DSLs: techniques and tools

Frédéric Fondement

modeling wizards

October, 2010
Example DSL models

\[ X(z) = Z\{x(n)\} = \sum_{n=-\infty}^{\infty} x(n) z^{-n} \]
Supporting DSLs
Plan

- Tools around models
- Supporting / modeling languages
  - Abstract syntax
  - Concrete syntax
  - Transformation
- Language models reuse
  - Reusing abstract syntax
  - Reusing tools
Model as a primary artifact

Repository architectural style

Roughly taken from Ian Sommerville
Supporting models

MODEL REPOSITORY
Supporting models

GRAPHICAL EDITOR

SYNCHRONIZATION

MODEL REPOSITORY

PARSING / SERIALIZING

TEXTUAL EDITOR

Take 3 units from cup 1 and close 1.
Supporting models
Supporting models

MODEL REPOSITORY

SERIALIZER

dvm.xml
XML Document
42 KB

XML File
Supporting models
Supporting models
Supporting models

MODEL REPOSITORY

MODEL REPOSITORY

TRANSLATION

TRANSLATION

MODEL INTERPRETOR

CODE

Modeling Wizards - DSLs: techniques and tools
Supporting models

INTERACTION
GRAPHICAL EDITOR

CONTROLLED STORAGE
CONSISTENCY CHECKING

OPERATIONALIZATION
MODEL REPOSITORY

MODEL INTERPRETOR

PARSING / SERIALIZING
TEXTUAL EDITOR

SYNCHRONIZATION

XMI File

MODEL REPOSITORY

TRANSLATION

XML Document
42 KB

SERIALIZER

CODE

TRANSLATION

MODEL INTERPRETOR

- 13 -

Modeling Wizards - DSLs: techniques and tools
Plan

- Tools around models

- Supporting / modeling languages
  - Abstract syntax
  - Concrete syntax
  - Transformation

- Language models reuse
  - Reusing abstract syntax
  - Reusing tools
Languages...
1 language: many sentences

Cost

Number of models

Dedicated tool support

No tool support
1 language: many sentences

Cost

Number of models

Dedicated tool support

Price of tool support

No tool support

Modeling Wizards - DSLs: techniques and tools
1 language: many sentences

A modeling language deserves to be supported.

Dedicated tool support

No tool support

Cost

Number of models
1 language: many sentences

A modeling language deserves to be supported.

Dedicated tool support made easy.

No tool support.

Cost vs. Number of models graph.
(Modeling) language
(Modeling) language

Concepts

Language

Interface

Modeling Wizards - DSLs: techniques and tools
(Modeling) language

Concepts

Meaning

Language

Interface
Modeling (modeling) language

- Concepts
- Meaning
- Interface

A model

Modeling Wizards - DSLs: techniques and tools
Modeling language concepts

- Interaction
  - Graphical Editor

- Controlled Storage
  - Consistency Checking
  - Model Repository
  - Serializer
  - XMI File

- Operationalization
  - Translation
  - Model Repository
  - Code

- Textual Editor
  - Parsing / Serializing

- Synchronization
  - Interoperability
  - Operationalization

- Model Interpreter
  - Translation
Modeling language concepts

- Interaction
- Graphical Editor
- Synchronization
- Parsing / Serializing
- Textual Editor

- Controlled Storage
- Consistency Checking
- Serializer
- XML File

- Operationalization
- Model Repository
- Translation

Modeling Wizards - DSLs: techniques and tools
Modeling language concepts

A model

A concepts modeling language!
(a.k.a. a metamodeling language)
Instances

In databases

![Database diagram with tables and instances]

<table>
<thead>
<tr>
<th>Name</th>
<th>Code_per</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durand</td>
<td>1</td>
</tr>
<tr>
<td>Dupont</td>
<td>2</td>
</tr>
<tr>
<td>Schmitt</td>
<td>3</td>
</tr>
<tr>
<td>Martin</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code_per</th>
<th>Code_org</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code_org</th>
<th>Organisme</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ENSISA</td>
</tr>
<tr>
<td>2</td>
<td>FLSH</td>
</tr>
<tr>
<td>3</td>
<td>FST</td>
</tr>
<tr>
<td>4</td>
<td>IUT</td>
</tr>
</tbody>
</table>
Instances

