

Domain-specific modeling

Ákos Horváth

Gábor Bergmann

Dániel Varró

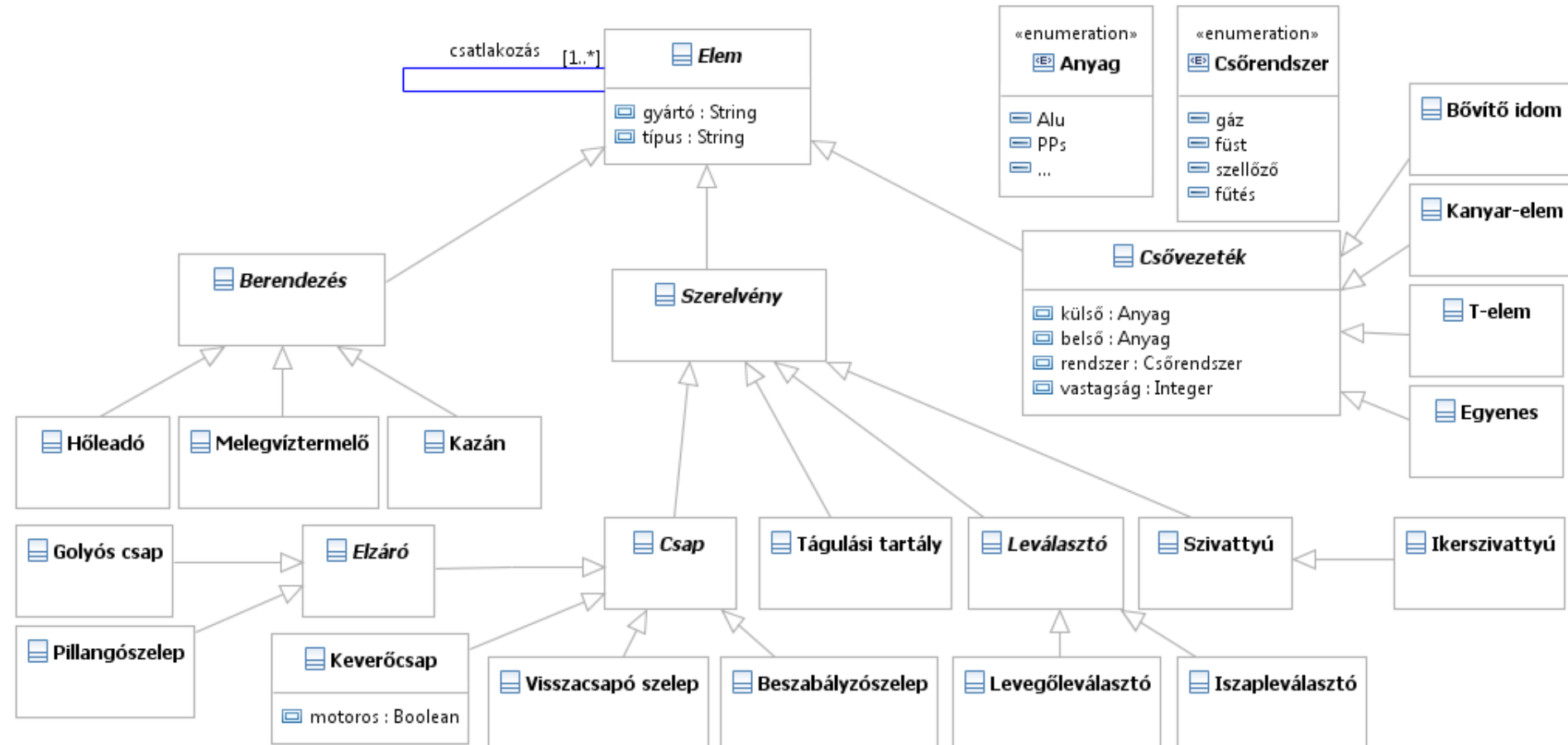
István Ráth

Model Driven Software Development

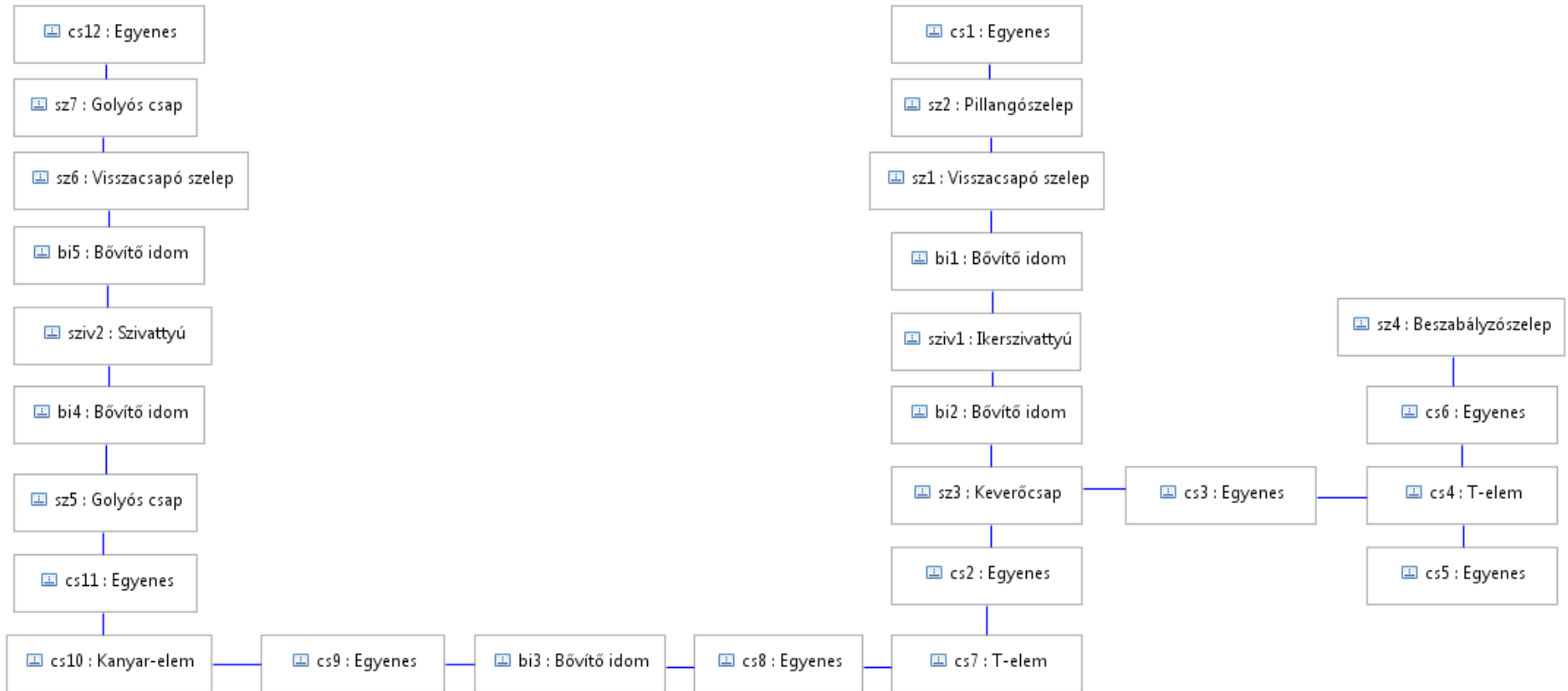
Lecture 3

MOTIVATION

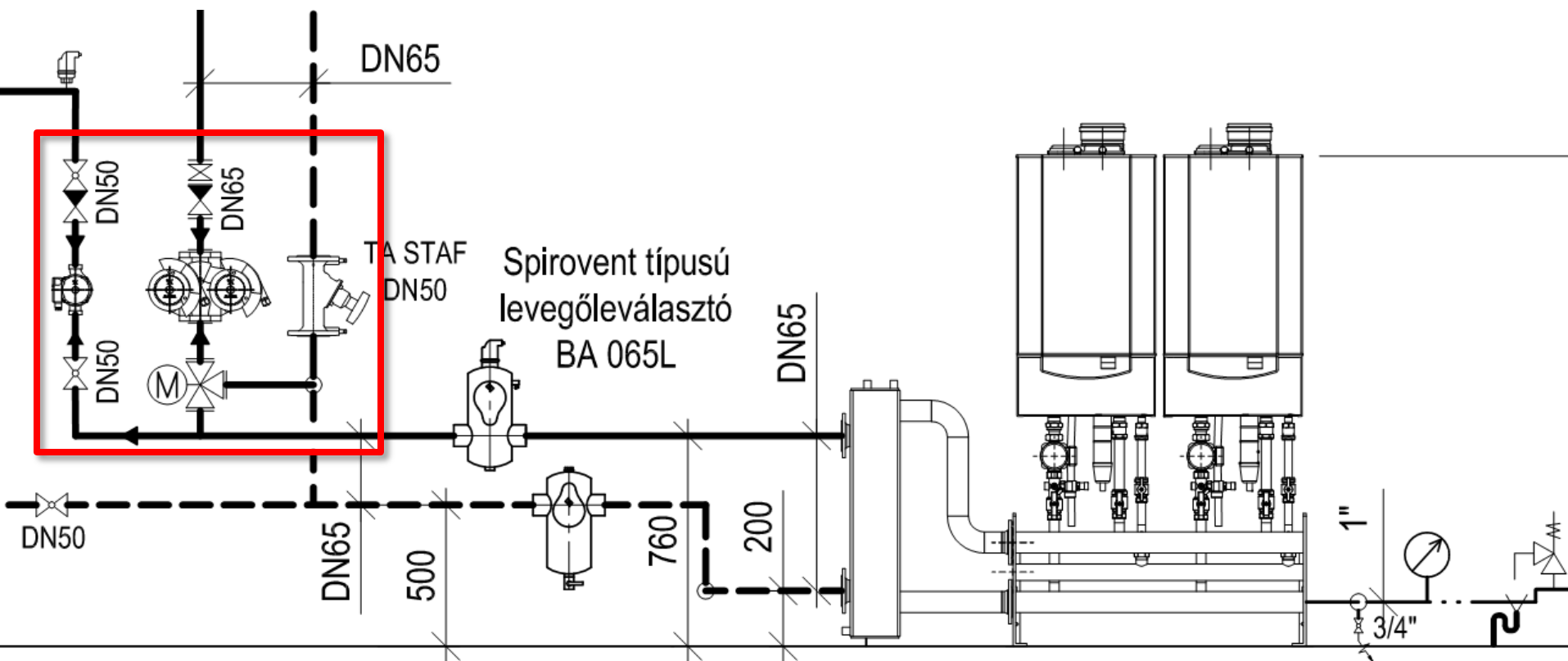
Example metamodel



Instance model, abstract syntax



Instance model, concrete syntax



Honeywell
keverőcsap
DN50 K_{vs} 40

Spirovent típusú
iszapleválasztó
BE 065L

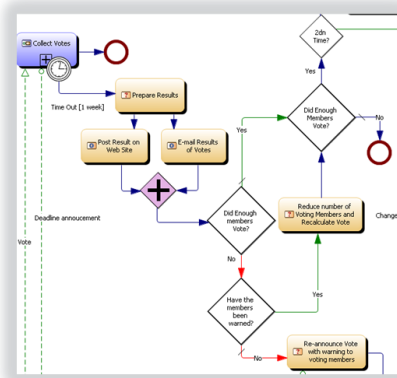
Remeha Quinta kaszkád
rendszer hidraulikus váltóval

OVERVIEW

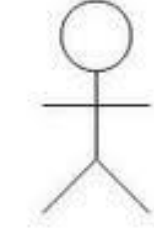
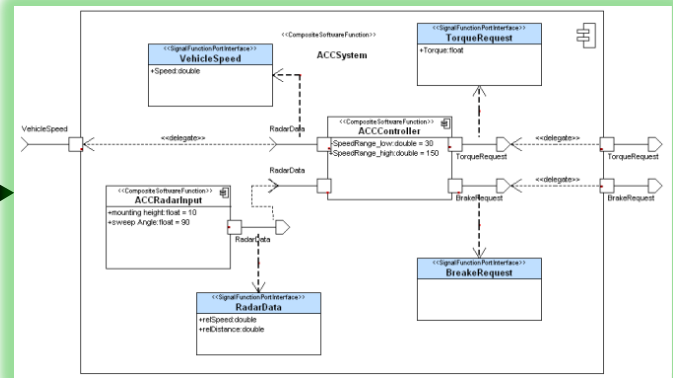
Domain specific modeling languages



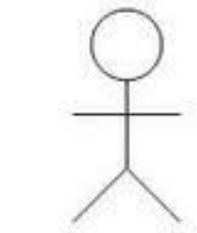
Business analyst



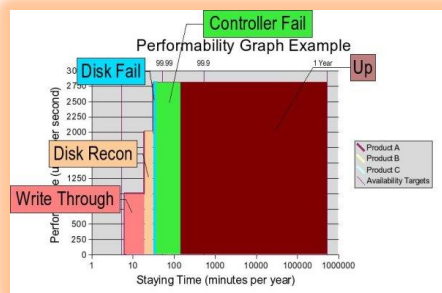
