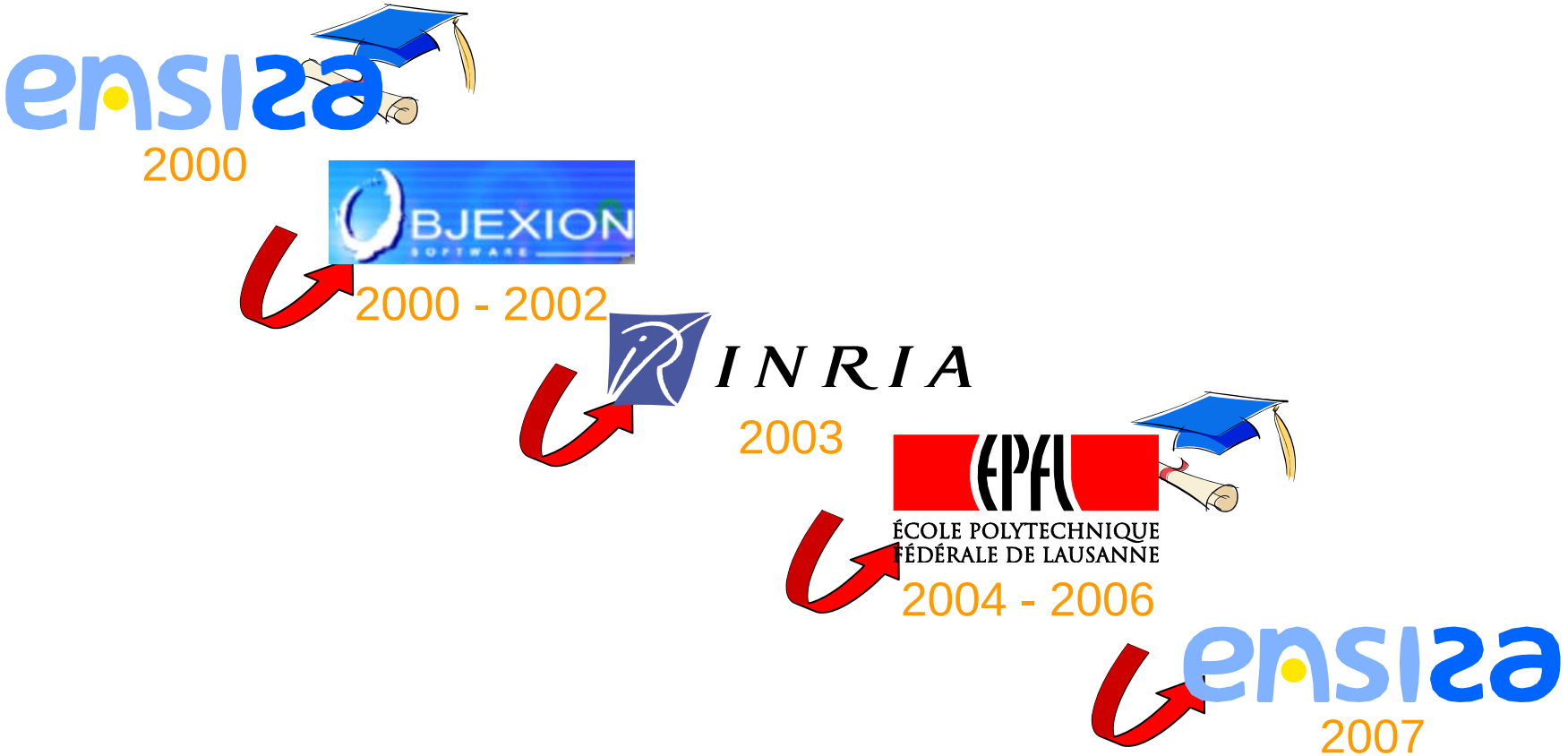
Model Driven Engineering: From Practice to Principles

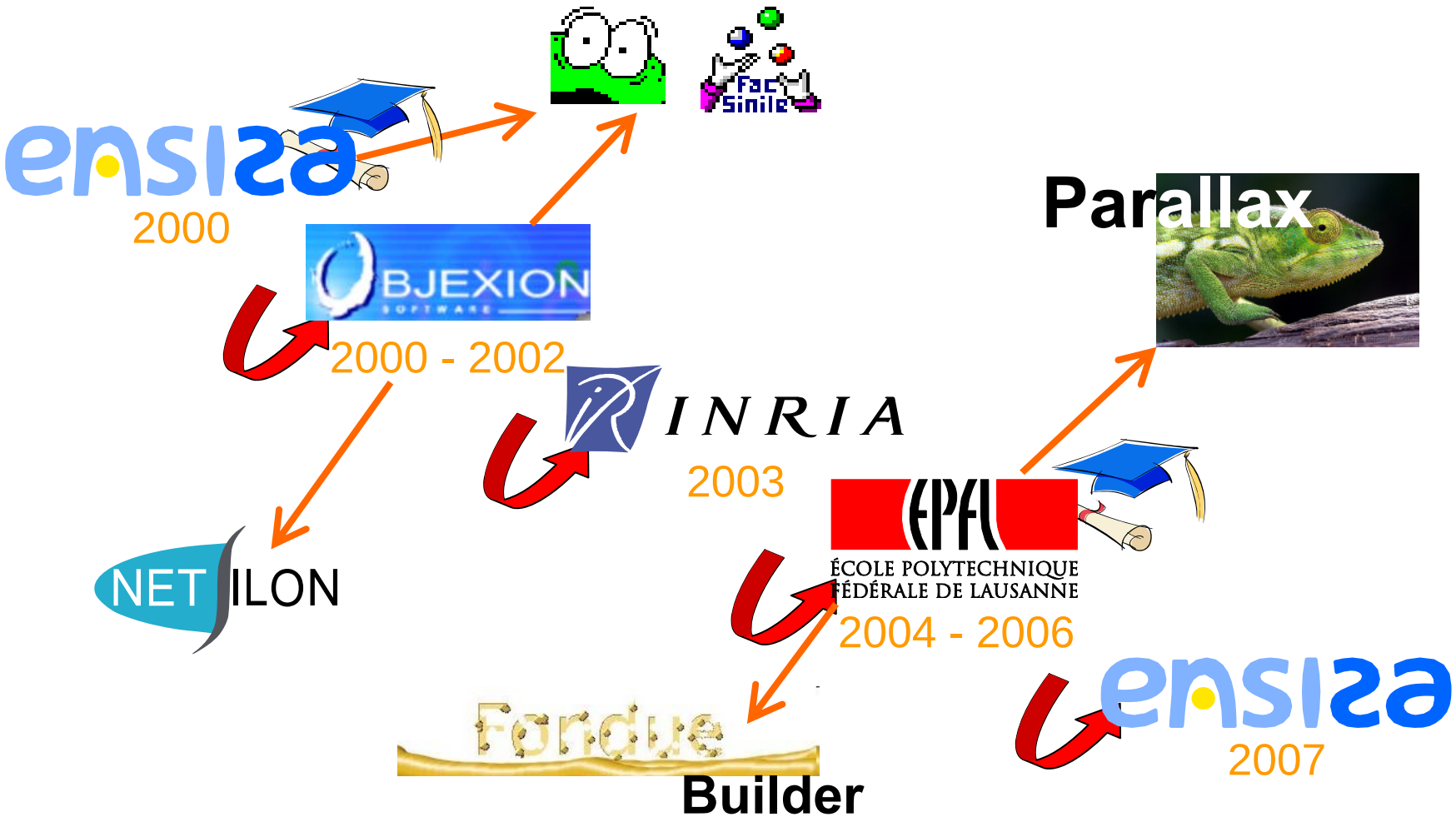
Frédéric Fondement

April 2008

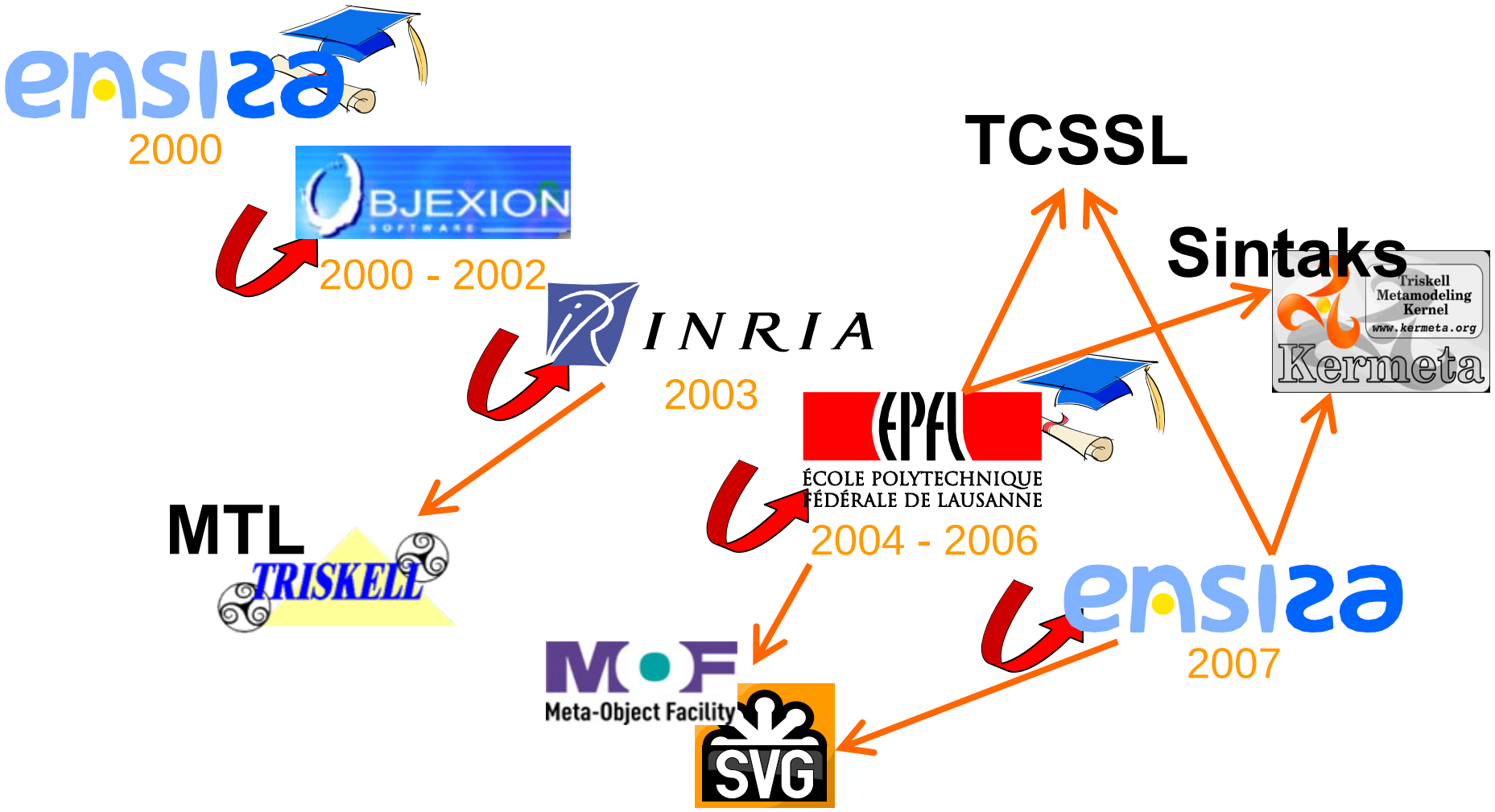
Short Vita


ensiza
2000
OBJEXION
SOFTWARE
2000 - 2002
INRIA
2003
EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
2004 - 2006
ensiza
2007

Practice


ensiza
2000
Fac Simile
BJEXION
SOFTWARE
2000 - 2002
Parallax
INRIA
2003
NET ILON
EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
2004 - 2006
Fondue
Builder
ensiza
2007

Principles


ensiza
2000
OBJEXION
SOFTWARE
2000 - 2002
INRIA
2003
MTL
TRISKELL
TCSSL
Sintaks
Triskell
Metamodeling
Kernel
www.kermeta.org
Kermeta
EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
2004 - 2006
ensiza
2007
M•F
Meta-Object Facility
SVG

Contents

- Model Driven Engineering
- The Netsilon Experience
  - Principles
  - Implementation
- Transformation: MTL
  - Principles
  - Implementation
- Modeling Languages: Concrete Syntax
  - Textual
  - Graphical
- Reuse

Emergency

Contents

Productivity Gains in SE

- ***Methodologies***
  - SADT
  - Fusion
  - OMT
  - Booch
  - Catalysis
  - RUP
  - Fondue
  - SEAM
  - …

Made possible/necessary thanks to/because of evolution of hardware…

- ***Abstraction Techniques***
  - Punched Cards
  - Assembly Code
  - Functional / Procedural Programming
  - Object-Oriented Programming
  - Patterns
  - Concurrent Programming
  - Component-Oriented Programming / Middleware
  - Design by Contracts
  - Aspect-Oriented Programming
  - Product Family Engineering
  - …

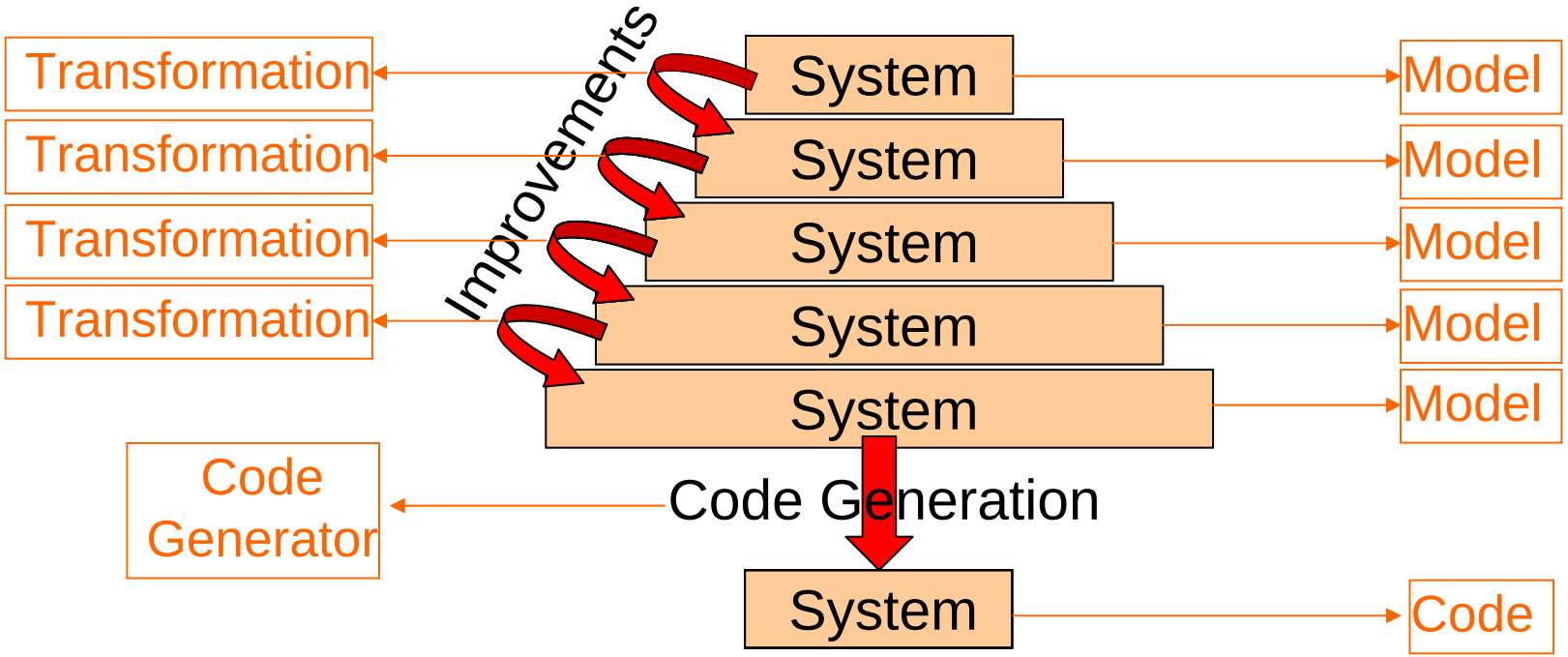
SE with Models

Model Driven Engineering (MDE)

"From contemplative to productive" (J.Bézivin)

Methodology

Organizes abstraction



Paper

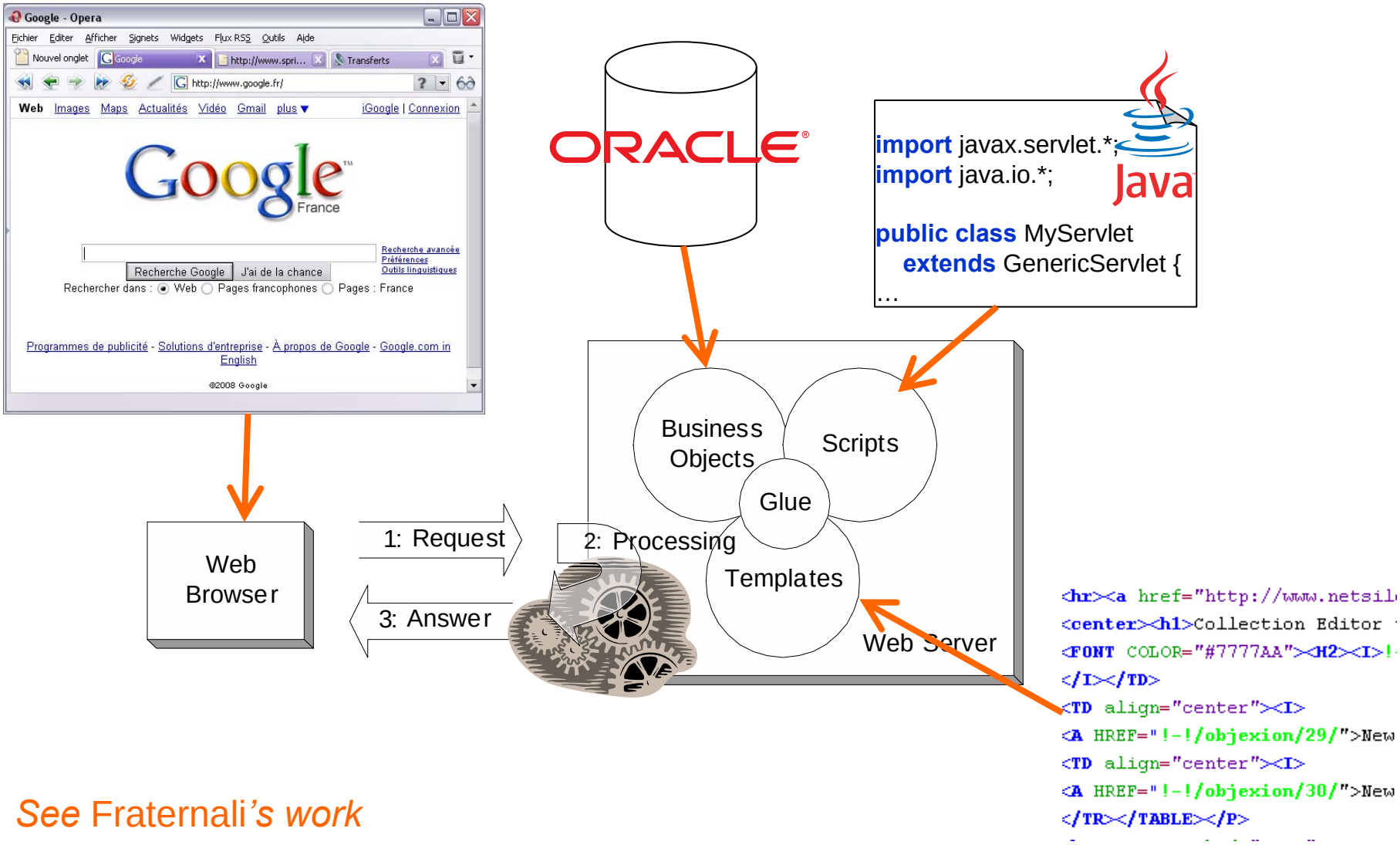
- Frédéric Fondement and Raul Silaghi, **Defining Model Driven Engineering Processes**., 3rd International Workshop in Software Model Engineering (WiSME@UML), satellite event of the UML 2004 Conference, Lisbon, Portugal, October 11, 2004.

Contents

Practice


2000
OBJEXION
SOFTWARE
2000 - 2002
Parallax
INRIA
2003
NETILON
EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
2004 - 2006
Fordue
Builder
ensiza
2007

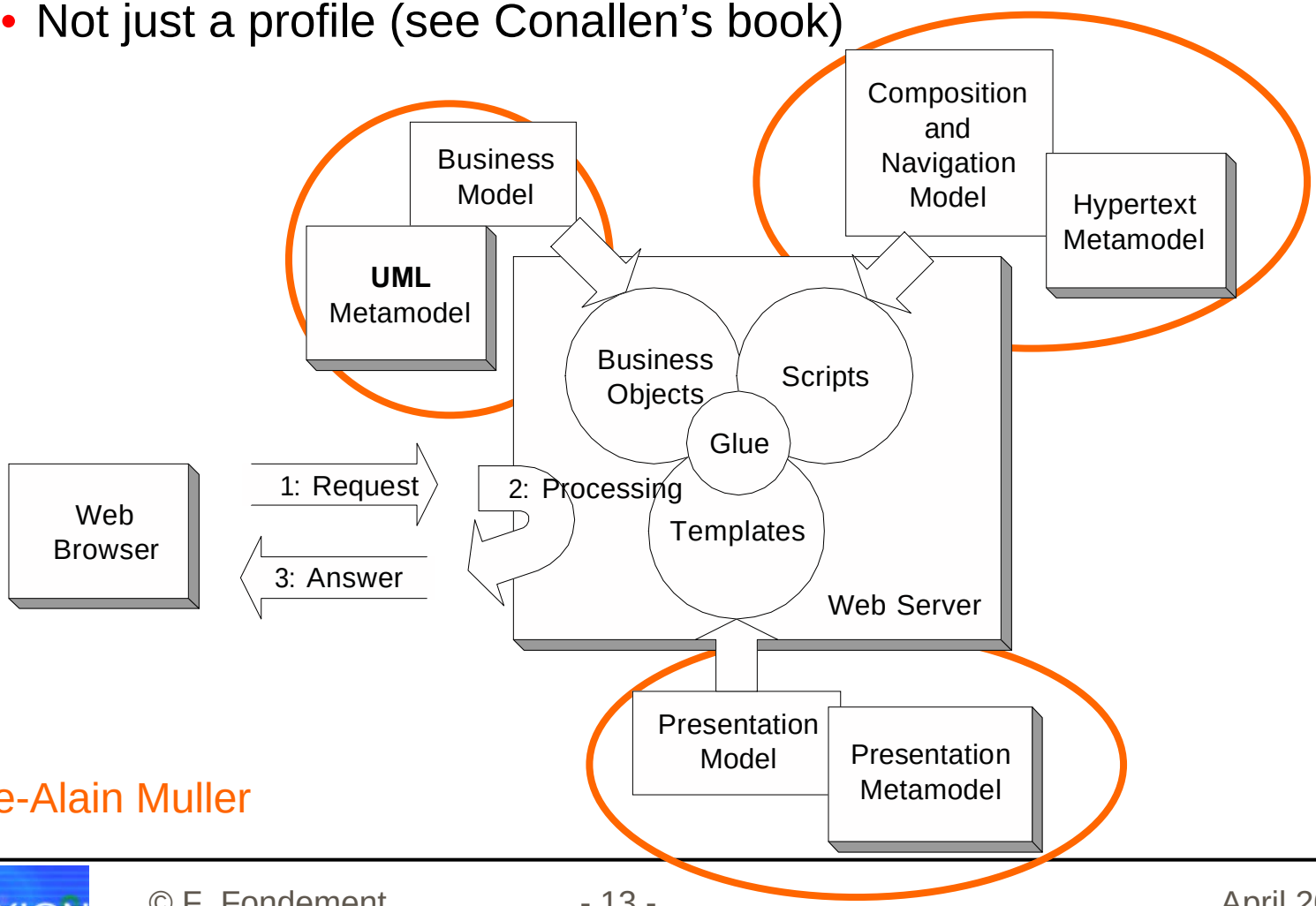
Context



See Fraternali's work

Netsilon

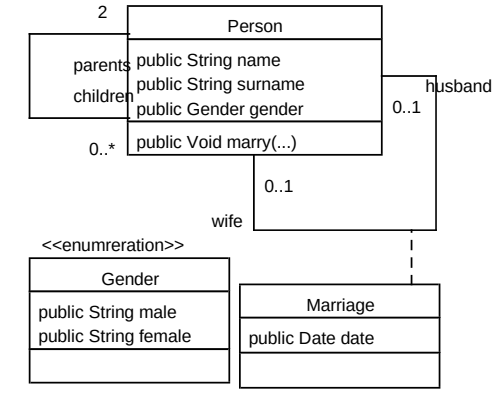
- DSL for Web Application Engineering
  - Not just a profile (see Conallen's book)

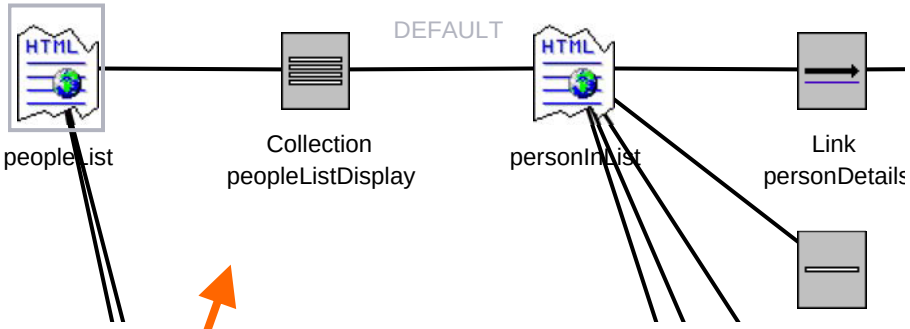


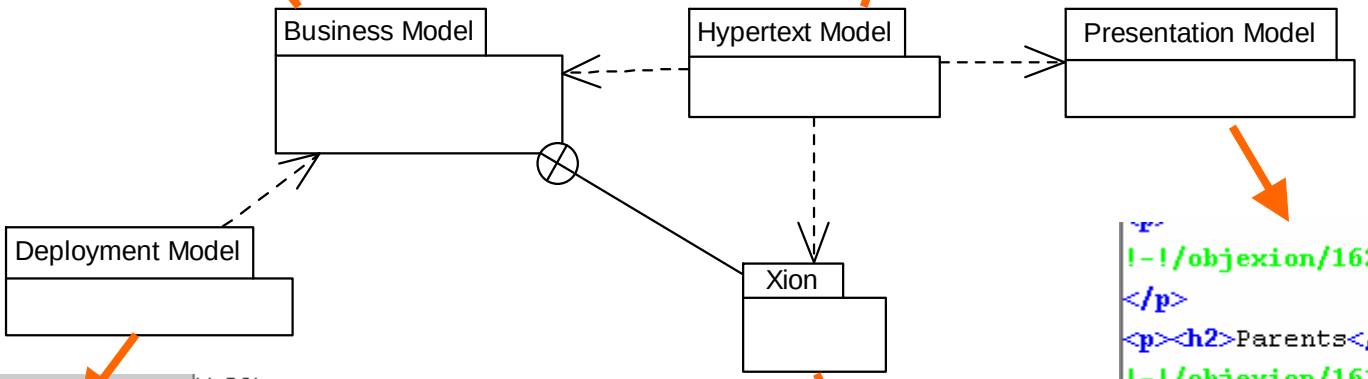
Key points

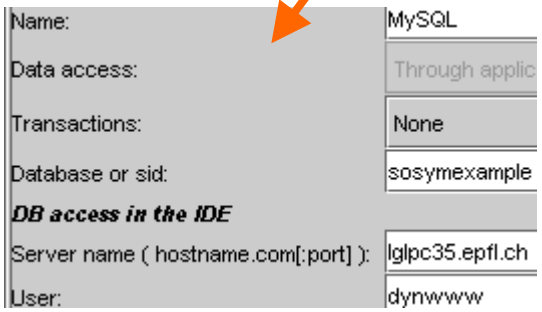
- **Various Platforms**
  - Application × Database servers
  - Interest of modeling
- **Efficiency**
- IHM
  - Text
  - Graph
- Reuse
  - Large models
- **Full generation and deployment**

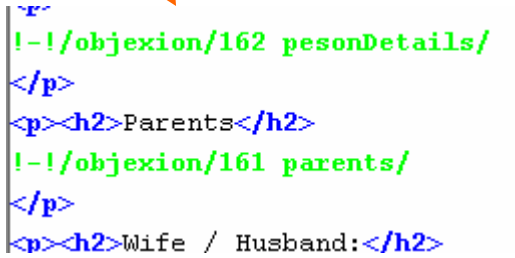
DSLs





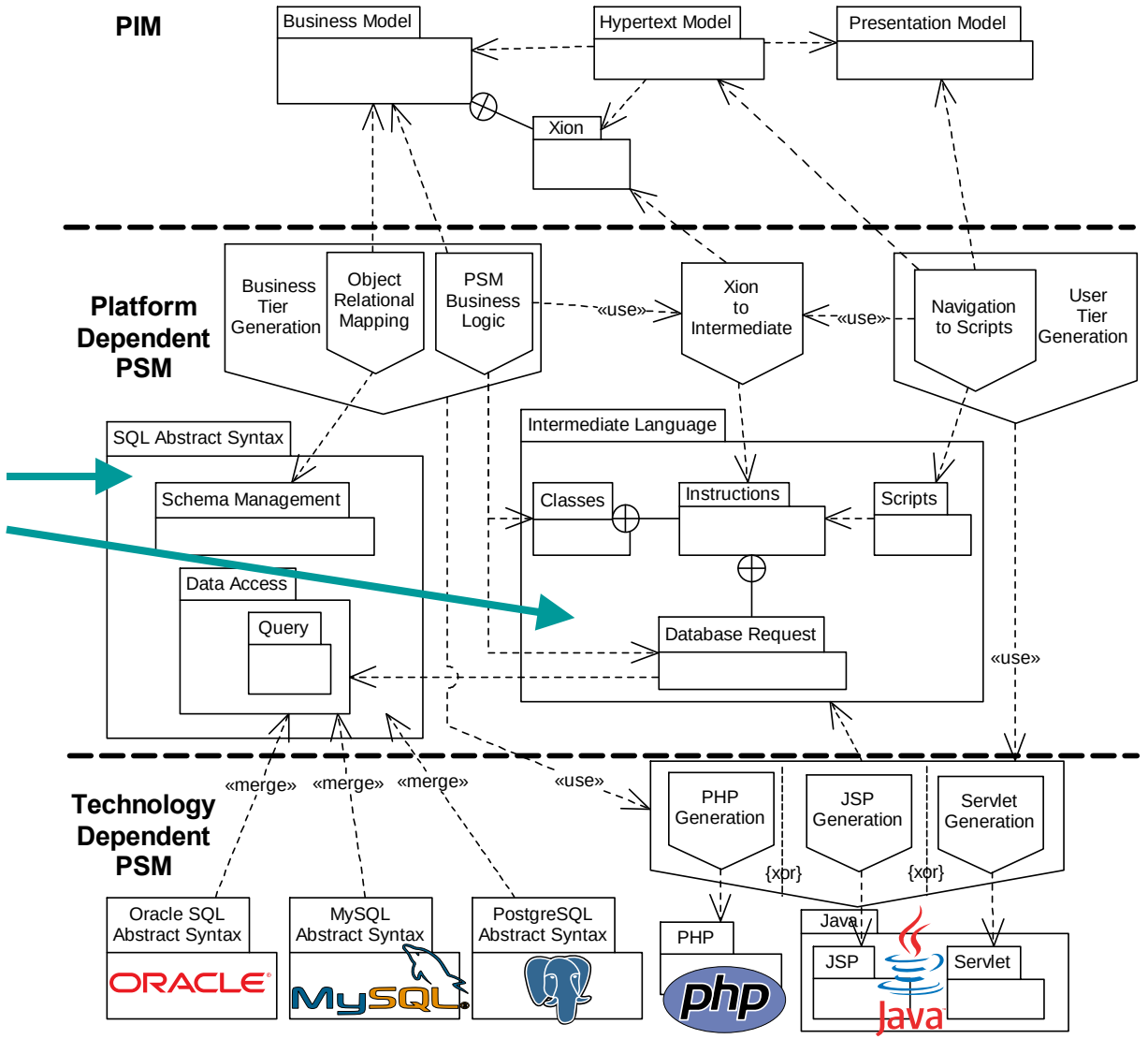






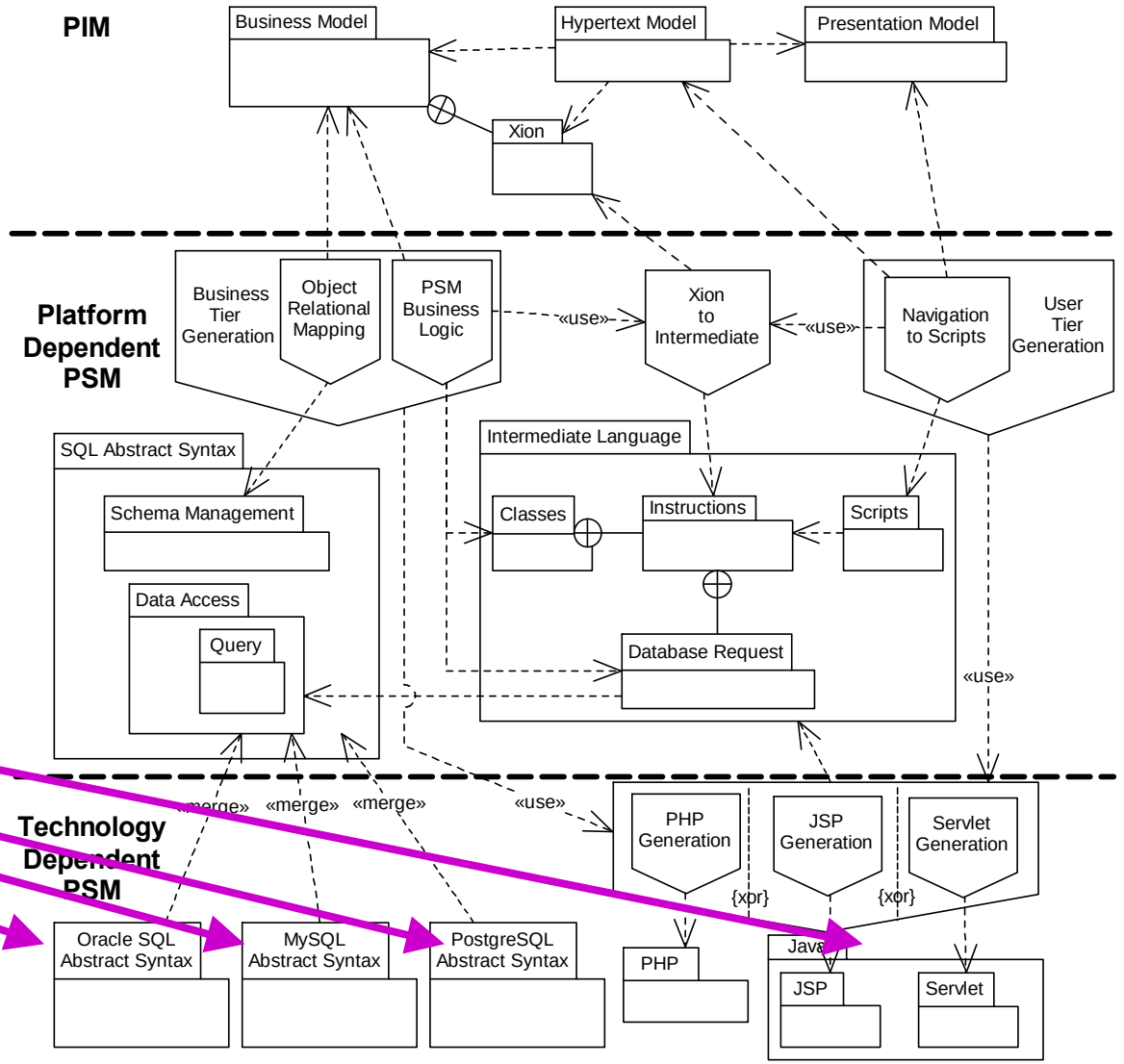

```
person.parents.children->asSet()->excluding(person)
->select(p : p.gender == #female)->sortedBy(p : p.name)
```

Generation Process



Intermediate Language

Generation Process



Intermediate Language

Target Models

- Composition
- Refinement

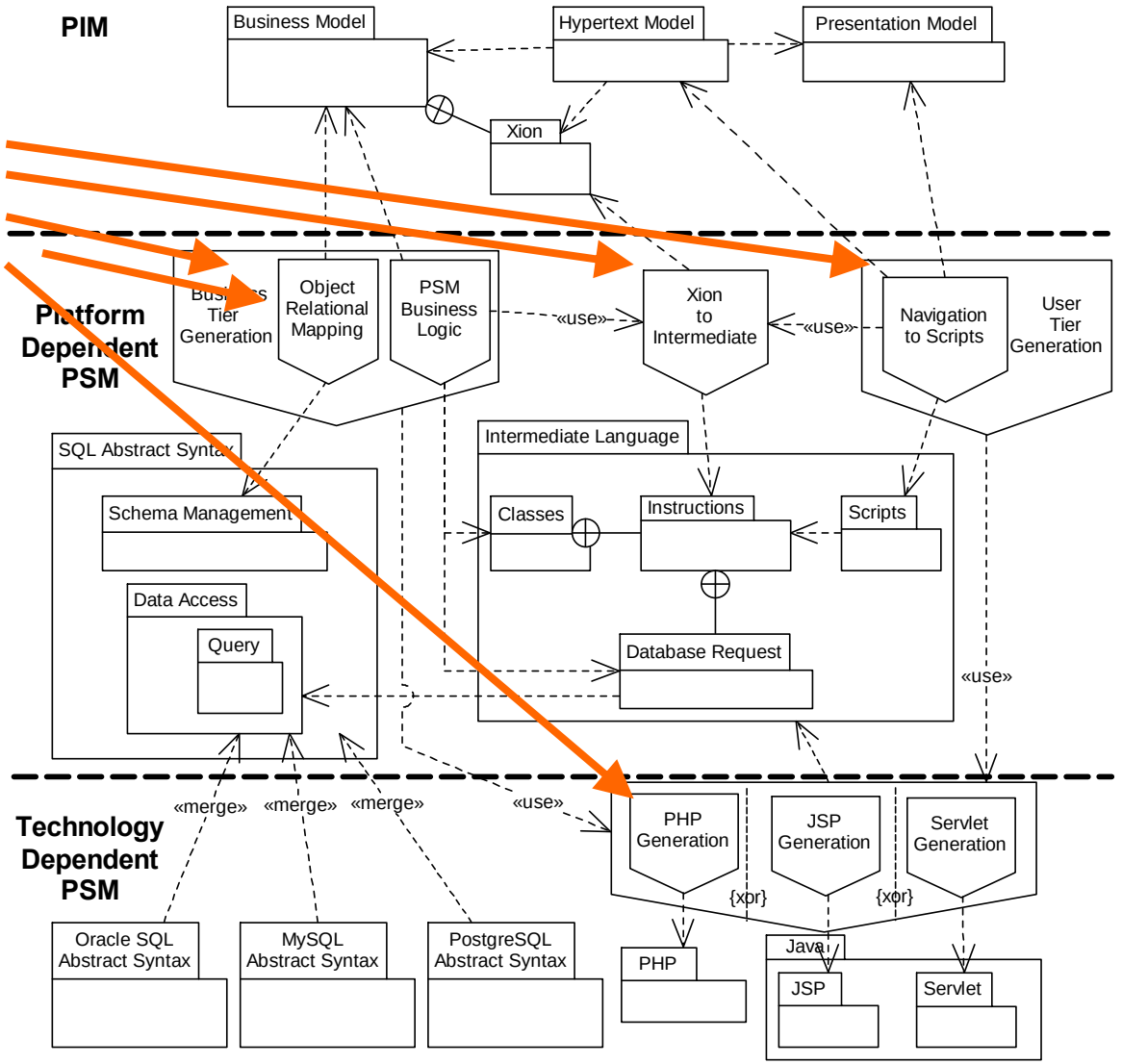
Generation Process

Model Transformations

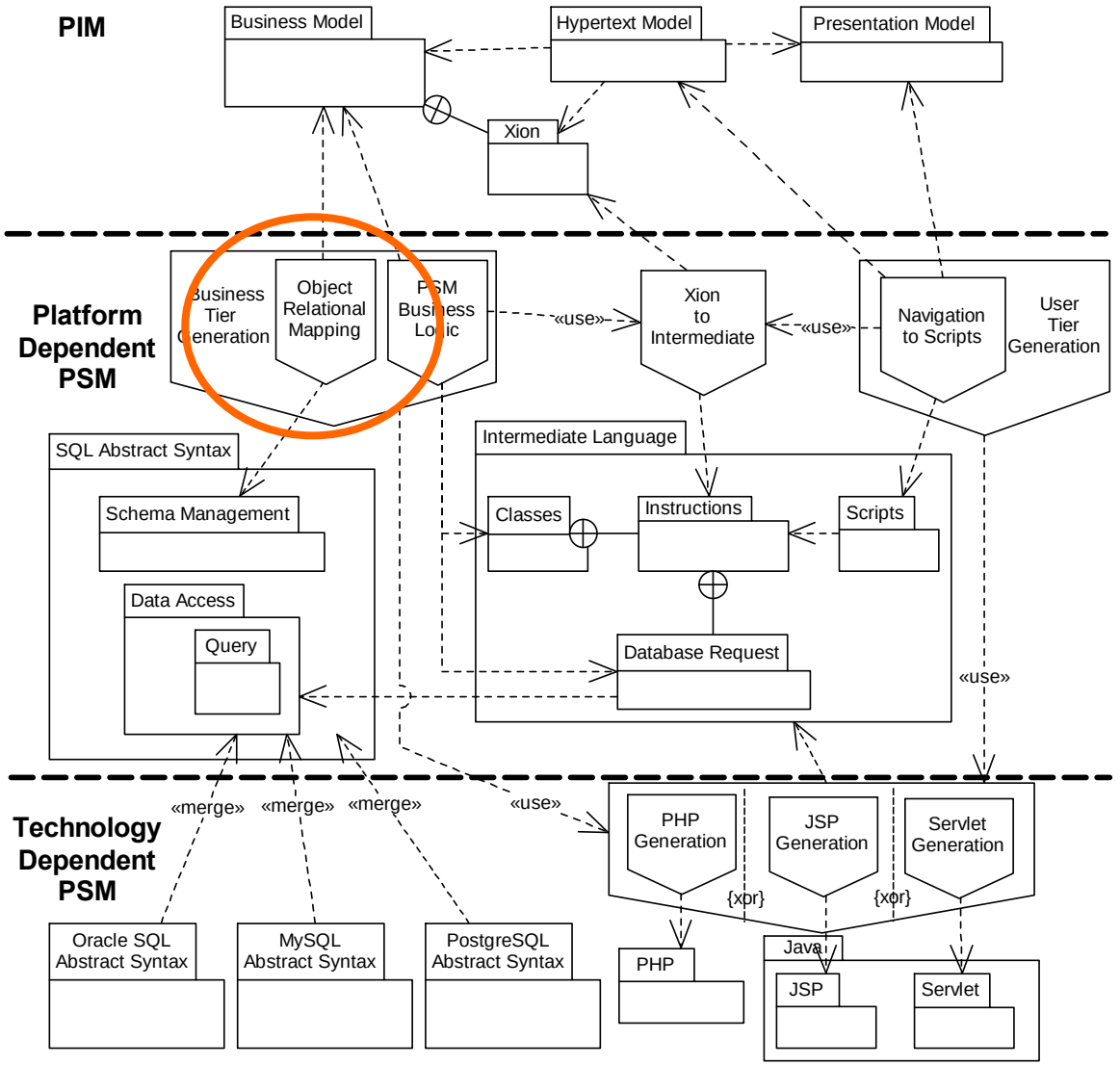
- Composition
- Selection
- Trace reuse

Intermediate Language

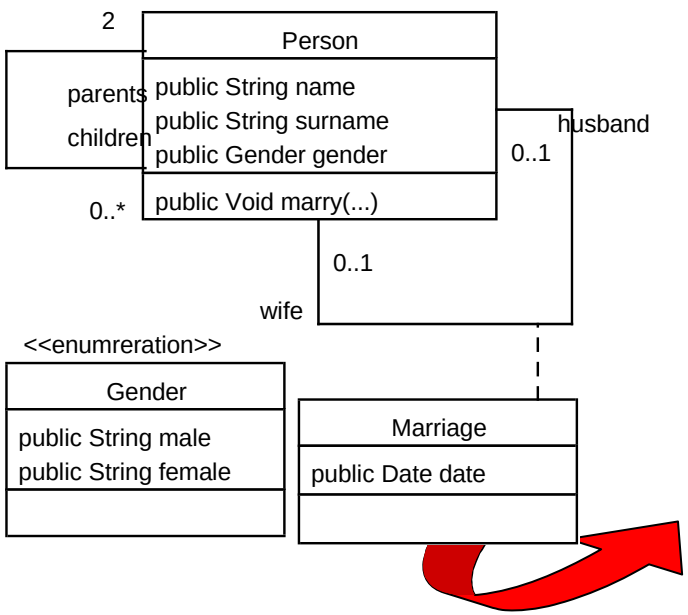
Target Models



Generation Process



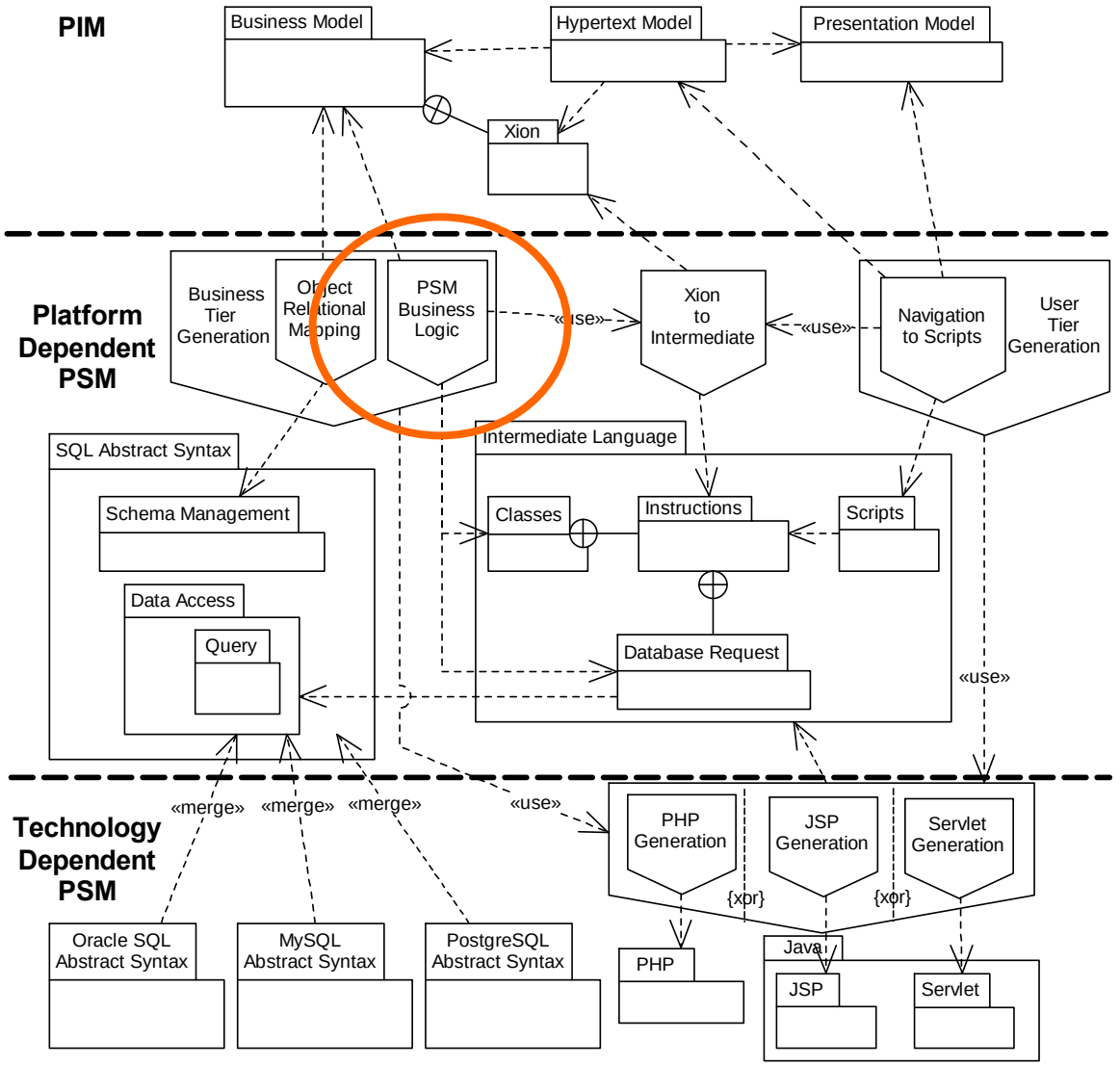
Object / Relational Mapping




```
instances
        (instances_id, classnum)
person
        (OID, name, surname, gender)
marriage
        (OID, #wife, #husband , date)
parents_children
        (#parents, #children)
```