- In databases

```
Table "Personnes"

<table>
<thead>
<tr>
<th>Nom</th>
<th>Code_per</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durand</td>
<td>1</td>
</tr>
<tr>
<td>Dupont</td>
<td>2</td>
</tr>
<tr>
<td>Schmitt</td>
<td>3</td>
</tr>
<tr>
<td>Martin</td>
<td>4</td>
</tr>
</tbody>
</table>
```

```
Table "Affiliation"

<table>
<thead>
<tr>
<th>Code_per</th>
<th>Code_org</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
```

```
Table "Organismes"

<table>
<thead>
<tr>
<th>Code_org</th>
<th>Organisme</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ENSISA</td>
</tr>
<tr>
<td>2</td>
<td>FLSH</td>
</tr>
<tr>
<td>3</td>
<td>FST</td>
</tr>
<tr>
<td>4</td>
<td>IUT</td>
</tr>
</tbody>
</table>
```
Instances

- In databases

Table "Personnes"

<table>
<thead>
<tr>
<th>Nom</th>
<th>Code_per</th>
</tr>
</thead>
<tbody>
<tr>
<td>Durand</td>
<td>1</td>
</tr>
<tr>
<td>Dupont</td>
<td>2</td>
</tr>
<tr>
<td>Schmitt</td>
<td>3</td>
</tr>
<tr>
<td>Martin</td>
<td>4</td>
</tr>
</tbody>
</table>

Table "Affiliation"

<table>
<thead>
<tr>
<th>Code_per</th>
<th>Code_org</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table "Organismes"

<table>
<thead>
<tr>
<th>Code_org</th>
<th>Organisme</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ENSISA</td>
</tr>
<tr>
<td>2</td>
<td>FLSH</td>
</tr>
<tr>
<td>3</td>
<td>FST</td>
</tr>
<tr>
<td>4</td>
<td>IUT</td>
</tr>
</tbody>
</table>

«instanceOf» «instanceOf»
Instances

- In databases

![Database diagram]

- Table "Personnes"
  - Nom: Durand, Dupont, Schmitt, Martin
  - Code_per: 1, 2, 3, 4

- Table "Affiliation"
  - Code_per: 1, 2, 3, 4
  - Code_org: 1, 3, 4

- Table "Organismes"
  - Code_org: 1, 2, 3, 4
  - Organisme: ENSISA, FLSH, FST, IUT
Instances

- In XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

  <xs:element name="personne" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNamespaceSchemaLocation="personne.xsd">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nom" type="xs:string"/>
        <xs:element name="prenom" type="xs:string"/>
        <xs:element name="date_naissance" type="xs:date"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<personne xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="personne.xsd">
  <nom>de Latour</nom>
  <prenom>Jean</prenom>
  <date_naissance>1967-08-13</date_naissance>
</personne>
```
Instances

- EBNF

program = 'PROGRAM' identifier 'BEGIN' (assignment ";")* 'END' ;
identifier = alphabetic_character (alphabetic_character | digit)* ;
number = ("-")? (digit)+ ;
string = "" (alphabetic_character | digit | other)* ""* "" ;
assignment = identifier , "=" , ( number | identifier | string ) ;

PROGRAM DEMO1 BEGIN
A0:=3;
B:=45;
H:=-100023;
D123:=B34A;
BABOON:=GIRAFFE;
TEXT:="Hello world!";
END

«conformsTo»
Instances

- With models
Metamodeling example

Type

Instance

Metamodel

Model

«instanceOf»

«metaObject»

Modeling Wizards - DSLs: techniques and tools
Metamodeling example

```plaintext
«metamodel»
DataBaseMM

DataBase
  dataBase
  table A_table_dataBase *

Table
  table
  column A_column_table *

NamedElement
  name : string

TypedElement
  isPrimaryKey : boolean

Column
  isPrimaryKey : boolean

DataType
  type
  A_type_typedElement
```

Modeling Wizards - DSLs: techniques and tools
Modeling language concepts

- Concepts for the language used to model a metamodel
Metamodeling example

Type

Instance

Metamodel

Model

- 39 -

Modeling Wizards - DSLs: techniques and tools
Metamodeling example

Instance Of

M3

M2

M1

Modeling Wizards - DSLs: techniques and tools
Typical metamodeling constructs

- Class
- Property
- Association
- Generalization
Metamodeling languages

- MOF
- Ecore (supported by Eclipse EMF)
- KM3 (AMMA)
- Xcore (XMF)
- SMD (Coral)
- ...
Metamodeling language property

- MMM can have MMM as a meta-metamodel
- MMM is thus a metamodel
- Thus, a metamodeling language can express itself
- MMM can have itself as a metamodel
Metamodeling language property

The following is possible:
(though not mandatory)
Example: EBNF

