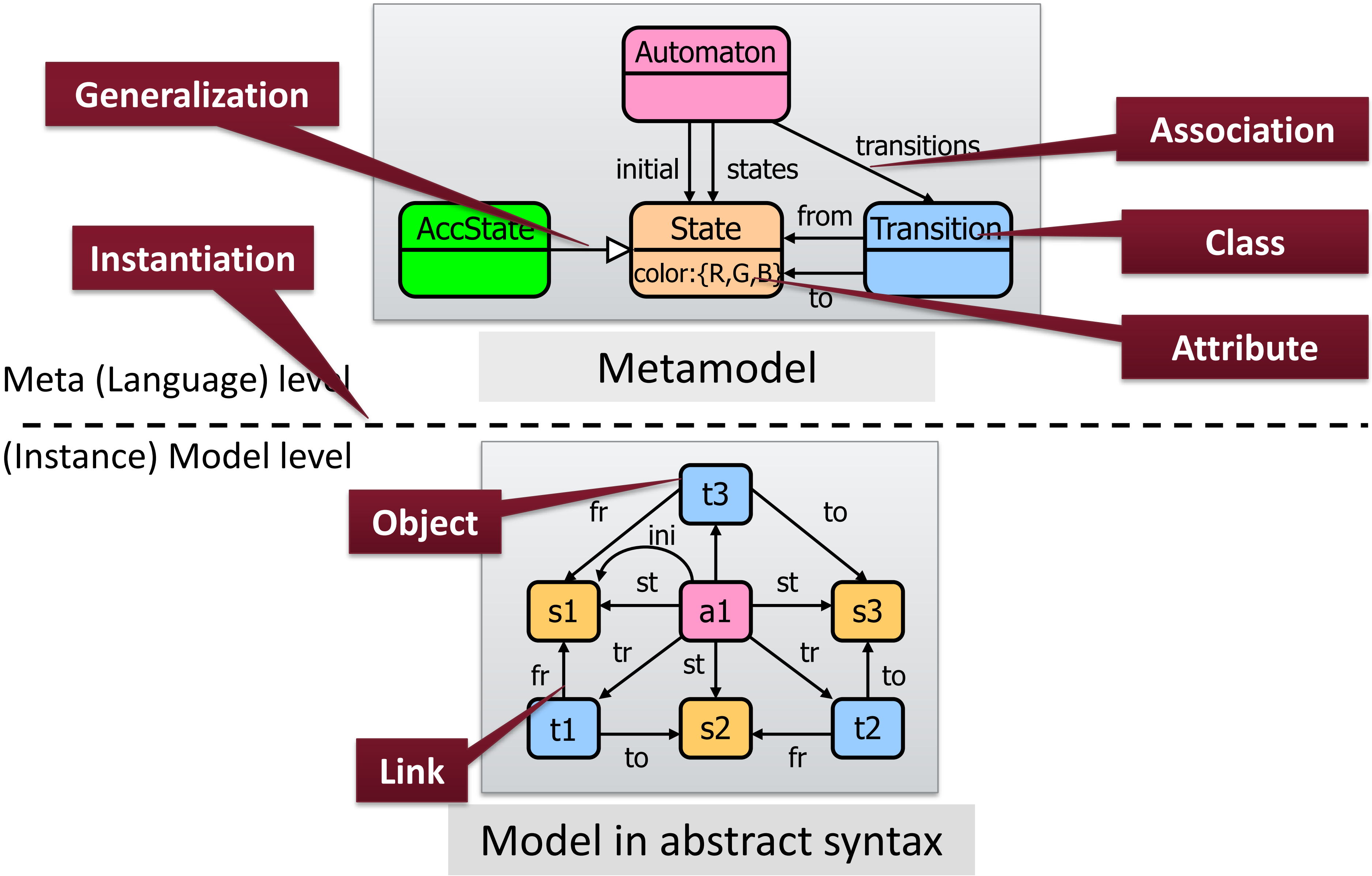


Aspects of Defining DSMLs

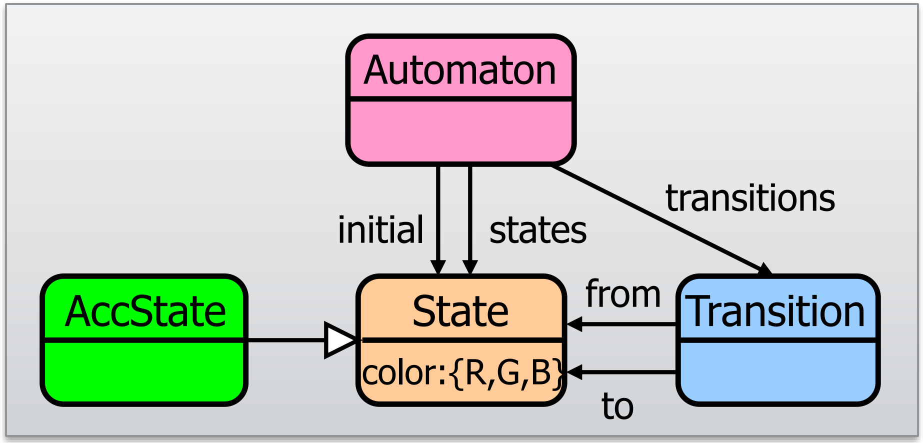
Designing modeling languages

- 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



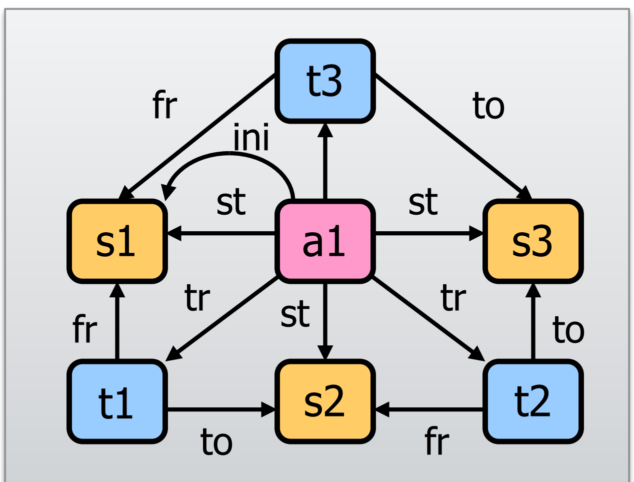
Revisiting the example



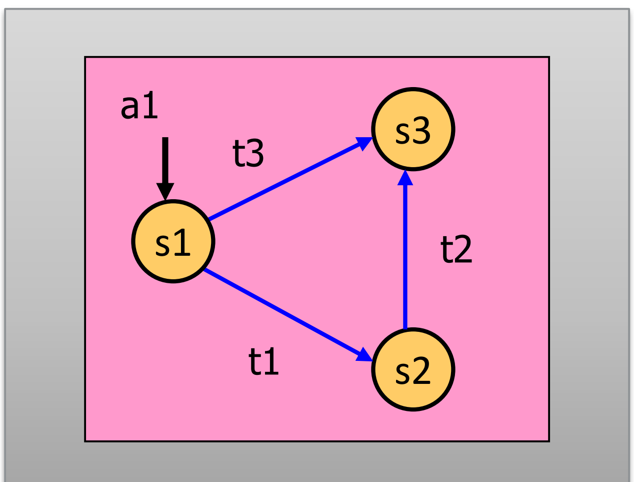
Metamodel

Meta (Language) level

Model level

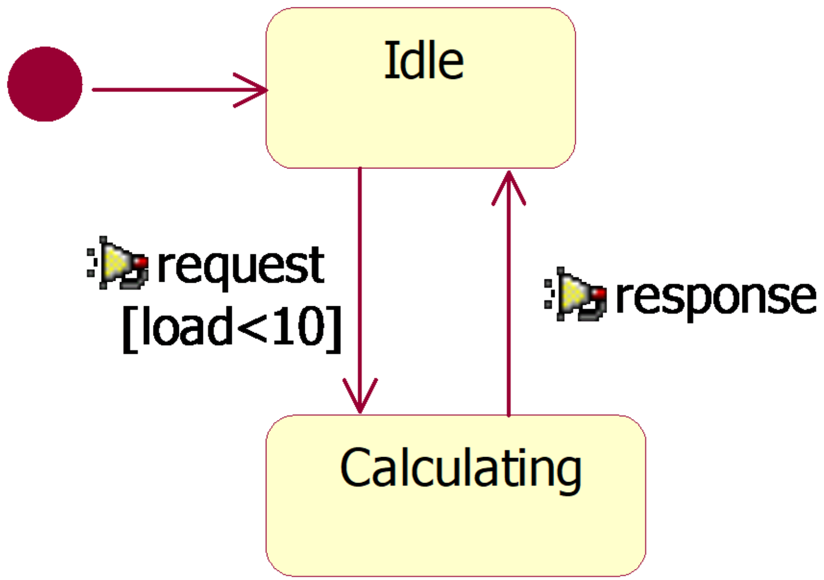


Abstract syntax



Concrete syntax

Example: Concrete Syntax



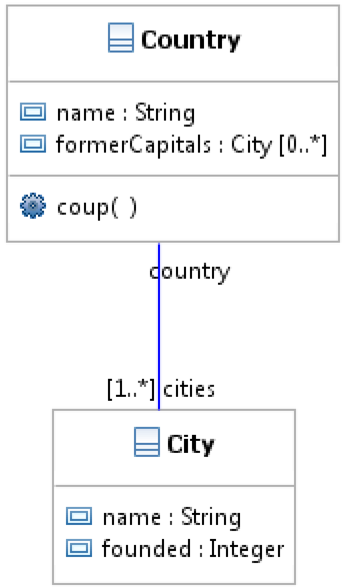

```
request() {
    if (state == "idle" &&
    this.load<10)
    state = "calculating";
}
response() {
    if (state == "calculating")
    state = "idle"
}
```

Graphical notation

Textual notation

Example: UML model

- <Package> geography
  - <Element Import> Boolean
  - <Element Import> String
  - <Element Import> UnlimitedNatural
  - <Element Import> Integer
  - <Class> Country
    - <Property> name : String
    - <Property> formerCapitals : City [0..*]
      - <Literal Unlimited Natural> *
      - <Literal Integer> 0
    - <Operation> coup ()
  - <Class> City
    - <Property> name : String
    - <Property> founded : Integer
  - <Association> A_country_cities
    - <Property> country : Country
      - <Literal Unlimited Natural> 1
      - <Literal Integer> 1
    - <Property> cities : City [1..*]
      - <Literal Unlimited Natural> *
      - <Literal Integer> 1




```
<?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="_le7b8C2v
    </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
      <lowerValue xmi:type="uml:LiteralInteger" value="1"
    </ownedEnd>
  </packagedElement>
```

Abstract Syntax

Graphical notation (Class Diagram)

Textual notation (XMI 2.1)

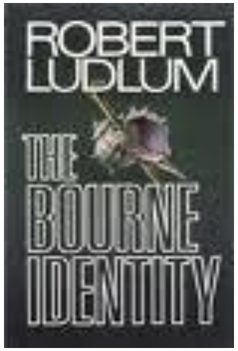
Multiplicity of Notations

- 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

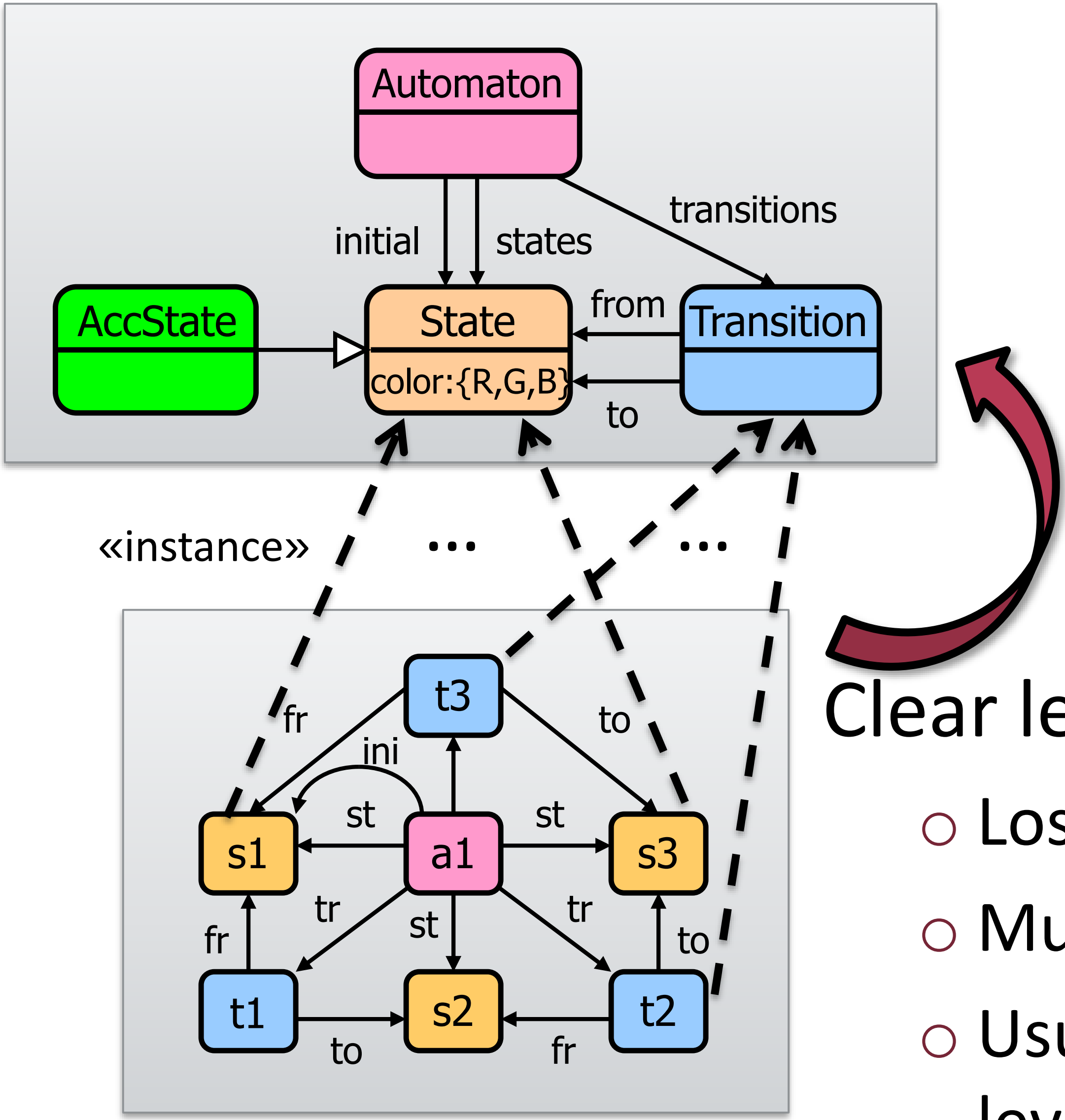
- Nodes
  - Film, Human, Novel, Psycho (film), Book, Man, Thriller, Work of Art, The Bourne Identity (novel), Genre, Robert Ludlum, Sir Alfred Hitchcock, this book here:



Demonstrated by the exercise:
- **Instantiation vs. subtyping**
- **Edge subtyping**
- **Multi-typing (intersection types)**
- **Metalevels**
- **Multi-level metamodeling**
- **Deep instantiation**

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

Metalevels



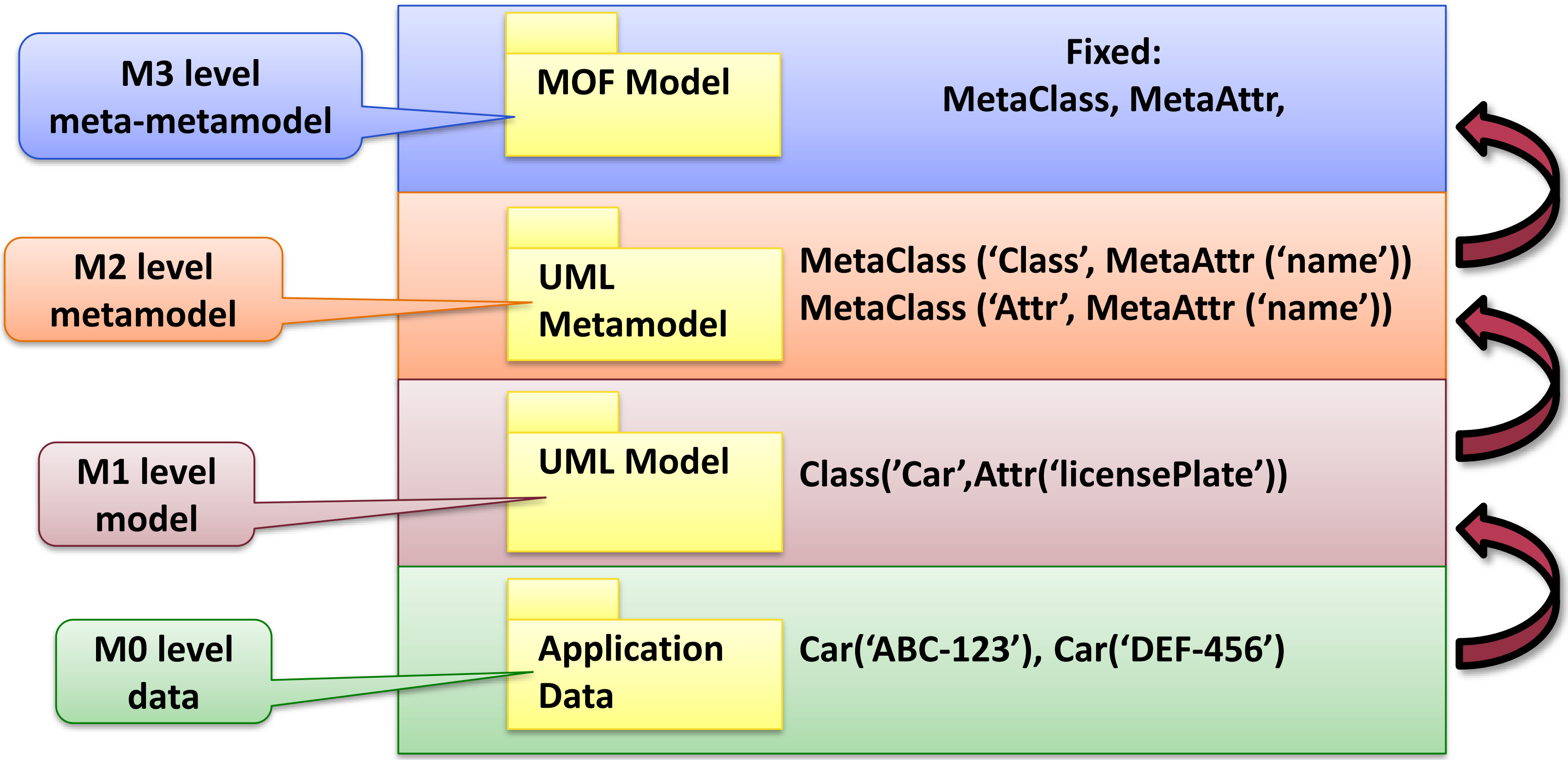
„Meta" relationship between models

Clear level separation:

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

Metalevels in MOF

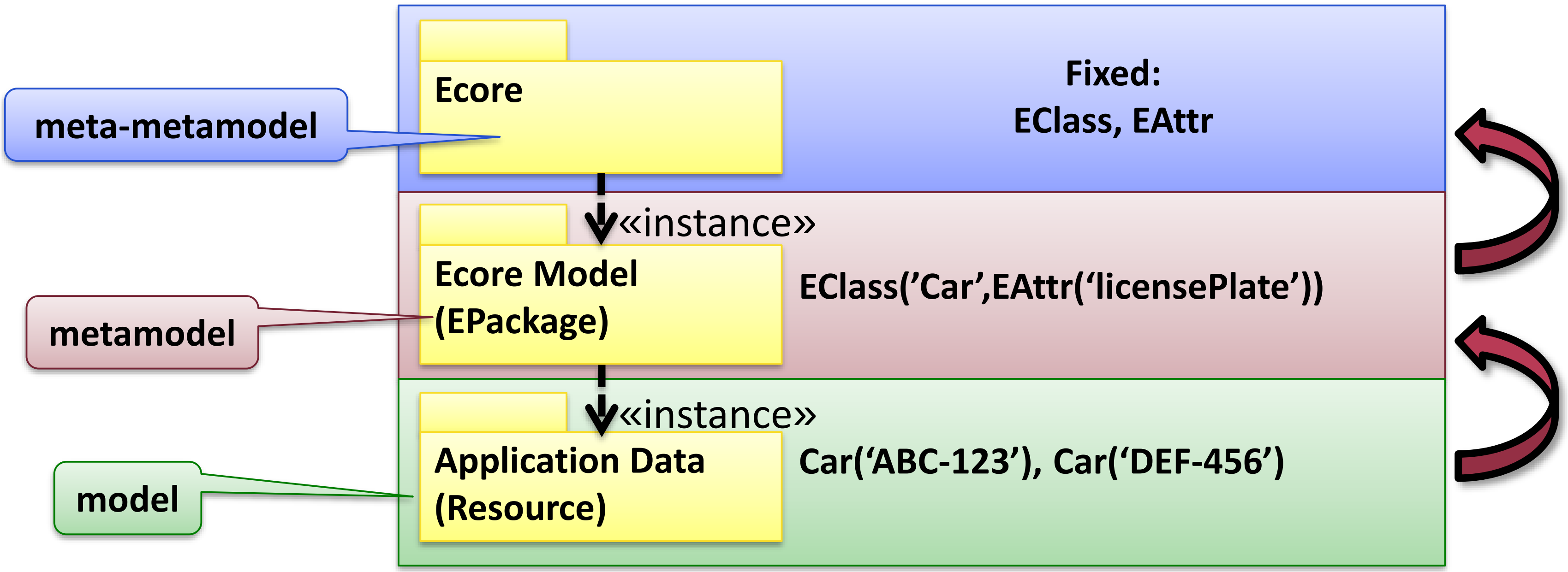
- OMG's MOF (Meta Object Facility)
  - 4-layer approach



  - Why exactly four levels?

Metalevels in other approaches

- EMF (Eclipse Modeling Framework)



- Multi-level metamodeling
  - VPM
  - Ontologies

SEMANTICS

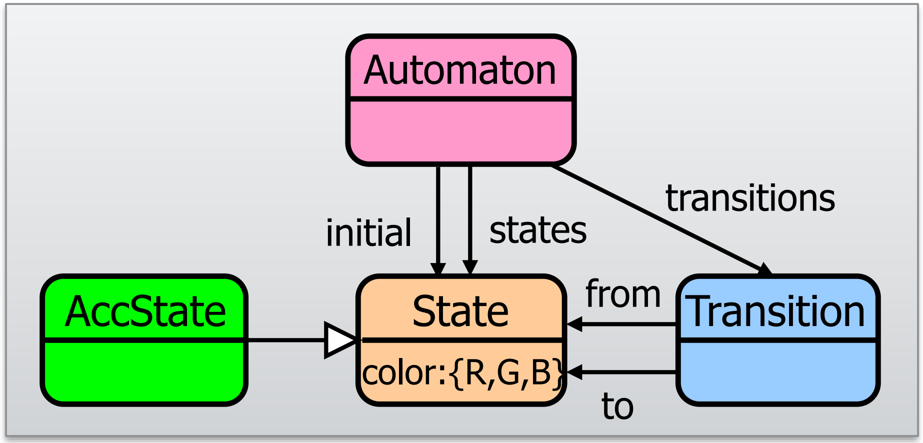
Semantics

- 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

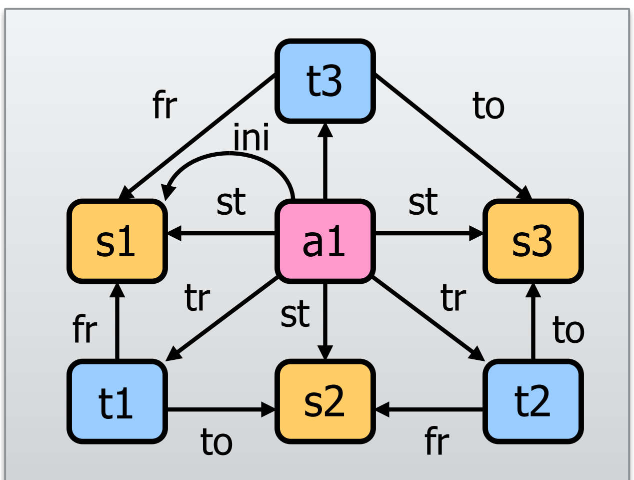
Example: Denotational semantics



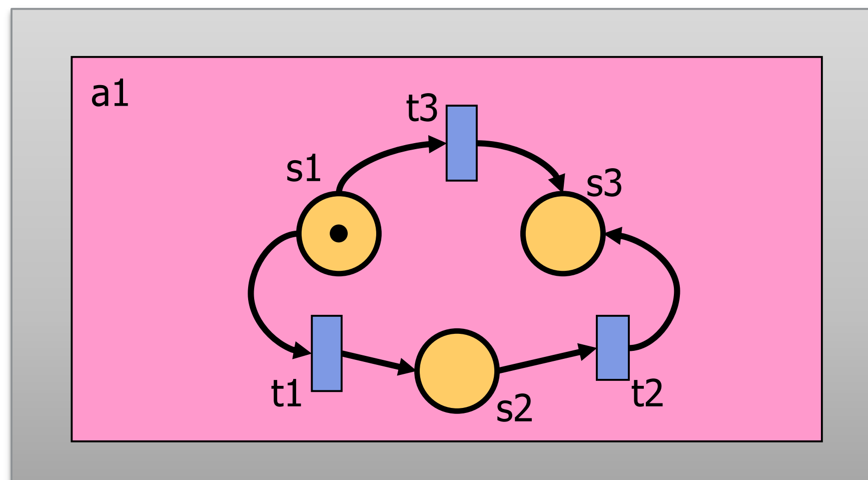
Metamodel

Meta (Language) level

Model level



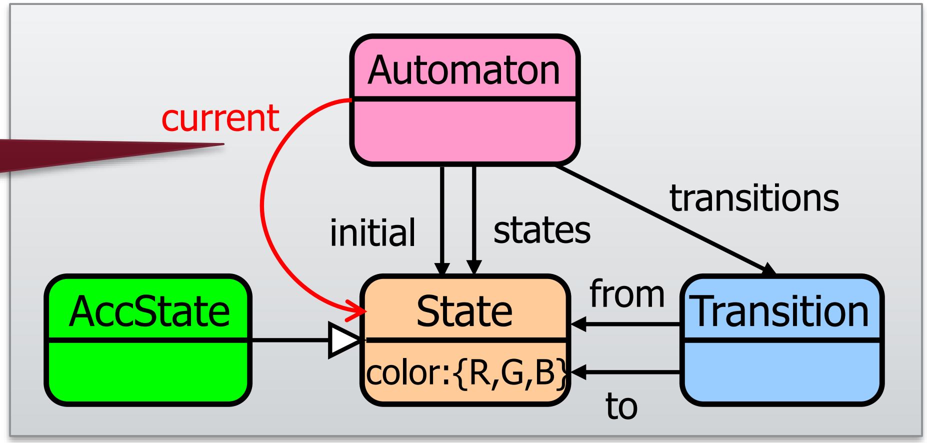
Abstract syntax



Semantic Domain

Example: Operational semantics

Dynamic feature

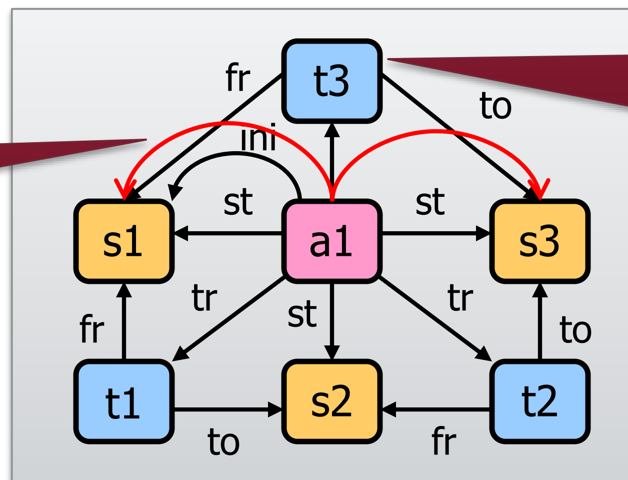


Metamodel

Meta (Language) level

(Instance) Model level

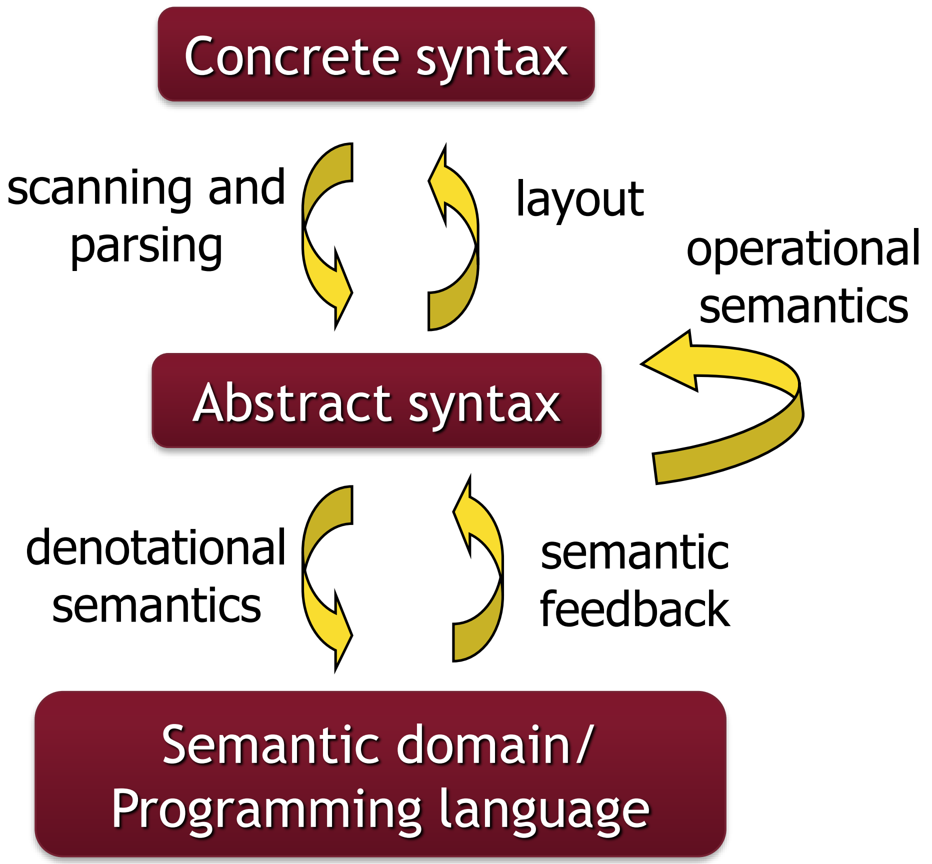
At first, ‘current ‘= ‘initial’