Business process



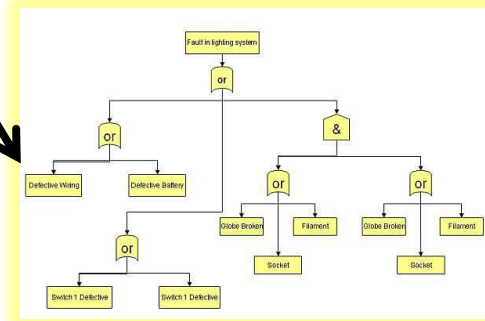
System designer



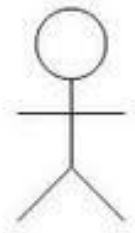
Dependability expert



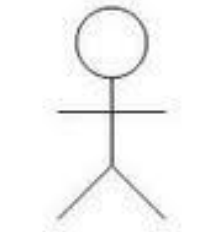
Dependability model



Risk model



Security expert



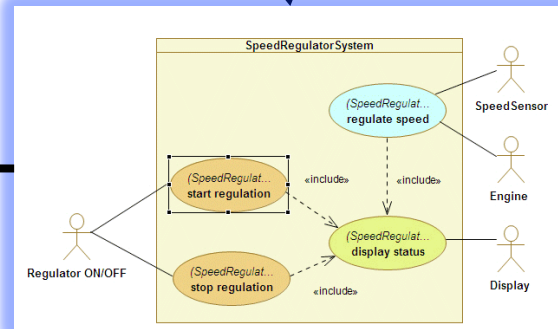
Software developer

```

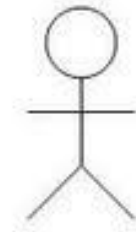
import com.lauchenauer.iStockHelper;
import com.lauchenauer.lib.util.Vertical;
protected CardLayout extends JDia
protected JButtonLayout mLayout;
protected JPanel mCredits;
public AboutDialog(JFrame owner) {
    super(owner);
    setModal(true);
    setUndecorated(true);
}
initUI();

protected void initUI() {
    setSize(440, 600);
    Container cont = getContentPane();
    JPanel p = ...
    
```

