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
  - …

- **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
  - …

Made possible/necessary thanks to/because of evolution of hardware…
**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
Model Driven Engineering (MDE)

“From contemplative to productive”

Organizes abstraction

System → Model
Transformation

System → Model
Transformation

System → Model
Transformation

System → Model
Transformation

System → Code
Code Generation

- Test → GENERATION
- Validation → PROTOTYPING
- Verification → MODEL CHECKING
Model Driven Engineering

- Test → GENERATION
- Validation → PROTOTYPING
- Verification → MODEL CHECKING
Model Driven Engineering

M2
Definition Level (method)

Source Languages (Metamodels)

<<conformsTo>>

Model Transformation

<<conformsTo>>

Target Languages (Metamodels)

M1
Application Level (project)

Source System

<<conformsTo>>

Target System

<<conformsTo>>
Model Driven Engineering

Source Languages (Metamodels) <<conformsTo>> Source System

Model Transformation

Target Languages (Metamodels) <<conformsTo>> Target System

M2
Definition Level (method)

M1
Application Level (project)
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

Concepts

Meaning

Language

Interface
Abstract Syntax

```
ModelElement
  name : String

StateMachine
  source : StateVertex
  outgoing : Transition
  target : State

StateVertex
  subvertex : StateVertex

PseudoState
  kind : PseudoStateKind

State
  top : 1

Transition
  trigger : Event

Composite State
  container : 0..1

Simple State
  container : 0..1

Final State
  container : 0..1

«enumeration»
  PseudoStateKind

initial choice...
```
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

