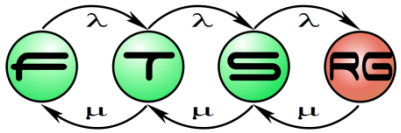
Metamodeling and Domain-specific modeling

Horváth Ákos
Bergmann Gábor
Dániel Varró
István Ráth



Agenda

- Metamodeling
  - UML profiles
  - Domain-specific modeling
  - Metalevels
  - Semantics
- Examples from engineering practice
- XMI: document design with metamodels

METAMODELING

Why?

- Let's do Model-based Development!
- Create **model**s that…
  - have well-defined, standardized form and meaning
  - are processable by computers
    - Storage, Parsing, Editing, Visualization,
    - Execution, Testing,
    - Analysis, Verification,
    - Translation, Transformation, Integration, Synchronization
  - are easy to use (create / understand)
- Need to design **modeling language**s

Where do you find metamodels?

- Different application domains around UML
  - SysML (systems engineering)
  - SPEM (process modeling)
  - CWM (data warehousing)
  - MARTE (real-time and embedded systems)
- BPEL, BPMN (business processes)
- Tropos (requirements modeling)
- AutoSAR (automotive industry)
- …

Designing modeling languages

Designing modeling languages

- Core concept: **metamodeling**
  - Design methodology of modeling languages
  - Metamodel = model of a modeling language

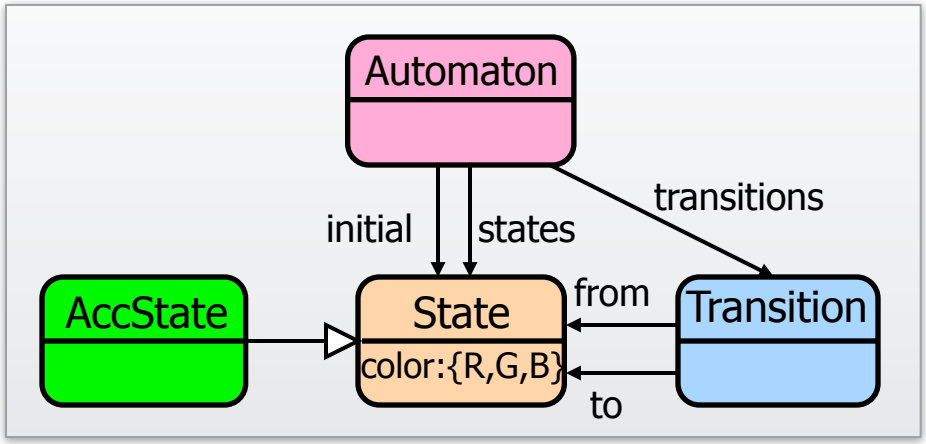
Designing modeling languages

- Core concept: **metamodeling**
  - Design methodology of modeling languages
  - Metamodel = model of a modeling language
- 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
  - ??? (something is missing… we'll come back later)

Abstract syntax (Metamodel)

- Metamodel = model of a modeling language
  - „Meta” = above, beyond, transcending
- Goal: to define
  - The vocabulary of concepts in the language
  - How they can be combined to form models
- Contents:
  - Definition of concepts
  - Relationships between these concepts
    - Abstraction/Specialization (Taxonomy)
  - Constraints, well-formedness rules (e.g. multiplicity)

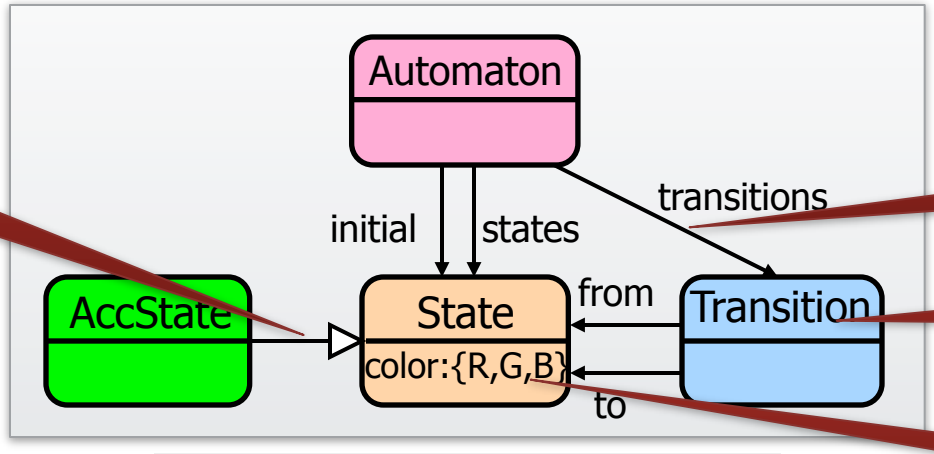
Example



Metamodel

Meta (Language) level

Example



Generalization

Association

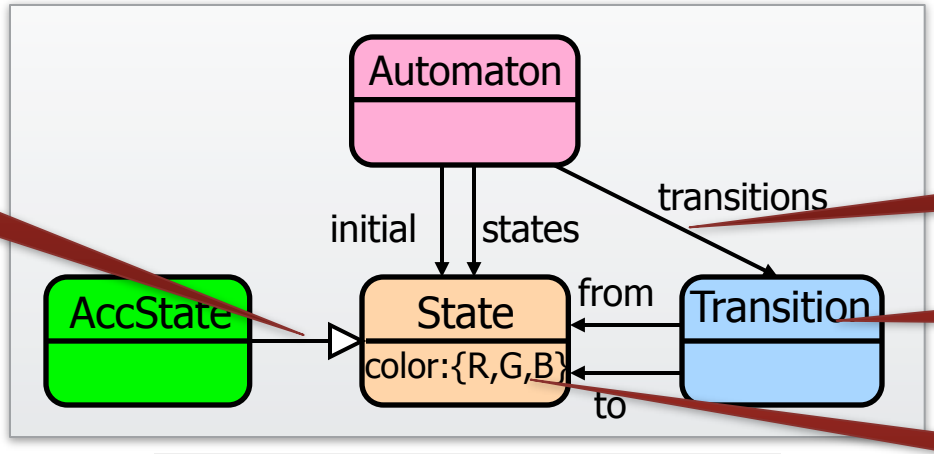
Class

Attribute

Metamodel

Meta (Language) level

Example



Generalization

Instantiation

Association

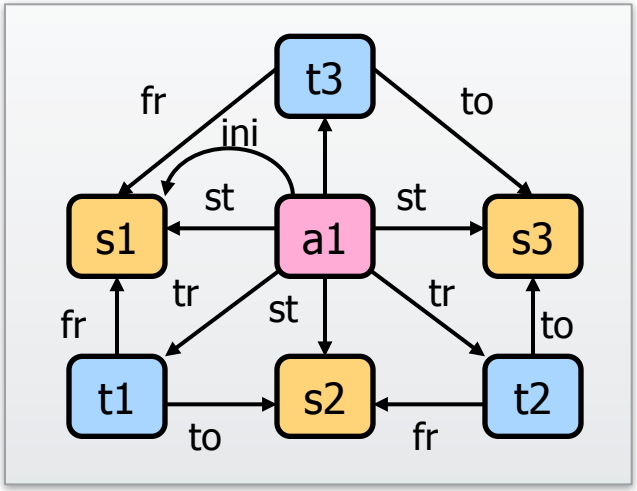
Class

Attribute

Metamodel

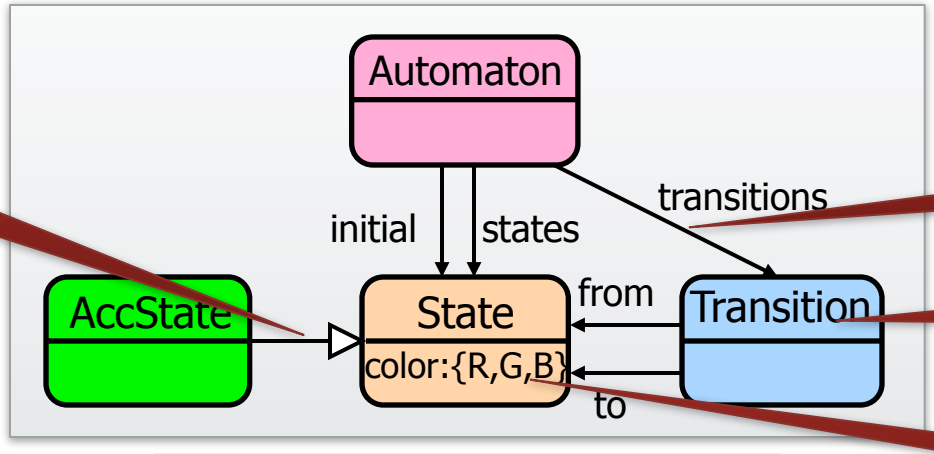
Meta (Language) level

(Instance) Model level



Model in abstract syntax

Example



Generalization

Association

Instantiation

Class

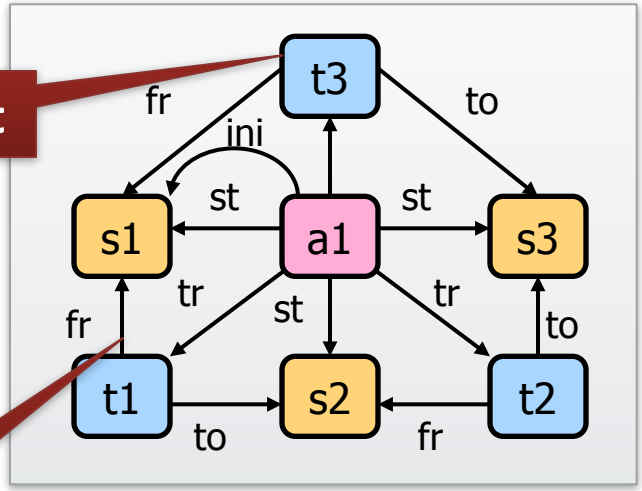
Attribute

Metamodel

Meta (Language) level

(Instance) Model level

Object



Link

Model in abstract syntax

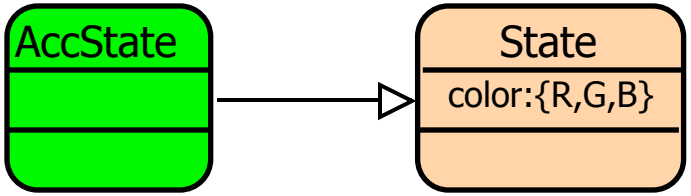
Well-formedness rules

- Multiplicity constraints
  - At most one: 0..1 / Many: *
  - Lower bound is often meaningful (enforcement?)
- Aggregation/Containment
  - At most one parent for each model element
- Language specific constraints:
  - Examples
    - Each state of an automaton must have a unique name
    - Transitions must connect states of their own automaton
    - The initial state is one of the states of the automaton
  - Expressed in e.g. OCL

Instantiation and Generalization

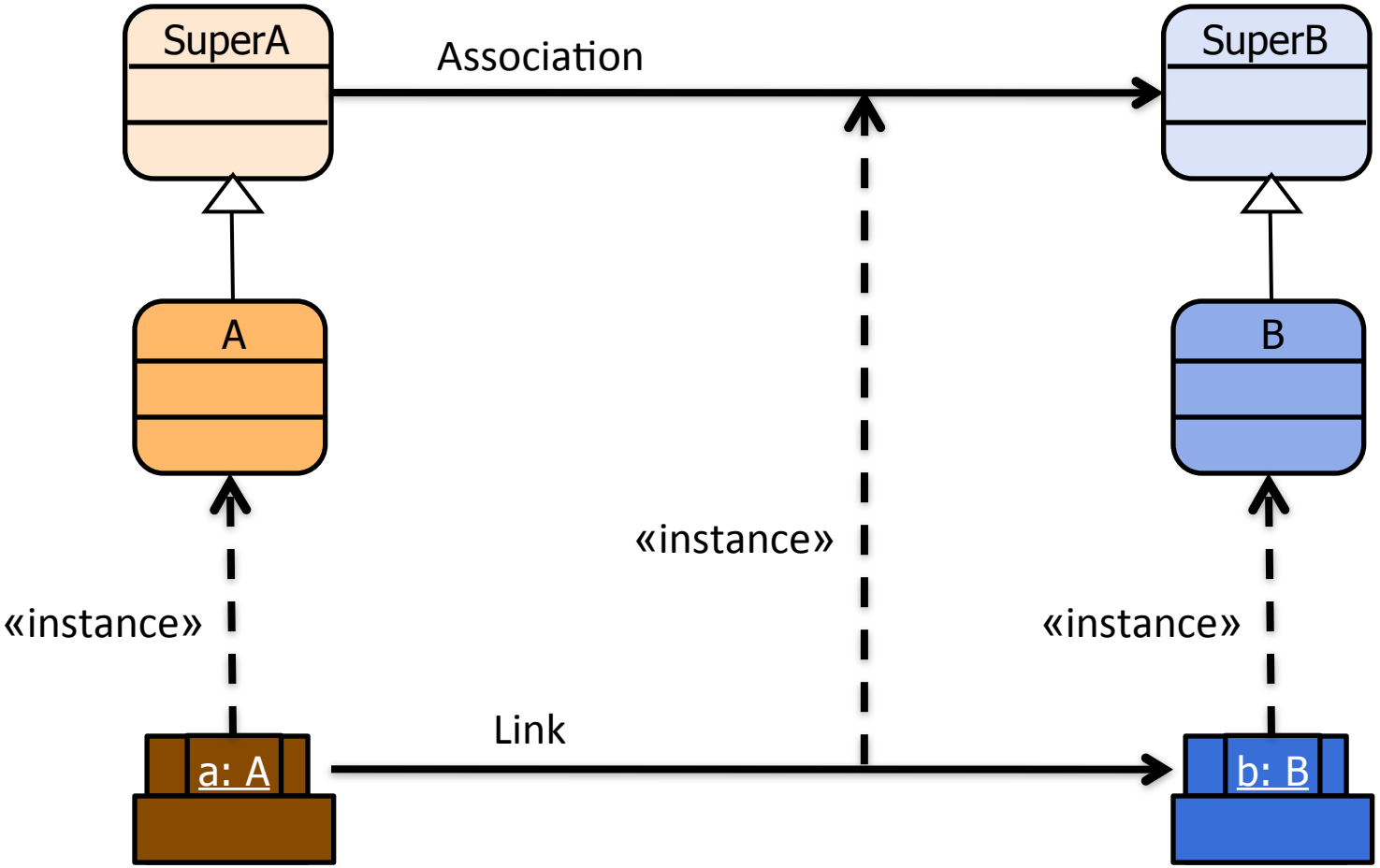
- Classification/Typing
  - inverse: Instantiation

- Generalization / Supertyping / Abstraction
  - inverse: Specialization / Subtyping / Refinement



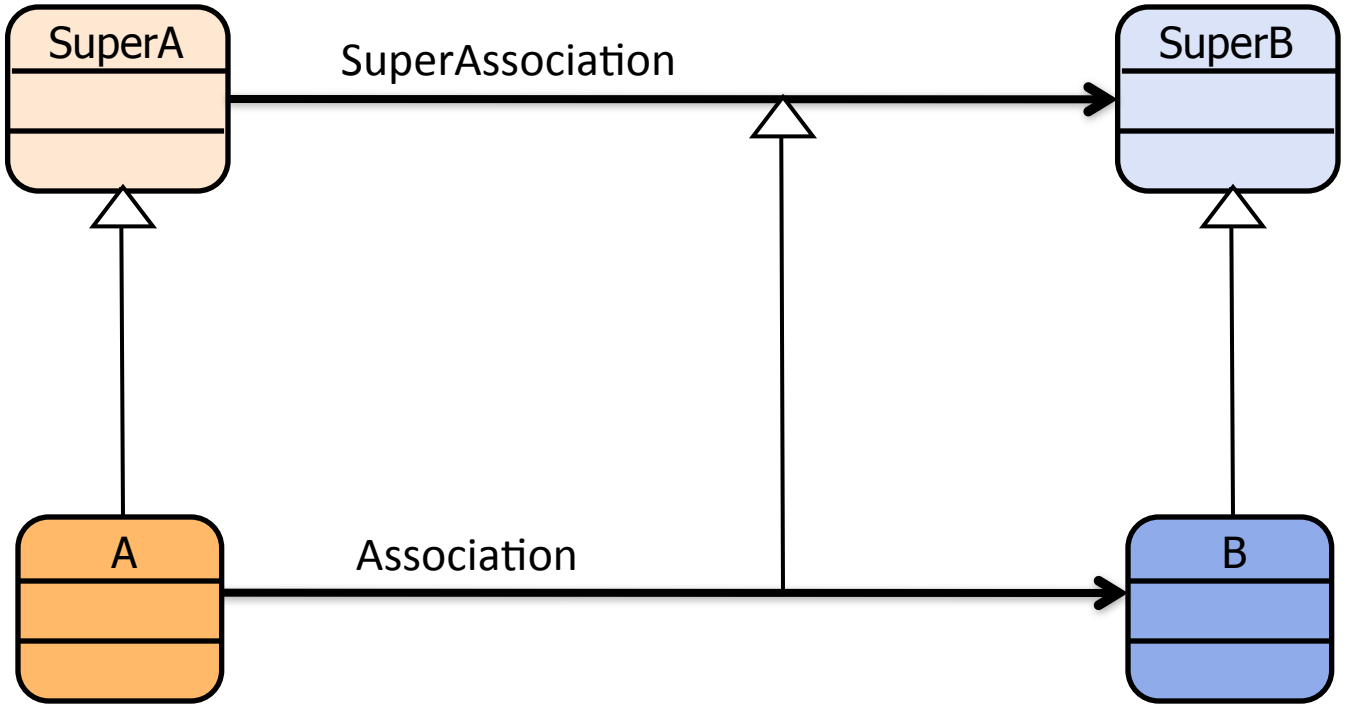
- More is implied than what is explicitly given
  - transitive semantics
    - **self**.supertypes→**includesAll**(**self**.supertypes.supertypes)
  - extends the typing relation
    - **self**. instances->**includesAll**(**self**. subtypes.instances)