Programming language



Software model

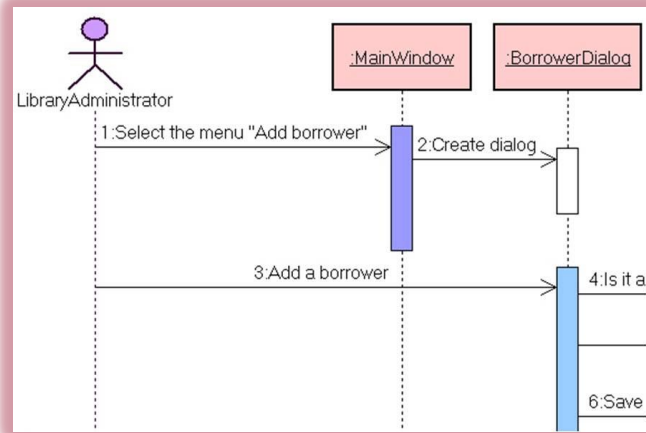


Software architect

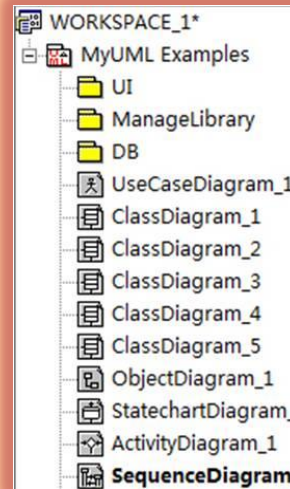


Usage example of DSMs

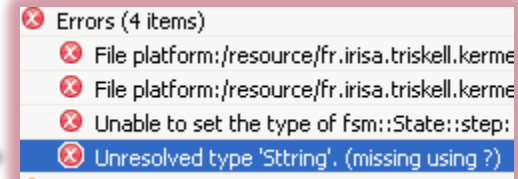
Concrete syntax



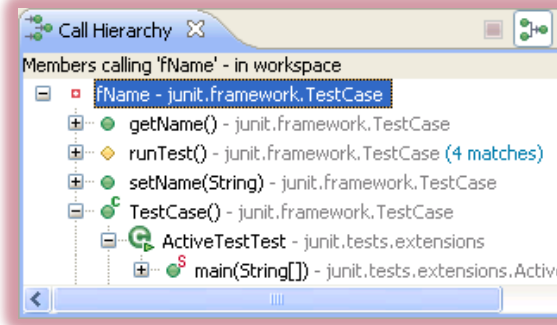
Abstract syntax



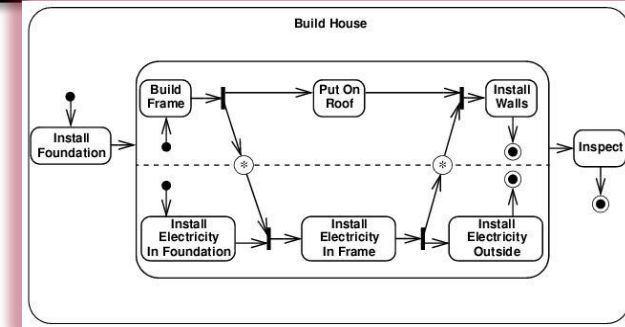
Well-formedness constraints



Behavioural semantics, simulation, refactoring



Call graph (view)



State machines (different DSM)

Structure of DSMs

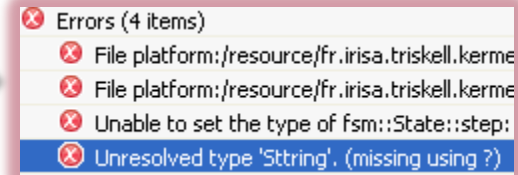
Graphical syntax



Abstract syntax



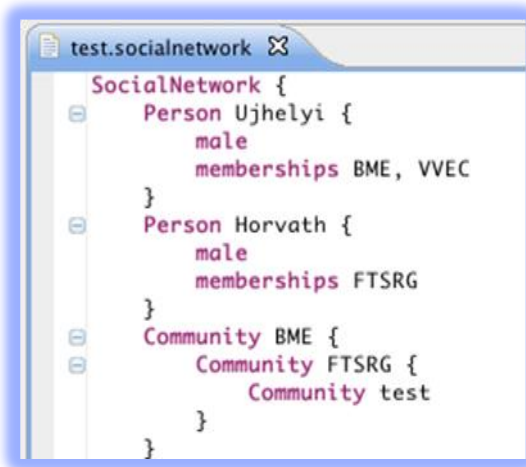
Well-formedness constraints



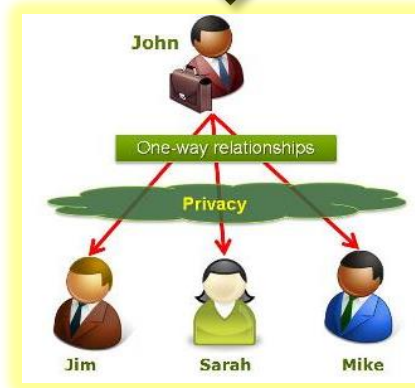
Behavioural semantics, simulation, refactoring

Code generation

Mapping



Textual syntax



View

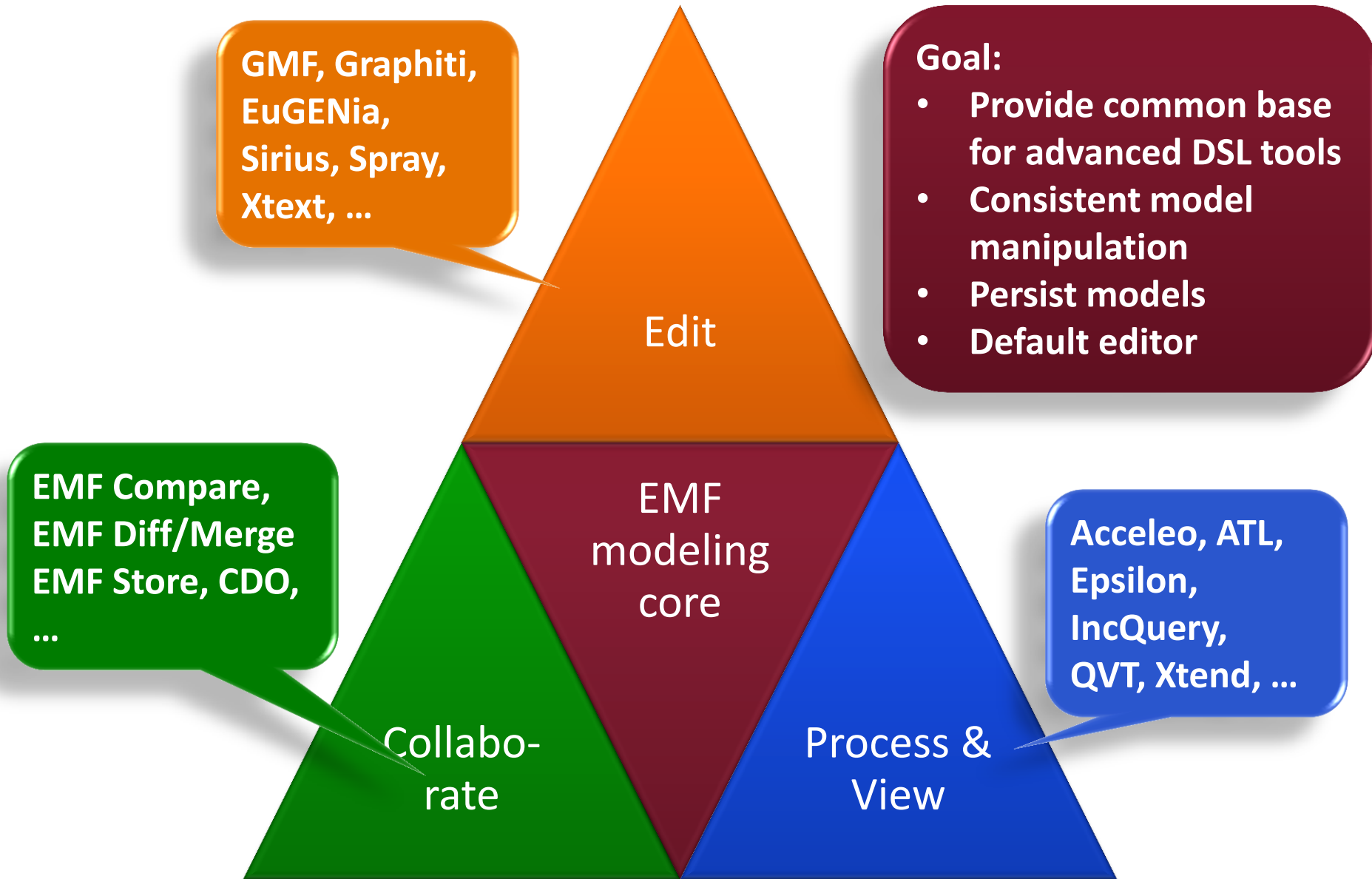
```
</membership>
<profile defaultProvider="Sitefinity">
  <providers>
    <clear/>
    <add name="Sitefinity" connectionS
  </providers>
  <properties>
    <add name="FirstName"/>
    <add name="LastName"/>
    <!-- SNP specific properties -->
    <add name="NickName" />
    <add name="Gender" />
  </properties>
</profile>
```

Code
(documentation,
configuration)