```
rules = (rule)* ';';
rule = identifier '=' ruleBody;
ruleBody = choice | sequence | atom | '(' ruleBody ')' (('?'|'*'|'+'))?;
choice = ruleBody ('|' ruleBody)*;
sequence = ruleBody (ruleBody)*;
atom = identifier | """ (character | digit | other)* """;
identifier = character (character | digit)*;
```

«conformsTo»
Example: XSD

An XML Schema schema document for XML Schema schema documents:
http://www.w3.org/2001/XMLSchema.xsd
© W3C
Nobody’s perfect…

…even metamodeling languages!

Metamodel:
- Structure that makes possible to construct models…
- …still, some of those possible models can be invalid (i.e. out of the desired language)!

\[\text{Person} \begin{array}{c} \text{parents} \ 0,2 \ \text{children} \\
\end{array} \]

[Diagram showing relationships between Person, Bob, Alice, and Philip]
Constraints

- Metamodel:
  - Structure that makes possible to construct models

- Static semantics
  a.k.a. Well formedness rules
  - Expressions restricting the number of possible models

Consistency model
\[ \Downarrow \]
Metamodel

Modeling Wizards - DSLs: techniques and tools
Consistency Model

Object Constraint Language

- A model query language
- Can be adapted to other class languages than UML
- Can be used to
  - Additional operations / derived attributes
  - Constraints (!)
  - Transformation languages
  - Code generation rules
  - ...

Other languages (e.g. Check), same principles
Modeling (modeling) language

A model

Concepts

Language

Interface

A model

Modeling Wizards - DSLs: techniques and tools
Supporting models

INTERACTION
GRAPHICAL EDITOR

CONTROLLED STORAGE
CONSISTENCY CHECKING
MODEL REPOSITORY
XMI File
SERIALIZER
PARSING / SERIALIZING
TEXTUAL EDITOR
SYNCHRONIZATION
MODEL REPOSITORY

OPERATIONALIZATION
MODEL REPOSITORY
CODE
TRANSATION
MODEL INTERPRETOR

Modeling Wizards - DSLs: techniques and tools
Language description

<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>name</td>
<td>name</td>
<td>contents</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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
```

+ layout constraints
Textual concrete syntax

Possible language gap…

Abstract Syntax (Metamodel) «conformsTo»

Model «conformsTo»

Code Generator «conformsTo»

Semantic analysis «conformsTo»

Intermediate tree «conformsTo»

Code «conformsTo»

Templates «conformsTo»

Grammar «conformsTo»

M2: white
M1: grey

Modeling Wizards - DSLs: techniques and tools
Textual concrete syntax

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

M2: white
M1: grey

Textual concrete syntax models

Modeling Wizards - DSLs: techniques and tools
Textual concrete syntax

- Abstract Syntax (Metamodel)
- Templates
- Code Generator
- Model
- Text
- Grammar

Modeling Wizards - DSLs: techniques and tools
Textual concrete syntax

Abstract Syntax (Metamodel) «conformsTo»

Model

Reversible Text Processor

SINTAKS

Text «conformsTo»

Grammar

Textual concrete syntax model

M2: white
M1: grey
Textual concrete syntax patterns

- A ::= (pub?=‘public’)? ‘AnA’ name=ID
- A ::= B | C | D
- A ::= (bs+=B)+
Textual concrete syntax

Abstract Syntax (Metamodel) «conformsTo» Model

Grammar «conformsTo» Text

Text Processor

Modeling Wizards - DSLs: techniques and tools
Textual concrete syntax

M2: white
M1: grey

Abstract Syntax (Metamodel) «conformsTo»

Model «conformsTo» Text Processor

Xtext Text

Grammar «conformsTo»