Type Conformance of Edges



Type Conformance of Edges

- Subtyping of edges
  - Not allowed in e.g. UML

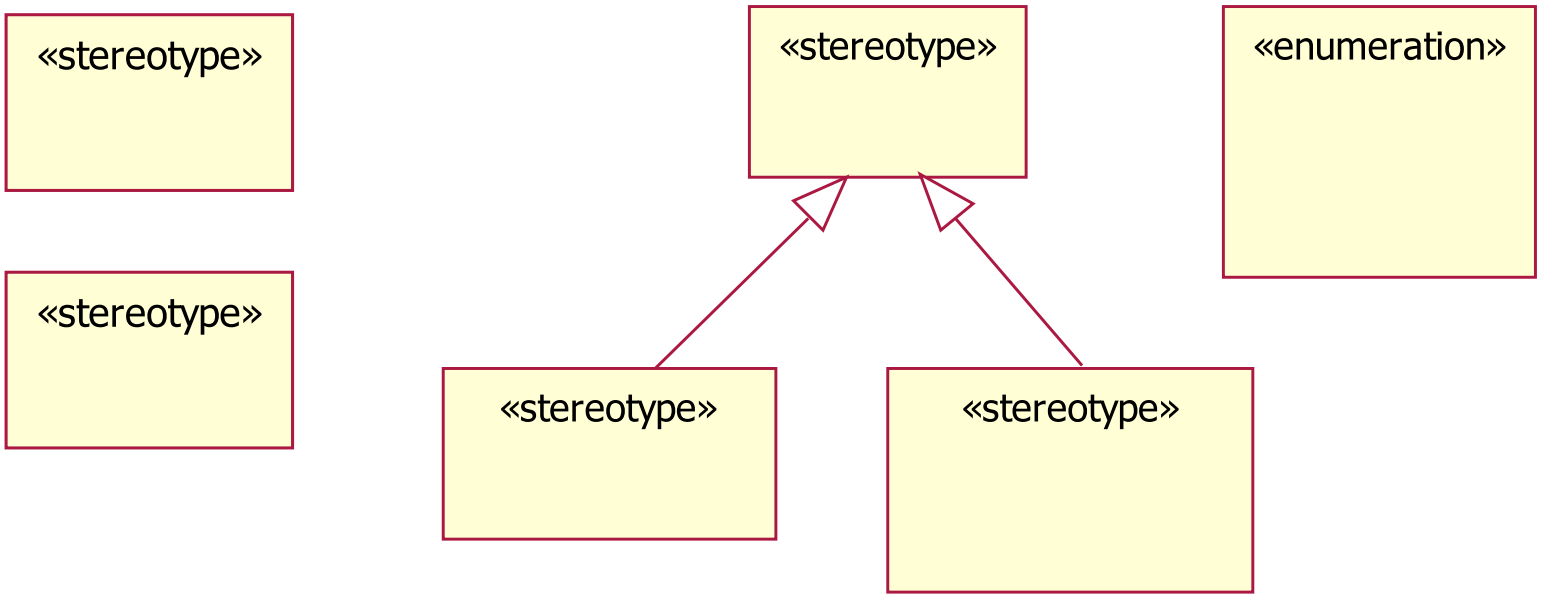


UML PROFILES

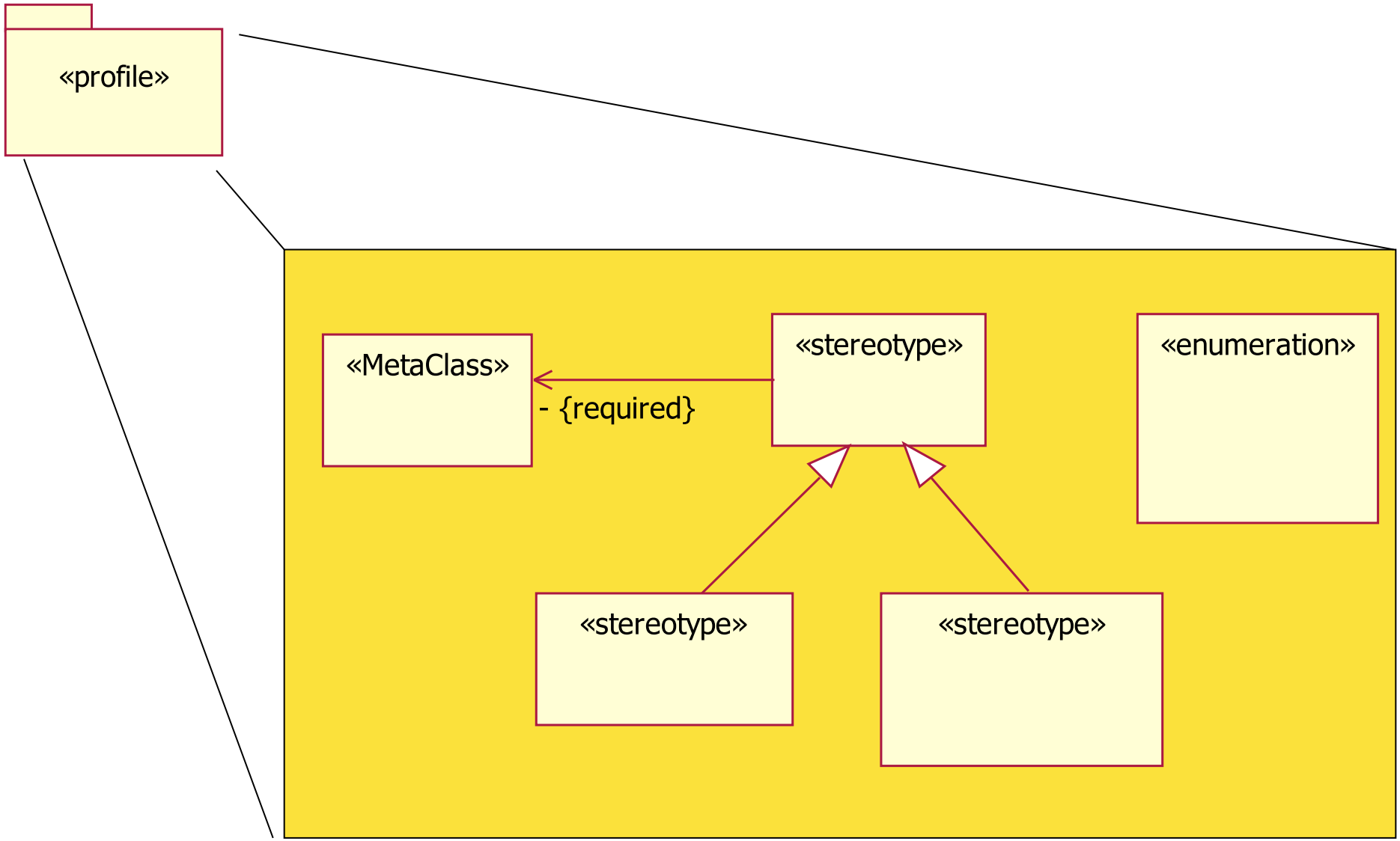
Goals of UML Profiles

- **Goal**: Extend the core UML metamodel by tailoring it to the problem domain
- **Process**:
  - Define stereotypes for domain constructs
  - Map stereotypes to the UML language
  - Apply stereotypes to a UML model
- **Limitation**:
  only extension of the metamodel
  - No modification
  - No new language constructs

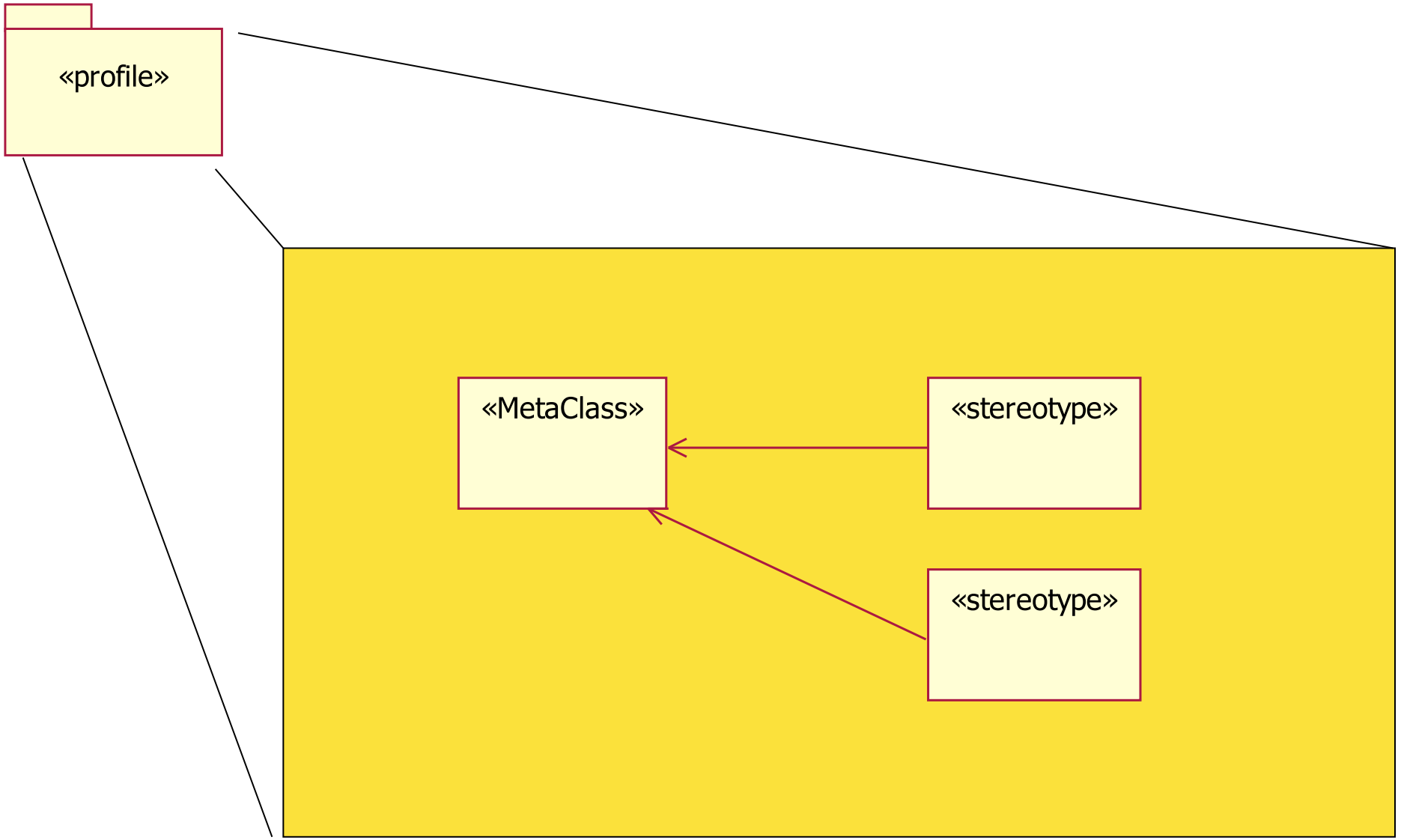
Example: Defining EJB Stereotypes



Mapping the EJB Profile to UML

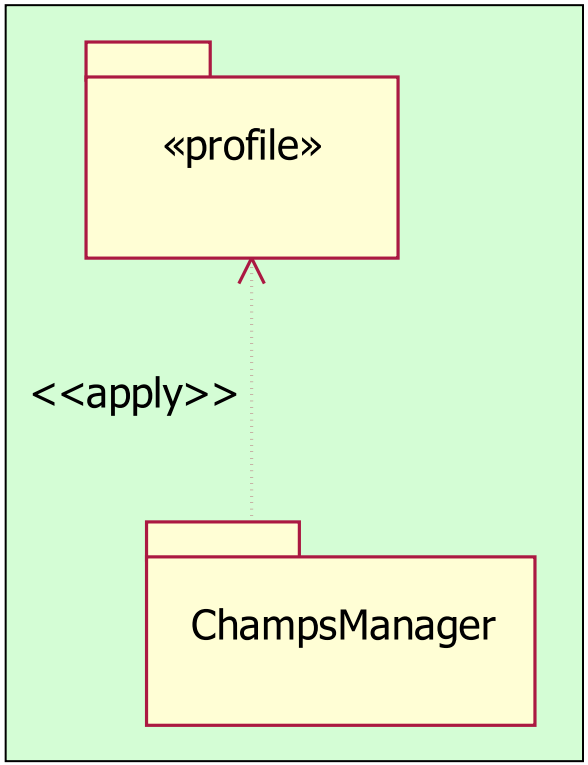


Mapping the EJB Profile to UML



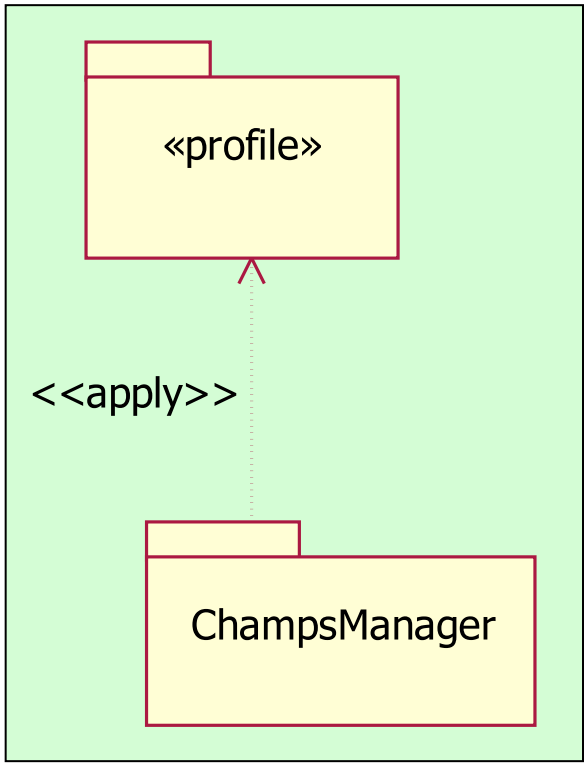
Applying a Profile in a UML model

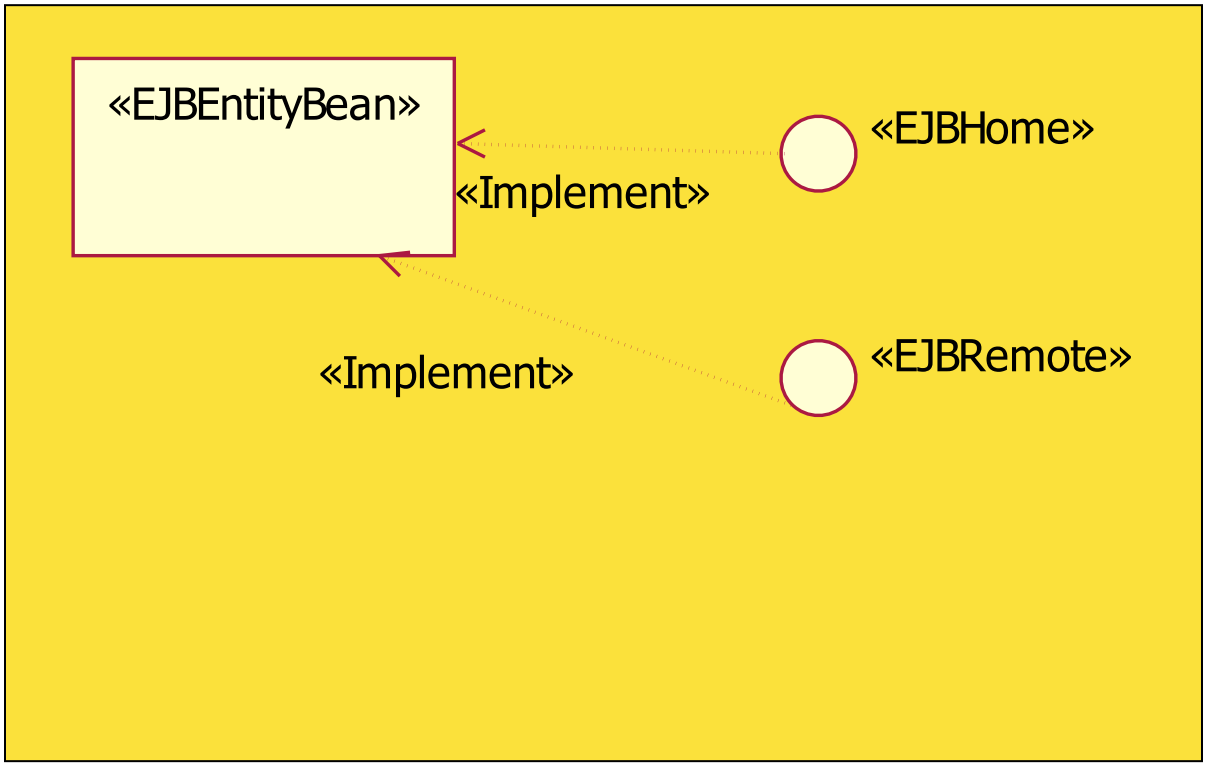
Applying a Profile in a UML model



- **Declare** that a profile is to be used in a UML model

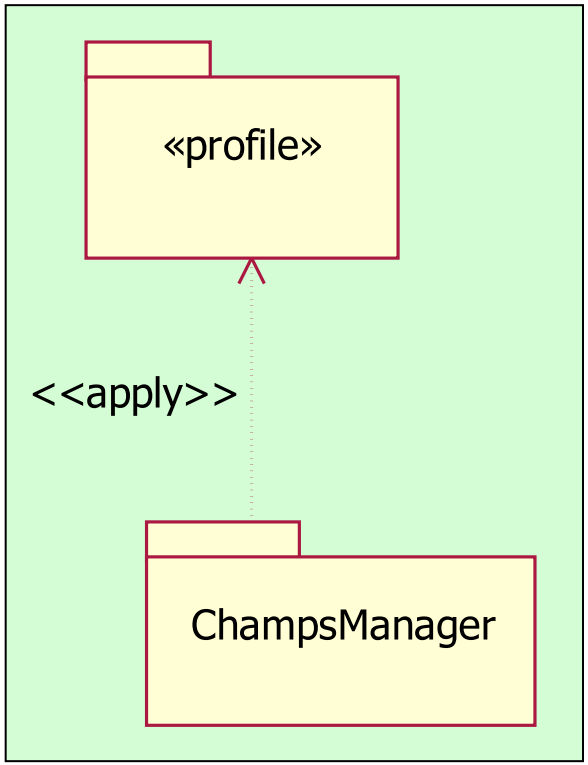
Applying a Profile in a UML model

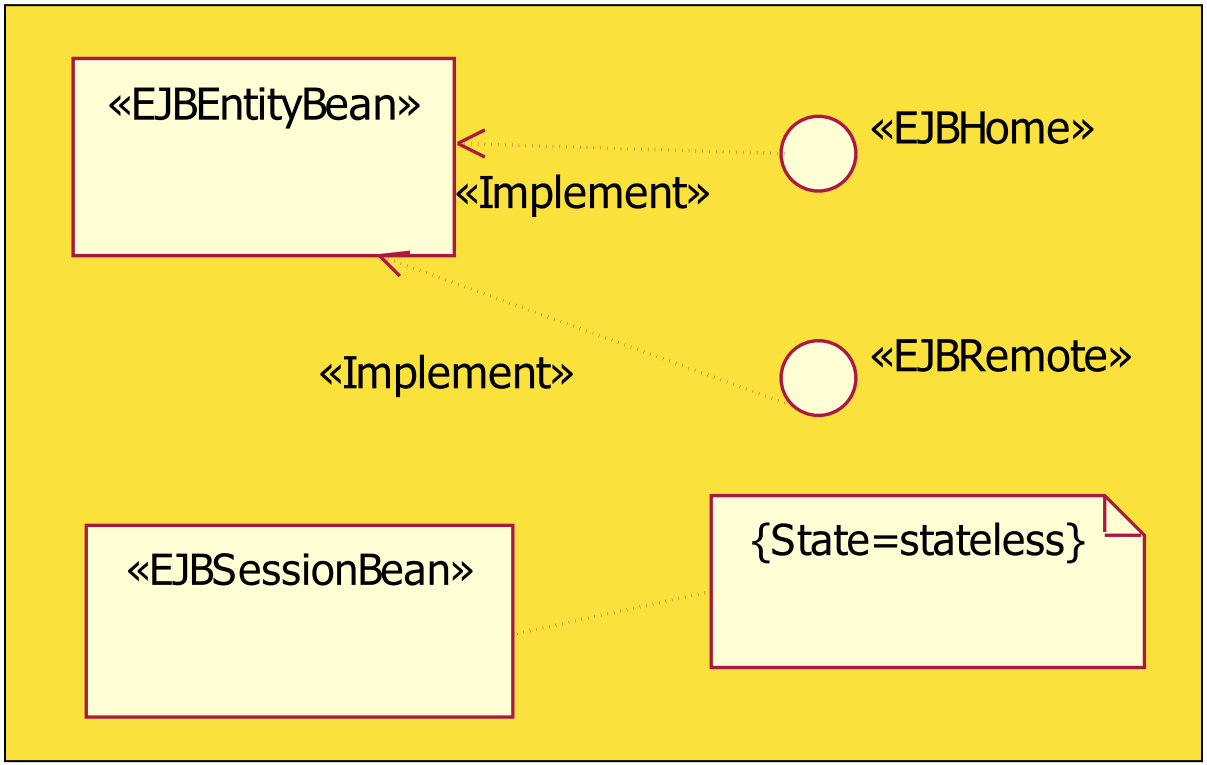




- **Declare** that a profile is to be used in a UML model
- **Apply the profile** on appropriate UML elements