Possible evolution: ‘current’ is redirected along a transition

Model in abstract syntax

Relationship of models



DOMAIN-SPECIFIC MODELING LANGUAGES IN ENGINEERING PRACTICE

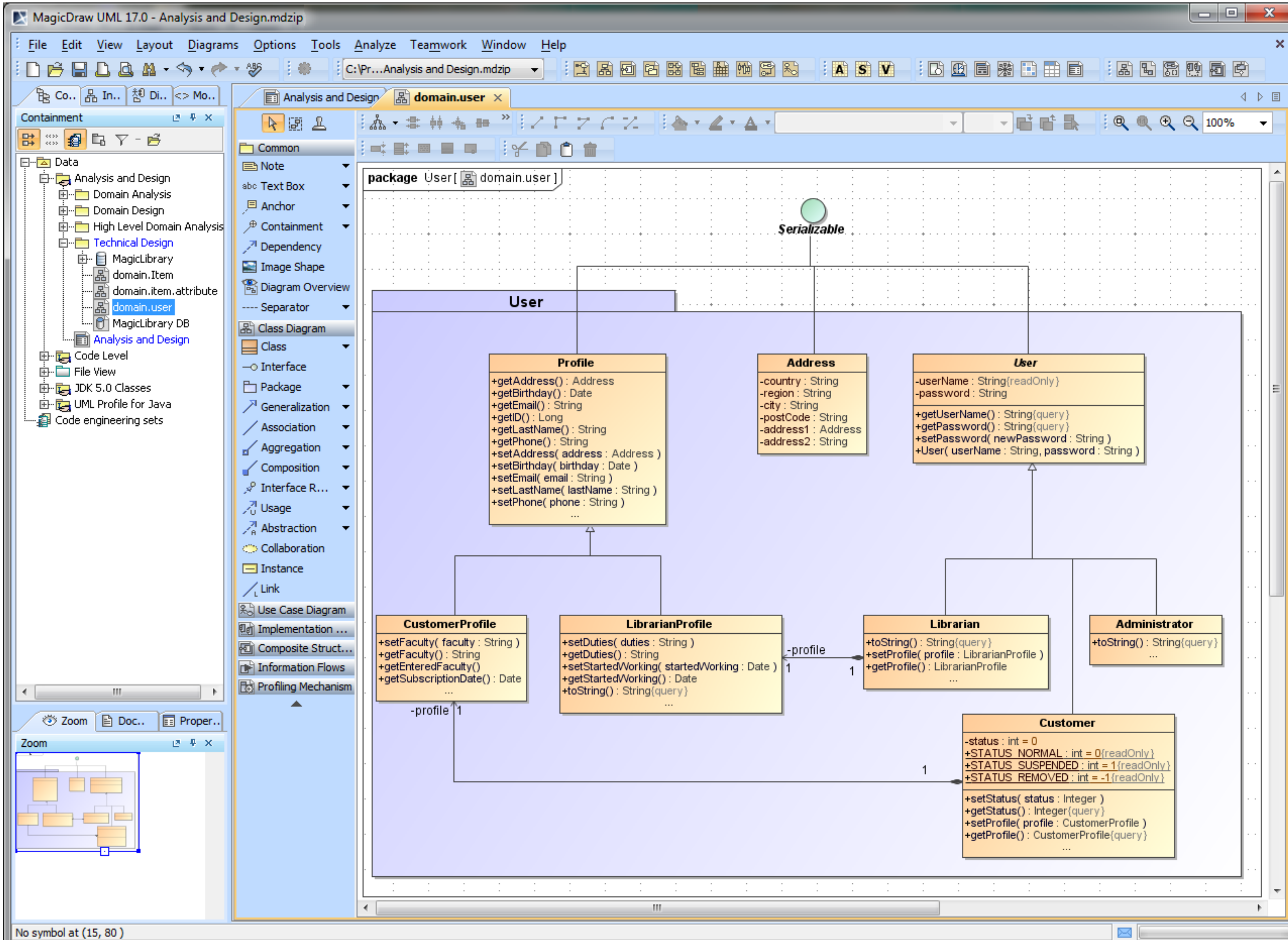
Well known DSLs

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

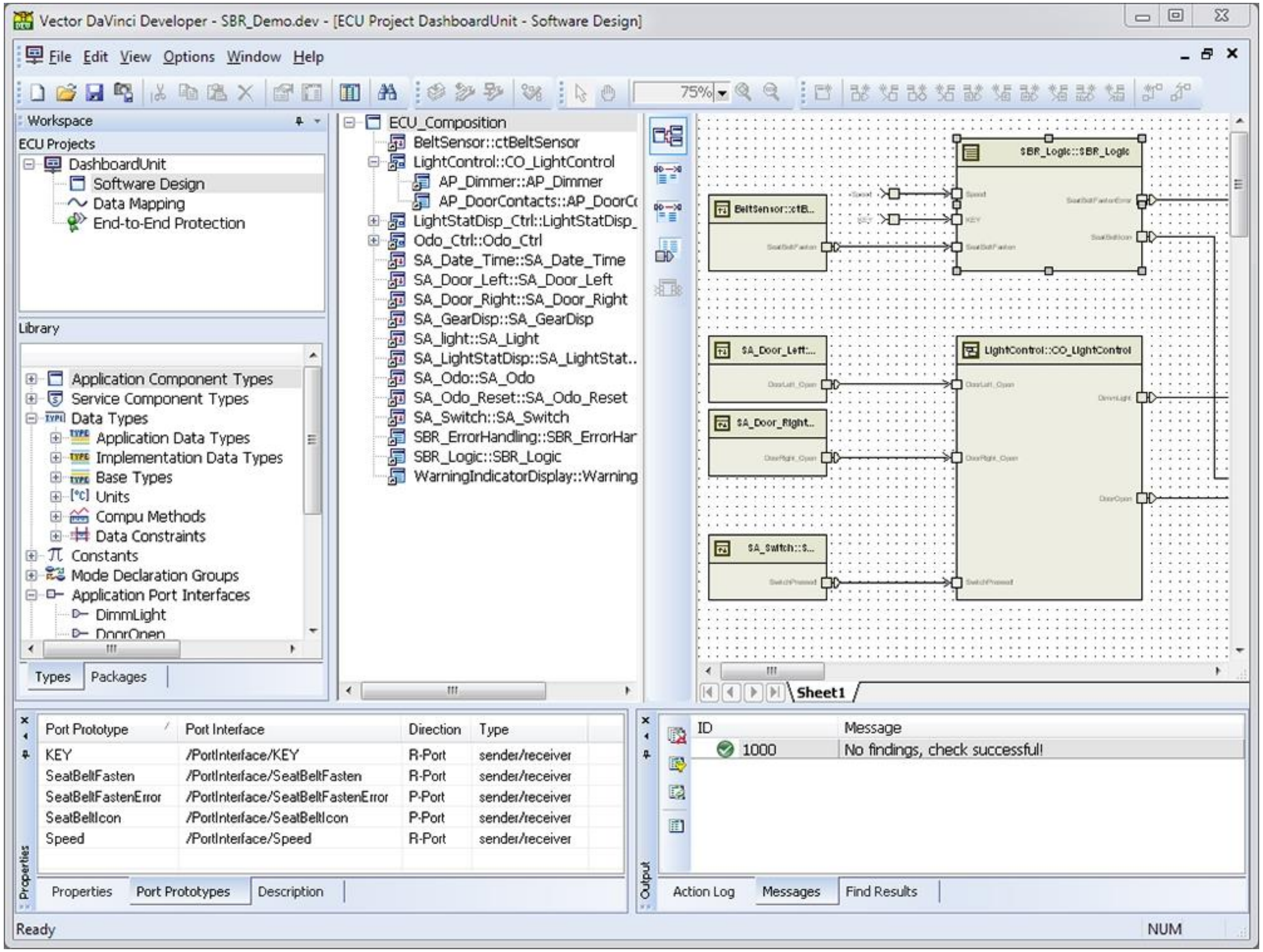
Industry standard DSMLs

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

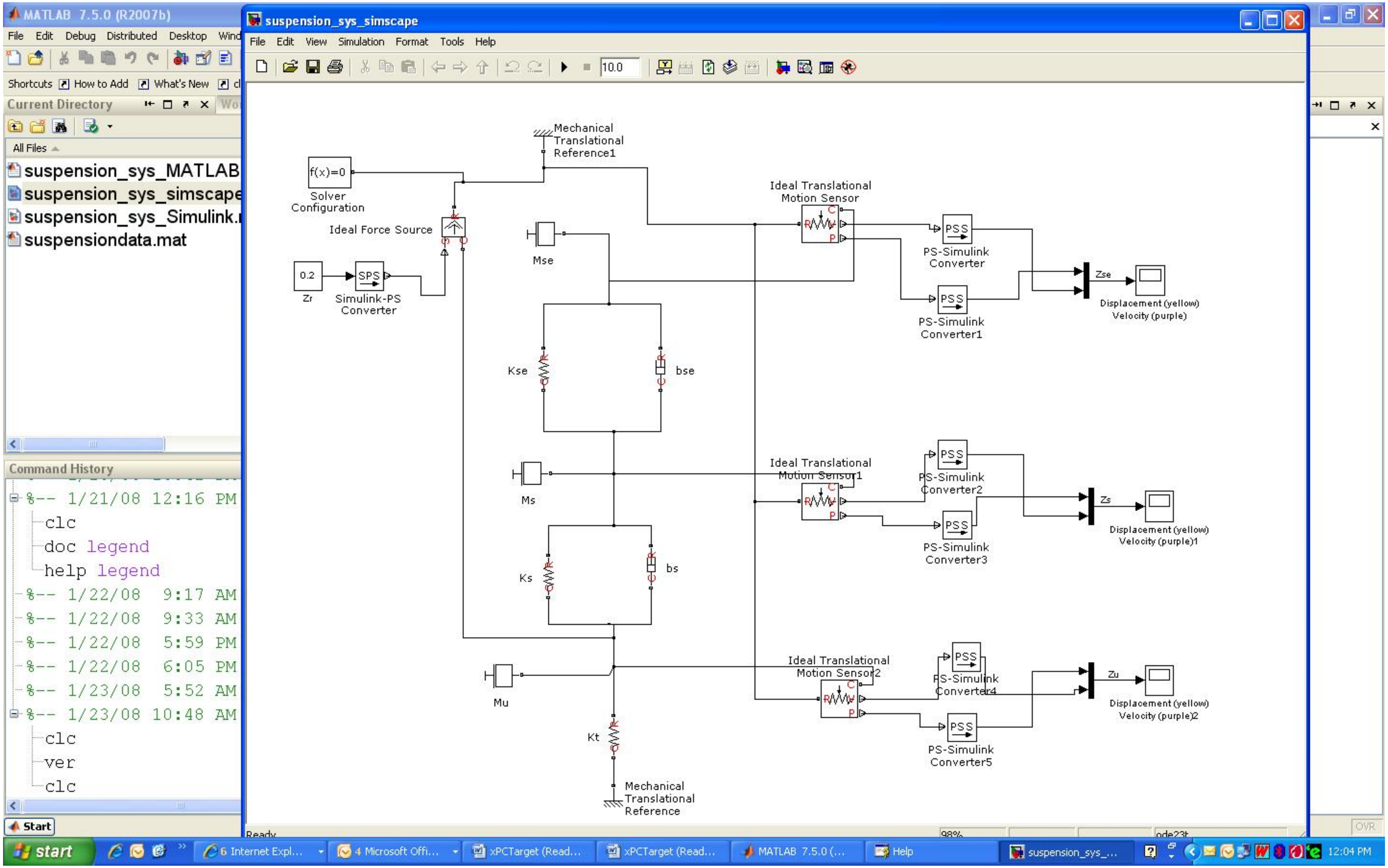
SysML: MagicDraw



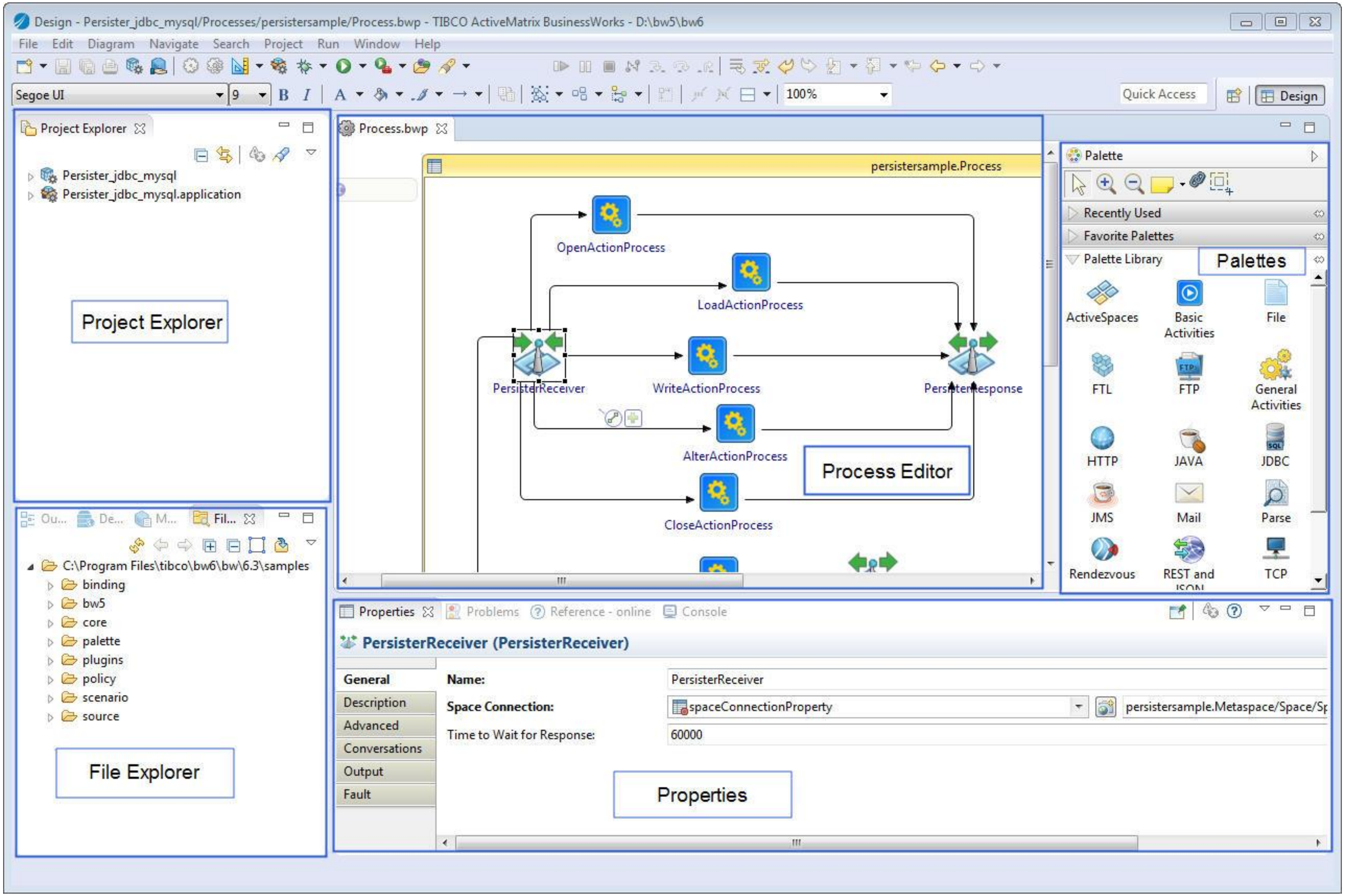
AUTOSAR: Vector DaVinci Developer



Matlab Simulink



TIBCO Business Studio



DSM Technologies

- MATLAB
- Rational Software Architect

COTS

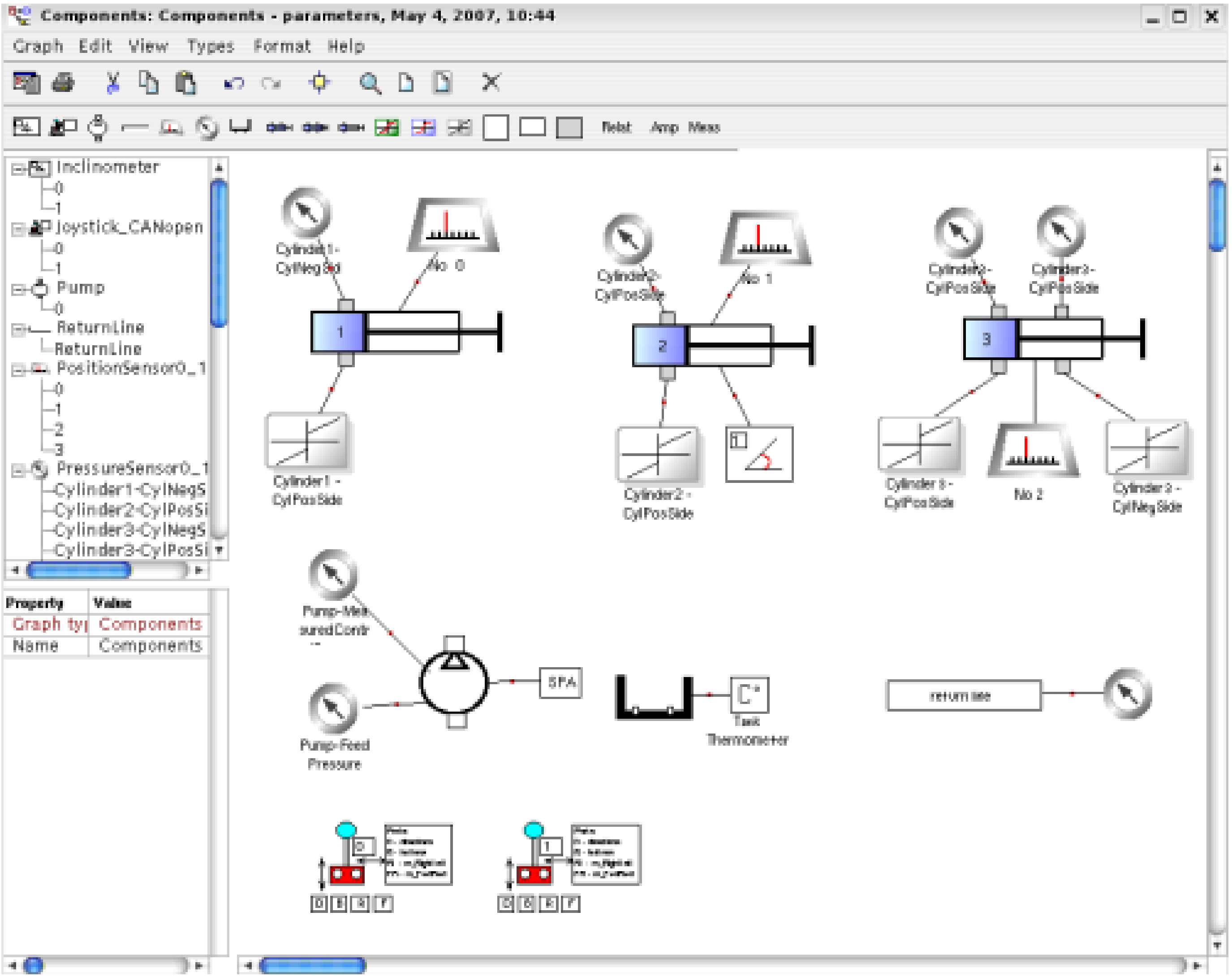
- Eclipse
  - EMF, Sirius
  - Xtext/Xcore/etc.
- Microsoft
  - DSL Tools (Visual Studio) / M / Oslo etc.
- MetaCase
  - MetaEdit+
- JetBrains MPS

Language engineering (industry)