Aspects of Defining DSMLs



Role of EMF/Ecore technology in DSL



DOMAIN SPECIFIC MODELING

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

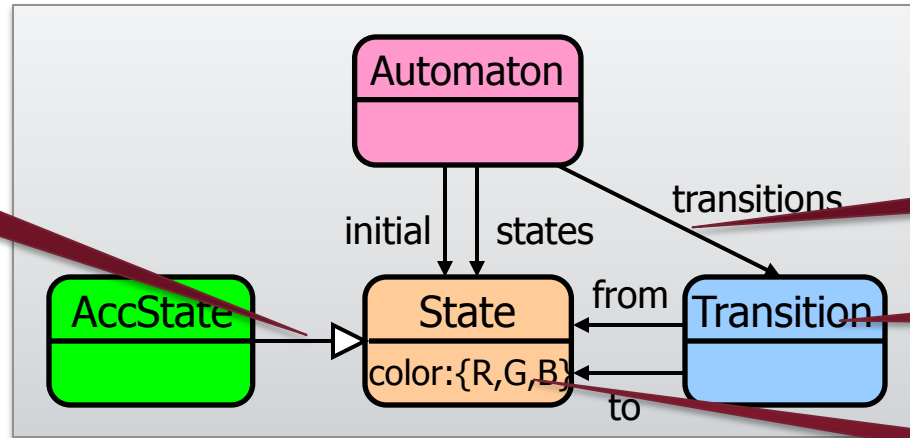
Generalization

Instantiation

Association

Class

Attribute



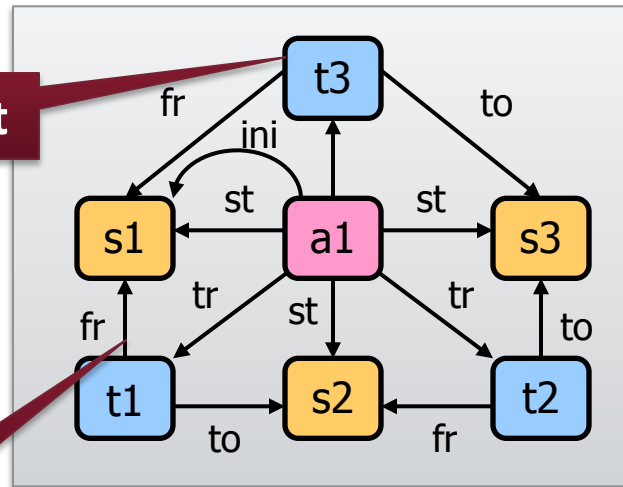
Metamodel

Meta (Language) level

(Instance) Model level

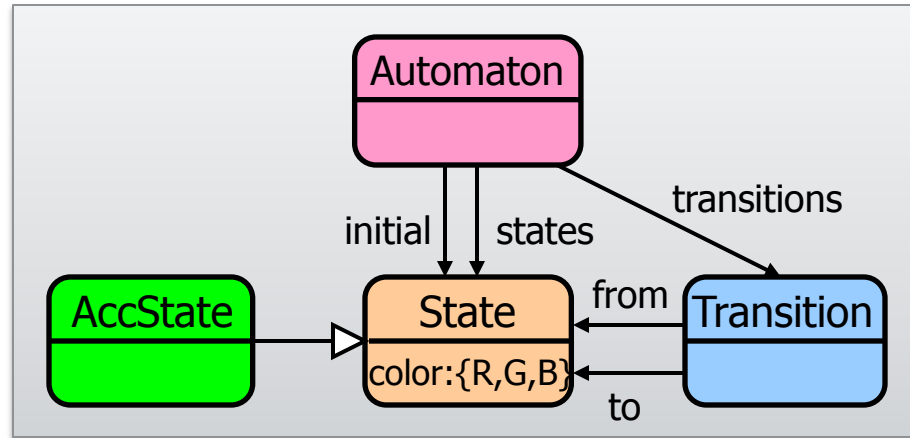
Object

Link



Model in abstract syntax

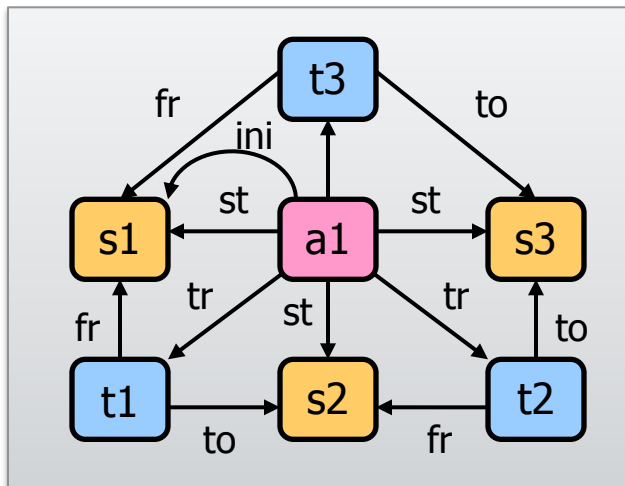
Revisiting the example



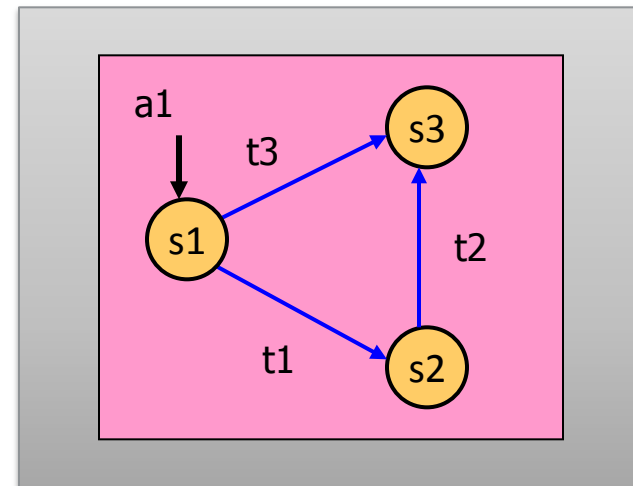
Metamodel

Meta (Language) level

Model level

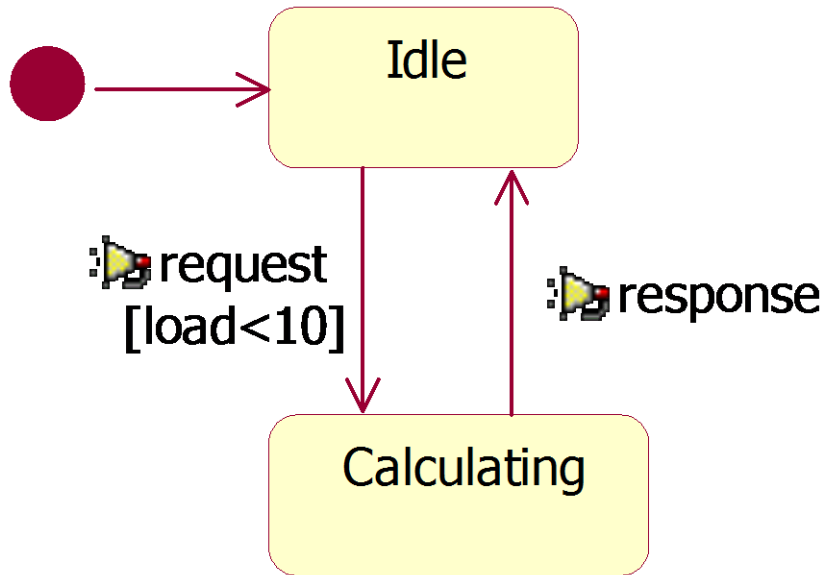


Abstract syntax



Concrete syntax

Example: Concrete Syntax



Graphical notation

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

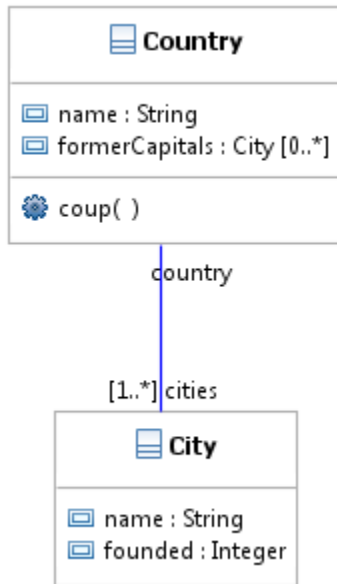
Textual notation

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

- <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:id="_fHicEC2vEd-VCP9iY9GYHg">
  <ownedAttribute name="name" aggregation="composite" xmi:id="_y"
    <type xmi:type="uml:PrimitiveType" href="pathmap://UML2:PrimitiveType.uml#name">
  </ownedAttribute>
  <ownedAttribute name="formerCapitals" aggregation="composite" xmi:id="_y"
    <upperValue value="*" xmi:type="uml:LiteralUnlimitedNatural" href="pathmap://UML2:LiteralUnlimitedNatural.uml#>
    <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_y" value="0" href="pathmap://UML2:LiteralInteger.uml#>
  </ownedAttribute>