Applying a Profile in a UML model





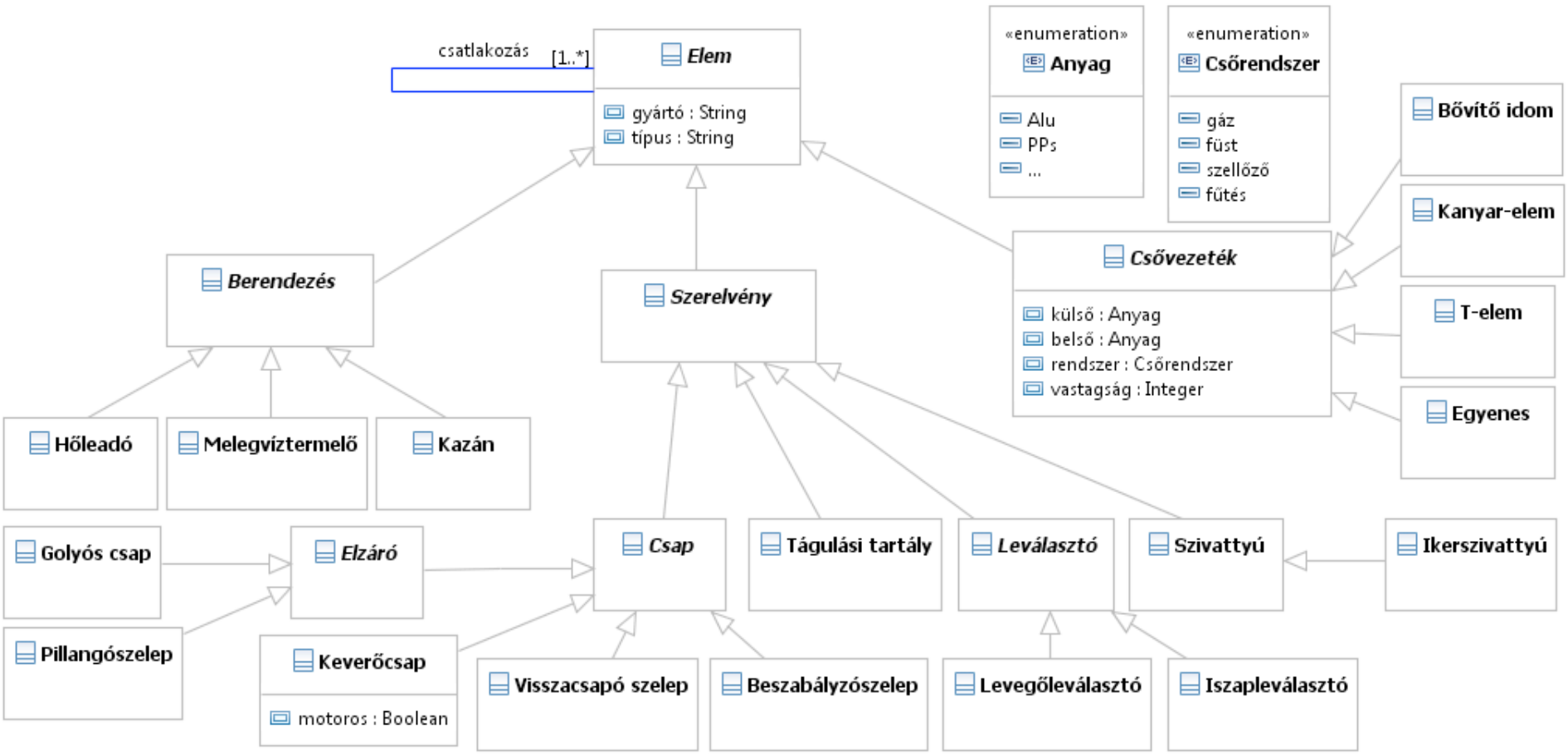
- **Declare** that a profile is to be used in a UML model
- **Apply the profile** on appropriate UML elements
- Use **tagged values** for attributes defined in stereotypes

Evaluation of UML

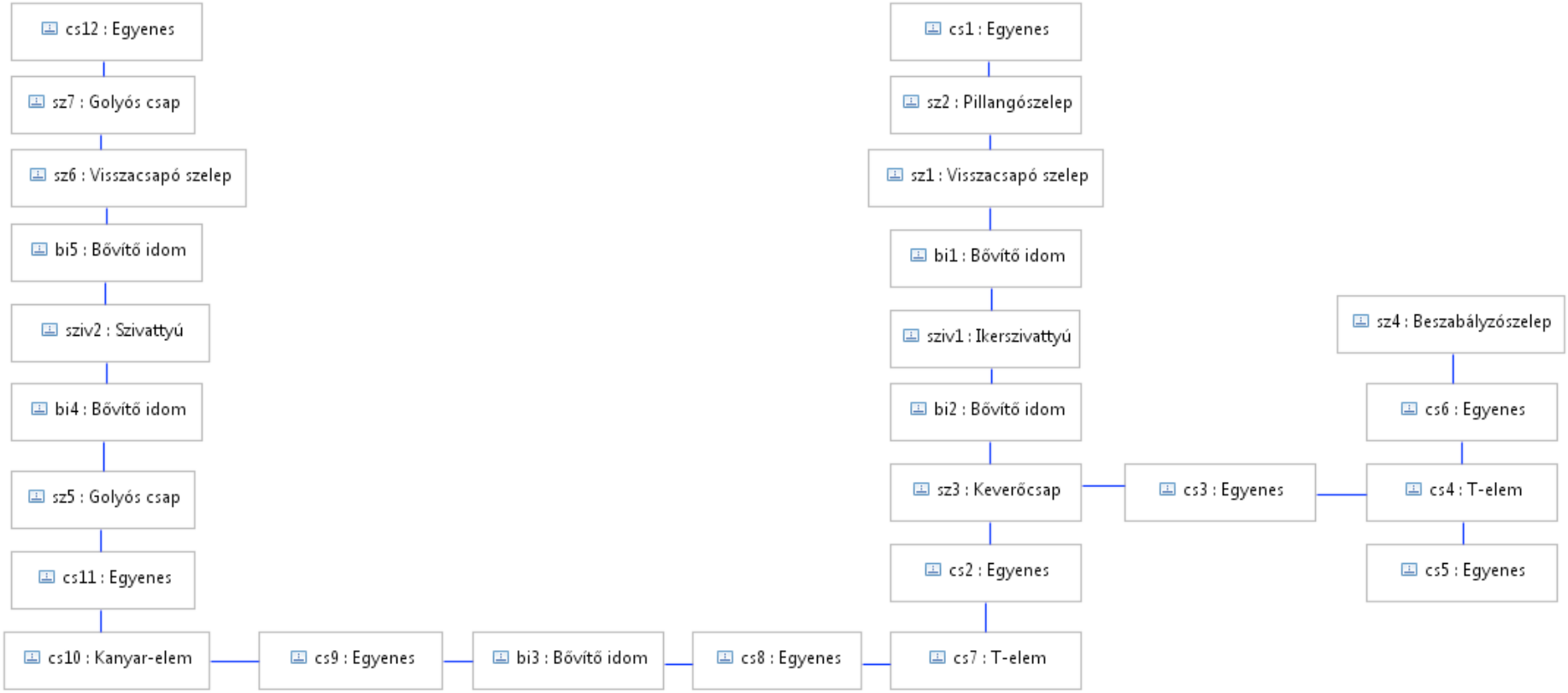
- Advantages:
  - Standard common language
  - Visual notation
- Disadvantages:
  - Imprecise semantics
    - No single, integrated solution for all UML artifacts
  - Limited scope and flexibility
    - UML Profiles (Stereotypes)
    - For software engineers, by software engineers
    - Unified (NOT Universal) Modeling Language
    - Bloat

DOMAIN SPECIFIC MODELING

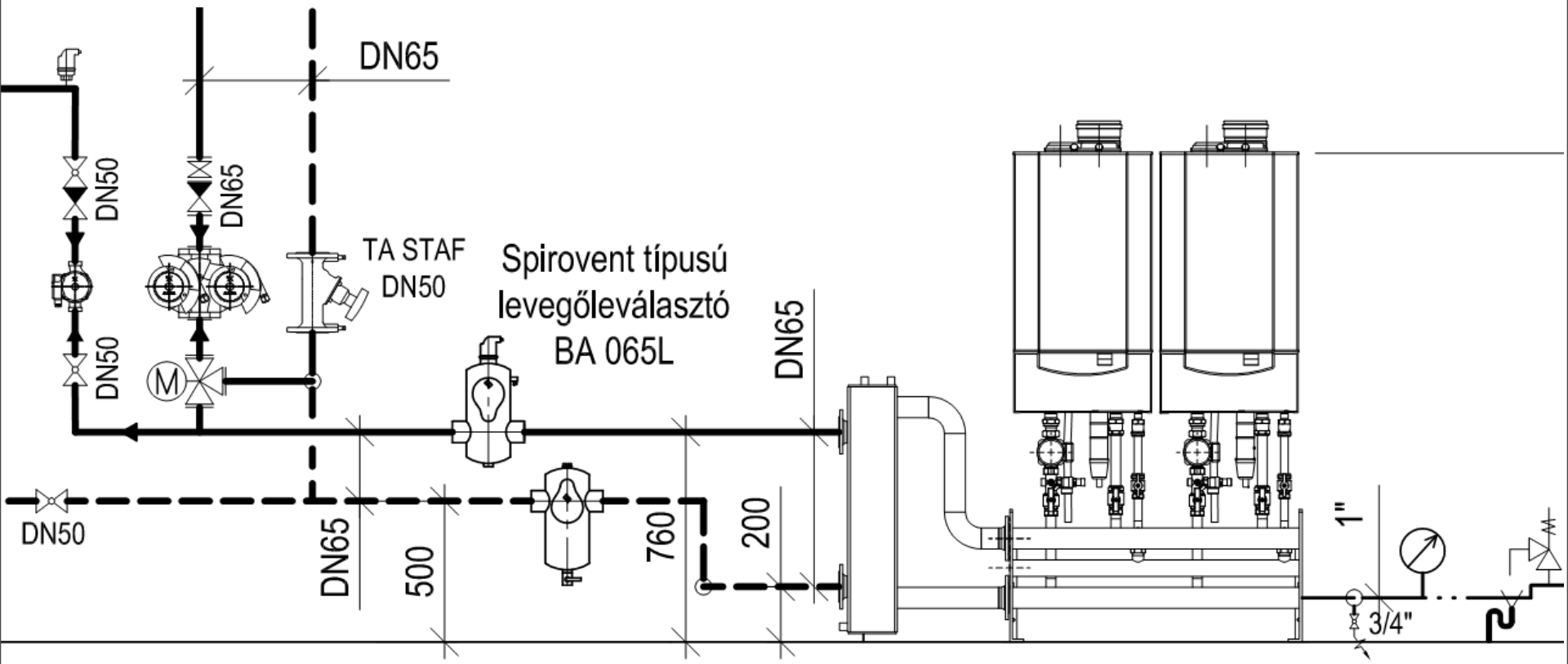
Example metamodel



Instance model, abstract syntax



Instance model, concrete syntax

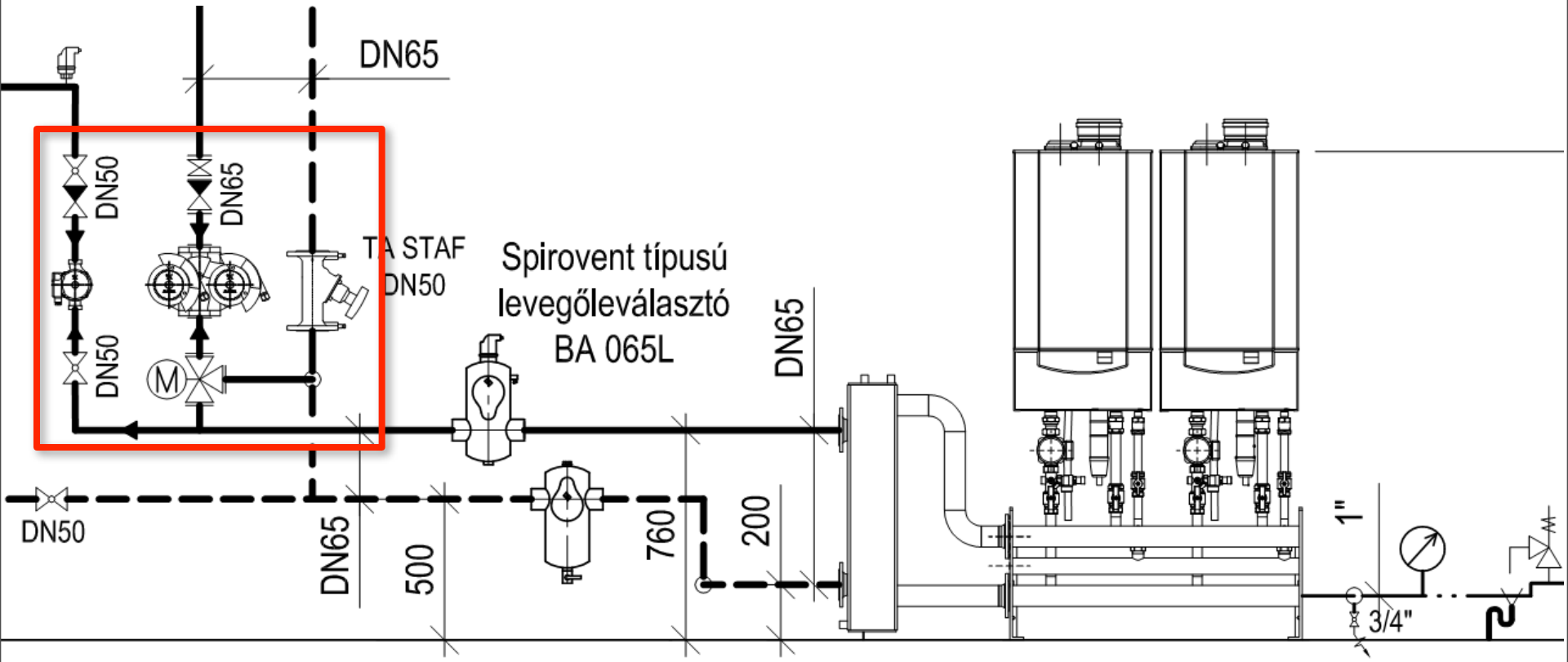


Honeywell keverőcsap DN50 K_{VS} 40

Spirovent típusú iszapleválasztó BE 065L

Remeha Quinta kaszkád rendszer hidraulikus váltóval

Instance model, concrete syntax



Honeywell keverőcsap DN50 K_{VS} 40

Spirovent típusú iszapleválasztó BE 065L

Remeha Quinta kaszkád rendszer hidraulikus váltóval

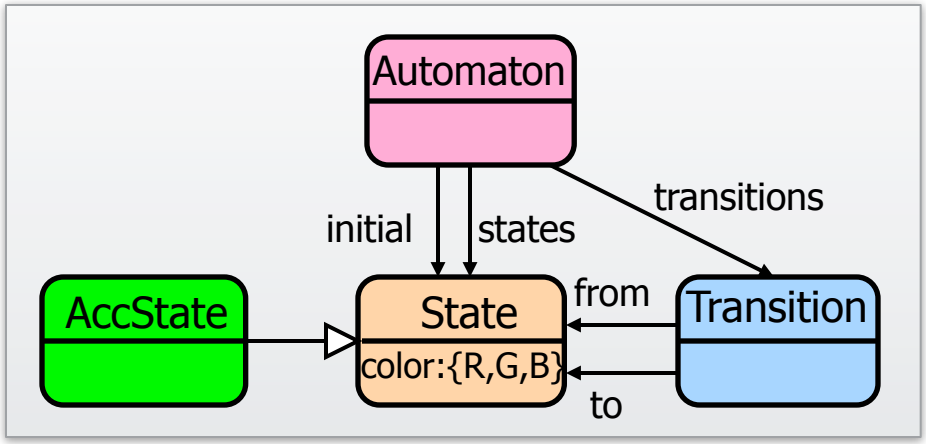
Designing modeling languages

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
  - ??? (something is missing… we'll come back later)
  - **Concrete syntax**
    - Textual notation
    - Visual notation