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
```

<= An (M1) sentence
Interface Definition

Abstract Syntax + Concrete Syntax(es)

<table>
<thead>
<tr>
<th>Transition</th>
<th>SimpleState</th>
<th>Composite State</th>
<th>FinalState</th>
<th>PseudoState (initial)</th>
<th>PseudoState (choice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>event&lt;&gt;</td>
<td>name</td>
<td>name</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>contents</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

● In practice
  • Layout constraints
  • User interactions
Abstract Syntax  + Concrete Syntax(es)

<table>
<thead>
<tr>
<th>Transition</th>
<th>SimpleState</th>
<th>Composite State</th>
<th>FinalState</th>
<th>PseudoState (initial)</th>
<th>PseudoState (choice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-event&gt;</td>
<td>name</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<= An (M1) sentence
Meaning Definition

Abstract Syntax + Concrete Syntax(es) + Semantics

M2

Research issue

SC Interpreter

SC Compiler

SC MM

SQL MM
Language Definition

Abstract Syntax + Concrete Syntax(es)

Metamodel

Concrete Syntax(es)

Language Definition

Abstract Syntax

Concrete Syntax

Metamodel

Concrete Syntax
State of the art

Grammarware

MDE

Graphics

Visual Languages
Strategy

Grammarware

MDE

Graphics

Visual Languages
Contents

- Introduction
  - Software Engineering
  - Model Driven Engineering
  - Language Definition

- Concrete Syntaxes
  - Textual concrete syntax definition
  - Graphical concrete syntax definition

- Conclusions
Typical Implementation

- Abstract Syntax (Metamodel)
- Model
- Templates
- Code Generator
- Code
- Semantic analysis
- Compiler compiler
- Intermediate tree
- Grammar

White: M2
Grey: M1
Typical Implementation

Abstract Syntax (Metamodel)

Model

Code Generator

Semantic analysis

Compiler compiler

Intermediate tree

Grammar

Templates

AndroMDA

ANTLR

White: M2
Grey: M1
Typical Implementation

Abstract Syntax (Metamodel) «conformsTo» Code Generator

Model «conformsTo» Semantic analysis

Code Generator «conformsTo» Compiler

Intermediate tree «conformsTo» Grammar

Templates «conformsTo» Code

White: M2
Grey: M1
**Typical Implementation**

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

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

White: M2
Grey: M1

Abstract Syntax (Metamodel) «conformsTo»
Model «conformsTo»
Abstract Syntax (Metamodel) «conformsTo»
Code Generator «conformsTo»
Semantic analysis «conformsTo»
Intermediate tree

Code generation templates, Grammar, and Semantic analysis

- Double conformance!
  - Double specification…
  - Triple maintenance points…
  - No automated coherence proof…
No concrete syntax
• Help yourself!
Inspired from
- EBNF
- Text structure
- Netsilon
Inspired from

- EBNF
- Netsilon

• Model Navigation
Prototypes: Sintaks

Abstract Syntax (Metamodel) «conformsTo» Textual Concrete Syntax Spec. «conformsTo» Model «conformsTo» Text

Sintaks
Prototypes: TCSSL Tools

TCSSL tools

Abstract Syntax (Metamodel)

Textual Concrete Syntax Spec.

Reversible Text Processor

Text

Model

Template

Code

ANTLR

Grammar

Intermediate tree

Compiler

Semantics analysis

Jet Code Generator

Abstract Syntax (Metamodel)

«conformsTo»

«conformsTo»

«conformsTo»

«conformsTo»

«conformsTo»

«conformsTo»

«conformsTo»

«conformsTo»

«conformsTo»

«conformsTo»

November 2007
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 `later` rule
  - A problem for rule precedence
- Tests limited to comparisons
  - Tests interpreted as actions at analysis

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

Flexibility

Safety
Question

What is the good Gap?
Contents

- Introduction
  - Model Driven Engineering
  - Language Definition

- Concrete Syntaxes
  - Textual concrete syntax definition
  - Graphical concrete syntax definition

- Conclusions
Typical Implementation

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

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

White: M2
Grey: M1

Interaction
Mapping
Representation Language

Abstract Syntax (Metamodel)

«conformsTo»

SVG Renderer

predefined interaction library
Representation Templates
Graphical Syntax (Metamodel)

Diagram
Representation Data

Meta-CASE

Model

Language

Interaction «conformsTo»

Meta-CASE «conformsTo»

Diagram «conformsTo»

Representation Data «conformsTo»
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

<table>
<thead>
<tr>
<th>Abstract Syntax</th>
<th>Mapping</th>
<th>Graphical Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition</td>
<td>TransitionDM</td>
<td>SVGTransition</td>
</tr>
<tr>
<td>Simple State</td>
<td>Simple StateDM</td>
<td>SVGSimpleState</td>
</tr>
<tr>
<td>Composite State</td>
<td>Composite StateDM</td>
<td>SVGCompositeState</td>
</tr>
</tbody>
</table>

- Transition contains (1..1) TransitionDM
- Simple State contains (1) Simple StateDM
- Composite State contains (0..1) Composite StateDM
- SVGTransition connects (1) start
- SVGArrowEnd
- SVGArrowEnd
- SVGText
  - text: String
- SVGSimpleState connects (1) nearby
- SVGText
- SVGCompositeState contains (1) name
- SVGText
- SVGLine
- SVGContents

«Interface» GraphicalObject
contains() connects() nearby() overlaps()
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

Graphical Syntax

SVGTransition

1

«connects»

start

SVGArc

SVGArcEnd

1

«connects»

end

SVGArc

1

«connects»

event

SVGText

text: String

SVGCompositeState

name: SVGText

separator: SVGLine

state name

contents: SVGContents
Solving appearance

Graphical Syntax

SVGTransition

SVGArrowEnd

SVGArrowEnd

SVGText
text:String

SVGSimpleState

SVGSimpleState

SVGCompositeState

0..1

SVGText

0..1

SVGLine

0..1

SVGContents

<svg ...>
  <g id="$$" ...>
    <rect id="back_$$" .../>
    <text id="name_$$" .../>
    <line id="end_$$" .../>
    <rect id="contents_$$" .../>
  </g>
...</svg>
Solving appearance

Graphical Syntax

SVGTransition

«connects» start

1

SVGArrowEnd

«connects» end

1

SVGArrowEnd

«connects» «nearby» event

0..1

SVGSimpleState

«contains» name

1

SVGText

text:String

SVGChoice

SVGInitial

...
Graphical concrete syntax definition

- Concrete syntax model
  - Fix concrete syntax elements
  - Fix relationship with abstract syntax

- Concrete syntax graphical design
  - Fix appearance
  - Fix layout constraints
  - Fix edition facilities
  - Fix 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.

```xml
<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

<table>
<thead>
<tr>
<th>Interface</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>BorderSlidable</td>
<td>Stickable</td>
</tr>
<tr>
<td>DirectionAdjustable</td>
<td>Translatable</td>
</tr>
<tr>
<td>Locatable</td>
<td>BorderFindable</td>
</tr>
<tr>
<td>Positionable</td>
<td>OriginGettable</td>
</tr>
<tr>
<td>Containable</td>
<td>Container</td>
</tr>
<tr>
<td>Editable</td>
<td>Etc...</td>
</tr>
</tbody>
</table>
Solving edition facilities

CompositeState template

```xml
<svg ...
  <g dpi:component="Containable, Translatable, ...">
    <rect dpi:component="BorderFindable, ...">
    <rect dpi:component="Container, ...">
    <text dpi:component="Editable, ...">
        ...
    </text>
  </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

```xml
<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

```xml
<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

Graphical Syntax (Metamodel)  Mapper  Abstract Syntax (Metametaol)

Graphical Editor  Model Repository

↑  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

- **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

Very Abstract Metamodel

«model transformation» Pass 1

Tuned Very Abstract Metamodel

«Weaves In»

Very Abstract Metamodel

«model transformation aspect» Tuned Pass 1

Tuned Decorated Metamodel

«Weaves In»

Less Abstract Metamodel

«model transformation» Pass N

Tuned Concrete Metamodel

«Weaves In»

«model transformation aspect» Tuned Pass N

Concrete Metamodel
Thank you!