  <ownedOperation name="coup" xmi:id="_fHicEC2vEd-VCP9iY9GYHg"
    <ownedParameter direction="return" xmi:id="_le7b8C2vEd-VCP9iY9GYHg"
      <type xmi:type="uml:PrimitiveType" href="pathmap://UML2:PrimitiveType.uml#>
    </ownedParameter>
  </ownedOperation>
</packagedElement>
<packagedElement xmi:type="uml:Class" name="City" xmi:id="_Xq"
  <ownedAttribute name="name" aggregation="composite" xmi:id="_y"
    <type xmi:type="uml:PrimitiveType" href="pathmap://UML2:PrimitiveType.uml#name">
  </ownedAttribute>
  <ownedAttribute name="founded" aggregation="composite" xmi:id="_y"
    <type xmi:type="uml:PrimitiveType" href="pathmap://UML2:PrimitiveType.uml#>
  </ownedAttribute>
</packagedElement>
<packagedElement xmi:type="uml:Association" xmi:id="_Xq"
  <ownedEnd name="cities" type="__KgpUC2vEd-VCP9iY9GYHg"
    <upperValue value="*" xmi:type="uml:LiteralUnlimitedNatural" href="pathmap://UML2:LiteralUnlimitedNatural.uml#>
    <lowerValue xmi:type="uml:LiteralInteger" value="1" href="pathmap://UML2:LiteralInteger.uml#>
  </ownedEnd>
</packagedElement>
</uml:Package>
  
```

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

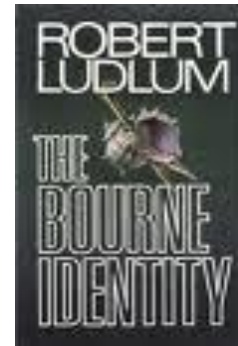
METALEVELS

■ 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:

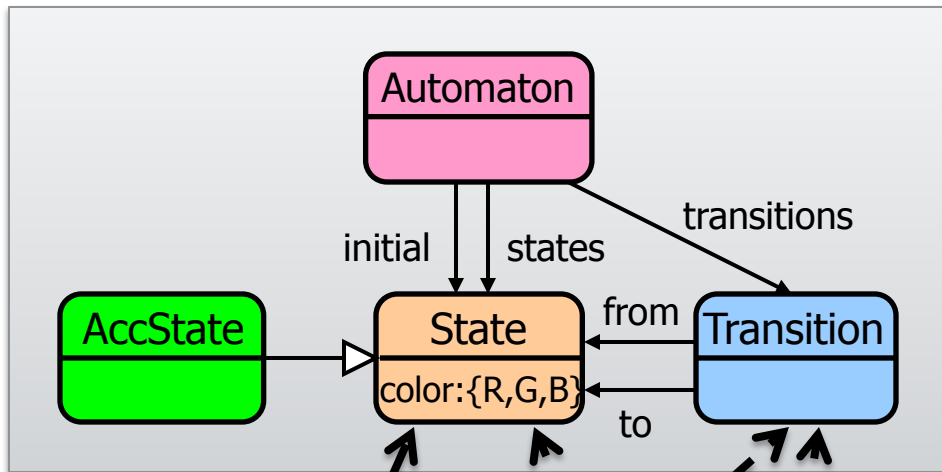
- Instantiation vs. subtyping
- Edge subtyping
- Metalevels
- Multi-level metamodeling
- Deep instantiation



■ Edges

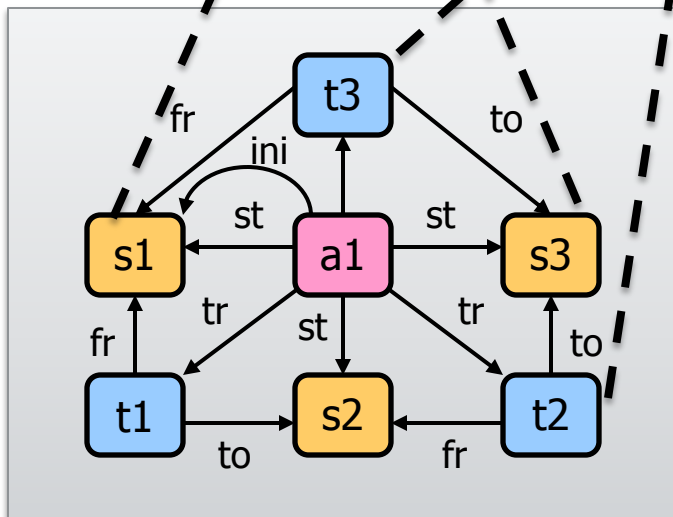
