Concrete Syntax Definition for Modeling Languages

Frédéric Fondement

PhD. thesis Public Defense

Swiss Federal Institute of Technology in Lausanne
Software Engineering Laboratory

November 2007

Contents

- Introduction
  - Software Engineering
  - Model Driven Engineering
  - Language Definition
- Concrete Syntaxes
  - Textual concrete syntax definition
  - Graphical concrete syntax definition
- Conclusions and outlook

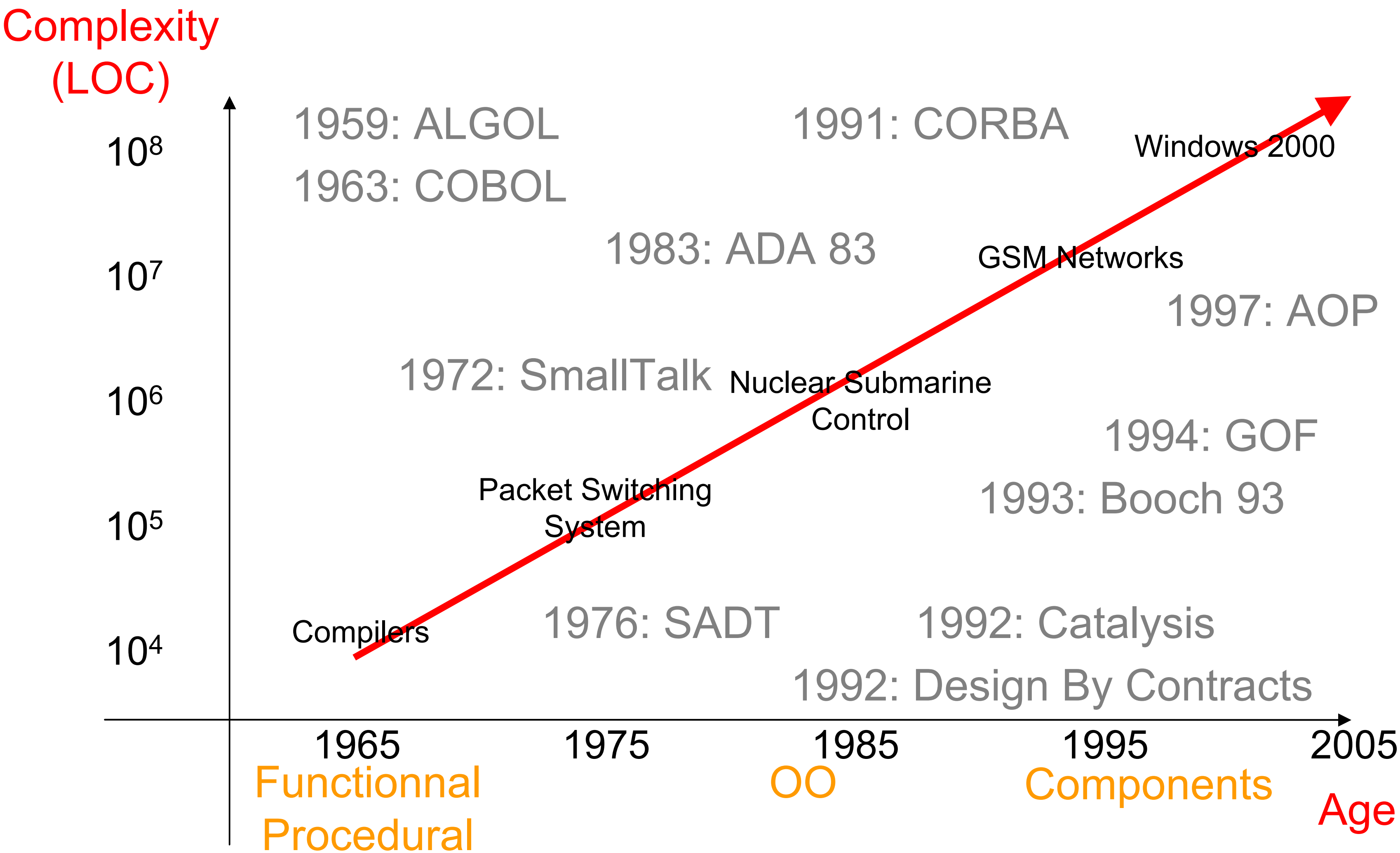
Productivity Gains in SE

- ***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
  - …

Consequence: Production Gains



Remark: A new paradigm needs around 10 years to be mature

Major Lessons

- SE Best Practices
  - Reuse / Develop for reuse
  - Refinement / Refactoring
  - Prototyping
  - Test / Verification / Validation
  - Communication / Documentation

- Problems
  - Requirements ***change***
    - Nokia: 50% changed after finalization ; 60% of them at least twice
  - Platforms ***change***
    - OS, Languages, Databases, Middleware, …
  - Development technology ***change***
    - Compilers, Code generators, Frameworks, …
  - People ***change***
    - Especially for systems developed 30 years ago !

Models

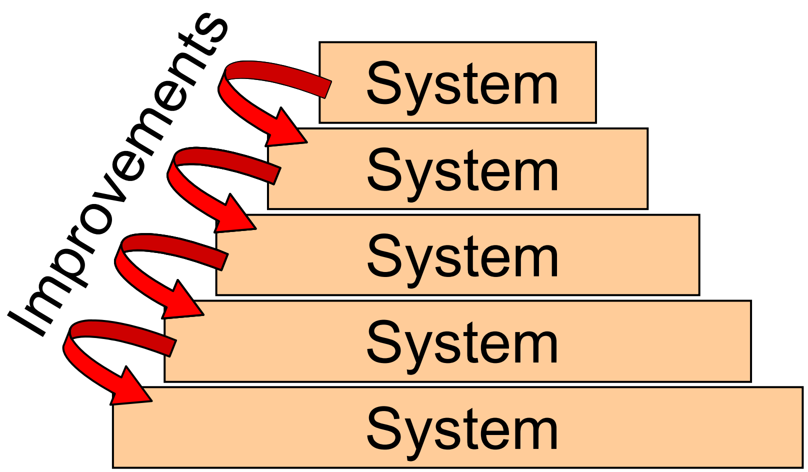
- Abstract away details
  - Use paradigms rather than technologies
  - Enhance productivity and production !
- Human friendly
  - Different views of the same system
- Cheap to manipulate and maintain
- Models at any abstraction level
  - Use cases, Business models, SDL, CCM, B, …

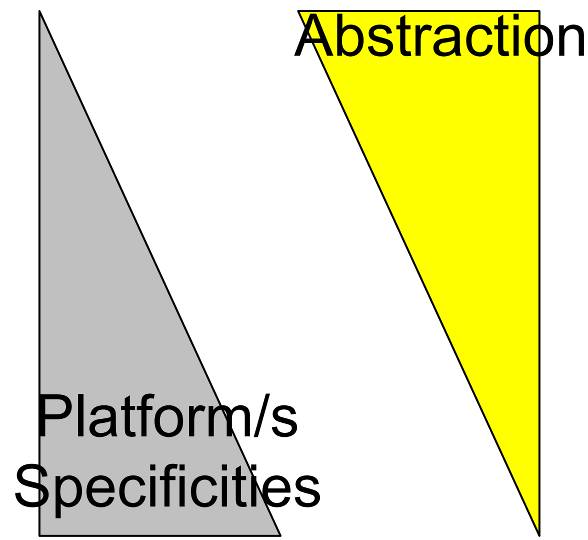
Used most of the time for

- Documentation / communication
- Analysis
- Prototyping

Often drawings out of sync with (code) reality !

Software Engineering





- Test
- Validation
- Verification

- Iteration ≠
- Level of Abstraction ≠
- Level of Detail

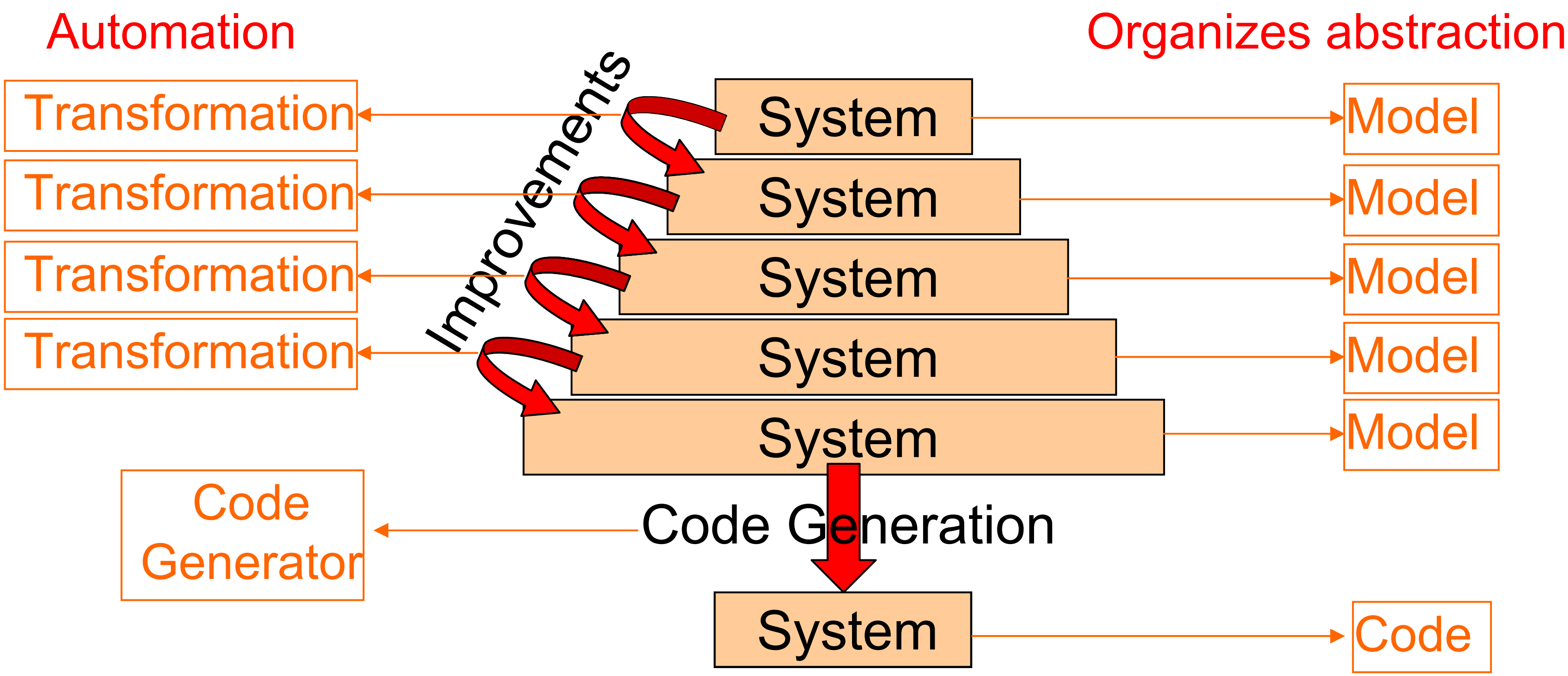
Contents

- Introduction
  - Software Engineering
  - Model Driven Engineering
  - Language Definition
- Concrete Syntaxes
  - Textual concrete syntax definition
  - Graphical concrete syntax definition
- Conclusions

SE with Models

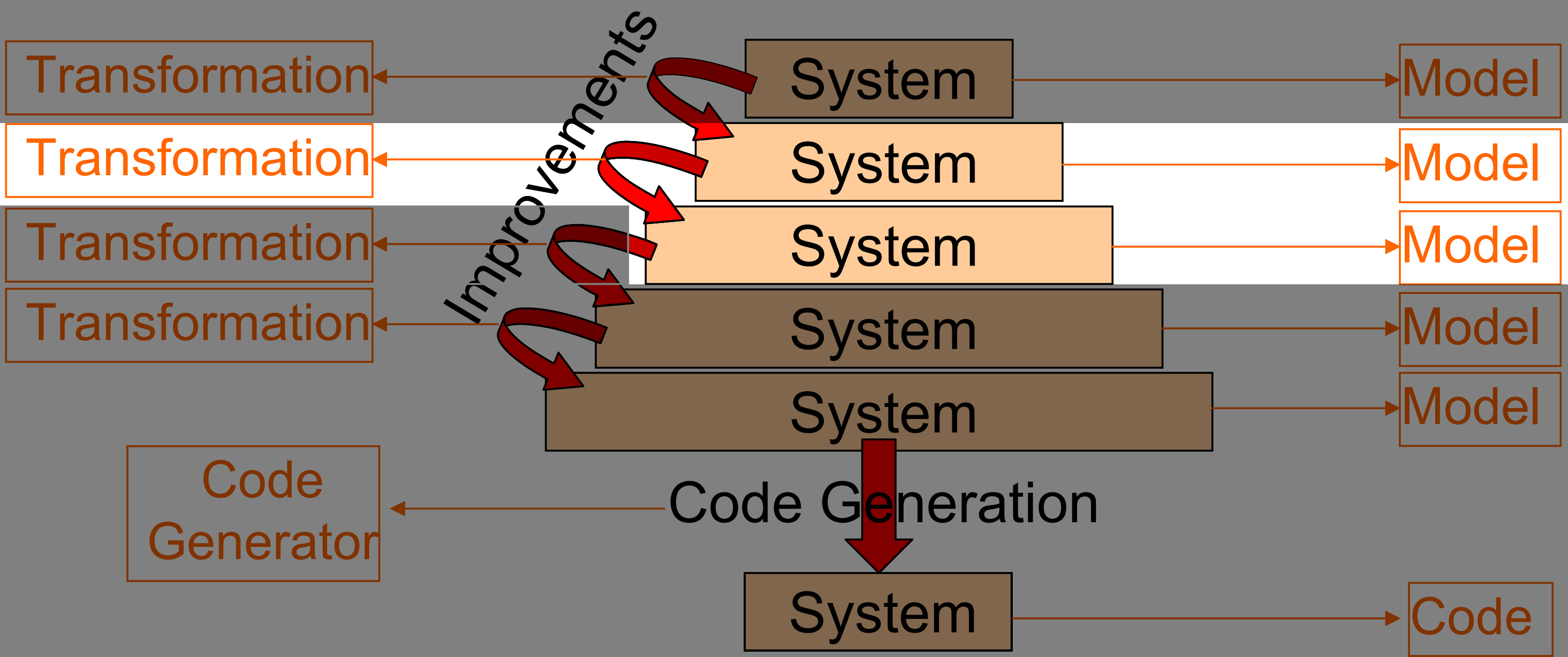
Model Driven Engineering (MDE)

"From contemplative to productive"



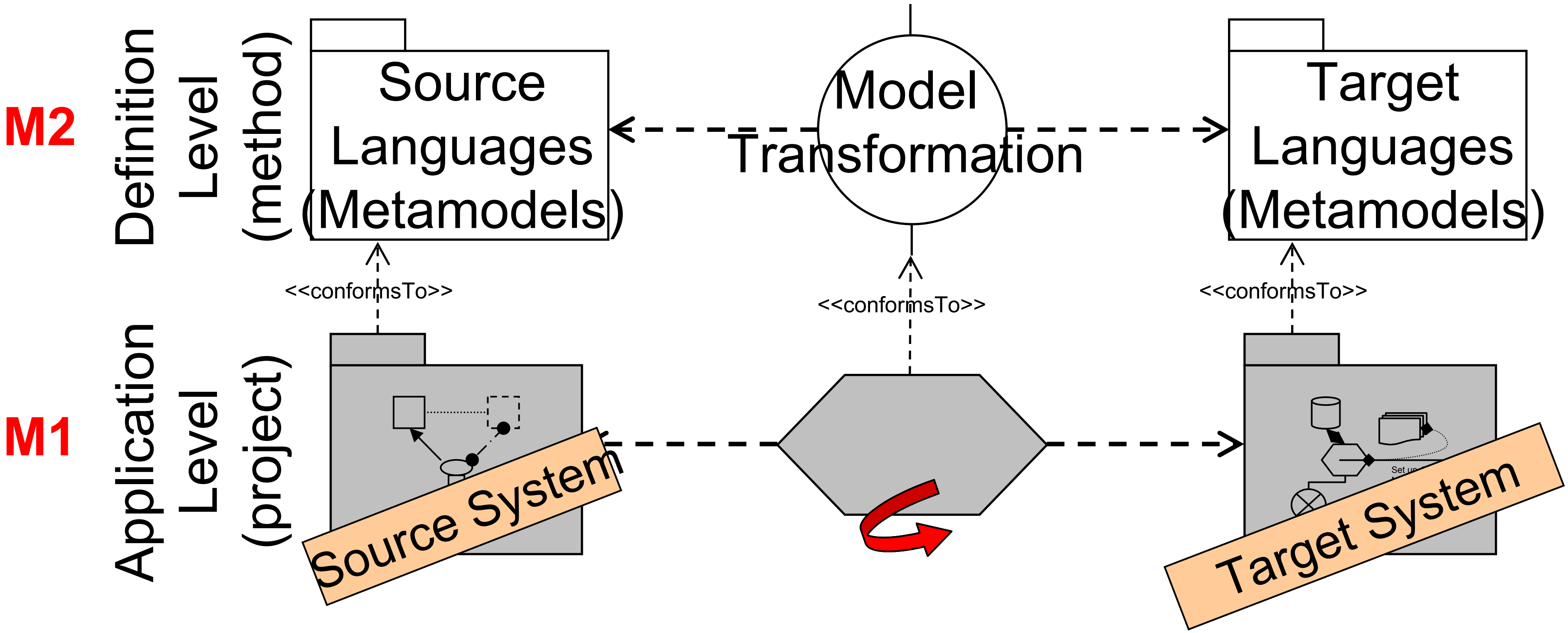
- Test → **GENERATION**
- Validation → **PROTOTYPING**
- Verification → **MODEL CHECKING**