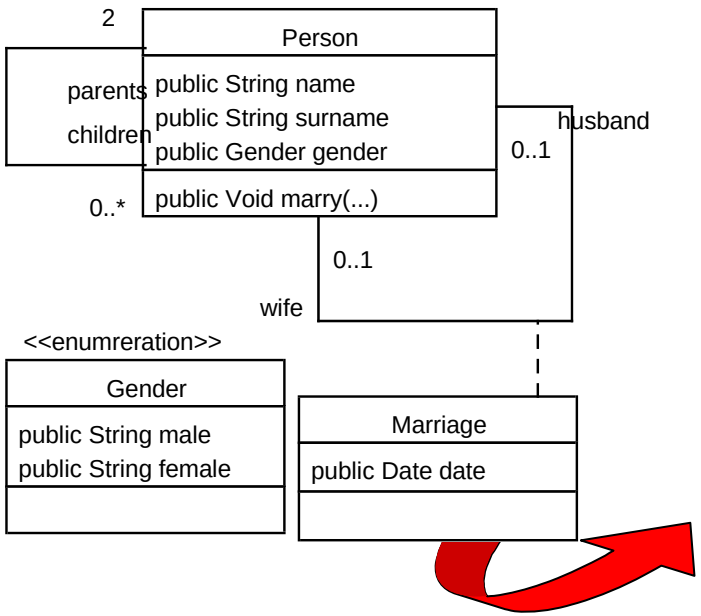
See Marcos' *work*

Generation Process



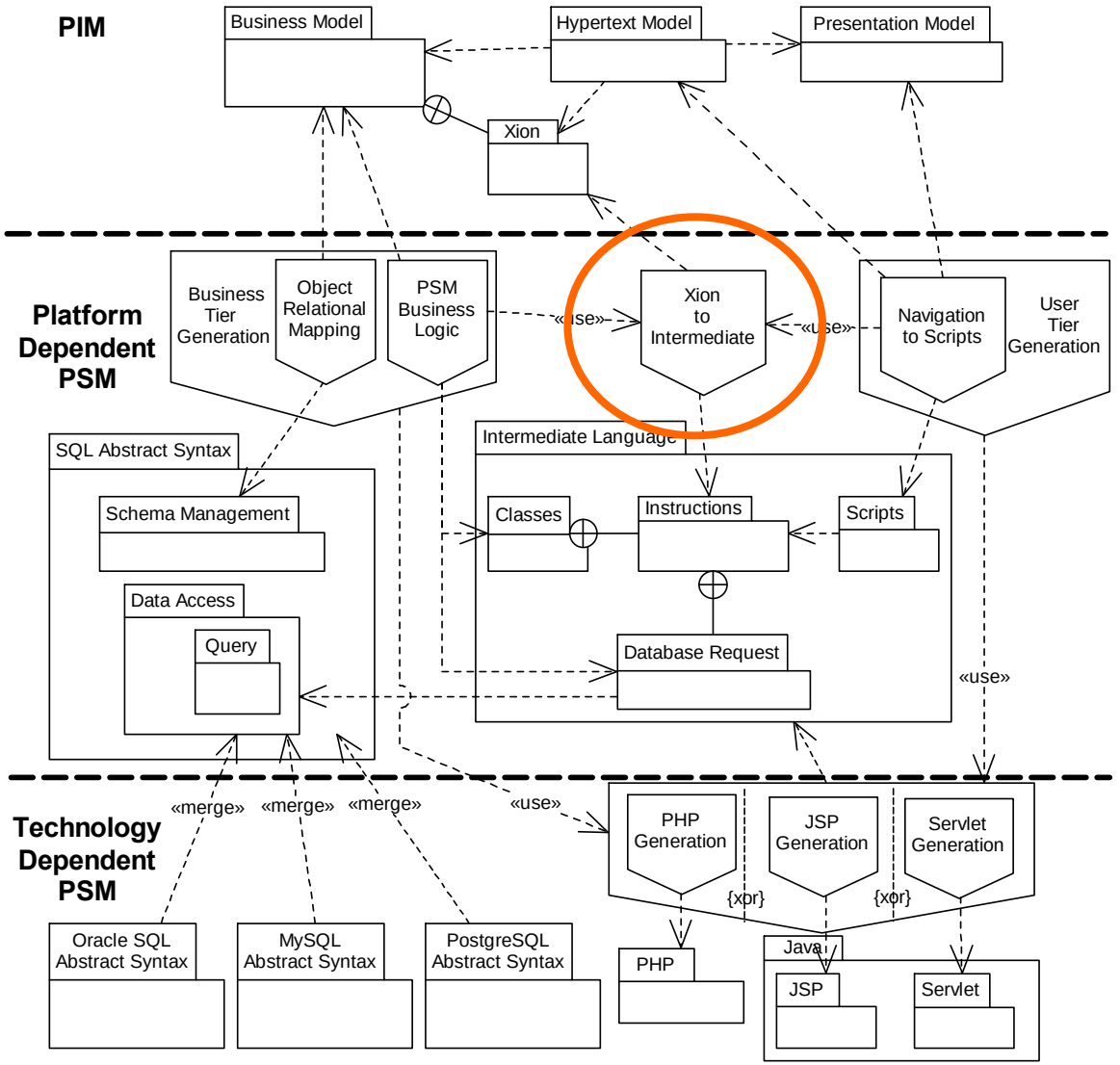
Object / Relational Mapping

- Encapsulation




```
class Person
  attribute oid : String

  function get_name : String
    return SQL_exec(
         'SELECT person.name
          FROM person
          WHERE person.OID = ' + oid
       )
  fin get_name

…
```

Generation Process



Compiling Xion

- Naïve approach

aPerson.children->collect(name)

```
Set(Person) tmp1 = SQL_exec(
        'SELECT children
         FROM parents_children
         WHERE parents = ' + aPerson.oid
       )
Set(String) tmp2 = []
Foreach e in tmp1 do
    tmp2.add(e.get_name())
End foreach
```

1

n

Compiling Xion

- Optimized Approach

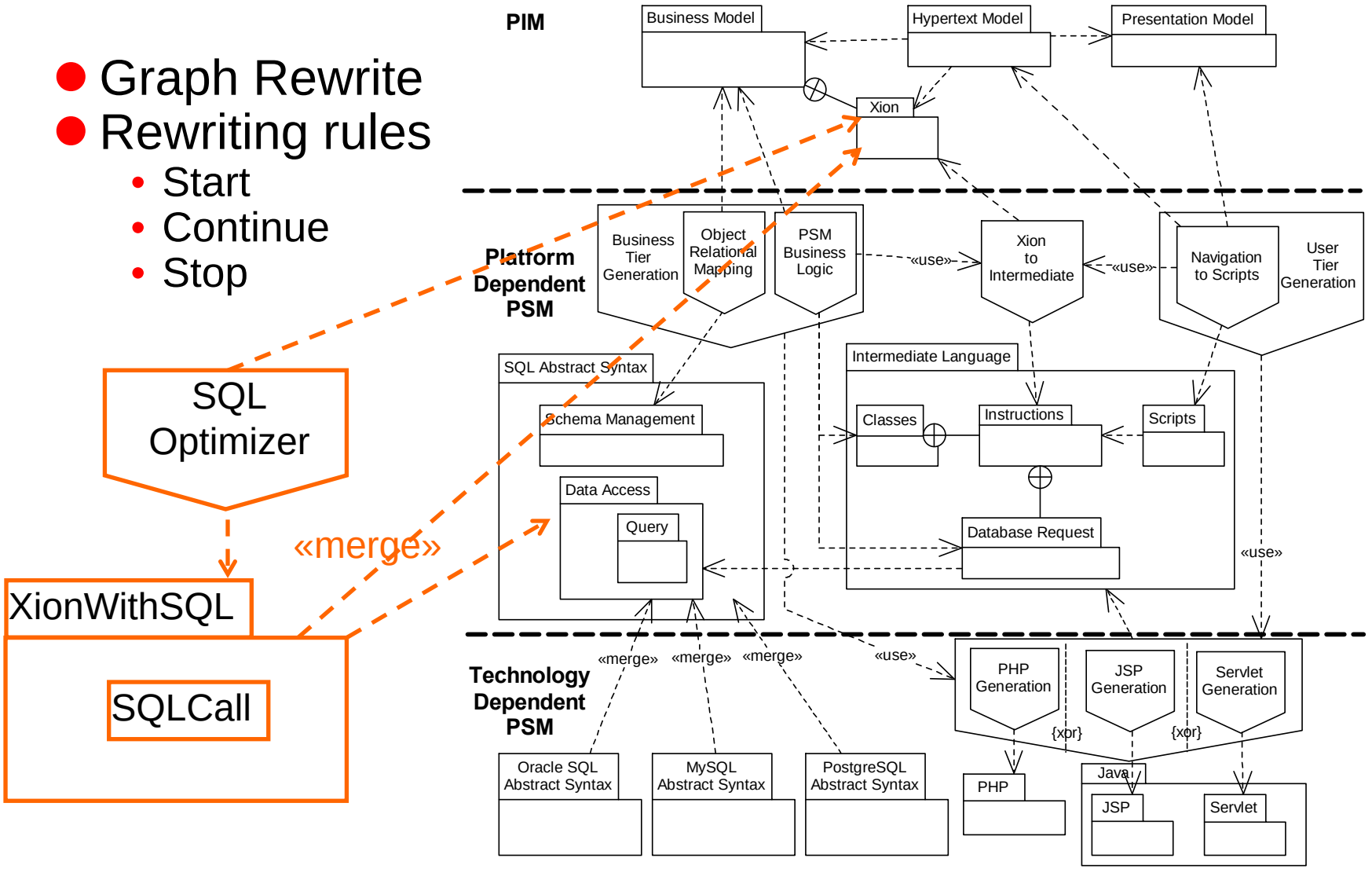
aPerson.children->collect(name)

```
SQL_exec(
    ' SELECT person.name
      FROM person
      WHERE person.OID in (
            SELECT parents_children.children
            FROM parents_children
            WHERE parents_children.parents = '
                  + aPerson.oid + ')')
```

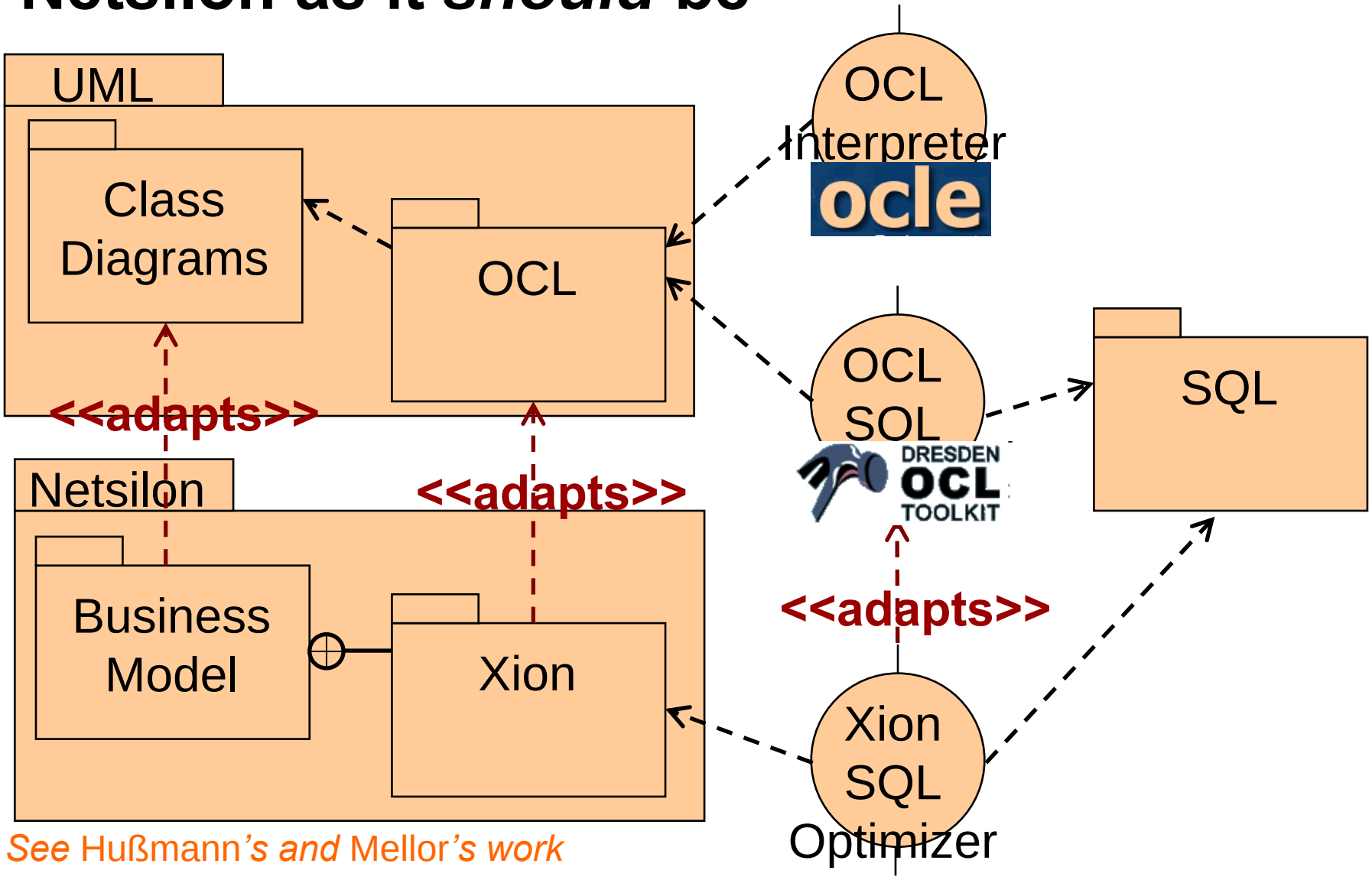
1

Generation Process

- Graph Rewrite
- Rewriting rules
  - Start
  - Continue
  - Stop



Netsilon as it *should* be



See Hußmann*'s and* Mellor*'s work*

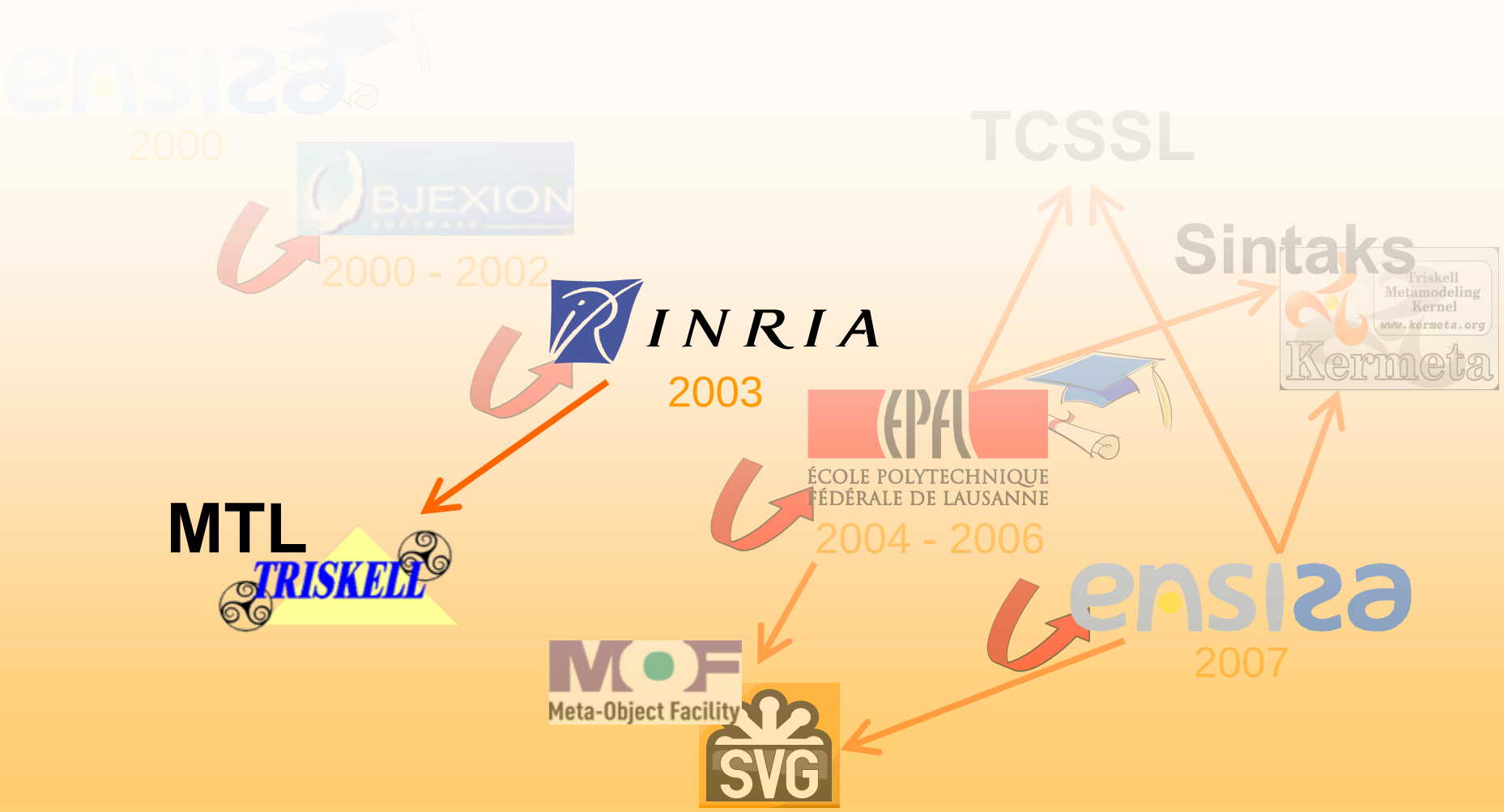
Paper

- Pierre-Alain Muller, Philippe Studer, Frédéric Fondement, and Jean Bézivin, **Platform independent web application modeling and development with Netsilon**., Software and System Modeling (SoSyM) 4 (2005), no. 4, pp. 424–442.

Contents

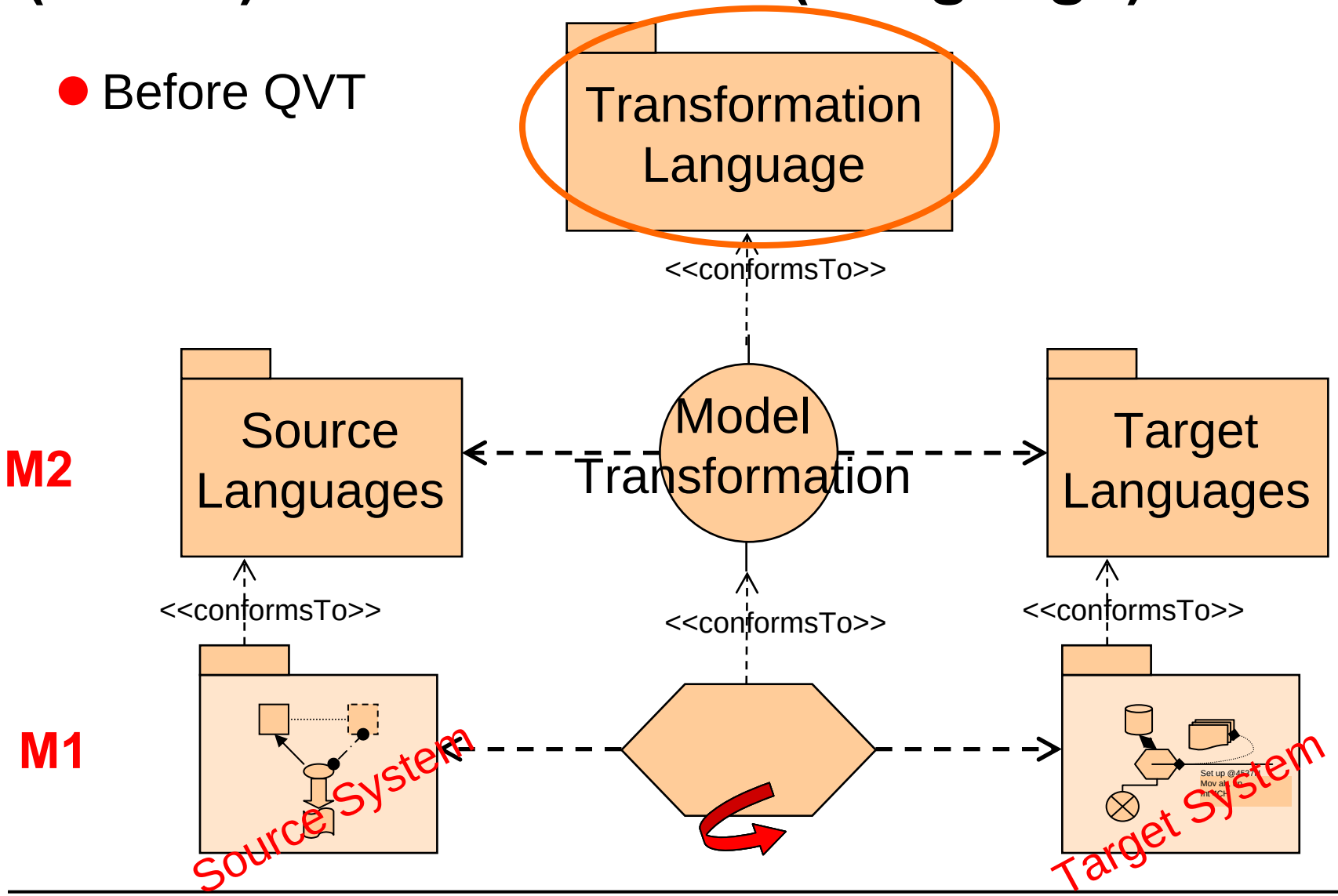
- Model Driven Engineering
- The Netsilon Experience
  - Principles
  - Implementation
- Transformation: MTL
  - Principles
  - Implementation
- Modeling Languages: Concrete Syntax
  - Textual
  - Graphical
- Reuse

Principles


2000
2000 - 2002
INRIA
2003
TCSSL
Sintaks
Triskell Metamodeling Kernel
www.kermeta.org
Kermeta
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
2004 - 2006
MTL
TRISKELL
Meta-Object Facility
SVG
2007

(Model) Transformation (Language)

- Before QVT



MTL General Principles

MTL = Object-Oriented Modular Transformation Lang.:

« Old » well-known techniques

- OCL
  - One of the best solution for model querying (cf. Xion)
  - Standard library
  - Object Oriented
- Side effects
  - Model modification
  - MTL objects modification
- Structure
  - UML class diagrams

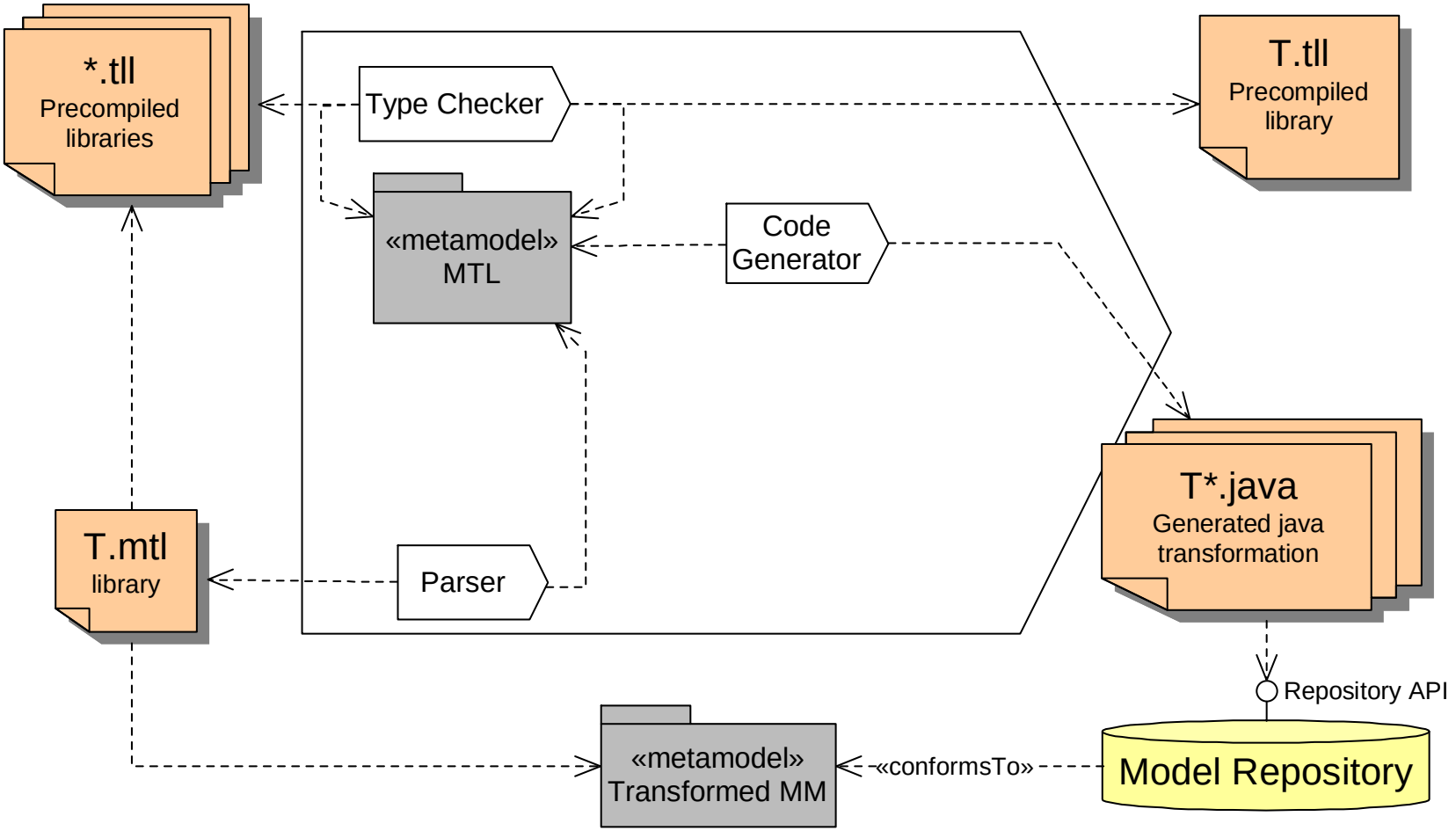
The MTL specificity

- MTL Libraries
  - Models to be manipulated as parameters
    - Including MTL Models
  - Inheritance (higher-order hierarchies)

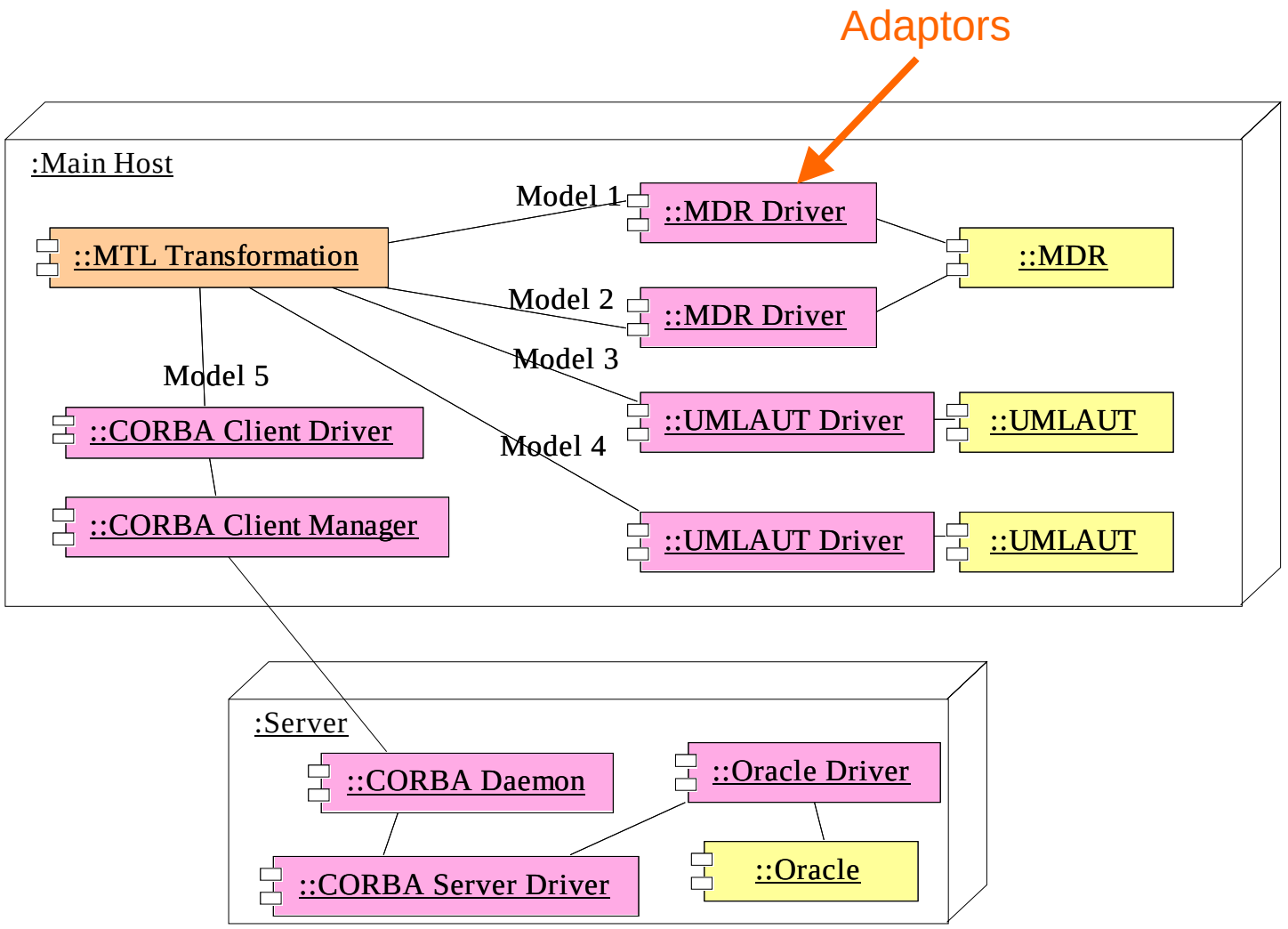
Key Points

- **Transformation reuse**
- Multi-model manipulation
  - One kind of model is a set of MTL instances
- **Independency from the model repositories**
- **Improvement process**
- Compilation of all concepts to Java
  - Multiple Inheritance
  - Higher-order hierarchies

Compiler Insight

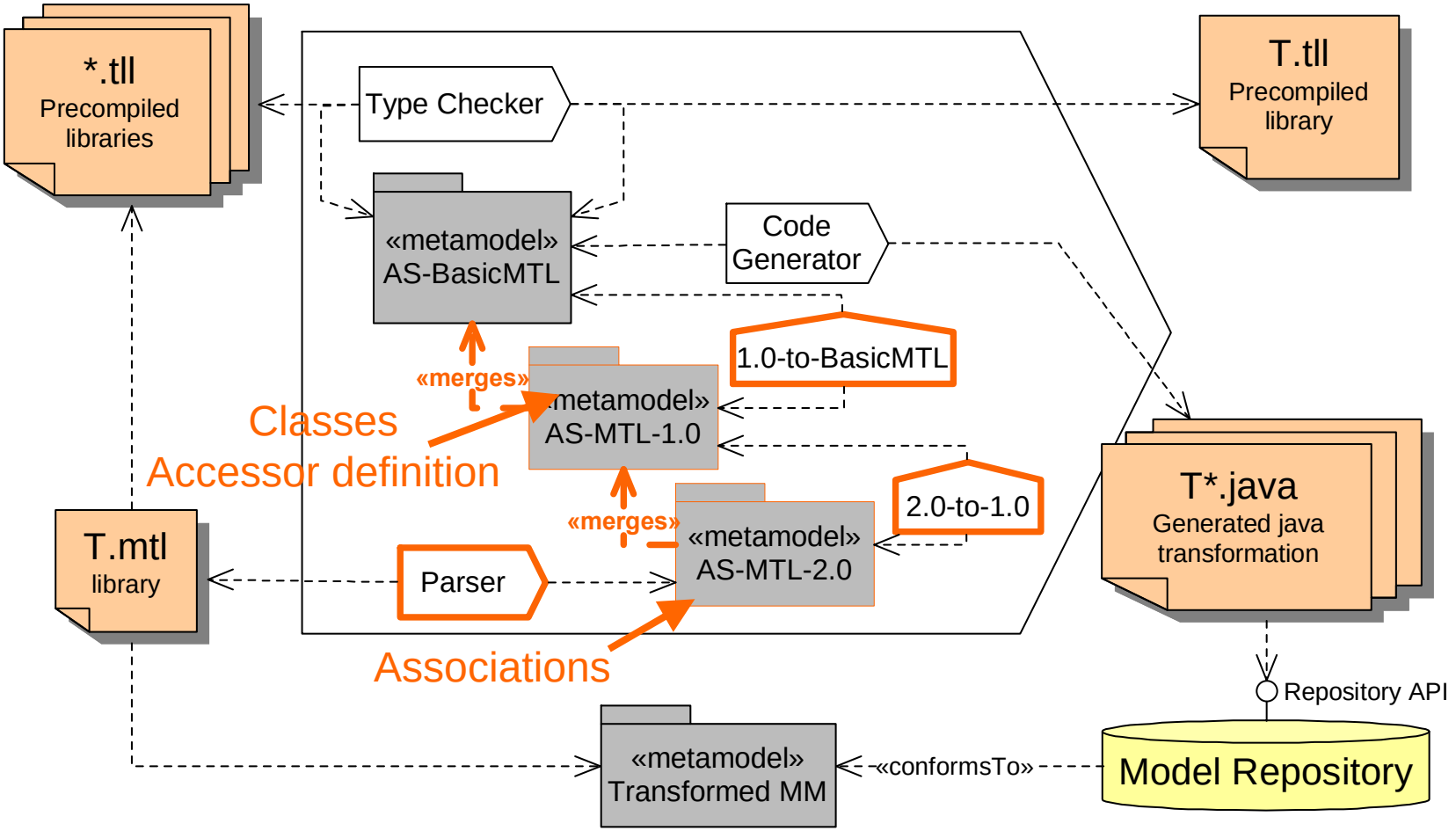


Accessing Models



MTL Improvement Process

- (MTL) transformations



Lessons Learned

- Inspired from Netsilon (BM + Xion)
  - Business Model M1=>M2
- Evolved into Kermeta
  - Kermeta MM is an extension of ECORE
- Lessons learned
  - Power of higher-order hierarchies (merge / refines)
  - Problems to write adaptors
    - In MTL
    - Repository Drivers
  - Lack of parser evolutivity

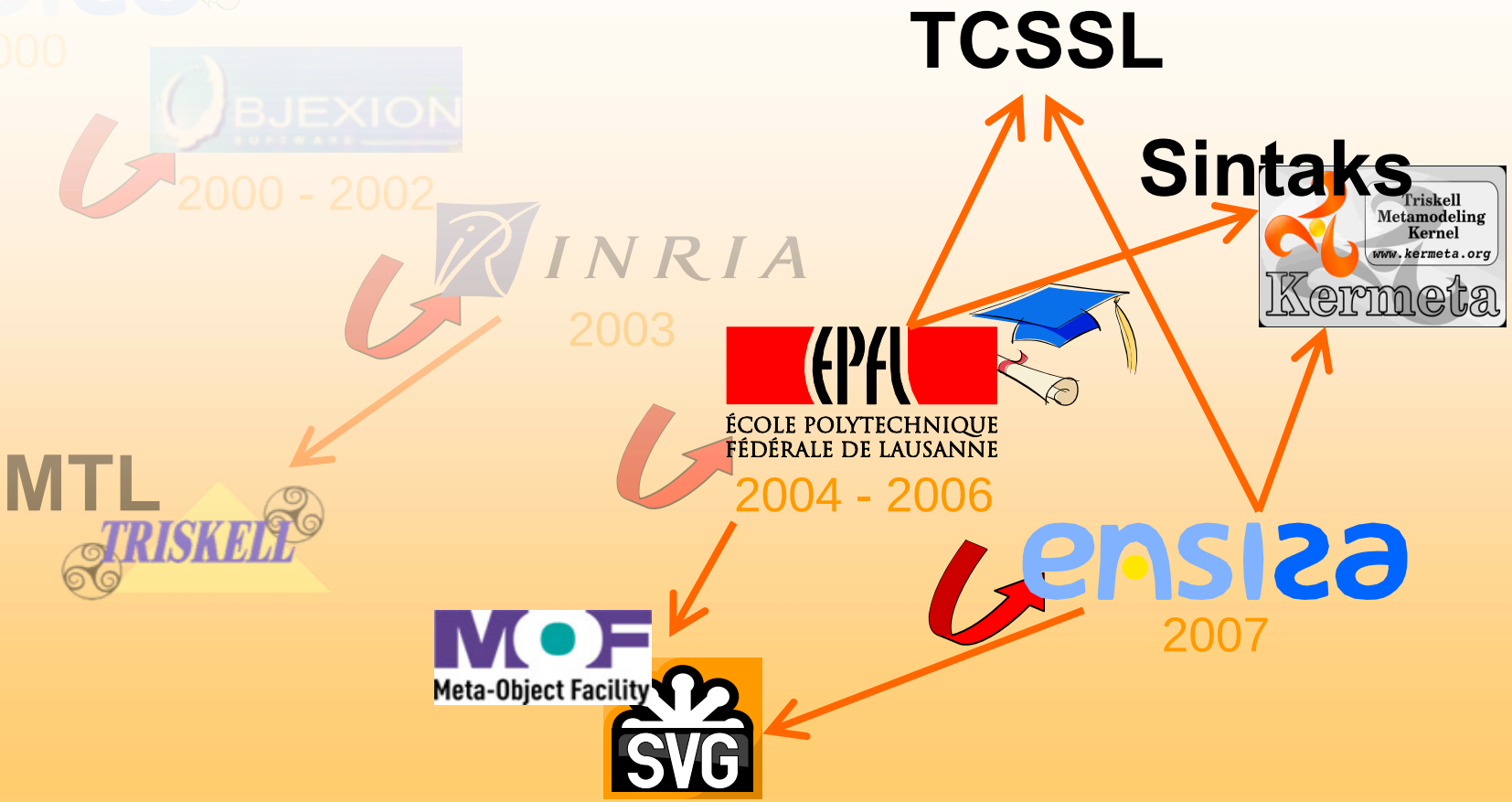
Paper

- Pierre-Alain Muller, Franck Fleurey, Didier Vojtisek, Zoé Drey, Damien Pollet, Frédéric Fondement, Philippe Studer, and Jean-Marc Jézéquel, **On executable meta-languages applied to model transformations**., Model Transformations In Practice Workshop, satellite event of the MoDELS 2005 Conference, Montego Bay, Jamaica, October 3rd, 2005.