- written by, directed by, creator, subtype, instance

Metalevels



„Meta” relationship between models

«instance» ...

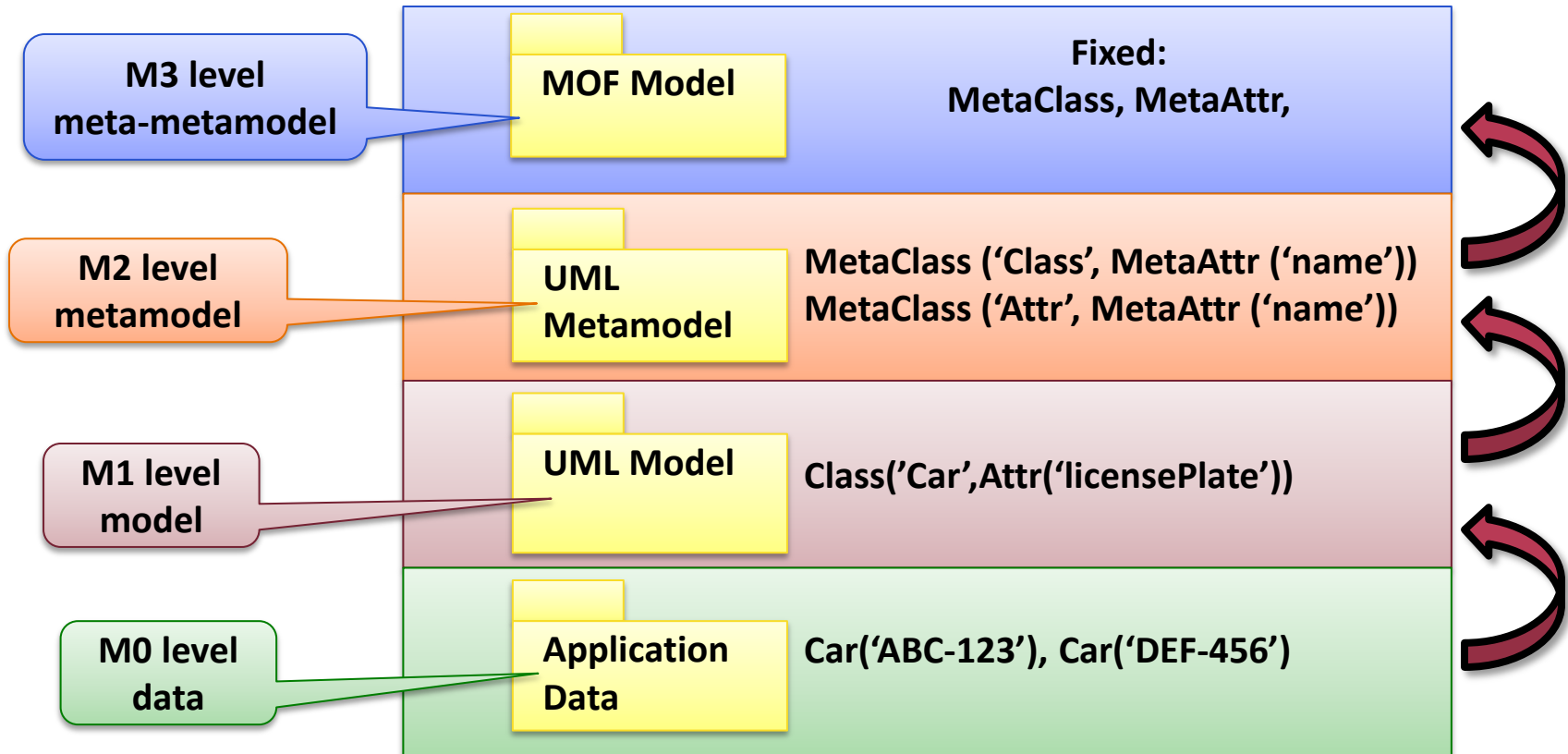


Clear level separation:

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

Metalevels in MOF

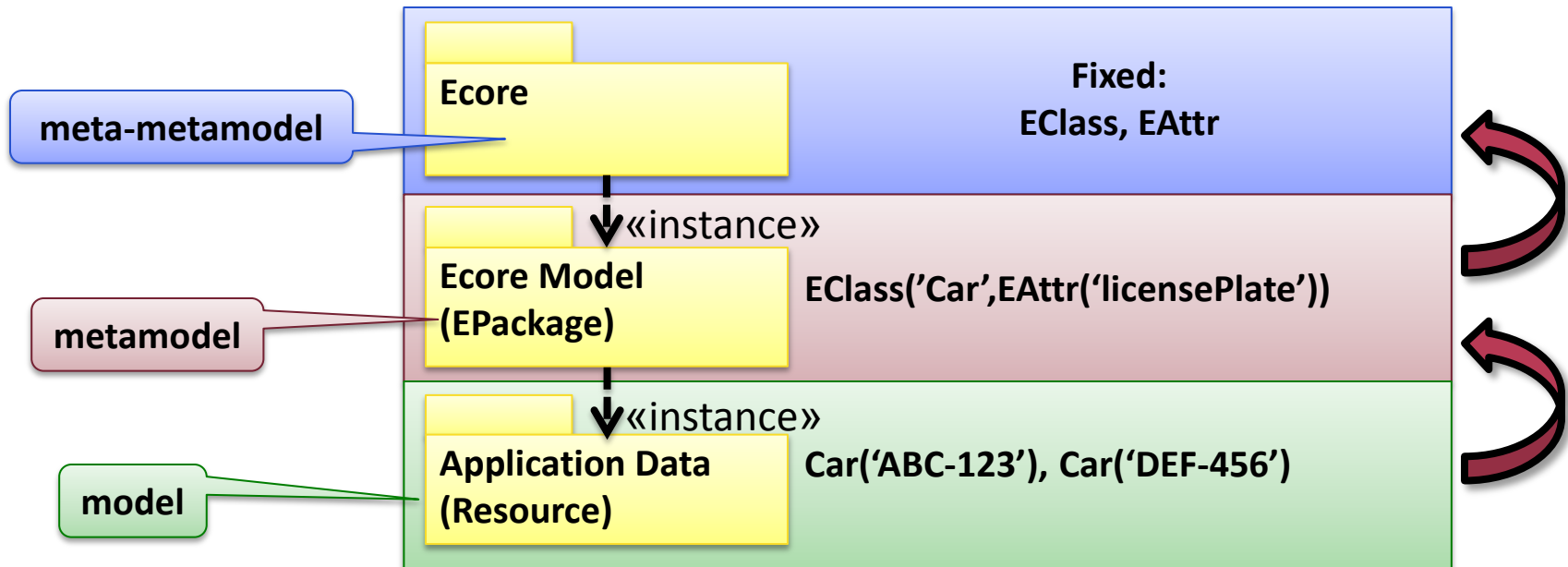
- **OMG's MOF (Meta Object Facility)**
 - 4-layer approach



- 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

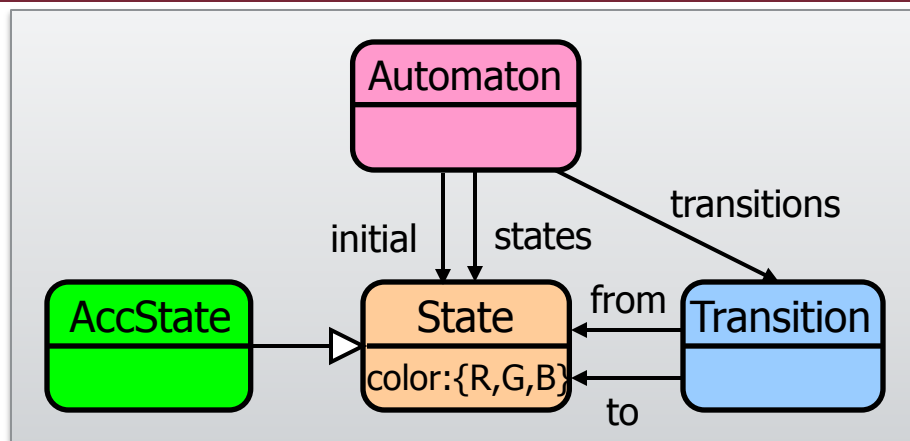
■ Operational

- 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 semantics

■ Denotational (Translational)

- translating concepts in one language to another language (called **semantic domain**)
- „compiled”
- E.g. explaining state machines as Petri-net

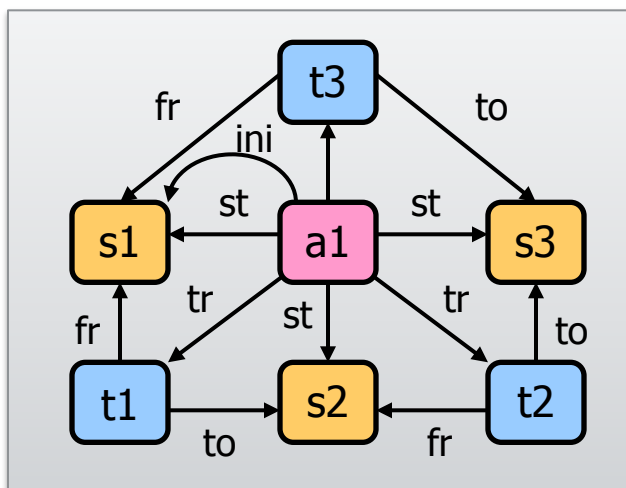
Example: Denotational semantics



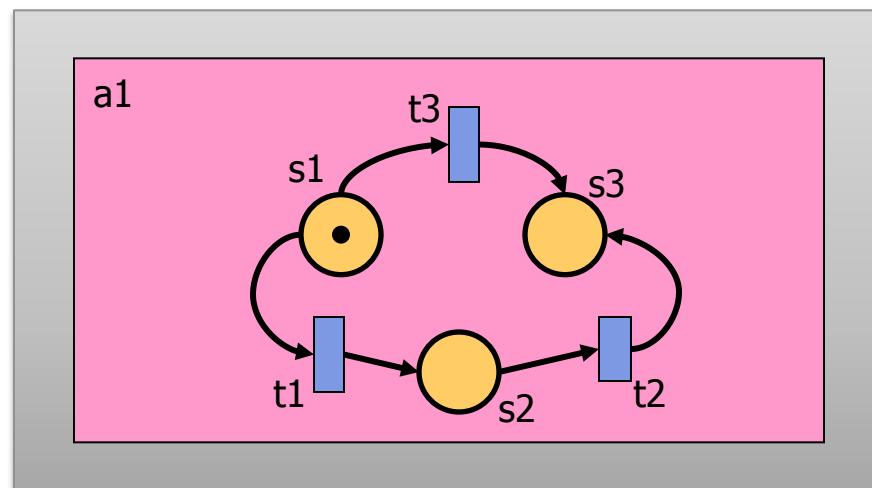
Metamodel

Meta (Language) level

Model level



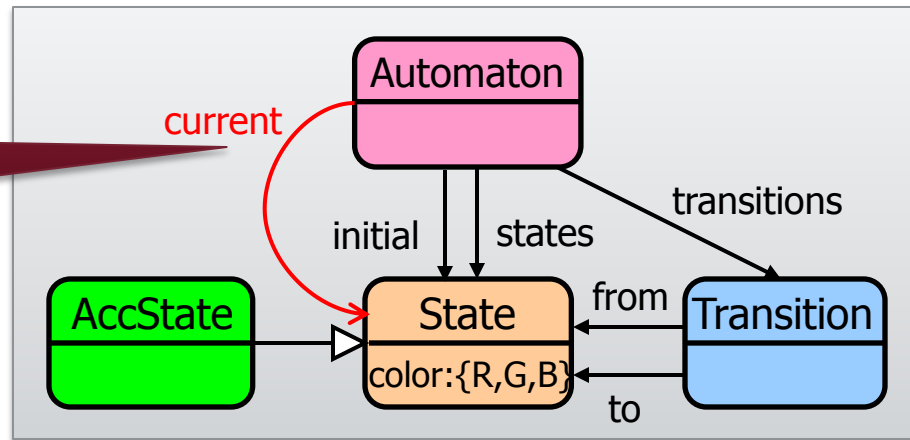
Abstract syntax



Semantic Domain

Example: Operational semantics

Dynamic feature

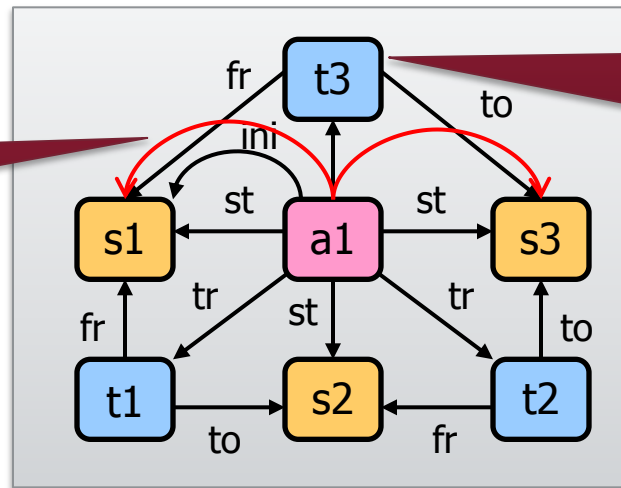


Metamodel

Meta (Language) level

(Instance) Model level

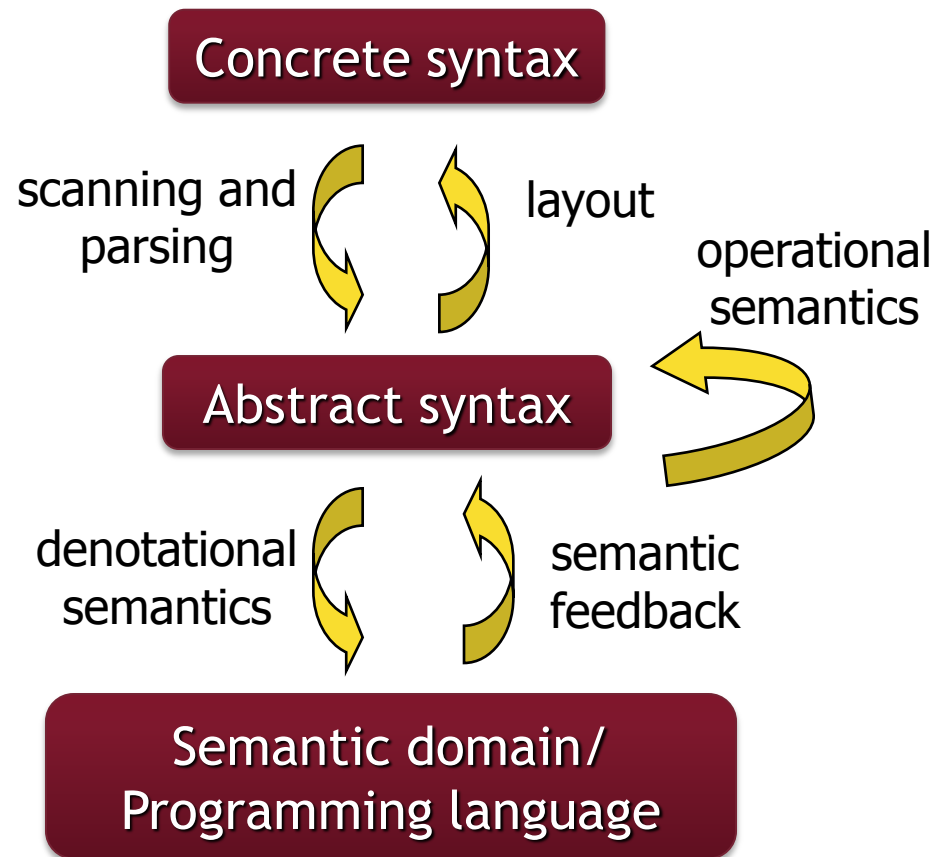
At first, 'current' = 'initial'



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

Technologies

- MATLAB
- Rational Software Architect

COTS

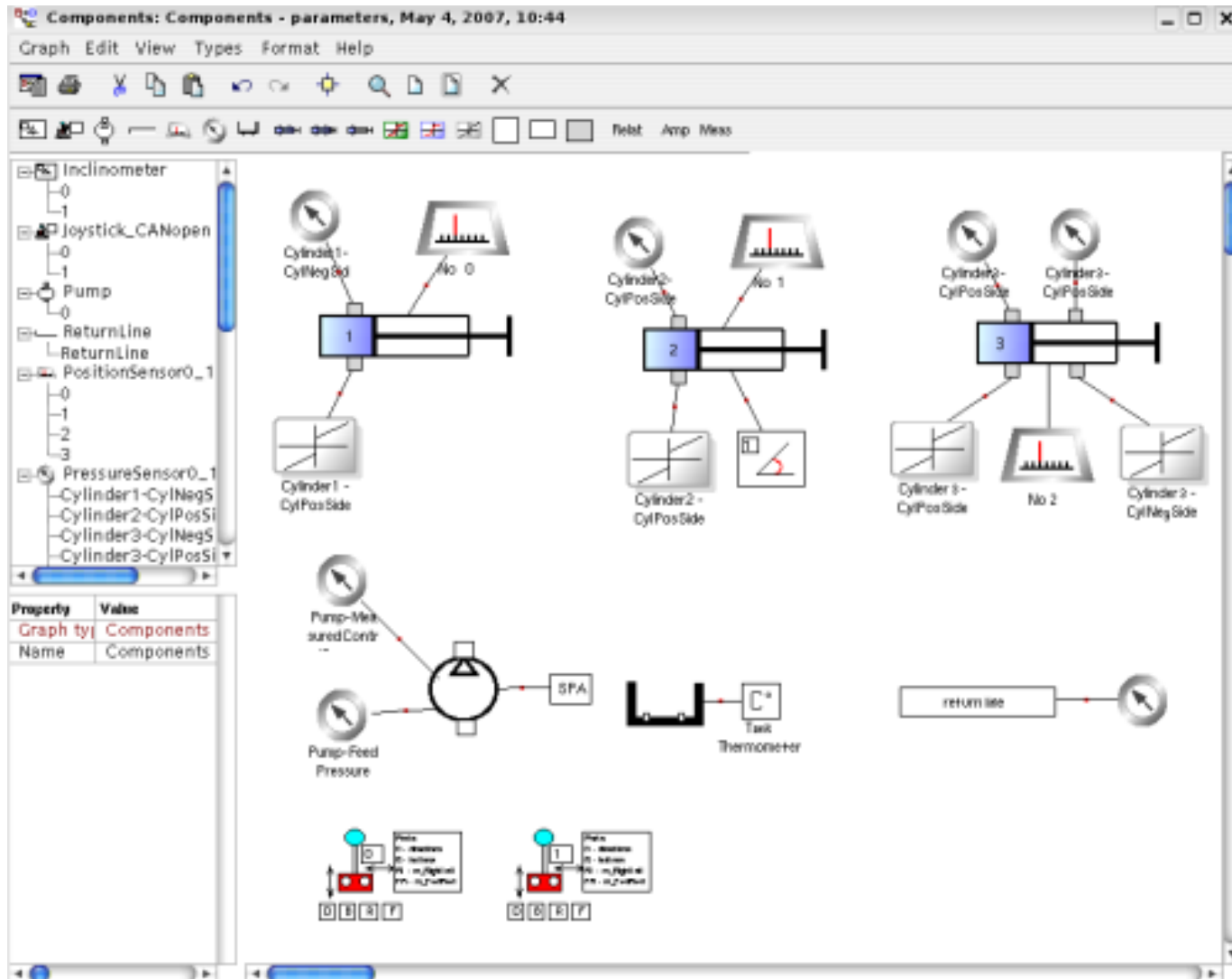
- Eclipse
 - EMF, Sirius
 - Xtext/Xcore/etc.
- Microsoft
 - DSL Tools (Visual Studio) / M / Oslo etc.
- MetaCase