Model Driven Engineering

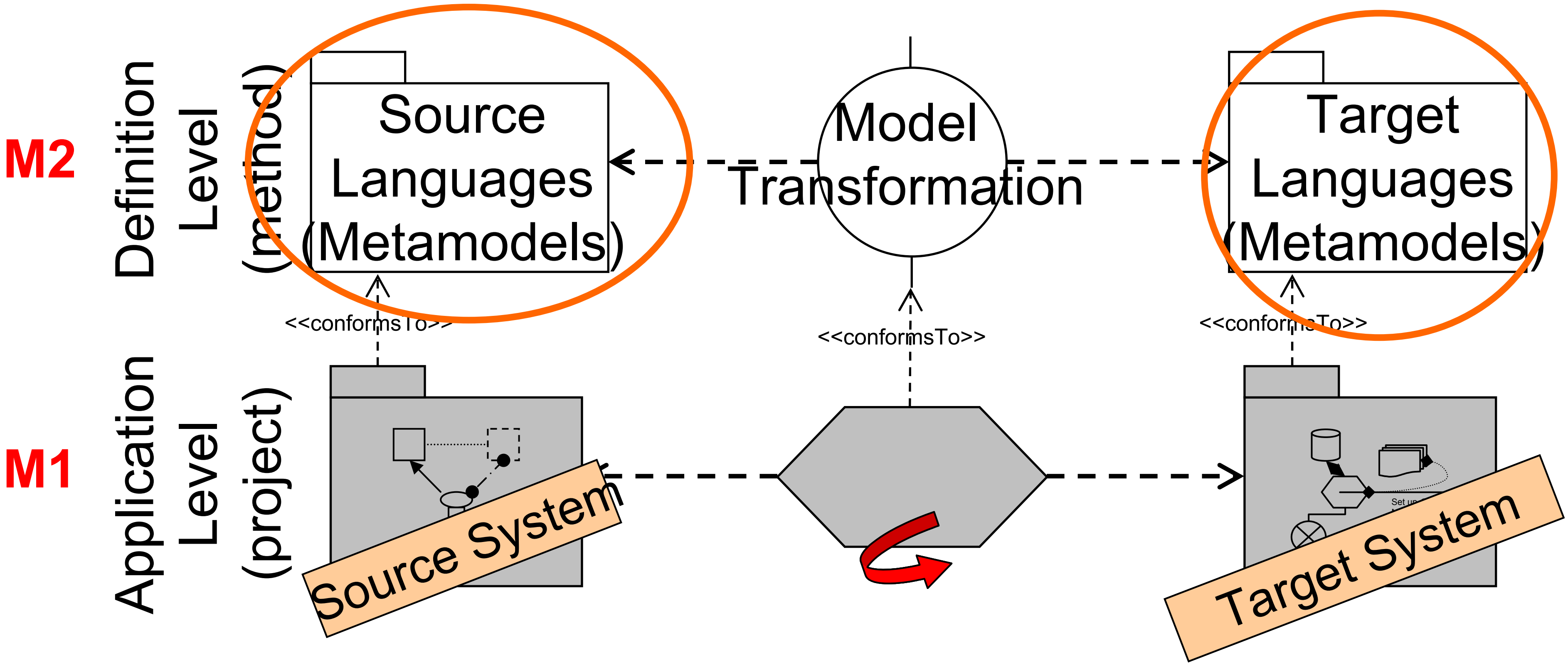


- Test → **GENERATION**
- Validation → **PROTOTYPING**
- Verification → **MODEL CHECKING**

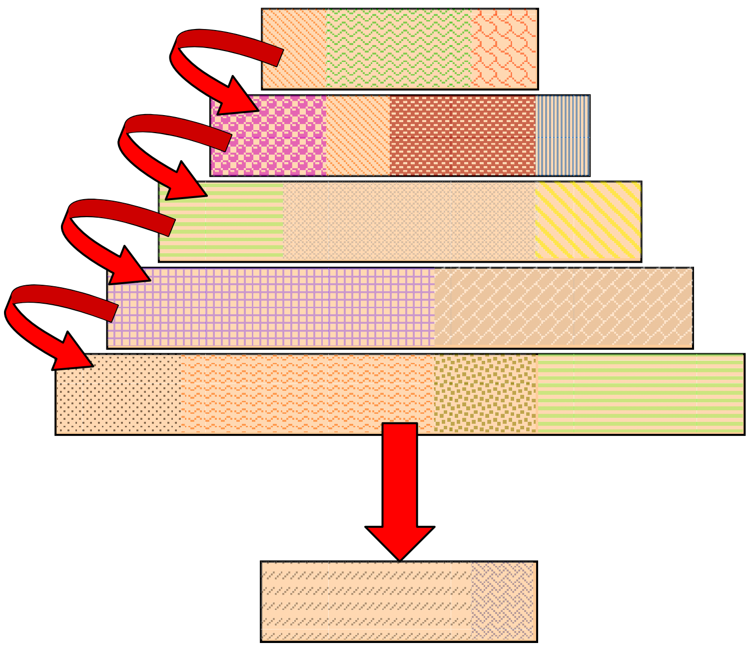
Model Driven Engineering



Model Driven Engineering



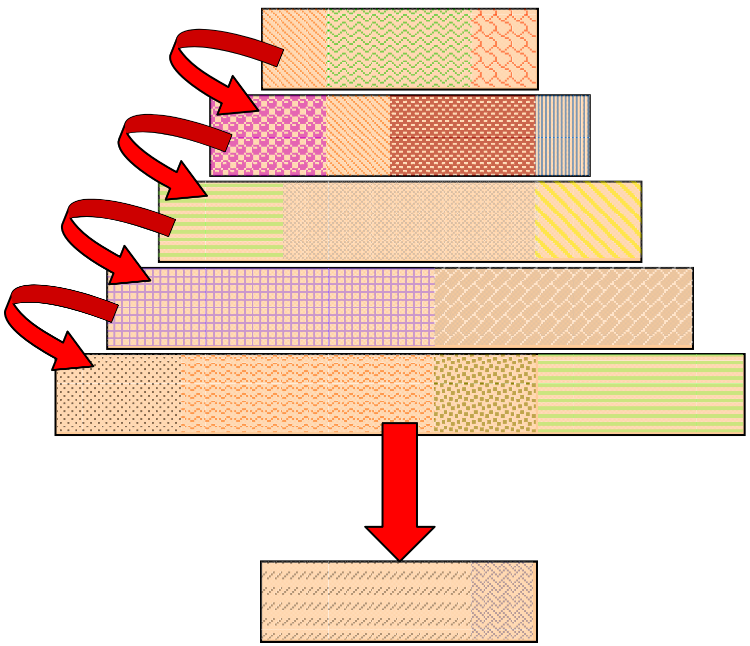
Questions Raised



- What language for describing systems ?
  - E.g. E-Banking
    - From Use Cases…
    - …to Oracle Schema + EJB + JSP
  - E.g. Subway
    - From Use Cases…
    - … to Wireless JavaCard
  - …
- One unique language is not enough !
  - Domain Specific Languages
    - Small languages
  - Define YOURSELF the right level of abstraction !

Domain Specific Modeling

Questions Raised



Domain Specific Modeling

=

Proliferation of languages !

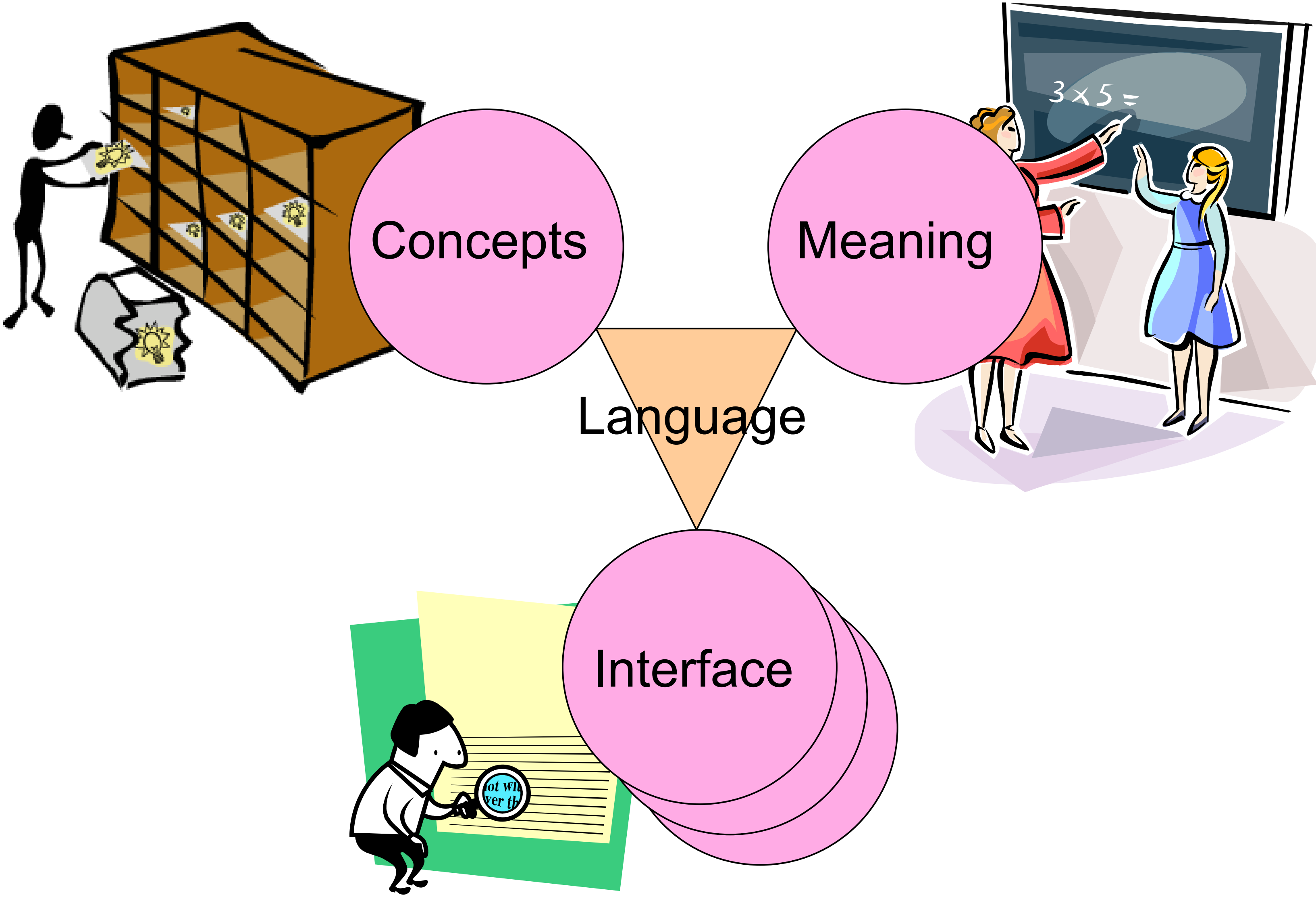
- **Support for language engineering**

Contents

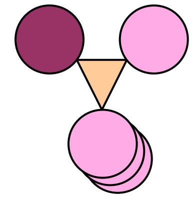
- Introduction
  - Software Engineering
  - Model Driven Engineering
  - Language Definition
- Concrete Syntaxes
  - Textual concrete syntax definition
  - Graphical concrete syntax definition
- Conclusions

Language Definition

M2

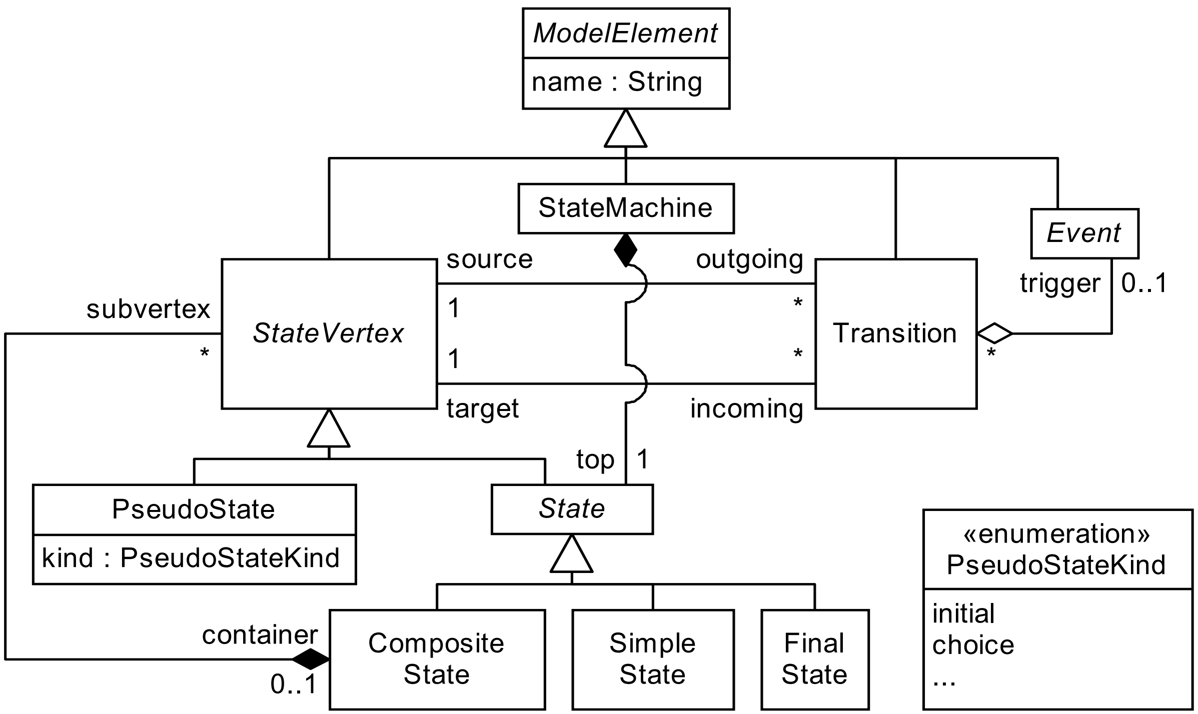


Concepts Definition

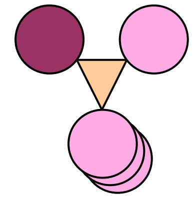


M2

Abstract Syntax

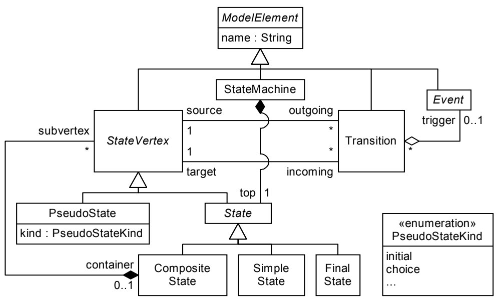


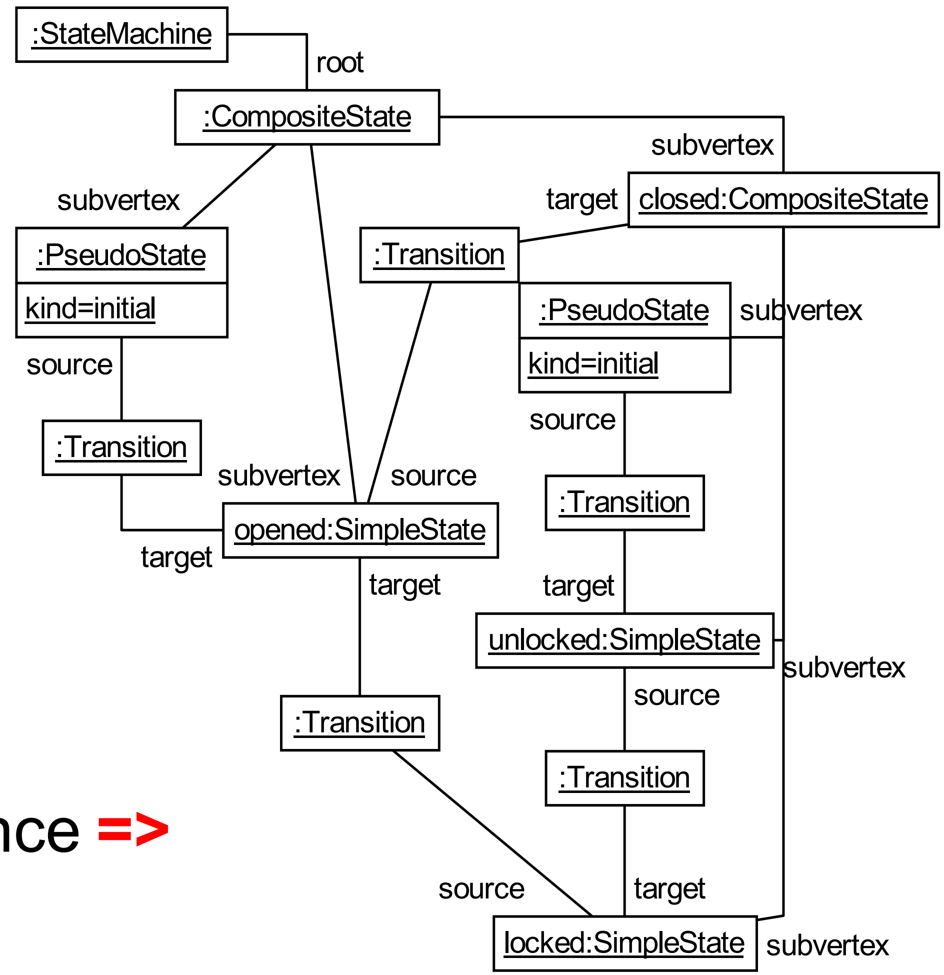
Concepts Definition



M1

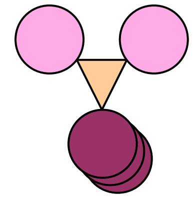
Abstract Syntax





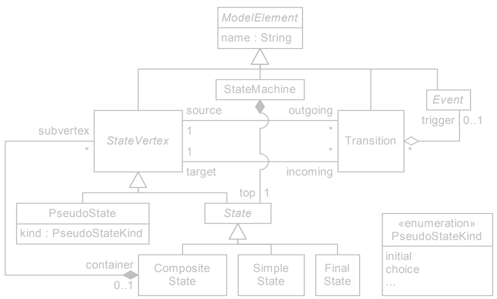
An (M1) sentence =>

Interface Definition



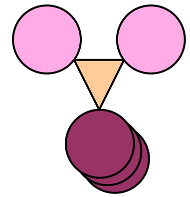
M2

Abstract Syntax + Concrete Syntax(es)



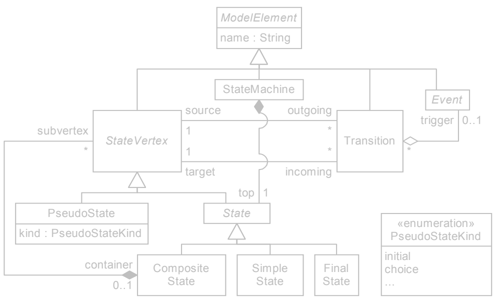

```
sm ::= "Statemachine" IDENT compositeState
state ::= normalState | pseudostate
normalState ::= "initial"? (simpleState | compositeState)
simpleState ::= "State" IDENT
compositeState ::= "CompositeState IDENT? LCURLYBRACKET
                   (state | transition)* RCURLYBRACKET
transition ::= "Transition"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

Interface Definition



M1

Abstract Syntax + Concrete Syntax(es)



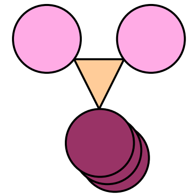

```
sm ::= "Statemachine" IDENT compositeState

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

...