Designing modeling languages

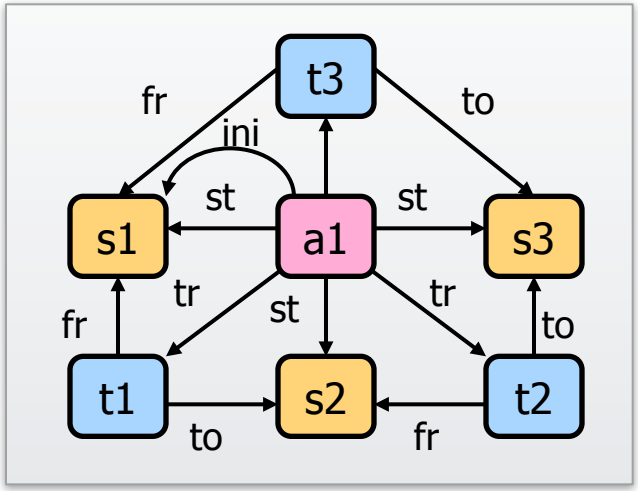
Relationship of concepts



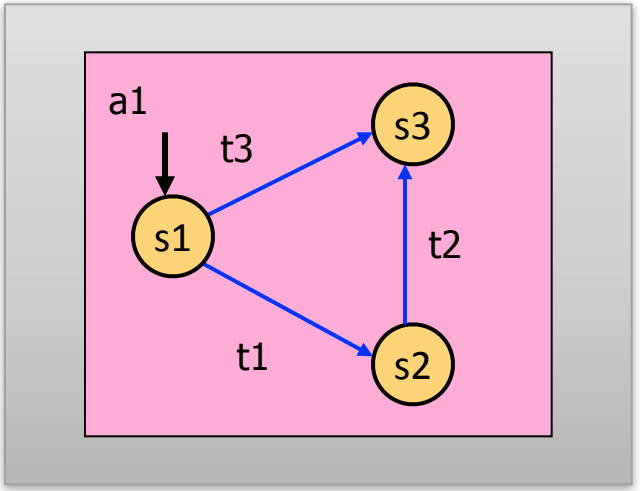
Meta (Language) level

Metamodel

Model level



Abstract syntax

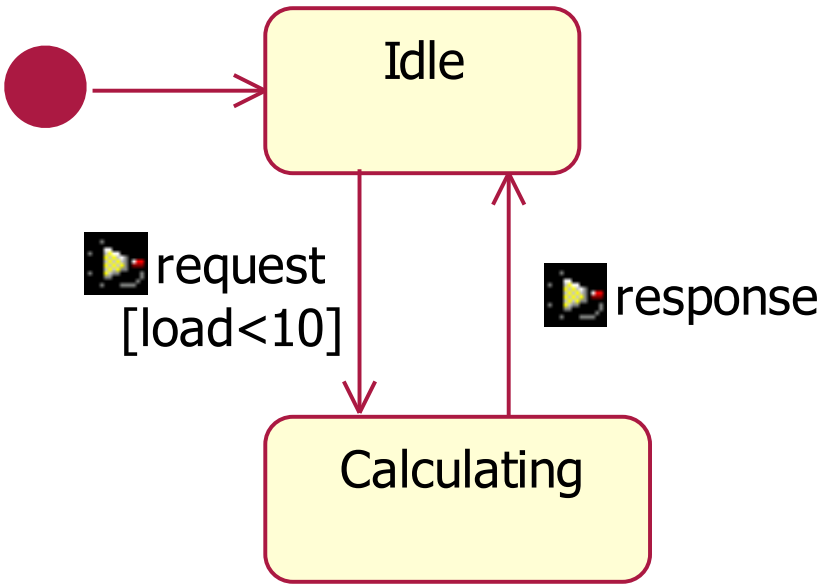


Concrete syntax

Textual vs. Visual

- Textual notation:
  - + Easy to write: Able to capture complex expressions
  - - Difficult to read: Difficult to comprehend and manage after certain complexity
- 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: Concrete Syntax



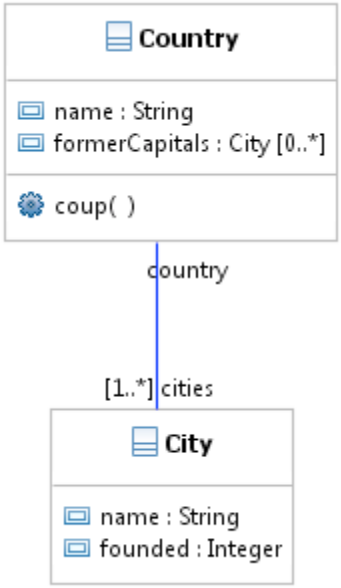

```
request() {
    if (state == "idle" &&
    this.load<10)
    state = "calculating";
}
response() {
    if (state == "calculating")
    state = "idle"
}
```