Contents

- Model Driven Engineering
- The Netsilon Experience
  - Principles
  - Implementation
- Transformation: MTL
  - Principles
  - Implementation
- Modeling Languages: Concrete Syntax
  - Textual
  - Graphical
- Reuse

Principles

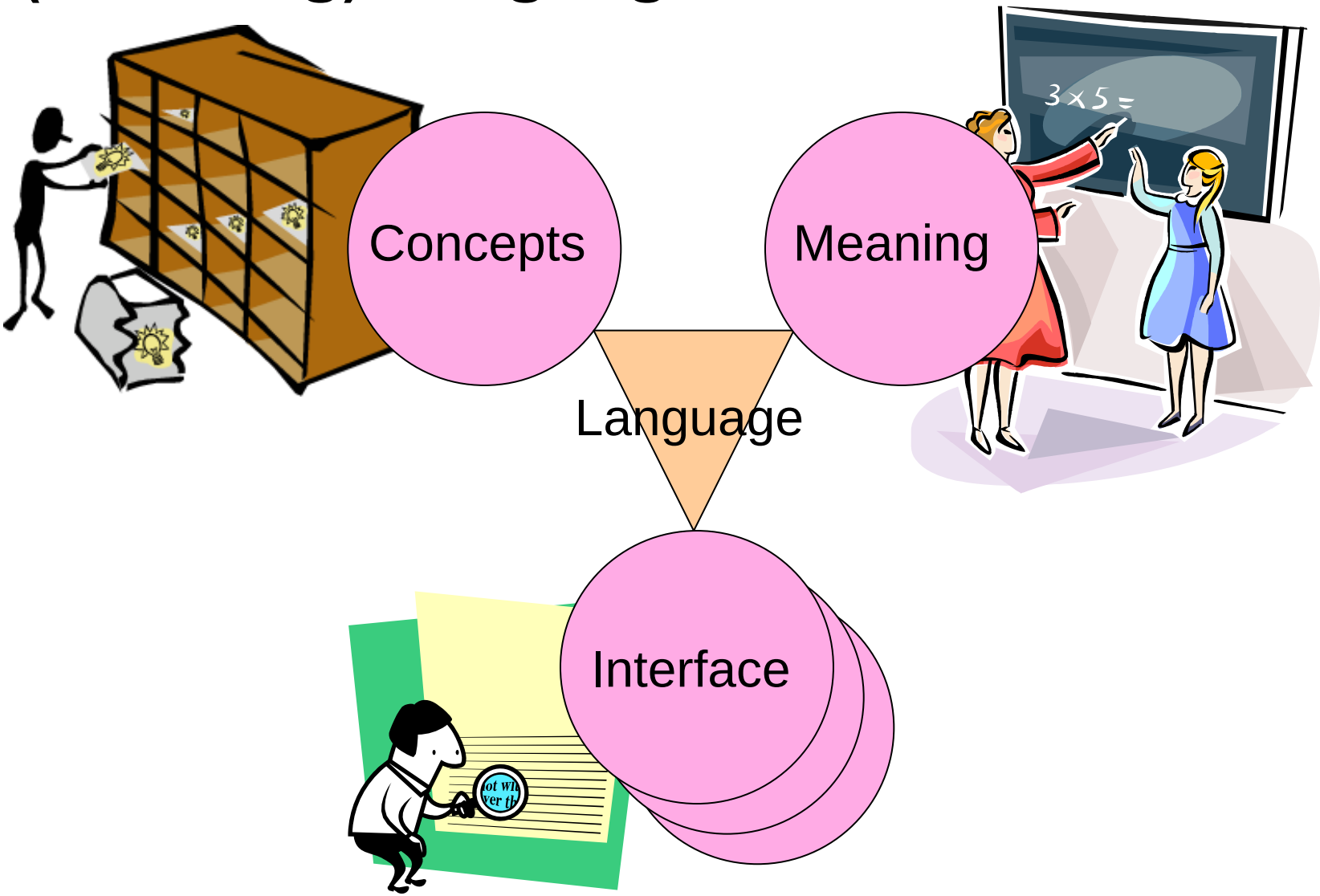

2000
BJEXION
2000 - 2002
INRIA
2003
TCSSL
Sintaks
Triskell
Metamodeling
Kernel
www.kermeta.org
Kermeta
EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
2004 - 2006
MTL
TRISKELL
ensisa
2007
MOF
Meta-Object Facility
SVG

Context

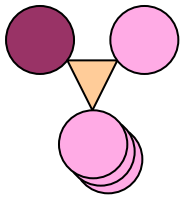
- Development of Netsilon
  - Xion
  - Hypertext Model
- Development of MTL
  - Parser
- Tooling the Fondue Method
  - Fusion + adapted UML Models

(Modeling) Language Definition

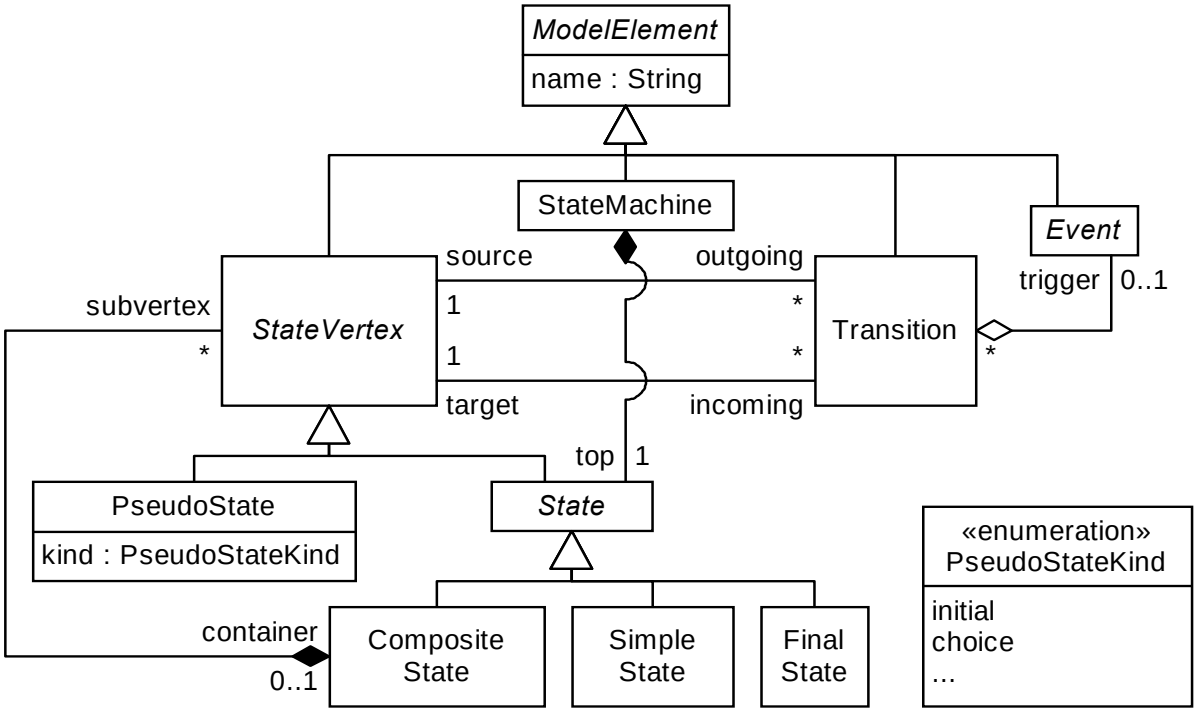
M2



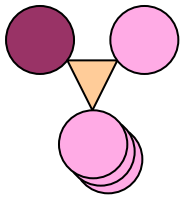
Concepts Definition



Abstract Syntax

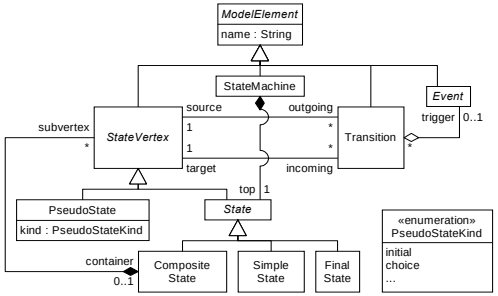


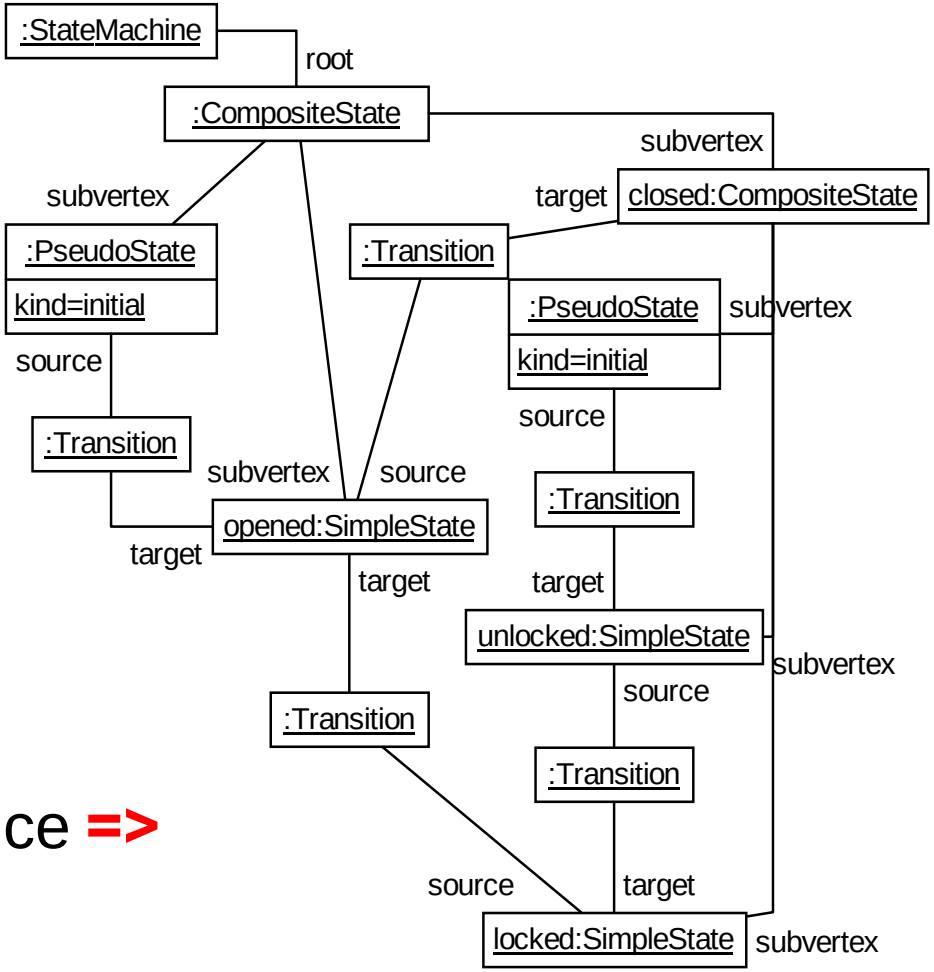
Concepts Definition



M1

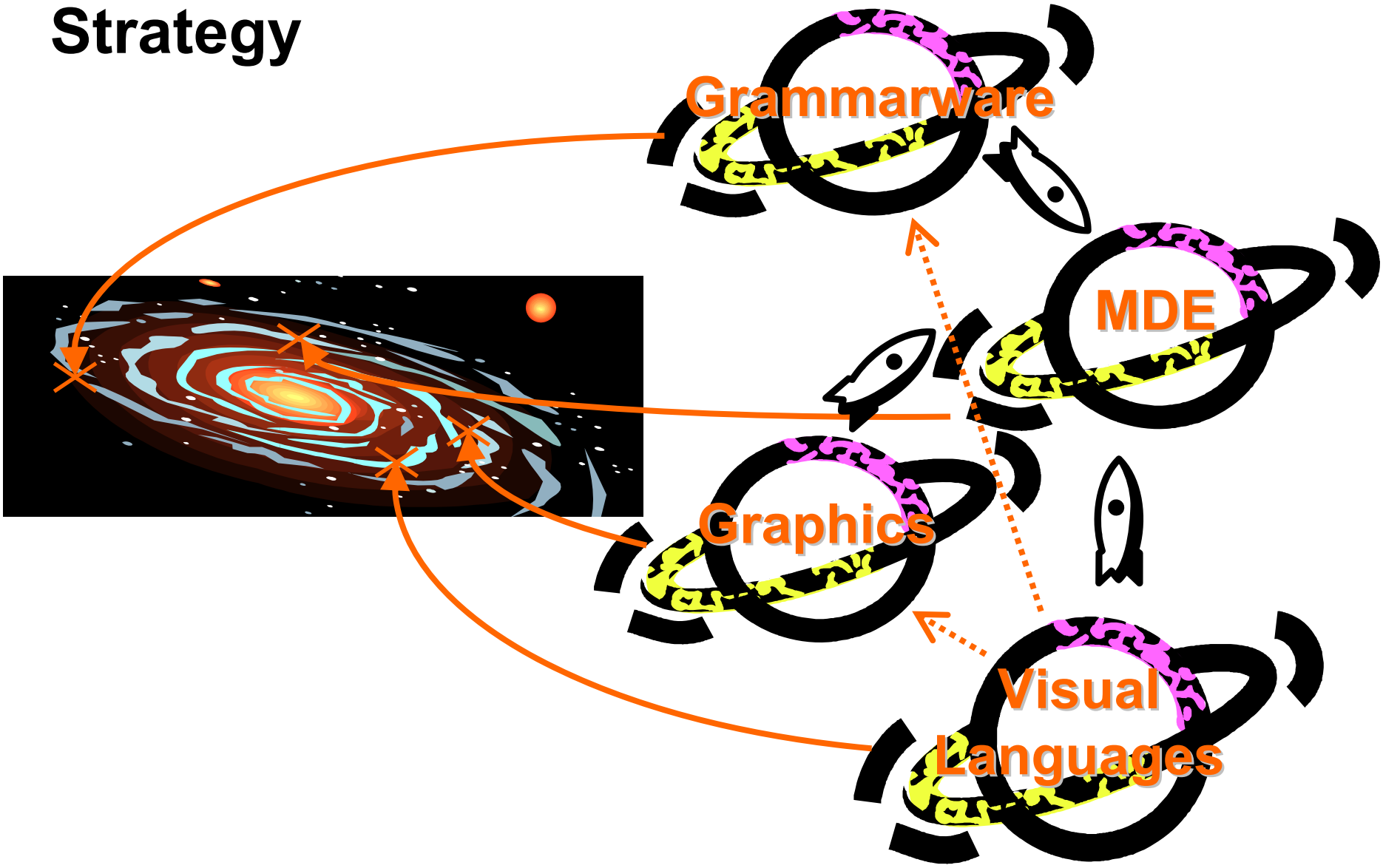
Abstract Syntax





An (M1) sentence **=>**

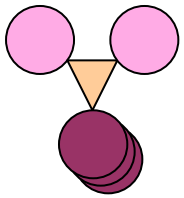
Strategy



Contents

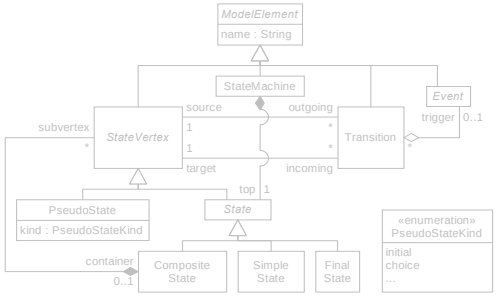
- Model Driven Engineering
- The Netsilon Experience
  - Principles
  - Implementation
- Transformation: MTL
  - Principles
  - Implementation
- Modeling Languages: Concrete Syntax
  - Textual
  - Graphical
- Reuse

Interface Definition



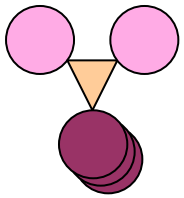
M2

Abstract Syntax + Concrete Syntax(es)



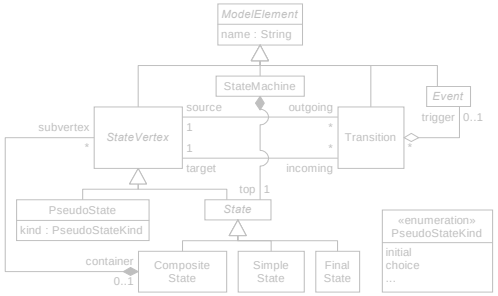

```
sm ::= "Statemachine" IDENT compositeState

state ::= normalState | pseudostate

normalState ::= "initial"? (simpleState | compositeState)

simpleState ::= "State" IDENT

compositeState ::= "CompositeState IDENT? LCURLYBRACKET
                       (state | transition)* RCURLYBRACKET

transition ::= "Transition" IDENT? "from" IDENT
                     "to" IDENT ("on" IDENT)?

pseudoState ::= "FinalState" IDENT | "Choice" IDENT
```

Interface Definition



M1

Abstract Syntax + Concrete Syntax(es)




```
sm ::= "Statemachine" IDENT compositeState

state ::= normalState | pseudostate

normalState ::= "initial"? (simpleState | compositeState)

simpleState ::= "State" IDENT

compositeState ::= "CompositeState IDENT? LCURLYBRACKET
                   (state | transition)* RCURLYBRACKET

transition ::= "Transition" IDENT? "from" IDENT
                  "to" IDENT ("on" IDENT)?

pseudoState ::= "FinalState" IDENT | "Choice" IDENT
```

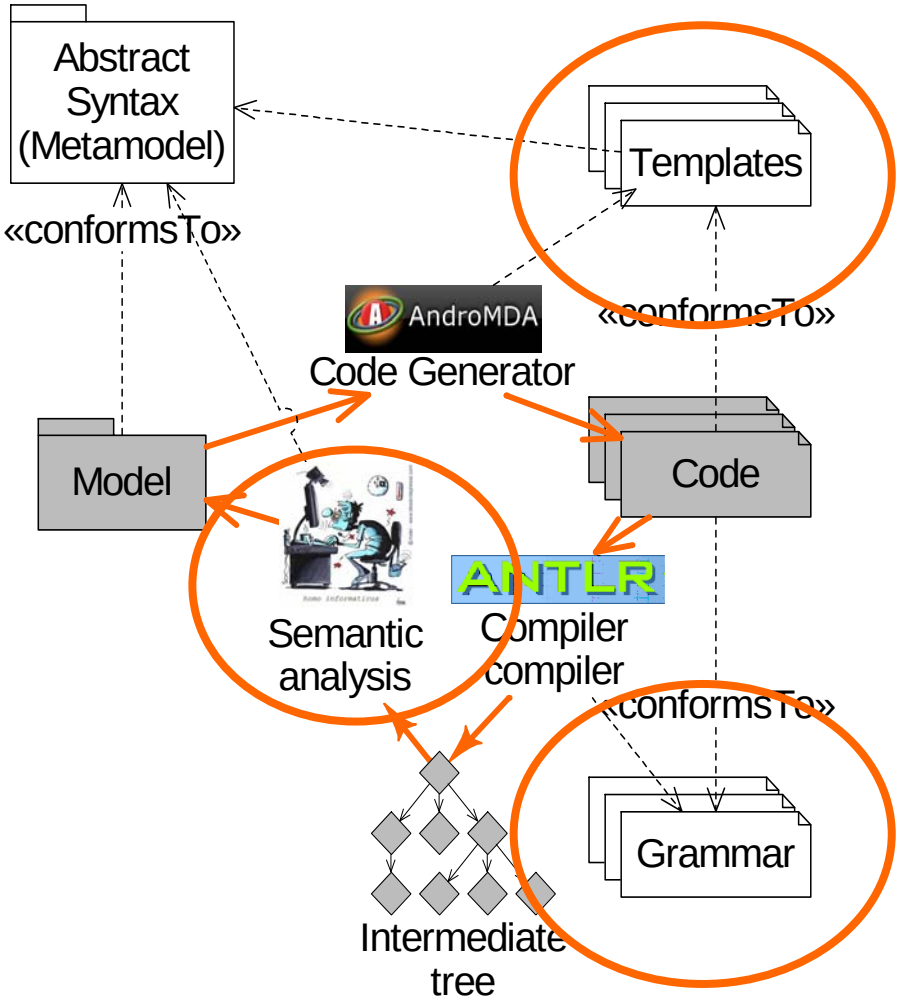
StateMachine Door
CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    Transition from unlocked
      to locked on lock

<= An (M1) sentence

...

Typical Implementation

White: M2
Grey: M1

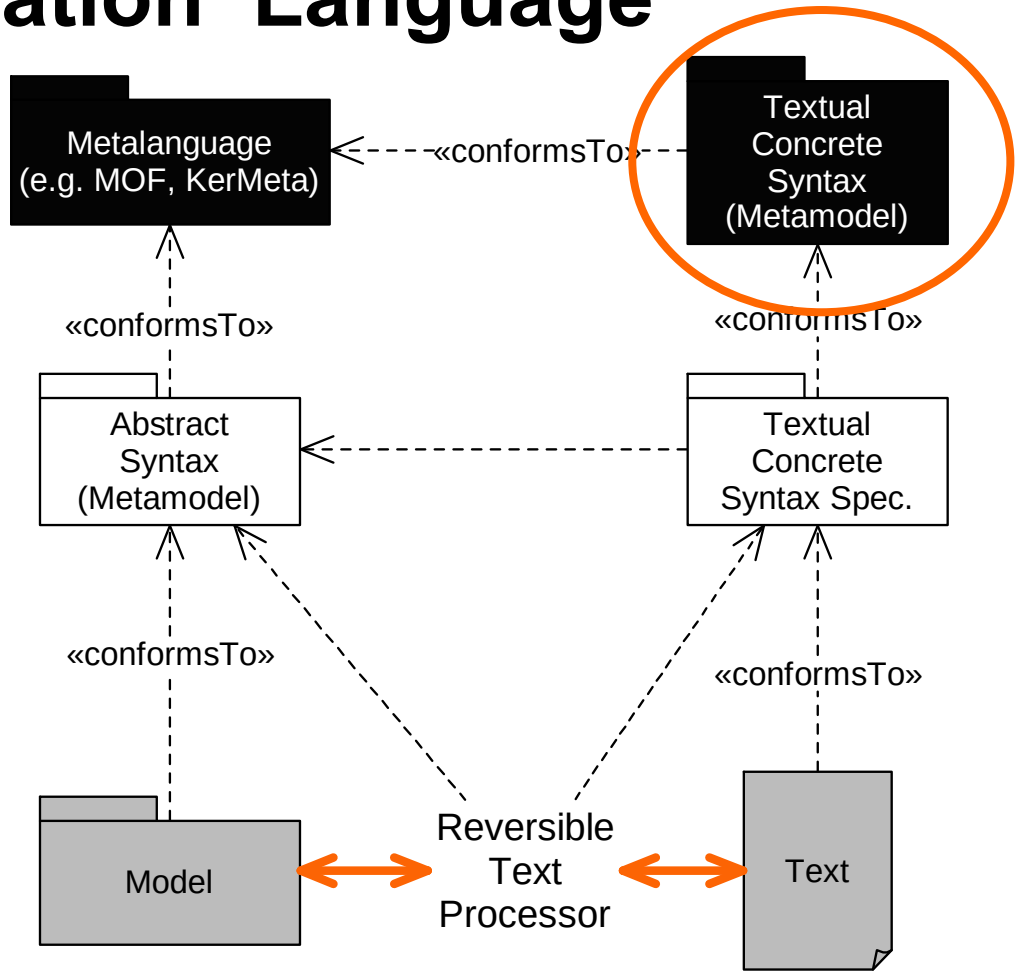


Key Points

- Reversible process
- Unique specification
- Tradeoff between
  - Complexity of the definition
  - Flexibility of the possible concrete syntaxes

Specification Language

Black: M3
White: M2
Grey: M1

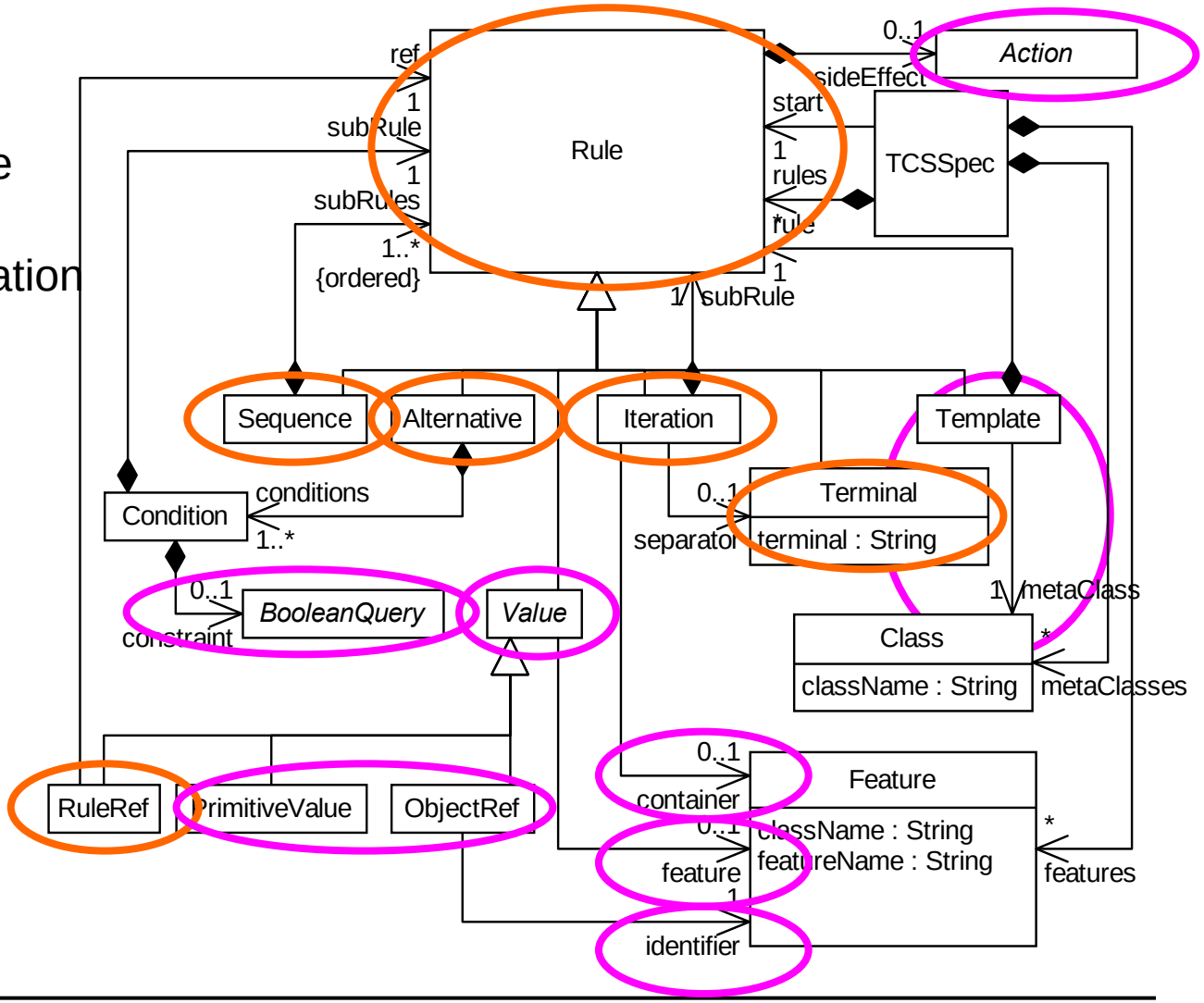


- No concrete syntax
  - Help yourself !

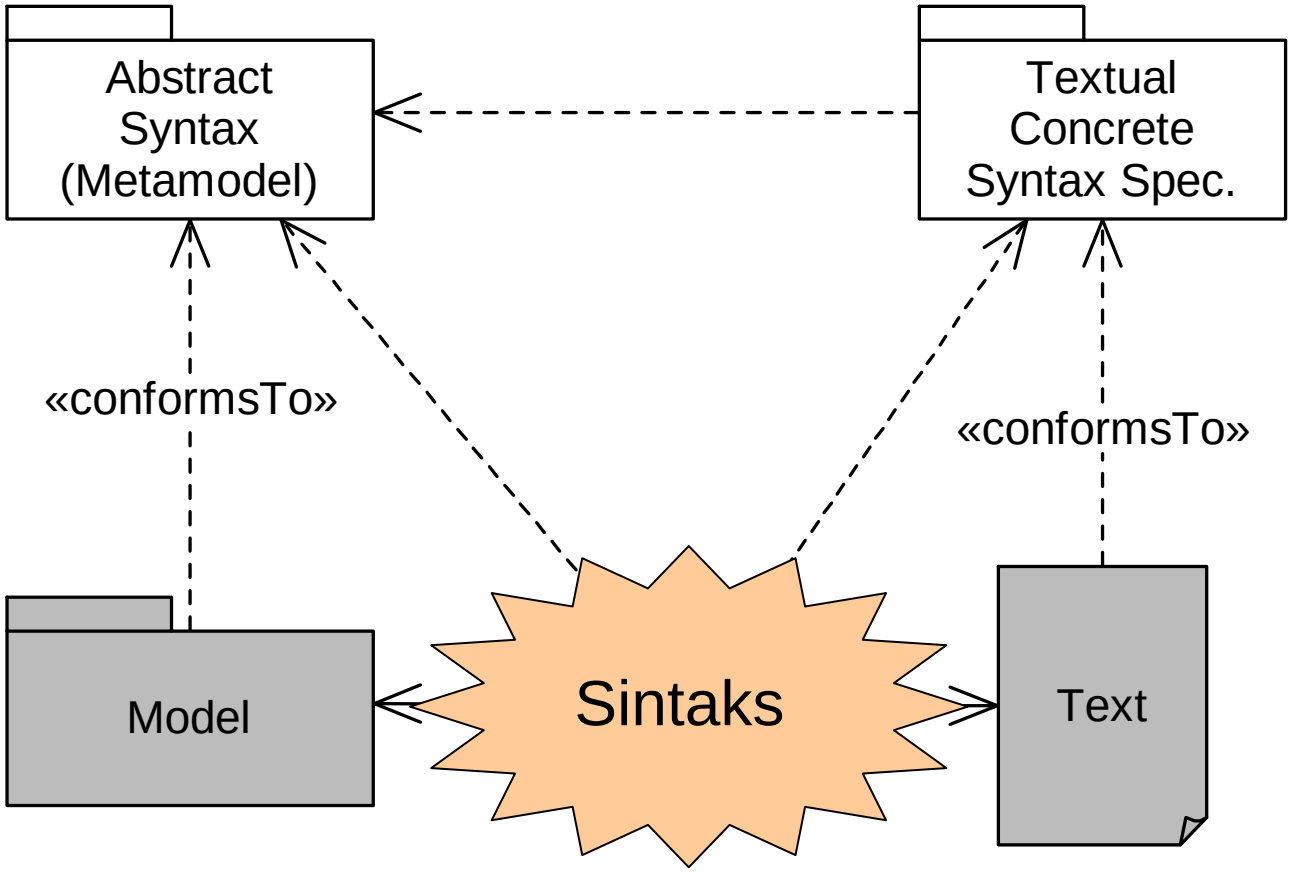
TCSSpec Metamodel

Inspired from

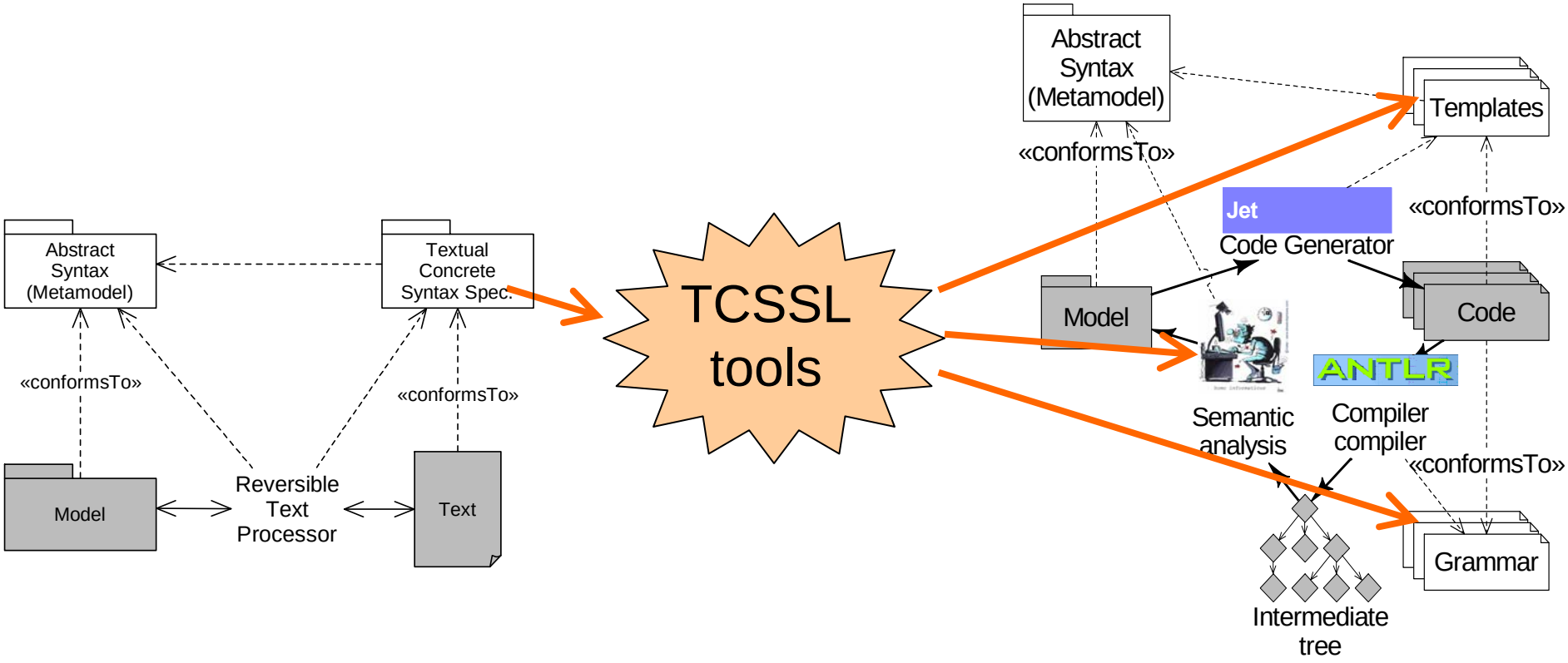
- EBNF
  - Text structure
- Netsilon
  - Model Navigation



Prototypes: Sintaks



Prototypes: TCSSL Tools



Papers

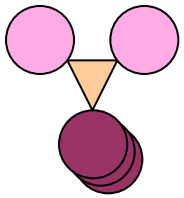
- Pierre-Alain Muller, Frédéric Fondement, Franck Fleurey, Michel Hassenforder, Rémi Schnekenburger, Sébastien Gérard, and Jean-Marc Jézéquel, **Model-driven analysis and synthesis of textual concrete syntax**., Software and System Modeling (SoSyM), 2008, (DOI: 10.1007/s10270-008-0088-x - to appear).

- Frédéric Fondement, Rémi Schnekenburger, Sébastien Gérard and Pierre-Alain Muller: **Metamodel-Aware Textual Concrete Syntax Specification**, Technical Report LGL-REPORT-2006-005, Swiss Federal Institute of Technology in Lausanne, Switzerland, December 2006.

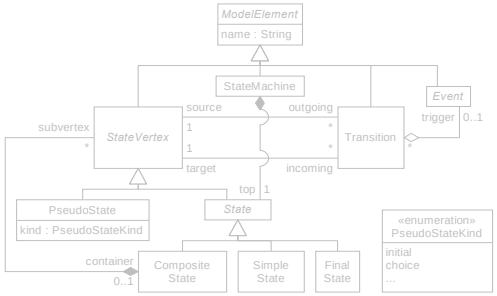
Contents

- Model Driven Engineering
- The Netsilon Experience
  - Principles
  - Implementation
- Transformation: MTL
  - Principles
  - Implementation
- Modeling Languages: Concrete Syntax
  - Textual
  - Graphical
- Reuse

Interface Definition

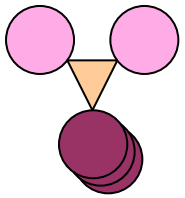


Abstract Syntax + Concrete Syntax(es)



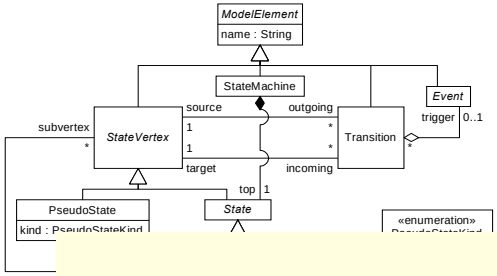
Transition	SimpleState	Composite State	FinalState	PseudoState (initial)	PseudoState (choice)
–event⟩	name	name / contents	◉	●	○

Interface Definition

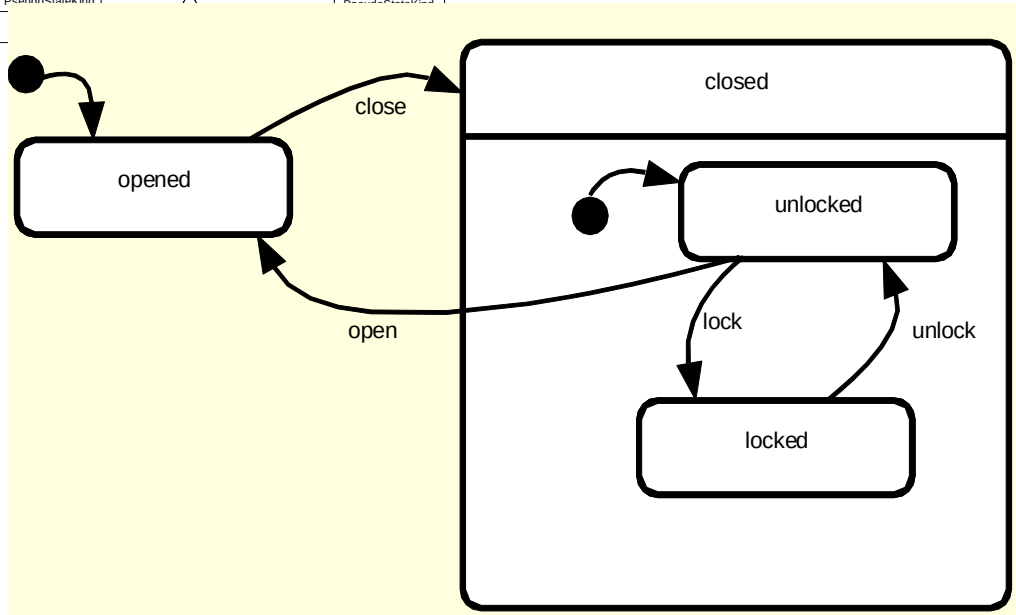


M1

Abstract Syntax + Concrete Syntax(es)



Transition	SimpleState	Composite State	FinalState	PseudoState (initial)	PseudoState (choice)
event	name	name / contents	◉	●	○

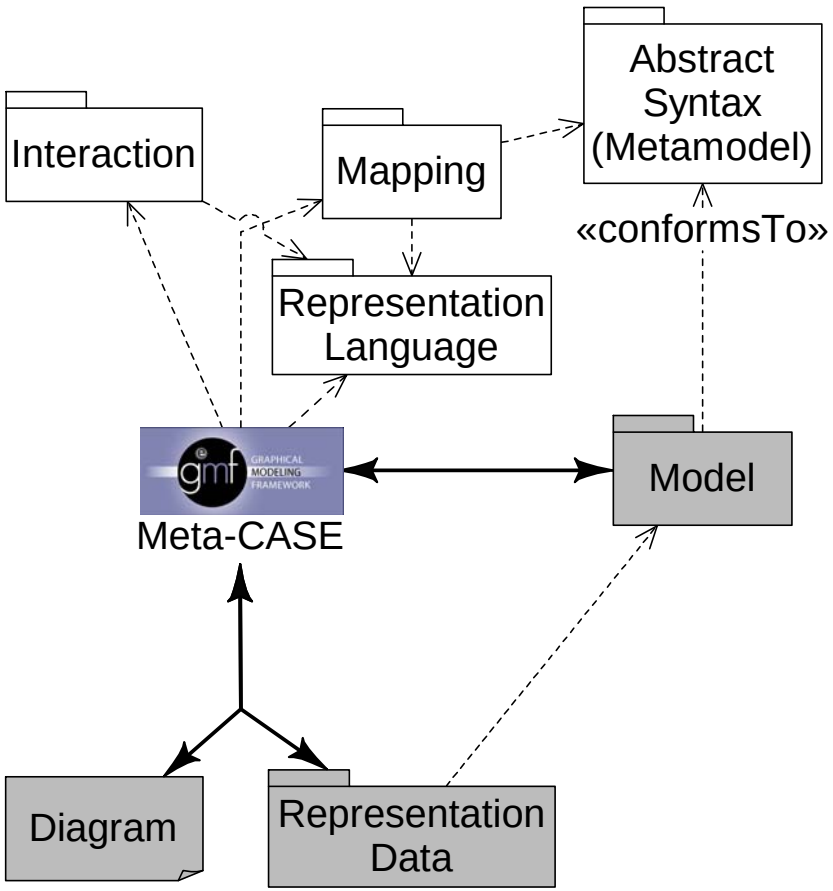


<= An (M1) sentence

- In practice
  - Layout constraints
  - User interactions

Typical Implementation

White: M2
Grey: M1



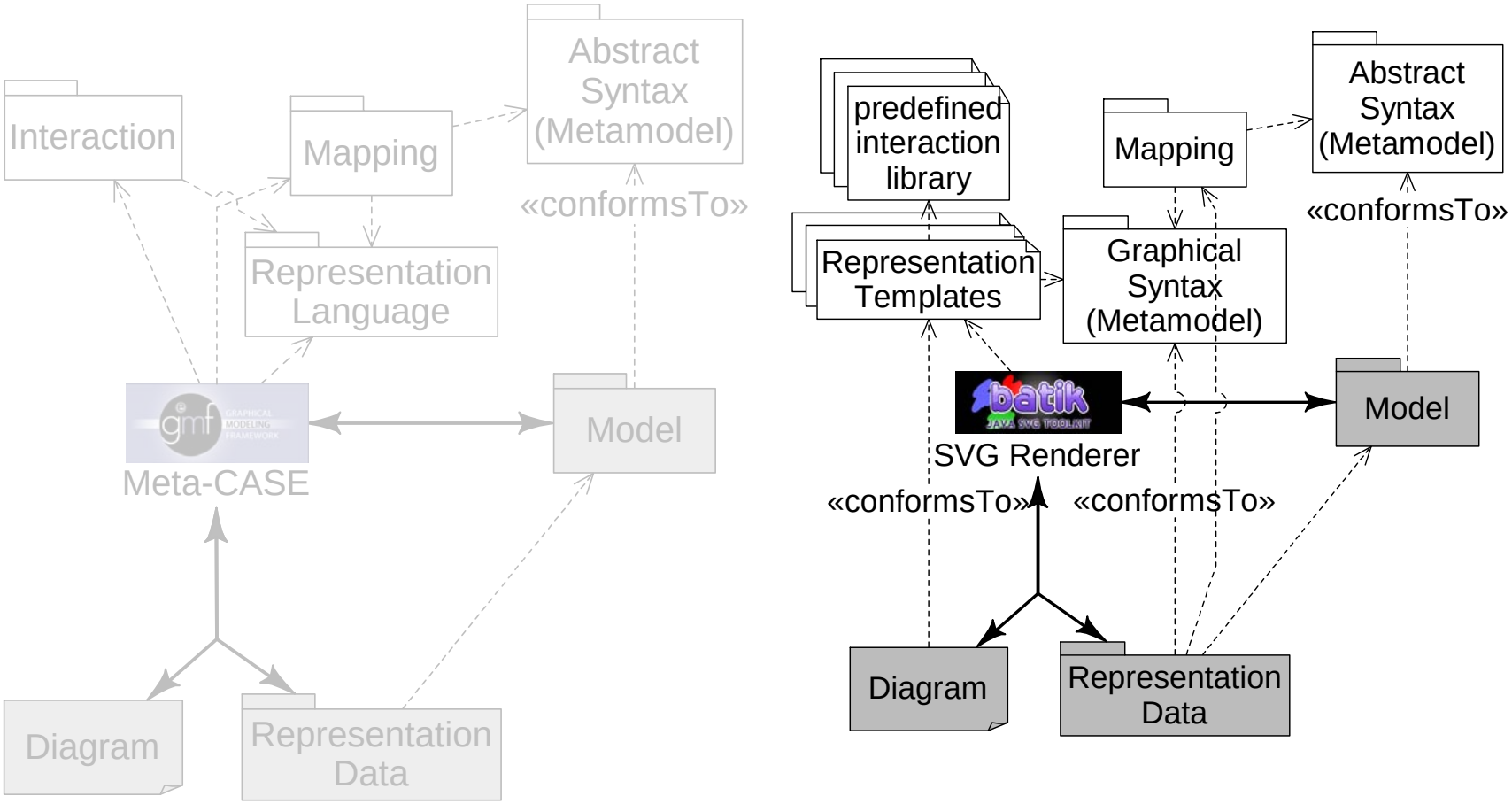
- A lot of variability
- Not dedicated to graphic designers
- Not adopted by industry yet
  - MVC with 2D graphical libraries