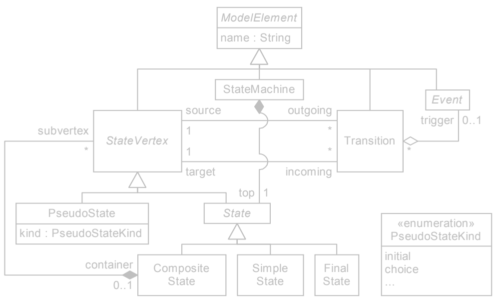
<= An (M1) sentence

Interface Definition



M2

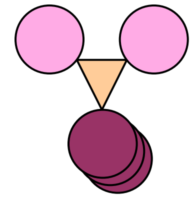
Abstract Syntax + Concrete Syntax(es)



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

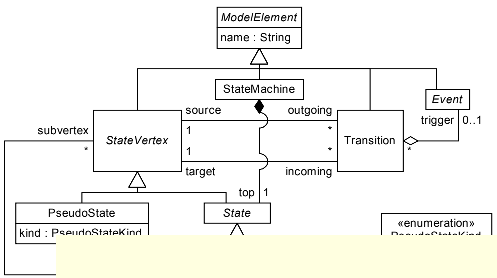
- In practice
  - Layout constraints
  - User interactions

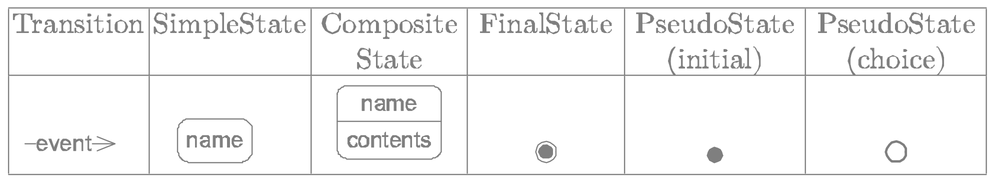
Interface Definition

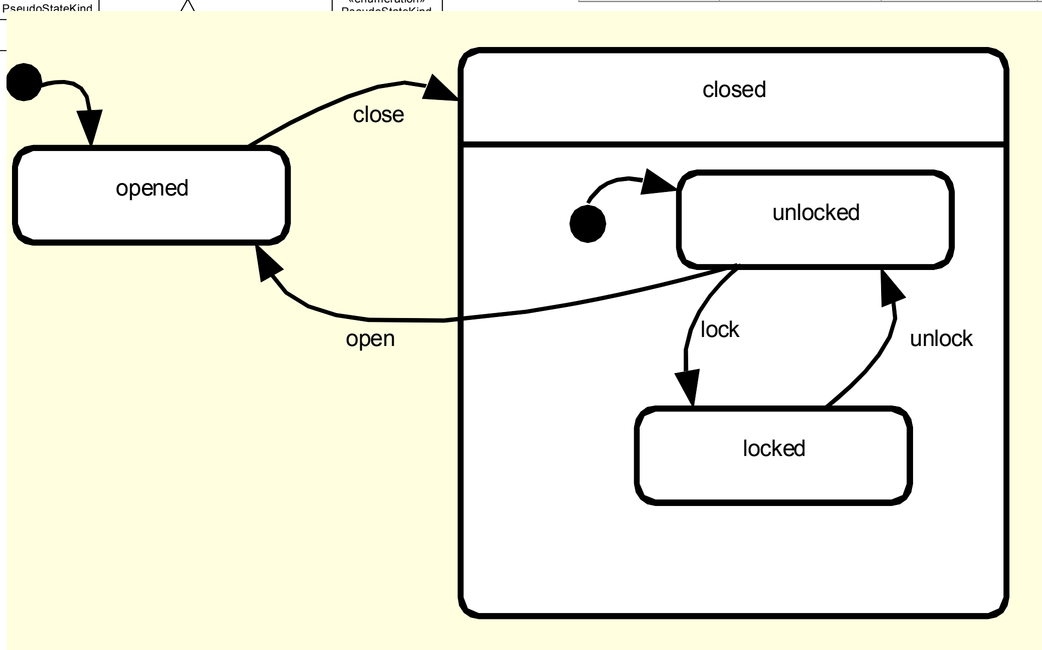


M1

Abstract Syntax + Concrete Syntax(es)

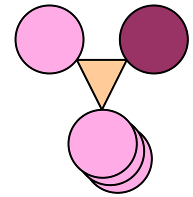






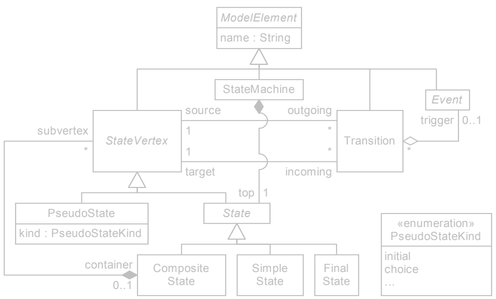
<= An (M1) sentence

Meaning Definition



M2

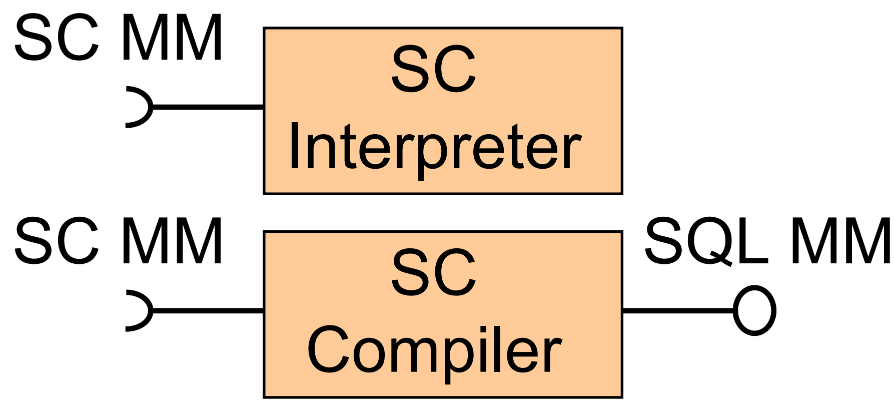
Abstract Syntax + Concrete Syntax(es) + Semantics



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
```

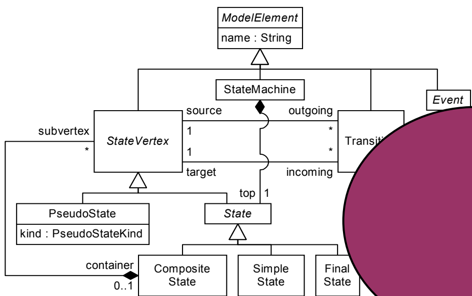


- Research issue

Language Definition

M2

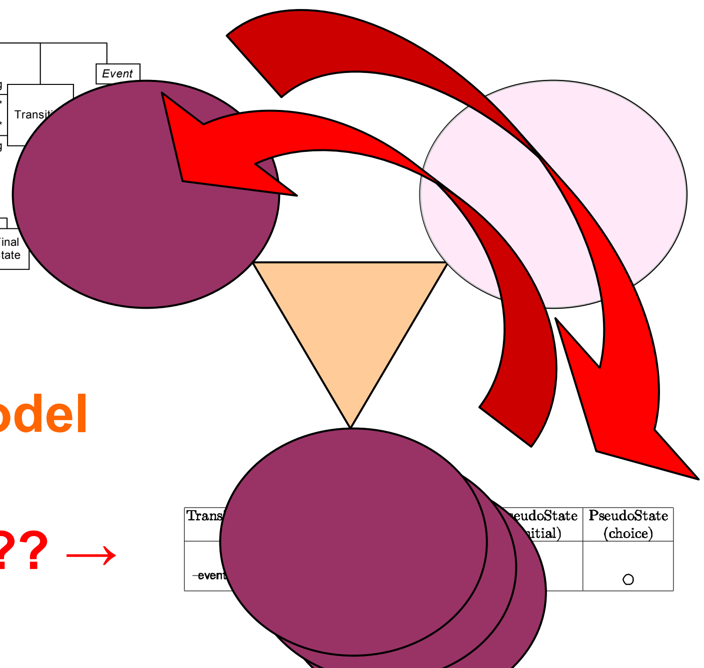
Abstract Syntax + Concrete Syntax(es) ntics



↑

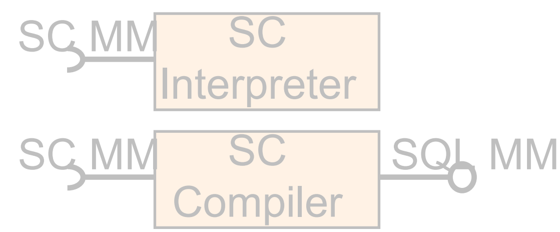
Metamodel

??? →

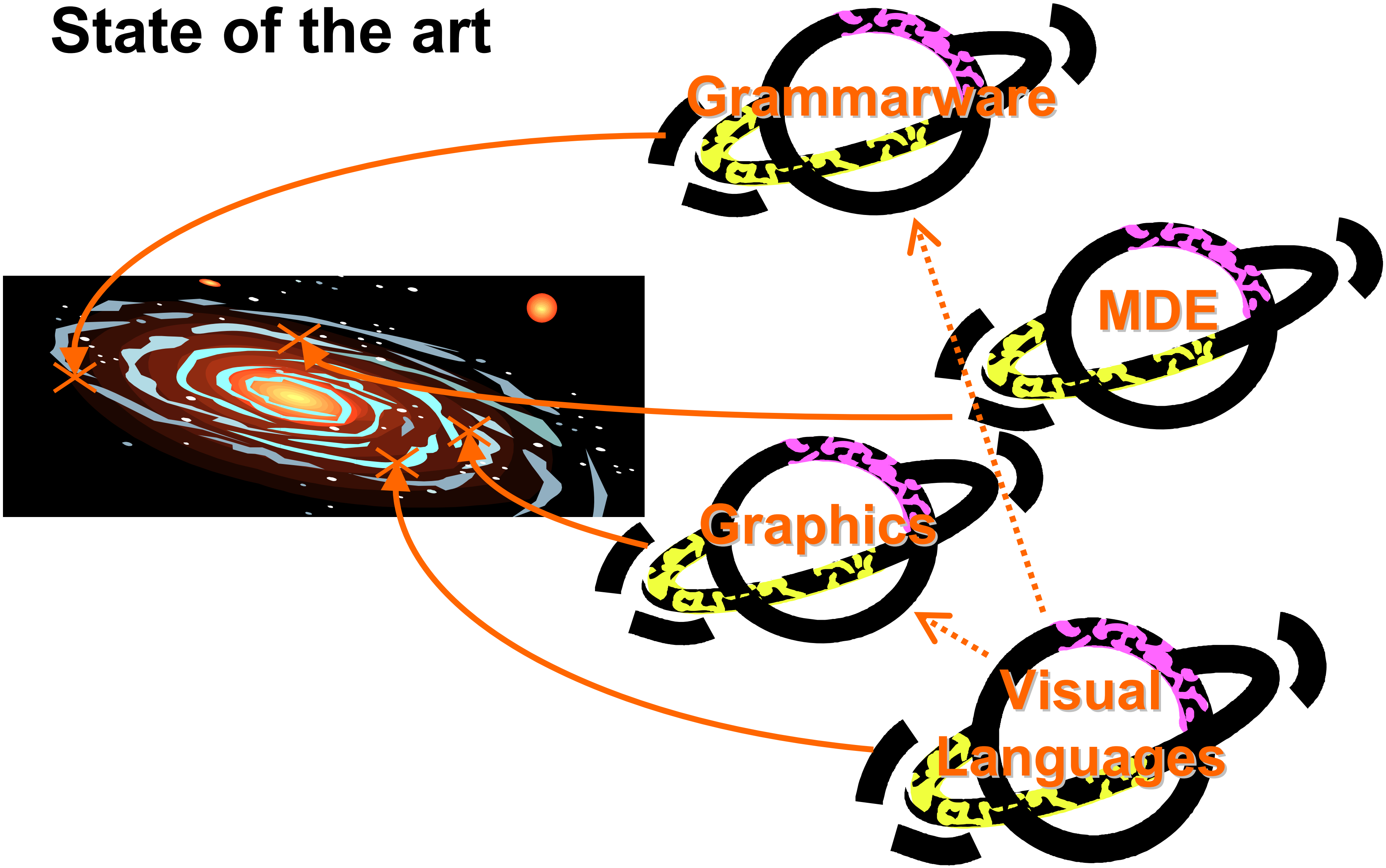


??? →

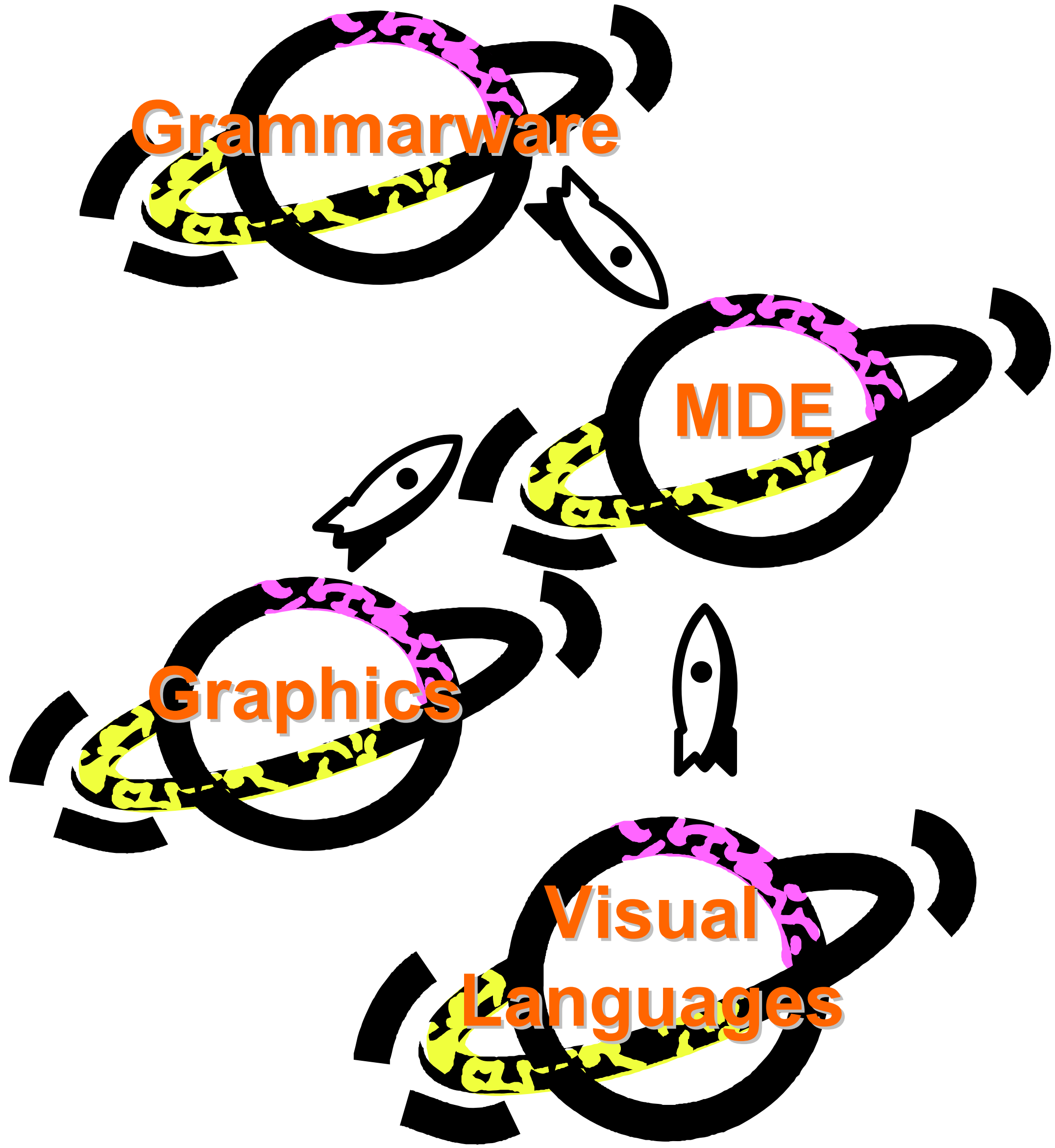
```
sm ::= "Statemachi                              state
state ::= normalState | pseudostate
normalState ::= "initial"? (simpleState | compositeState)
simpleState ::= "State" IDENT
compositeState ::= "CompositeState«  IDENT? LCURLYBRACKET
(state | transition)* RCURLYBRACKET
transition ::= "Transition"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