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..*]
    - 0,1,* <Literal Unlimited Natural> *
    - -1,0, <Literal Integer> 0
  - <Operation> coup ()
- <Class> City
  - <Property> name : String
  - <Property> founded : Integer
- <Association> A_country_cities
  - <Property> country : Country
    - 0,1,* <Literal Unlimited Natural> 1
    - -1,0, <Literal Integer> 1
  - <Property> cities : City [1..*]
    - 0,1,* <Literal Unlimited Natural> *
    - -1,0, <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 value="1" xmi:type="uml:LiteralInteger"
    /ownedEnd>
    ownedEnd name=                                    d-VCP9iY9GYHg
      <upperValue xm                                  tedNatural" x
      <lowerValue xm                                  r" value="1"
    </ownedEnd>
  </packagedElement>
</uml:Package>
```

Abstract Syntax

Graphical notation (Class Diagram)

Textual notation (XMI 2.1)

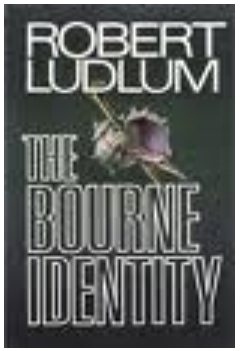
Multiplicity of Notations

- 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

METALEVELS

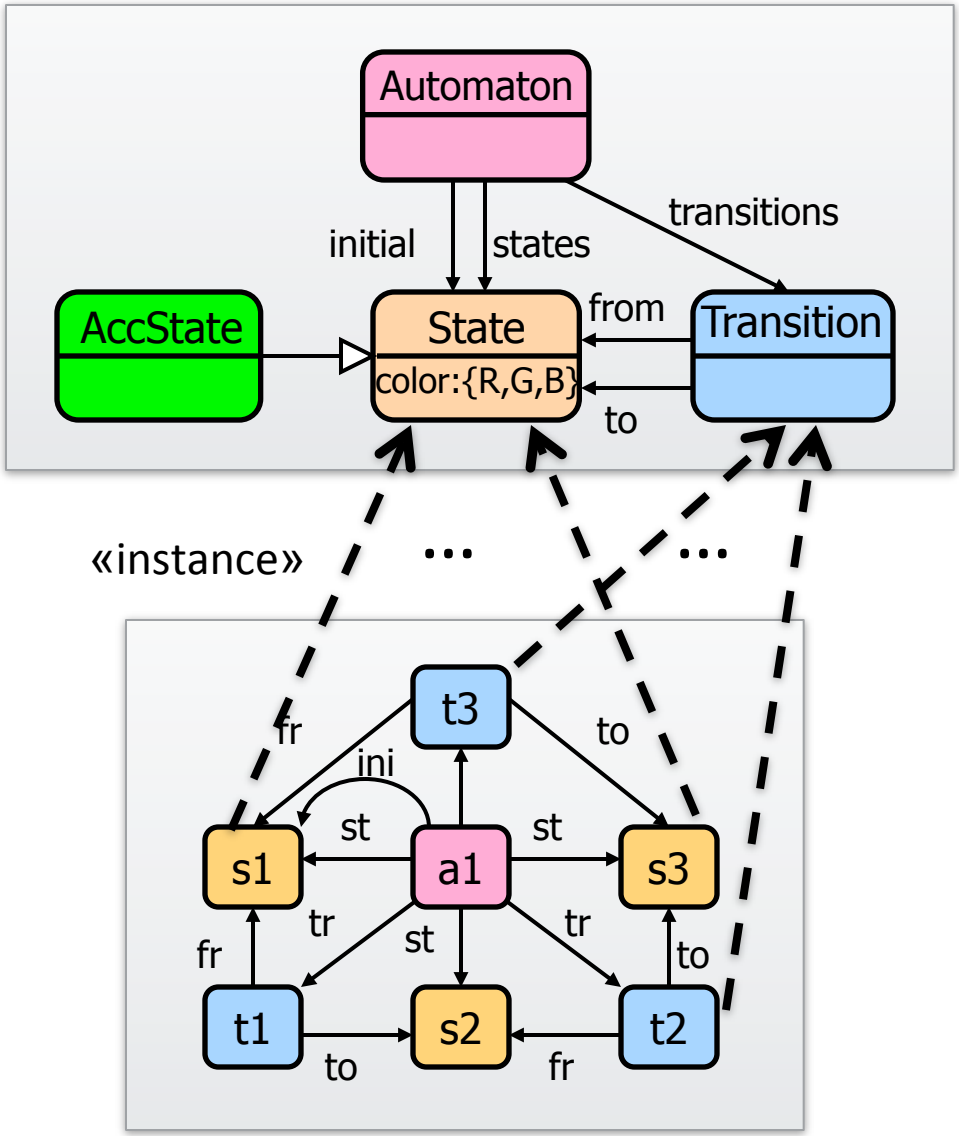
Exercise Metalevels

- 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:

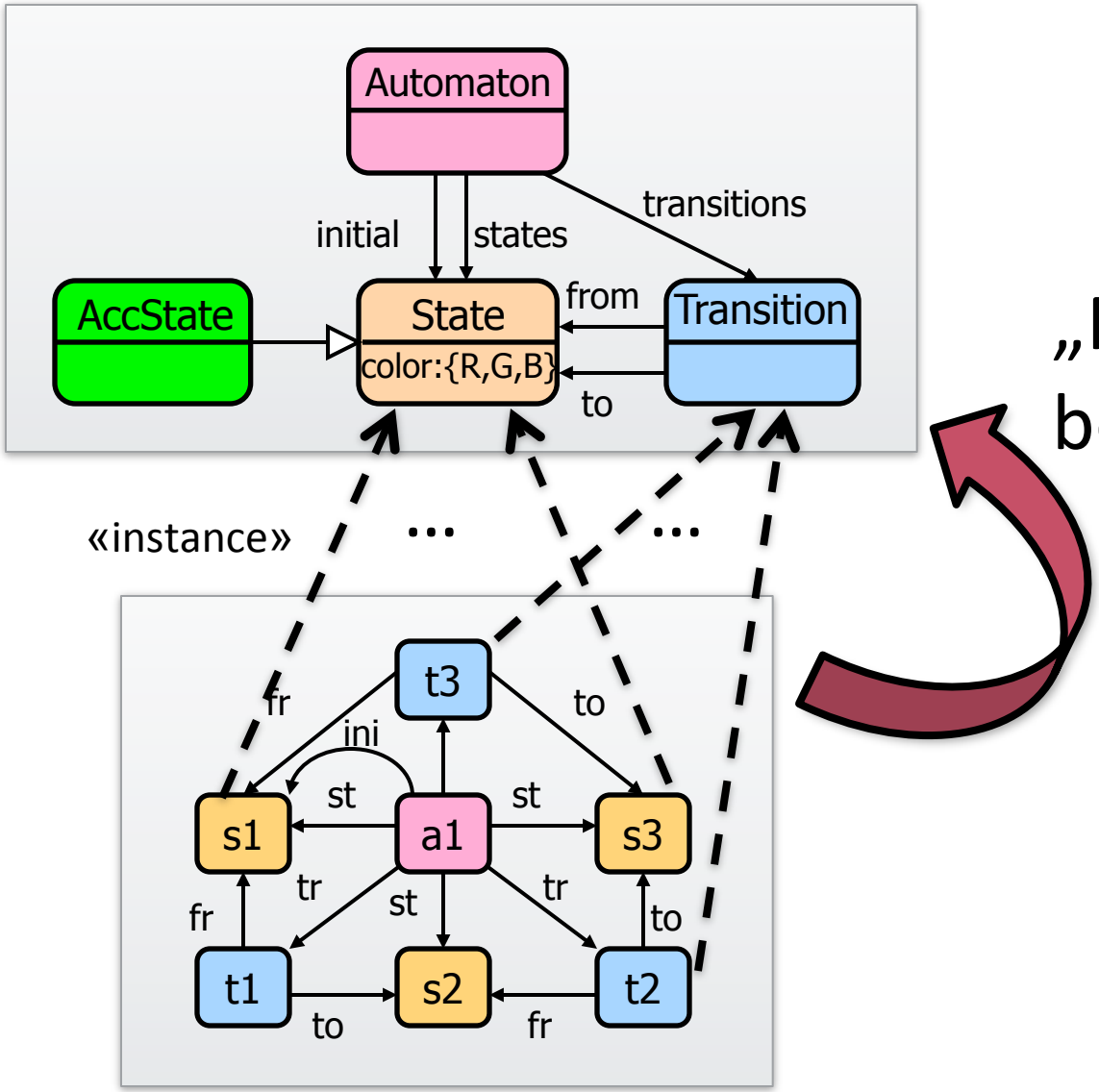


- Edges
  - written by, directed by, creator, subtype, instance

Metalevels

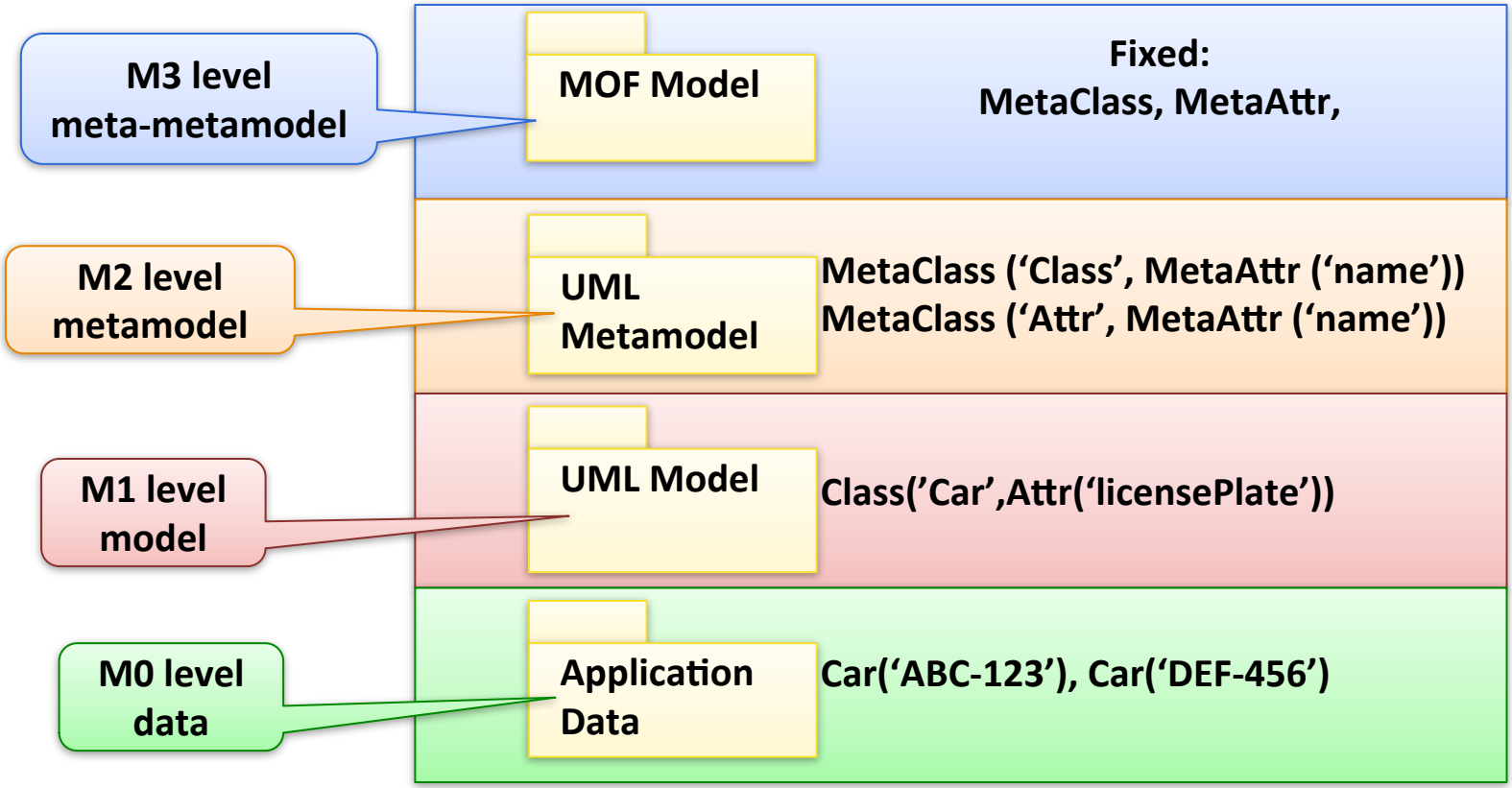

Automaton
initial
states
transitions
AccState
State
color:{R,G,B}
from
to
Transition
«instance»
t3
fr
ini
to
st
st
s1
a1
s3
tr
st
tr
fr
to
t1
s2
t2
to
fr

Metalevels

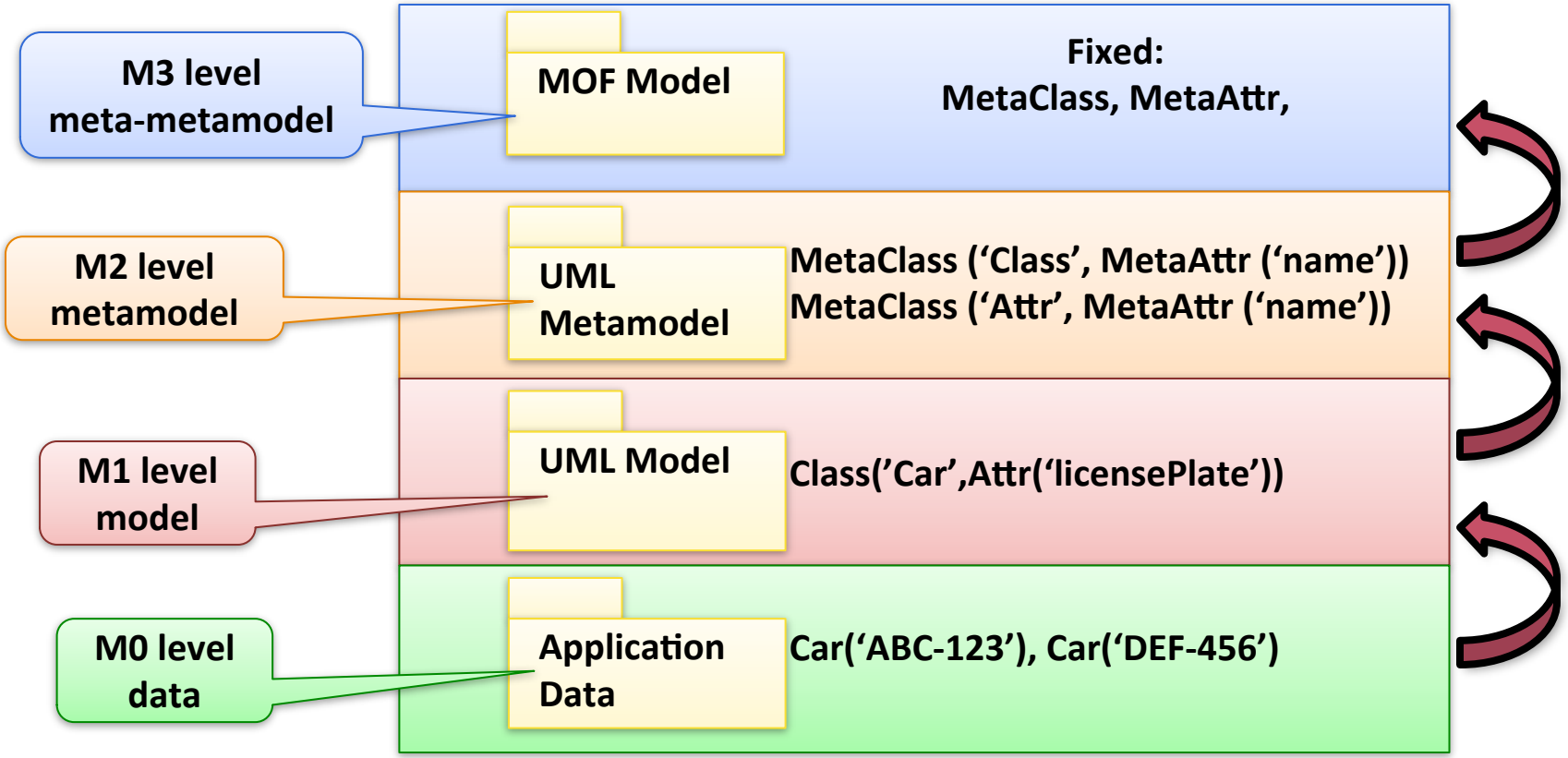


„Meta" relationship between models

Metalevels in MOF

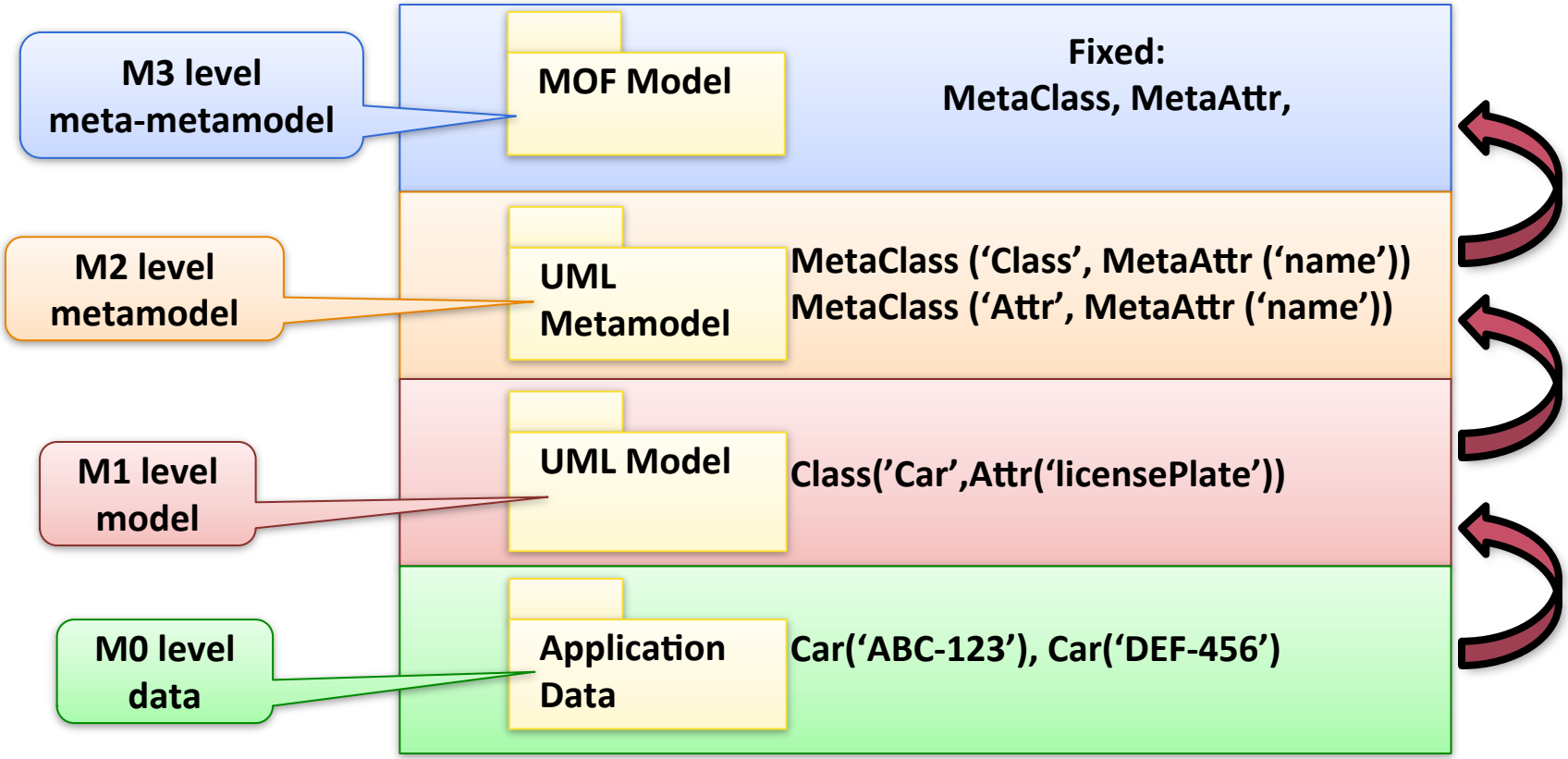


Metalevels in MOF



Metalevels in MOF

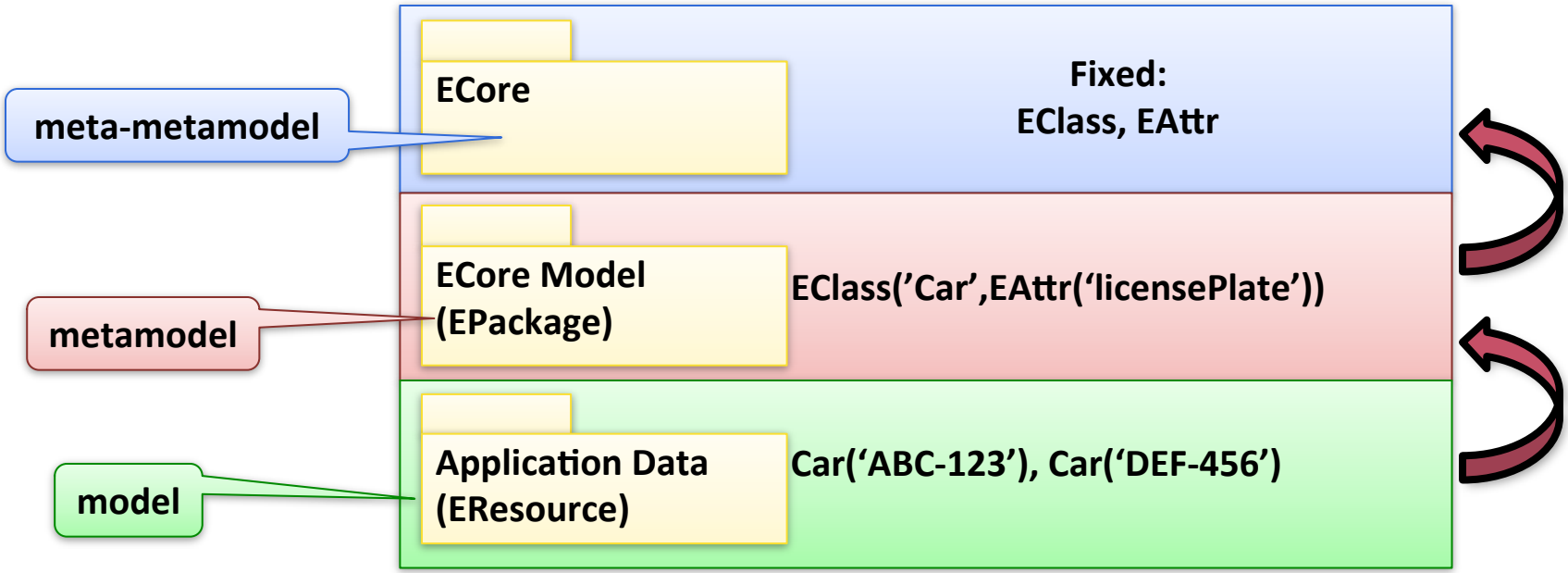
- OMG's MOF (Meta Object Facility)
  - 4-layer approach



  - Why exactly four levels?

Metalevels in other approaches

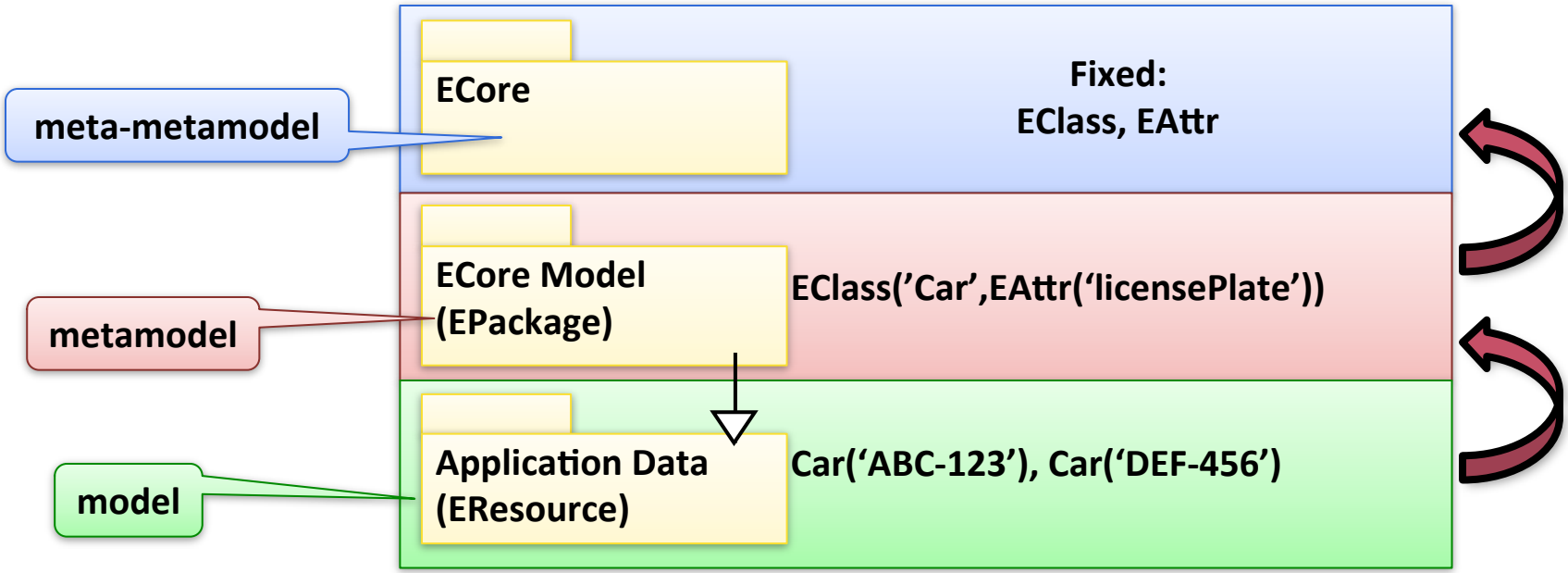
- EMF (Eclipse Modeling Framework)



- Multi-level metamodeling
  - VPM
  - Ontologies

Metalevels in other approaches

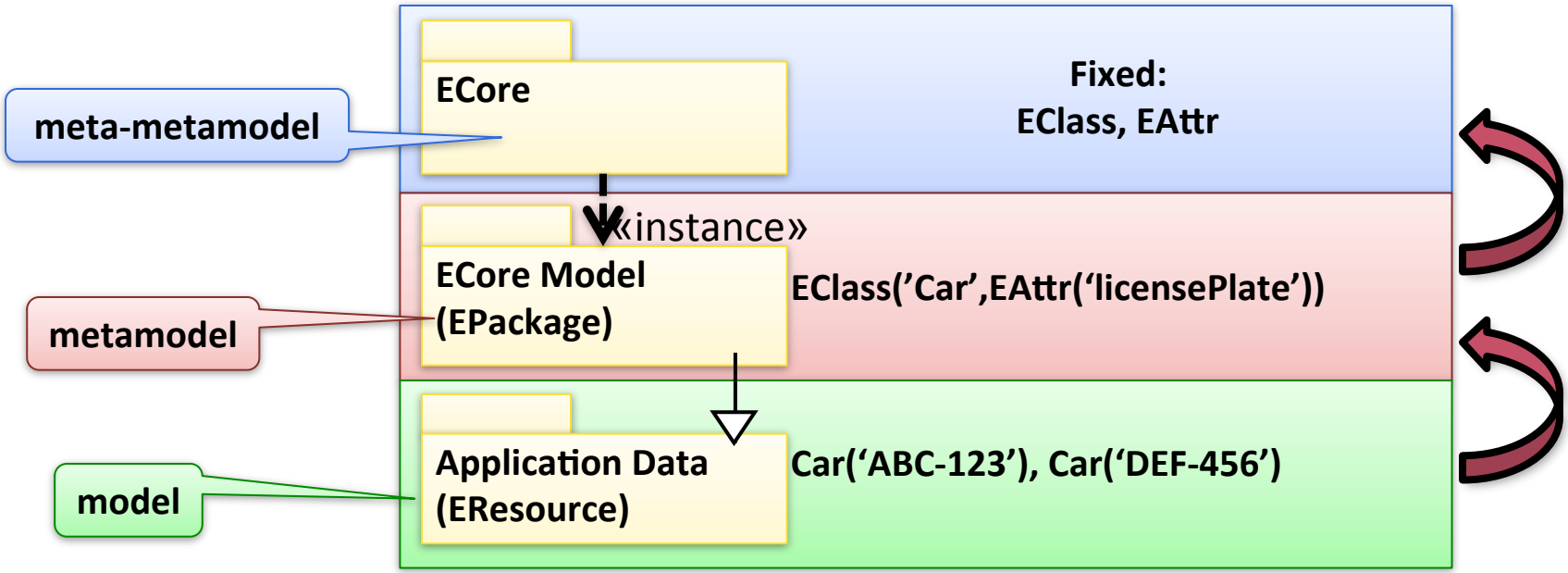
- EMF (Eclipse Modeling Framework)



- Multi-level metamodeling
  - VPM
  - Ontologies

Metalevels in other approaches

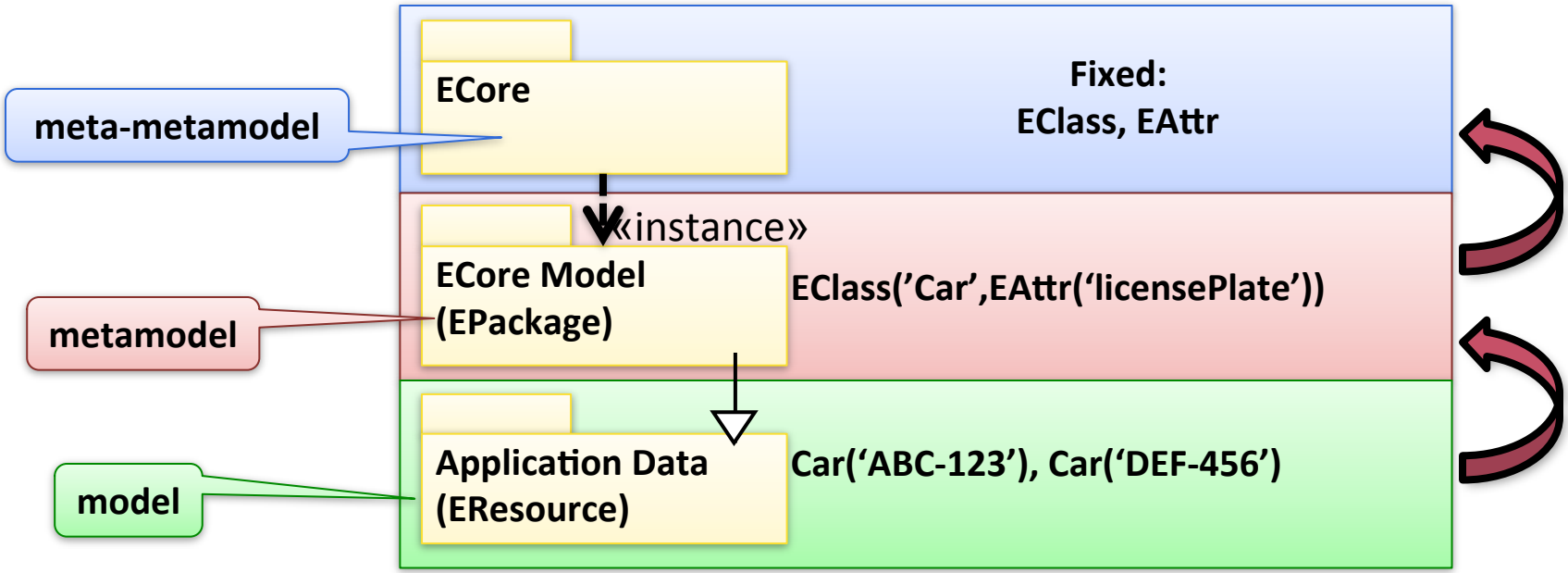
- EMF (Eclipse Modeling Framework)



- Multi-level metamodeling
  - VPM
  - Ontologies

Metalevels in other approaches

- EMF (Eclipse Modeling Framework)



- Multi-level metamodeling
  - VPM
  - Ontologies

SEMANTICS

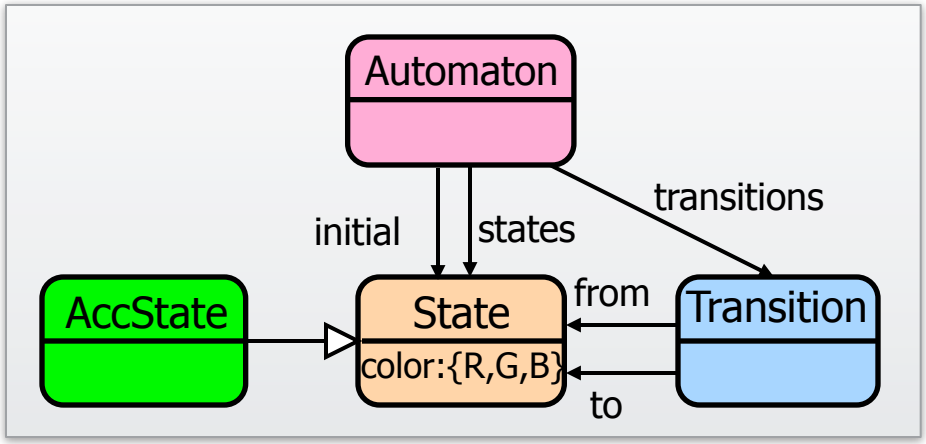
Semantics

- 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

- **Denotational** (Translational): translating concepts in one language to another language (called **semantic domain**)
  - „compiled”
  - E.g. explaining state machines as Petri-net
- **Operational**: modeling the operational behavior of language concepts
  - „interpreted”
  - Sometimes dynamic features are introduced only for formalizing dynamic sematics

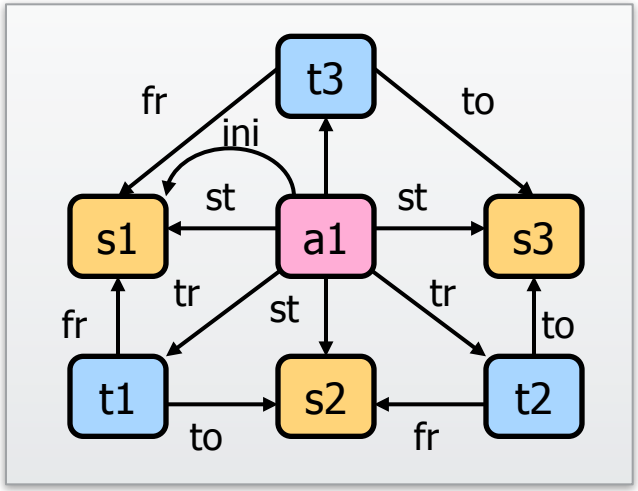
Example: Denotational semantics