Key Points

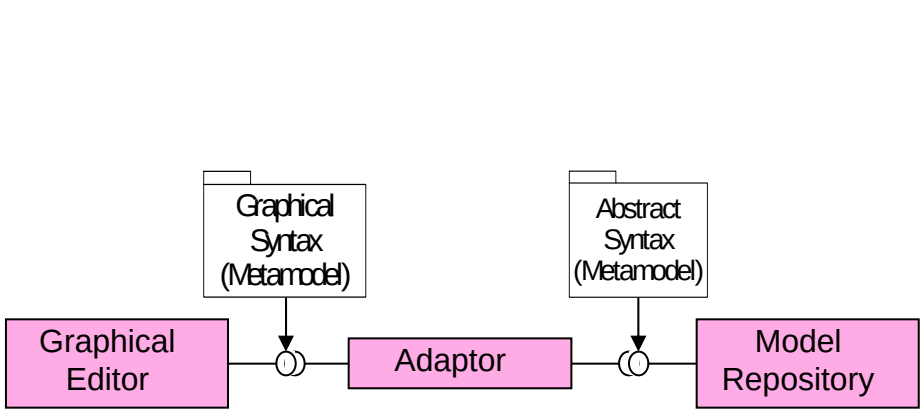
- Not limited to connection-based languages
- Reversible mapping
- Versatile representation language
- Clear representation data structure
- Library of reusable interactions

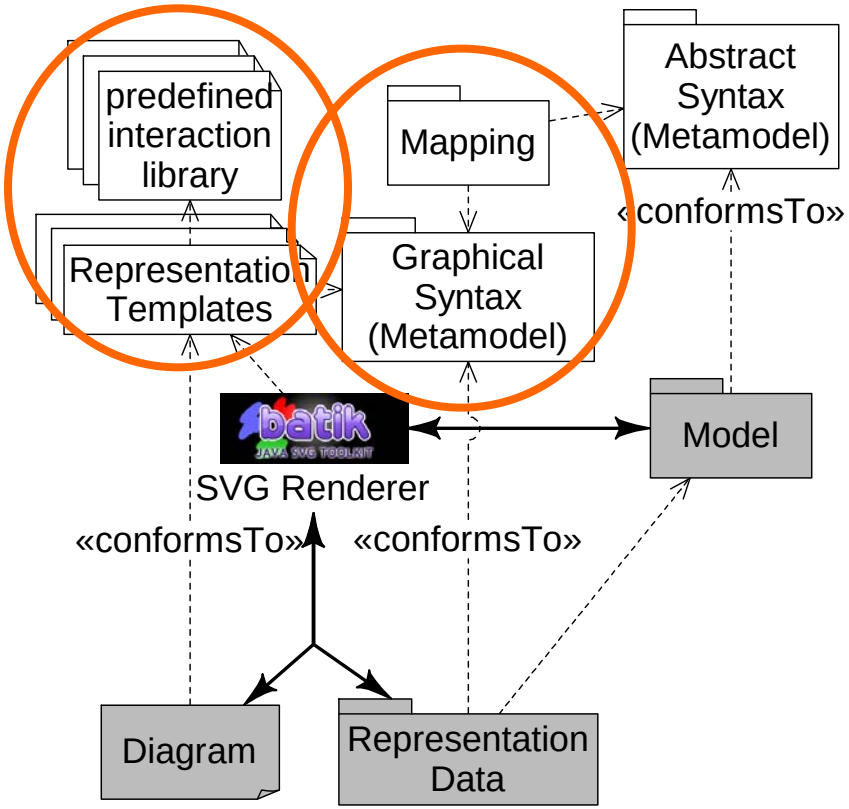
Idea

White: M2
Grey: M1



Graphical concrete syntax definition

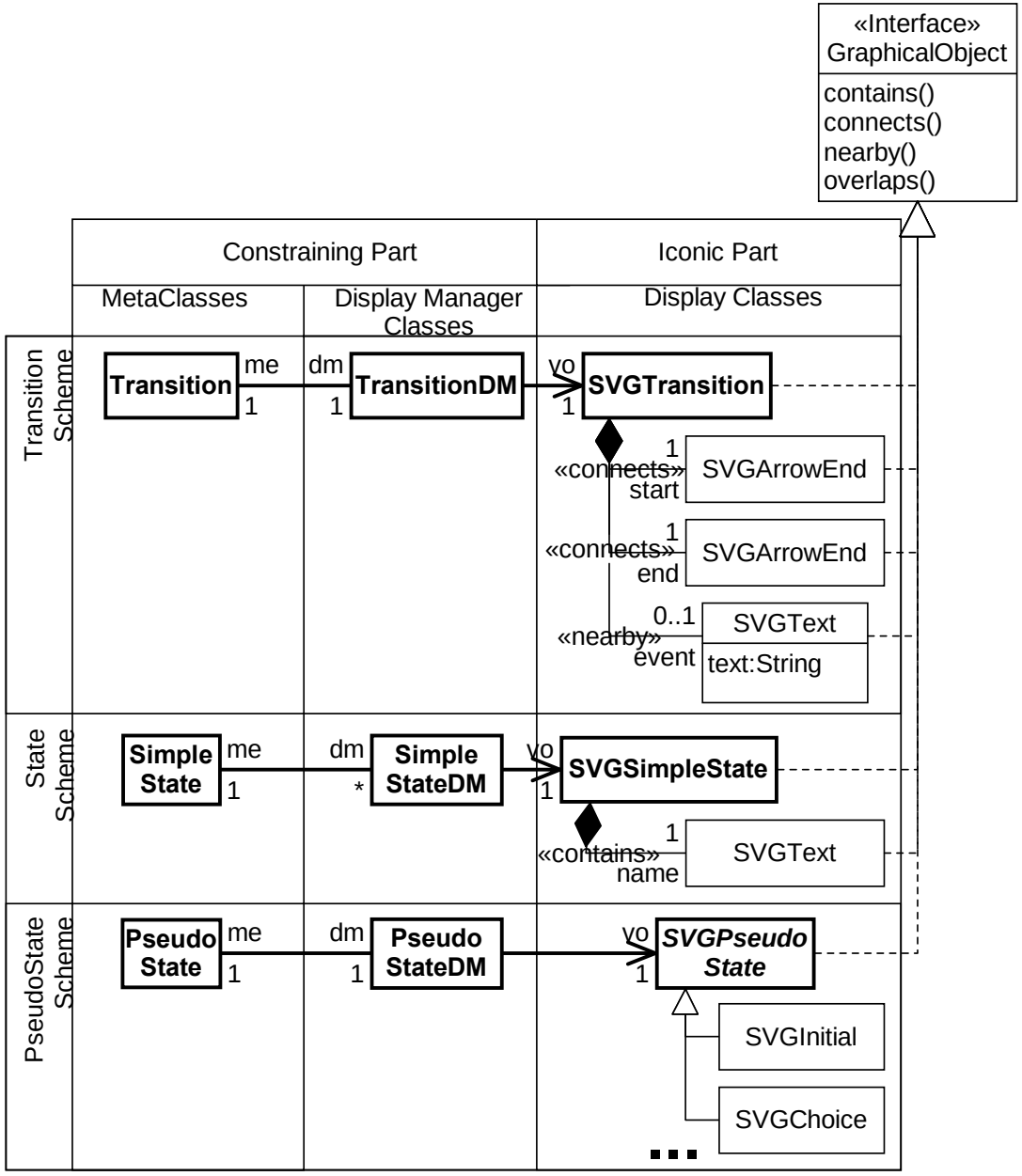




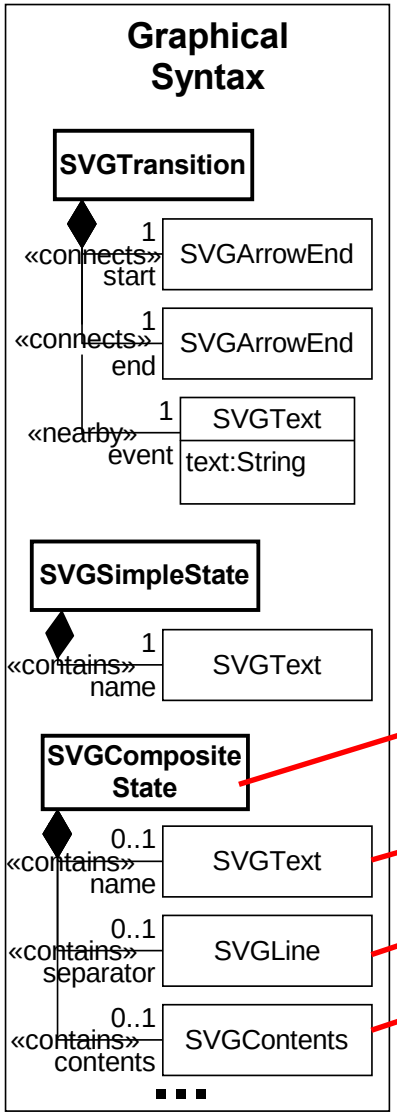
Adaptor

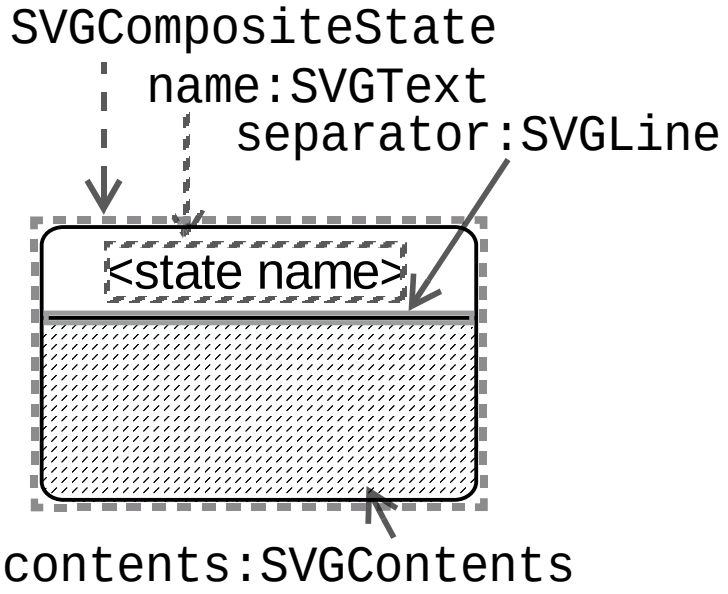
- Synchronization through OCL constraints solving
- Implementation issues
- Here an example for synchronizing abstract with a concrete syntax model

```
context TransitionDM inv:
  if self.me.trigger->isEmpty()
  then self.vo.event->isEmpty()
  else self.vo.event.text
            = self.me.trigger.name
  endif
```



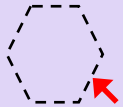
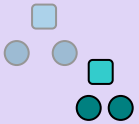
Representation






```
<svg …>
 < g id="$$" …>
   <rect id="back_$$" …/>
   <text id="name_$$ " …/>
   <line id="end_$$ " …/>
   <rect id="contents_$$" …/>
   …
 </g>
</svg>
```

User interactions

Interface		Interface	
BorderSlidable		Stickable	
DirectionAdjustable		Translatable	
Locatable		BorderFindable	
Positionable		OriginGettable	
Containable		Container	
Editable		Etc...	

See Beaudoux*'s work*

Representation Link: DopiDOM events

- Events depend on DopiDOM component
- Reaction to events defined in templates
  - Java JMI or EMF, KerMETA, Xion, MTL, etc.
- Initial / Load / Save scripts

`CompositeState` template

```
<svg
onCreation="s=model.getCompositeStateDM().createCompositeStateDM();
">
 <text name="name_$$" var_self="$s" dpi:component="Editable, …"
  onChange="self.setName(content);"…/>
 …
</svg>
```

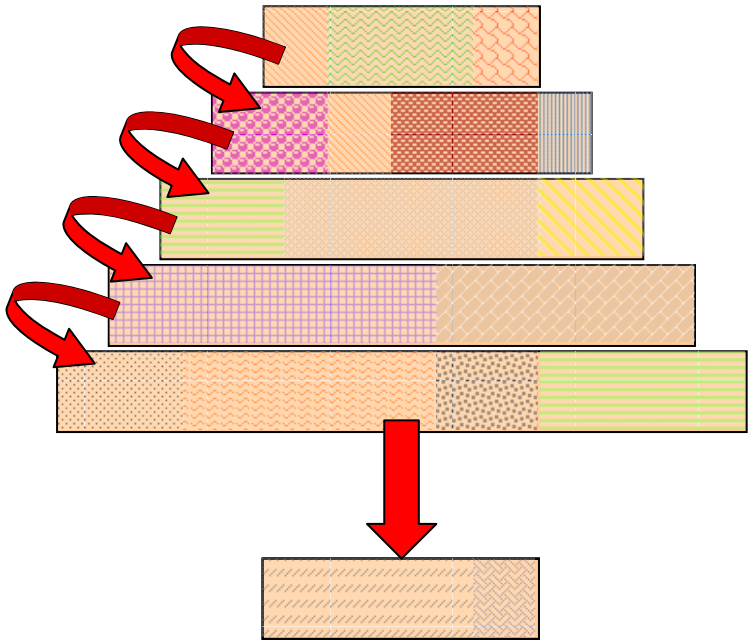
Papers

- Frédéric Fondement and Thomas Baar, **Making metamodels aware of concrete syntax**., First European Conference on Model Driven Architecture Foundations and Applications - ECMDA-FA (Alan Hartman and David Kreische, eds.), Nuernberg, Germany, November 7-10, 2005, Lecture Notes in Computer Science, vol. 3748, Springer, 2005, pp. 190–204.

- Frédéric Fondement, **Graphical concrete syntax rendering with SVG**., Fourth European Conference on Model Driven Architecture Foundations and Applications - ECMDA-FA (Philippe Desfray, Alan Hartman, Richard Paige, Arend Rensink, Andy Schürr, Regis Vogel, Jos Warmer, eds.), Berlin, Germany, June 9-12, 2008, Lecture Notes in Computer Science, Springer, 2008, to appear.

Contents

Questions Raised



Domain Specific Modeling

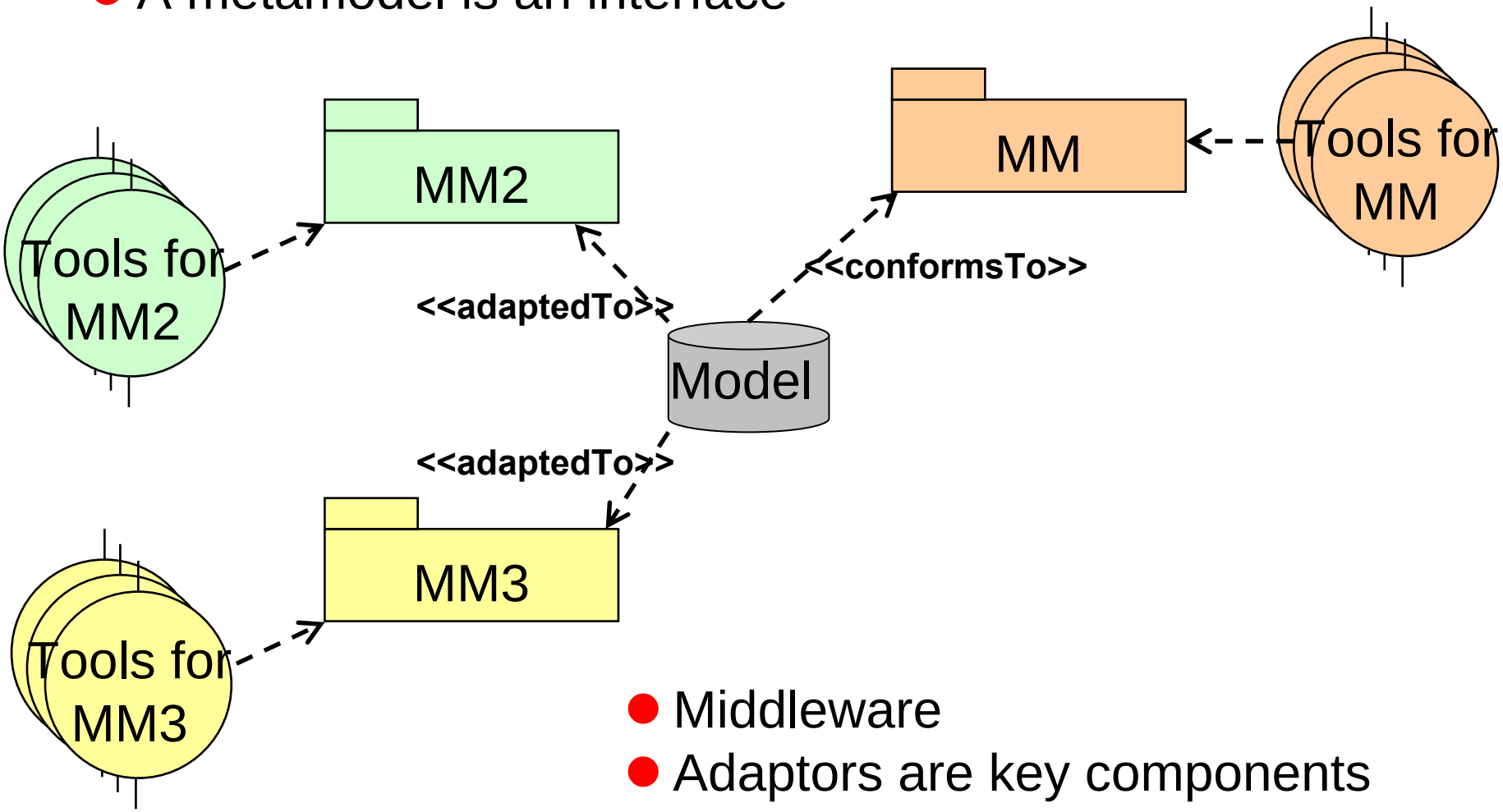
=

Proliferation of languages !

- **Support for language engineering**

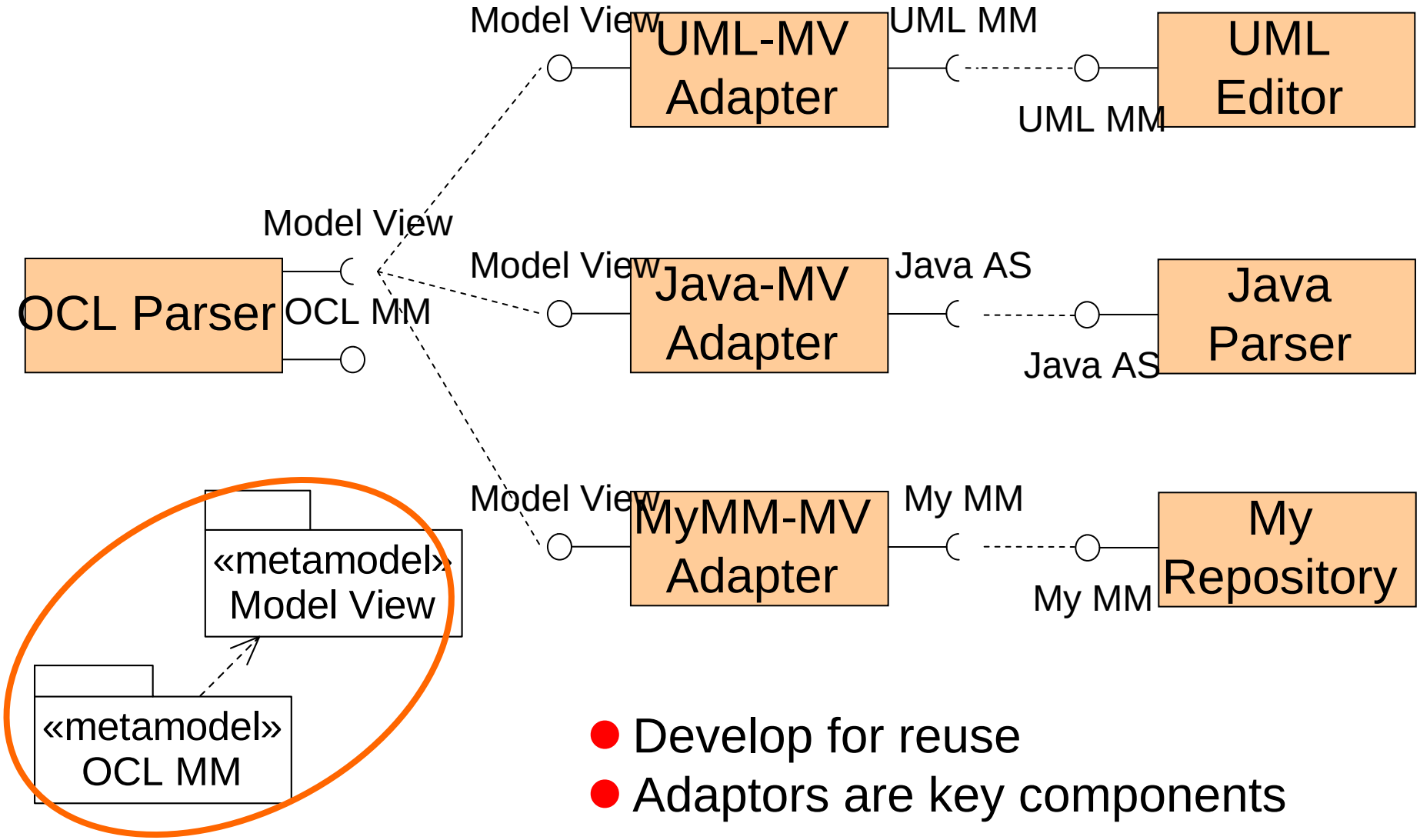
Reusing Tools

- A metamodel is an interface



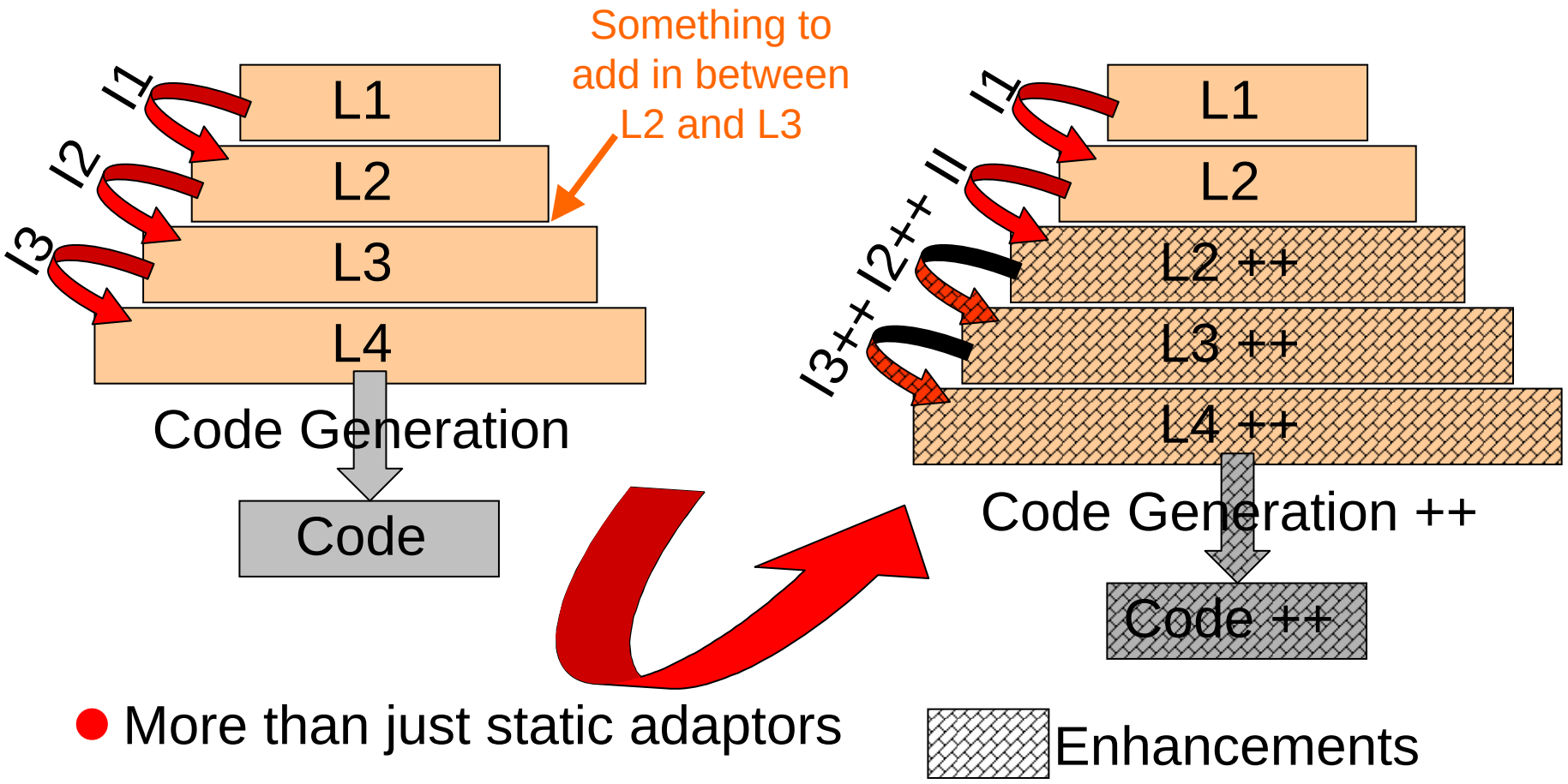
- Middleware
- Adaptors are key components

Reusing Languages



- Develop for reuse
- Adaptors are key components

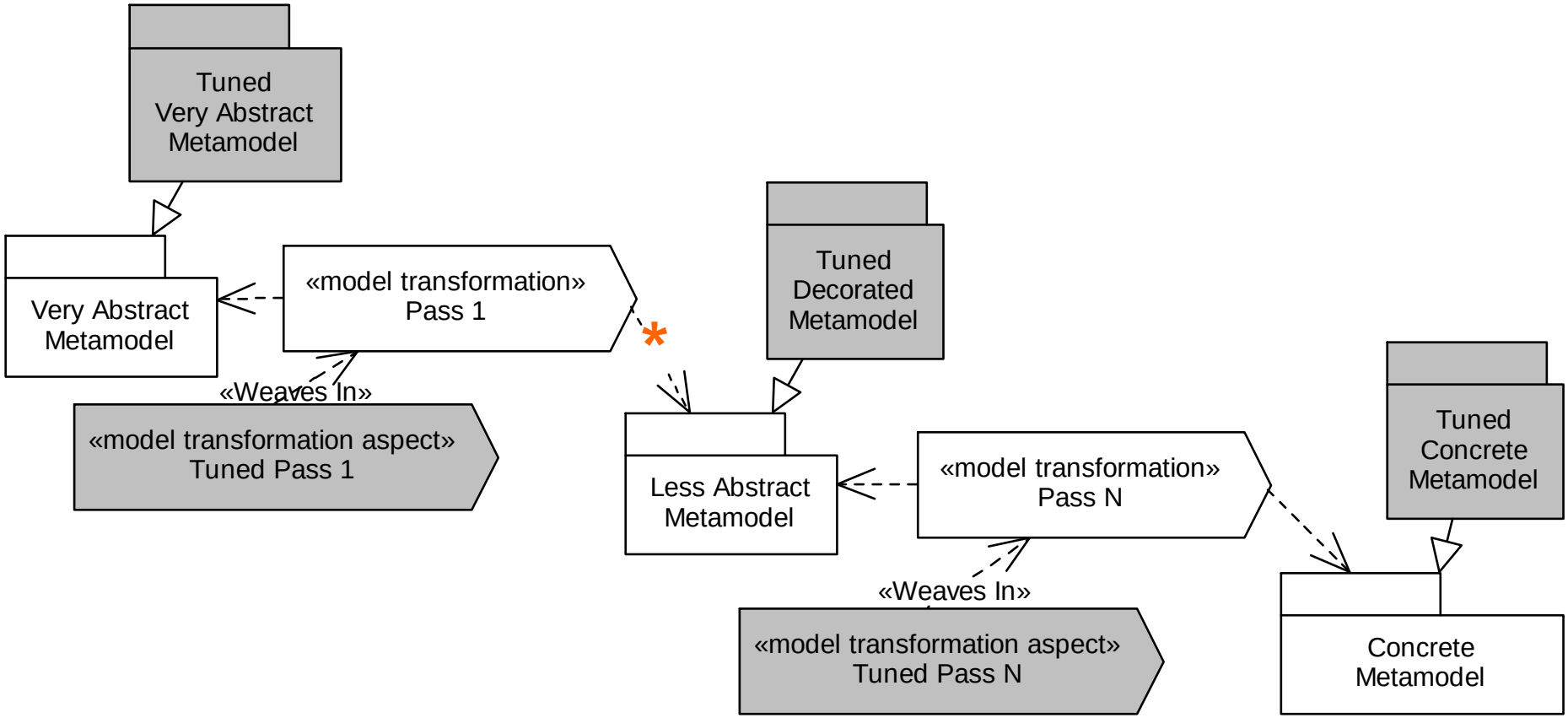
Reusing MDE Processes



- More than just static adaptors

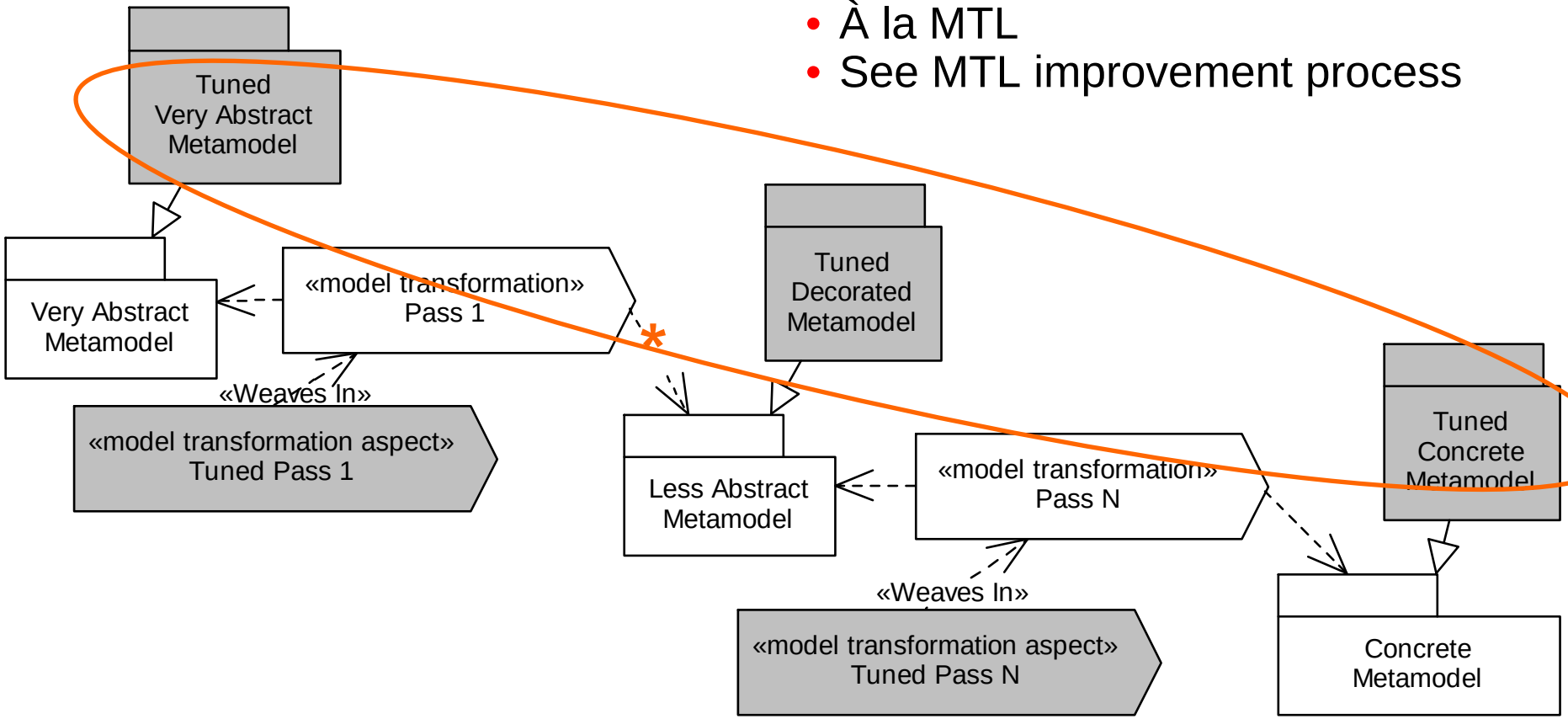
Enhancements

Tuning MDE Artefacts



Tuning MDE Artefacts

- Higher-order hierarchies
  - À la MTL
  - See MTL improvement process

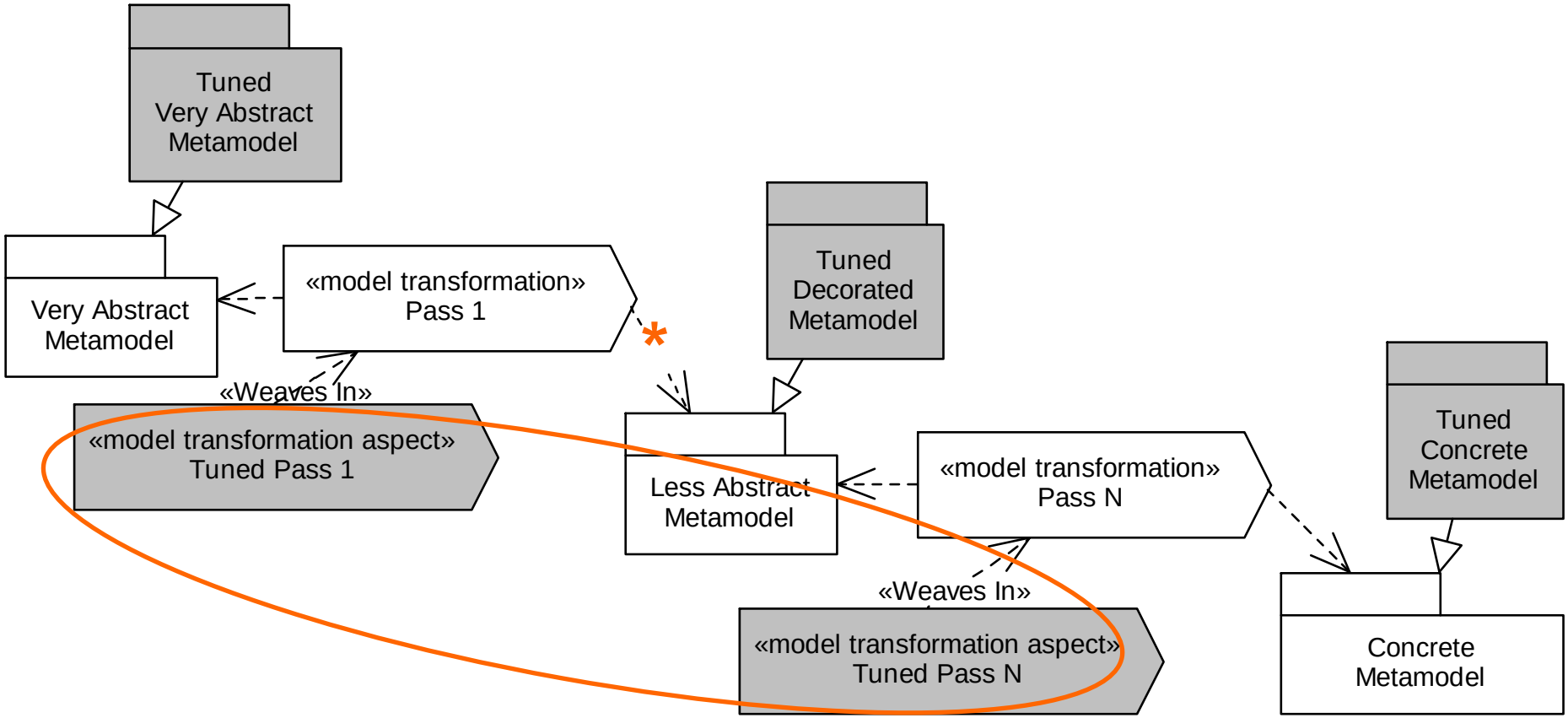


- Missing CS Tuning here

Paper

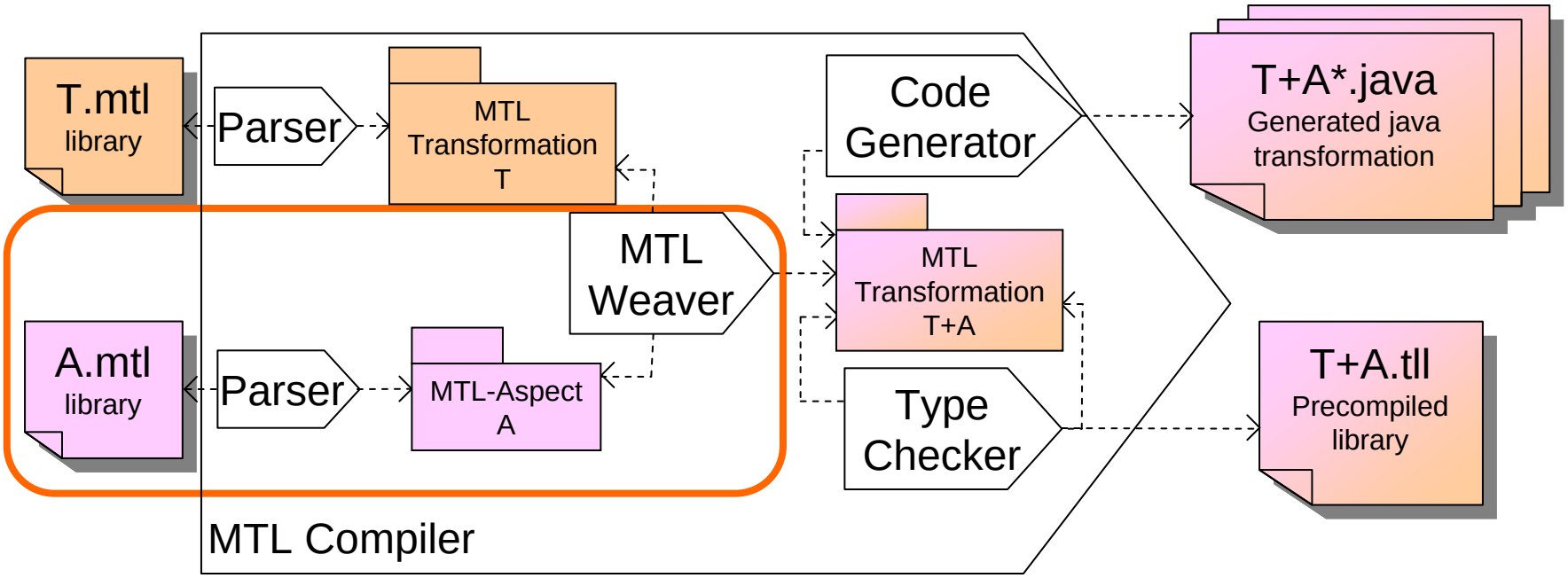
- Raul Silaghi, Frédéric Fondement, and Alfred Strohmeier, **Towards an MDA-oriented UML profile for distribution**., 8th International IEEE Enterprise Distributed Object Computing Conference - EDOC, Monterey, California, September 20-24 2004, IEEE Computer Society, 2004, pp. 227–239.

Tuning MDE Artefacts



Aspects on MTL Transformations

- Language extensions for defining aspects on transformation
- Enhance slightly the compilation process
- Reuse the concrete syntax as it is



Paper

- Raul Silaghi, Frédéric Fondement, and Alfred Strohmeier, **“Weaving” MTL model transformations**., Model Driven Architecture (Uwe Aßmann, Mehmet Aksit, and Arend Rensink, eds.), Lecture Notes in Computer Science, vol. 3599, Springer, 2004, pp. 123–138.

Contributions

Application of MDE in different domains

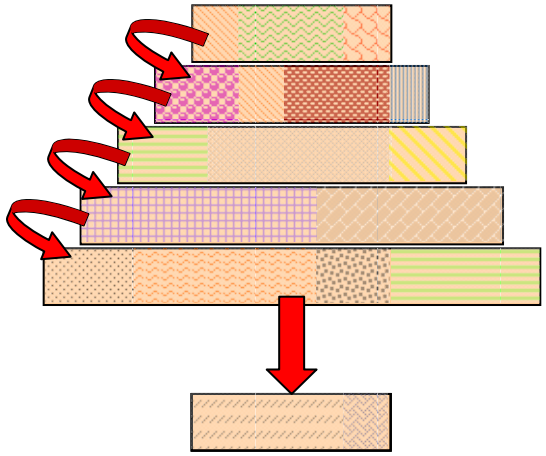
- Web applications
- Product line engineering
- Distributed systems
- Fondue Method

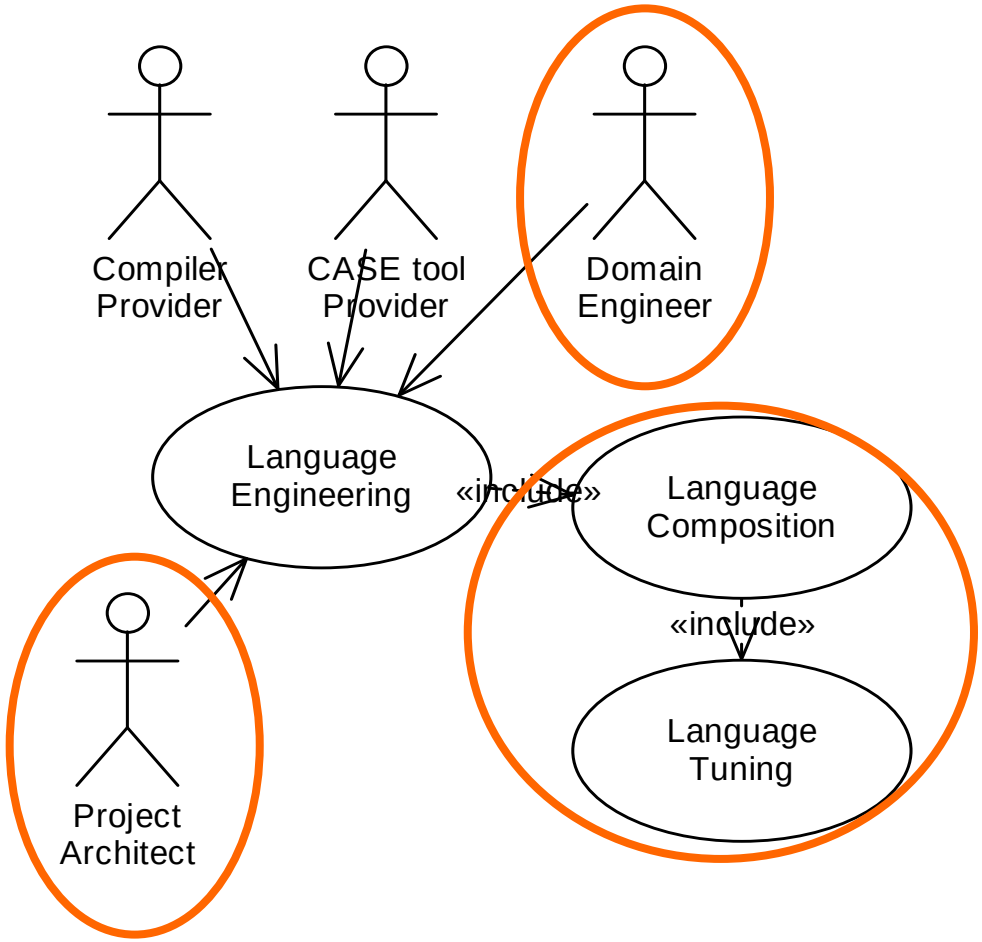
Improvement of modeling language engineering

- Model transformations
  - OO imperative languages
  - Interest of higher-order hierarchies in improvement process
- Syntaxes
  - Language theory
  - (Triple) Graph Grammars
  - Vector graphics

Conclusion

- Agile MDE Definition
  - Knowledge from real specialists !
  - “off-the-shelf (MDE) components”
  - Adaptable to each project





Thank you !