State of the art



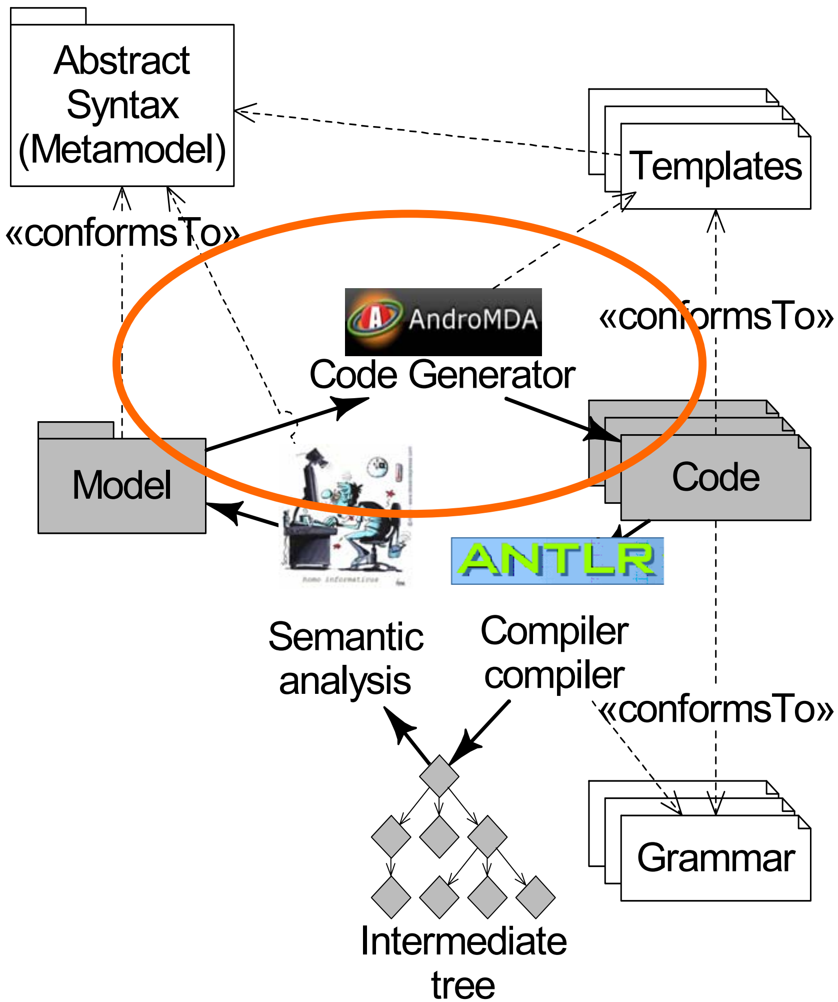
Strategy



Contents

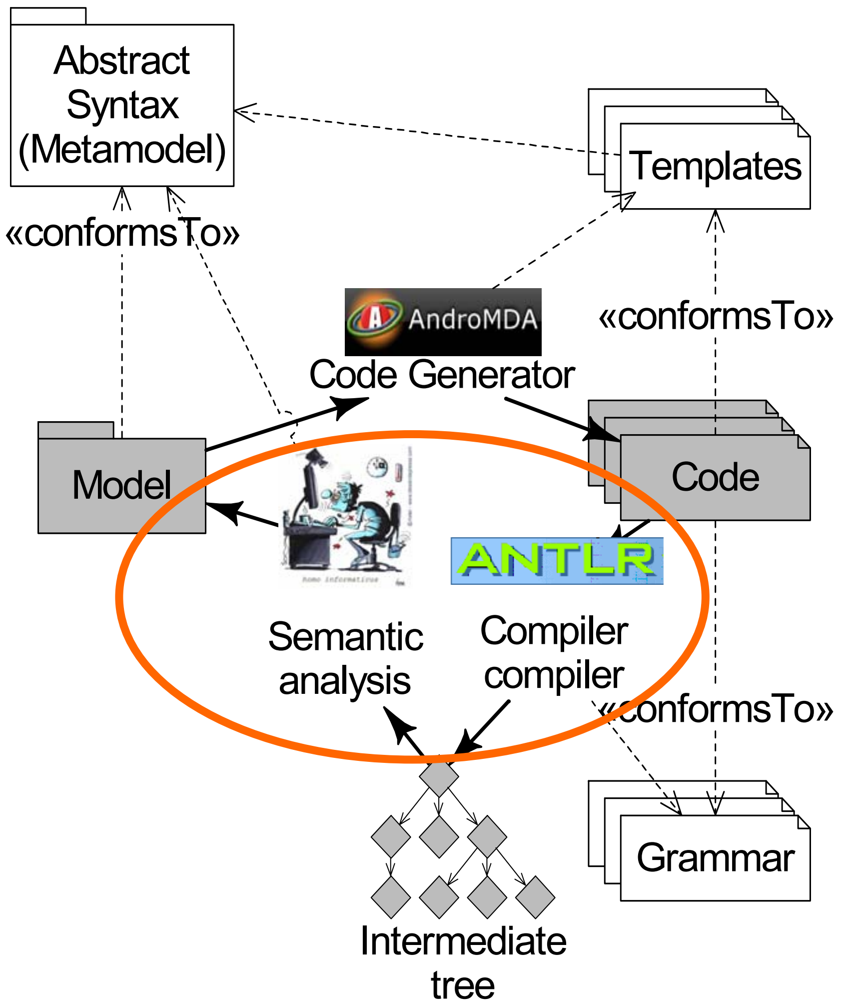
Typical Implementation

White: M2
Grey: M1



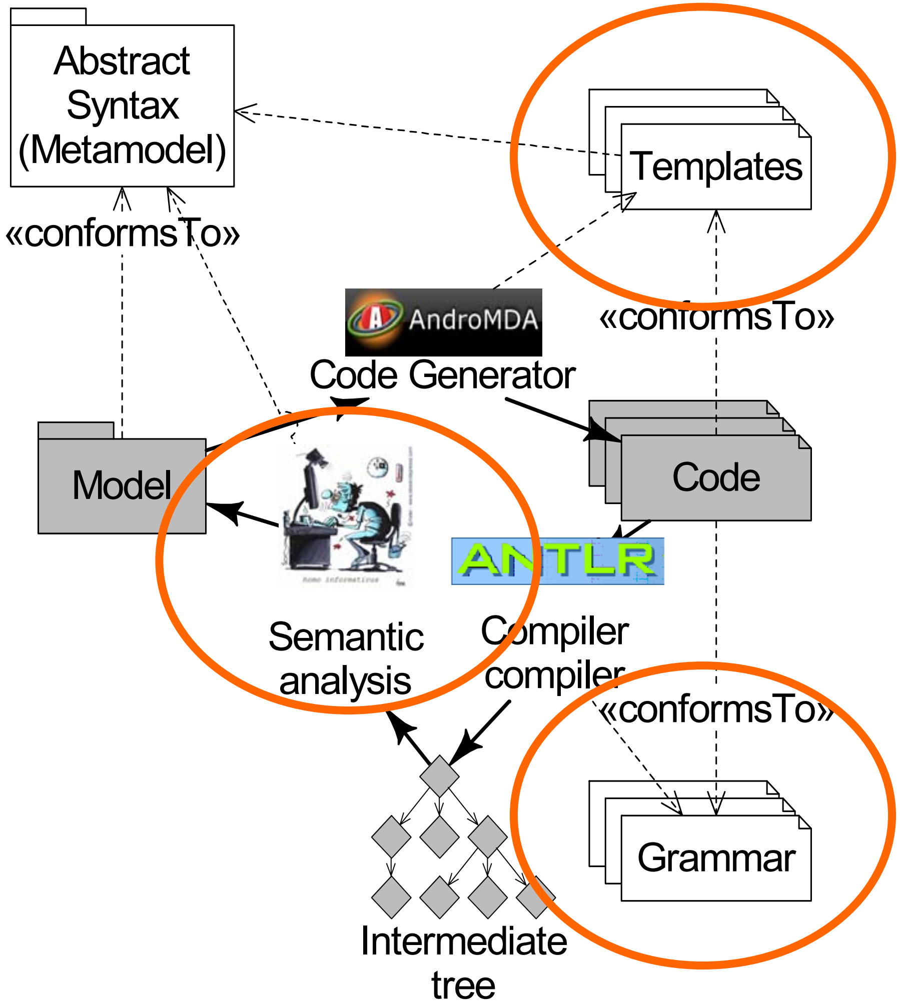
Typical Implementation

White: M2
Grey: M1



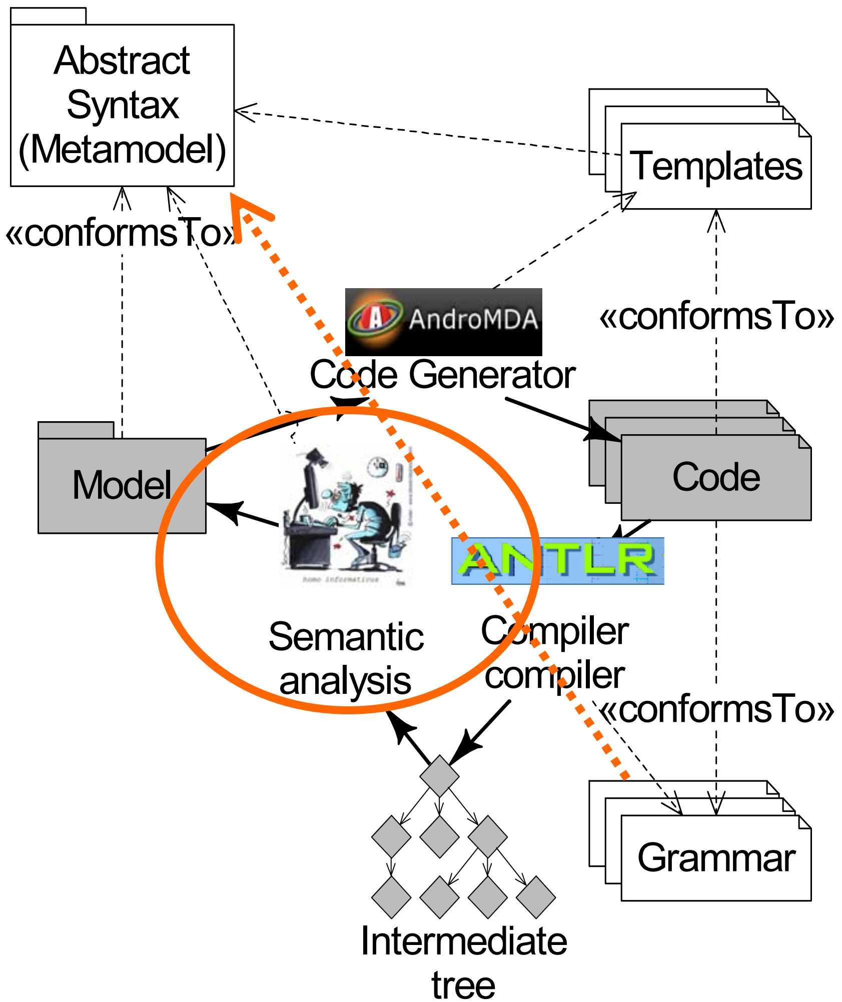
Typical Implementation

White: M2
Grey: M1



Typical Implementation

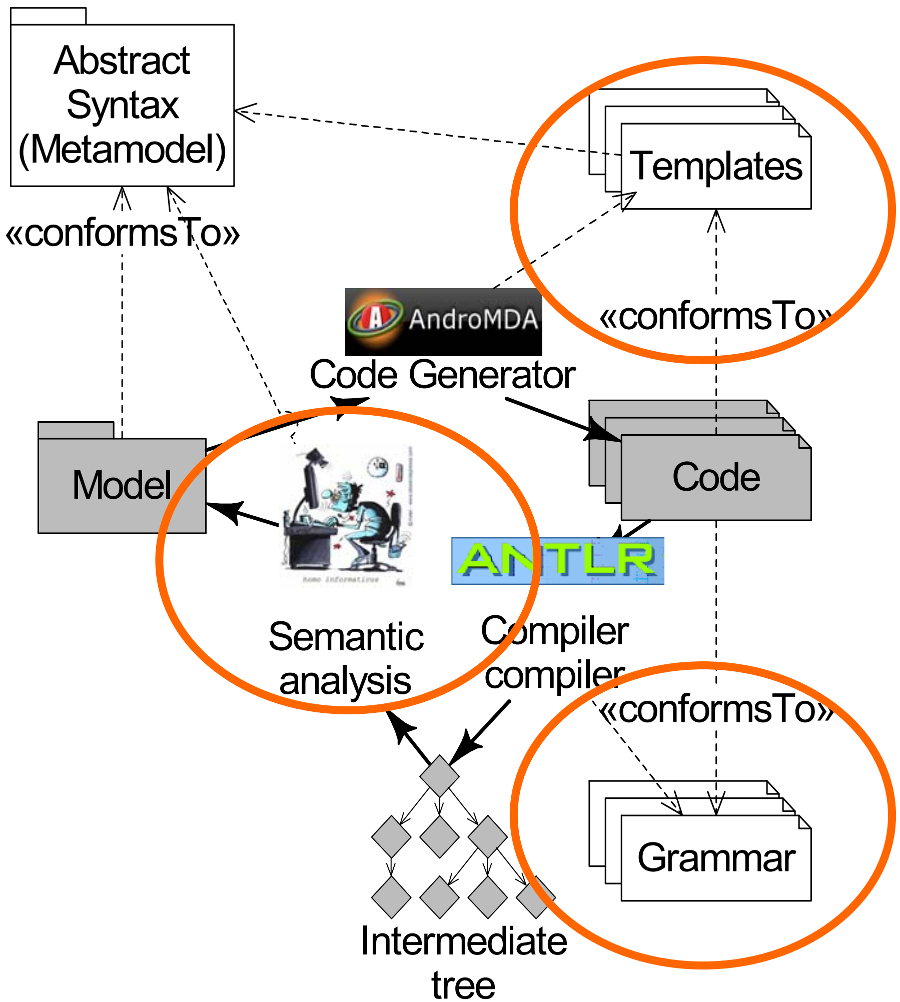
White: M2
Grey: M1



- Concrete syntax tree and model
  - Not connected !
  - Different nature !
  - →Hard to relate….
  - →Hand-coded…

Typical Implementation

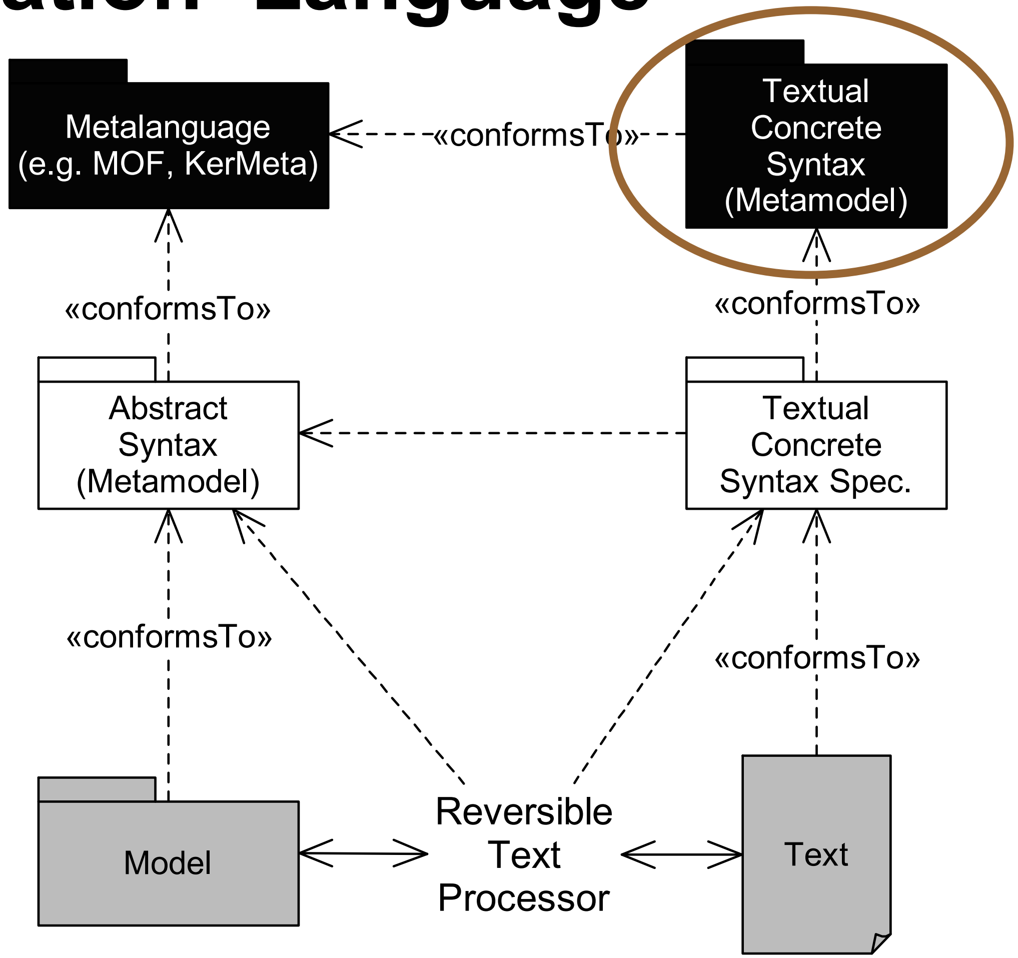
White: M2
Grey: M1



- Code generation templates, Grammar, and Semantic analysis
  - Double conformance !
  - →Double specification…
  - →Triple maintenance points…
  - →No automated coherence proof…

Specification Language

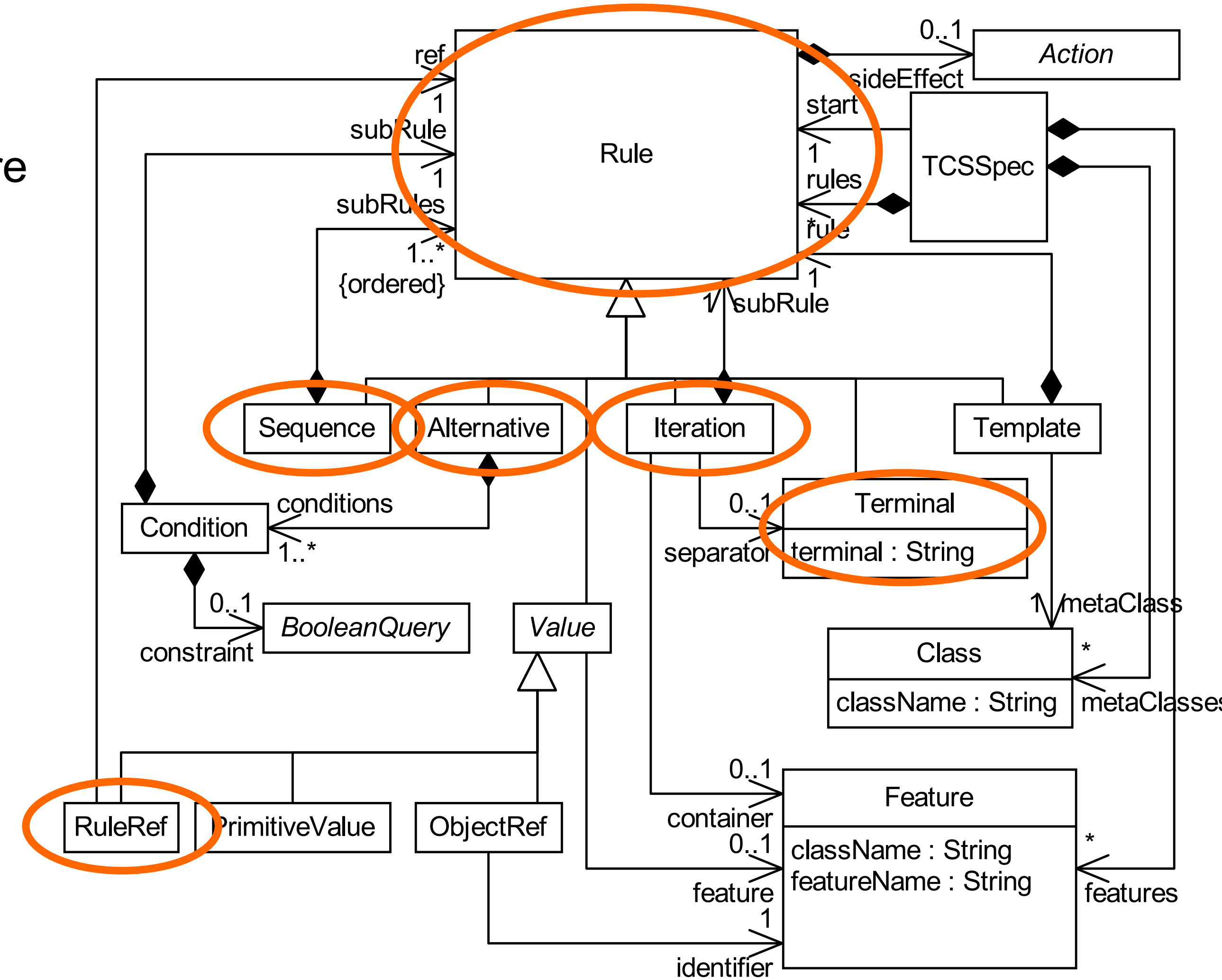
Black: M3
White: M2
Grey: M1



- No concrete syntax
  - Help yourself !