- WebGME, Kermeta

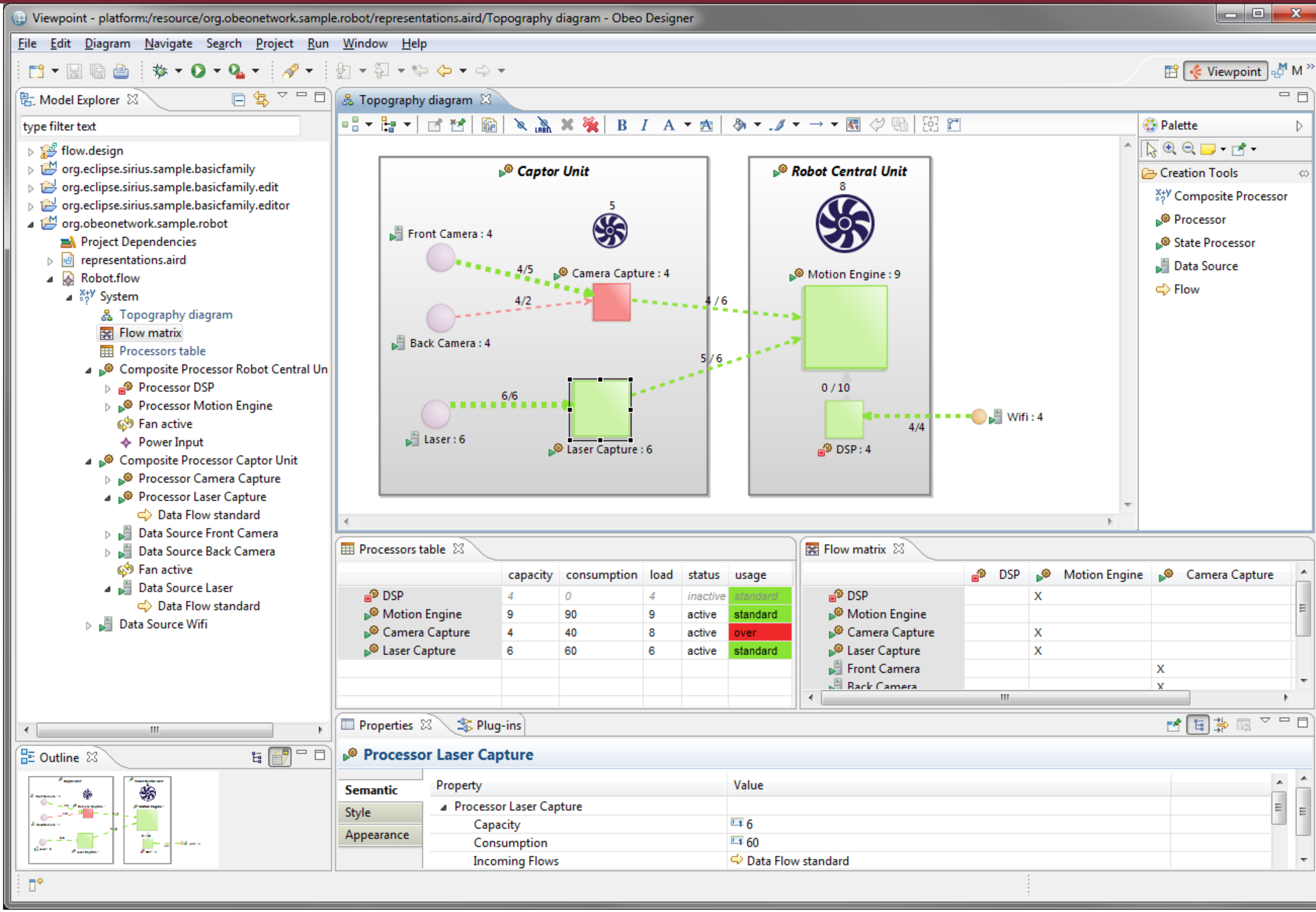
Academia

MetaEdit+

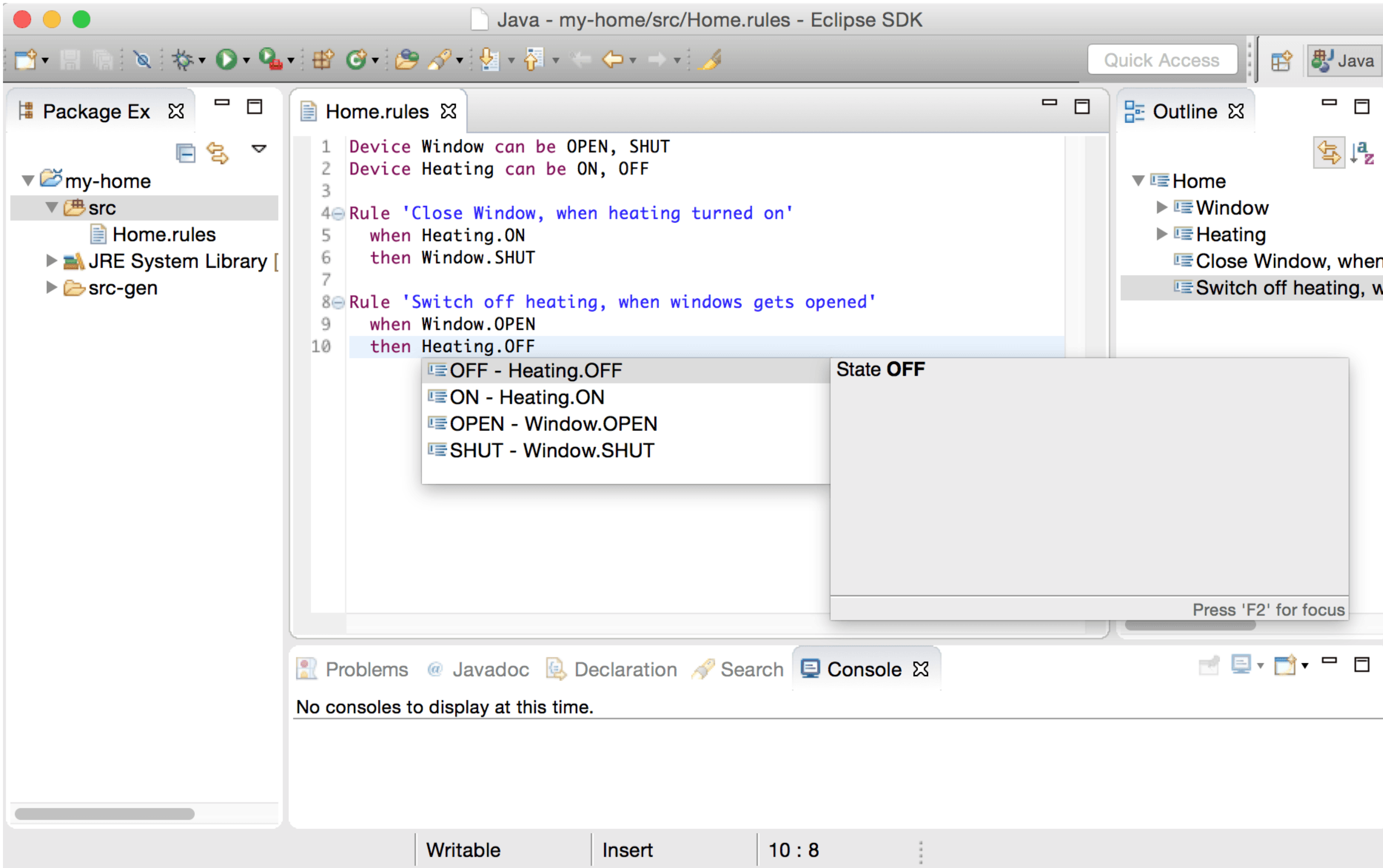


Components: Components - parameters, May 4, 2007, 10:44
Graph Edit View Types Format Help
Inclinometer
Joystick_CANopen
Pump
ReturnLine
PositionSensor0_1
PressureSensor0_1
Cylinder1-CylNegS
Cylinder2-CylPosSi
Cylinder3-CylNegS
Cylinder3-CylPosSi
Property
Value
Graph typ
Components
Name
Components
SPA
Tank
Thermometer

Eclipse Sirius

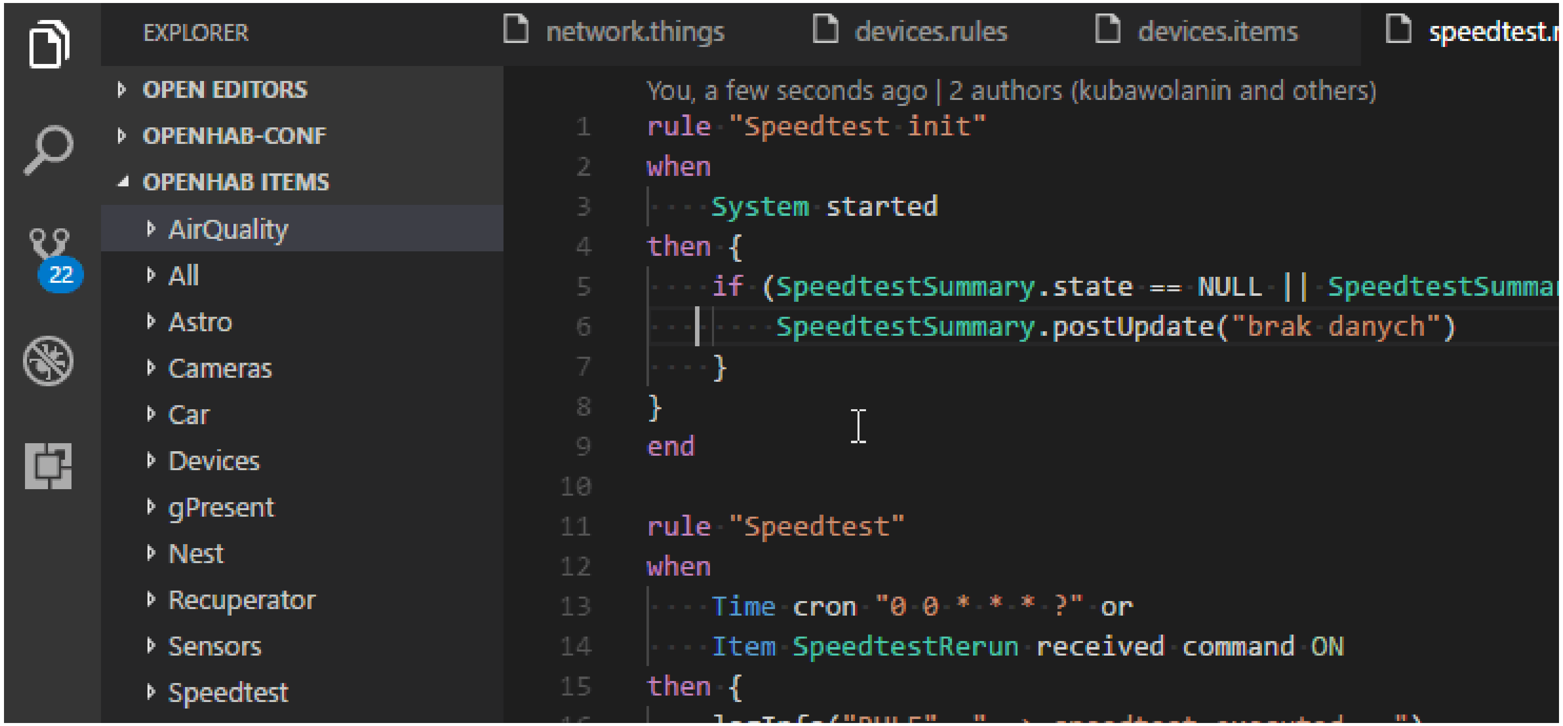


Xtext

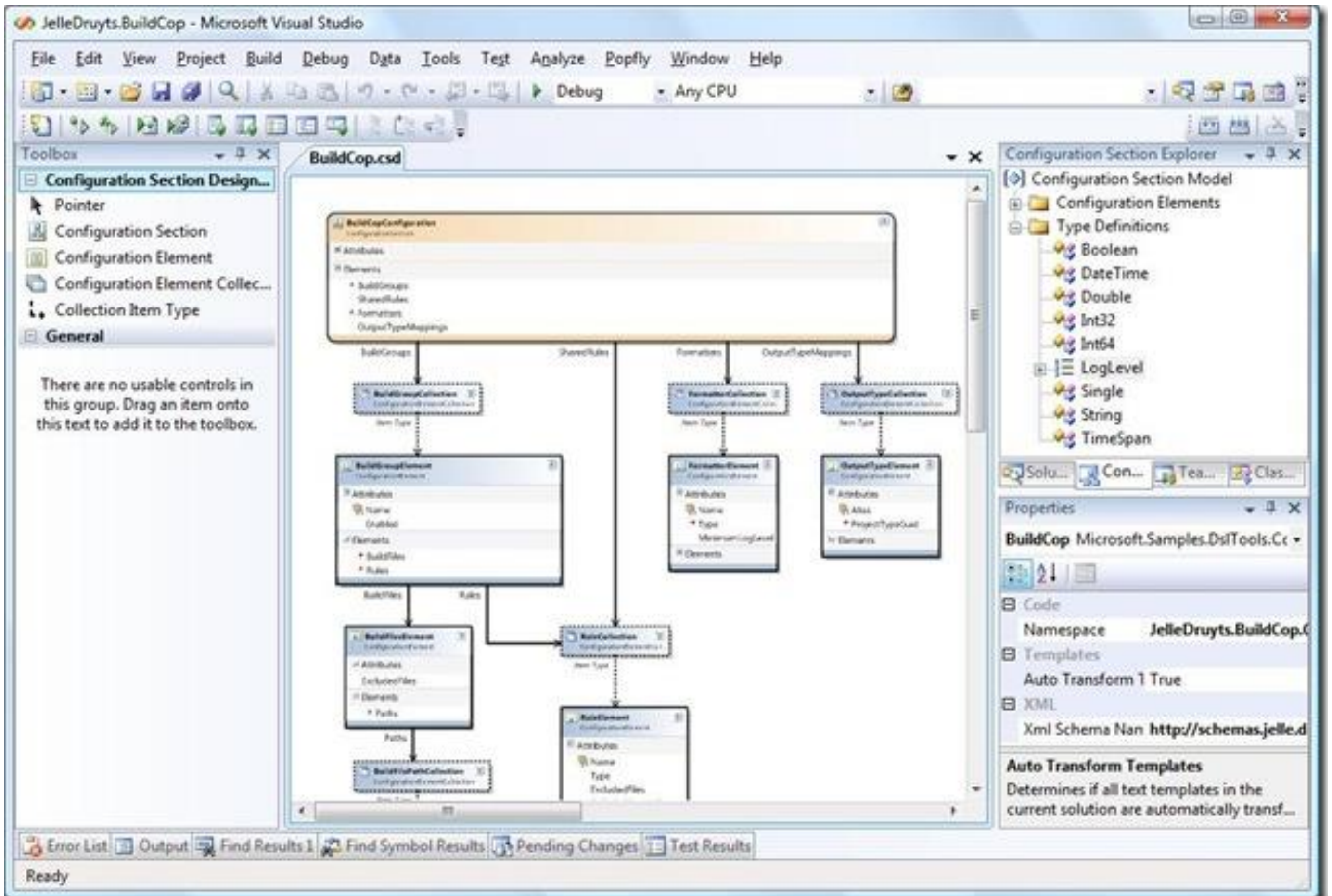


Java - my-home/src/Home.rules - Eclipse SDK
Quick Access
Java
Package Ex
my-home
src
Home.rules
JRE System Library [
src-gen
Home.rules
Device Window can be OPEN, SHUT
Device Heating can be ON, OFF
Rule 'Close Window, when heating turned on'
when Heating.ON
then Window.SHUT
Rule 'Switch off heating, when windows gets opened'
when Window.OPEN
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

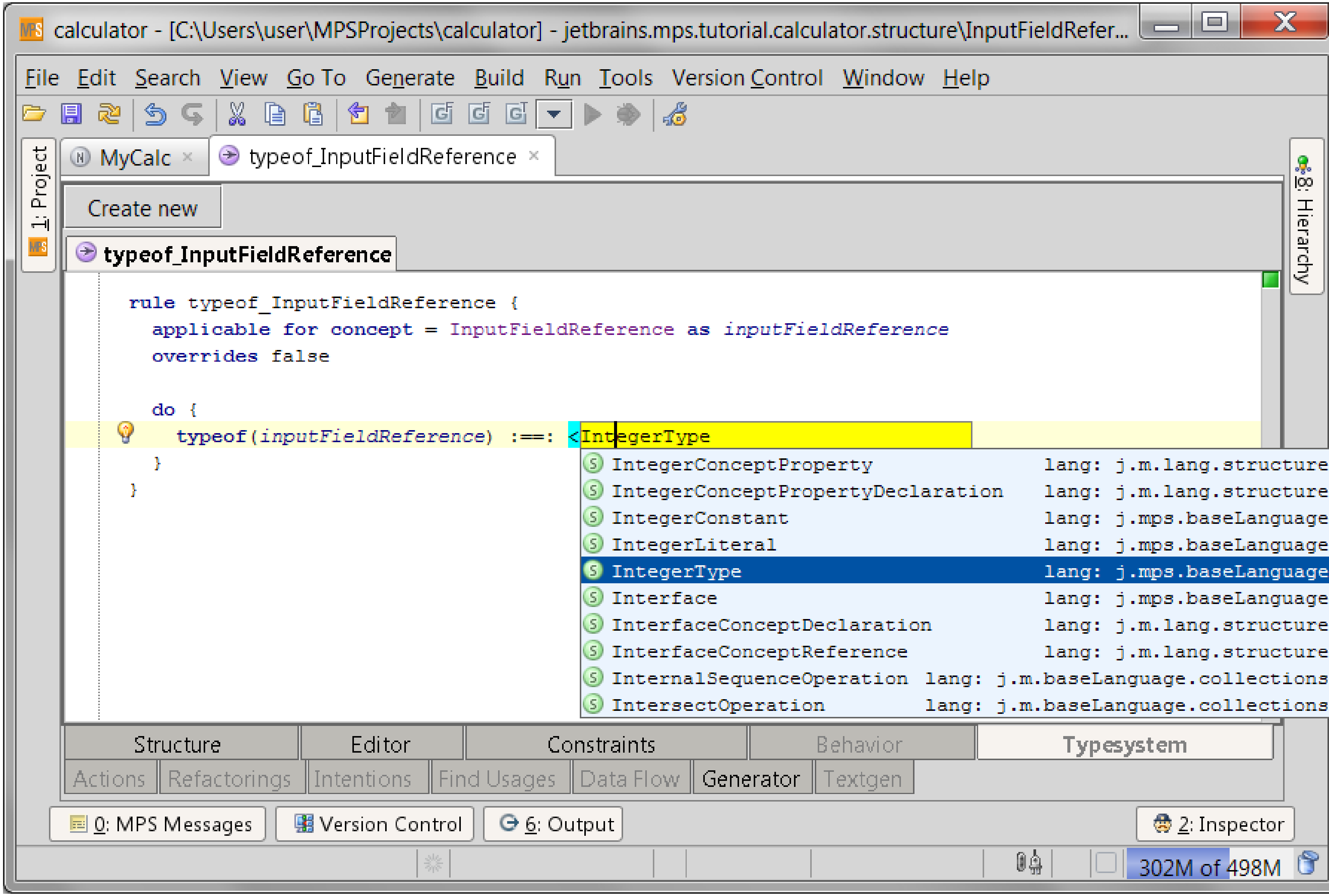
Xtext


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

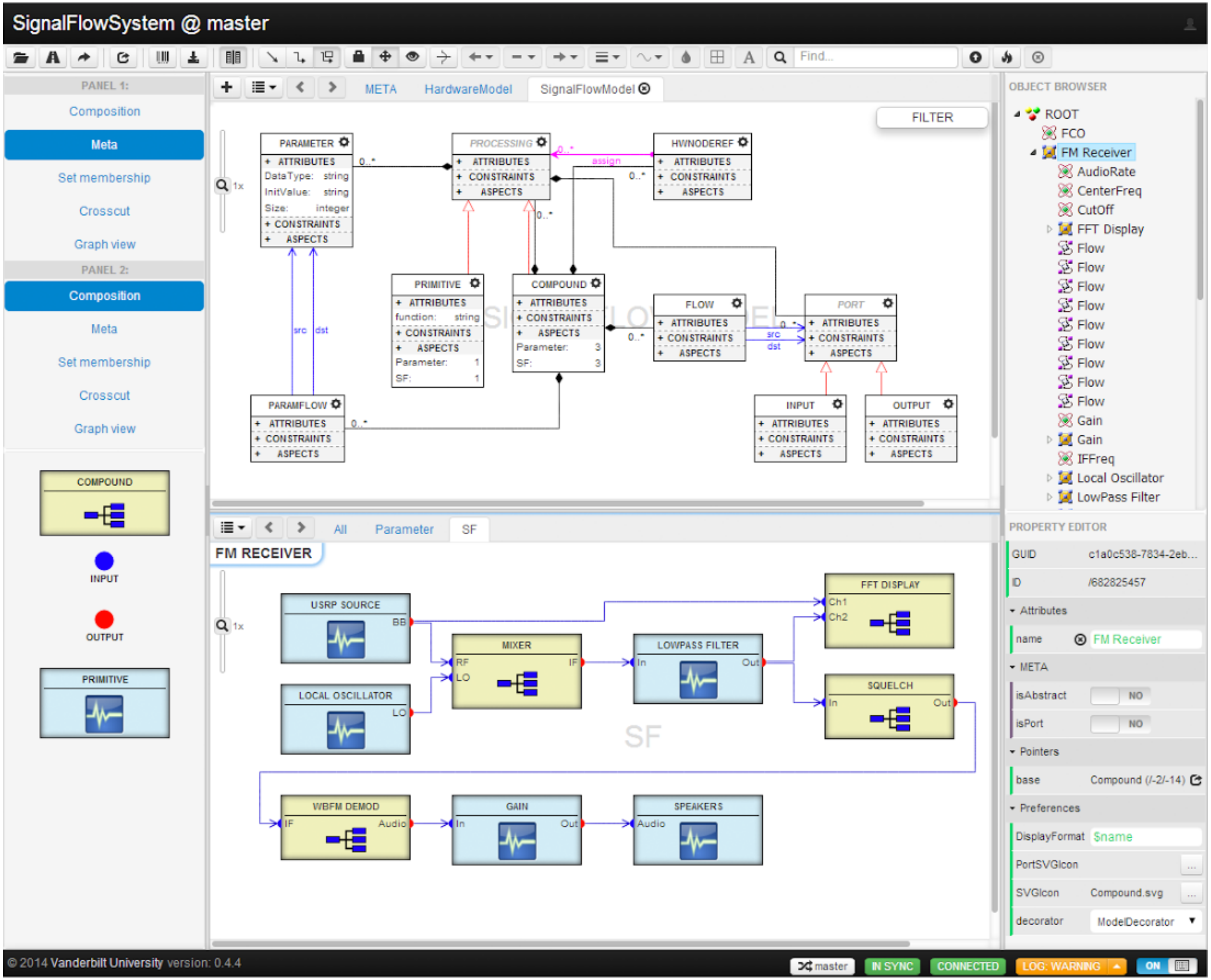
Microsoft DSL Tools



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
1: Project
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
8: Hierarchy
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