Emergency Slides

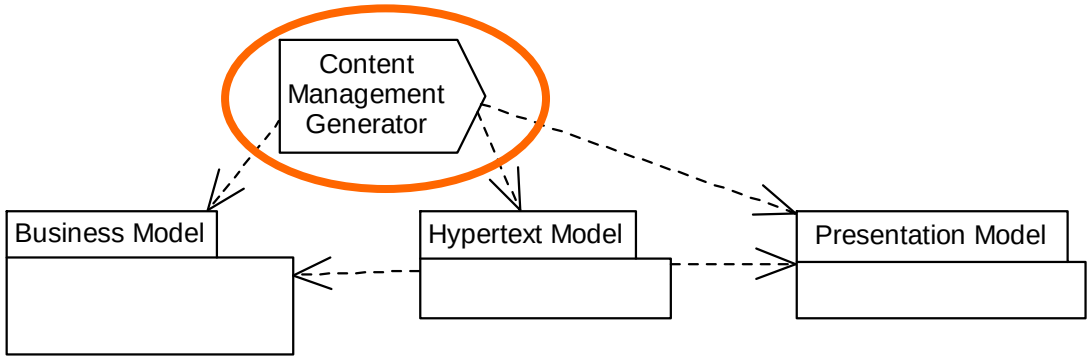
- Netsilon Details
- Language Definition
  - Textual Concrete Syntax
  - Graphical Concrete Syntax
- MTL Aspects
- Adding an additional abstraction layer
- Adaptors

Emergency Slides

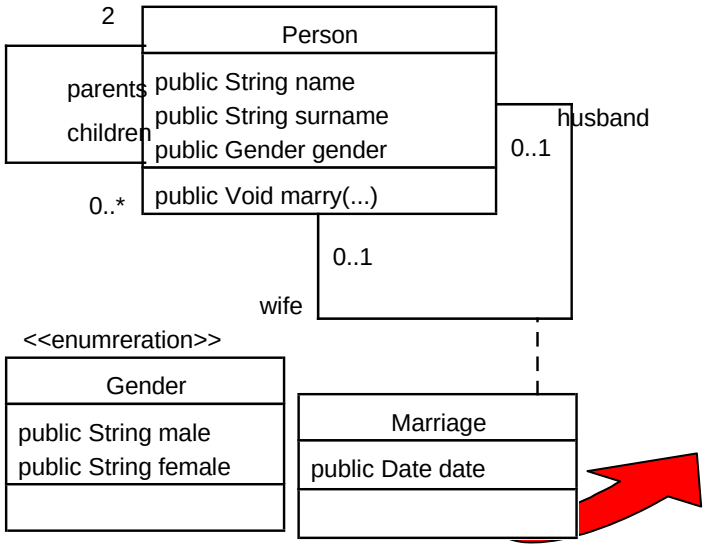
- Netsilon Details
- Language Definition
  - Textual Concrete Syntax
  - Graphical Concrete Syntax
- MTL Aspects
- Adding an additional abstraction layer
- Adaptors

Object Administrator

M2



M1



Attributes

OID	101c7aad7875cfc5099e256302ada3cd
String Business_ Model::Person.name	Fedorovna
String Business_ Model::Person.surname	Alexandra
Business_ Model::Gender Business_ Model::Person.gender	female

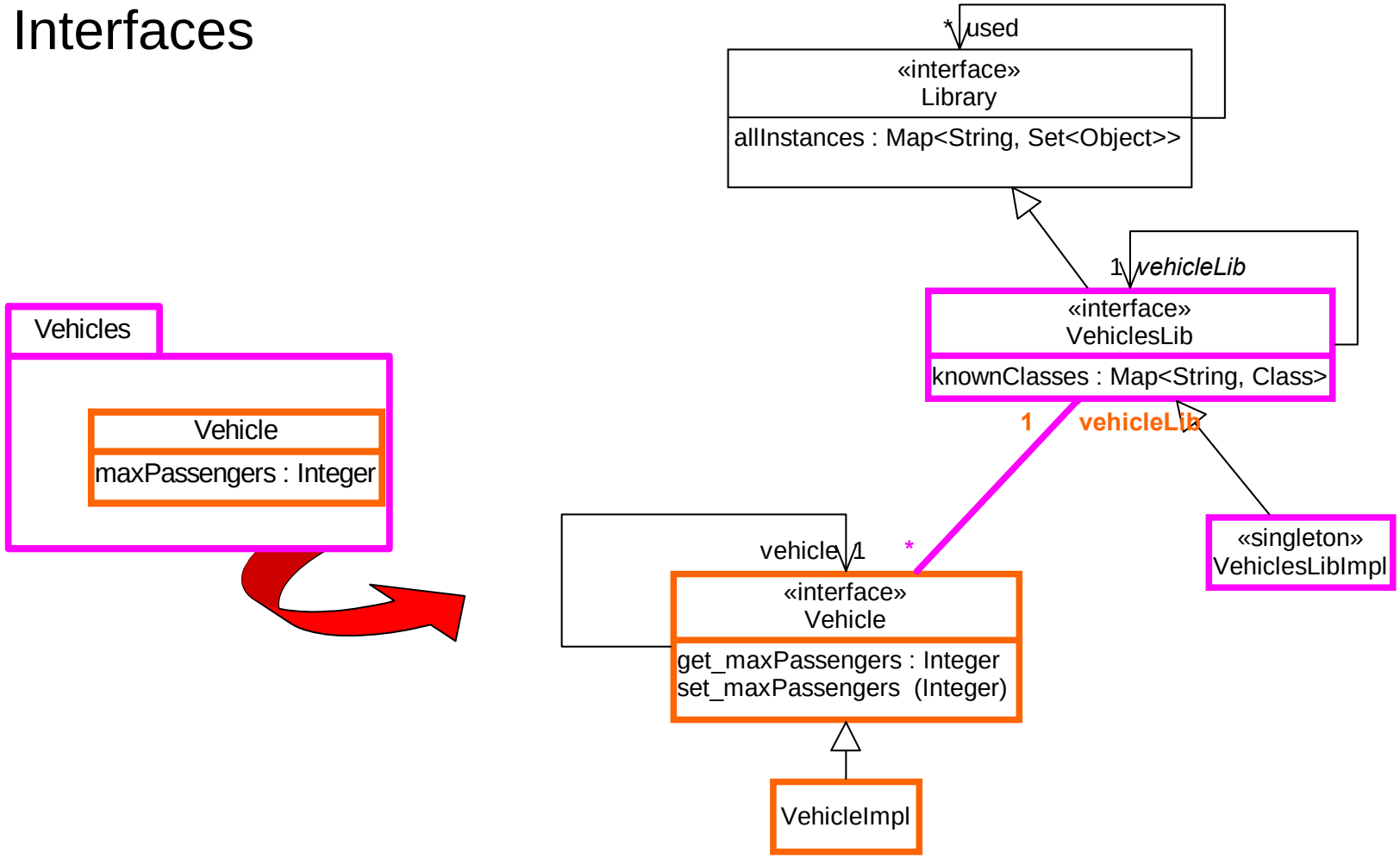
Submit Reset

Links

Business_ Model::Person	OID	name	surname	gender	
Business_ Model::Person.husband	101be85ec9c195d88658871c9ec425d8	Alexandrovitch Rom...	Nicolas	male	

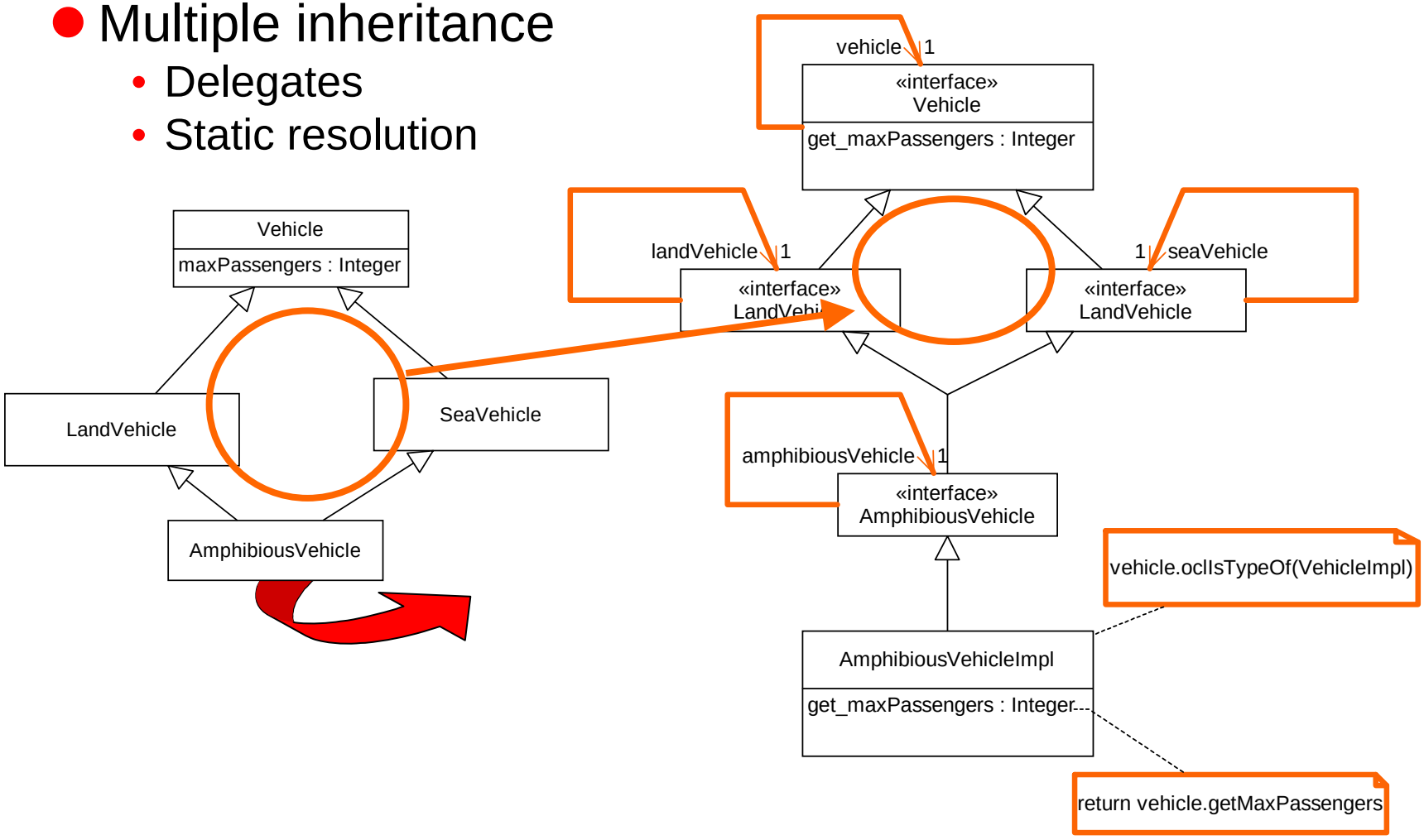
Transformation to Java

- Interfaces



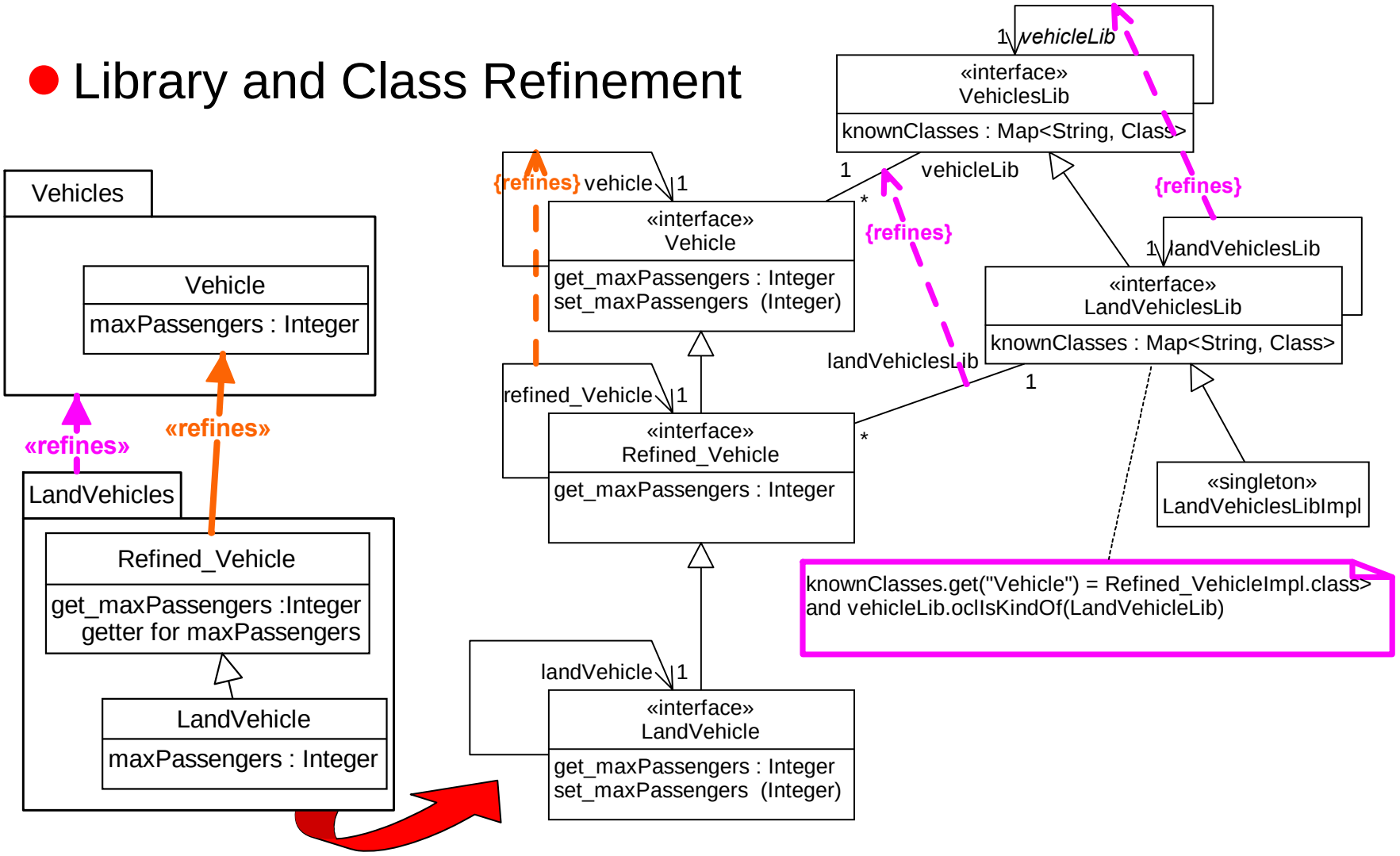
Transformation to Java

- Multiple inheritance
  - Delegates
  - Static resolution



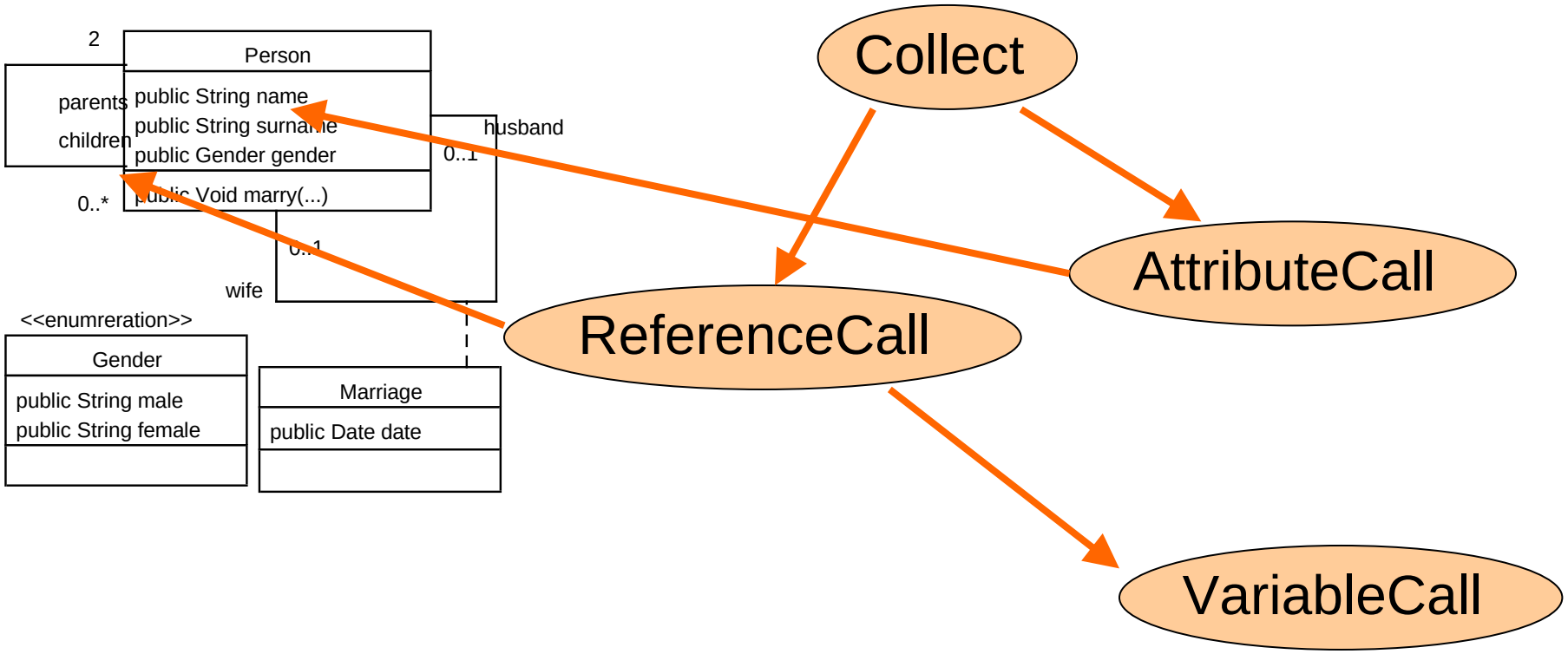
Transformation to Java

- Library and Class Refinement



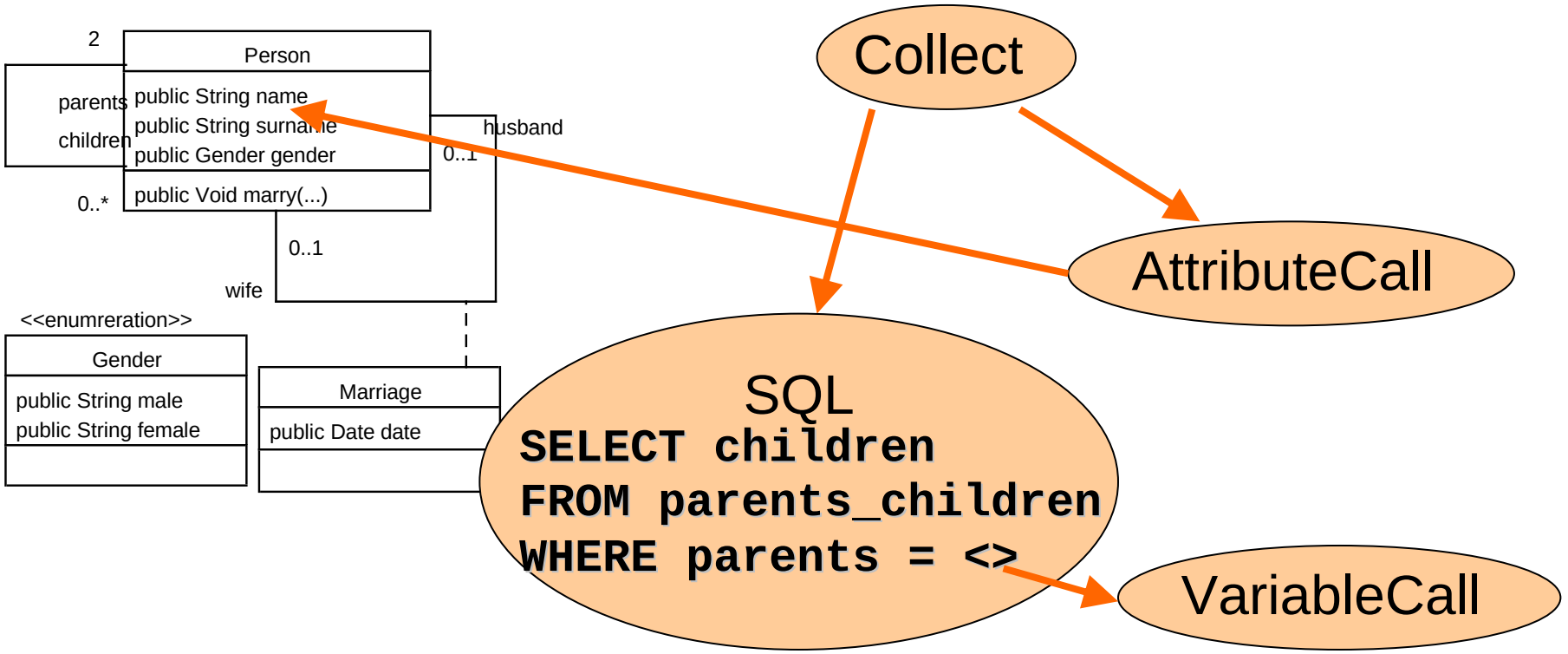
Optimization Scheme

aPerson.children->collect(name)



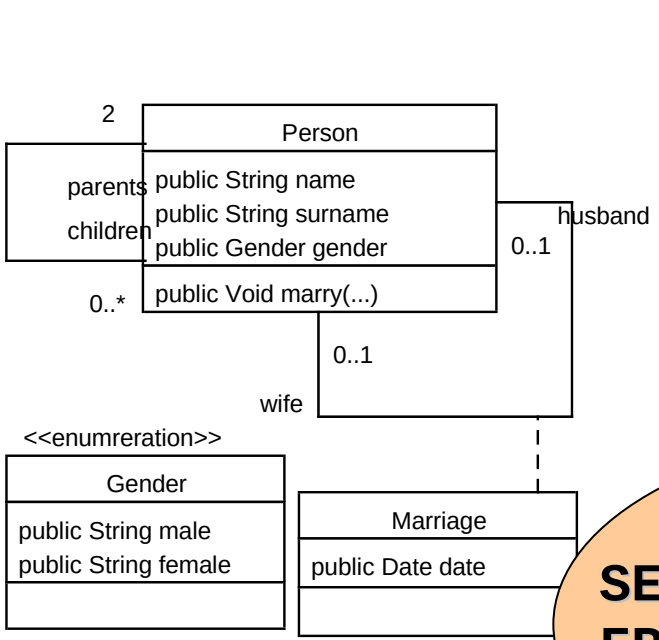
Optimization Scheme

aPerson.children->collect(name)



Optimization Scheme

aPerson.children->collect(name)



SQL
```
SELECT name
FROM person
WHERE parents IN (<>)
```

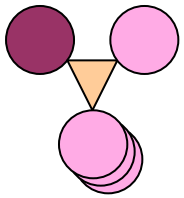
SQL
```
SELECT children
FROM parents_children
WHERE parents = <>
```

VariableCall

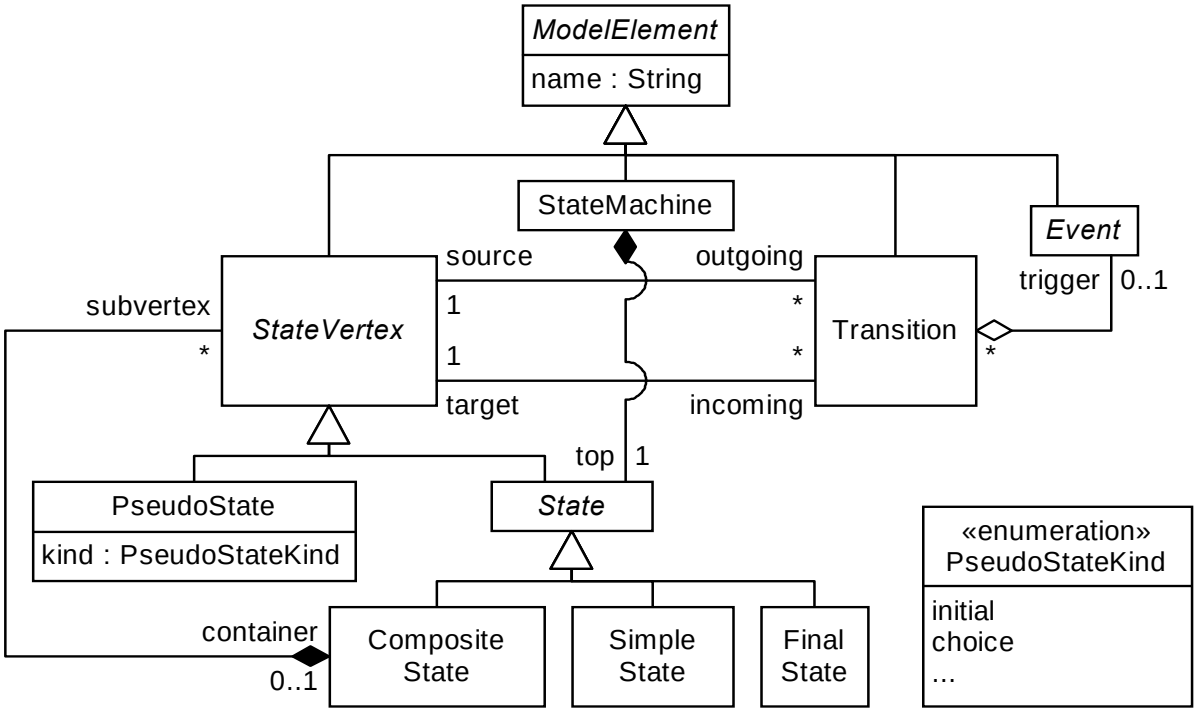
Emergency Slides

- Netsilon Details
- Language Definition
  - Textual Concrete Syntax
  - Graphical Concrete Syntax
- MTL Aspects
- Adding an additional abstraction layer
- Adaptors

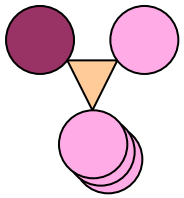
Concepts Definition



Abstract Syntax

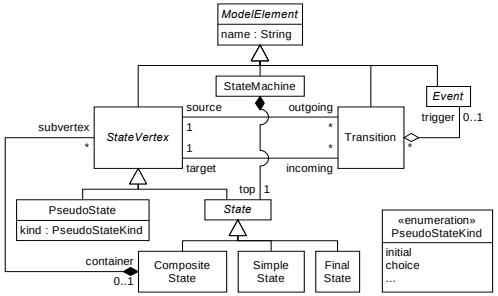


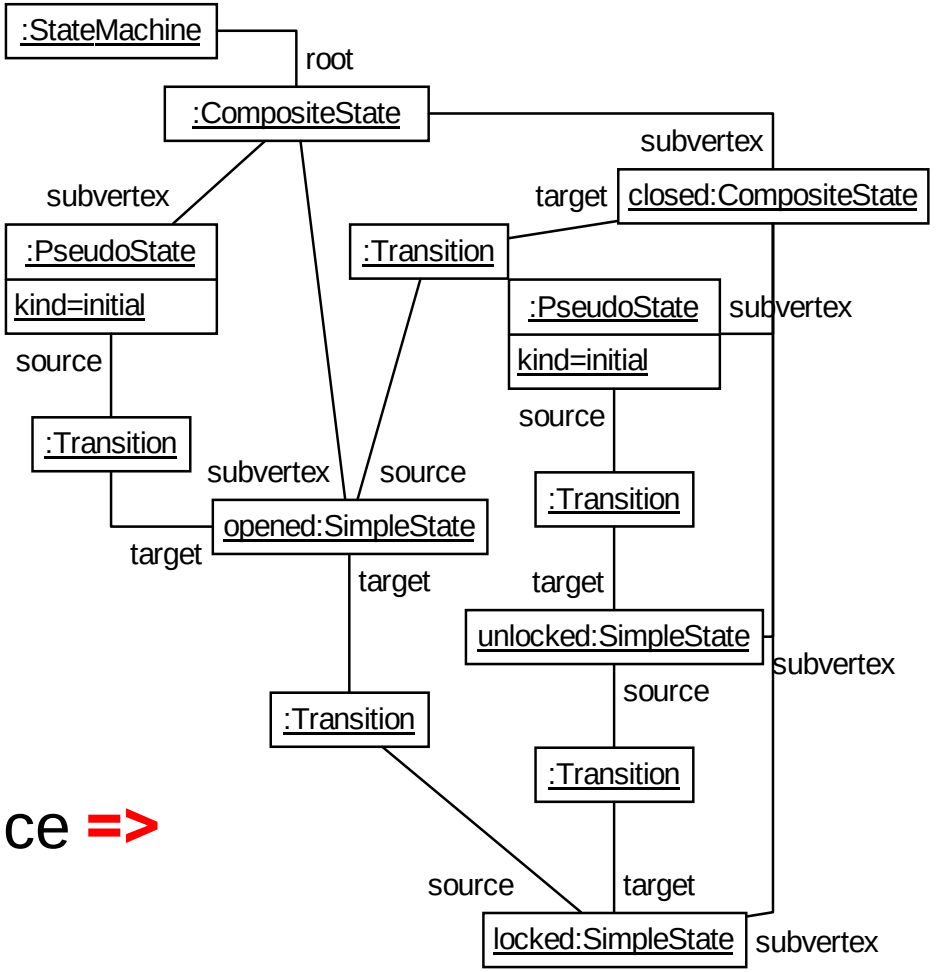
Concepts Definition



M1

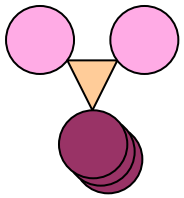
Abstract Syntax



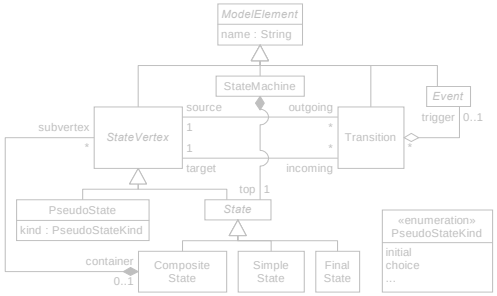


An (M1) sentence **=>**

Interface Definition

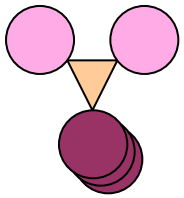


Abstract Syntax + Concrete Syntax(es)



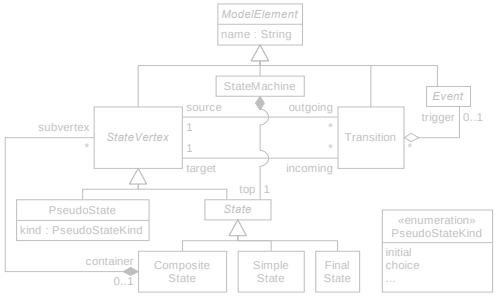

```
sm ::= "Statemachine" IDENT compositeState

state ::= normalState | pseudostate

normalState ::= "initial"? (simpleState | compositeState)

simpleState ::= "State" IDENT

compositeState ::= "CompositeState IDENT? LCURLYBRACKET
                     (state | transition)* RCURLYBRACKET

transition ::= "Transition" IDENT? "from" IDENT
                    "to" IDENT ("on" IDENT)?

pseudoState ::= "FinalState" IDENT | "Choice" IDENT
```

Interface Definition



M1

Abstract Syntax + Concrete Syntax(es)



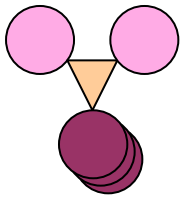

```
sm ::= "Statemachine" IDENT compositeState

state ::= normalState | pseudostate

normalState ::= "initial"? (simpleState | compositeState)

simpleState ::= "State" IDENT

compositeState ::= "CompositeState IDENT? LCURLYBRACKET
                    (state | transition)* RCURLYBRACKET

transition ::= "Transition" IDENT? "from" IDENT
                  "to" IDENT ("on" IDENT)?

pseudoState ::= "FinalState" IDENT | "Choice" IDENT
```

StateMachine Door
CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    Transition from unlocked
      to locked on lock

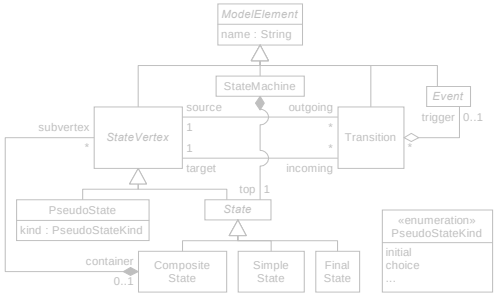
<= An (M1) sentence

...

Interface Definition

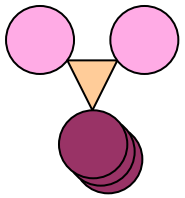


Abstract Syntax + Concrete Syntax(es)



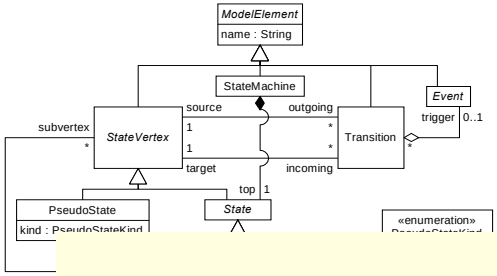
Transition	SimpleState	Composite State	FinalState	PseudoState (initial)	PseudoState (choice)
–event⟩	name	name / contents	◉	●	○

Interface Definition

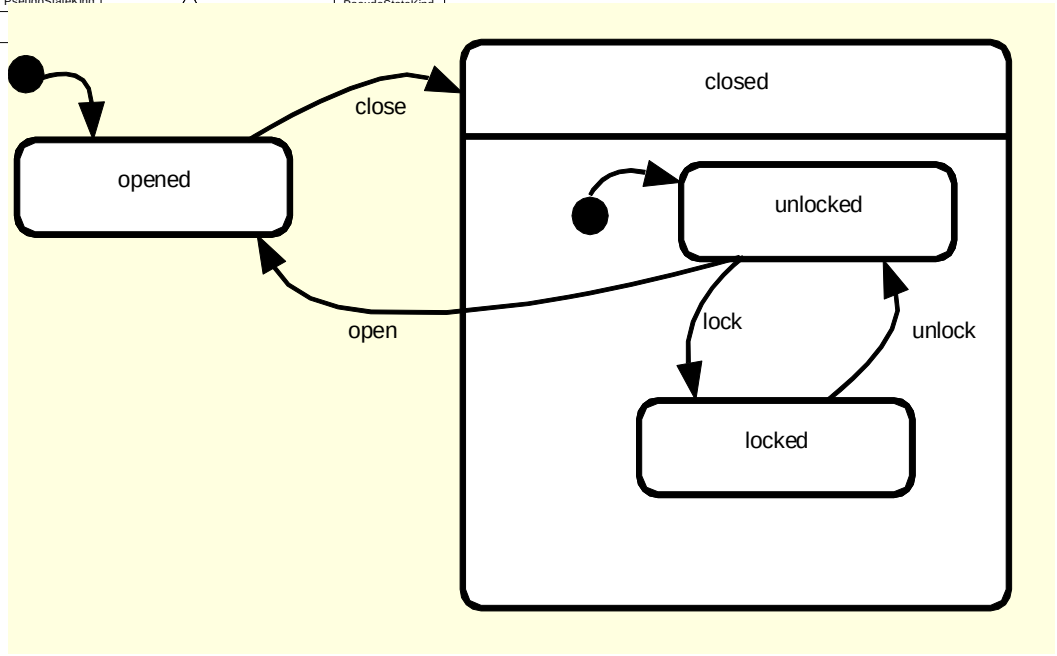


M1

Abstract Syntax + Concrete Syntax(es)

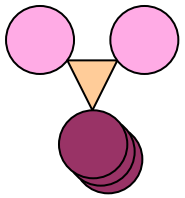


Transition	SimpleState	Composite State	FinalState	PseudoState (initial)	PseudoState (choice)
event	name	name / contents	◉	●	○

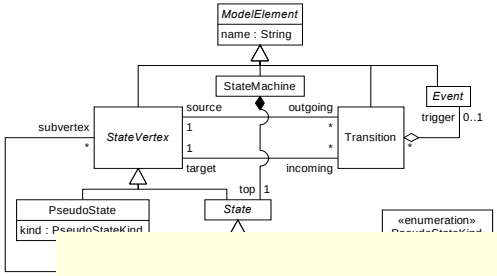


<= An (M1) sentence

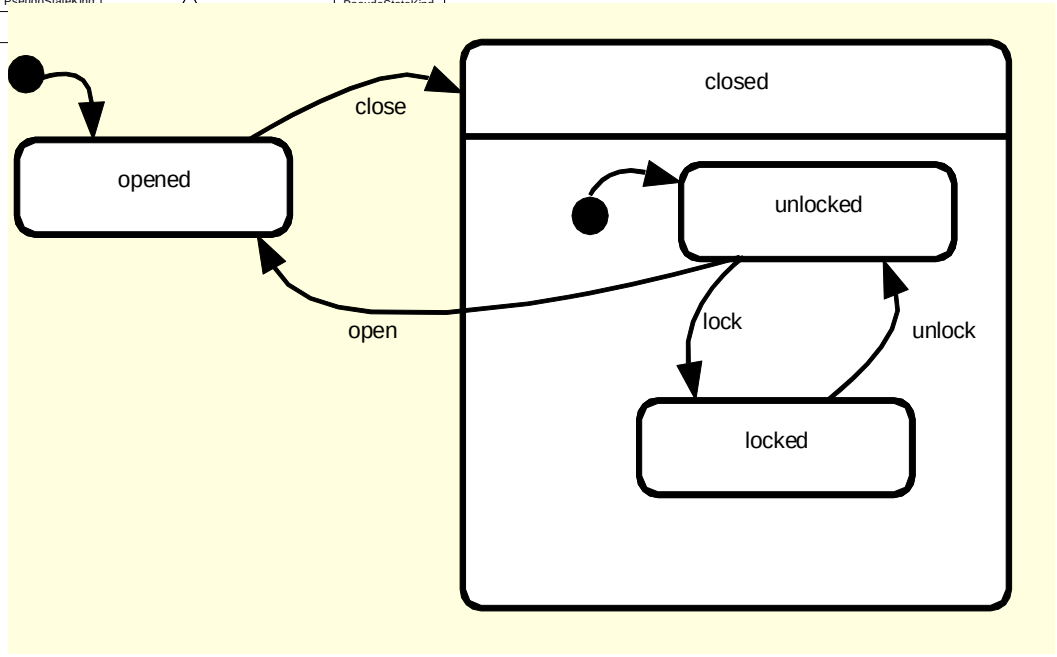
Interface Definition



Abstract Syntax + Concrete Syntax(es)



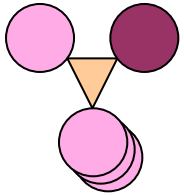
Transition	SimpleState	Composite State	FinalState	PseudoState (initial)	PseudoState (choice)
event	name	name / contents	◉	●	○



<= An (M1) sentence

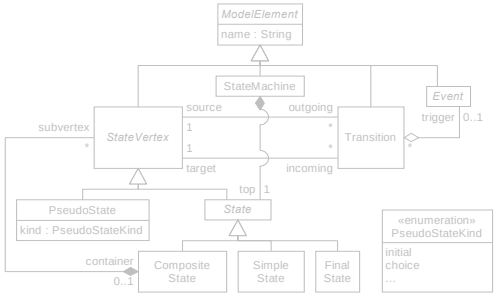
- In practice
  - Layout constraints
  - User interactions

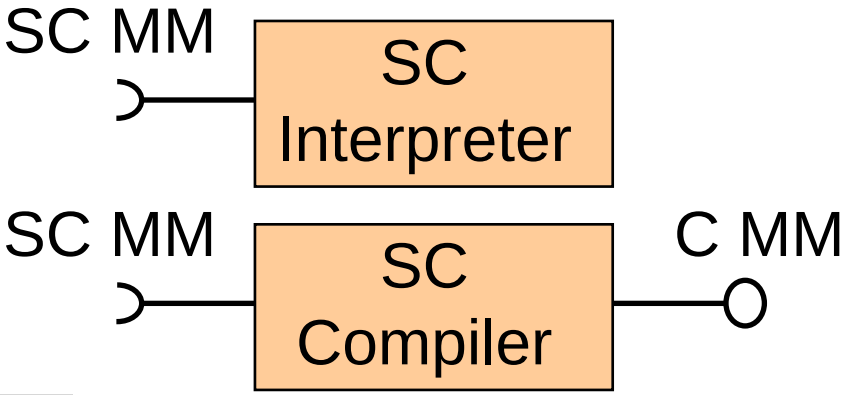
Meaning Definition



M2

Abstract Syntax + Concrete Syntax(es) + Semantics





Transition	SimpleState	Composite State	FinalState	PseudoState (initial)	PseudoState (choice)
-event->	name	name / contents			

or

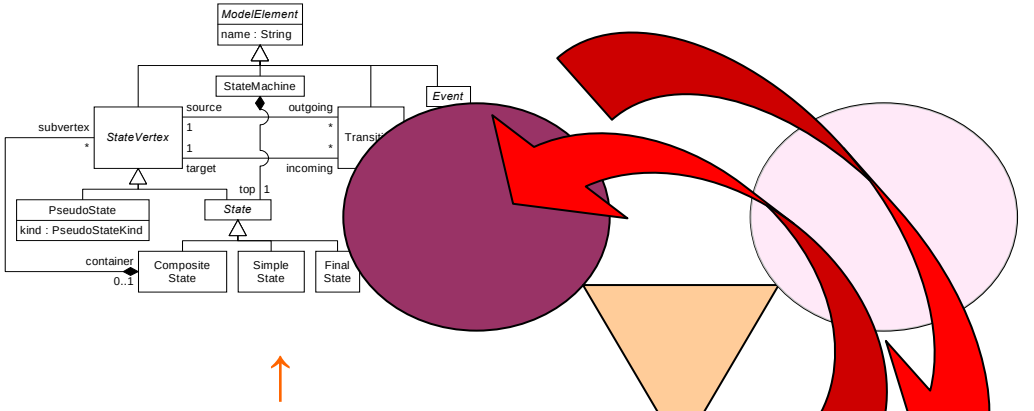
```
sm ::= "Statemachine" IDENT compositeState
state ::= normalState | pseudostate
normalState ::= "initial"? (simpleState | compositeState)
simpleState ::= "State" IDENT
compositeState ::= "CompositeState«  IDENT? LCURLYBRACKET
(state | transition)* RCURLYBRACKET
transition ::= "Transition" IDENT? "from" IDENT "to"
IDENT ("on" IDENT)?
pseudoState ::= "FinalState" IDENT | "Choice" IDENT
```

- KerMeta ?
- Model transformation ?
- Research issue !