 - MetaEdit+
- JetBrains MPS

Language
engineering
(industry)

- GEMS, GME, ViatraDSM

Academia

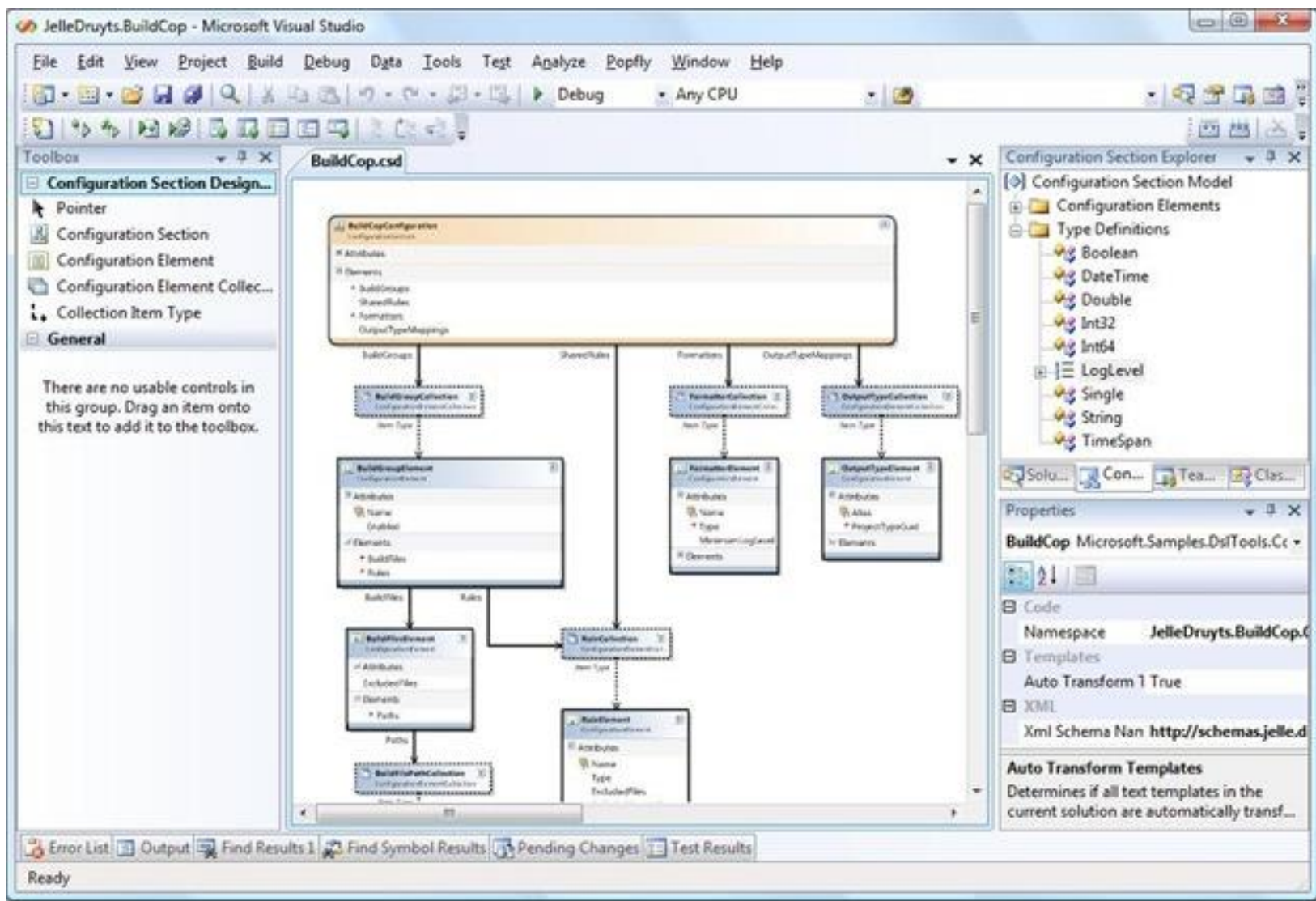
MetaEdit+



Eclipse GMF

The screenshot displays the Eclipse IDE interface for a project named "Apollo". The Package Explorer on the left shows a project structure with a "src" folder containing a "bingo" package and a "bingo.player" package. The main editor window shows the "model.auml_diagram" file, which contains a Java class named "BINGO". The class has several attributes: "DEBUG:boolean=false", "controlPaneTitle:String", "statusPaneTitle:String", and "windowName:String". The "main()" method is highlighted, and a tooltip is displayed over the "getLocalHost()" call, showing the method signature: "public static InetAddress getLocalHost() throws UnknownHostException". The diagram view on the right shows a UML class diagram with several classes and their relationships. The Outline view on the far right shows a list of classes: BEFC, CHECK, COUNT, GAME, PLAY, and WAIT. The bottom of the IDE shows the Problems, Javadoc, and Declaration views.

Microsoft DSL Tools



MPS

The screenshot shows the MPS IDE interface. The main editor displays a rule definition for `typeof_InputFieldReference`. The rule is applicable for the concept `InputFieldReference` and overrides `false`. The rule body contains a `typeof` operation that is currently being edited, with a dropdown menu showing suggestions for the type.