WebGME


SignalFlowSystem @ master
PANEL 1:
Composition
Meta
Set membership
Crosscut
Graph view
PANEL 2:
Composition
Meta
Set membership
Crosscut
Graph view
COMPOUND
INPUT
OUTPUT
PRIMITIVE
META
HardwareModel
SignalFlowModel
FILTER
PARAMETER
PROCESSING
HWNODEREF
PRIMITIVE
COMPOUND
FLOW
PORT
PARAMFLOW
INPUT
OUTPUT
ATTRIBUTES
CONSTRAINTS
ASPECTS
assign
src
dst
All
Parameter
SF
FM RECEIVER
USRP SOURCE
MIXER
LOWPASS FILTER
FFT DISPLAY
LOCAL OSCILLATOR
SQUELCH
WBFM DEMOD
GAIN
SPEAKERS
OBJECT BROWSER
ROOT
FCO
FM Receiver
AudioRate
CenterFreq
CutOff
FFT Display
Flow
Gain
IFFreq
Local Oscillator
LowPass Filter
PROPERTY EDITOR
GUID
ID
Attributes
name
FM Receiver
META
isAbstract
isPort
Pointers
base
Compound (/-2/-14)
Preferences
DisplayFormat
$name
PortSVGIcon
SVGIcon
Compound.svg
decorator
ModelDecorator
© 2014 Vanderbilt University version: 0.4.4
master
IN SYNC
CONNECTED
LOG: WARNING
ON

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

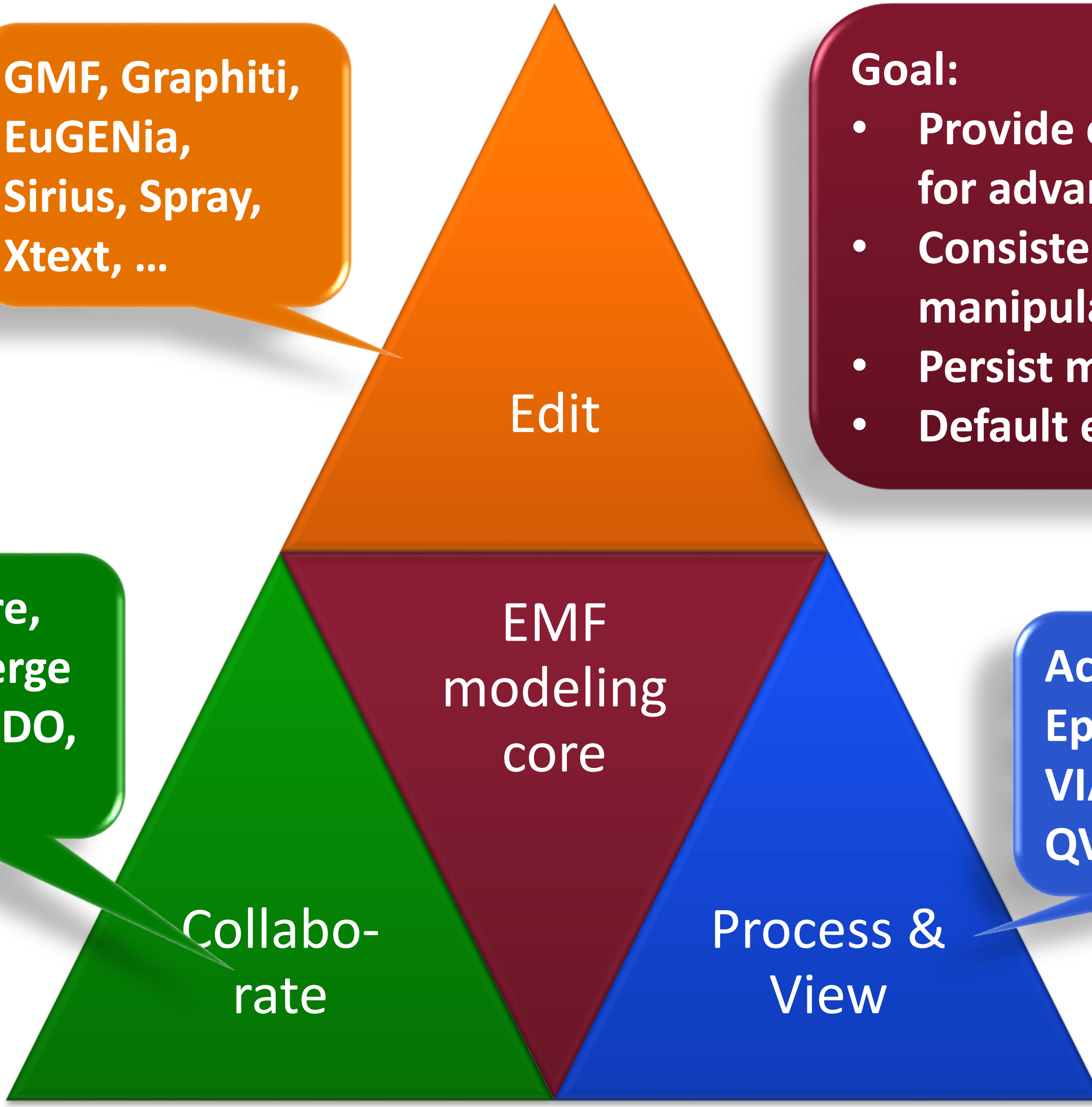
What does EMF provide?

- 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

GMF, Graphiti, EuGENia, Sirius, Spray, Xtext, …

Goal:
- **Provide common base for advanced DSL tools**
- **Consistent model manipulation**
- **Persist models**
- **Default editor**

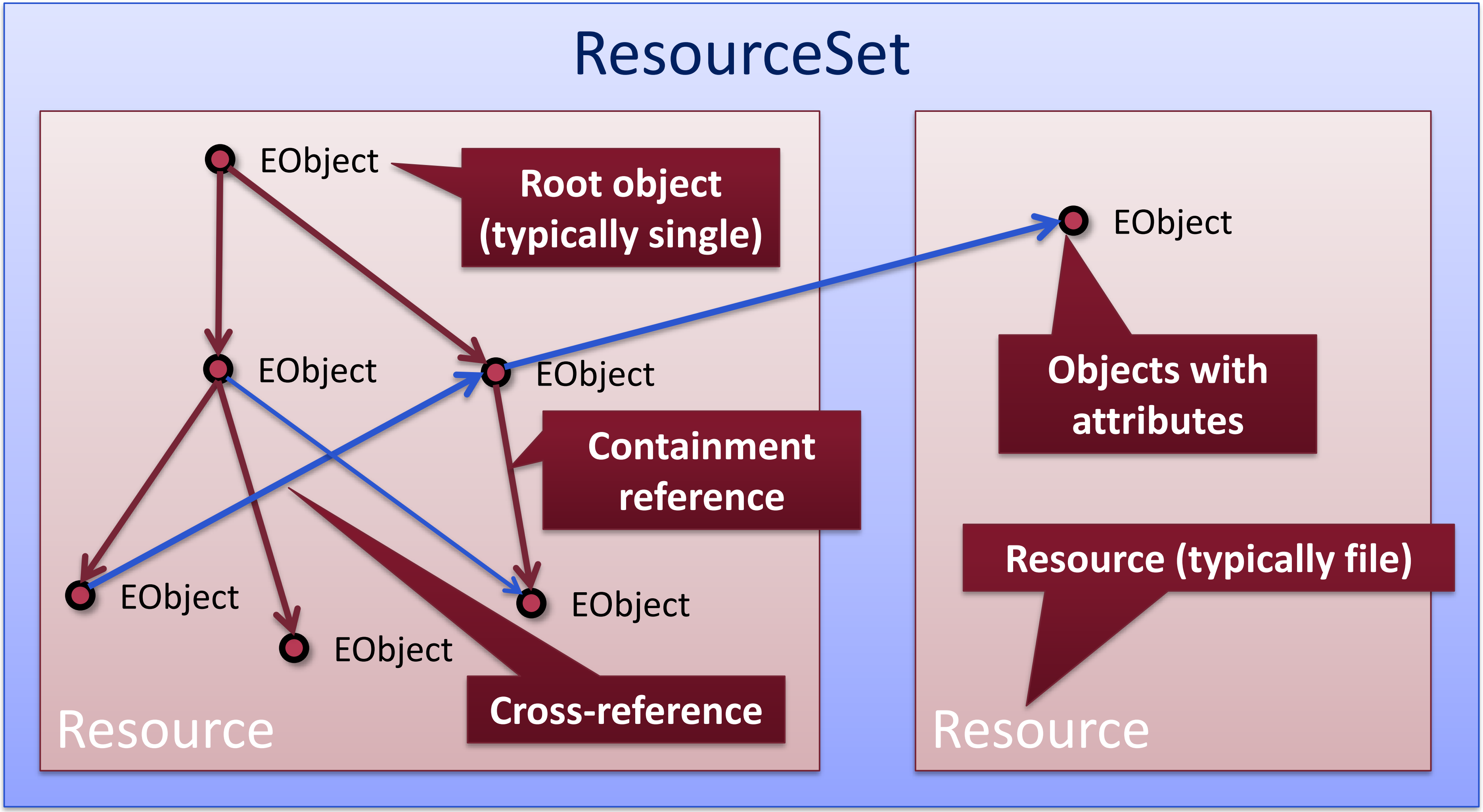


EMF Compare, EMF Diff/Merge EMF Store, CDO, …

Acceleo, ATL, Epsilon, VIATRA, QVT, Xtend, …

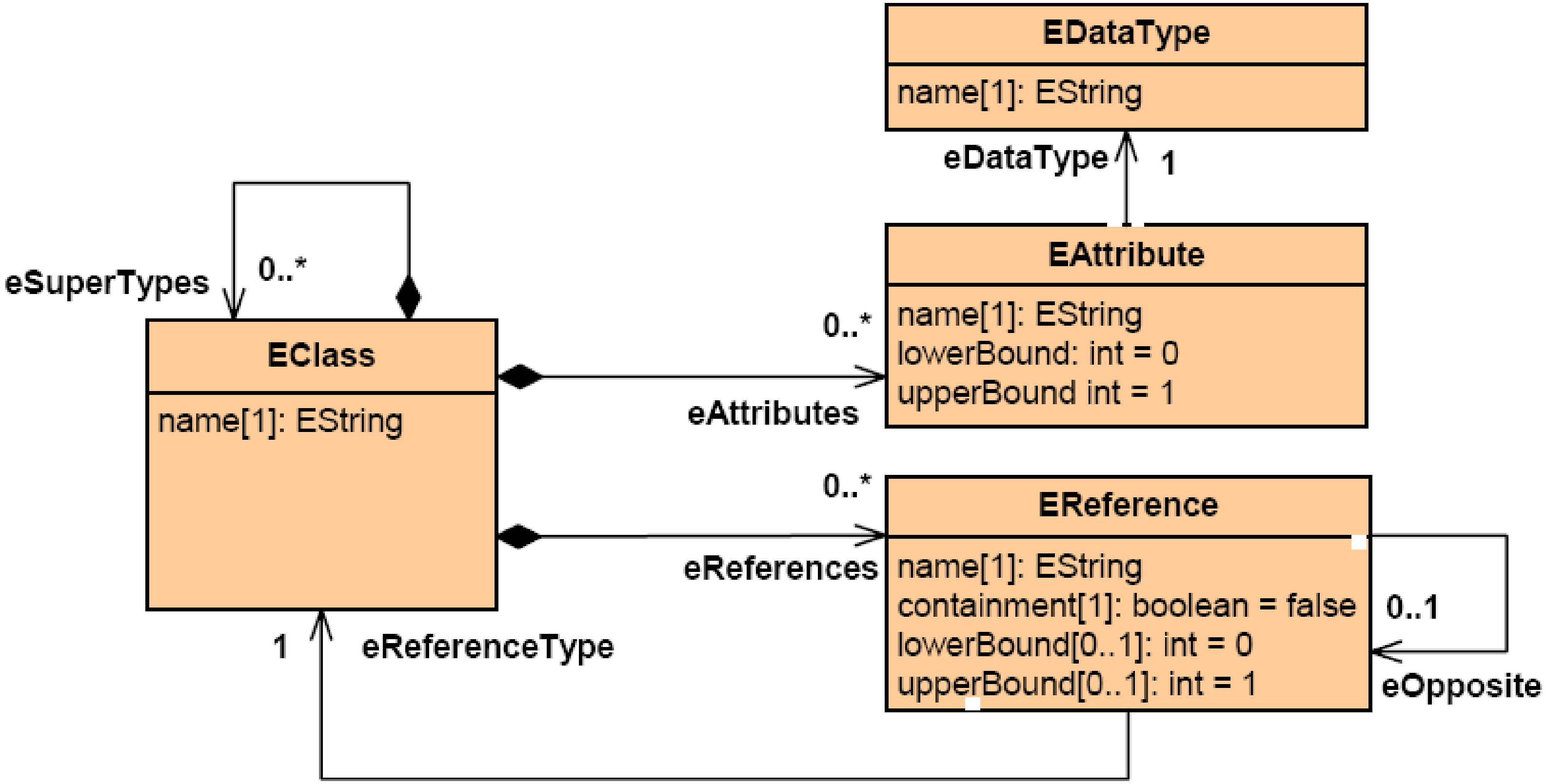
EMF model structure

- Containment hierarchy



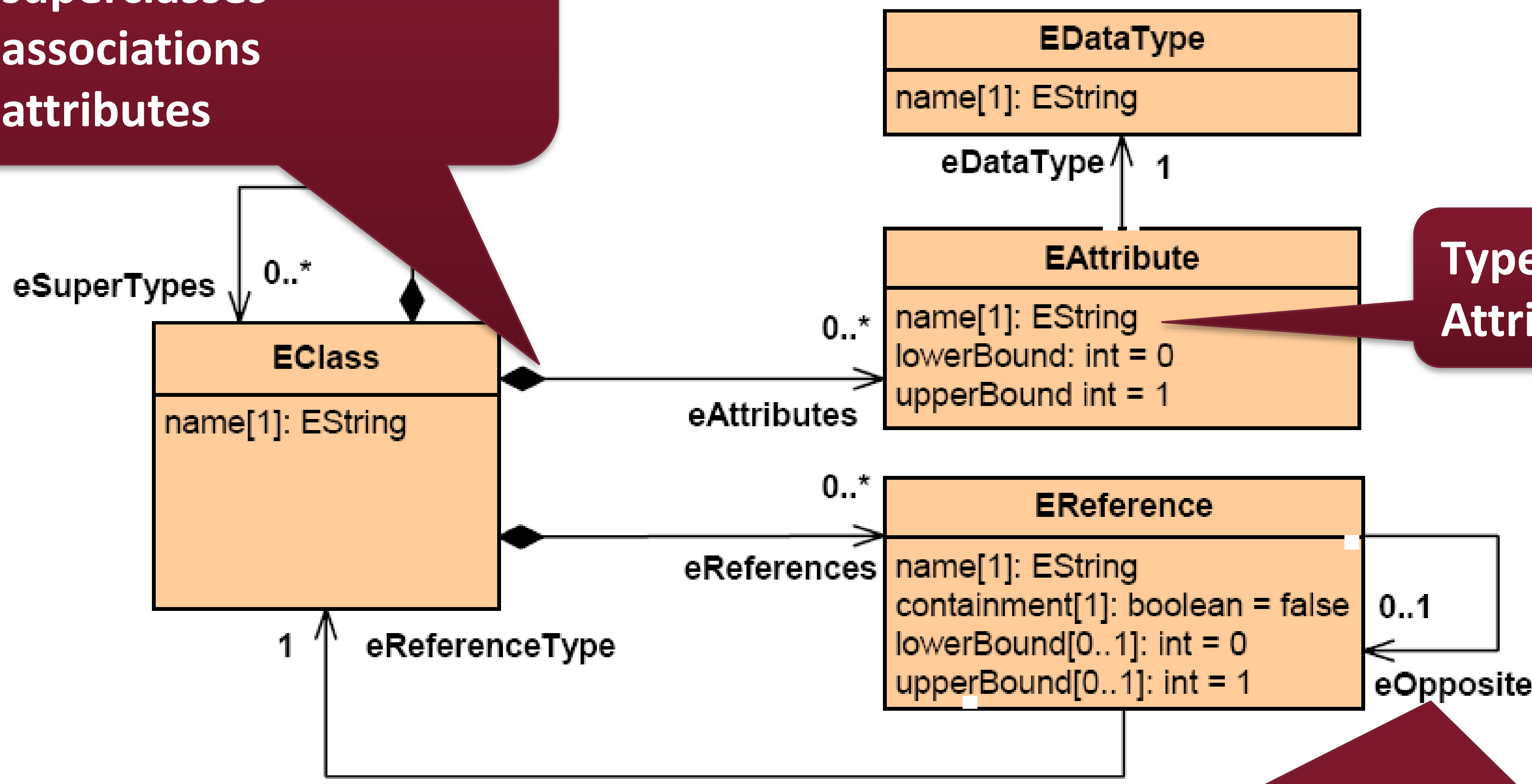
ECORE METAMODELLING

Core Ecore constructs



Core Ecore constructs

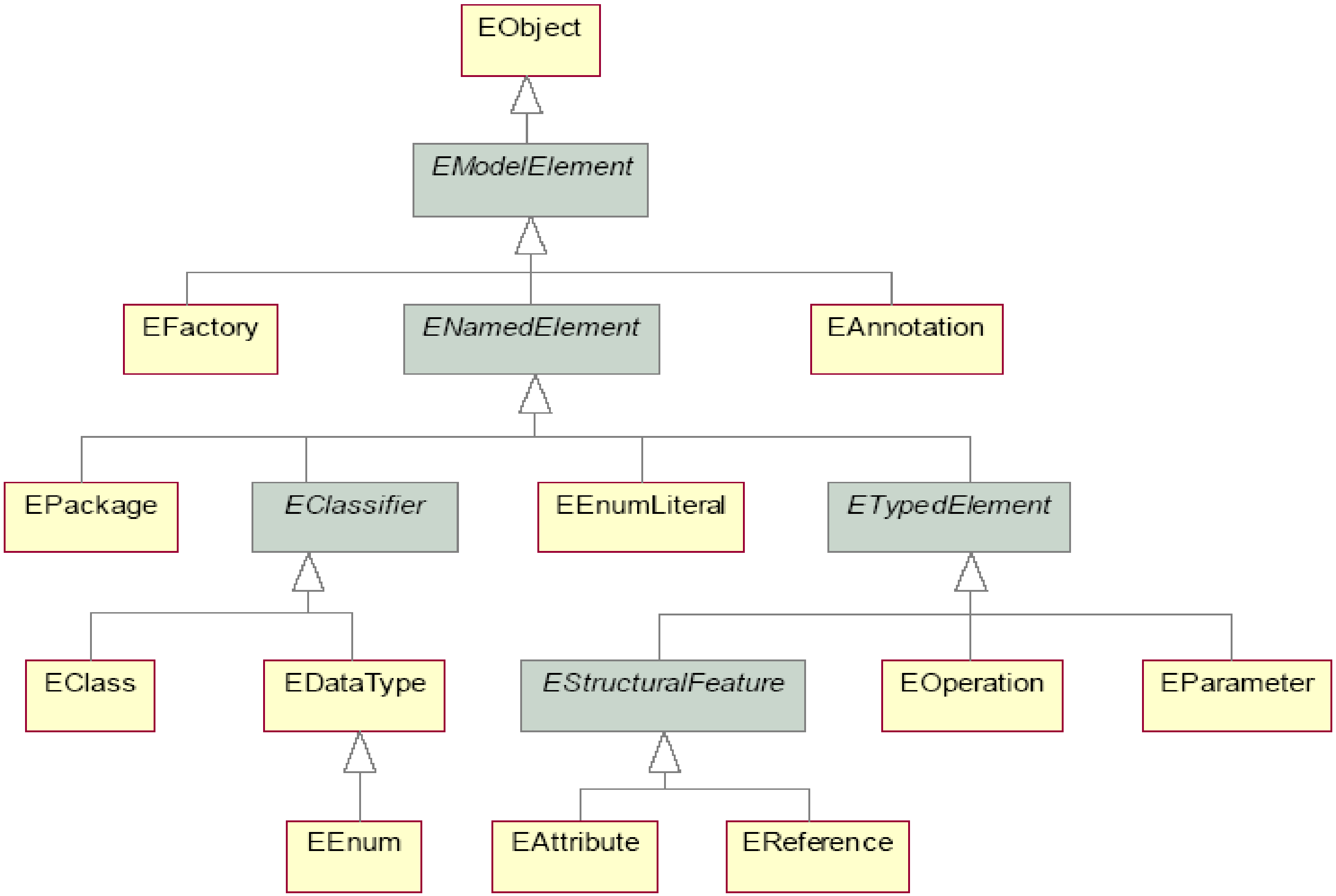
Class with arbitrary num. of
- **superclasses**
- **associations**
- **attributes**