Language Definition

M2

Abstract Syntax + Concrete Syntax(es) ntics



↑

Metamodel

SC MM — SC Interpreter

SC MM — SC Compiler — SQL MM

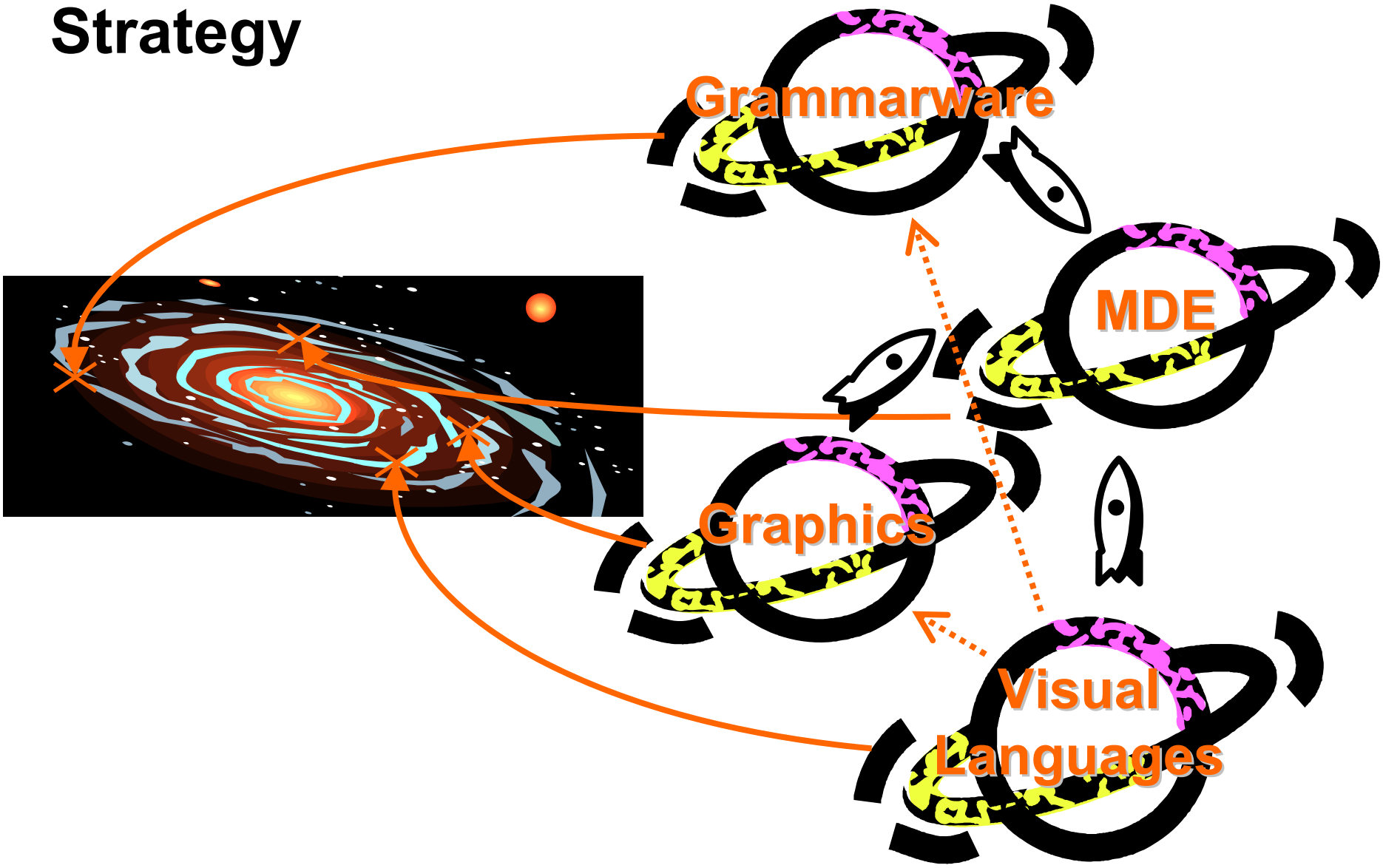
??? →

	PseudoState (choice)
-event	○

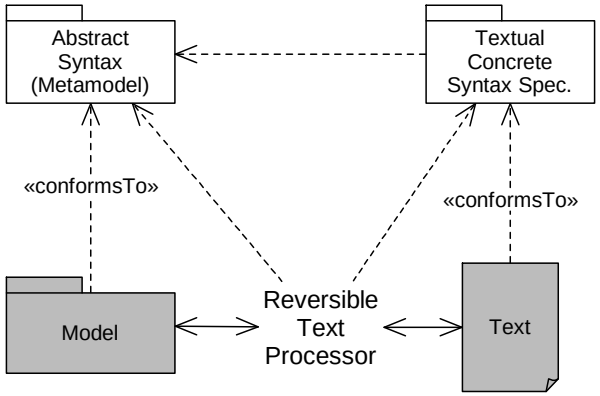
??? →

```
sm ::= "Statemachi                    tate
state ::= normalState | pseudoState
normalState ::= "initial"? (simpleState | compositeState)
simpleState ::= "State" IDENT
compositeState ::= "CompositeState«  IDENT? LCURLYBRACKET
(state | transition)* RCURLYBRACKET
transition ::= "Transition" IDENT? "from" IDENT "to"
IDENT ("on" IDENT)?
pseudoState ::= "FinalState" IDENT | "Choice" IDENT
```

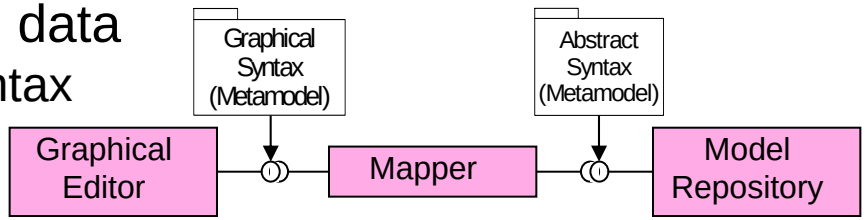
Strategy



Contributions



- Textual concrete syntax
  - “mapping” metamodel
- Approach to graphical concrete syntax specification
  - Metamodel for representation data
    - Interface for the concrete syntax
  - Mapping to abstract syntax



- Technology for graphical concrete syntax realization
  - Representation using SVG templates
  - Library of possible user interactions

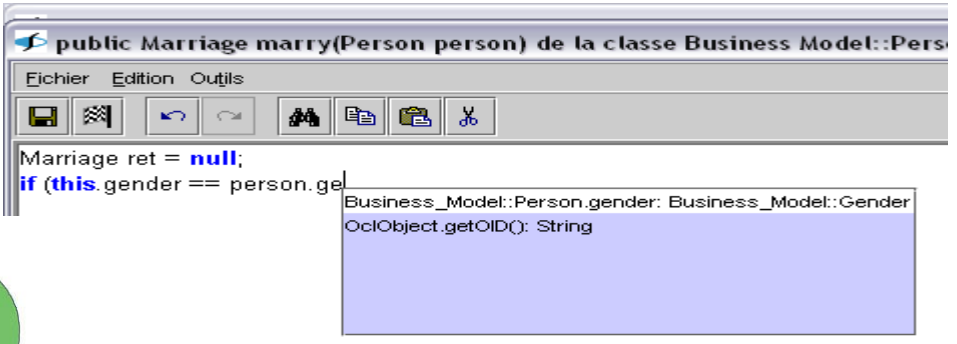
```
<svg onCreation =“Java| …” …>
 <g dpi:component=“Contained, Translatable, …” …>
  <text dpi:component=“Editable, …” …/>
  …
 </g>
</svg>
```

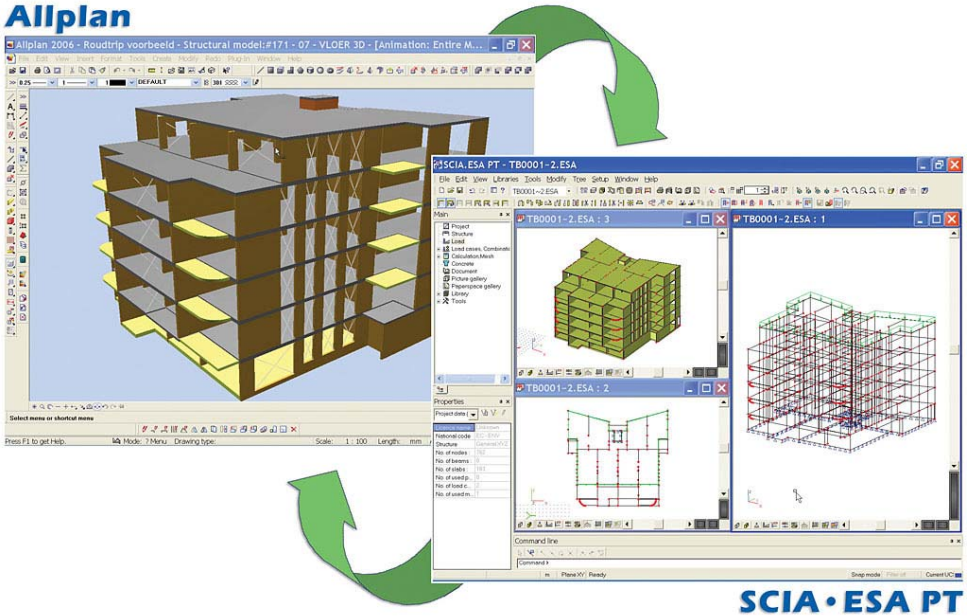
- Other technologies apply
  - (Triple) Graph Grammars
  - GMF, Topcased, …

Analytic –vs.– Interactive CS

- Solutions to textual and graphical CS are very different
  - Textual => usually analytic
  - Graphical => usually interactive

- Unification of solutions ?
- Inversion of solutions ?

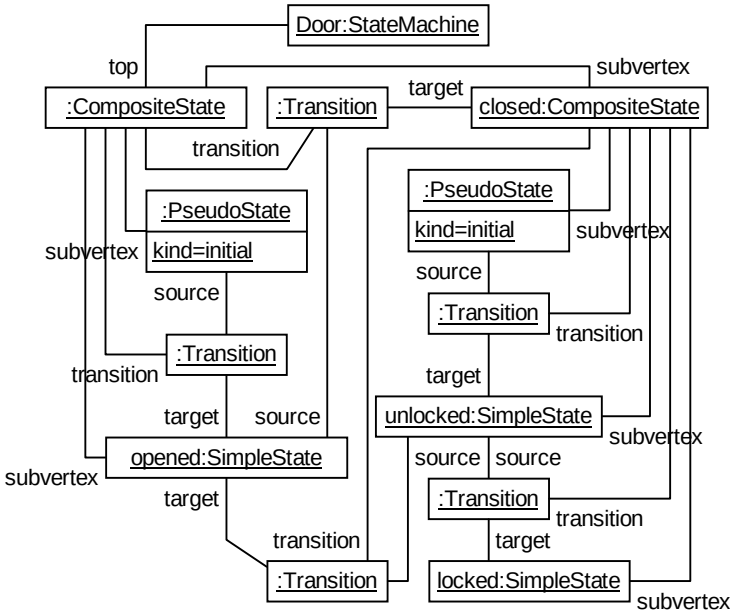




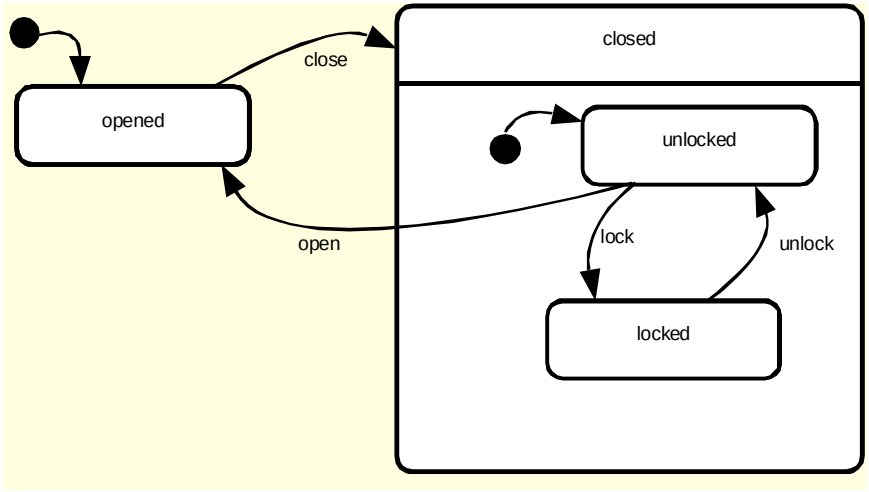
Emergency Slides

- Netsilon Details
- Language Definition
  - Textual Concrete Syntax
  - Graphical Concrete Syntax
- MTL Aspects
- Adding an additional abstraction layer
- Adaptors

Example



The Model



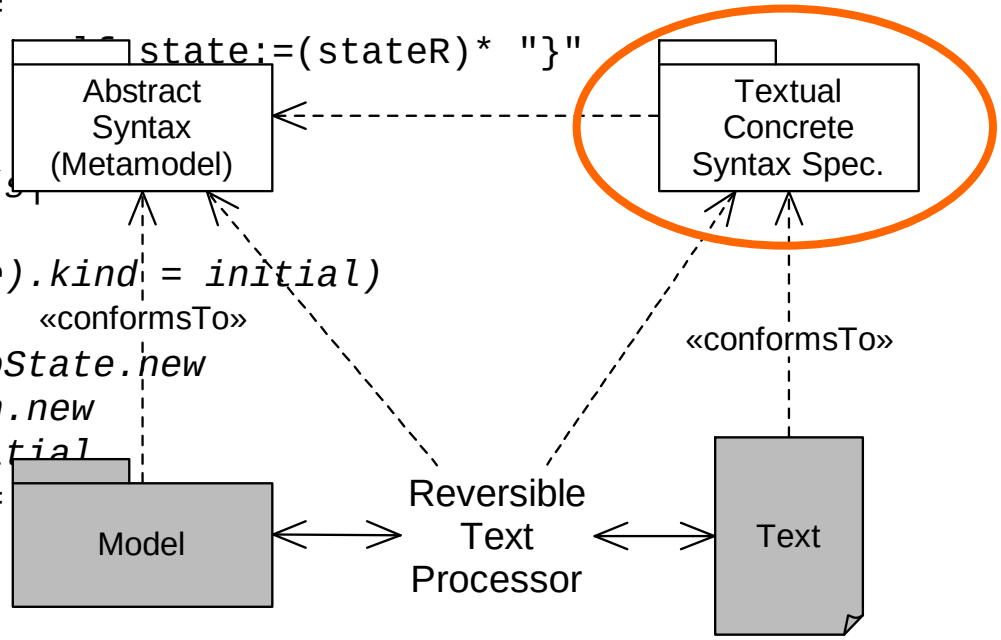
A Graphical Representation

A Textual Representation

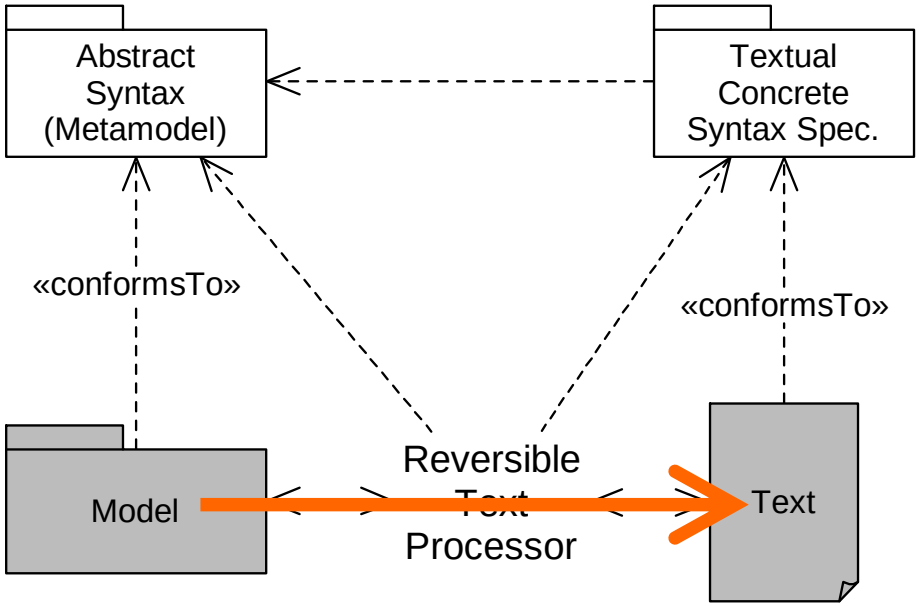
```
StateMachine Door

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    Transition from unlocked to locked on lock
    Transition from locked to unlocked on unlock
    Transition from unlocked to opened on open
  }
  Transition from opened to closed on close
}
```

Example TCSS

```
start template for StateMachine ::=
"StateMachine" self.name self.top:=stateR;

rule stateR ::=
   {OCL| self.oclIsKindOf(SimpleState)}? ssr
|  {OCL| self.oclIsKindOf(CompositeState)}? csr

template csr for CompositeState ::=
init "CompositeState" self.name "{"          state:=(stateR)* "}"

rule init ::=
{OCL| self.incoming.source->exists(s|
        s.oclIsKindOf(PseudoState)
        and s.oclAsType(PseudoState).kind = initial)
}? "initial" =>{KerMeta|
  var _init:PseudoState init PseudoState.new
  var _t:Transition init Transition.new
  _init.kind := PseudoStateKind#initial
  _t.source :=  _init  _t.target :=
…
```

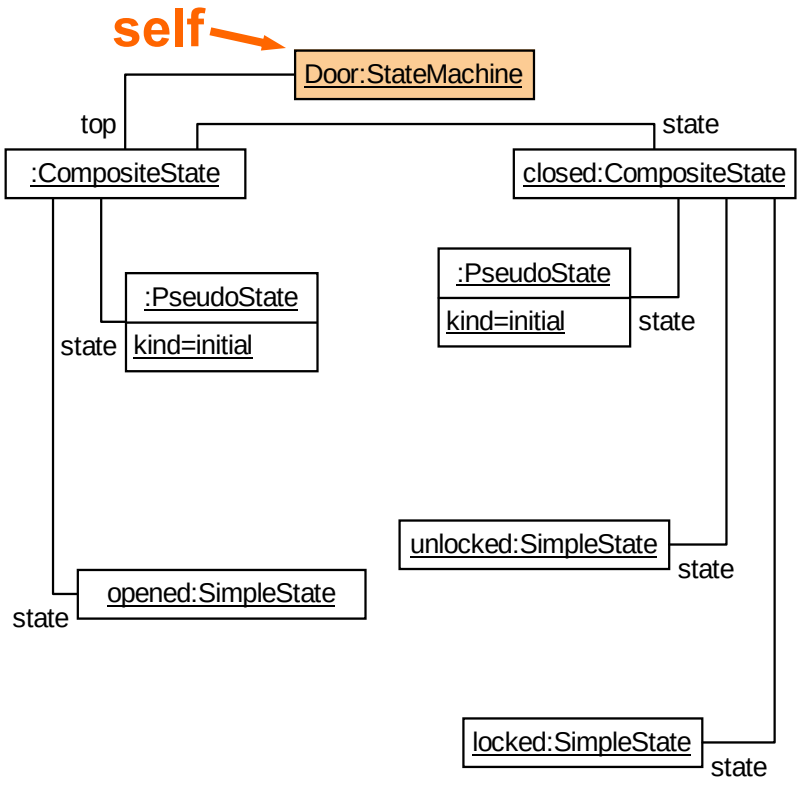


Example: Synthesis (i.e. model to text)



Example: Synthesis

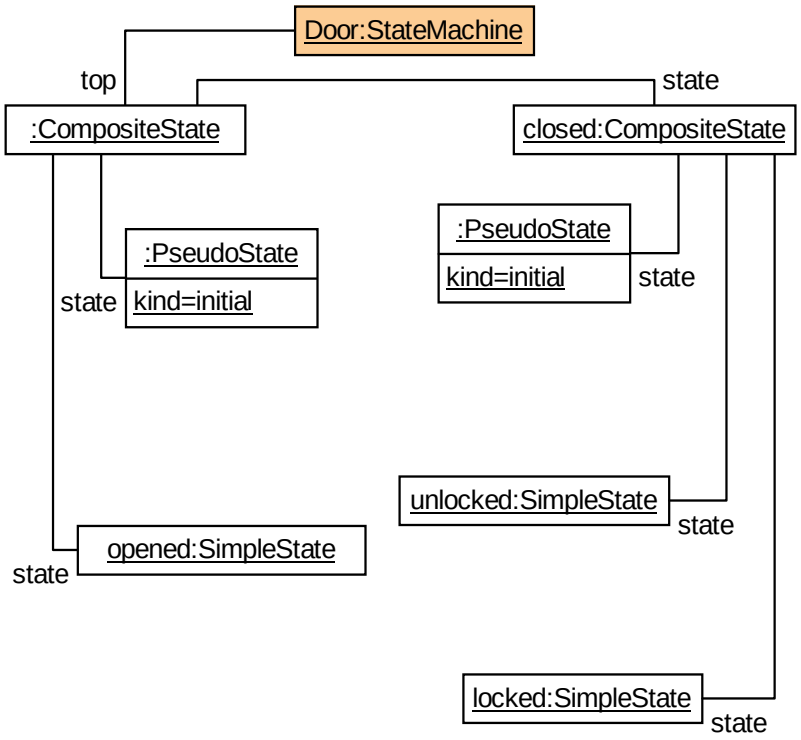
```
start template for StateMachine ::=
"StateMachine" self.name self.top:=stateR;
```



Example: Synthesis

- Rule stack
  - StateMachine (Door)

```
start template for StateMachine ::=
"StateMachine" self.name self.top:=stateR;
```

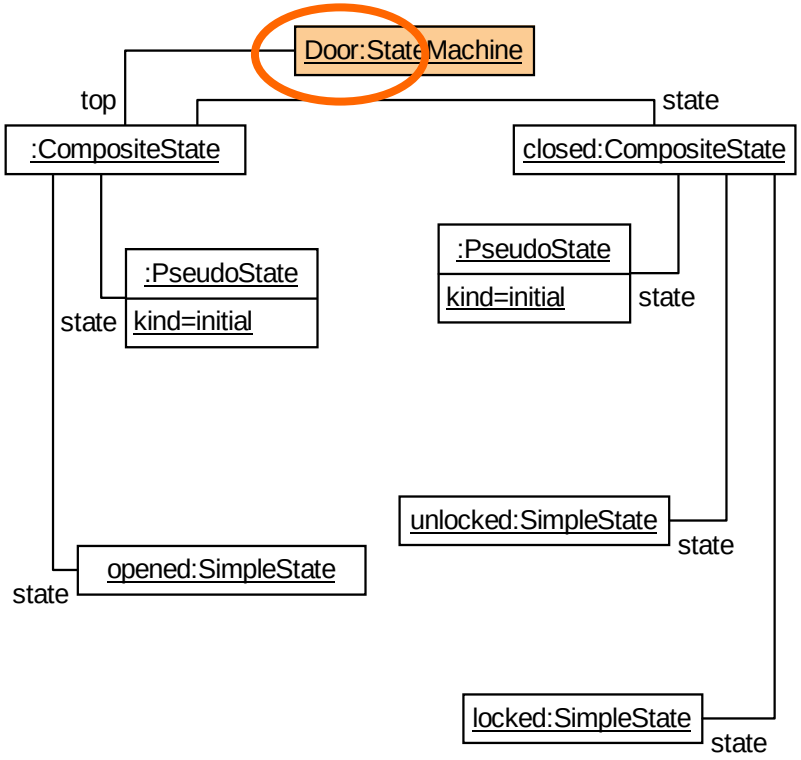


StateMachine

Example: Synthesis

- Rule stack
  - StateMachine (Door)

```
start template for StateMachine ::=
"StateMachine" self.name self.top:=stateR;
```

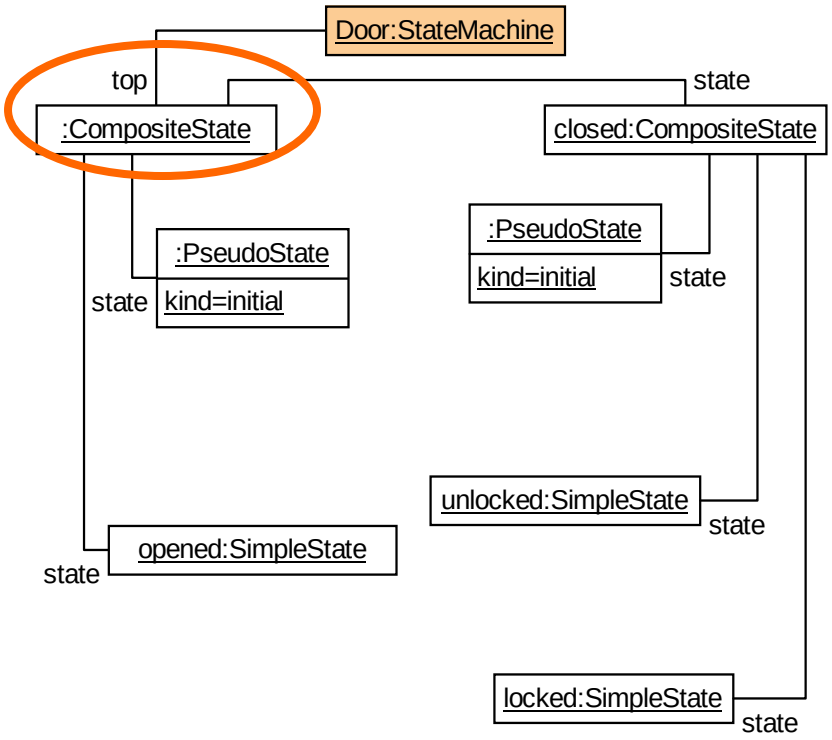


StateMachine Door

- Rule stack
  - StateMachine (Door)

Example: Synthesis

```
start template for StateMachine ::=
"StateMachine" self.name self.top:=stateR;
```



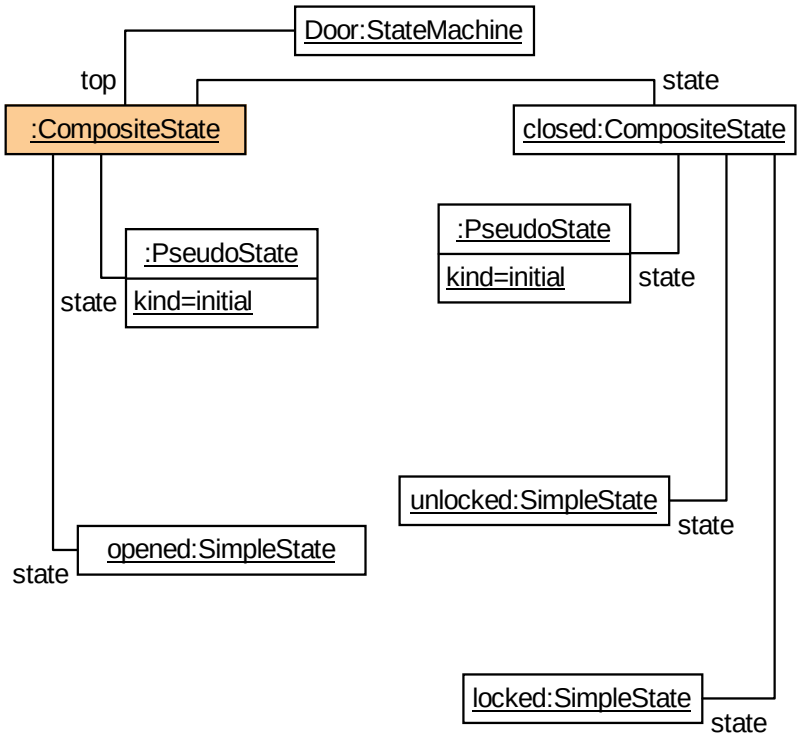
StateMachine Door

Example: Synthesis

- Rule stack
  - StateMachine (Door)
  - stateR (Door.top)

```
rule stateR ::=
    {OCL| self.oclIsKindOf(SimpleState)}? ssr
|   {OCL| self.oclIsKindOf(CompositeState)}? csr
```

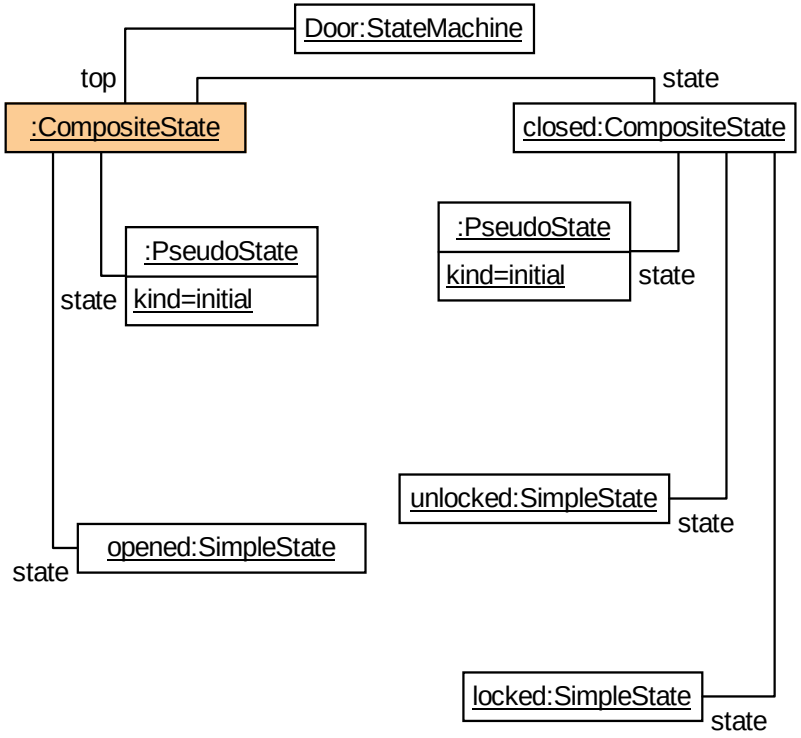
StateMachine Door



Example: Synthesis

- Rule stack
  - StateMachine (Door)
  - stateR (Door.top)
  - csr (Door.top)

```
template csr for CompositeState ::=
init "CompositeState" self.name "{"
self.state:=(stateR)* "}"
```



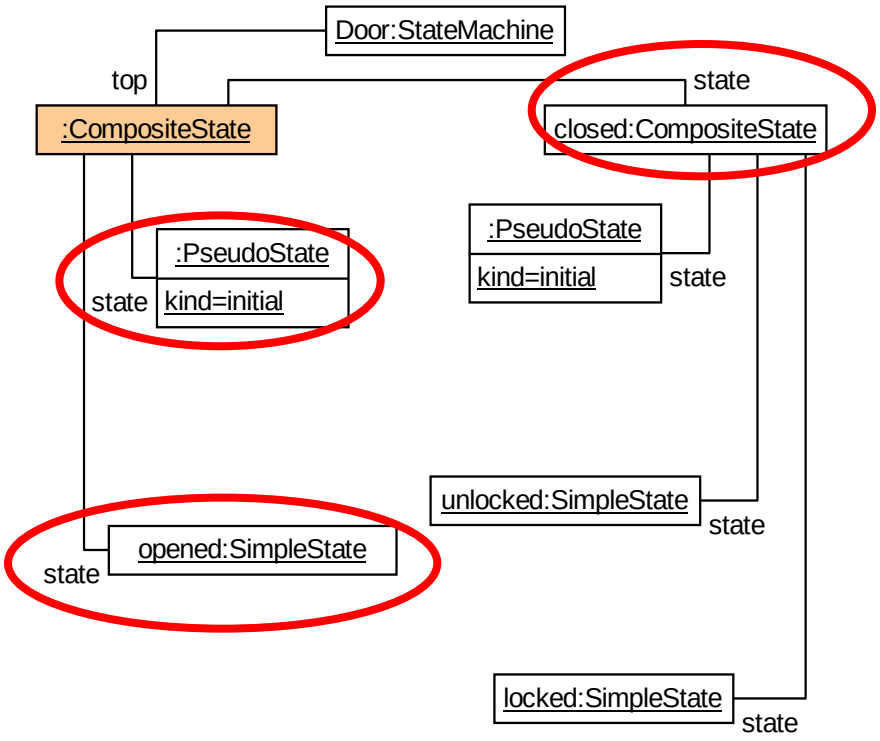
StateMachine Door

CompositeState {

Example: Synthesis

- Rule stack
  - StateMachine (Door)
  - stateR (Door.top)
  - csr (Door.top)

```
template csr for CompositeState ::=
init "CompositeState" self.name "{"
self.state:=(stateR)* "}"
```



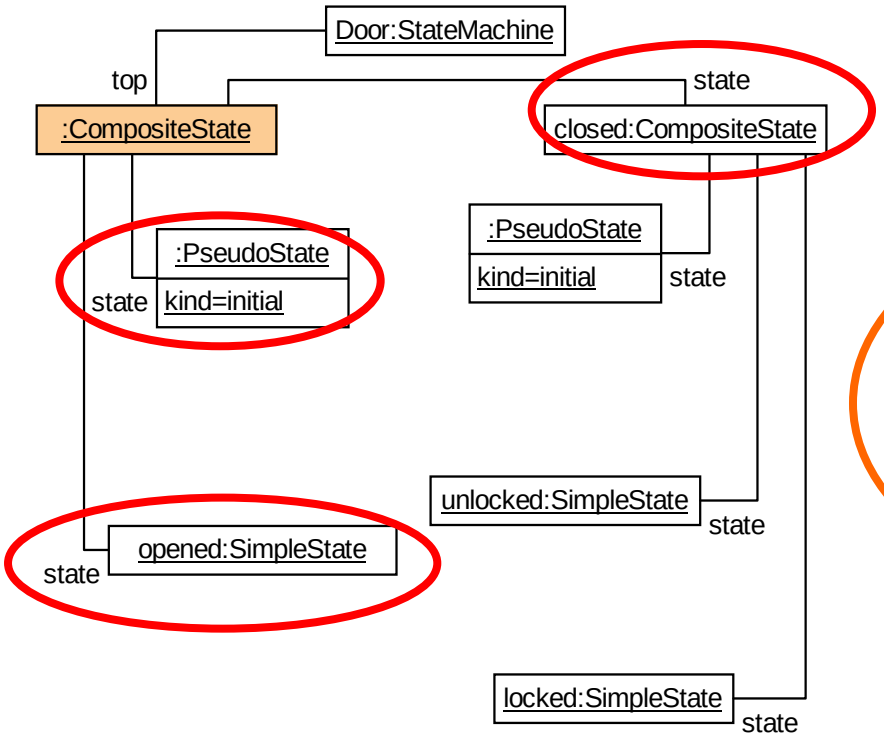
StateMachine Door

CompositeState {

Example: Synthesis

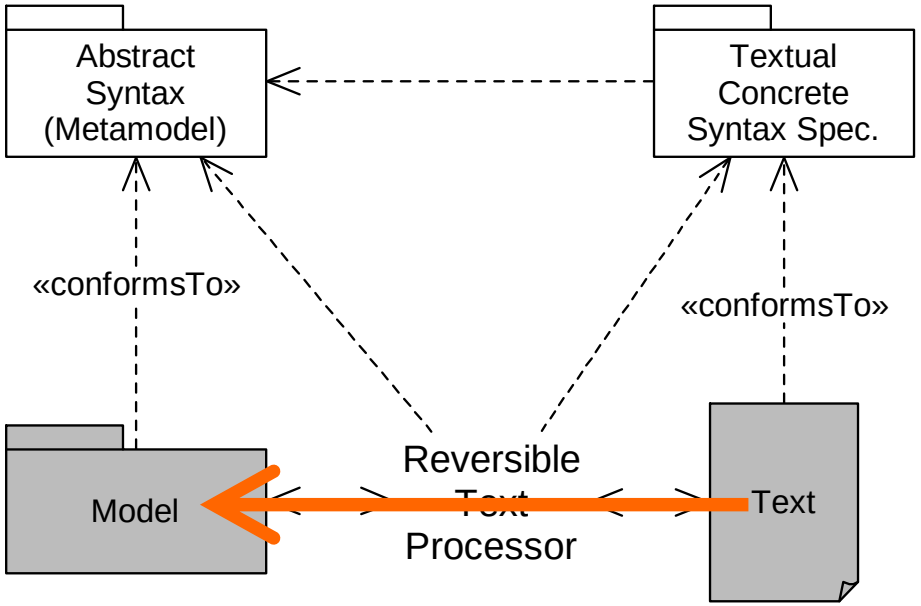
- Rule stack
  - StateMachine (Door)
  - stateR (Door.top)
  - csr (Door.top)

```
template csr for CompositeState ::=
init "CompositeState" self.name "{"
self.state:=(stateR)* "}"
```




```
StateMachine Door

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    ...
  }
  ...
}
```

Example: Analysis (i.e. text to model)



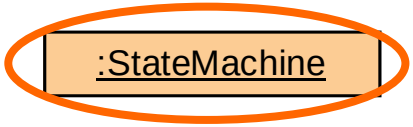
- Rule stack
  - StateMachine (Door)

Example: Analysis

```
start template for StateMachine ::=
"StateMachine" self.name self.top:=stateR;
```

StateMachine Door

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    …
  }
  …
}



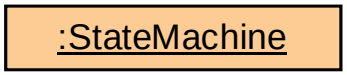
- Rule stack
  - StateMachine (Door)

Example: Analysis

```
start template for StateMachine ::=
"StateMachine" self.name self.top:=stateR;
```

StateMachine Door

```
CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    ...
  }
  ...
}
```



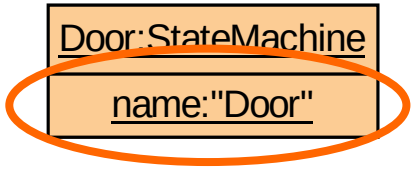
Example: Analysis

- Rule stack
  - StateMachine (Door)

```
start template for StateMachine ::=
"StateMachine" self.name self.top:=stateR;
```

StateMachine Door

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    …
  }
  …
}



- Rule stack
  - StateMachine (Door)

Example: Analysis

```
start template for StateMachine ::=
"StateMachine" self.name self.top:=stateR;
```

StateMachine Door |

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    …
  }
  …
}

Example: Analysis

- Rule stack
  - StateMachine (Door)
  - stateR

```
rule stateR ::=
   {OCL| self.oclIsKindOf(SimpleState)}? ssr
|  {OCL| self.oclIsKindOf(CompositeState)}? csr
```

???
Let's try
This !

```
StateMachine Door |

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    ...
  }
  ...
}
```

Door:StateMachine

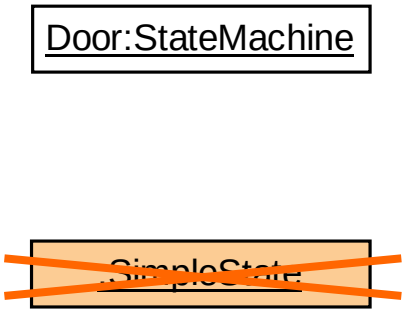
Example: Analysis

- Rule stack
  - StateMachine (Door)
  - stateR
  - ssr (SimpleState)

```
template ssr for SimpleState ::=
init "State" self.name
```

Ooops !
Backtrack !

```
StateMachine Door

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    ...
  }
  ...
}
```



Example: Analysis

- Rule stack
  - StateMachine (Door)
  - stateR

Let's try this, then !

```
rule stateR ::=
    {OCL| self.oclIsKindOf(SimpleState)}? ssr
  | {OCL| self.oclIsKindOf(CompositeState)}? csr
```

```
StateMachine Door |

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    ...
  }
  ...
}
```

Door:StateMachine

Example: Analysis

- Rule stack
  - StateMachine (Door)
  - stateR
  - csr (:CompositeState)

```
template csr for CompositeState :=
init "CompositeState" self.name "{"
self.state:=(stateR)* "}"
```

```
StateMachine Door

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    ...
  }
  ...
}
```

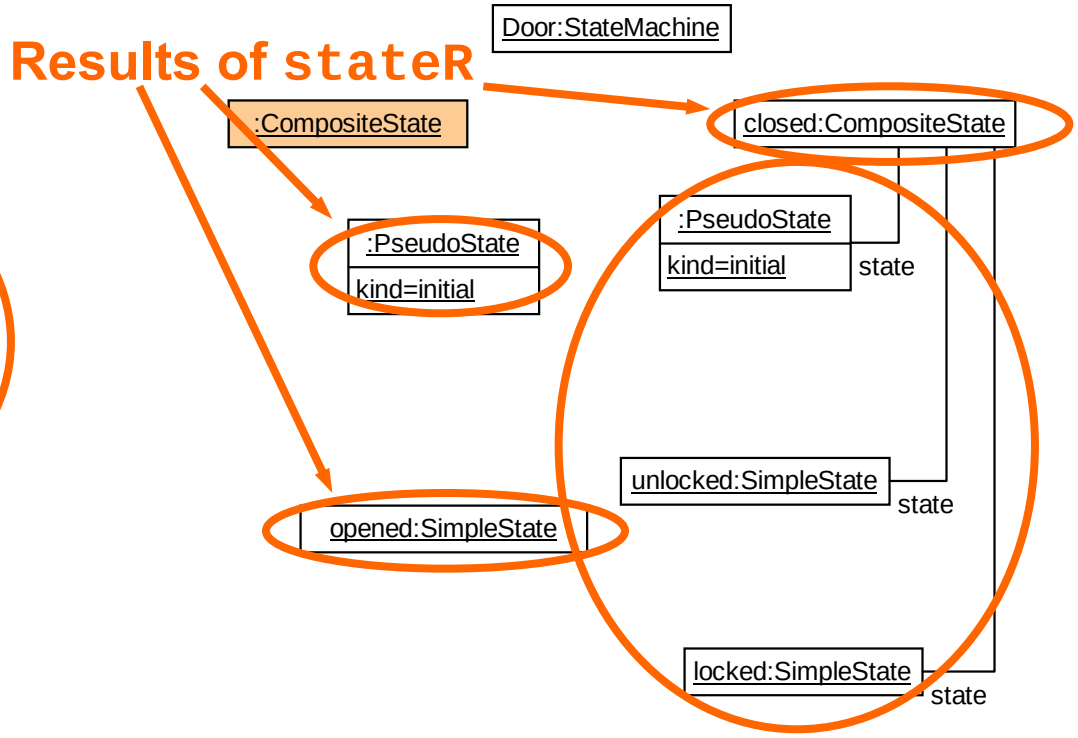
Example: Analysis

- Rule stack
  - StateMachine (Door)
  - stateR
  - csr (:CompositeState)

```
template csr for CompositeState ::=
init "CompositeState" self.name "{"
self.state:=(stateR)* "}"
```

StateMachine Door

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    …
  }
  …
}



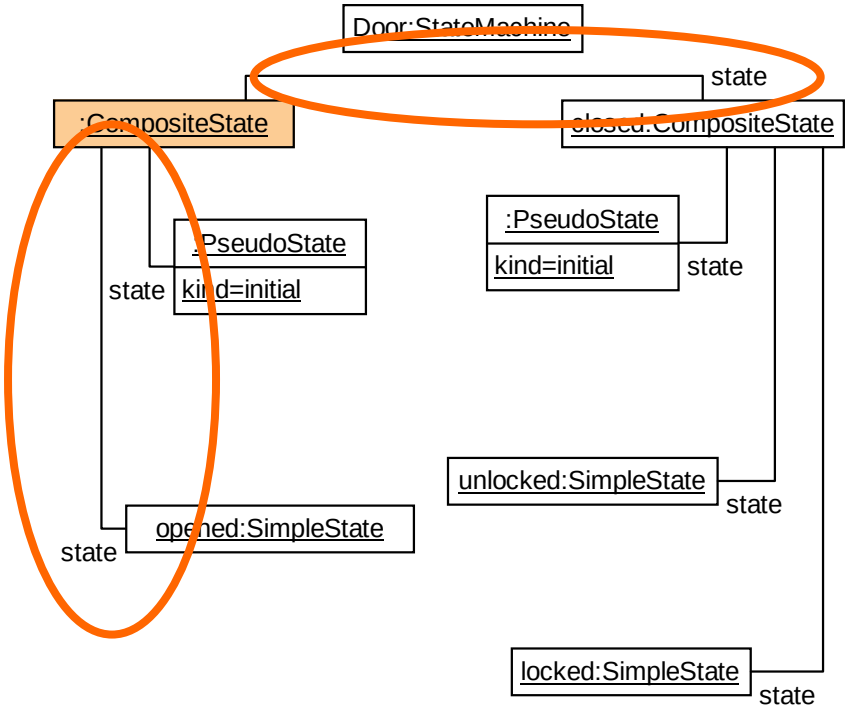
Example: Analysis

- Rule stack
  - StateMachine (Door)
  - stateR
  - csr (:CompositeState)

```
template csr for CompositeState ::=
init "CompositeState" self.name "{"
self.state:=(stateR)* "}"
```

StateMachine Door

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    …
  }
  …
}



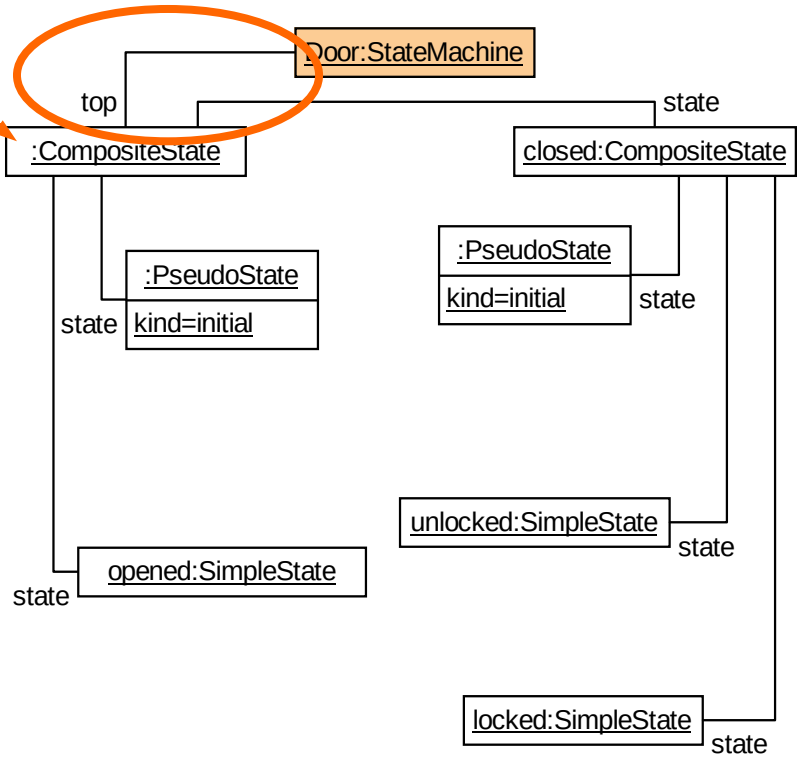
Example: Analysis

- Rule stack
  - StateMachine (Door)

```
start template for StateMachine ::=
"StateMachine" self.name self.top:=stateR;
```

```
StateMachine Door

CompositeState {
  initial State opened
  CompositeState closed {
    initial State unlocked
    State locked
    ...
  }
  ...
}
```