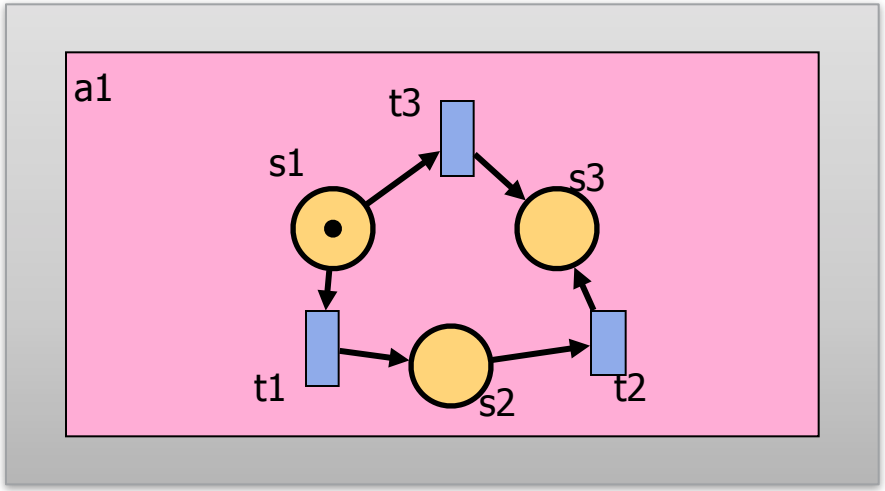
Meta (Language) level

Metamodel

Model level

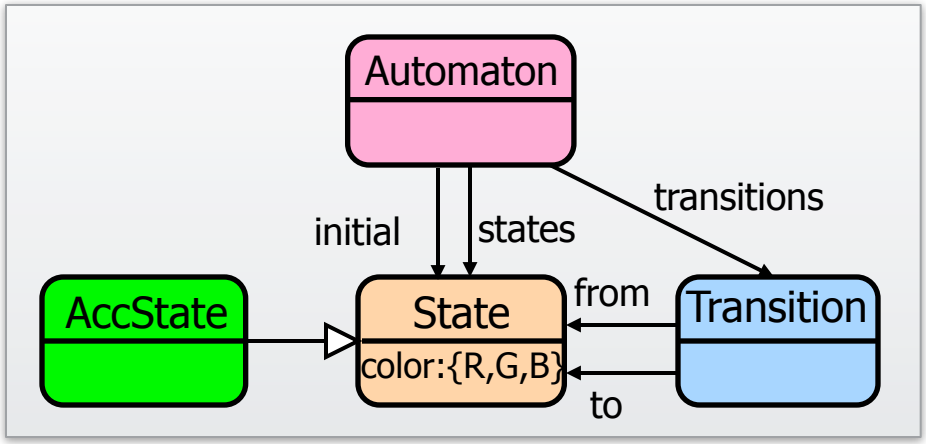


Abstract syntax



Semantic Domain

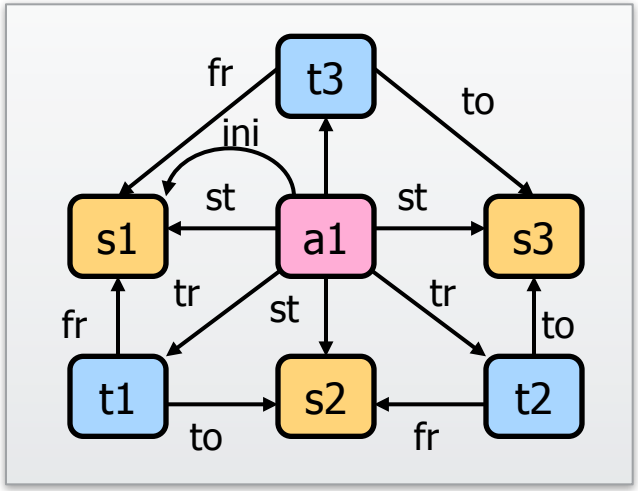
Example: Operational semantics



Meta (Language) level

Metamodel

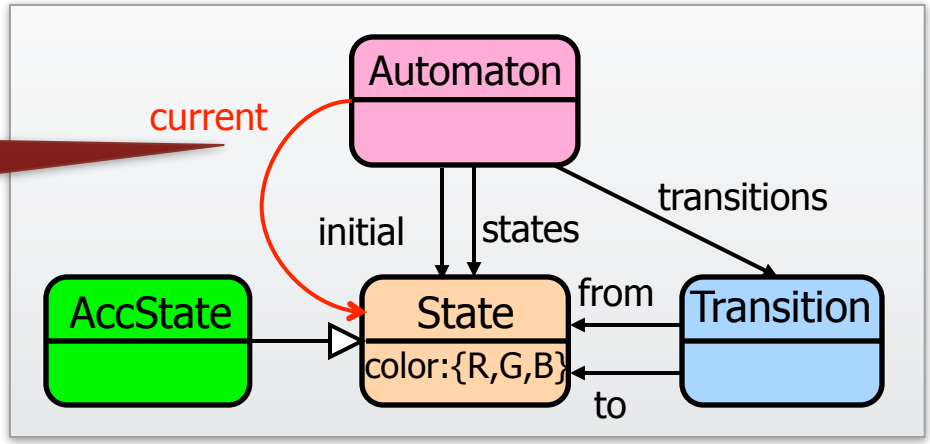
(Instance) Model level



Model in abstract syntax

Example: Operational semantics

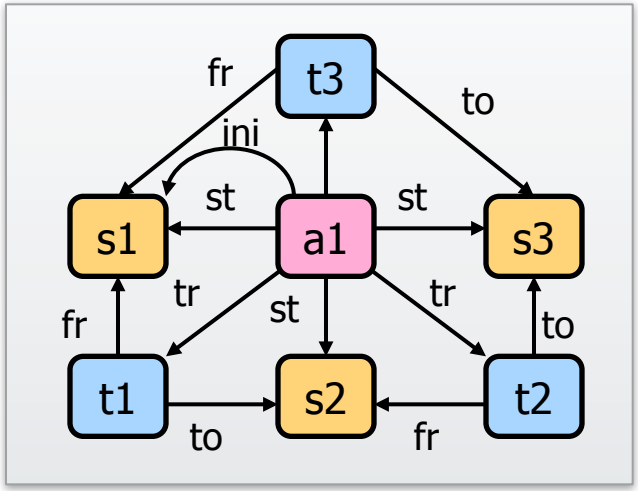
Dynamic feature



Meta (Language) level

Metamodel

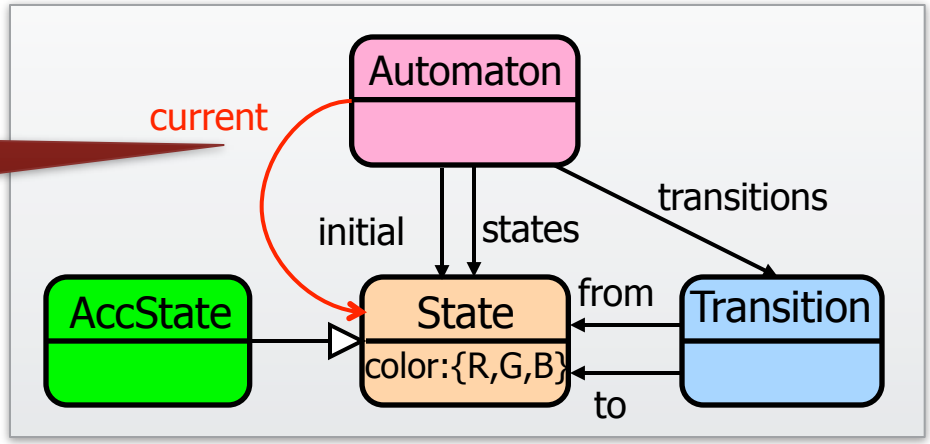
(Instance) Model level



Model in abstract syntax

Example: Operational semantics

Dynamic feature

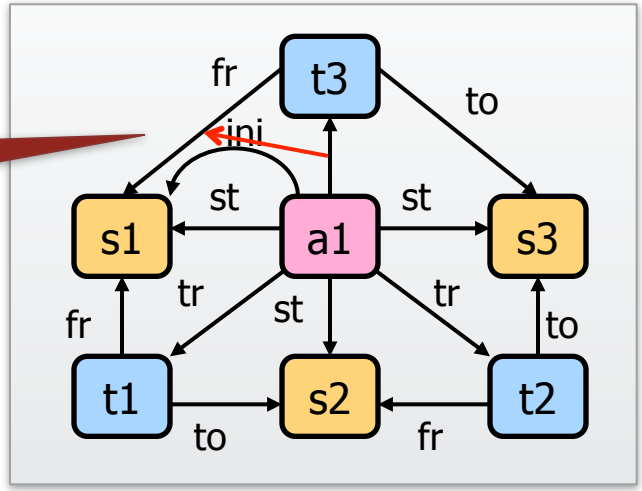


Metamodel

Meta (Language) level

(Instance) Model level

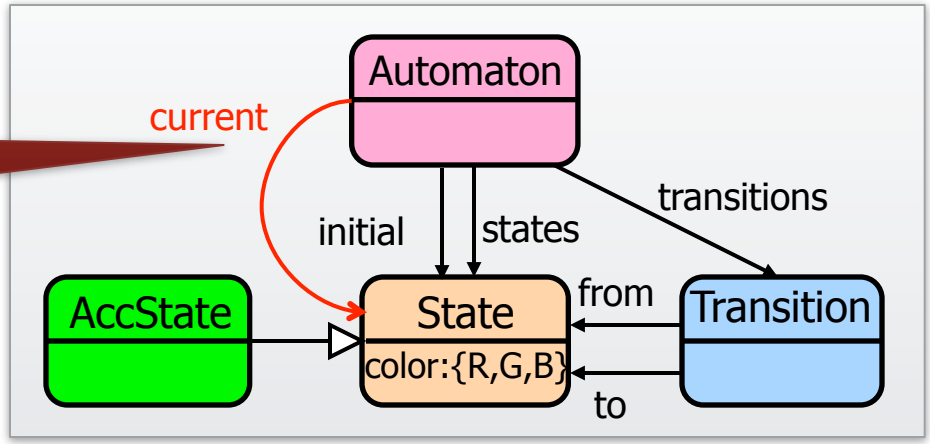
At first,
'current '= 'initial'



Model in abstract syntax

Example: Operational semantics

Dynamic feature

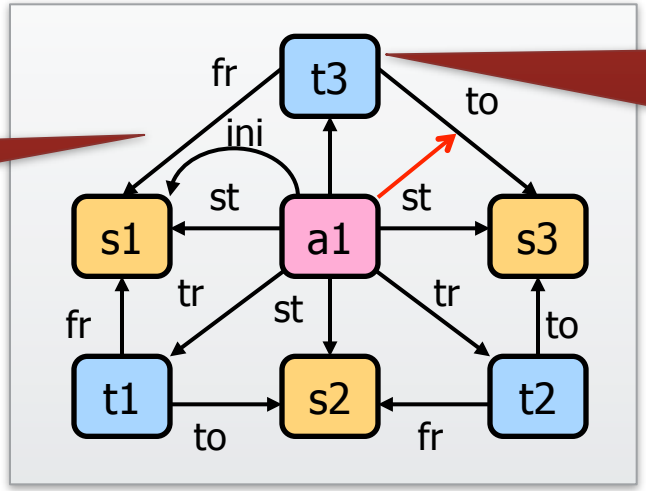


Metamodel

Meta (Language) level

(Instance) Model level

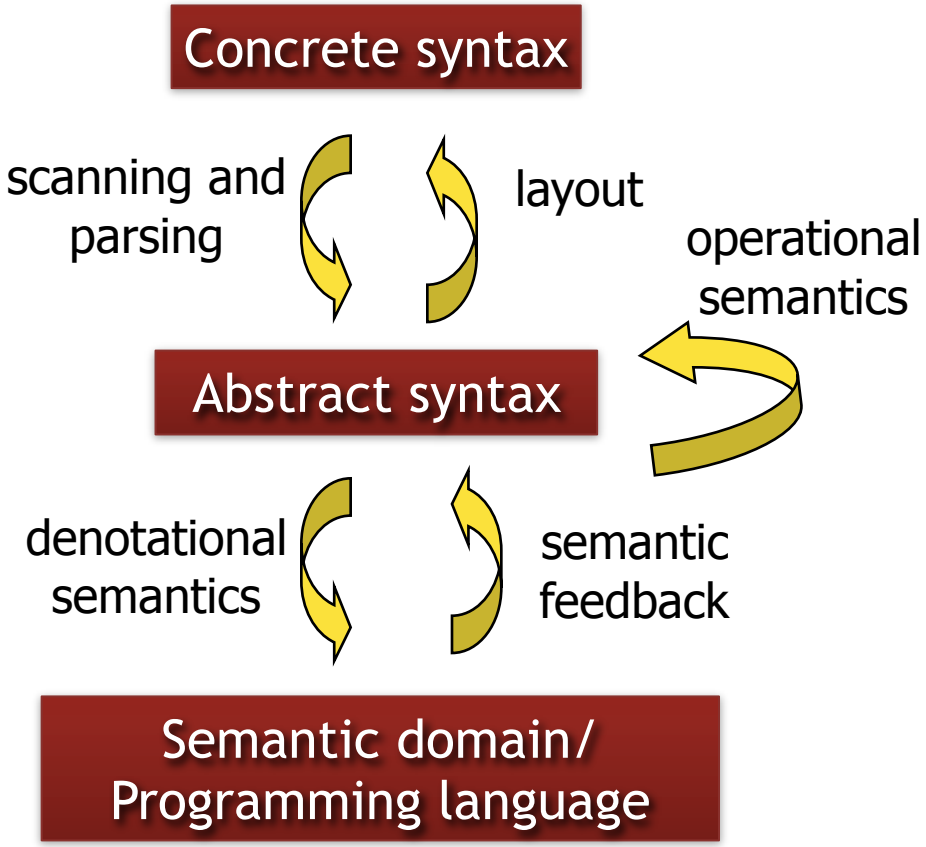
At first, 'current '= 'initial'



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, Mata, 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

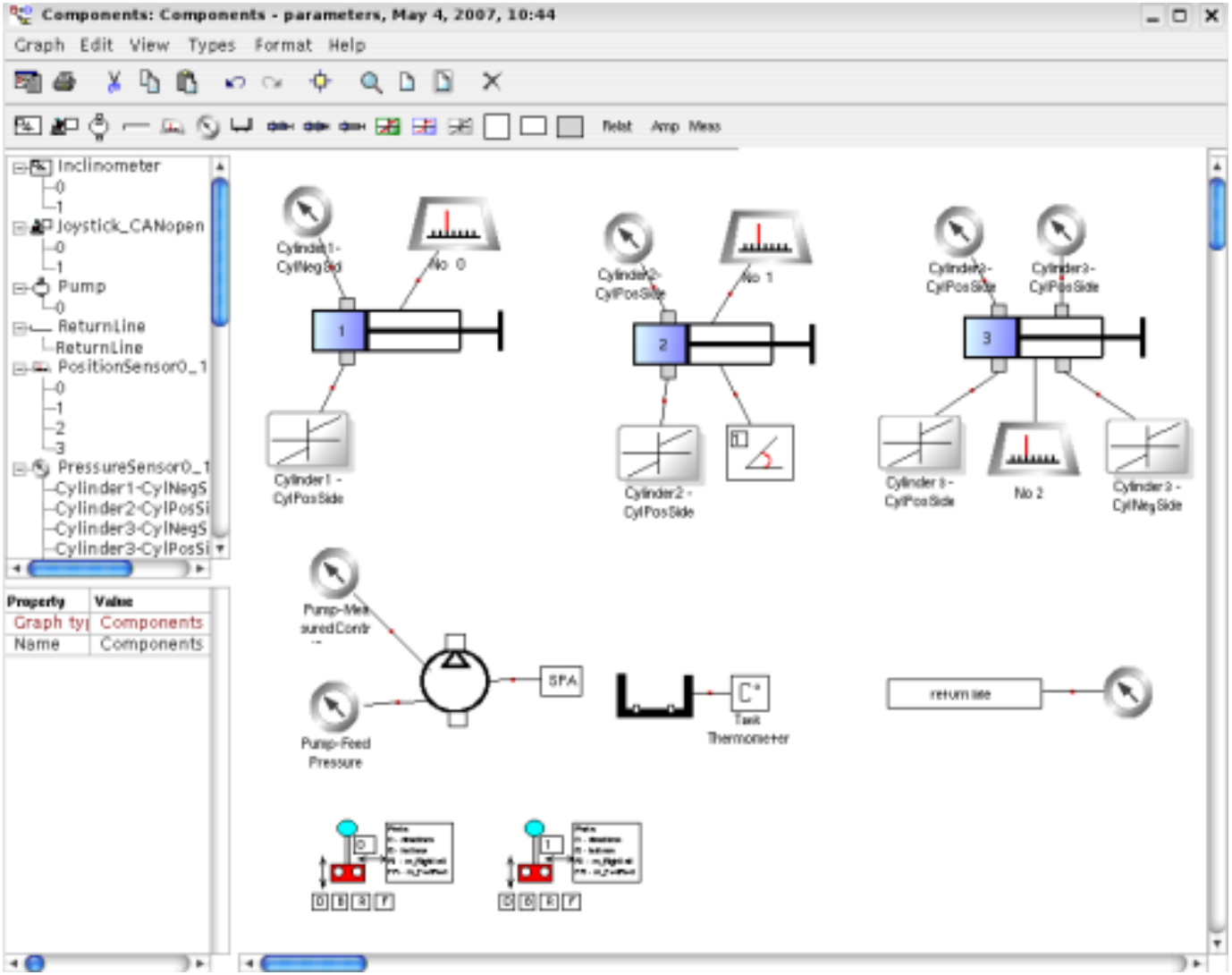
- Eclipse
  - EMF
  - openArchitectureWare
- Microsoft
  - DSL Tools (Visual Studio)
- MetaCase
  - MetaEdit+
- JetBrains MPS

Language engineering (industry)

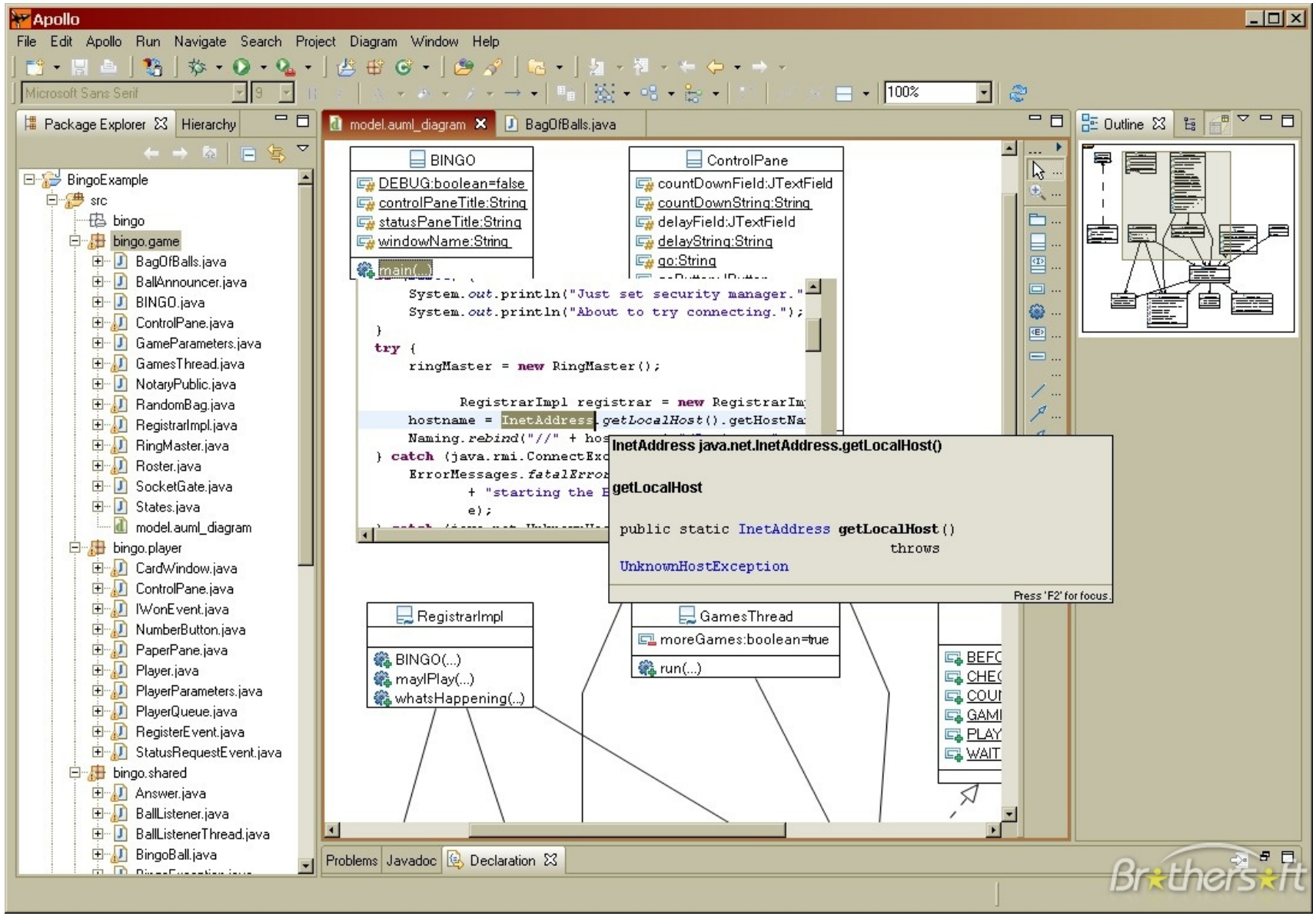
- GEMS, GME, ViatraDSM

Academia

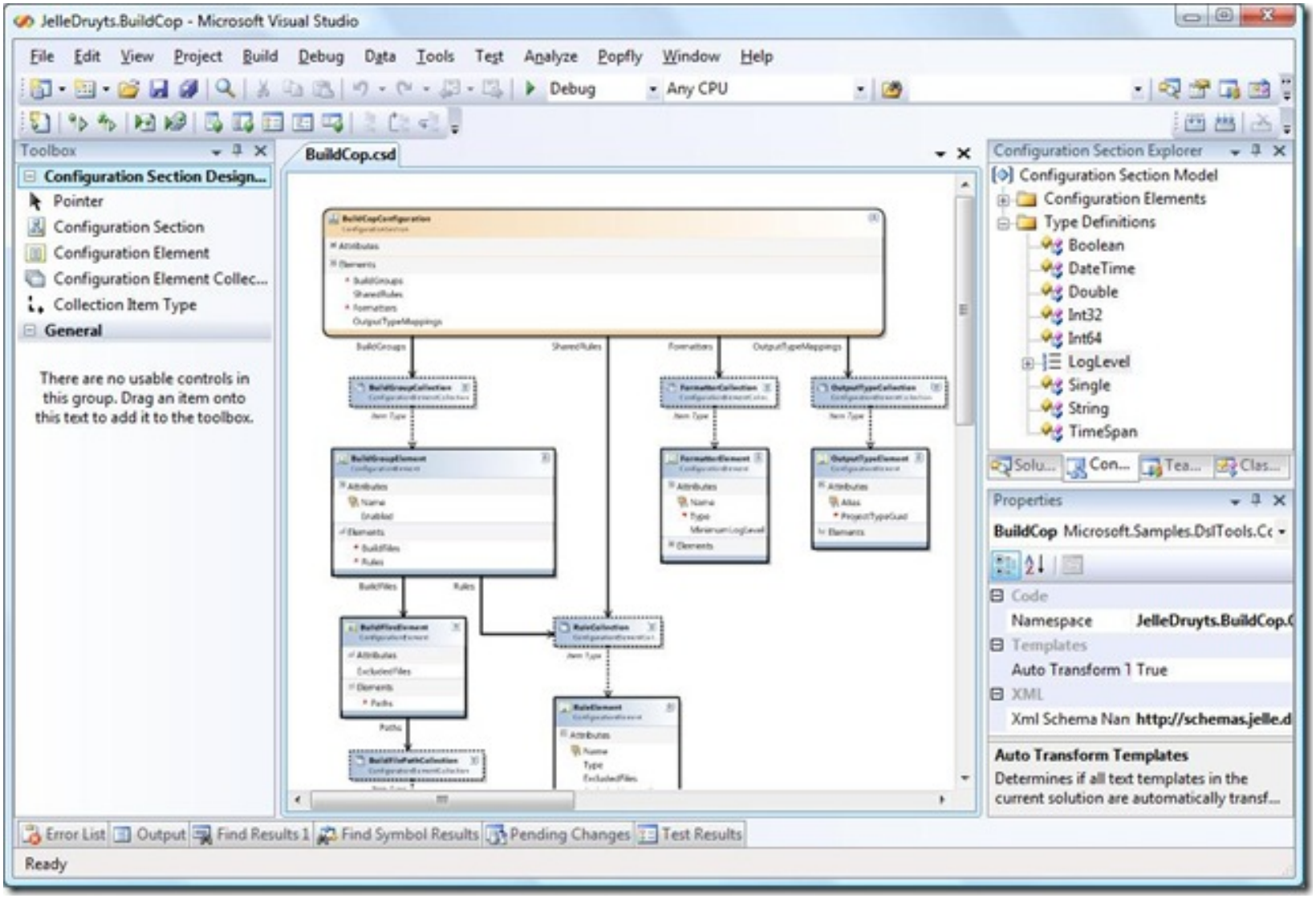
MetaEdit+


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
Tank Thermometer
return line

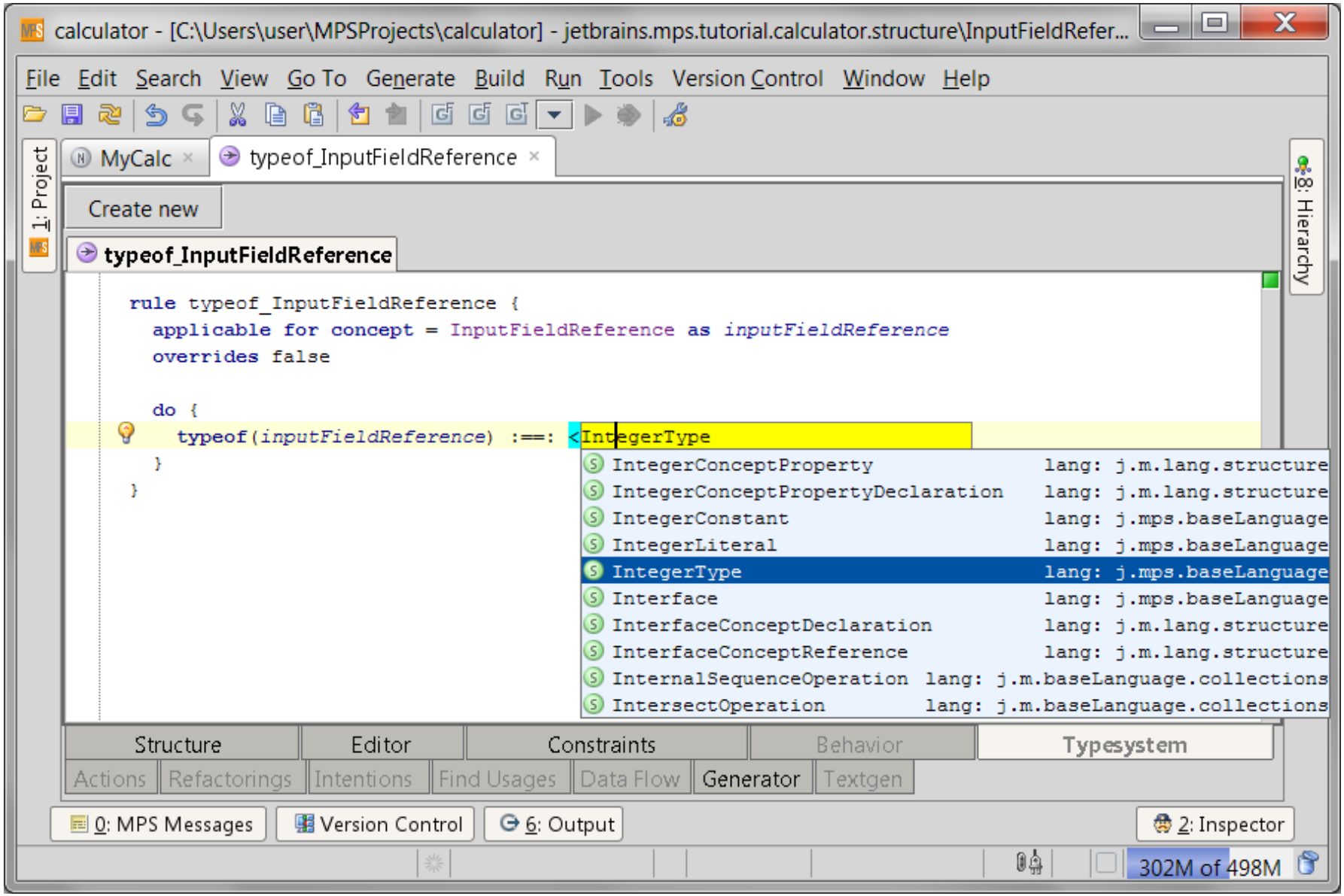
Eclipse GMF


Apollo
File Edit Apollo Run Navigate Search Project Diagram Window Help
Package Explorer
Hierarchy
BingoExample
src
bingo
bingo.game
BagOfBalls.java
BallAnnouncer.java
BINGO.java
ControlPane.java
GameParameters.java
GamesThread.java
NotaryPublic.java
RandomBag.java
RegistrarImpl.java
RingMaster.java
Roster.java
SocketGate.java
States.java
model.auml_diagram
bingo.player
CardWindow.java
ControlPane.java
IWonEvent.java
NumberButton.java
PaperPane.java
Player.java
PlayerParameters.java
PlayerQueue.java
RegisterEvent.java
StatusRequestEvent.java
bingo.shared
Answer.java
BallListener.java
BallListenerThread.java
BingoBall.java
model.auml_diagram
BagOfBalls.java
BINGO
DEBUG:boolean=false
controlPaneTitle:String
statusPaneTitle:String
windowName:String
main(...)
ControlPane
countDownField:JTextField
countDownString:String
delayField:JTextField
delayString:String
go:String
System.out.println("Just set security manager.");
System.out.println("About to try connecting.");
try {
ringMaster = new RingMaster();
RegistrarImpl registrar = new RegistrarIm
hostname = InetAddress.getLocalHost().getHostNa
Naming.rebind("//" + hos
} catch (java.rmi.ConnectExc
ErrorMessages.fatalError
+ "starting the R
e);
InetAddress java.net.InetAddress.getLocalHost()
getLocalHost
public static InetAddress getLocalHost()
throws
UnknownHostException