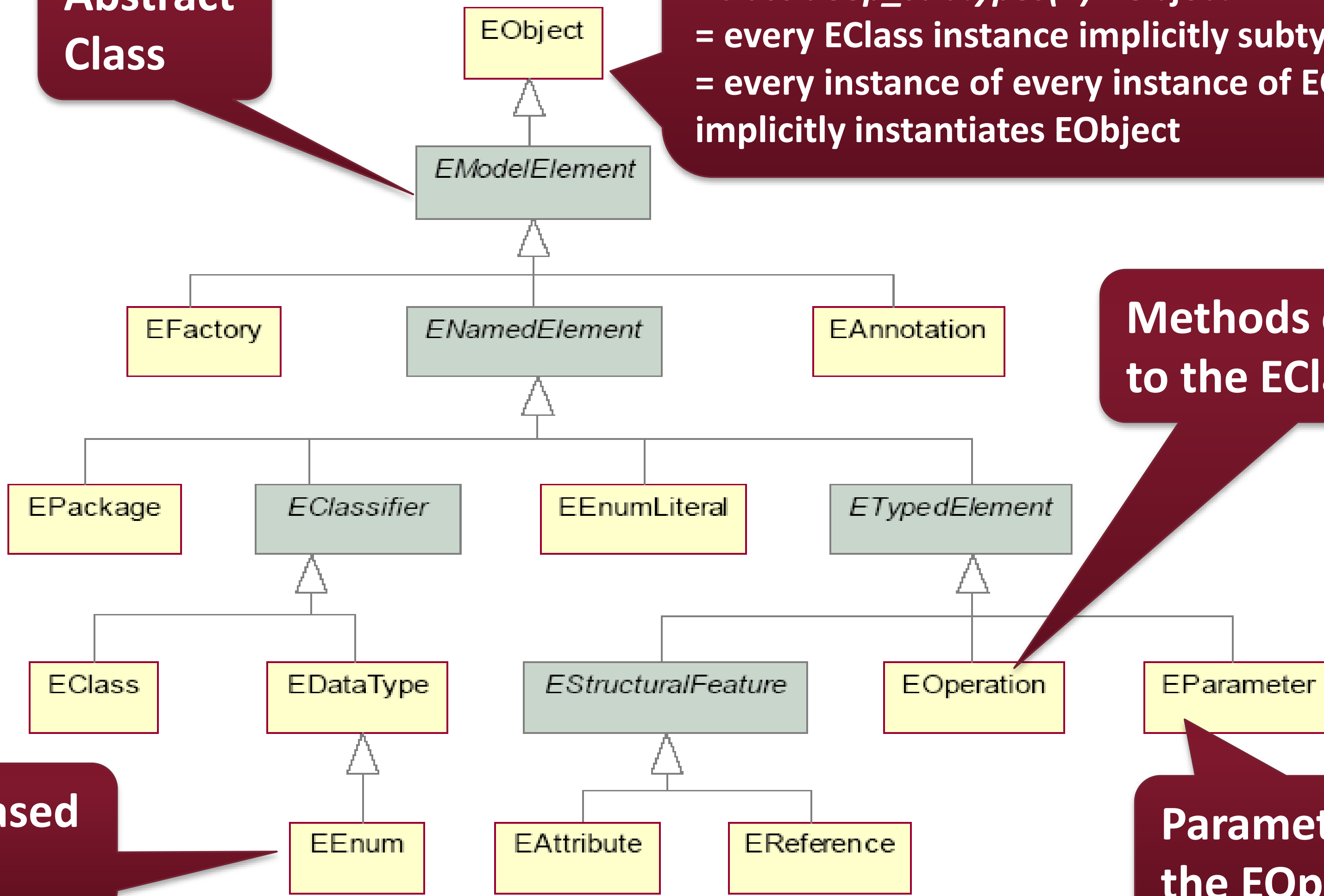
Typed Attribute

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

Complete Ecore hierarchy



Complete Ecore hiearchy



Abstract Class

Aside:
EClass *deep_subtypes(2)* EObject
= every EClass instance implicitly subtypes EObject
= every instance of every instance of EClass implicitly instantiates EObject

Methods connected to the EClasses

EMF-based Enums

Parameter for the EOperation

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

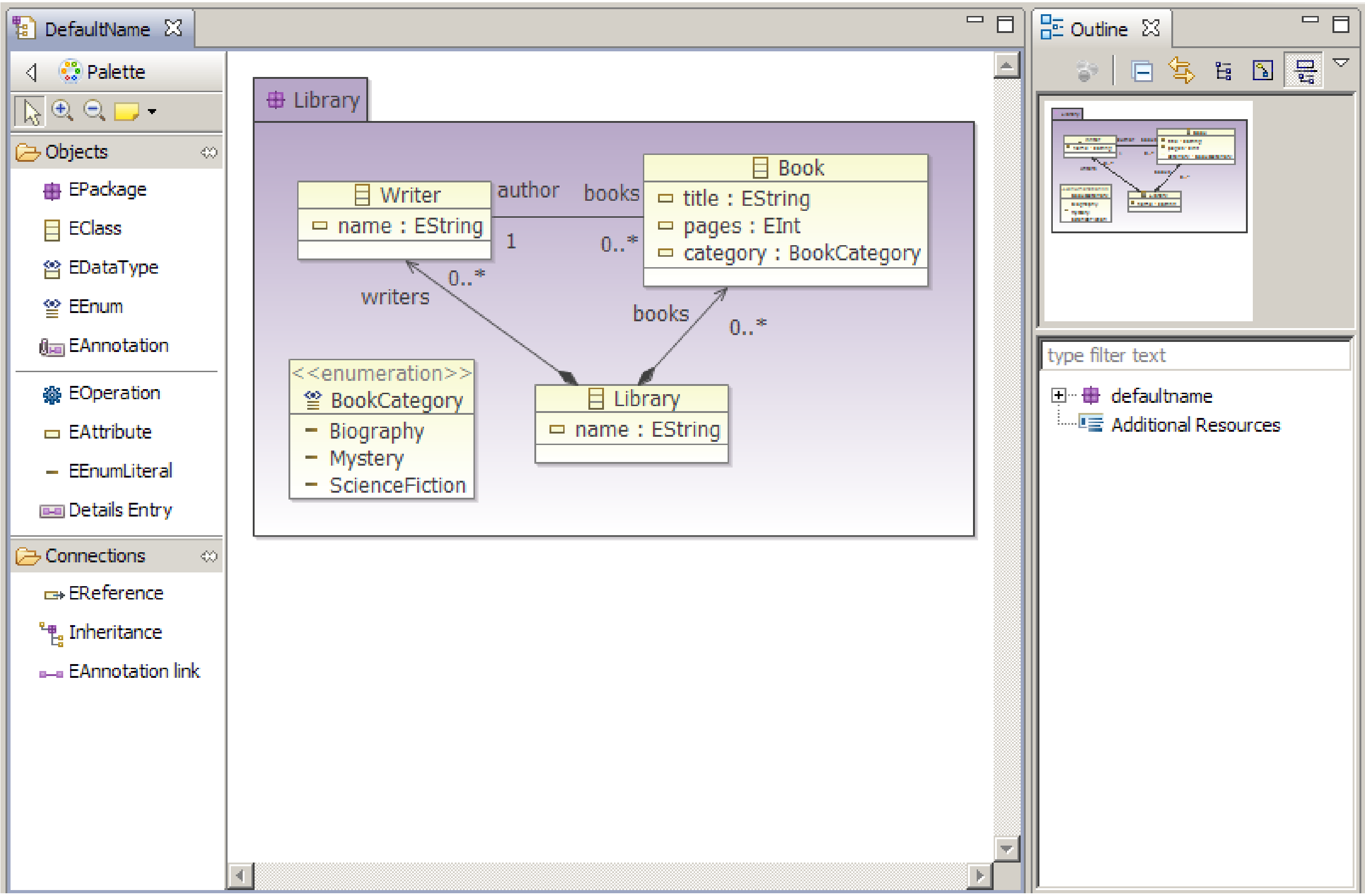
TOOLS, API AND UTILITIES

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

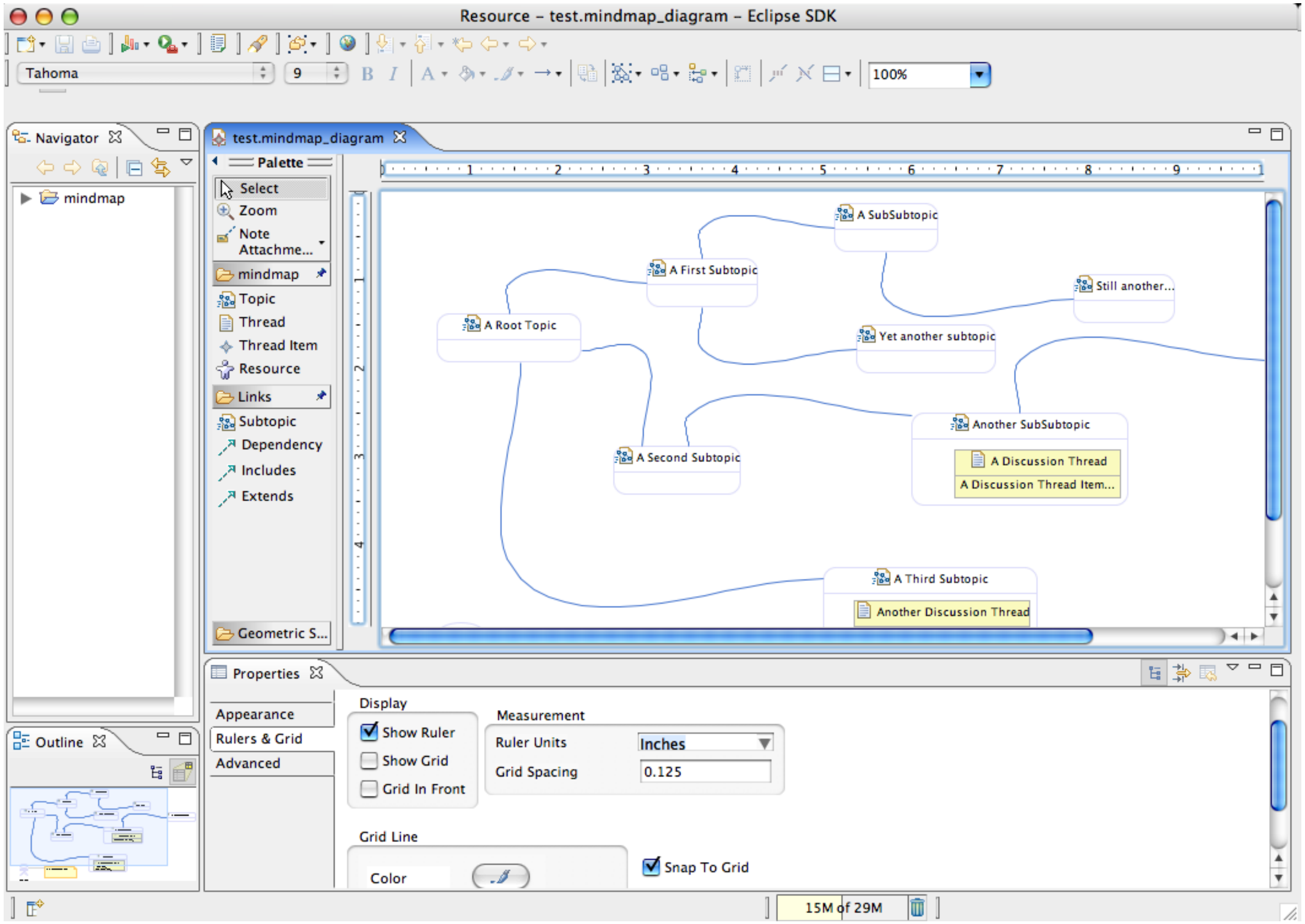
Ecore Tools: Ecore Diagram Editor

- Graphical DSL to define EMF metamodels
  - Based on GMF



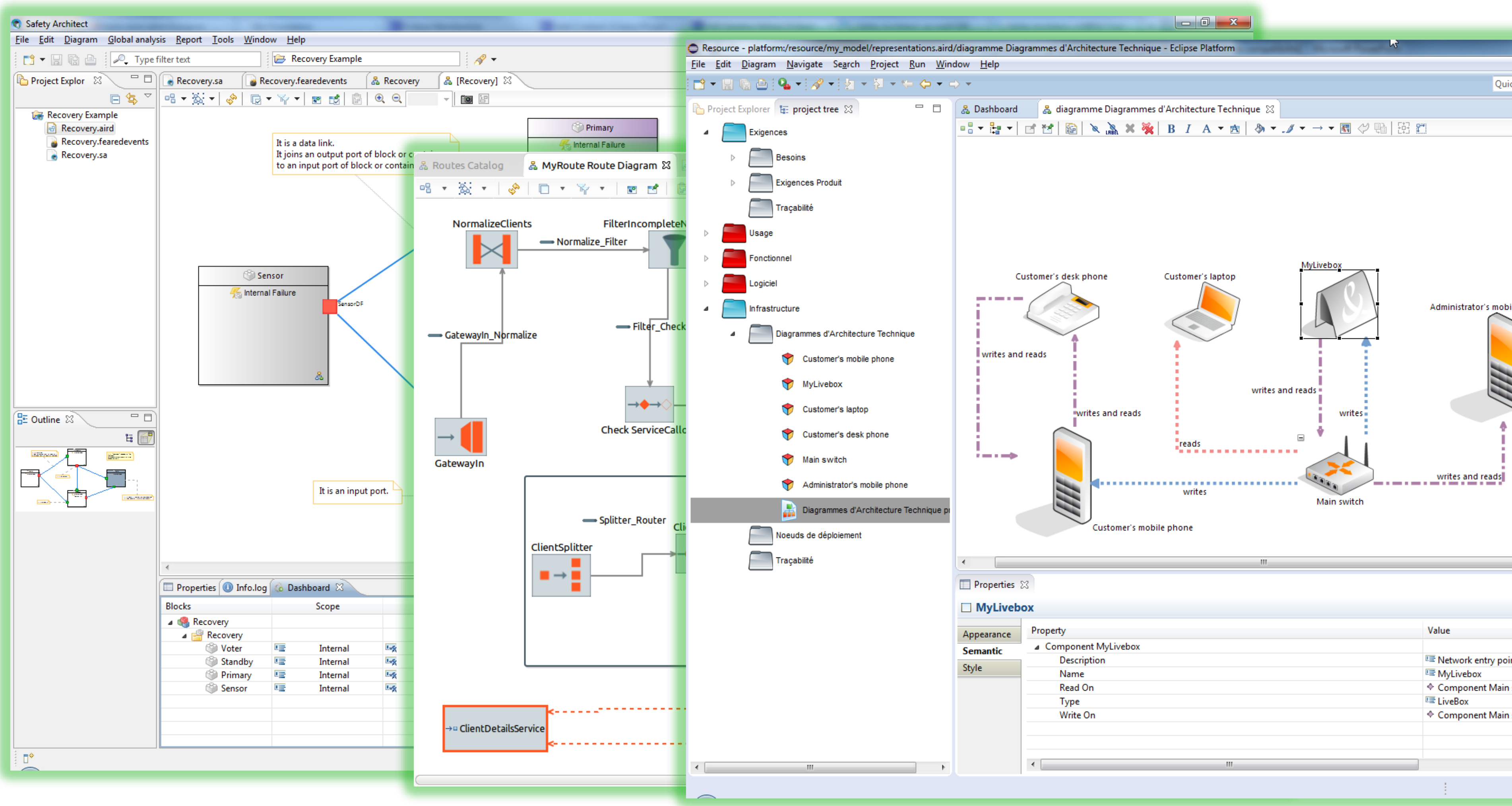
GMF

- DSL to define graphical concrete syntax



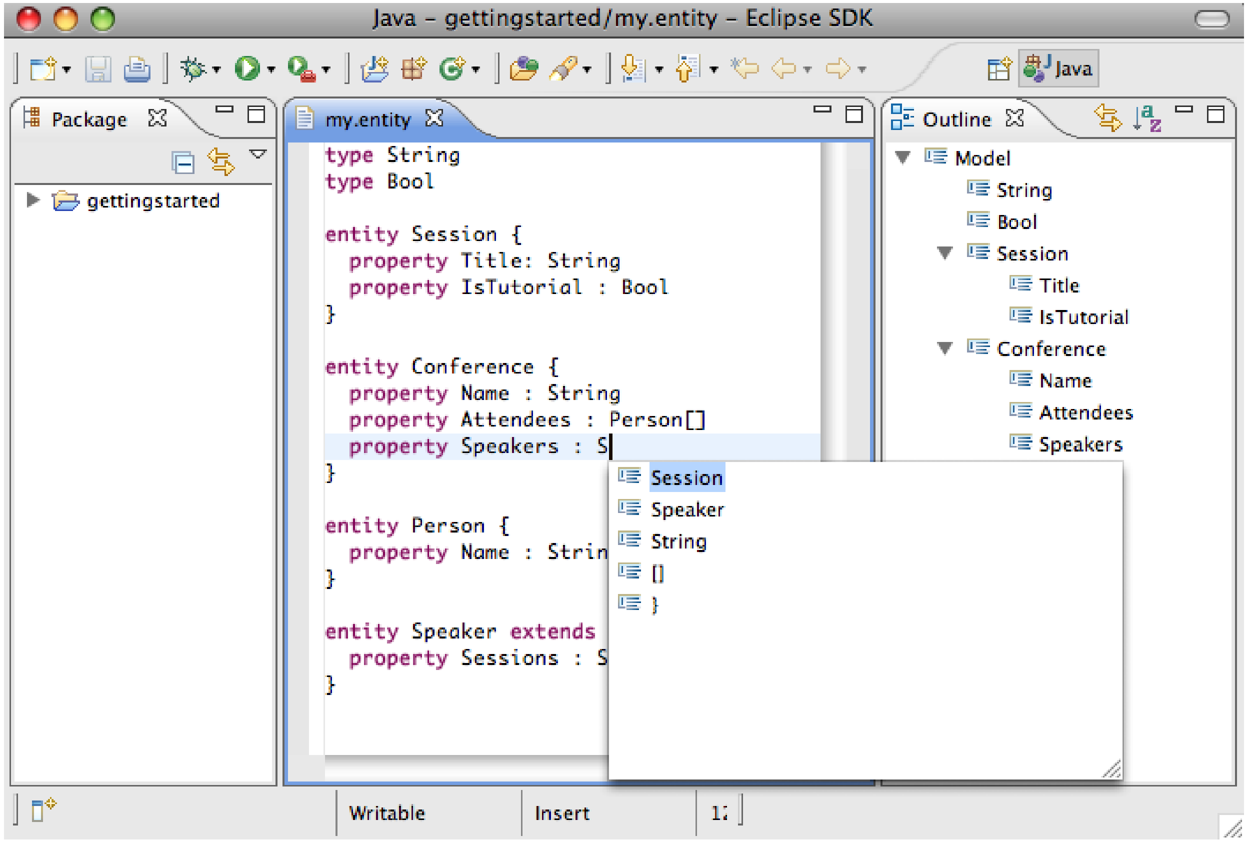
Sirius

- 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

Budapest University of Technology and Economics
Department of Measurement and Information Systems