TCSSpec Metamodel

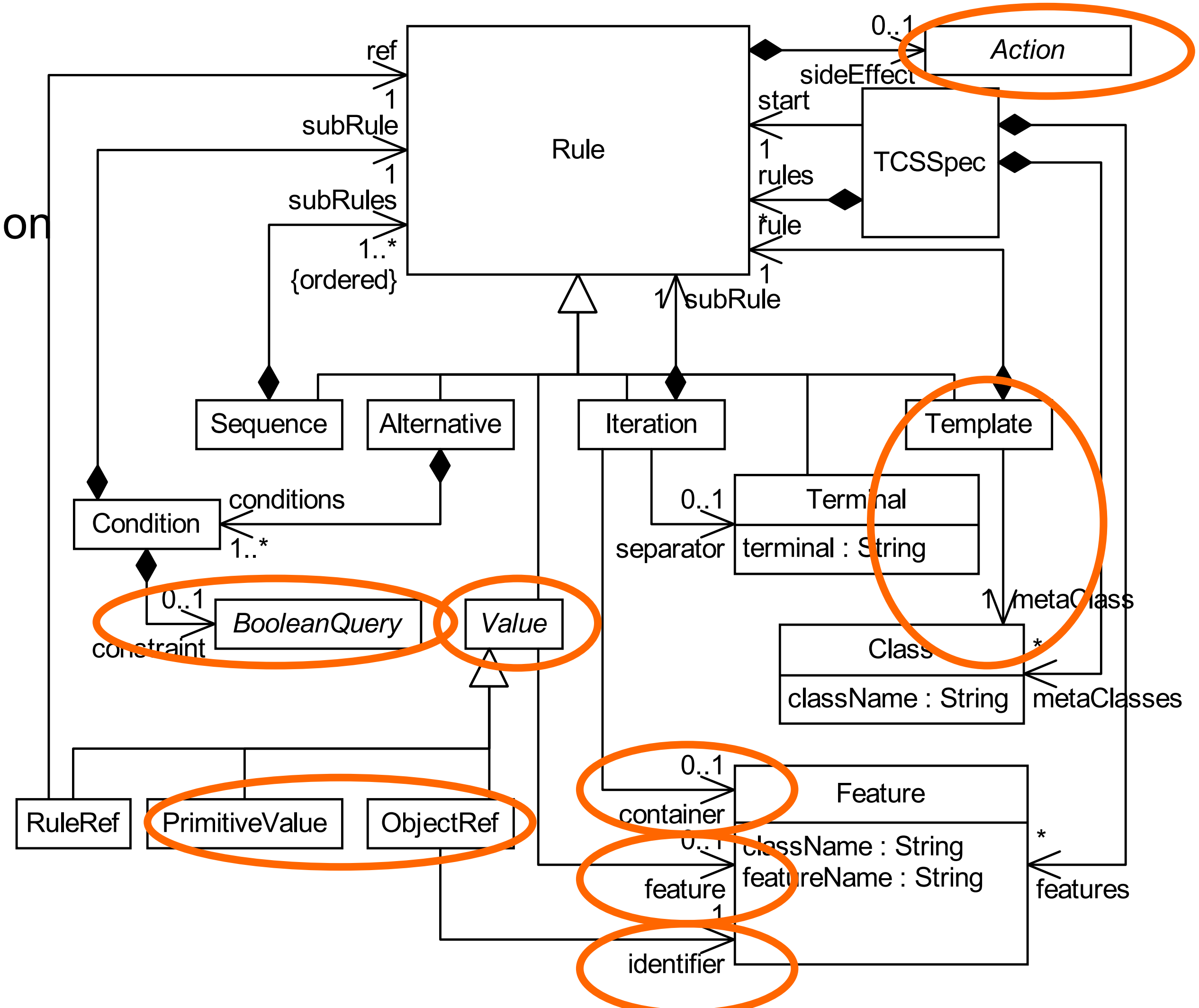
Inspired from
- EBNF
  - Text structure
- Netsilon



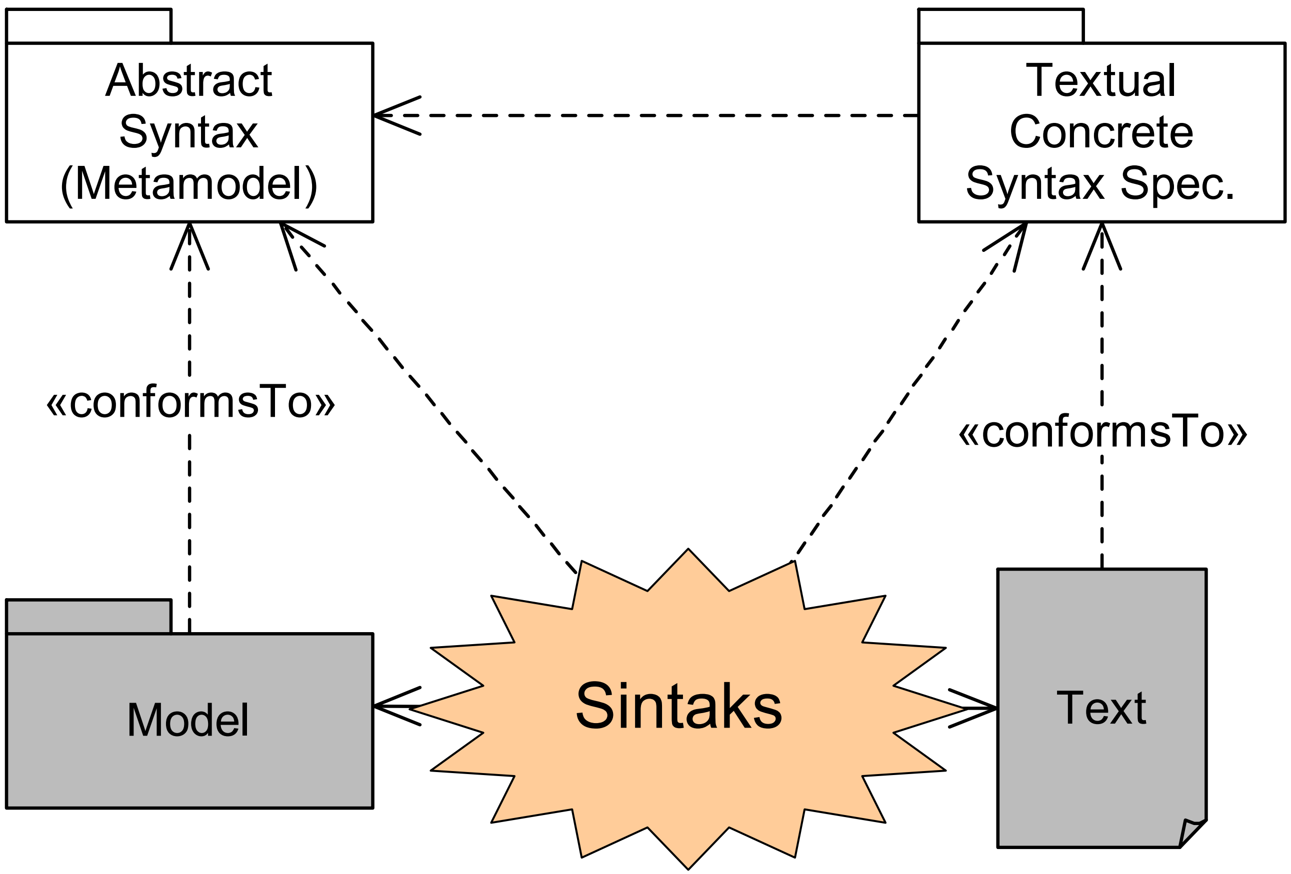
TCSSpec Metamodel

Inspired from

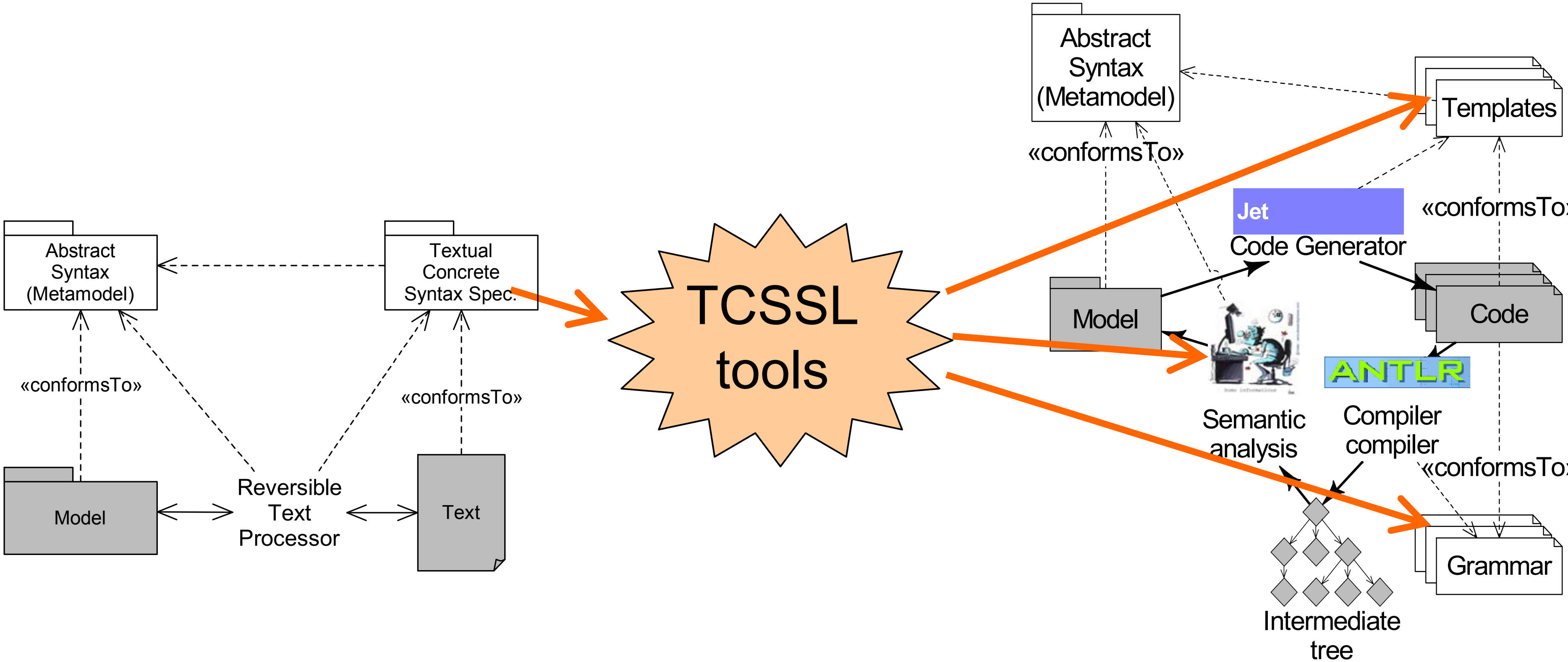
- EBNF
- Netsilon
  - Model Navigation



Prototypes: Sintaks



Prototypes: TCSSL Tools



Conceptual differences

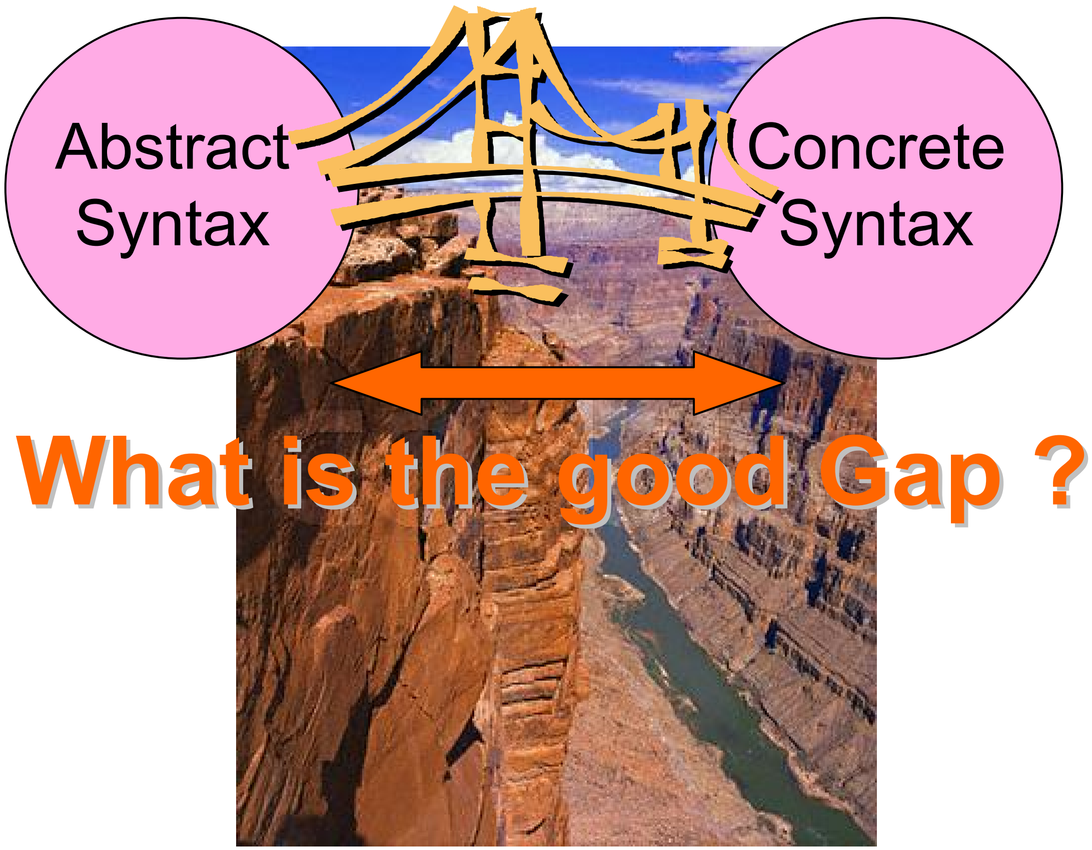
- TCSSL Tools (CEA)
  - Instantiation rules
    - Create / search / search and create if not exists
  - Auto-call of sub-rules according to inheritance hierarchy
    - Avoids the starteR rule
    - A problem for rule precedence
  - Tests limited to comparisons
    - Tests interpreted as actions at analysis

Flexibility

- Sintaks (UHA)
  - Test limited to specific queries
    - Attribute values
    - Object types
  - No actions
  - Late binding of references