```
rule typeof_InputFieldReference {
  applicable for concept = InputFieldReference as inputFieldReference
  overrides false

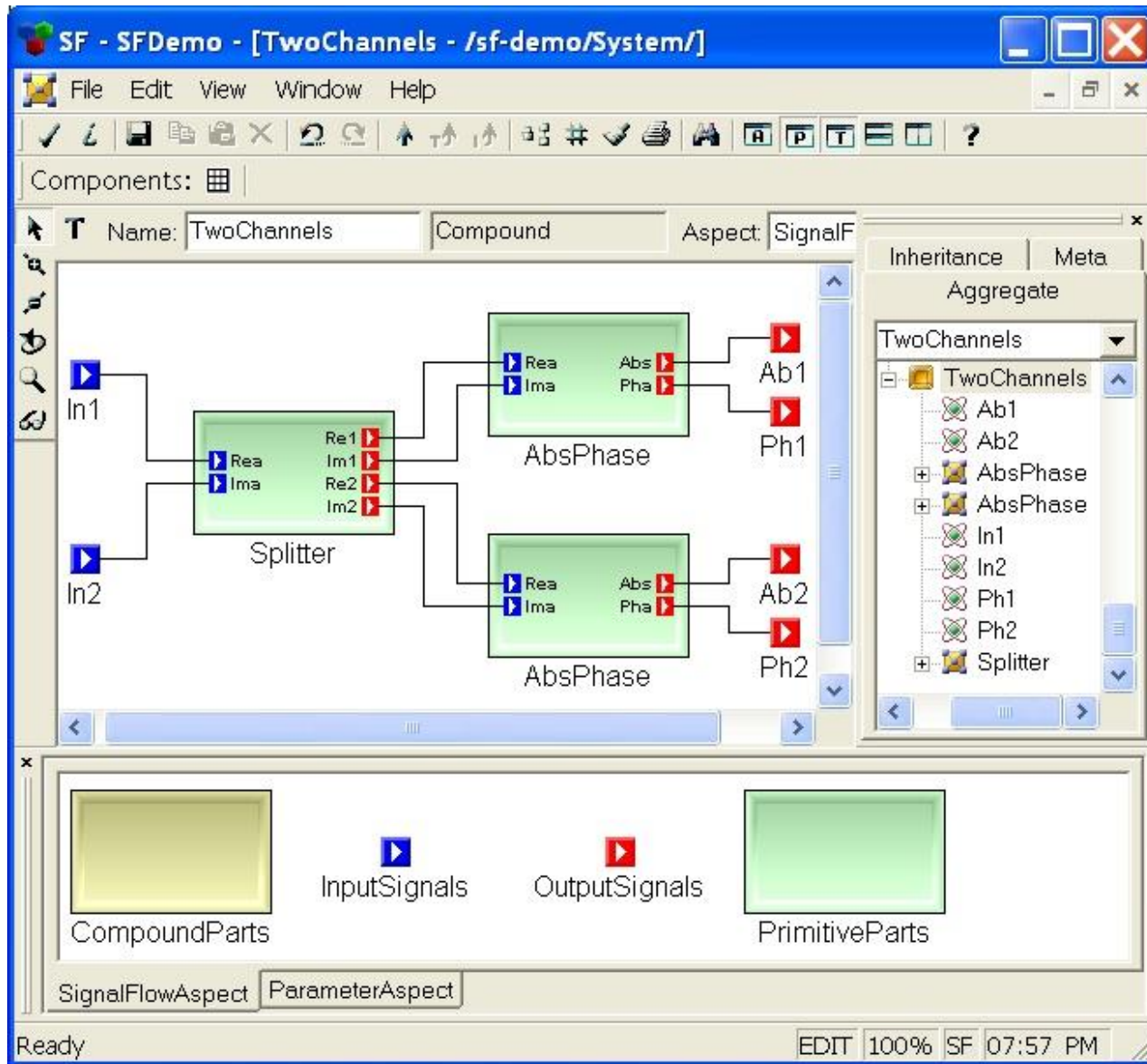
  do {
    typeof(inputFieldReference) ::= <IntegerType
  }
}
```

The dropdown menu lists the following suggestions:

| | |
|-----------------------------------|------------------------------------|
| 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 |

The IDE interface includes a menu bar (File, Edit, Search, View, Go To, Generate, Build, Run, Tools, Version Control, Window, Help), a toolbar, a project browser on the left, a hierarchy view on the right, and a bottom toolbar with tabs for Structure, Editor, Constraints, Behavior, Typesystem, Actions, Refactorings, Intentions, Find Usages, Data Flow, Generator, Textgen, and a status bar at the bottom showing memory usage (302M of 498M).

GME



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