Press 'F2' for focus.
RegistrarImpl
BINGO(...)
mayIPlay(...)
whatsHappening(...)
GamesThread
moreGames:boolean=true
run(...)
Outline
Problems
Javadoc
Declaration

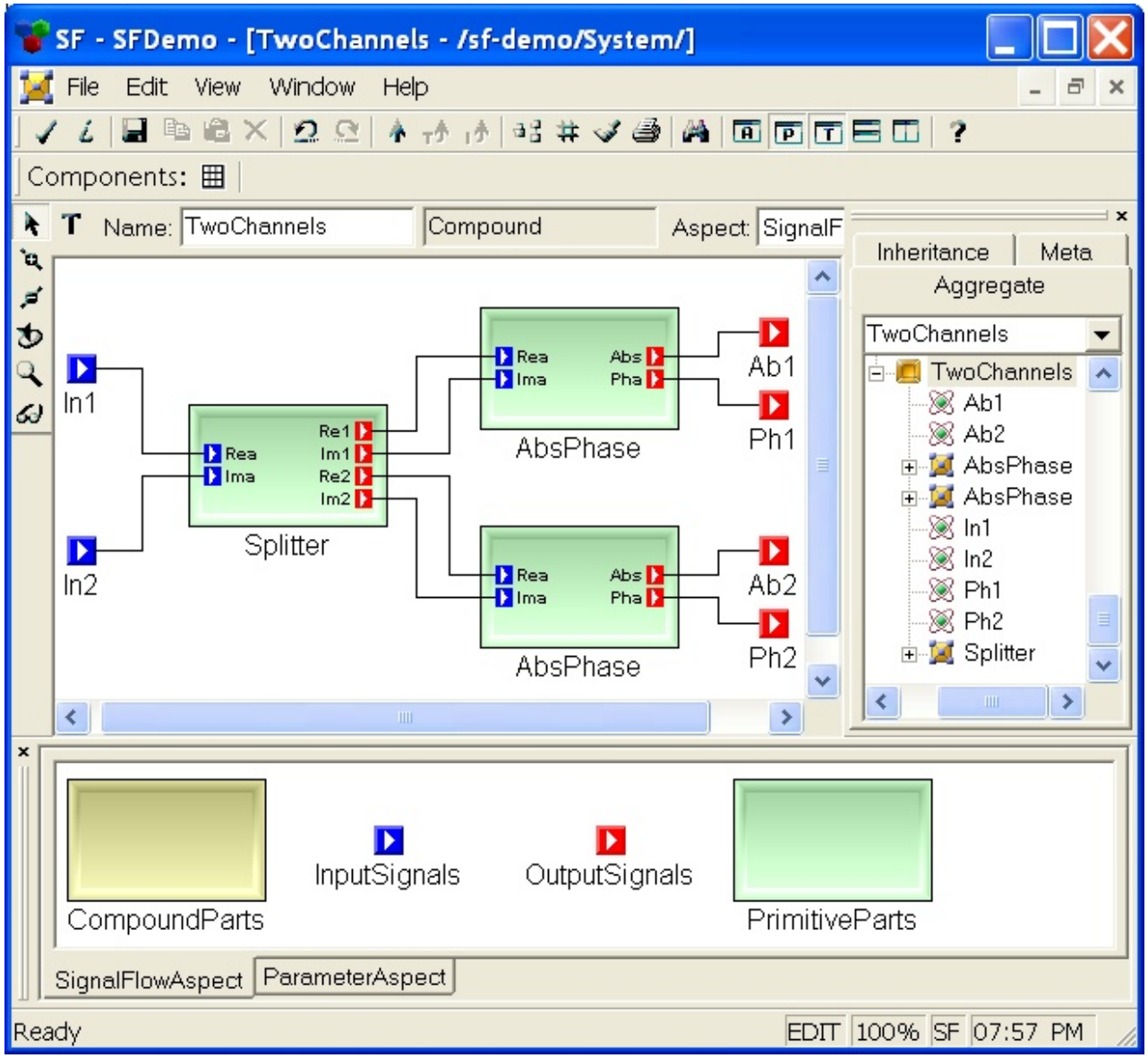
Microsoft DSL Tools



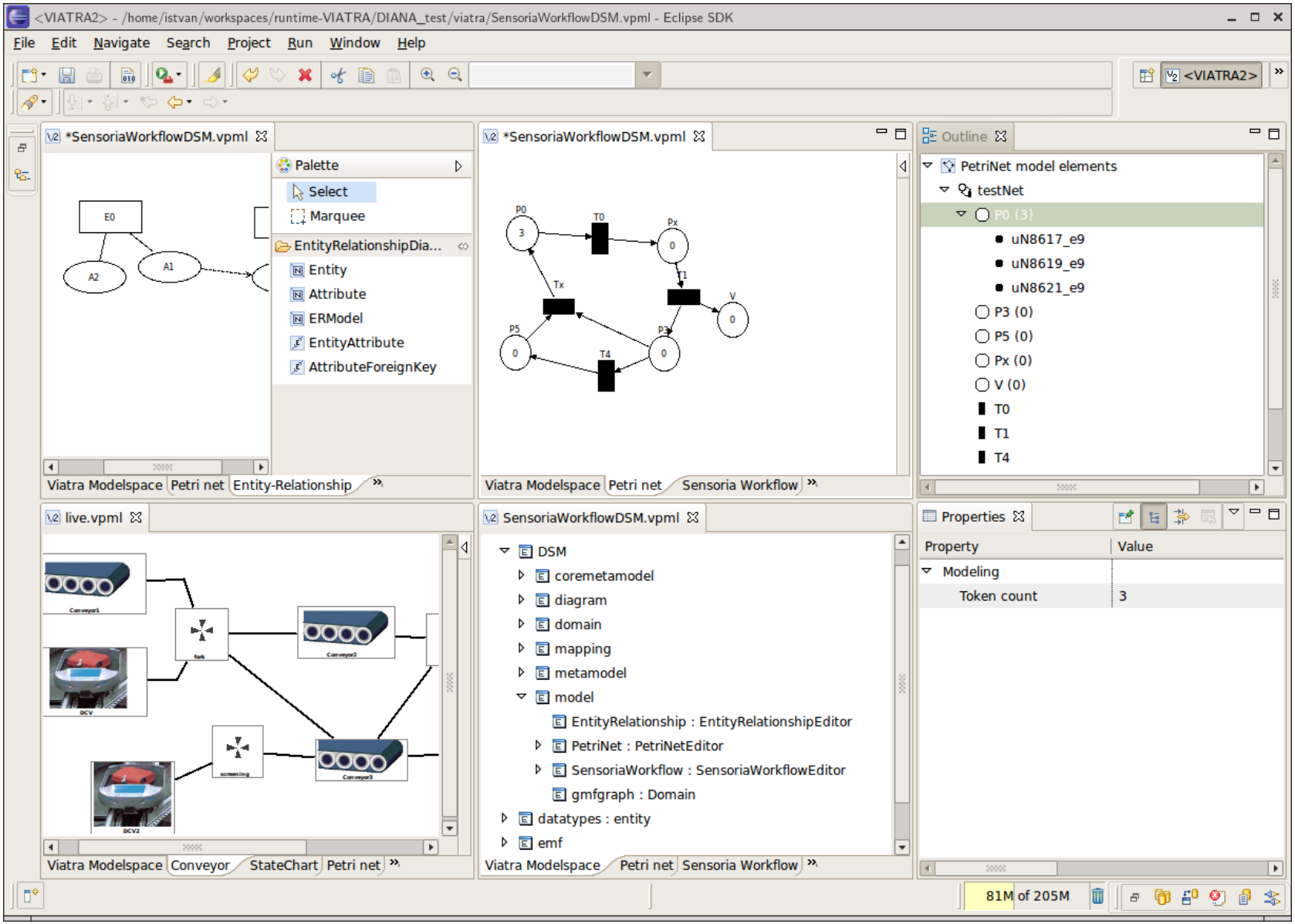
MPS


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
8: Hierarchy
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

GME


SF - SFDemo - [TwoChannels - /sf-demo/System/]
File Edit View Window Help
Components:
Name: TwoChannels
Compound
Aspect: SignalF
Inheritance
Meta
Aggregate
TwoChannels
In1
In2
Rea
Ima
Re1
Im1
Re2
Im2
Splitter
Abs
Pha
AbsPhase
Ab1
Ph1
Ab2
Ph2
Ab2
Ph2
Splitter
CompoundParts
InputSignals
OutputSignals
PrimitiveParts
SignalFlowAspect
ParameterAspect
Ready
EDIT
100%
SF
07:57 PM

ViatraDSM


<VIATRA2> - /home/istvan/workspaces/runtime-VIATRA/DIANA_test/viatra/SensoriaWorkflowDSM.vpml - Eclipse SDK
File Edit Navigate Search Project Run Window Help
*SensoriaWorkflowDSM.vpml
Palette
Select
Marquee
EntityRelationshipDia...
Entity
Attribute
ERModel
EntityAttribute
AttributeForeignKey
E0
A2
A1
Viatra Modelspace Petri net Entity-Relationship
*SensoriaWorkflowDSM.vpml
P0 T0 Px T1 V Tx P5 P3 T4
Viatra Modelspace Petri net Sensoria Workflow
Outline
PetriNet model elements
testNet
P0 (3)
uN8617_e9
uN8619_e9
uN8621_e9
P3 (0)
P5 (0)
Px (0)
V (0)
T0
T1
T4
live.vpml
Viatra Modelspace Conveyor StateChart Petri net
SensoriaWorkflowDSM.vpml
DSM
coremetamodel
diagram
domain
mapping
metamodel
model
EntityRelationship : EntityRelationshipEditor
PetriNet : PetriNetEditor
SensoriaWorkflow : SensoriaWorkflowEditor
gmfgraph : Domain
datatypes : entity
emf
Viatra Modelspace Petri net Sensoria Workflow
Properties
Property
Value
Modeling
Token count
3
81M of 205M

SUMMARY

Summary

- 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

XMI: DOCUMENT DESIGN WITH METAMODELS

Major design goals

1. XML exchange format for every MOF metamodel
2. inherited benefits of XML documents
3. automatized DTD / XML Schema generation
4. partial models exchangeable
5. independent validation
6. extensions, non-standard models

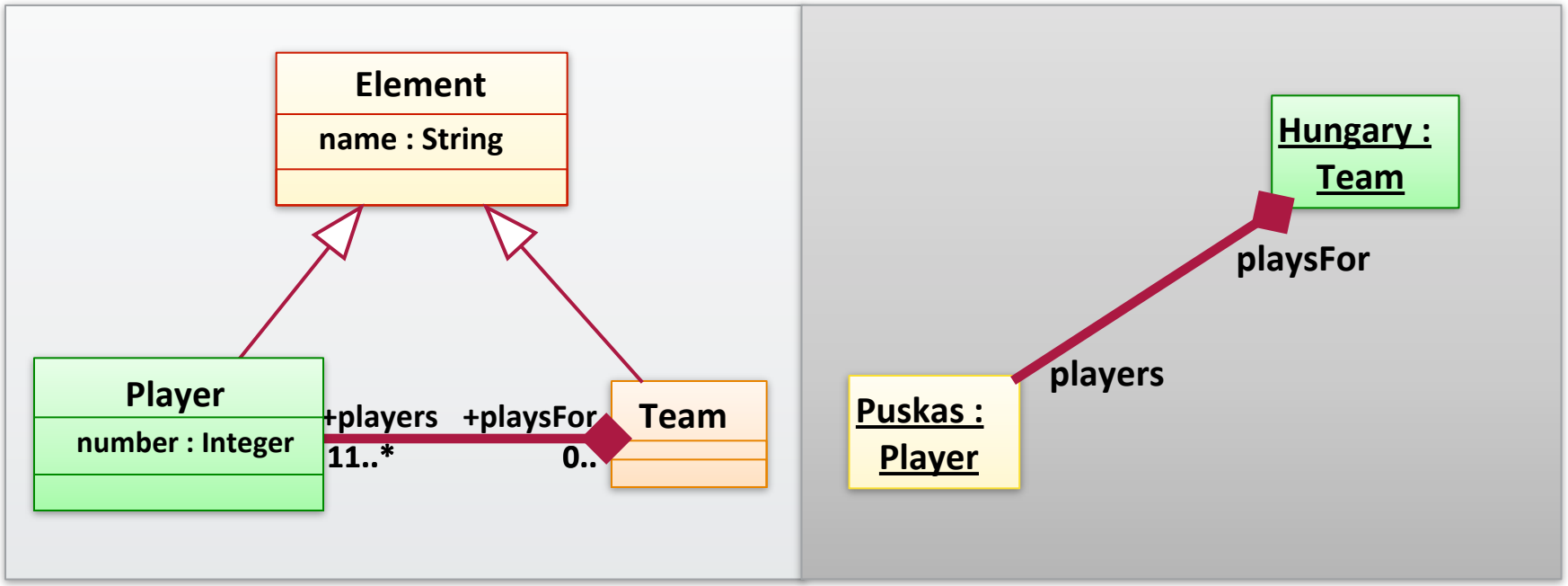
XML Metadata Interchange I.

- XML-based exchange formats
  - for models conforming to MOF metamodels
- XMI document
  - head (mandatory):
    - model maintenance (e.g. versions)
    - links (references): **xmi.id, xmi.idref**
    - extensions (tool-specific information)
    - CORBA types
  - body
    - encoding MOF-based models

XML Metadata Interchange II.

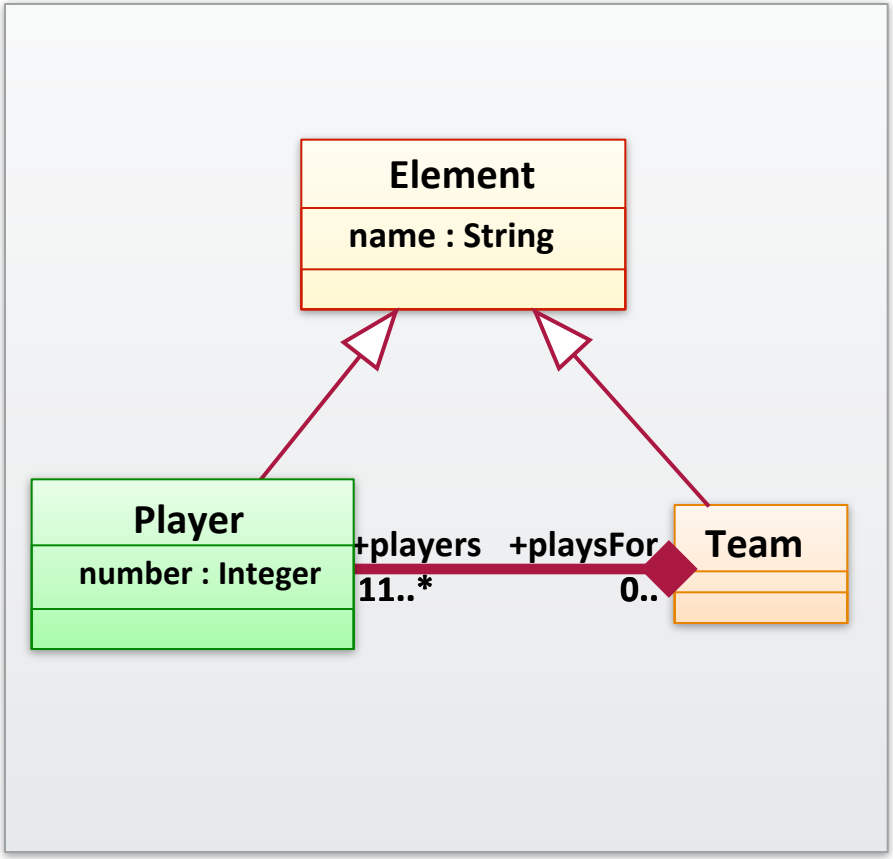
- XMI 1.0 syntax idiosyncracies
  - XML elements: dominating
    - mandatory: `xmi`, `xmi.header`, etc.
    - user-defined: Classes, Attributes, Packages
  - XML attributes: avoided
  - XML entities: inherited
- XML Schema / DTD
  - automatically derivable from MOF metamodels

Example: metamodel and model



- Team metamodel
  - level M2
- Team model
  - level M1

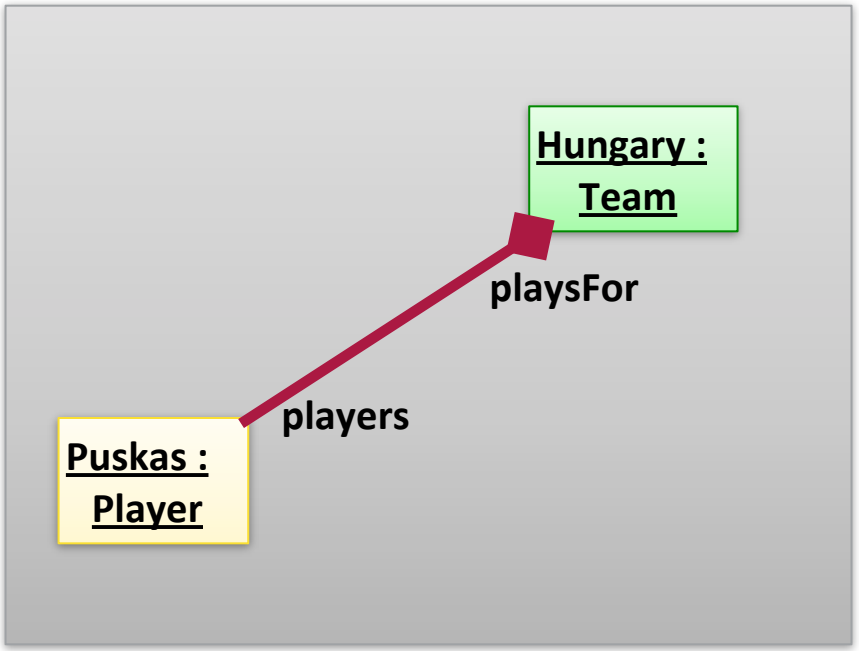
XML Schema for XMI document