Safety

Question

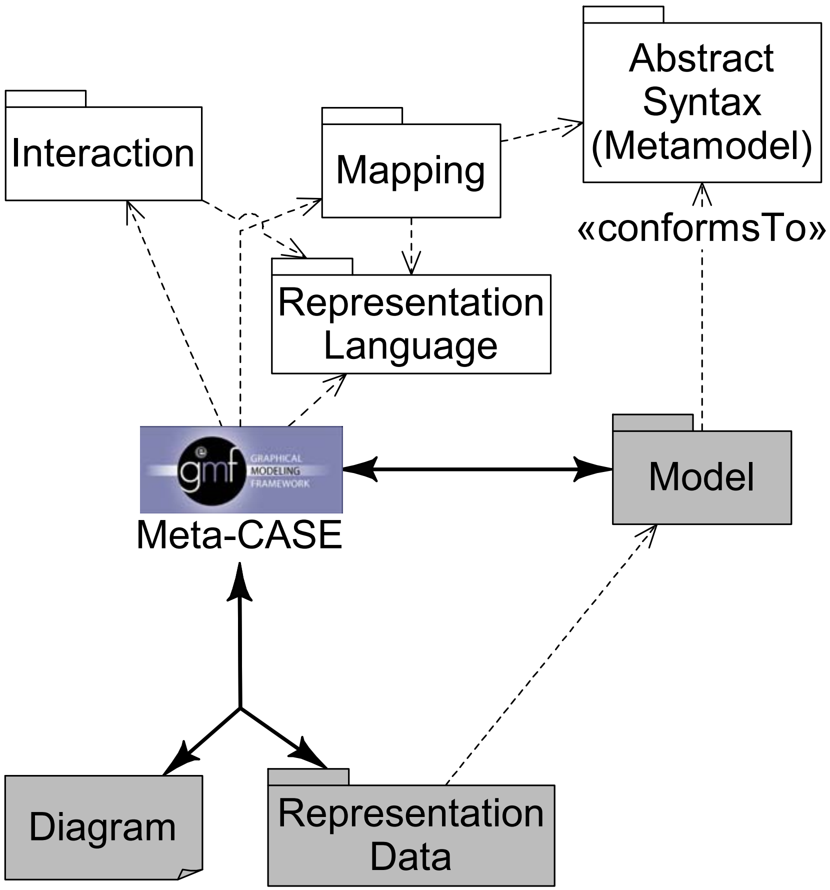


© Royalty-Free/Corbis

Contents

Typical Implementation

White: M2
Grey: M1



- Not adopted by industry yet
  - MVC with 2D graphical libraries

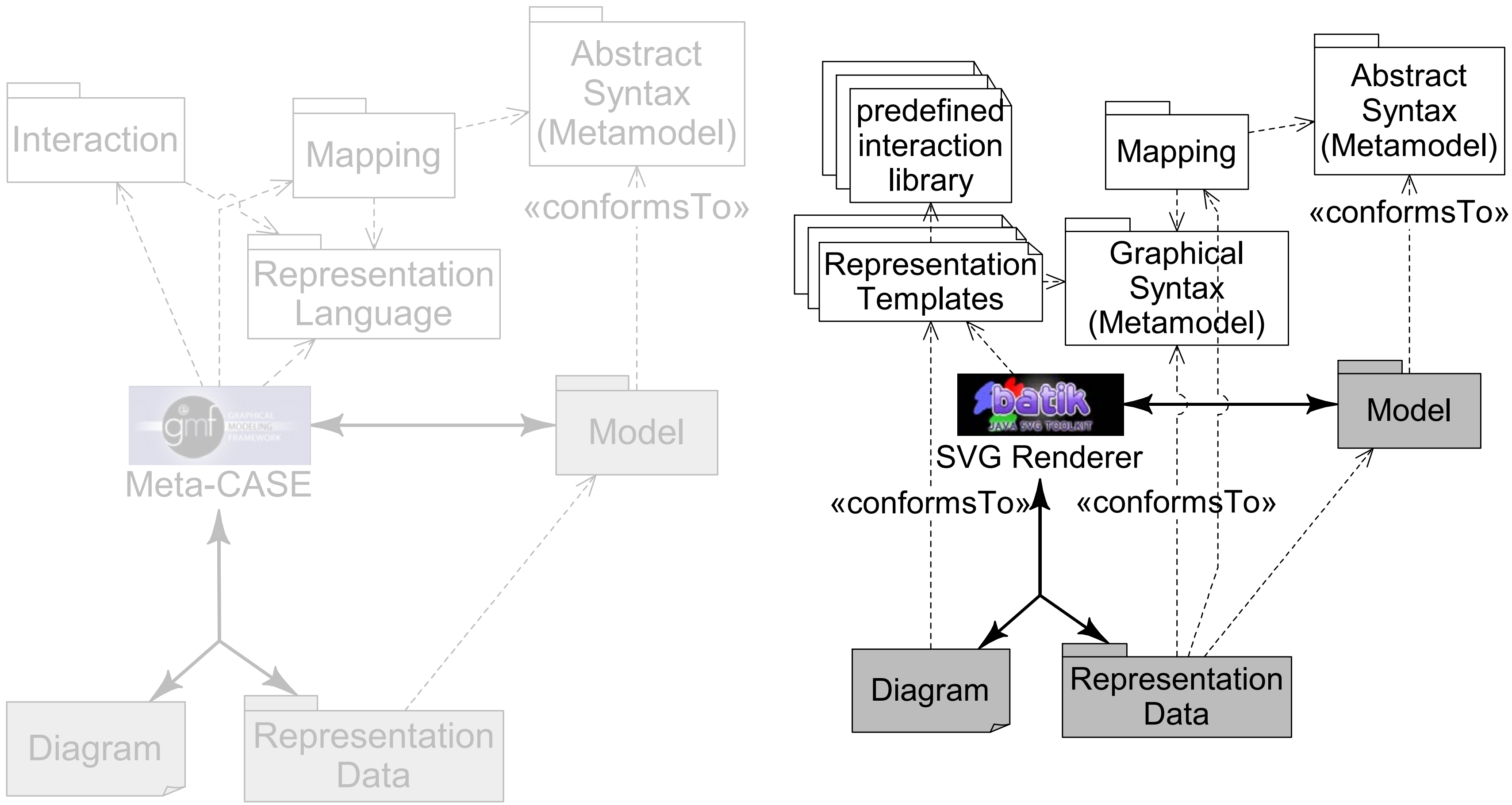
Problems

White: M2
Grey: M1

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

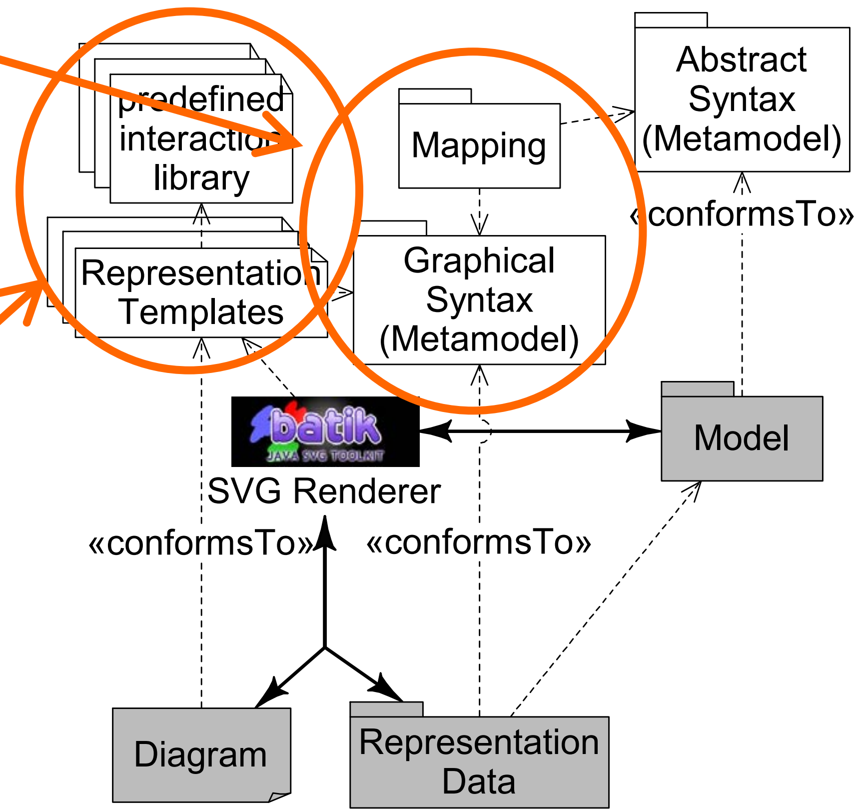
Idea

White: M2
Grey: M1



Graphical concrete syntax definition

- Concrete syntax model and mapping
  - 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



The Representation Language

- Render Vector Graphics
  - Render “Terminal” Symbols
  - As open as possible
- Controllable by an API (online rendering)
  - Implementation for interaction library
  - Possibility to specify variation points
    - Mean to access the model
- Possibility to specify layout constraints

The Representation Language

- ✓ Render Vector Graphics
  - ✓ Not connection-based only
  - ✓ As open as possible
- Controllable by an API (online rendering)
  - Implementation for interaction library
  - Possibility to specify variation points
    - Mean to access the model
- Possibility to specify layout constraints

SVG

The Representation Language

- ✓ Render Vector Graphics
  - ✓ Not connection-based only
  - ✓ As open as possible
- ✓ Controllable by an API (online rendering)
  - ✓ Implementation for interaction library
  - ✓ Possibility to specify variation points
    - ✓ Mean to access the model
- Possibility to specify layout constraints

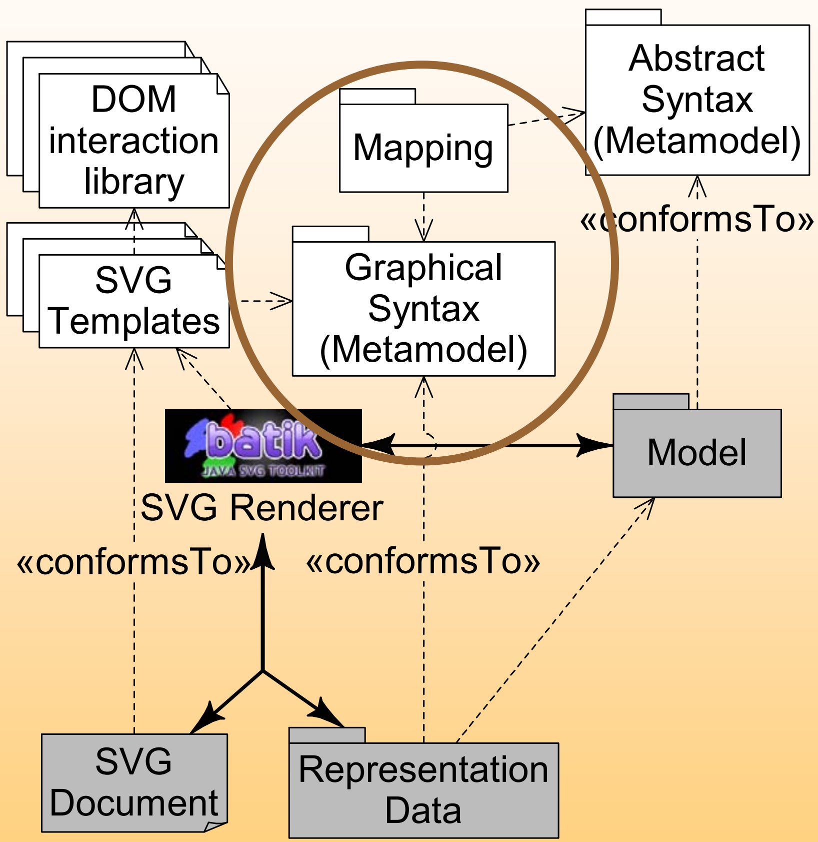
SVG + DOM

The Representation Language

- ✓ Render Vector Graphics
  - ✓ Not connection-based only
  - ✓ As open as possible
- ✓ Controllable by an API (online rendering)
  - ✓ Implementation for interaction library
  - ✓ Possibility to specify variation points
    - ✓ Mean to access the model
- ✓ Possibility to specify layout constraints

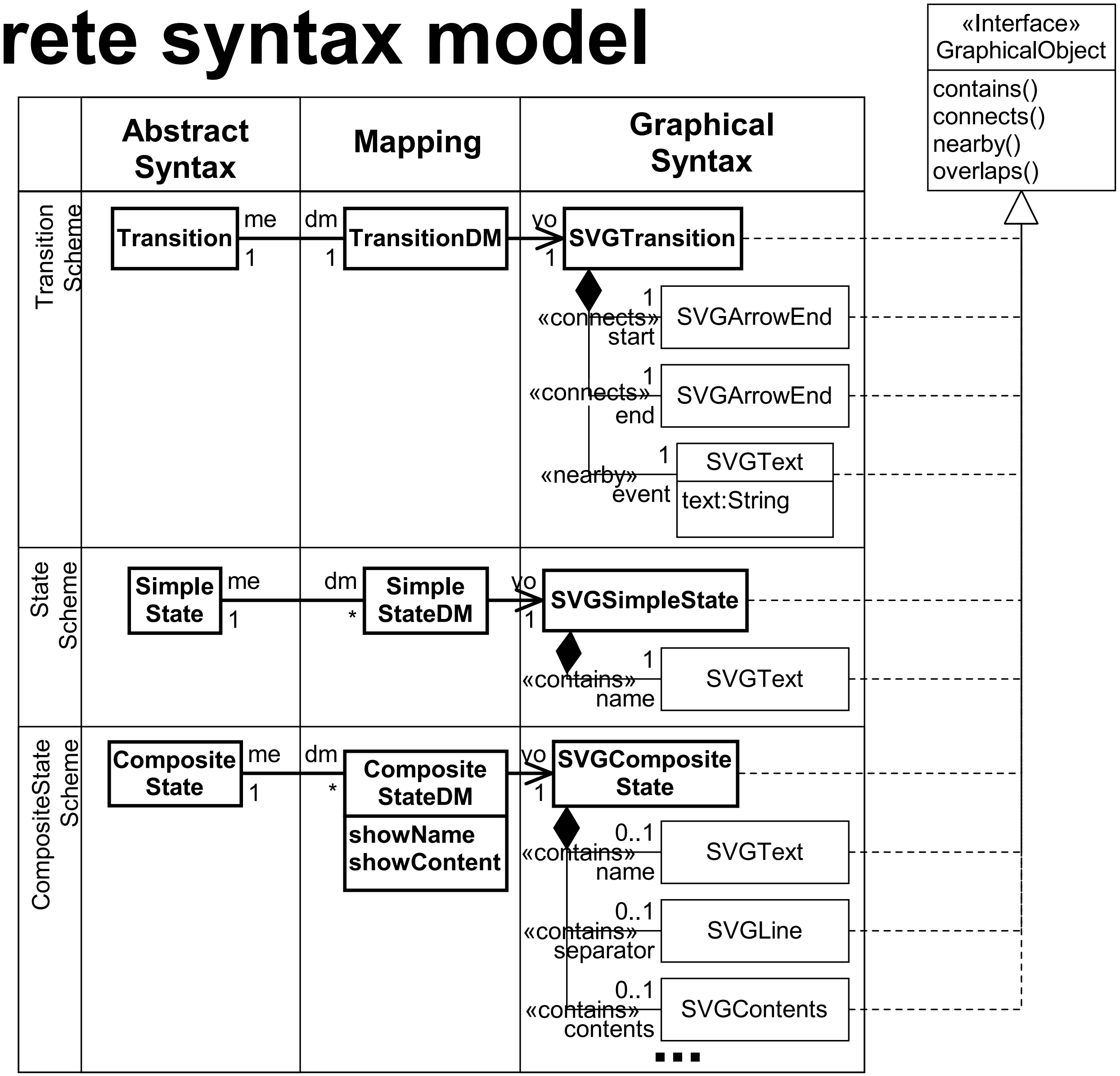
SVG + DOM + C-SVG

Graphical concrete syntax definition



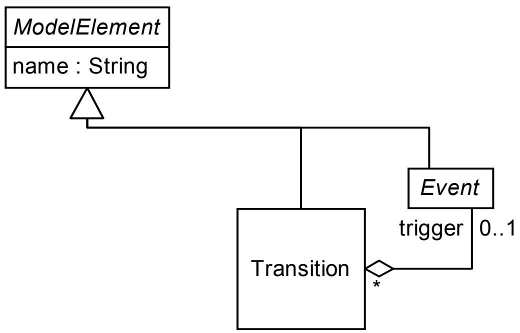
- Concrete syntax model
  - Fixes concrete syntax elements
  - Fixes relationship with abstract syntax
- Concrete syntax graphical design (with a demo)
  - Fixes appearance
  - Fixes layout constraints
  - Fixes edition facilities
  - Fixes link with concrete syntax model

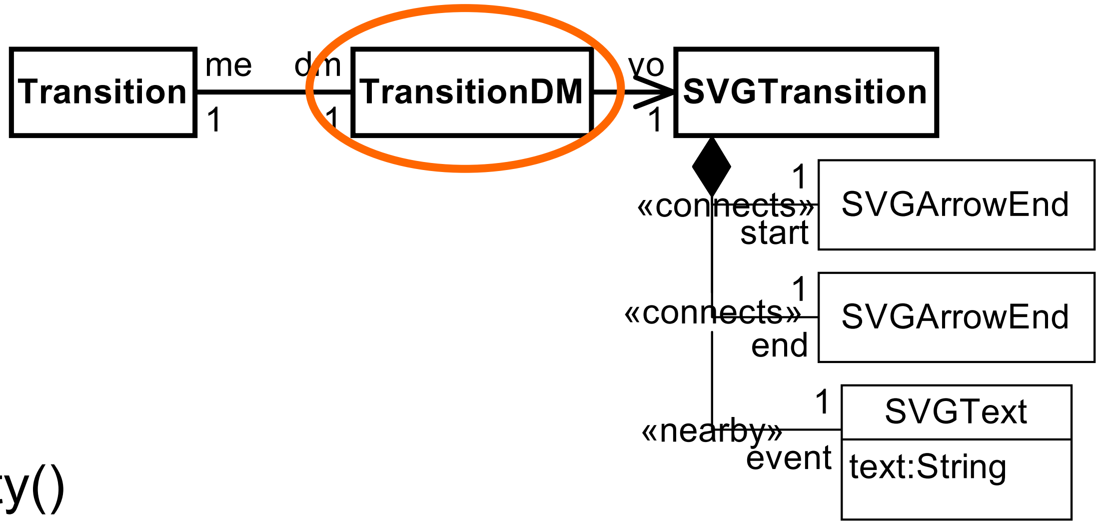
Concrete syntax model



Concrete syntax model

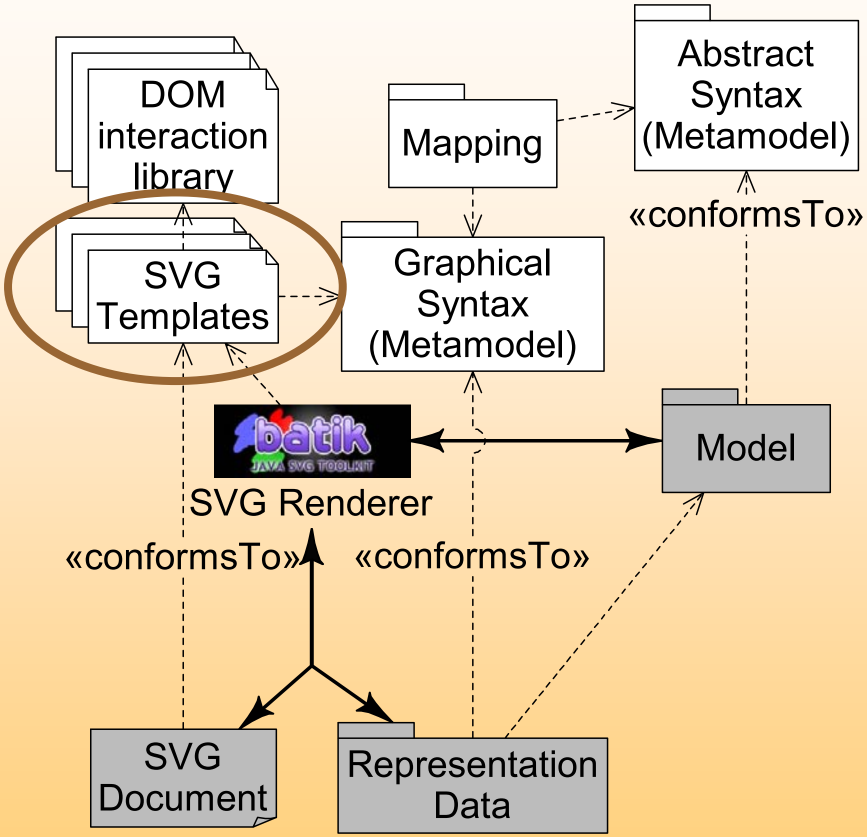
A text is shown on the top of transitions to represent the triggering event if it exists.