OCL Motivation

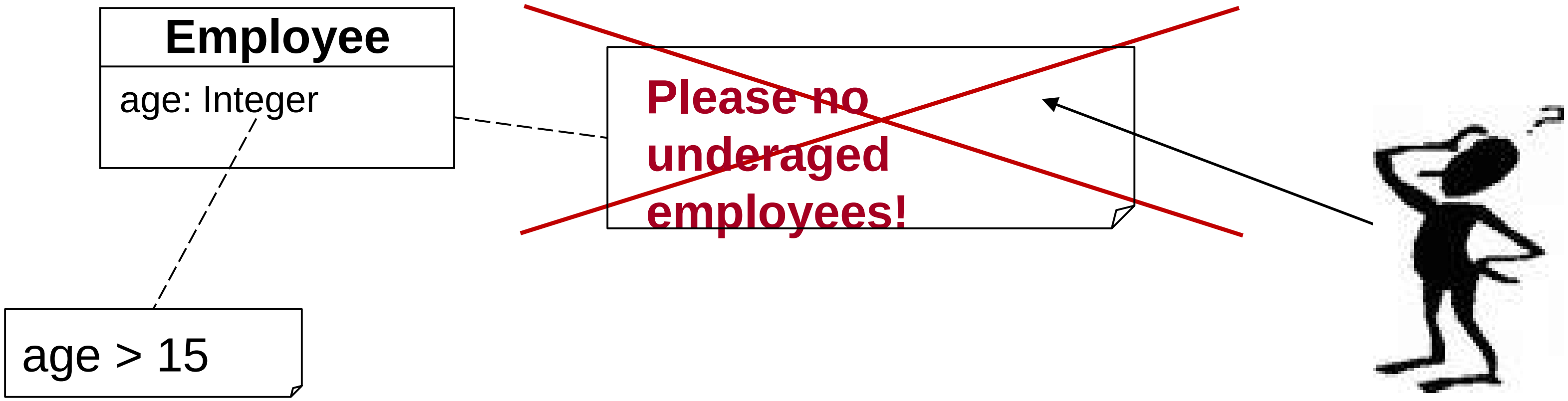
How to capture restrictions / constraints of domain classes?

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

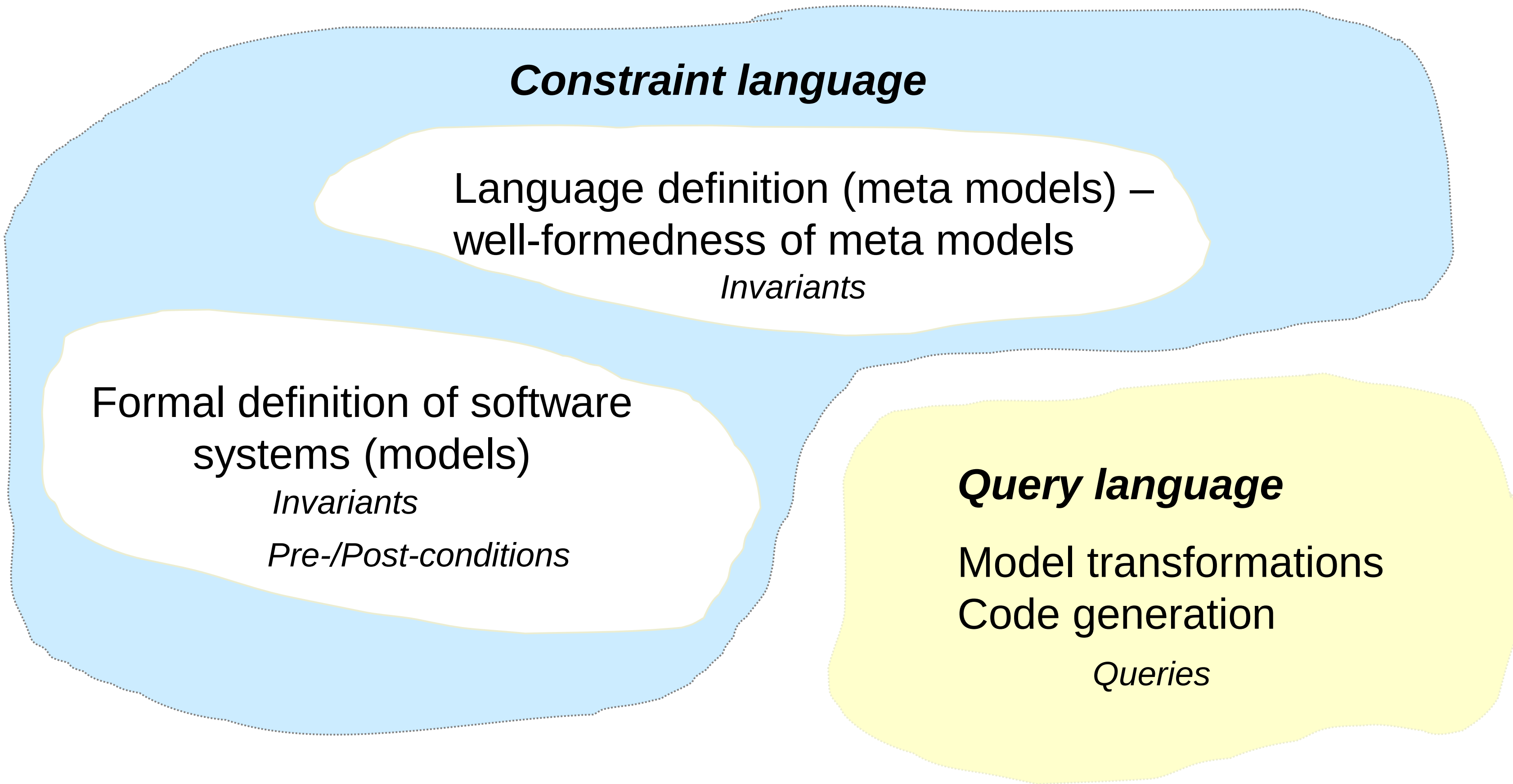
OCL usage

- 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*

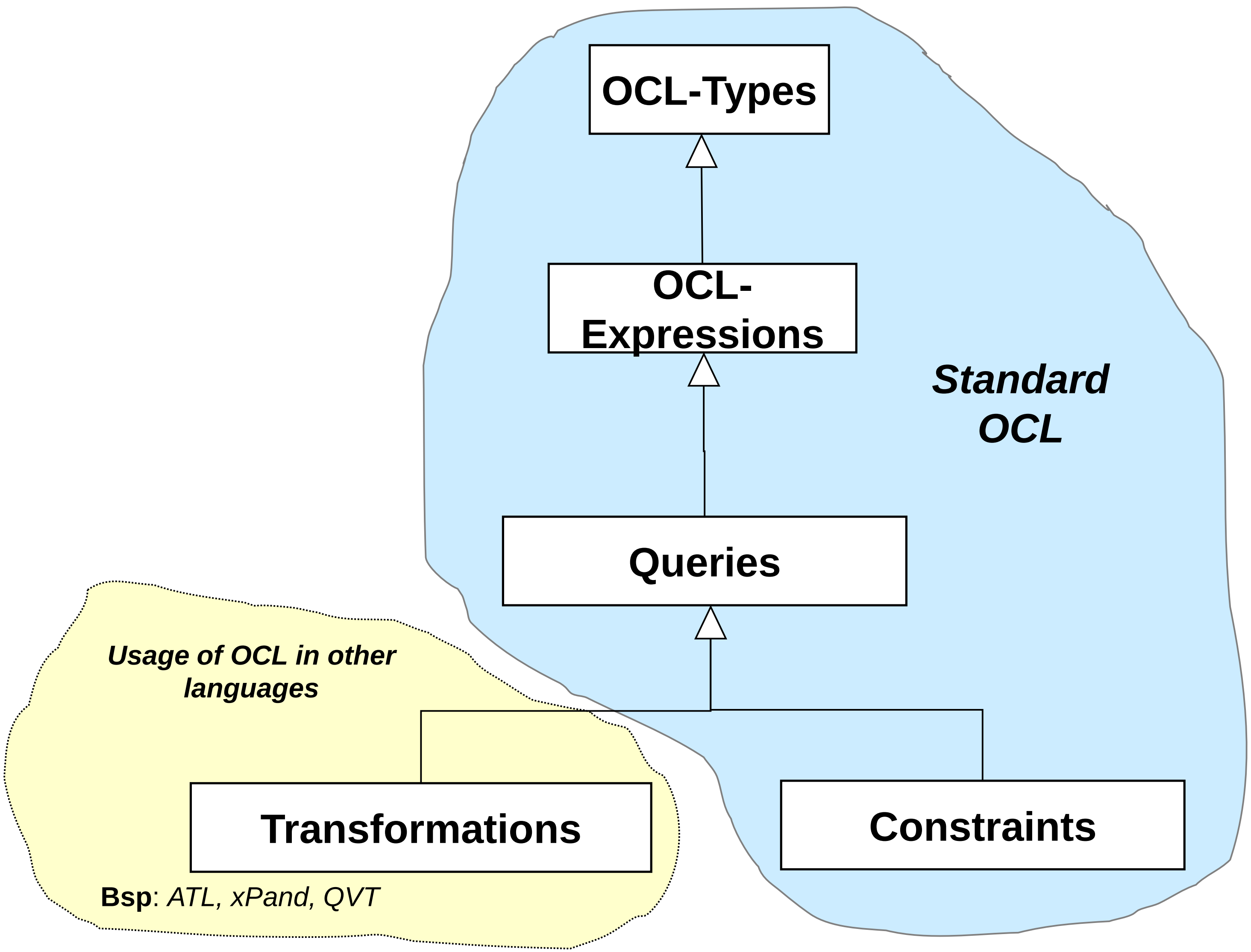
- Caution: Side effects are not allowed!
  - Operation `C::getAtt : String body: att` allowed in OCL
  - Operation `C::setAtt(arg) : T body: att = arg` **not** allowed in OCL

OCL usage

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



OCL usage

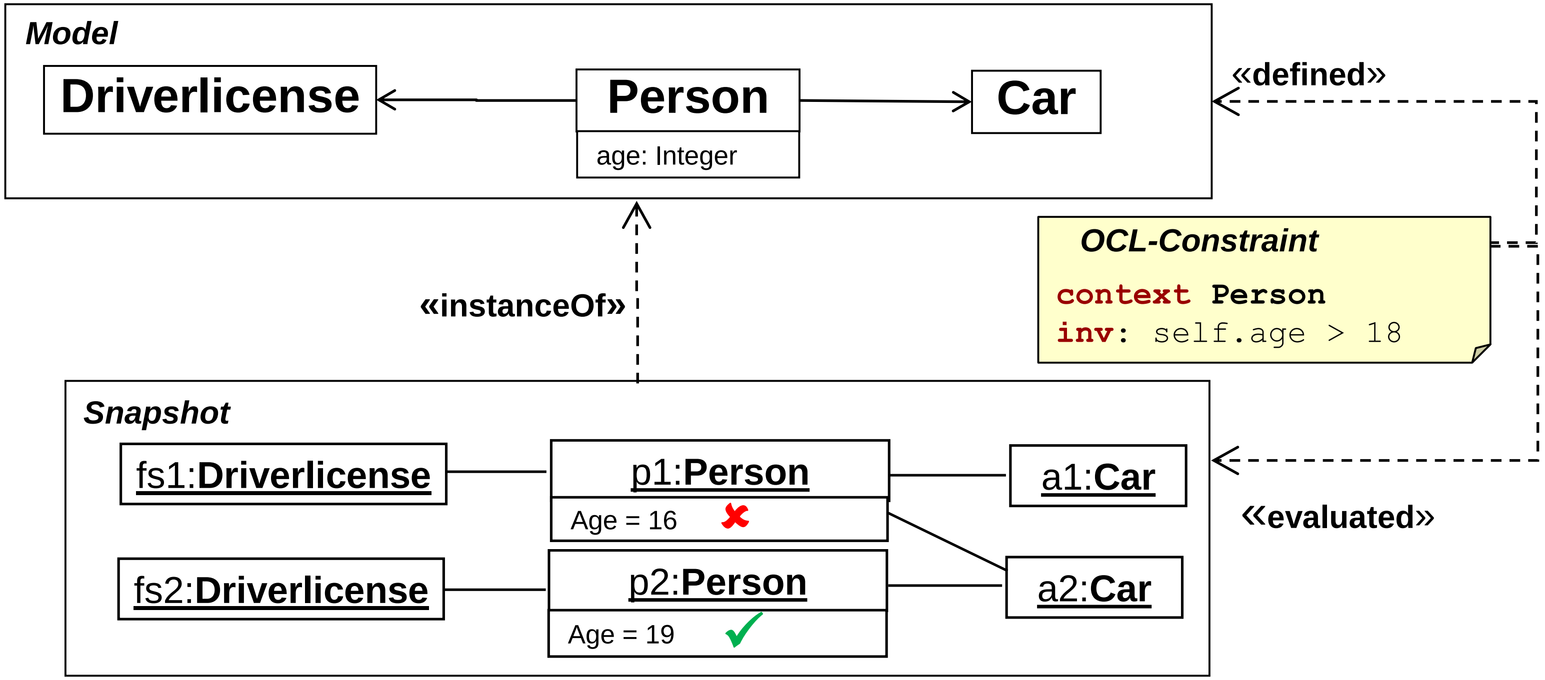

OCL-Types
OCL-Expressions
Standard OCL
Queries
Usage of OCL in other languages
Transformations
Constraints
Bsp: ATL, xPand, QVT