```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/
XMLSchema">
  <xs:complexType name="Element">
    <xs:attribute name="firstname"
            type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="Team">
    <xs:complexContent>
      <xs:extension base="Element">
        <xs:sequence>
          <xs:element name="players"
            type="Player" minOccurs="11"
            maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Player">
    <xs:complexContent>
      <xs:extension base="Element">
       <xs:attribute name="number"
            type="xs:integer">
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

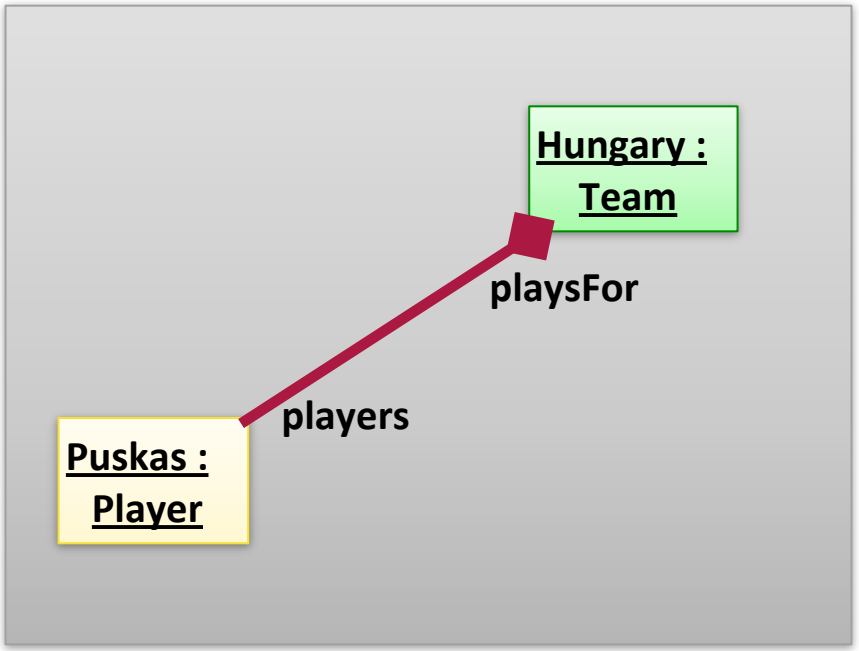
Example: XMI 1.0 document

```
<Team id='t1'>
   <Element.name>
      Hungary
   </Element.name>
   <Team.players>
     <Player id='p1'>
      <Element.name>
        Puskas
      </Element.name>
      <Player.number>
        10
      </Player.number>
      <Player.playsFor
        xmi.idref='t1'/>
      </Player>
   </Team.players>
</Team>
```



Example: XMI 1.1 document

```
<FB:Team id='t1'
        name='Hungary'>
  <FB:Team.players>
    <FB:Player id='p1'
          name='Puskas'
          number='10'
          playsFor='t1'/>
    </FB:Player>
  </FB:Team.players>
</FB:Team>
```



Example: XMI 2.0 document

```
<fb:Model xmlns:fb="…"
xmlns:xmi="…"
  <teams xmi.type="Team"
       xmi.id="t1"
       name="Hungary">
    <players xmi.id='p1'
       xmi.type=„Player”
       name='Puskas'
       number='10'
       playsFor='t1'/>
  </teams>
</fb:Model>
```