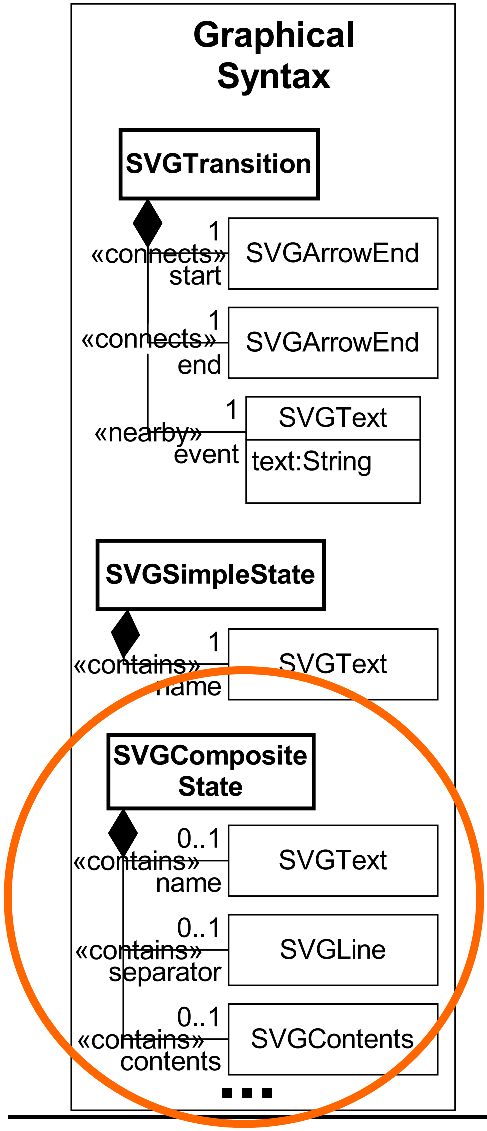

```
context TransitionDM inv:
  if self.me.trigger−>isEmpty()
  then self.vo.event.text.size() = 0
  else self.vo.event.text = self.me.trigger.name
  endif
```

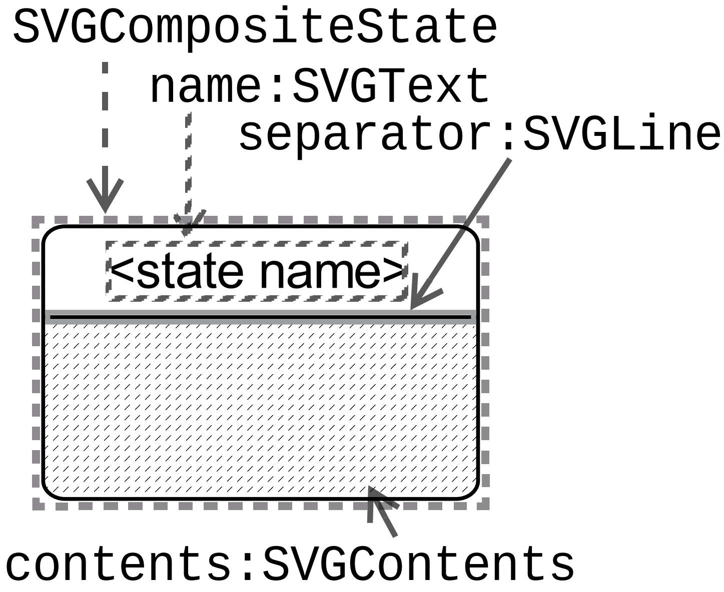
Graphical concrete syntax definition



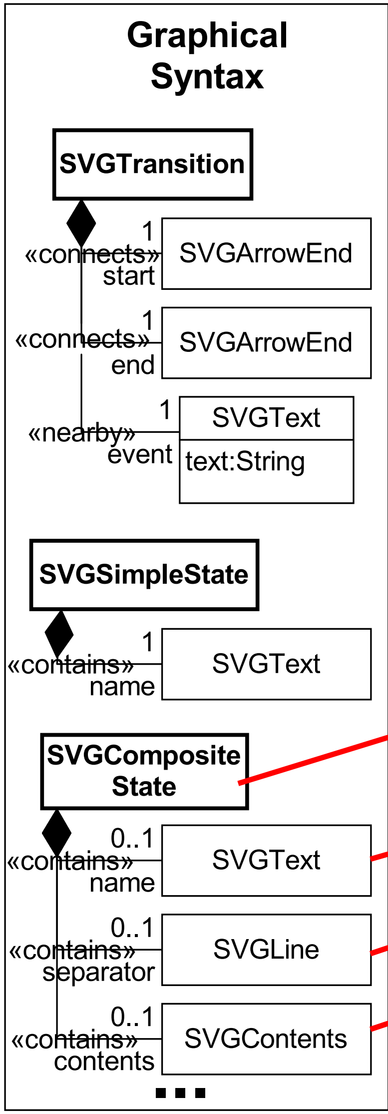
- Concrete syntax model
  - Fixes concrete syntax elements
  - Fixes relationship with abstract syntax
- Concrete syntax graphical design
  - Fixes appearance
  - Fixes layout constraints
  - Fixes edition facilities
  - Fixes link with concrete syntax model

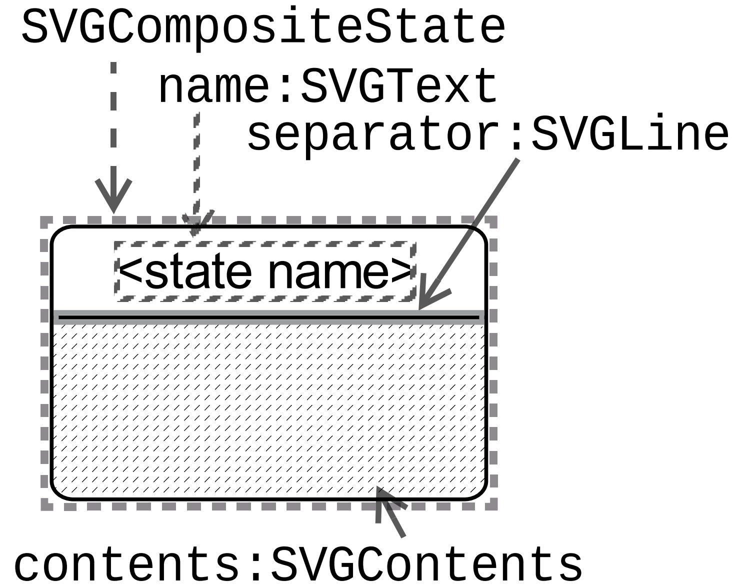
Solving appearance





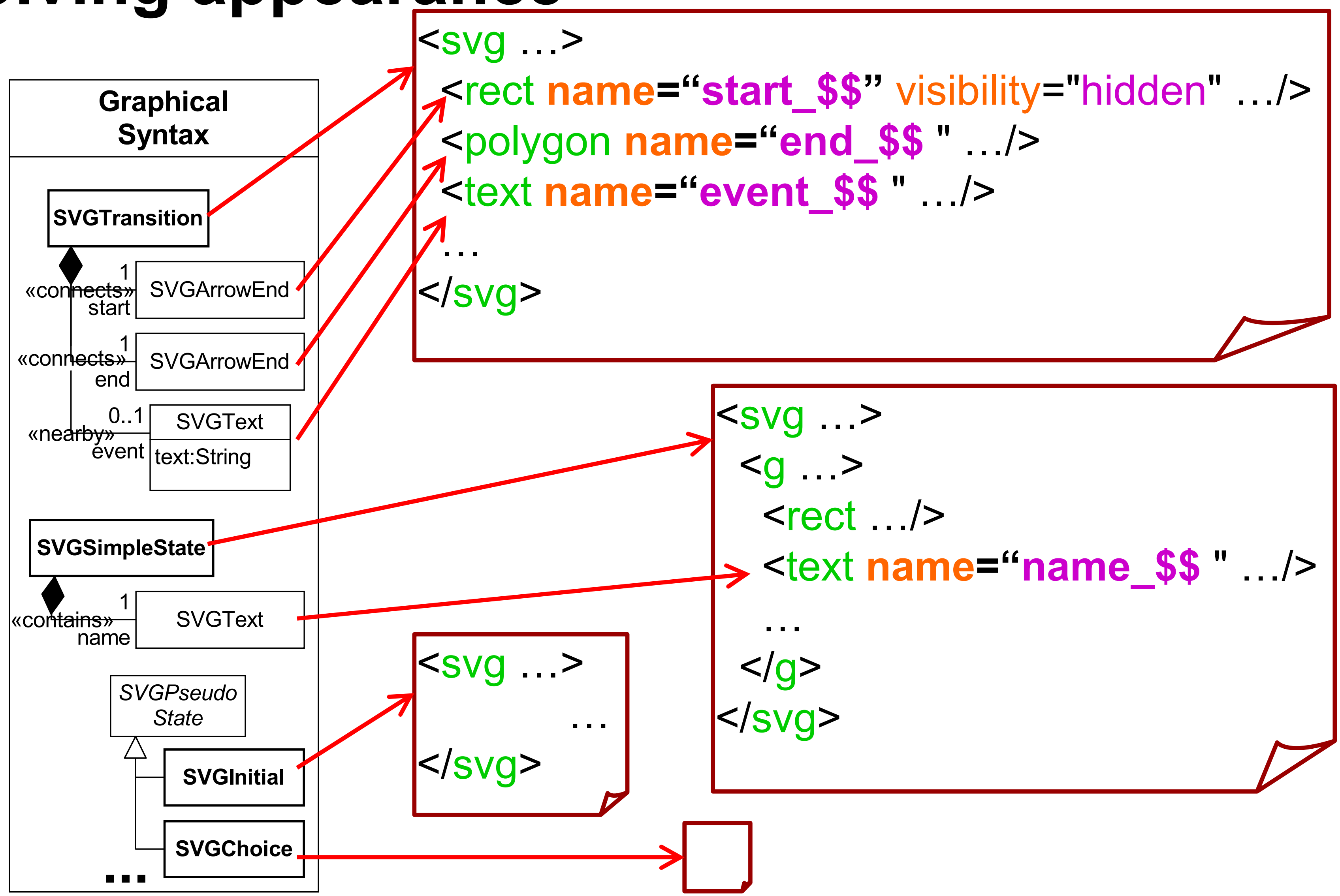
Solving appearance



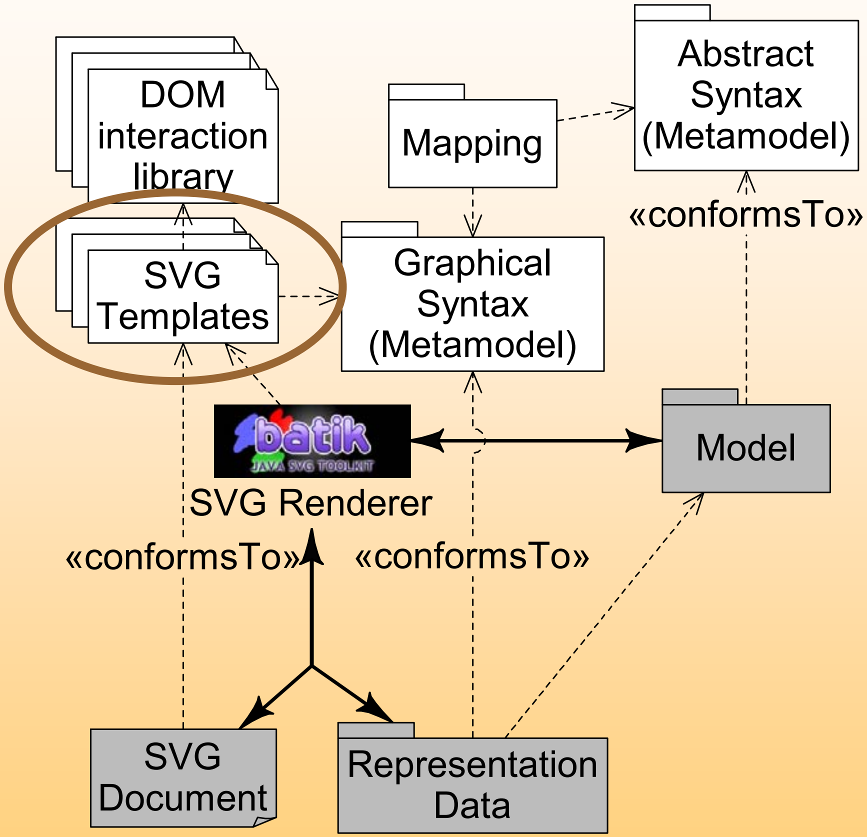



```
<svg …>
 < g id="$$" …>
  <rect id="back_$$" …/>
  <text id="name_$$ " …/>
  <line id="end_$$ " …/>
  <rect id="contents_$$" …/>
  …
 </g>
</svg>
```

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

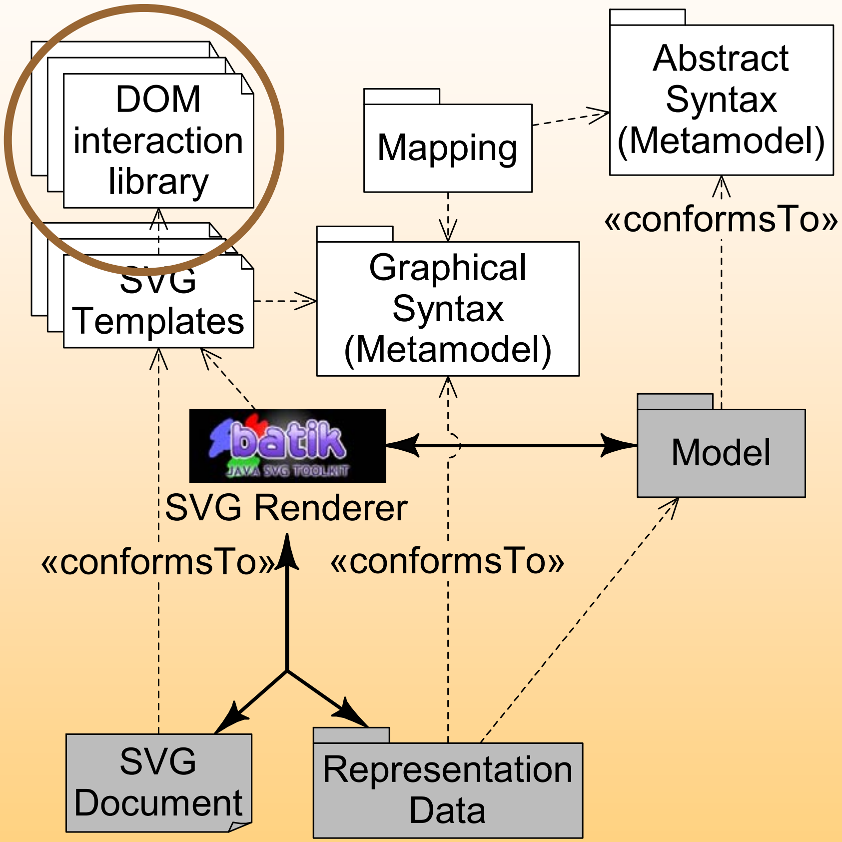
- OCL on graphical syntax metamodel => between elts
- C-SVG : one-way constraints (from Monash Uni.)

  `CompositeState` Template:

  Background should not be smaller than text.