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

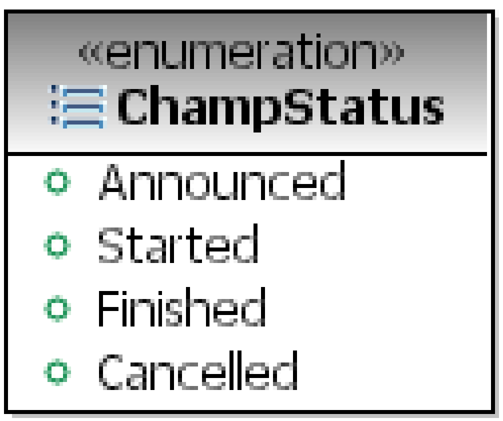


First OCL Examples

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 <> ''
```

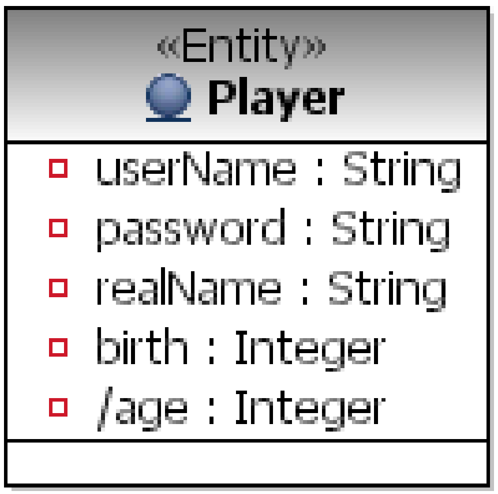
- Constraints on participants

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

Instance of the class

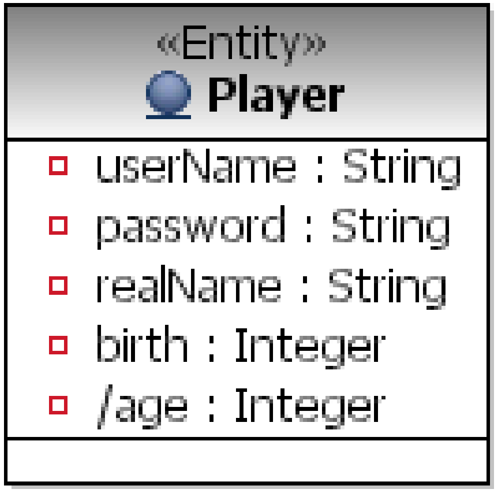
Navigation along attributes

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

Logical implication

- Name is unique

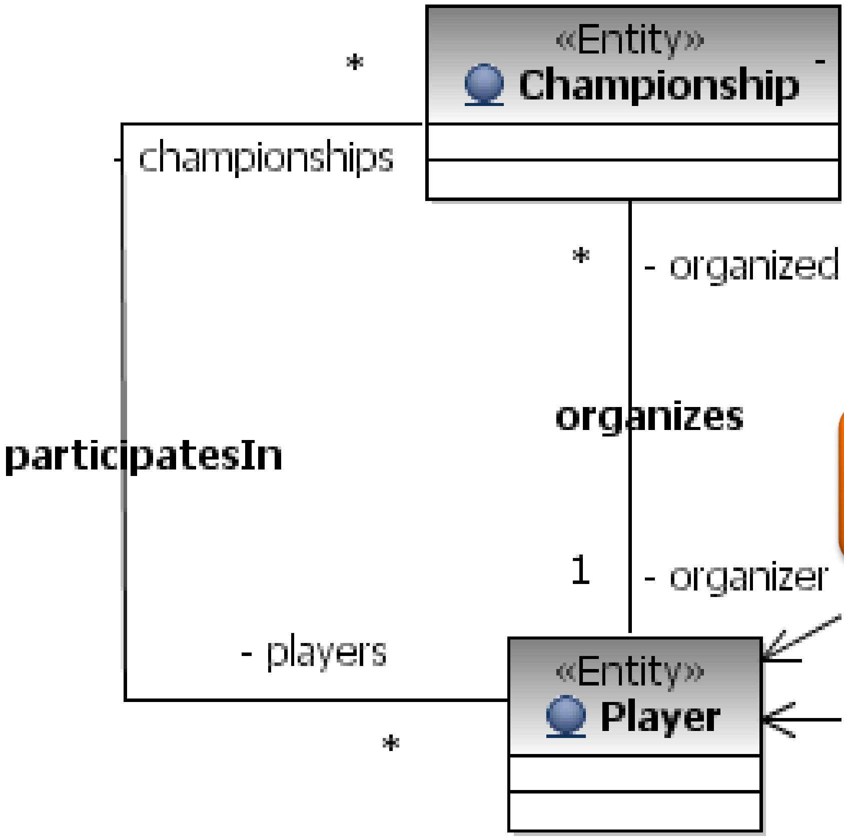
```
context Player inv:
  Player.allInstances()->
  forAll(p1, p2 | p1<>p2 implies
  p1.userName <> p2.userName)
```

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)

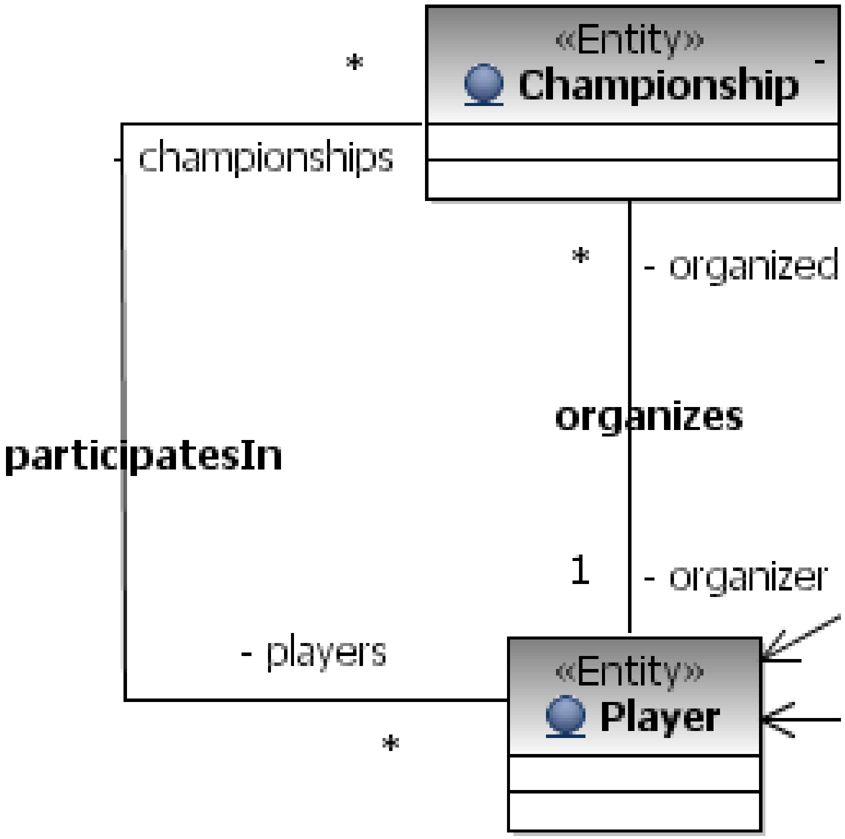
```
context Championship inv:
  self.players.birth > 1976
```

`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

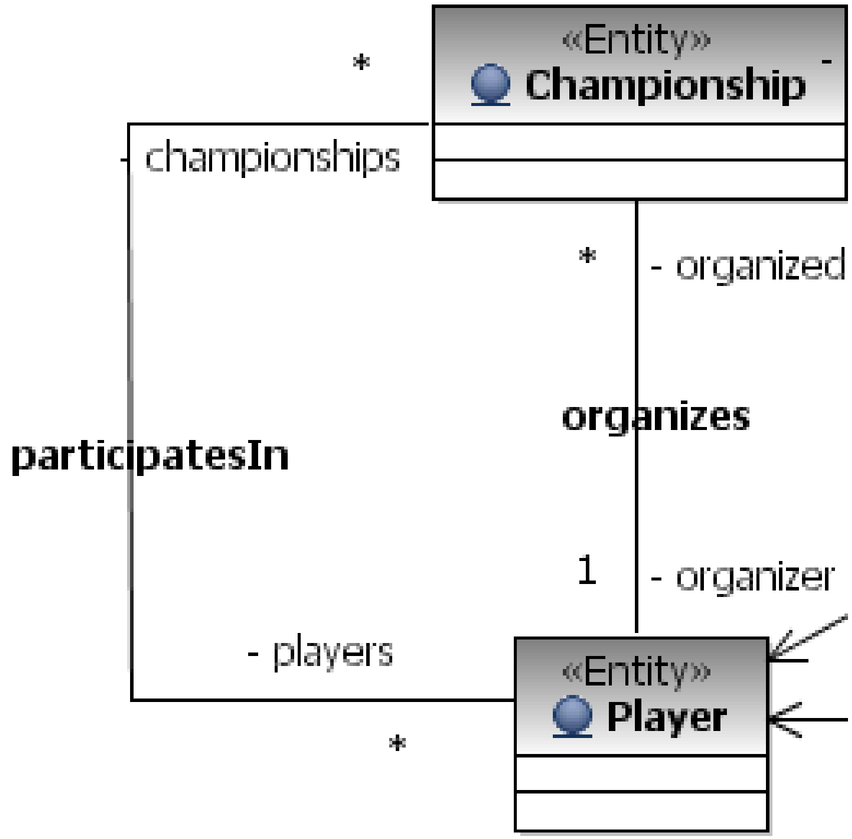
~~context Championship inv:
  self.organizer.organized=self~~

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

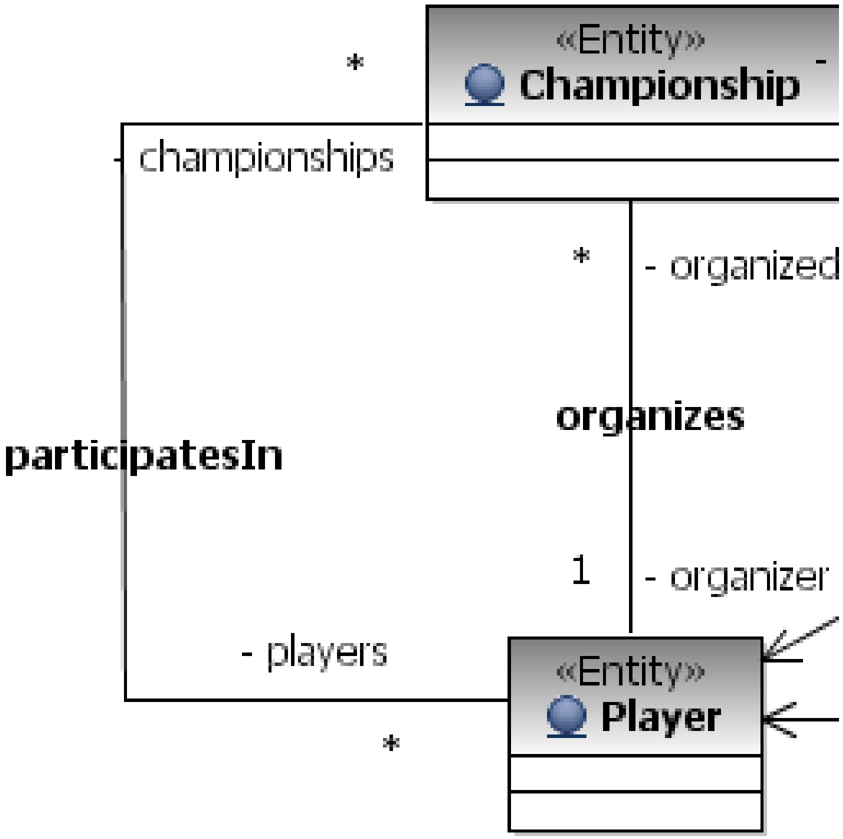
```
context Player inv:
  self.organized->exists(
  c | c.organizer = self)
```

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



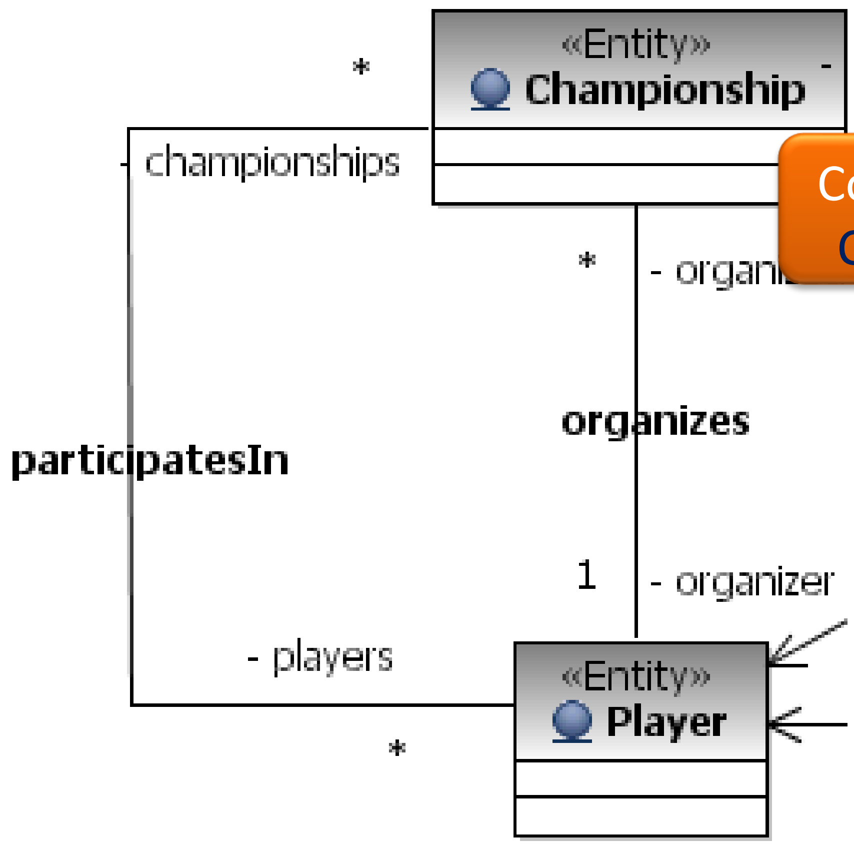
- If a bidirectional association exists between two objects then it is navigable from both directions

```
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
```



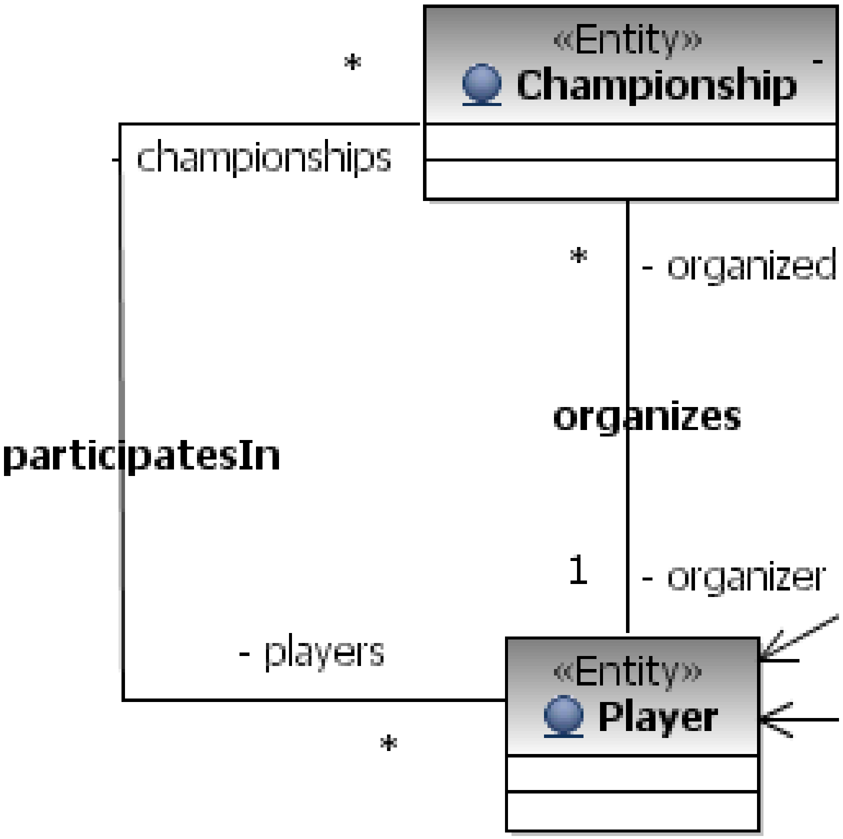
Context should be Championship

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

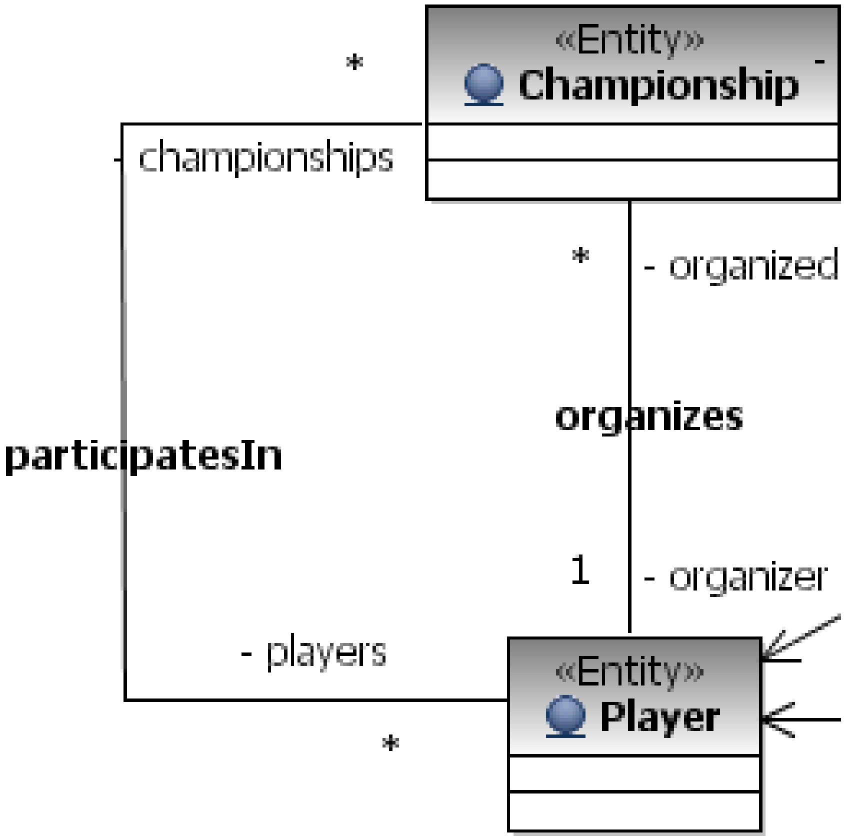
```
context Player inv:
   self.organized->
   forall(c1, c2 | c1<>c2 implies
   (c1.status = ChS::closed or
    c1.status = ChS::cancelled)
   or
   (c2.status = ChS::closed or
    c2.status = ChS::cancelled))
context Player inv:
   self.organized->select(c |
   c.status = ChS::announced or
   c.status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

Values of an enumeration

Application specific constraints

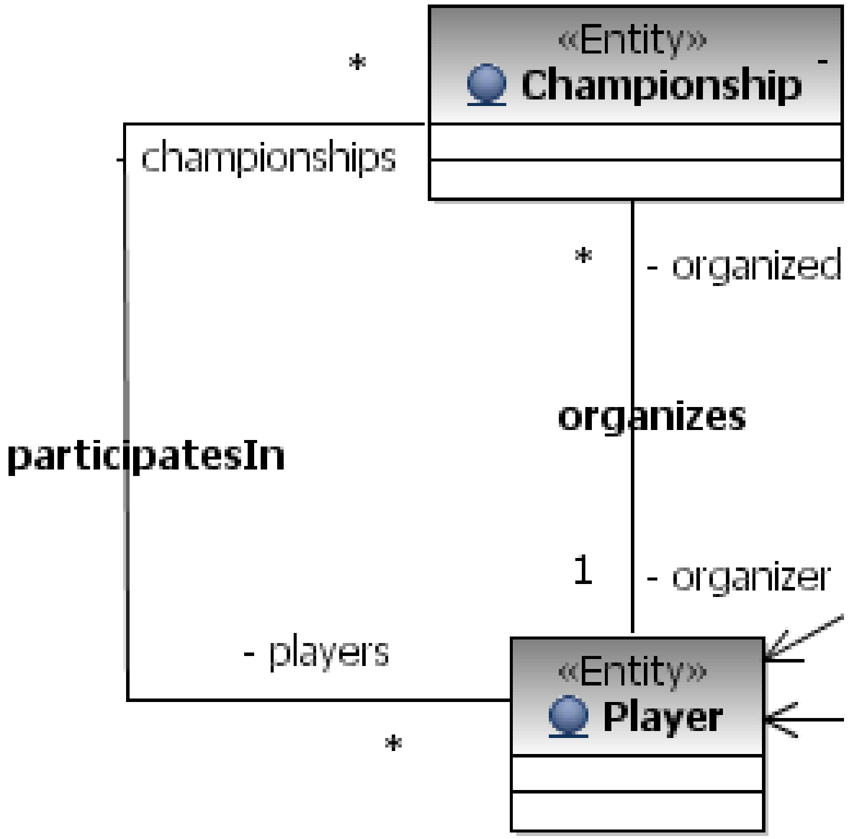
- A championship can only be started when the sufficient number of participants are present.

```
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:
  - `--`