Result of `stateR`

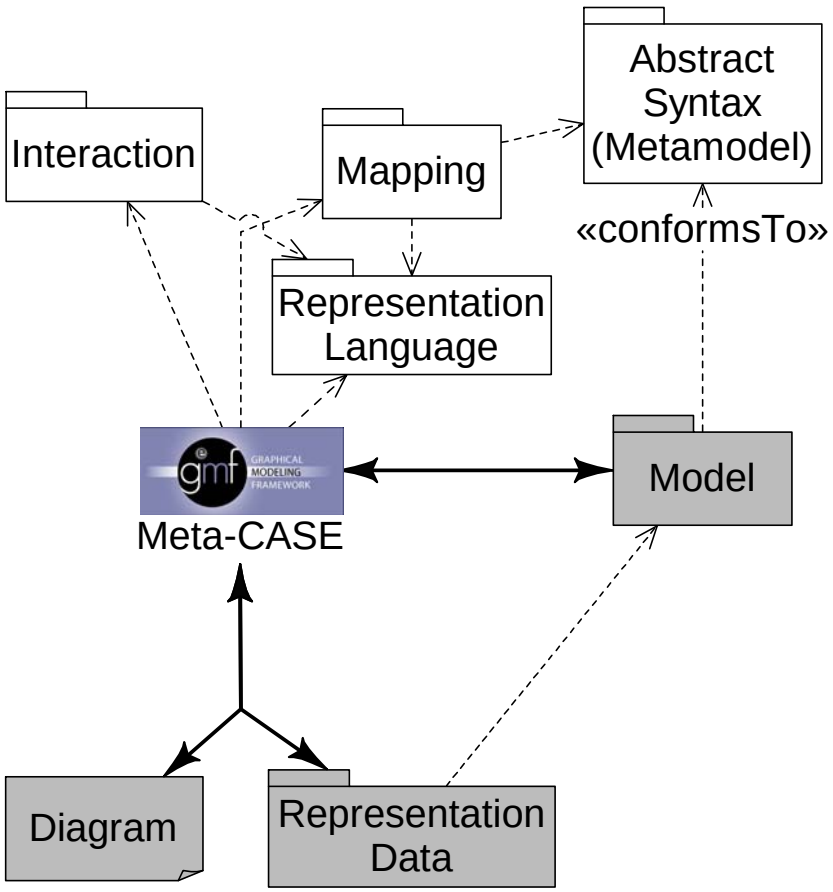


Emergency Slides

- Netsilon Details
- Language Definition
  - Textual Concrete Syntax
  - Graphical Concrete Syntax
- MTL Aspects
- Adding an additional abstraction layer
- Adaptors

Problems

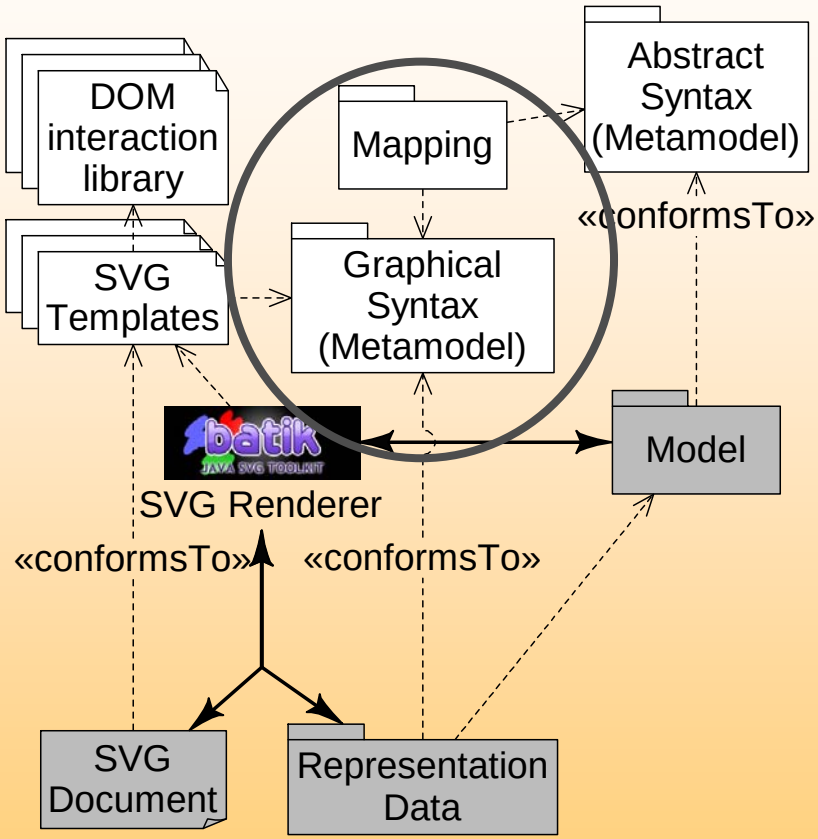
White: M2
Grey: M1



Usually:

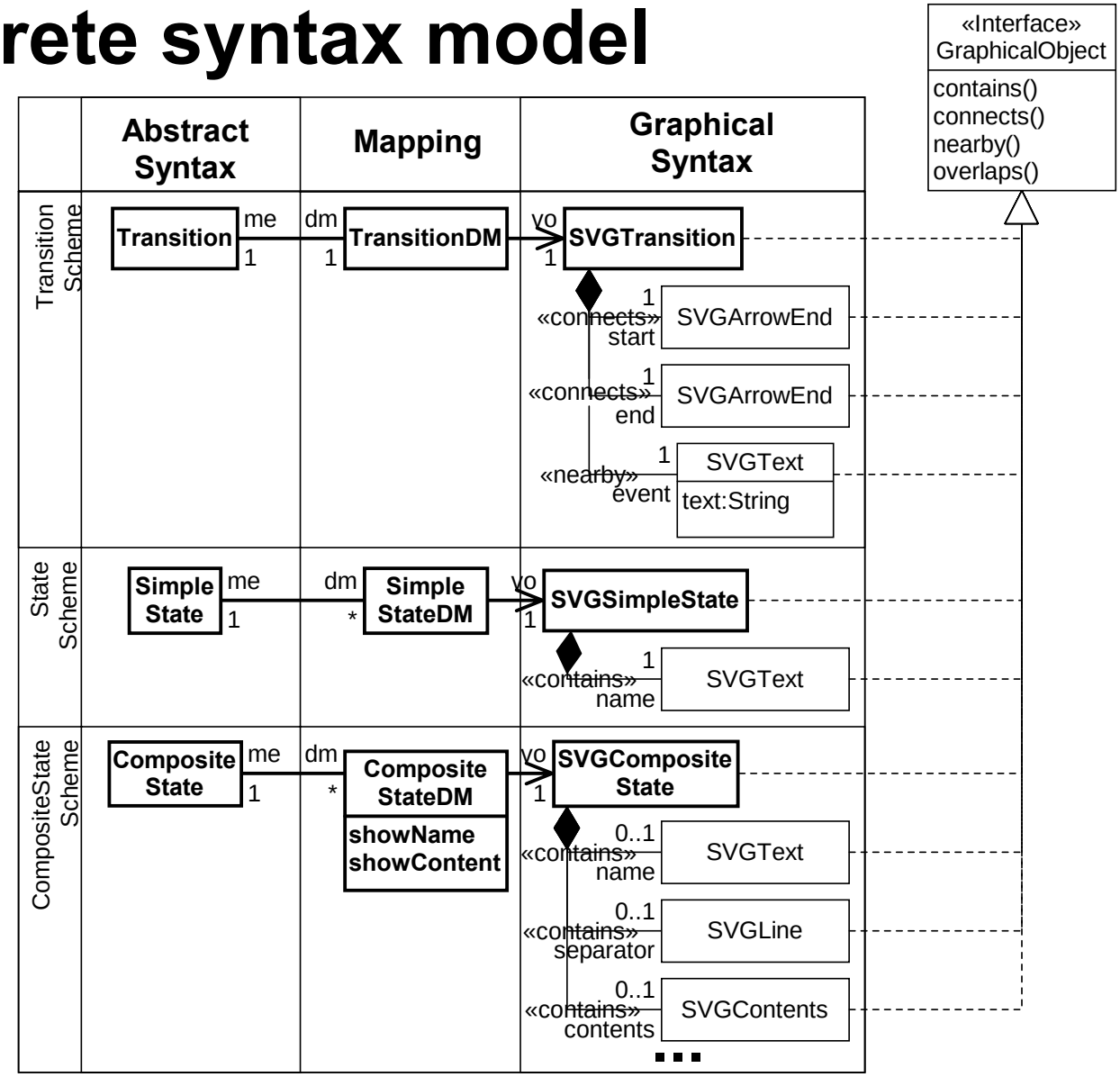
- Limited to connection-based languages
- Proprietary representation language
- Unclear representation data structure
- Recurrent interactions (if defined at all !)

Graphical concrete syntax definition



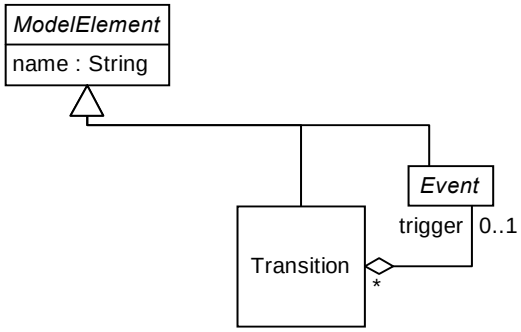
- Concrete syntax model
  - Fixes concrete syntax elements
  - Fixes relationship with abstract syntax
- Concrete syntax graphical design
  - Fixes appearance
  - Fixes layout constraints
  - Fixes edition facilities
  - Fixes link with concrete syntax model

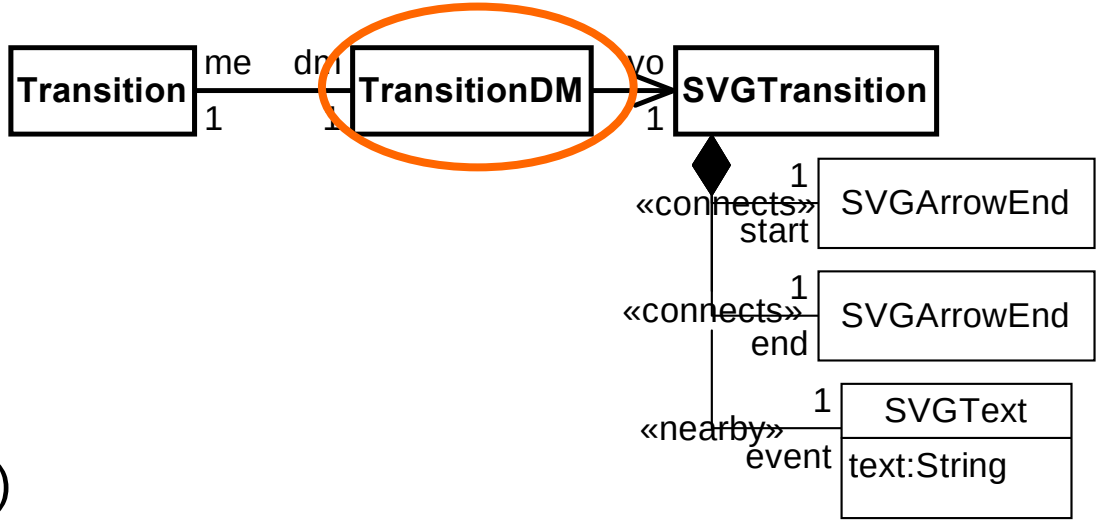
Concrete syntax model



Concrete syntax model

A text is shown on the top of transitions to represent the triggering event if it exists.

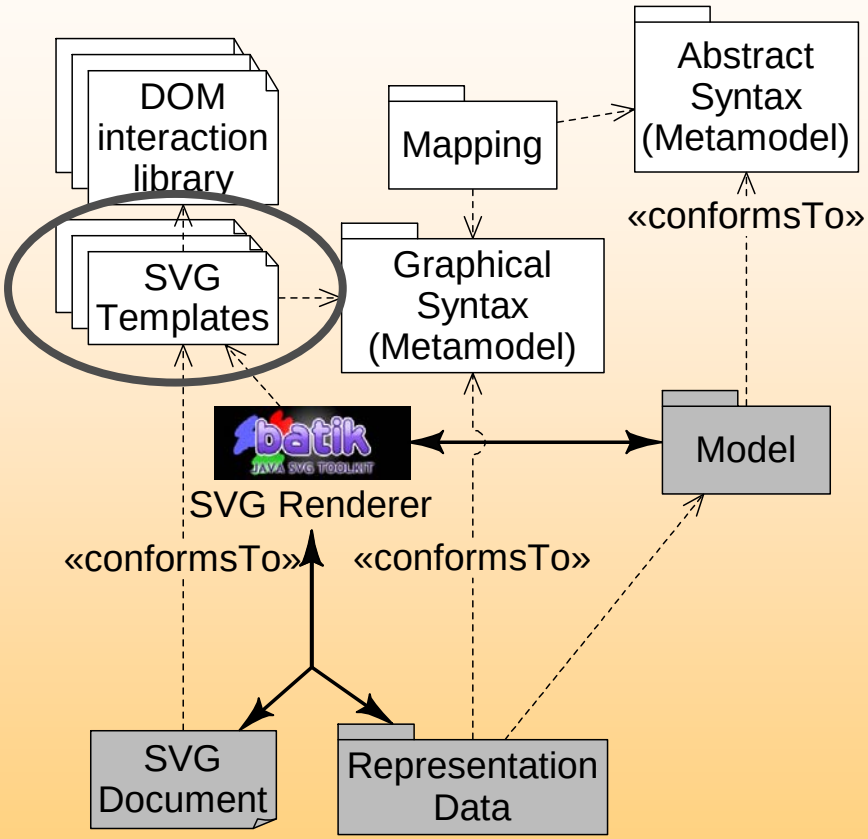




context TransitionDM **inv**:
  if self.me.trigger−>isEmpty()
  then self.vo.event.text.size() = 0
  else self.vo.event.text = self.me.trigger.name
  endif

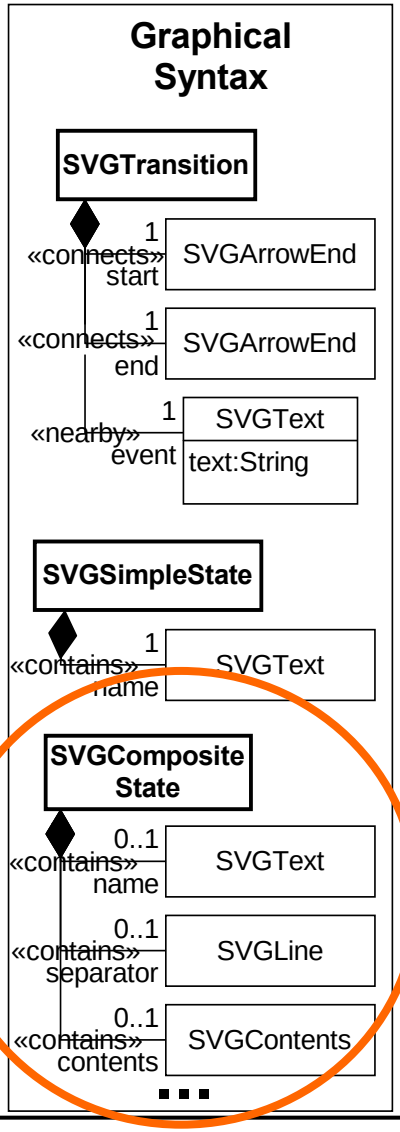
- Implementation issues

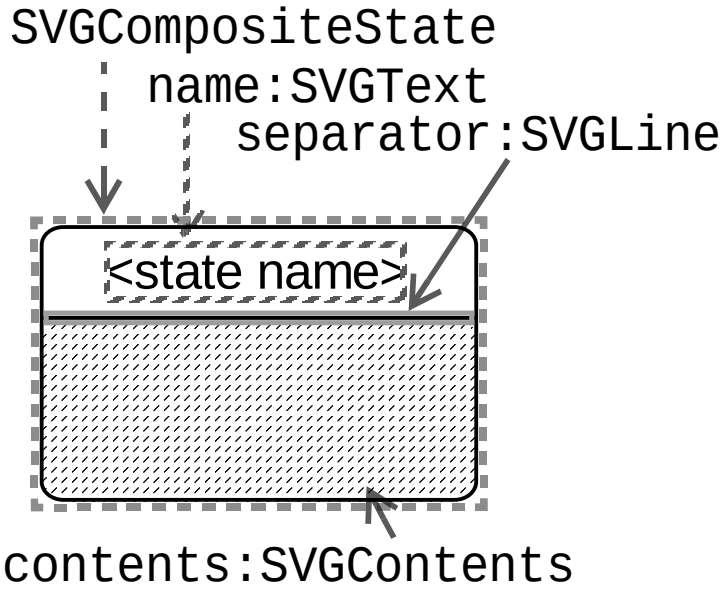
Graphical concrete syntax definition



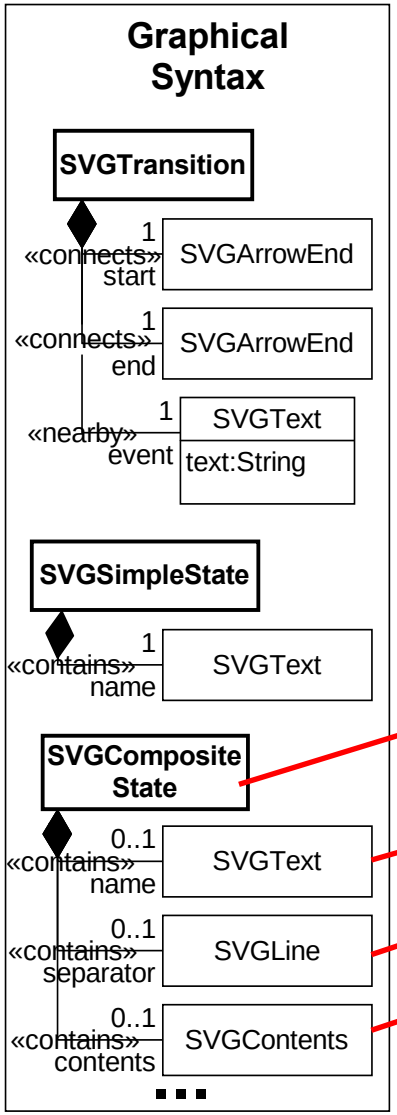
- Concrete syntax model
  - Fixes concrete syntax elements
  - Fixes relationship with abstract syntax
- Concrete syntax graphical design
  - Fixes appearance
  - Fixes layout constraints
  - Fixes edition facilities
  - Fixes link with concrete syntax model

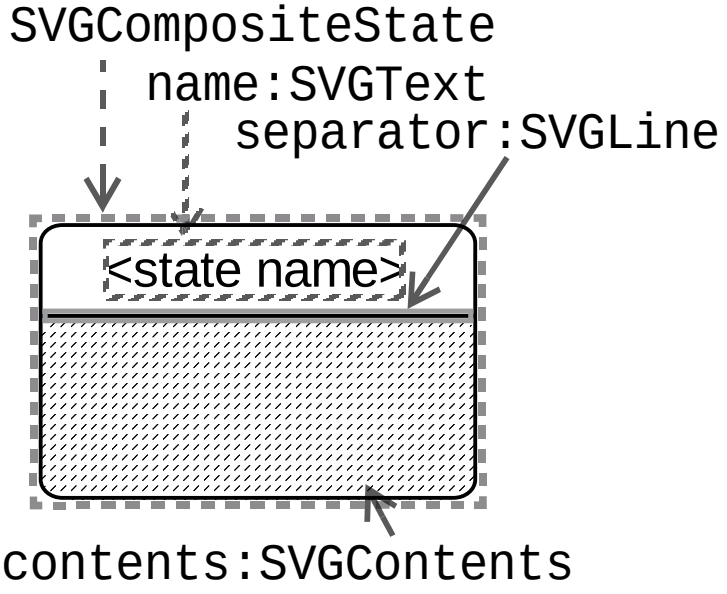
Solving appearance





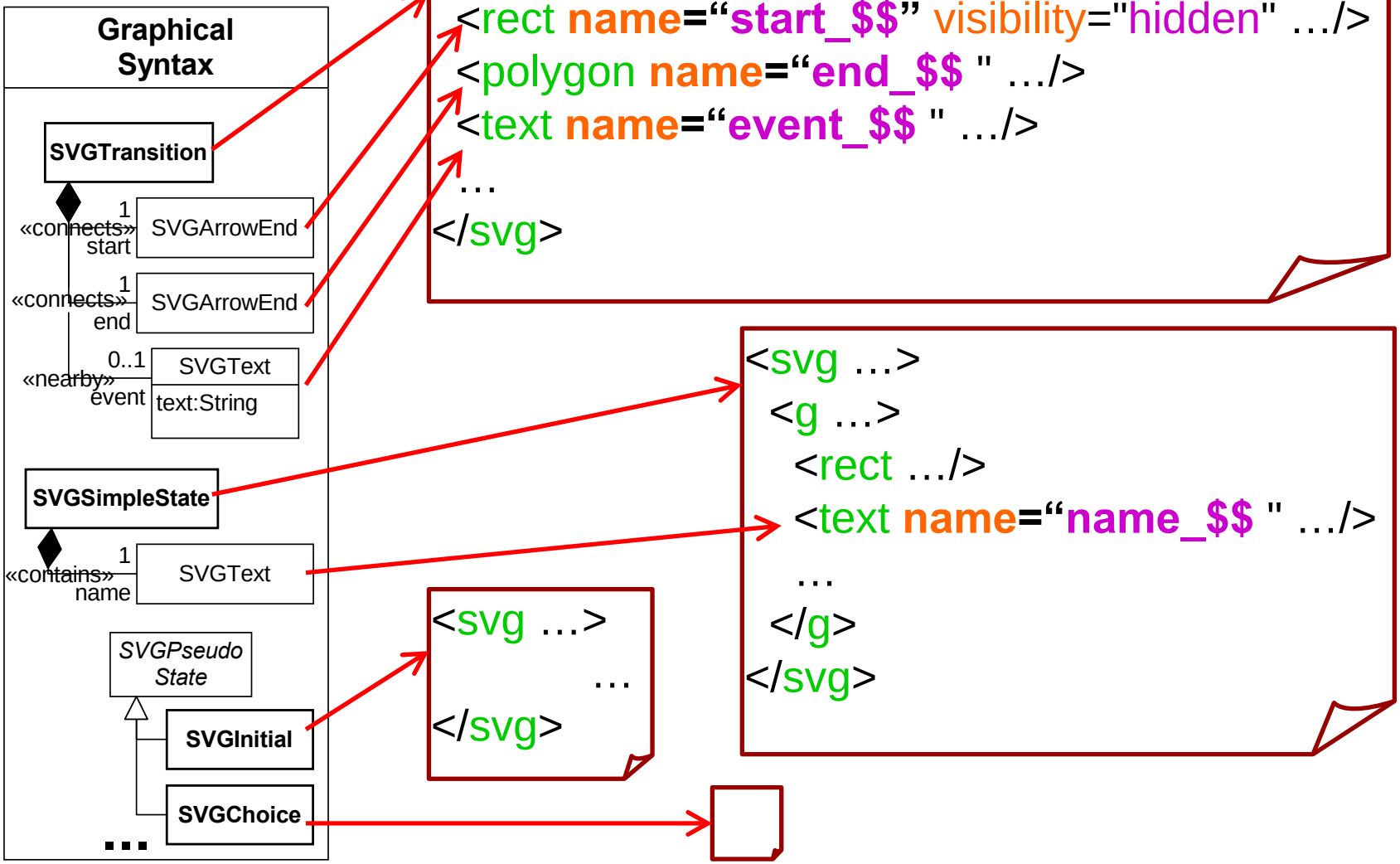
Solving appearance



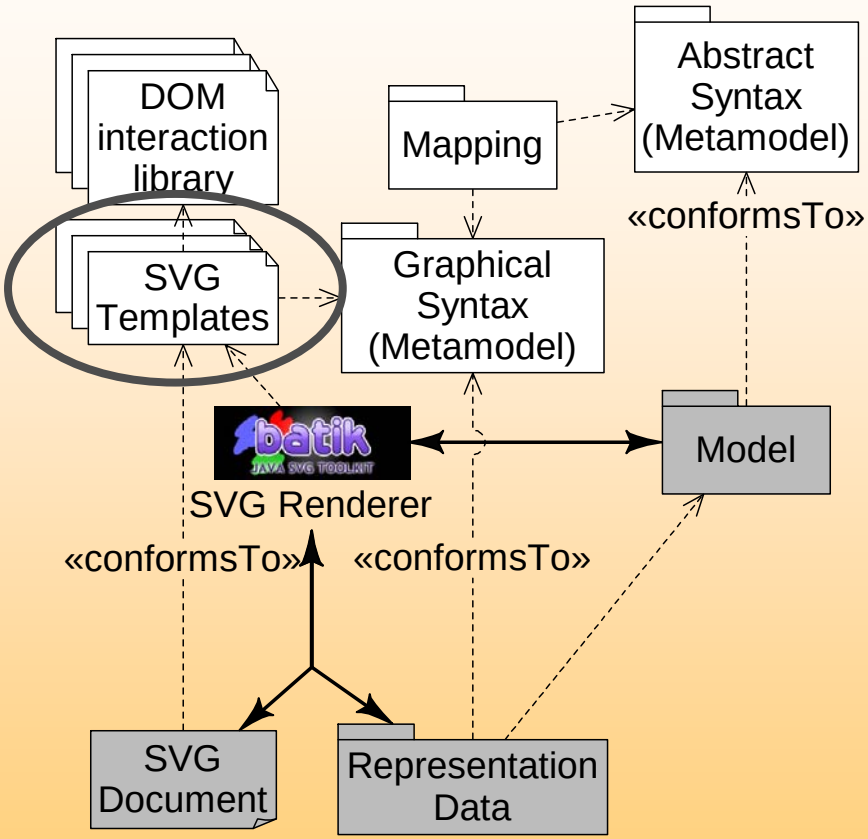



```
<svg …>
 < g id="$$" …>
   <rect id="back_$$" …/>
   <text id="name_$$ " …/>
   <line id="end_$$ " …/>
   <rect id="contents_$$" …/>
   …
 </g>
</svg>
```

Solving appearance



Graphical concrete syntax definition



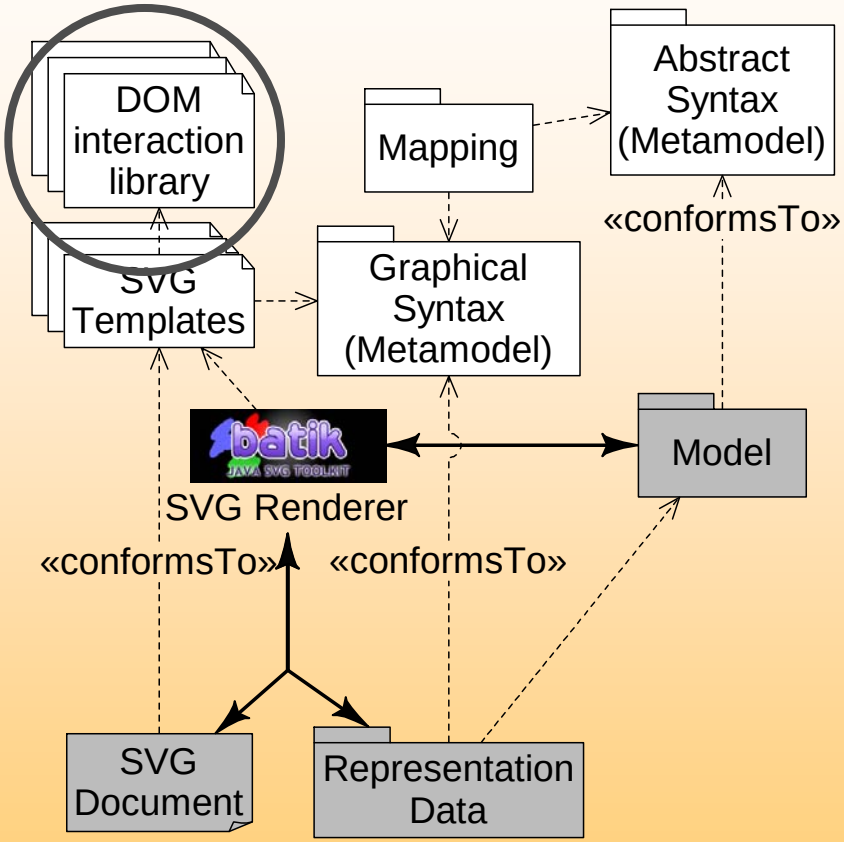
- Concrete syntax model
  - Fixes concrete syntax elements
  - Fixes relationship with abstract syntax
- Concrete syntax graphical design
  - Fixes appearance
  - Fixes layout constraints
  - Fixes edition facilities
  - Fixes link with concrete syntax model

Solving layout constraints

- OCL on graphical syntax metamodel => between elts
- C-SVG : one-way constraints (from Monash Uni.)
  `CompositeState` Template:
  Background should not be smaller than text.

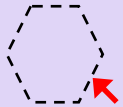
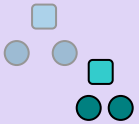
```
<svg …>
 <csvg:variable name="w_$$“
  value="c:max(c:width(c:bbox(id('name_$$')))  + 20, 150)"/>
 <rect …>
  <csvg:constraint attributeName="width" value="$w_$$"/>
 </rect>
 <text name=“name_$$” …>
  <csvg:constraint attributeName="x" value="$w_$$ div 2 - 75"/>
 </text>
 …
</svg>
```

Graphical concrete syntax definition



- Concrete syntax model
  - Fixes concrete syntax elements
  - Fixes relationship with abstract syntax
- Concrete syntax graphical design
  - Fixes appearance
  - Fixes layout constraints
  - Fixes edition facilities
  - Fixes link with concrete syntax model

DopiDOM components library

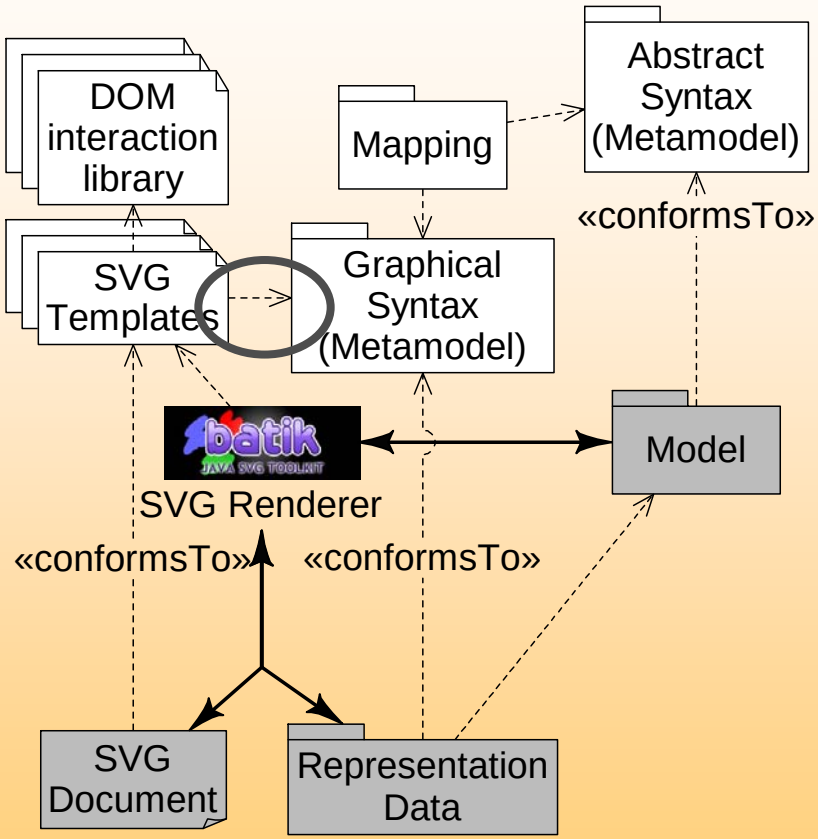
Interface		Interface	
BorderSlidable		Stickable	
DirectionAdjustable		Translatable	
Locatable		BorderFindable	
Positionable		OriginGettable	
Containable		Container	
Editable		Etc...	

Solving edition facilities

`CompositeState` template

```
<svg …>
 <g dpi:component=“Containable, Translatable, …” …>
  <rect dpi:component=“BorderFindable, …”…/>
  <rect dpi:component=“Container, …”…/>
  <text dpi:component=“Editable, …” …/>
  …
 </g>
</svg>
```

Graphical concrete syntax definition



- Concrete syntax model
  - Fixes concrete syntax elements
  - Fixes relationship with abstract syntax
- Concrete syntax graphical design
  - Fixes appearance
  - Fixes layout constraints
  - Fixes edition facilities
  - Fixes link with concrete syntax model

Representation Link: DopiDOM events

- Events depend on DopiDOM component
- Reaction to events defined in templates
  - Java JMI or EMF, KerMETA, Xion, etc.
- Initial / Load / Save scripts

`CompositeState` template

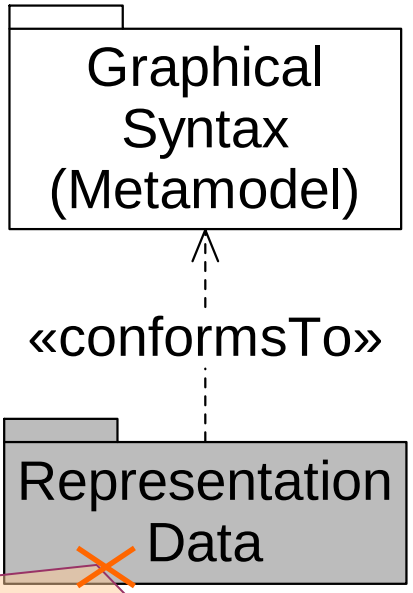
```
<svg
onCreation="s=model.getCompositeStateDM().createCompositeStateDM();
">
 <text name="name_$$" var_self="$s" dpi:component="Editable, …"
  onChange="self.setName(content);"…/>
 …
</svg>
```

Representation Link: Value events

- One listener synchronizing
  - An attribute value on the model with
  - an attribute value on the SVG document

`CompositeState` template revisited

```
<svg onCreation="s= …" … ></g … >
 <text name="name_$$" value="" …>
  <csvg:val value="../@value" />
  <updater
        attributeName="value"
        var_source="$s"
        slot="name" />
 </text>
 …
</g></svg>
```

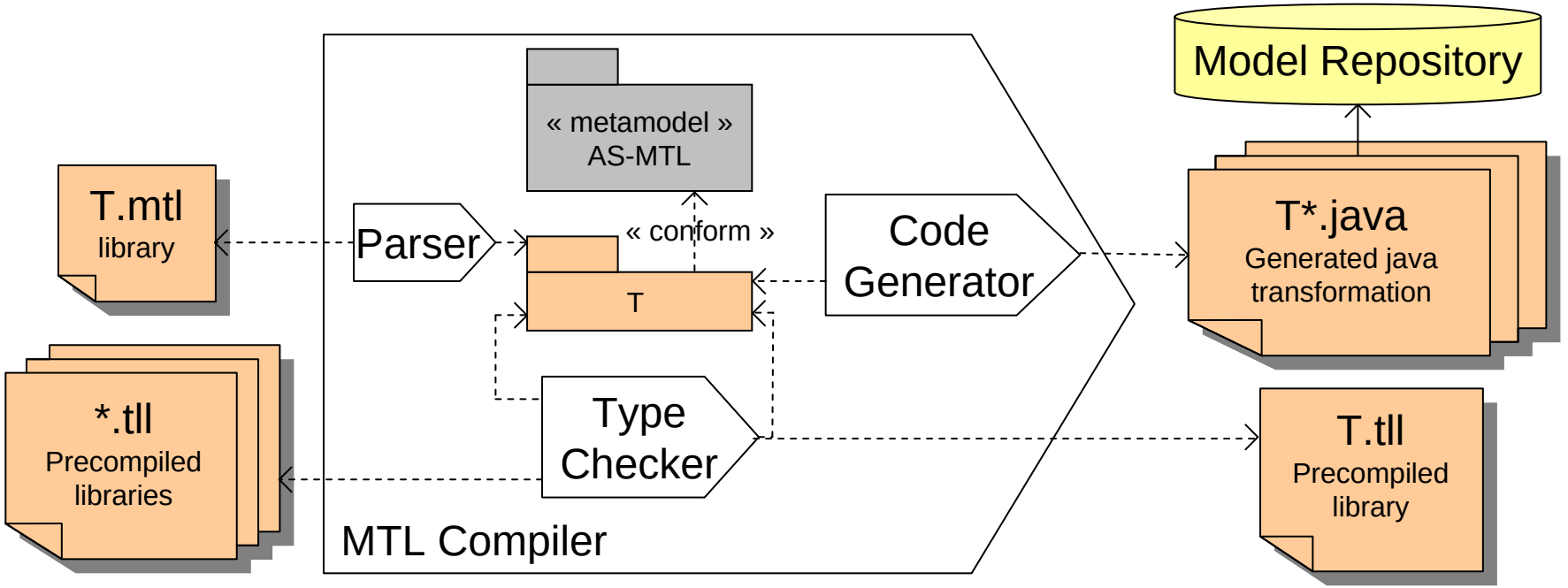


Emergency Slides

- Netsilon Details
- Language Definition
  - Textual Concrete Syntax
  - Graphical Concrete Syntax
- MTL Aspects
- Adding an additional abstraction layer
- Adaptors

Model Transformation Language (MTL)

- Object-Oriented Imperative Language



INRIA

Transforming an MTL Transformation

- Source Transformation

```
library MyTransformation;
main() : Standard::Void {
  new Transformer().run();
}
```

- Target Transformation

```
library MyTransformedTransformation;
main() : Standard::Void {
  new Transformer().run();
  'Message from the transformed transformation !'.toOut();
}
```

The Standard MTL Approach

```
lib : BasicMtlASTView::BasicMtlLibrary;
lib.name := 'MyTransformedTransformation';
foreach (op : BasicMtlASTView::Operation)
     in (lib.definedOperations)
  where (op.name.[=]('main')) {
    sl := new BasicMtlASTView::StringLiteral();
    sl.value := 'Message from the transformed transformation !';
    oi := new BasicMtlASTView::OperationCall();
    oi.name := 'toOut';
    oi.caller := sl;
    oi.arguments := newOrderedSet();
    op.instructions := op.instructions.append(oi);
}
```

Aspect-Oriented Programming (AOP)

- Aspects ≈ Transformation of Code
  - *Where* to change: *Pointcut {Join Points}*
  - *What* are the changes: *Advice*
- Aspects' Formalism (e.g., AspectJ)
  - Concrete syntax of the base language (e.g., Java)
  - Additional constructs and keywords (e.g., *aspect*, *after*, *pointcut*, etc.)
  - Easy to learn

MTL Extensions & MTL-Aspects

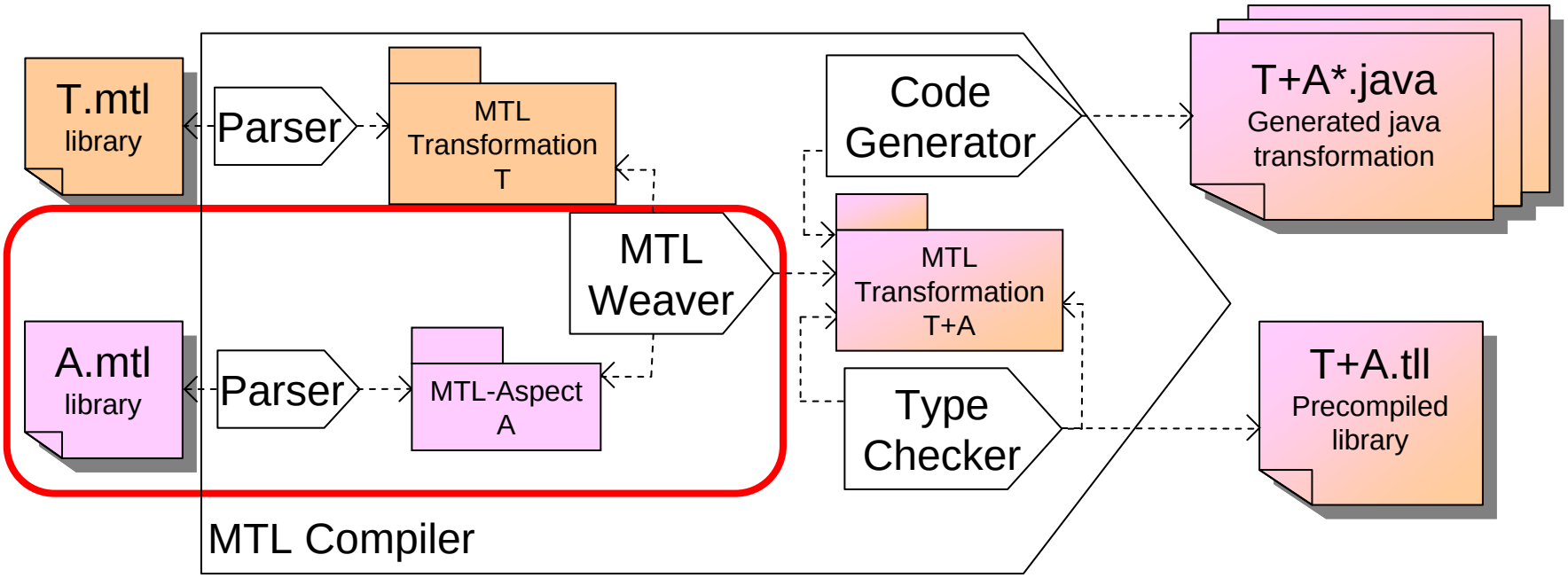
- We need here to extend the MTL language !

- The **Tag** MTL extension mechanism
  - key/values pair on an MTL element
  - part of the MTL Metamodel → the same MTL Parser
  - visibility of tagged elements in the model

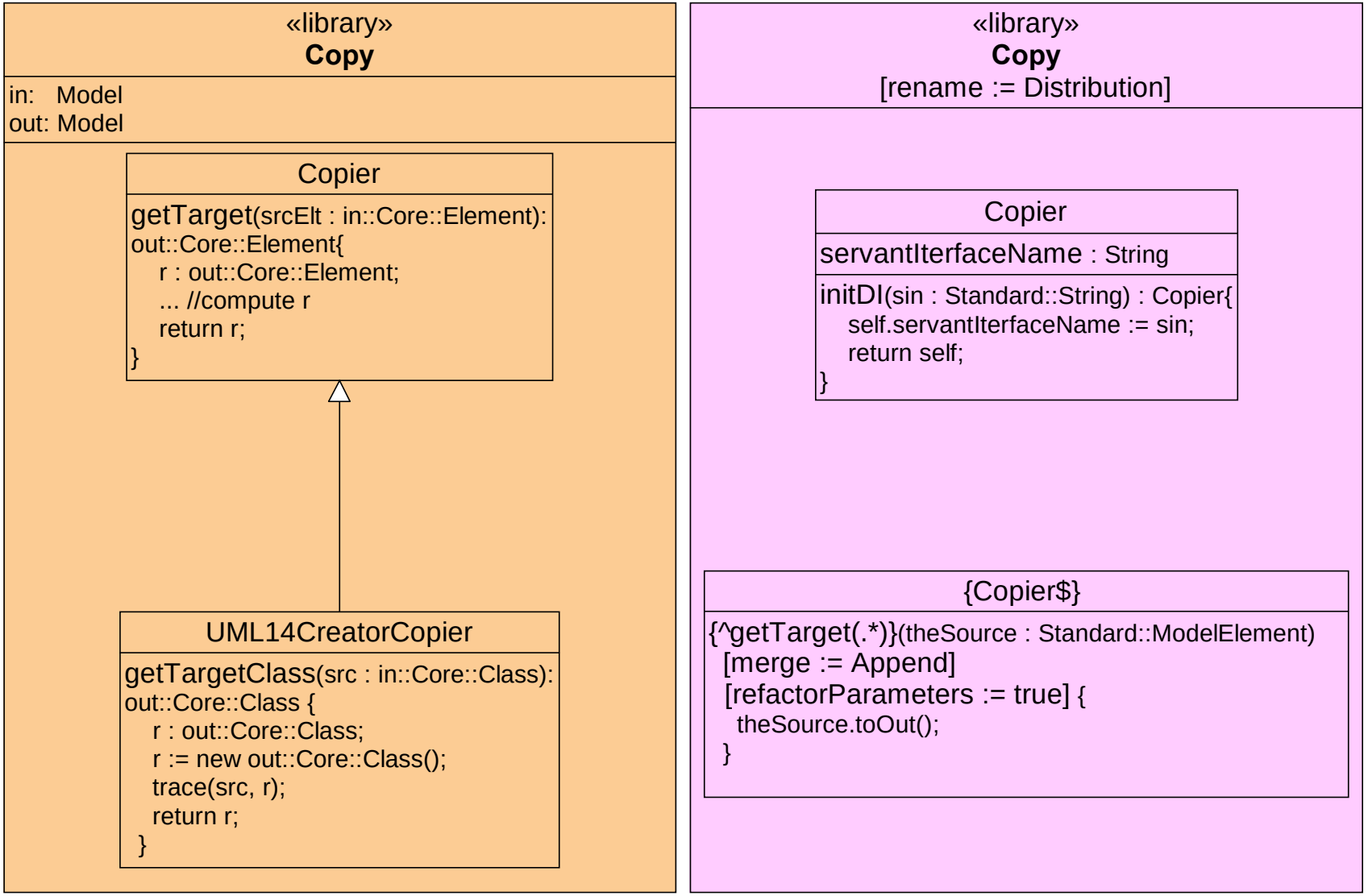
- MTL-Aspects: rely on the definition of new *tags*
  - Abstract syntax: **no difference** !
  - Concrete syntax: **no difference** !
  - Semantics: **different** !