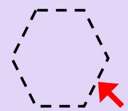
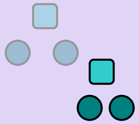
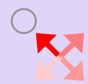
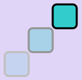
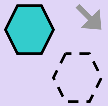
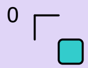
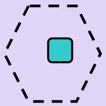
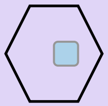
```
<svg …>
  <csvg:variable name="w_$$“
   value="c:max(c:width(c:bbox(id('name_$$'))) + 20, 150)"/>
  <rect …>
   <csvg:constraint attributeName="width" value="$w_$$"/>
  </rect>
  <text name=“name_$$” …>
   <csvg:constraint attributeName="x" value="$w_$$ div 2 - 75"/>
  </text>
  …
</svg>
```

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

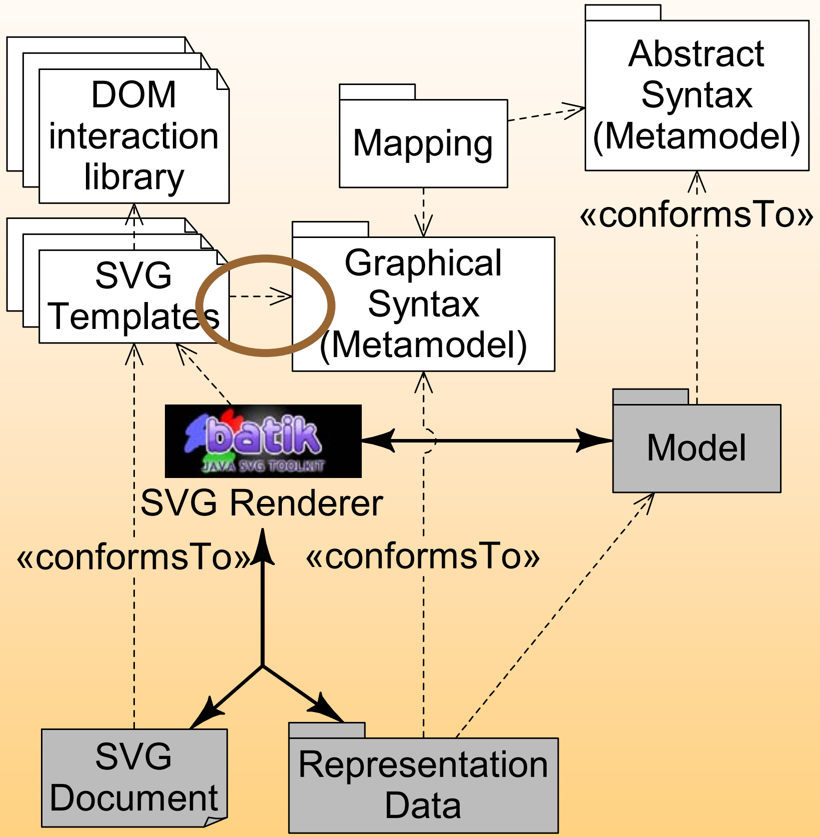
Interface		Interface	
BorderSlidable		Stickable	
DirectionAdjustable		Translatable	
Locatable		BorderFindable	
Positionable		OriginGettable	
Containable		Container	
Editable	edi|	Etc...	

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

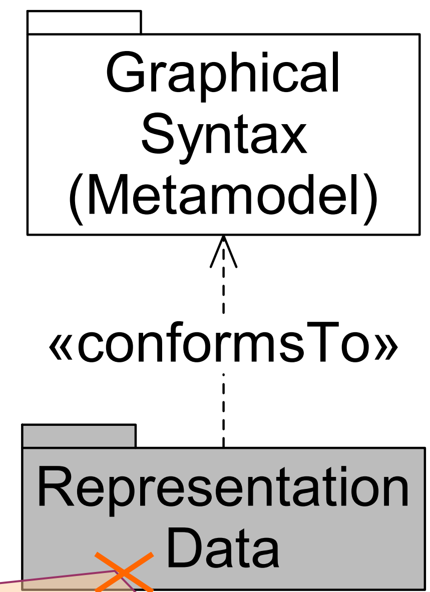
```
<svg
onCreation="s=model.getCompositeStateDM().createCompositeStateDM();
">
  <text name="name_$$" var_self="$s" dpi:component="Editable, …"
   onChange="self.setName(content);"…/>
  …
</svg>
```

Representation Link: Value events

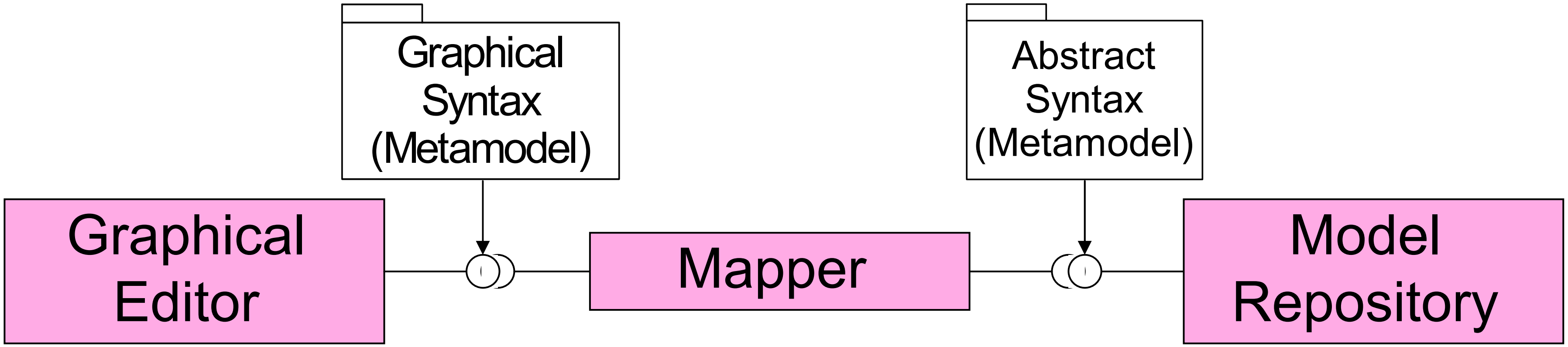
- One listener synchronizing
  - An attribute value on the model with
  - an attribute value on the SVG document

`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>
```



Separation of concerns



↑ **Relates a model and a graphical representation**

↑ **Relates two things of same nature**

- Avoids abstract/concrete syntax pollution
- Improves reusability
- Minimizes maintenance points
- Mapping can be complex (large gap !)

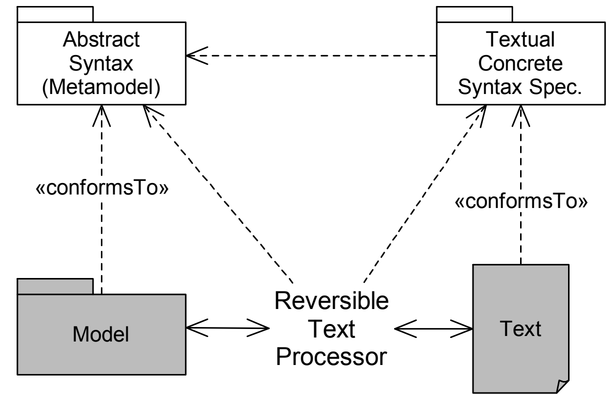
Contents

- Model Driven Engineering
- Concrete Syntaxes
  - Textual concrete syntax definition
  - Graphical concrete syntax definition
- Conclusions

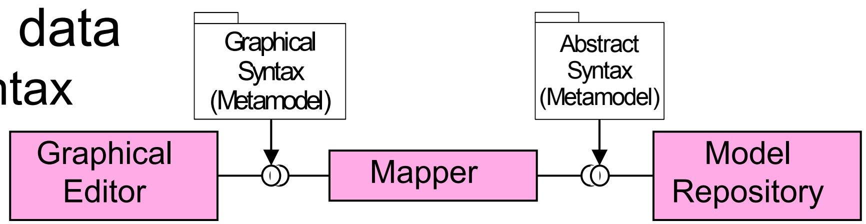
Conclusions

- Language proliferation (MDE+DSM)
  - Language engineering is a key
- Solutions to fill abstract/concrete syntax gap
  - Abstract syntax provided as a metamodel
  - Focus on executable specifications
    - Human readable/producible ?

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



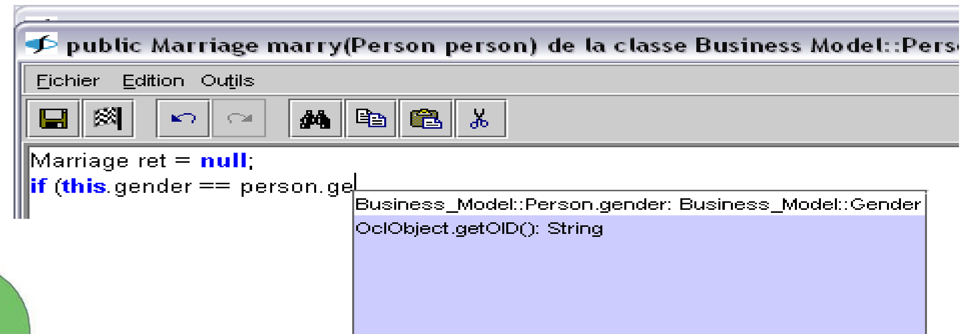
- Technology for graphical concrete syntax realization
  - Representation using SVG templates
  - Library of possible user interactions

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

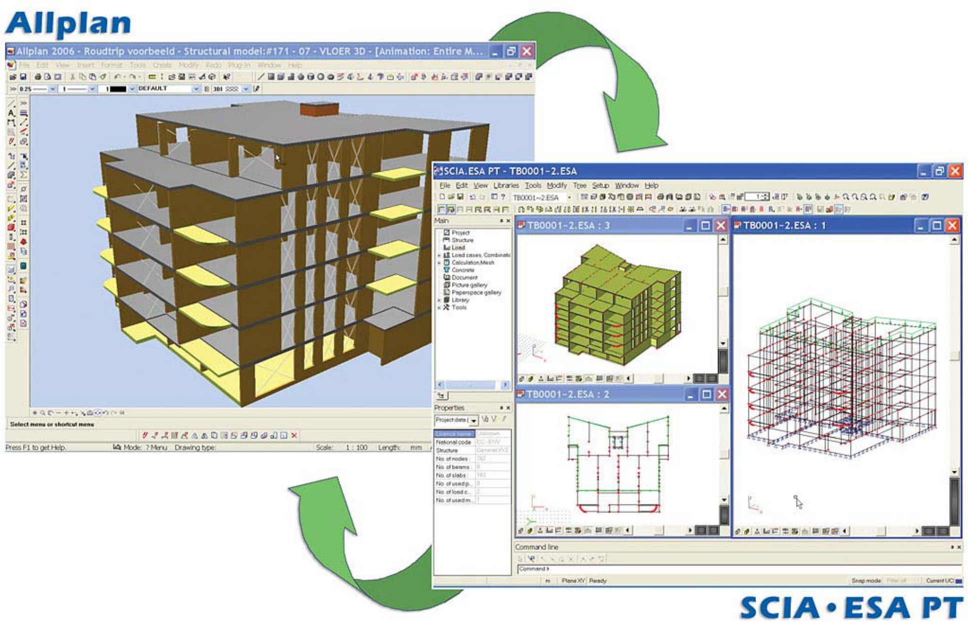
- Other technologies apply
  - (Triple) Graph Grammars
  - GMF, Topcased, …

Analytic –vs.– Interactive CS

- Solutions to textual and graphical CS are very different
  - Textual => usually analytic (with small gap)
  - Graphical => usually interactive (from no gap to large gap)

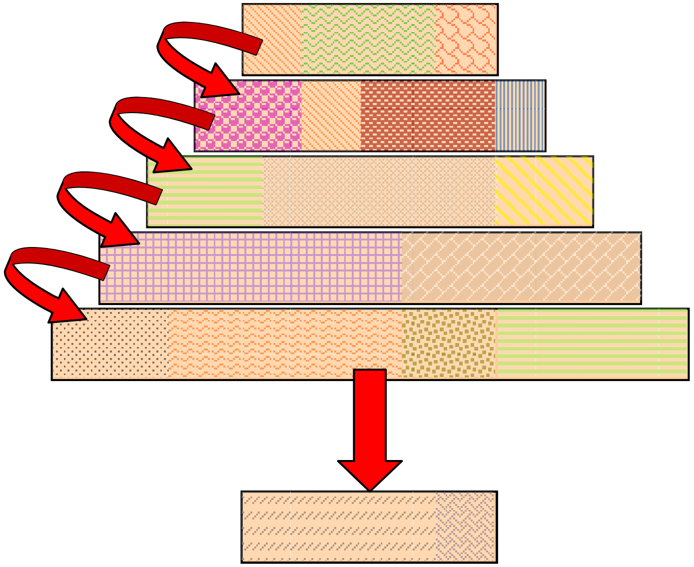


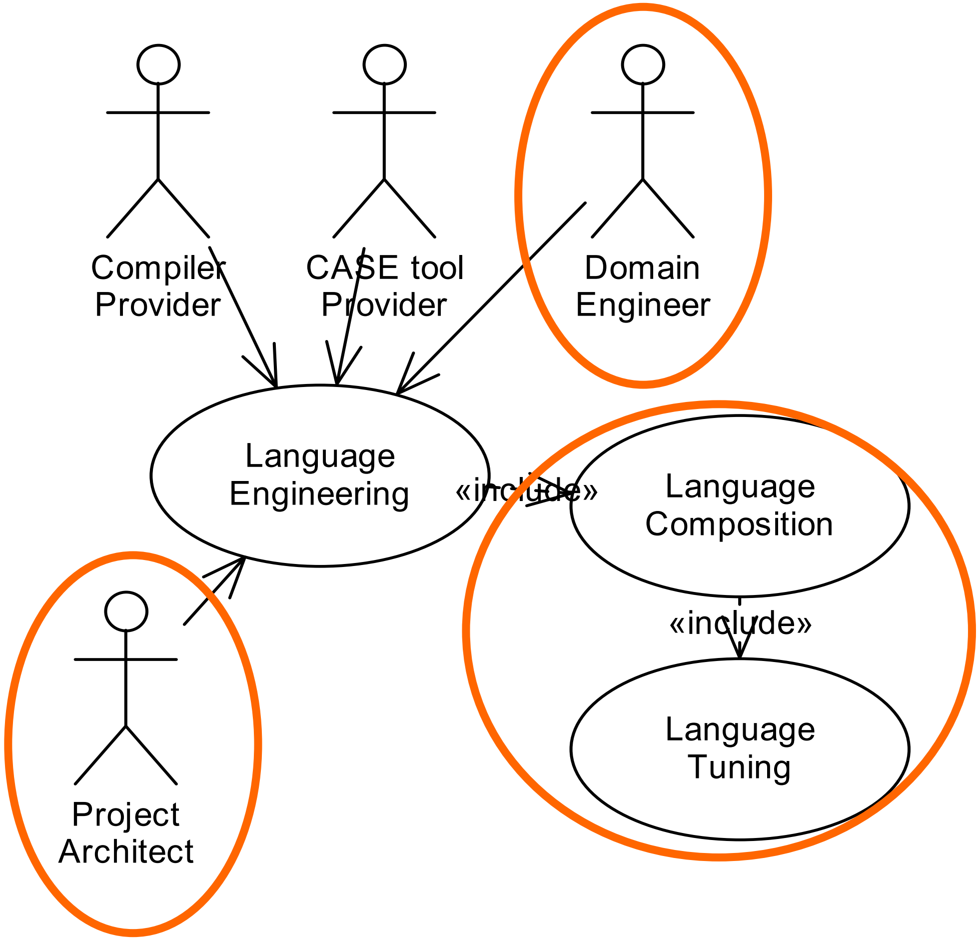
- Unification of solutions ?
- Inversion of solutions ?



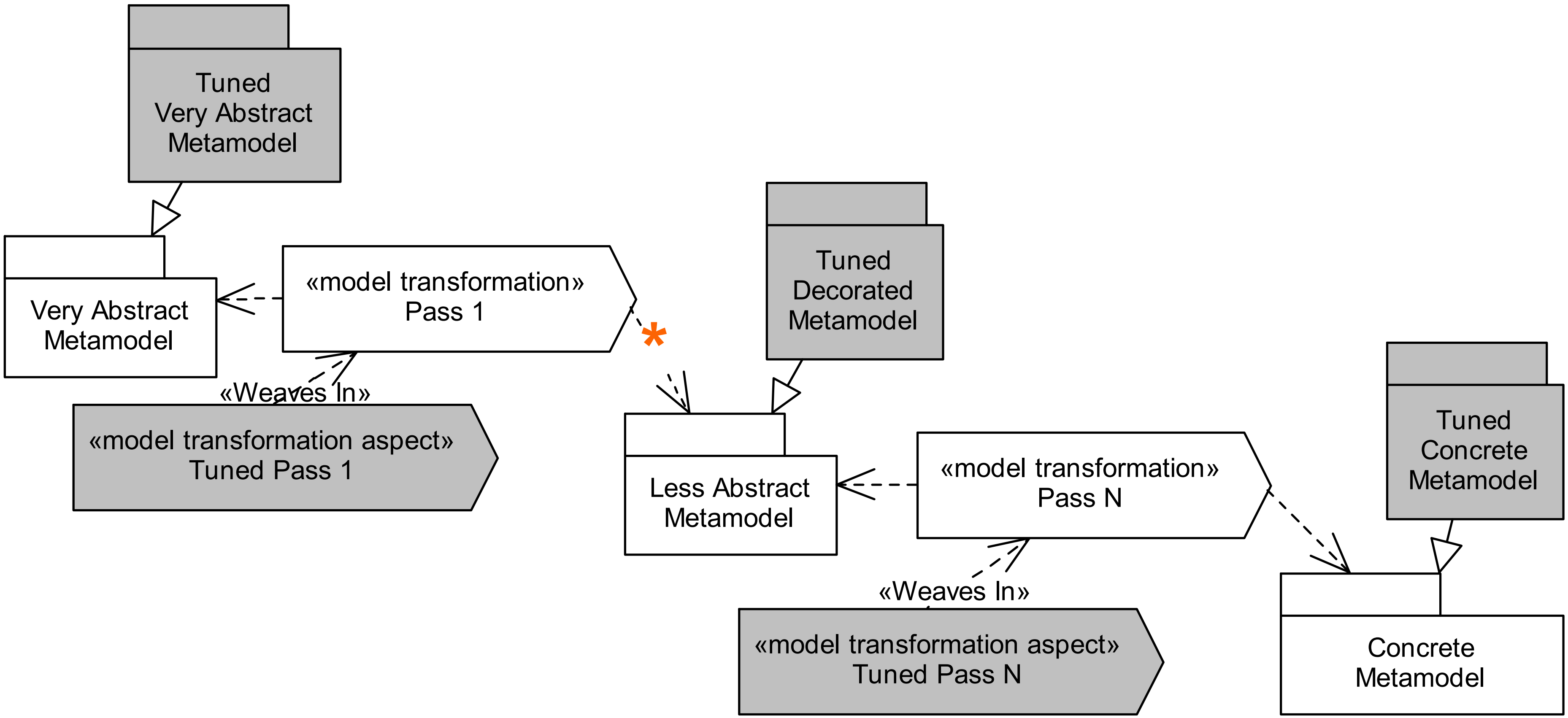
Towards "Agile" Language Engineering

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





Tuning MDE Artefacts



Thank you !