Aspects on MTL Transformations

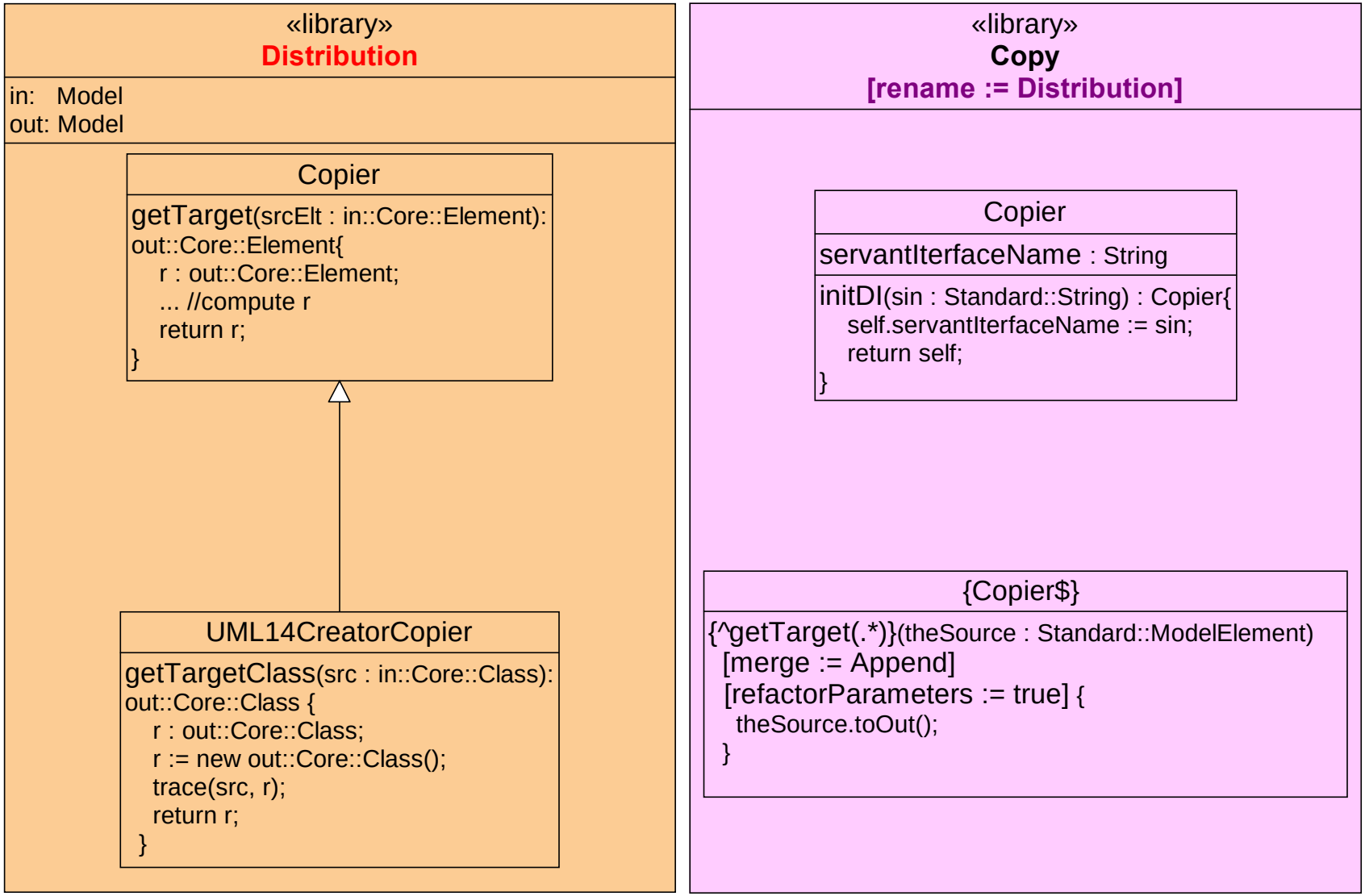
- Language extensions for defining aspects on transformation
- Enhance slightly the compilation process



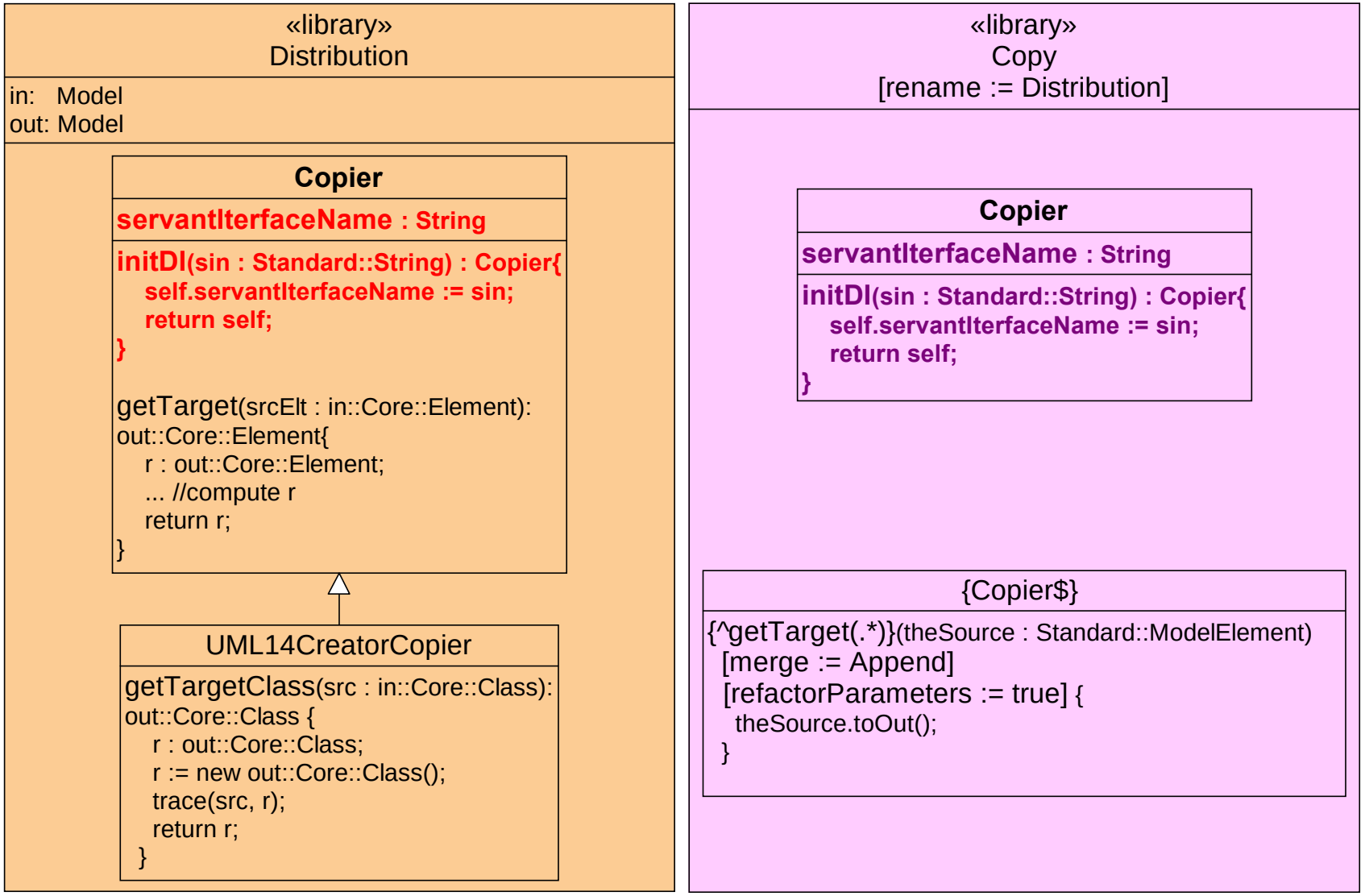
An Example


«library»
Copy
in: Model
out: Model
Copier
getTarget(srcElt : in::Core::Element):
out::Core::Element{
r : out::Core::Element;
... //compute r
return r;
}
UML14CreatorCopier
getTargetClass(src : in::Core::Class):
out::Core::Class {
r : out::Core::Class;
r := new out::Core::Class();
trace(src, r);
return r;
}
«library»
Copy
[rename := Distribution]
Copier
servantIterfaceName : String
initDI(sin : Standard::String) : Copier{
self.servantIterfaceName := sin;
return self;
}
{Copier$}
{^getTarget(.*)}(theSource : Standard::ModelElement)
[merge := Append]
[refactorParameters := true] {
theSource.toOut();
}

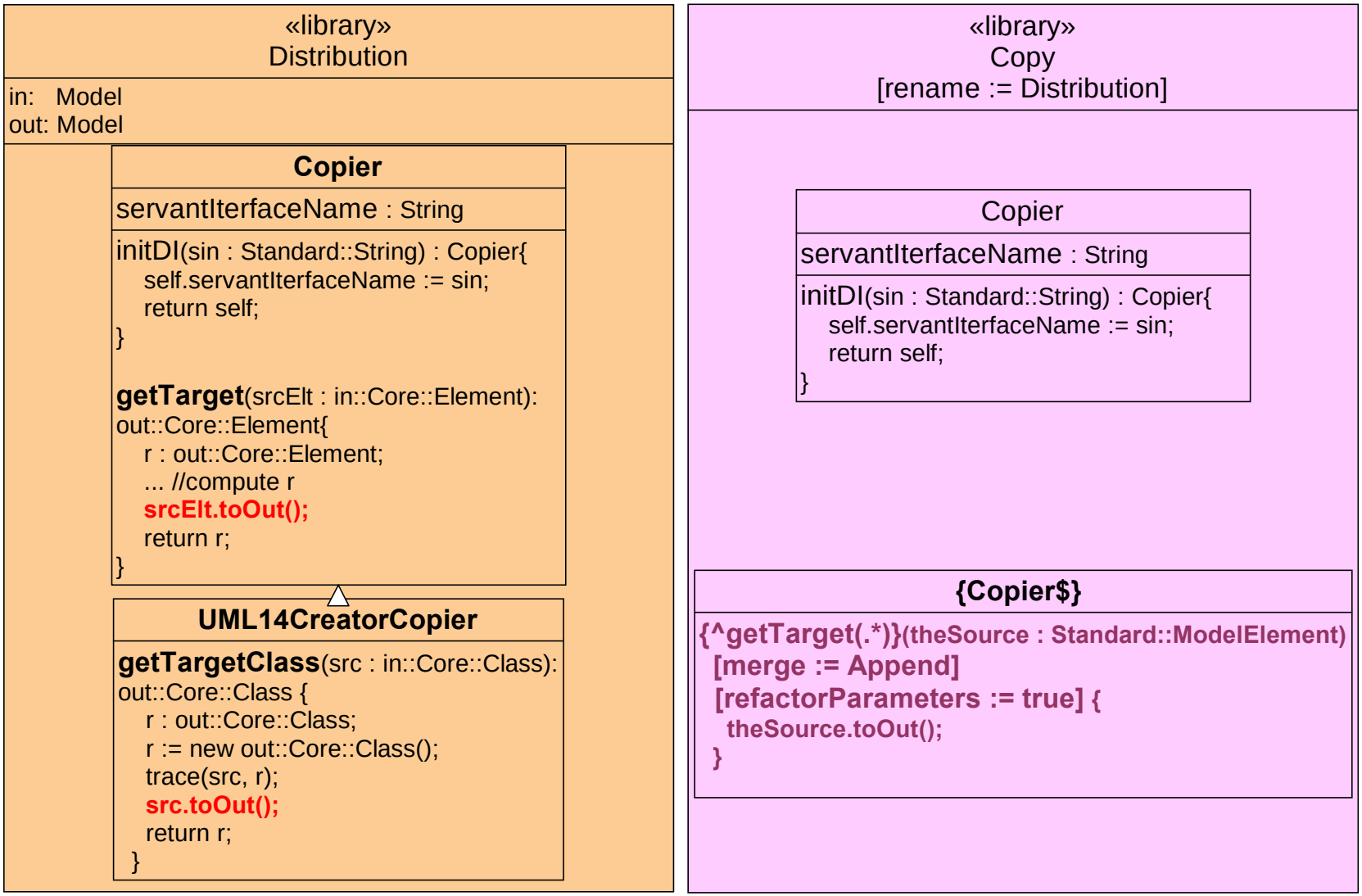
An Example


«library»
Distribution
in: Model
out: Model
Copier
getTarget(srcElt : in::Core::Element):
out::Core::Element{
r : out::Core::Element;
... //compute r
return r;
}
UML14CreatorCopier
getTargetClass(src : in::Core::Class):
out::Core::Class {
r : out::Core::Class;
r := new out::Core::Class();
trace(src, r);
return r;
}
«library»
Copy
[rename := Distribution]
Copier
servantIterfaceName : String
initDI(sin : Standard::String) : Copier{
self.servantIterfaceName := sin;
return self;
}
{Copier$}
{^getTarget(.*)}(theSource : Standard::ModelElement)
[merge := Append]
[refactorParameters := true] {
theSource.toOut();
}

An Example


«library»
Distribution
in: Model
out: Model
Copier
servantIterfaceName : String
initDI(sin : Standard::String) : Copier{
self.servantIterfaceName := sin;
return self;
}
getTarget(srcElt : in::Core::Element):
out::Core::Element{
r : out::Core::Element;
... //compute r
return r;
}
UML14CreatorCopier
getTargetClass(src : in::Core::Class):
out::Core::Class {
r : out::Core::Class;
r := new out::Core::Class();
trace(src, r);
return r;
}
«library»
Copy
[rename := Distribution]
Copier
servantIterfaceName : String
initDI(sin : Standard::String) : Copier{
self.servantIterfaceName := sin;
return self;
}
{Copier$}
{^getTarget(.*)}(theSource : Standard::ModelElement)
[merge := Append]
[refactorParameters := true] {
theSource.toOut();
}

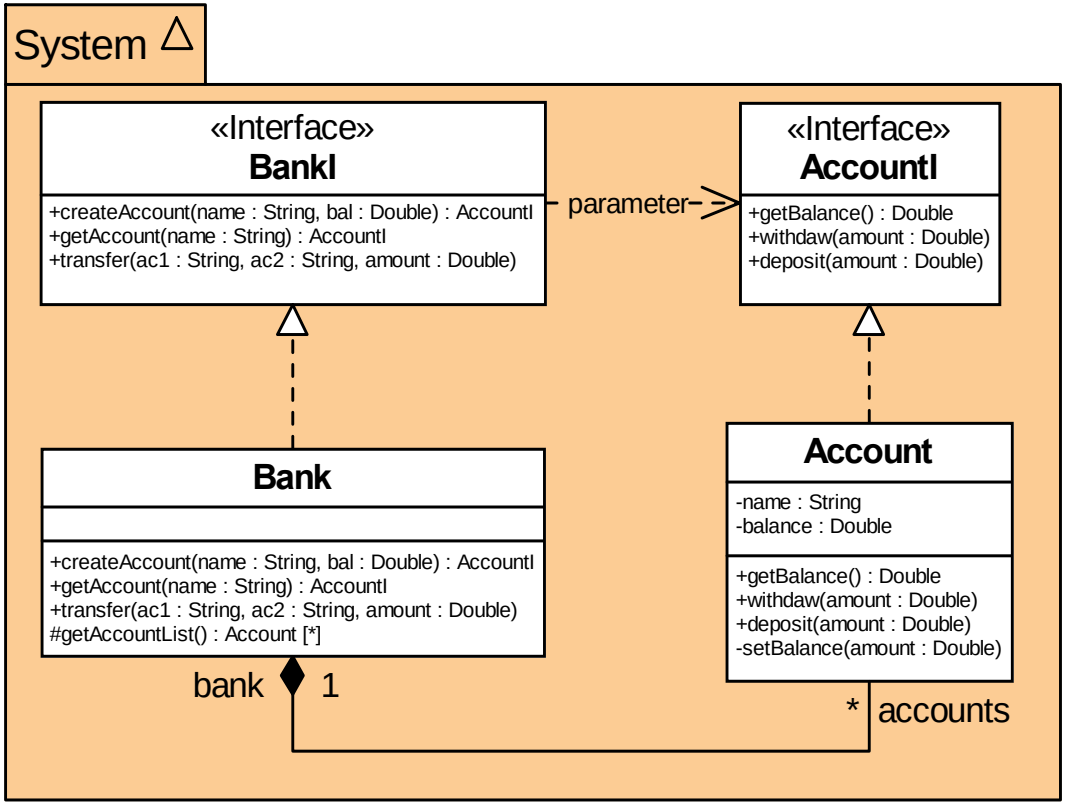
An Example


«library»
Distribution
in: Model
out: Model
Copier
servantIterfaceName : String
initDI(sin : Standard::String) : Copier{
self.servantIterfaceName := sin;
return self;
}
getTarget(srcElt : in::Core::Element):
out::Core::Element{
r : out::Core::Element;
... //compute r
srcElt.toOut();
return r;
}
UML14CreatorCopier
getTargetClass(src : in::Core::Class):
out::Core::Class {
r : out::Core::Class;
r := new out::Core::Class();
trace(src, r);
src.toOut();
return r;
}
«library»
Copy
[rename := Distribution]
Copier
servantIterfaceName : String
initDI(sin : Standard::String) : Copier{
self.servantIterfaceName := sin;
return self;
}
{Copier$}
{^getTarget(.*)}(theSource : Standard::ModelElement)
[merge := Append]
[refactorParameters := true] {
theSource.toOut();
}

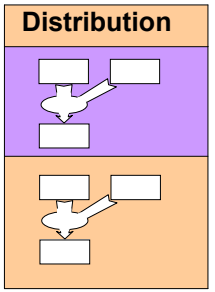
Emergency Slides

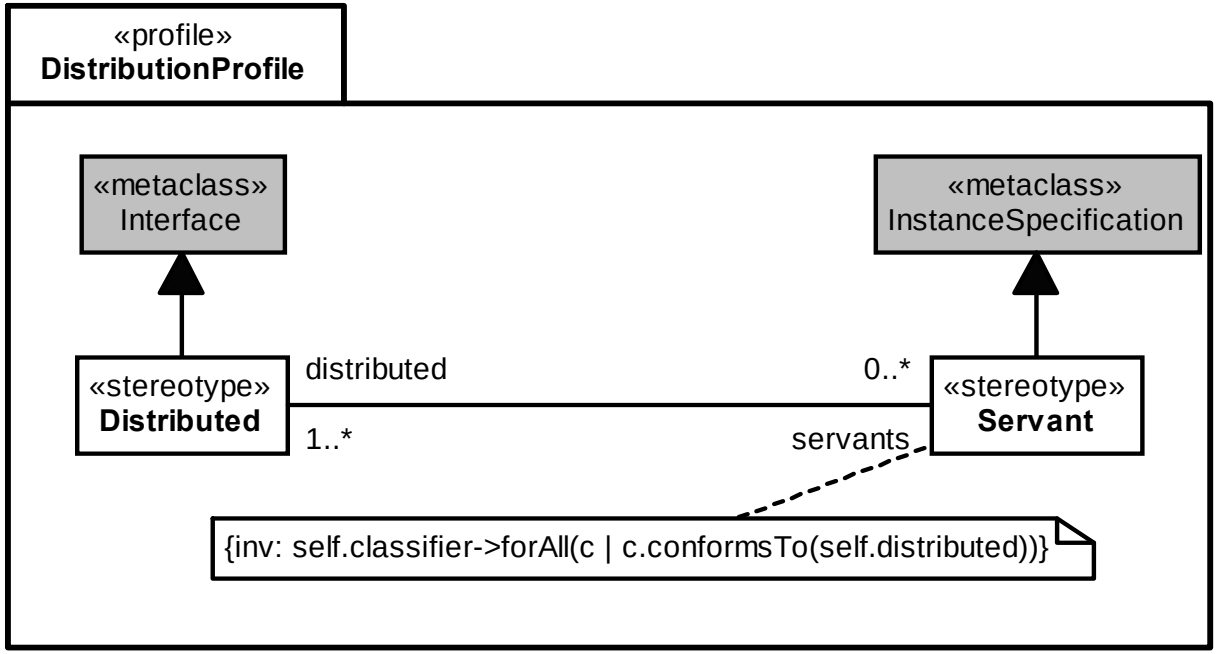
- Netsilon Details
- Language Definition
  - Textual Concrete Syntax
  - Graphical Concrete Syntax
- MTL Aspects
- Adding an additional abstraction layer
- Adaptors

An example

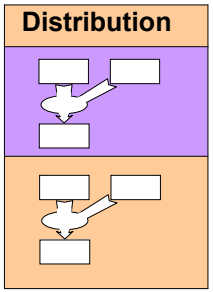

System
«Interface»
BankI
+createAccount(name : String, bal : Double) : AccountI
+getAccount(name : String) : AccountI
+transfer(ac1 : String, ac2 : String, amount : Double)
parameter
«Interface»
AccountI
+getBalance() : Double
+withdaw(amount : Double)
+deposit(amount : Double)
Bank
+createAccount(name : String, bal : Double) : AccountI
+getAccount(name : String) : AccountI
+transfer(ac1 : String, ac2 : String, amount : Double)
#getAccountList() : Account [*]
Account
-name : String
-balance : Double
+getBalance() : Double
+withdaw(amount : Double)
+deposit(amount : Double)
-setBalance(amount : Double)
bank
1
*
accounts

Distribution Profile

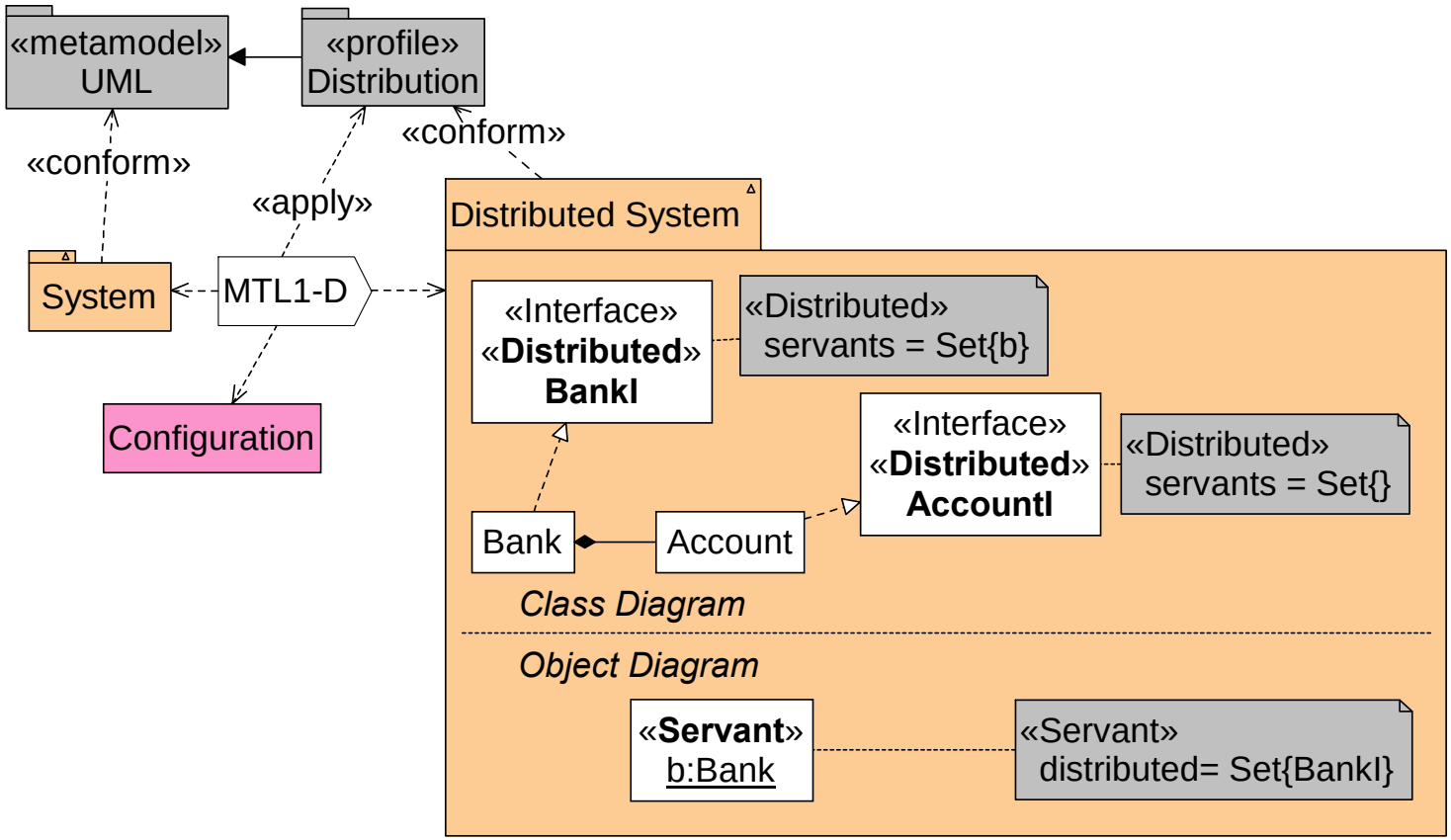




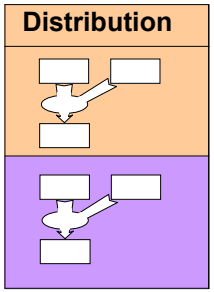
Platform integration



- “The BankI is distributed”



Abstract Distribution Realization



«profile»
DistributionProfile

«merge»

«profile»
AbstractDistributionRealizationProfile

«stereotype»
Servant

«metaclass»
Comment

«stereotype»
PublishedServant

servant

1

expositions

*

«stereotype»
Exposition

«metaclass»
Interface

1..*

exposedInterfaces

«stereotype»
Publisher

1

exposed

*

publisher

«stereotype»
ServiceRegistry

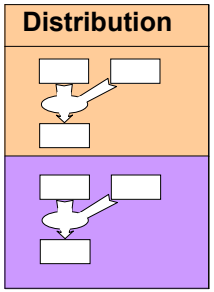
«stereotype»
NamingRegistry

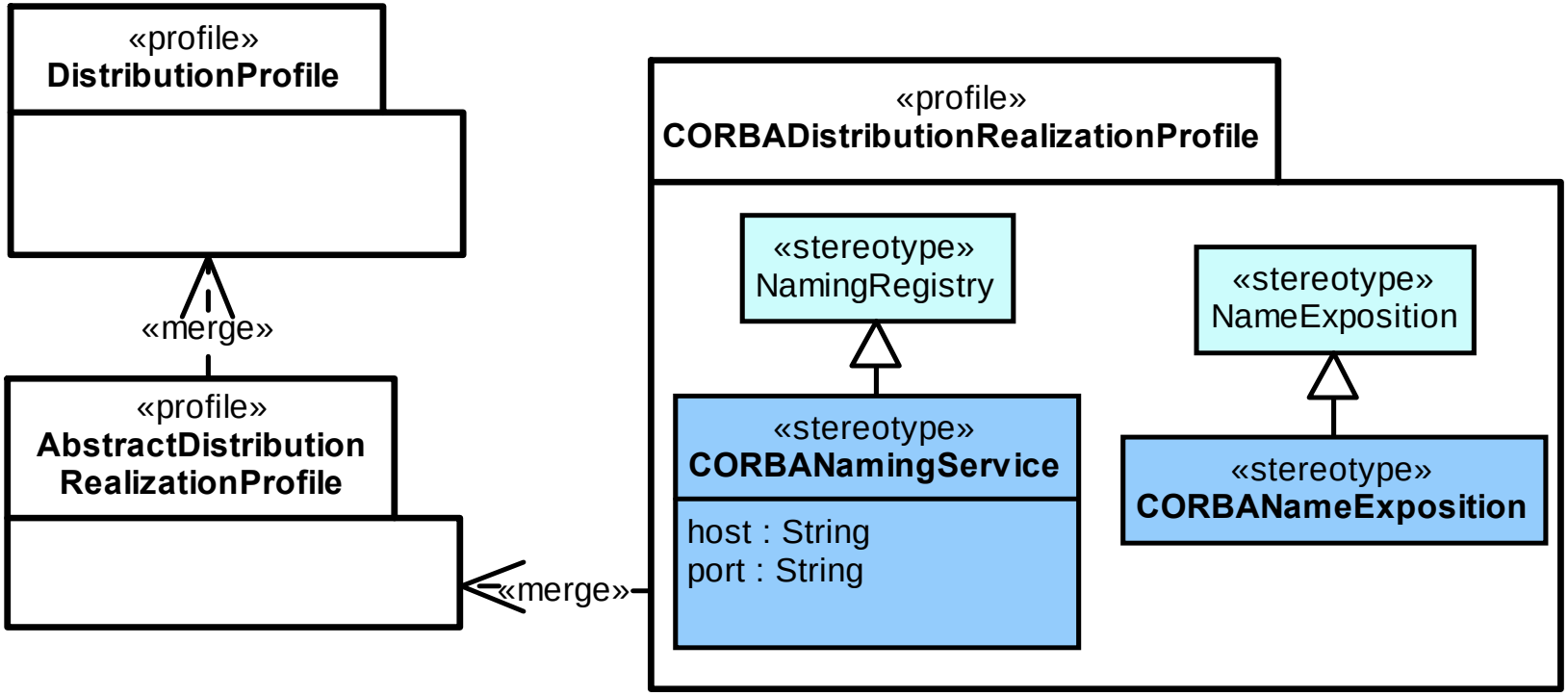
«stereotype»
InterfaceExposition

«stereotype»
NameExposition

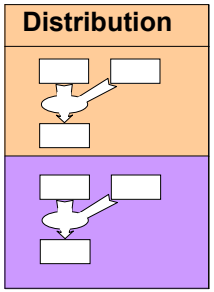
exposedNames : String [1..*]

CORBA Distribution Realization

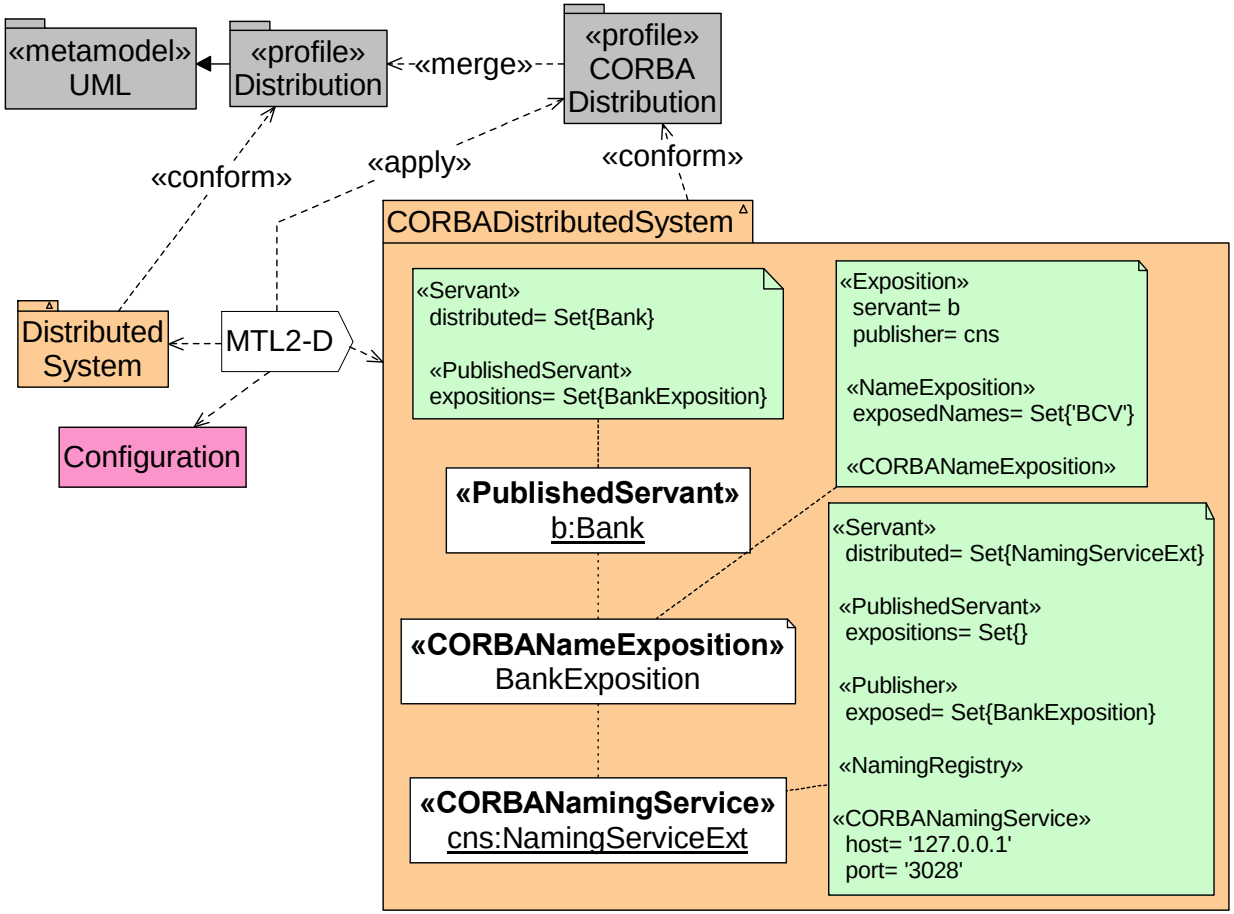




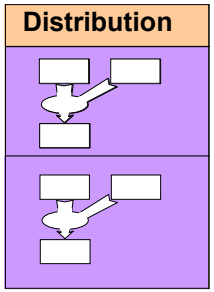
Platform integration

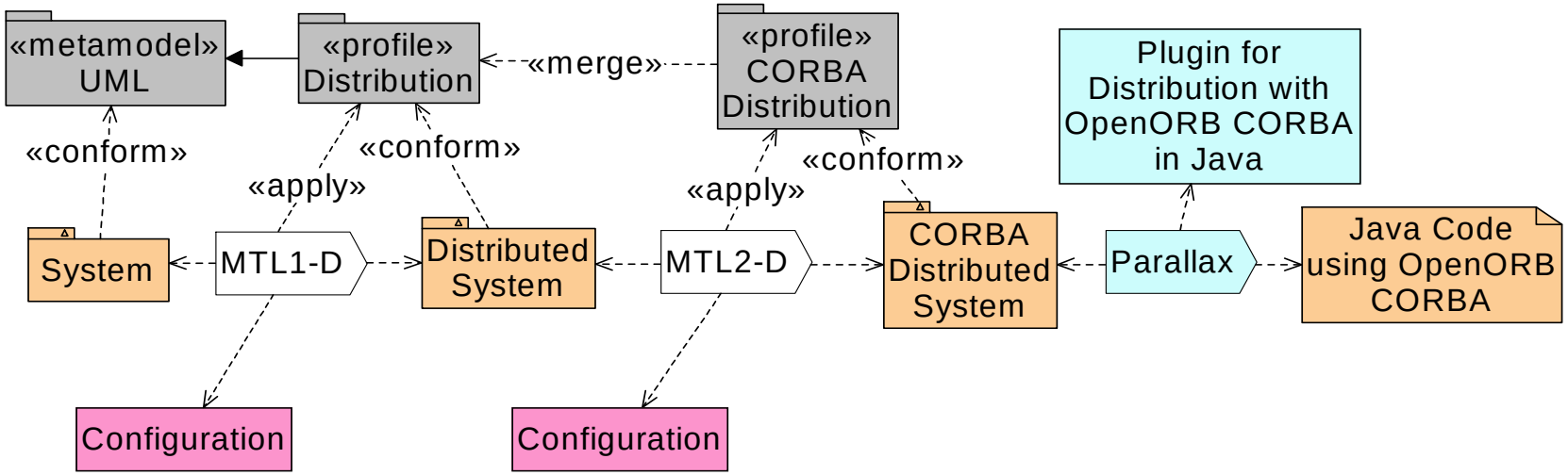


- “The b servant is bound to a CORBA naming service at 127.0.0.1, on port 3028, with name BCV”



Platform integration





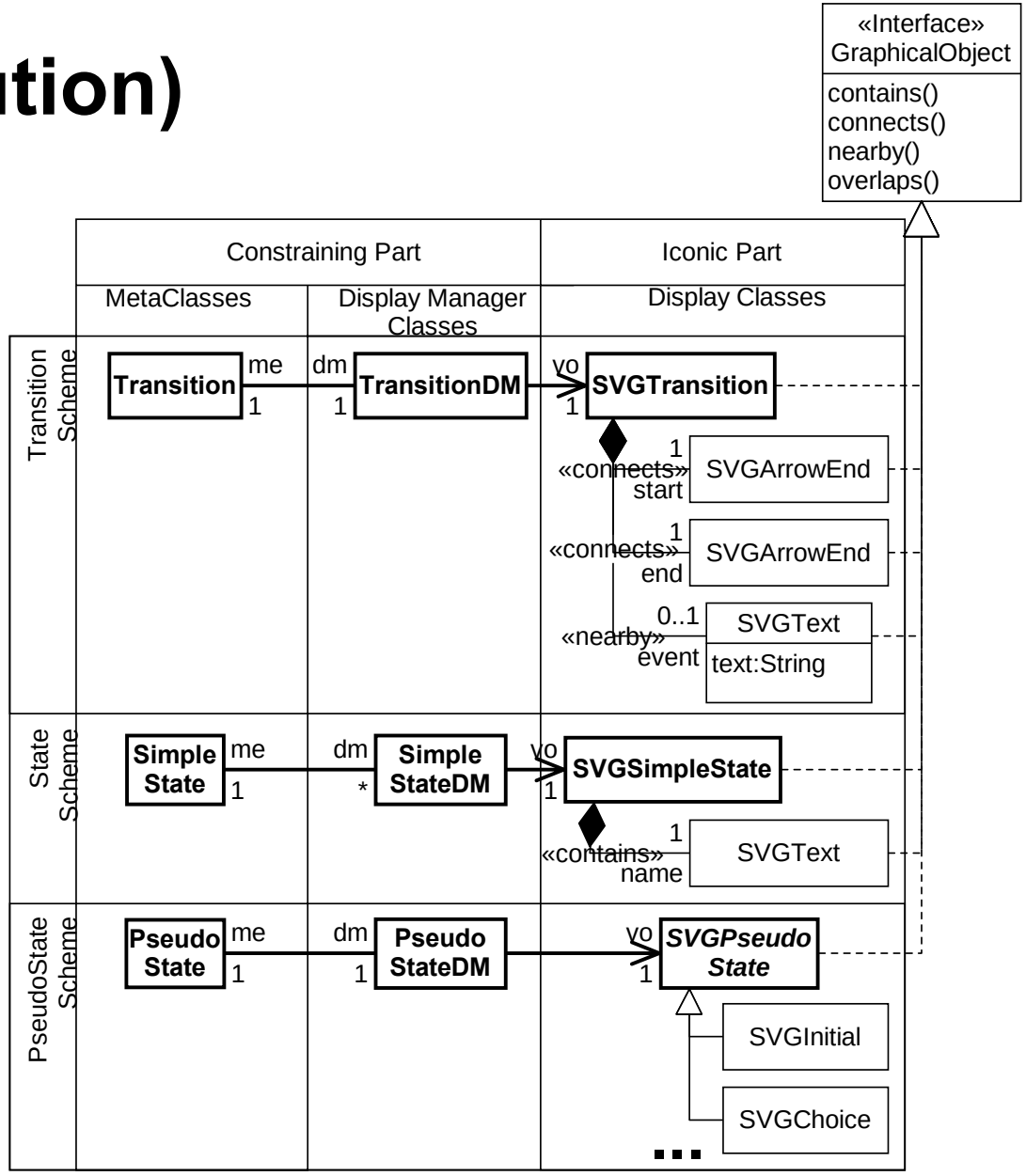
Emergency Slides

- Netsilon Details
- Language Definition
  - Textual Concrete Syntax
  - Graphical Concrete Syntax
- MTL Aspects
- Adding an additional abstraction layer
- Adaptors

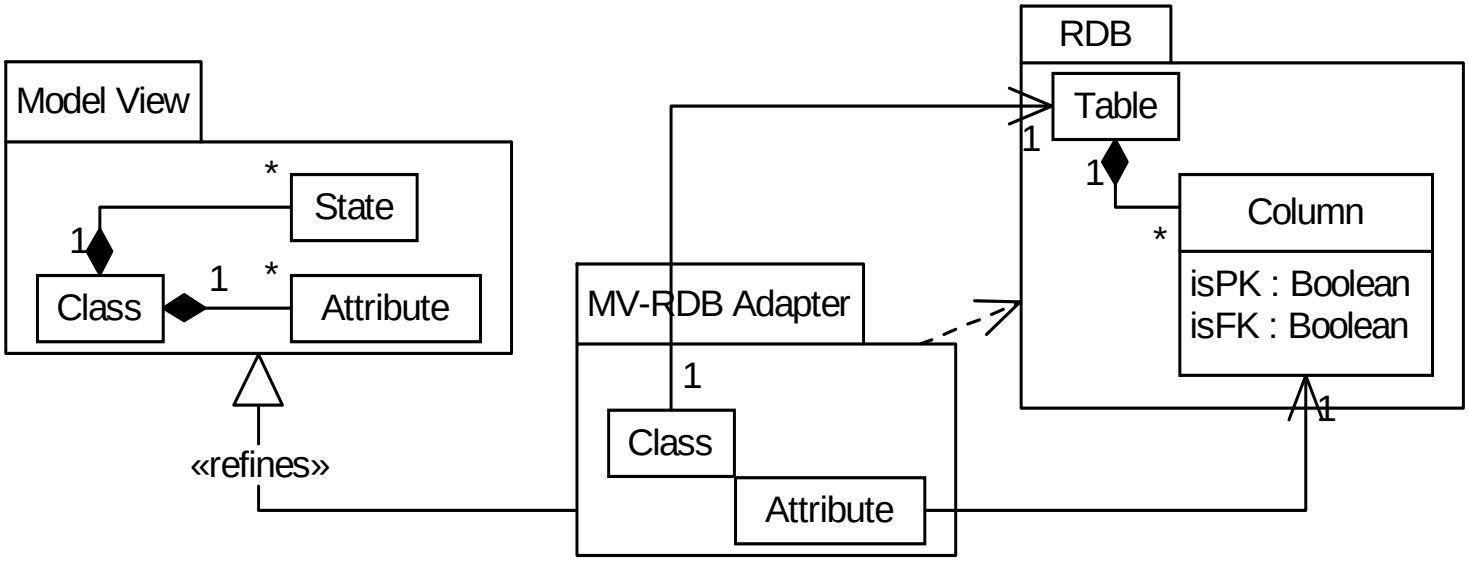
Adapters (1st solution)

- Synchronization through OCL constraints solving
- Implementation issues
- Here an example for synchronizing abstract with a concrete syntax model

```
context TransitionDM inv:
  if self.me.trigger->isEmpty()
  then self.vo.event->isEmpty()
  else self.vo.event.text
            = self.me.trigger.name
  endif
```



Adapters (2nd solution)




```
package MV-RDB_Adapter
   context Class
      inv : self.name = self.table.name
      inv : self.attribute.column = self.table.column->reject(isPK or isFK)
      inv : self.state->isEmpty
   context Attribute
      inv : self.name = self.attribute.name
endpackage
```

Adapters (Specification)

- Template-Based Engine
  - RHS Concept Recognition
- Read/Write View