Modeling Wizards - DSLs: techniques and tools
Textual concrete syntax
<?xml version="1.0" encoding="UTF-8"?>
<sc:StateMachine xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sc="http://dummy/sc" name="">
  <states xsi:type="sc:InitialState" outgoing="//@transitions.0"/>
  <states xsi:type="sc:SimpleState" name="Opened" incoming="//@transitions.0 //@transitions.2
  outgoing="//@transitions.1"/>
  <states xsi:type="sc:CompositeState" name="Closed" incoming="//@transitions.1">
    <states xsi:type="sc:SimpleState" name="Unlocked" incoming="//@states.2/@transitions.1 //@transitions.0"
    outgoing="//@states.2/@transitions.2"/>
    <states xsi:type="sc:InitialState" outgoing="//@states.2/@transitions.2"/>
    <states xsi:type="sc:SimpleState" name="Locked" incoming="//@states.2/@transitions.0
    outgoing="//@states.2/@transitions.1 //@transitions.2">
      <transitions event="lock" from="//@states.2/@states.0" to="//@states.2/@states.2"/>
      <transitions event="unlock" from="//@states.2/@states.2" to="//@states.2/@states.0"/>
      <transitions from="//@states.2/@states.1" to="//@states.2/@states.0"/>
    </states>
  </states>
  <transitions from="//@states.0" to="//@states.1"/>
  <transitions event="close" from="//@states.1" to="//@states.2"/>
  <transitions event="open" from="//@states.2/@states.2" to="//@states.1"/>
</sc:StateMachine>
Language description

<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</td>
<td>name</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>contents</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

+ layout constraints

-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
```

Modeling Wizards - DSLs: techniques and tools
Graphical concrete syntax

[Diagram]

- Modeling Wizards - DSLs: techniques and tools
Graphical concrete syntax

In practice, more or less
- mixed
- hidden
Graphical concrete syntax

Notions for
- **Shape**
  - Rectangle
  - Polyline
  - ...
- **Connector**
  - Connector end
  - ...

- **Representation Language**

- **Diagram**

- **Model**

- **Interaction**

- **Mapping**

- **Abstract Syntax (Metamodel)**
Graphical concrete syntax

- Possible graphical interactions
  - Move
  - Resize
  - …

- Layout constraints

Diagram

Representation Data

Model

Interaction

Representation Language

Mapping

Abstract Syntax (Metamodel)

Graphical Model Representer

«conformsTo»

Possible graphical interactions

Diagram

Representation Data
Layout constraints?
Layout constraints?
Graphical concrete syntax

- Mapping icons with model elements
  - Synchronization between diagram and model
Graphical concrete syntax

There is an icon III of shape XXX located on diagram DDD at position X, X sized W, H that represents element YYY in the model
Graphical concrete syntax

Diagram Interchange
Modeling (modeling) language

- Concepts
- Meaning
- Language

A model

Interface
A model
A model
A model

Modeling Wizards - DSLs: techniques and tools
Modeling language concepts

- Modeling Wizards - DSLs: techniques and tools
Model operationalization

Transformation problem

- Model to model transformation (M2M)
- Model to text transformation (M2T)
- Model interpretation
  - Only for computer-executable languages!
M2M Transformation model
Transformation model
Transformation model

M2
Definition Level
(method)
Source Language(s) (Metamodel(s))
<<conformsTo>>

Target Language(s) (Metamodel(s))
<<conformsTo>>

M1
Application Level (project)

<<conformsTo>>

<<conformsTo>>

<<conformsTo>>
Transformation model

M3

M2

Definition Level (method)

Source Language(s)
(Metamodel(s))

Target Language(s)
(Metamodel(s))

M1

Application Level (project)

Model Transformation Language

<<conformsTo>>

<<conformsTo>>

<<conformsTo>>
Information / intention origins
Information / intention origins
Reversibility

MISSING

MISSING
Reversibility

Trace model:
- to recover removed information
- to retrieve information source
Transformation hints

- Look for input model elements
  - According to properties
    - Metaclas
    - OCL can help
  - Create output model elements
    - Information trasmission

- 2 ways
  - Imperative
    - Clear steps
  - Declarative
    - Declare only transformation information
Imperative example : Kermeta

**operation** transform(inputModel : Package) : RDBMSModel is do

...  

// Create tables
inputModel.classes.select{ c | c.is_persistent }.each{ c |
  var table : Table init Table.new
  table.name := c.name
  class2table.storeTrace(c, table)
  result.table.add(table)}

// Create columns
inputModel.classes.select{ c | c.is_persistent }.each{ c |
  createColumns(class2table.getTargetElem(c), c)}

...

end

**operation** createColumns (t: Table, c : Class) is do ...

Declarative example:

```
rule PersistentClass2Table {
  from c : SimpleUML!Class {
    c.isPersistent
  }
  to t : RDBMS!Table {
    name <- c.name,
    ...
  }
}

rule ClassicAttribute2Column {
  from a : SimpleUML!Attribute {
    a.container.isPersistent
  }
  to c : RDBMS!Column {
    name <- a.name,
    type <- a.type.kindAsString
  }
}
```
Transformation languages

- **Declarative**
  - QVT relational
  - Triple graph grammars (MOFLON-TGG)
    - What to seek
    - What to produce
    - What to trace

- **Imperative**
  - QVT operational
  - Any programming language (Java, Python, …)
    - API offered by model repositories (JMI)

- **Incremental / Bidirectional**
  - Possible number of in/out/inout models/metamodels
    - E.g. refactorings
M2T: code generators

- Serialization methods
  - Added to metaclasses

- Based on templates

- Xpand example:

```xpand
DEFINE run FOR StateMachine

EXPAND run_sign FOR this

// State Machine «this.name»

EXPAND run_output

FOREACH this.states.select(e|!e.outgoing.isEmpty)
  SEPARATOR "\n"

EXPAND run FOREACH this.states

ENDDEFINE
```
Interpreting models

- Most of the time ad-hoc visitors
- “exec” function injections
  - Kermeta
Interpreting models

- Most of the time ad-hoc visitors
- “exec” function injections
- Model with Ecore:
Interpreting models

- Most of the time ad-hoc visitors
- “exec” function injections
- Model with Kermeta:
All-in-one tools

- Metamodeling language
- Concrete syntax definition language
  - Usually graphical editor for graphical concrete syntax definition
- Transformation language
  - … and / or code generator
All-in-one tools

- Metamodeling language

- Concrete syntax definition language
  - Usually graphical editor for graphical concrete syntax definition

- Transformation language
  - … and / or code generator
Co-evolution

GRAPHICAL EDITOR

CONSISTENCY CHECKING

MODEL REPOSITORY

PARSING / SERIALIZING

TEXTUAL EDITOR

SYNCHRONIZATION

SERIALIZER

XML File

MODEL INTERPRETOR

METAMODEL

MODEL REPOSITORY

CODE

TRANSITION

<<conformsTo>>
Reuse
Plan

- Tools around models
- Supporting / modeling languages
  - Abstract syntax
  - Concrete syntax
  - Transformation
- Language models reuse
  - Reusing abstract syntax
  - Reusing tools
Tuning languages

- Use decorations
  - UML Profiles
- Higher-order hierarchies
  - Package merge

Tool support
- Dedicated tools
- Existing tools improved (e.g. with AOP)
- “Folding” extensions to the original language

Decorated Metamodel

Metamodel

«model transformation»
Decoration Folding

«merge»
Sébastien will introduce that!

A language can become extensible by introducing the profile notion in its metamodel.
- So do UML and MOF…

Profiles

A language can become extensible by introducing the profile notion in its metamodel.
- So do UML and MOF…
Package merge

- Some kind of inheritance between packages
  - Multiple «merge» possible
    - Matching elements found by name
  - Ext::Element implicitly exists
Package merge

- Some kind of inheritance between packages
  - `Ext::Element` has a name attribute and is owned by an `Ext::Package`
Package merge

- MOF meta-metamodel designed for reuse
Improving package merge

Name matching a bit naïve…

- Model composition:
  Complex composition directives
    • Kompose

- Model comparison / matching:
  Comparing independent metamodels
    • Versions of the same metamodel
    • Expected vs. obtained metamodel
    • Identification of “matcheable” elements
    • Federated models
    • EMF Compare
Plan

- Tools around models
- Supporting / modeling languages
  - Abstract syntax
  - Concrete syntax
  - Transformation
- Language models reuse
  - Reusing abstract syntax
  - Reusing tools
Tuning transformations

Metamodel A

Transformation

Metamodel B

Tuned Metamodel A

«Weaves In»

«model transformation aspect»

Tuned Transformation
Tuning transformations

Higher-order transformations:
Transformations that transforms transformations!
Tuning transformations

An aspect language for transformation

Aspect ?

- Extension of a source language
  - E.g.: AspectJ is an extension of Java

- Directives to change the extended language
  - E.g.: At these places, that must be inserted.
Tuning transformations

An aspect language for transformation
Tuning transformations

An aspect language for transformation

```
Aspect Model Transformation Language

«extends»

Model Transformation Language

«conformsTo»

Transformation

«conformsTo»

Aspect Weaver
```
Conclusion

• DSLs
  • Few concepts
  • Limited set of users
  • Tool support is a key factor
  • Few resources

• Languages
  • Abstract syntax, concrete syntax, translation
  • Reuse

• DSLs for DSLs
  • Languages to express languages
    • Various approaches
    • Tool support by generation or interpretation
Conclusion

Problems

- Abstractions for languages
  - Syntax, semantics, reuse…
- Quality
- Maintainability
- Scaling
- Optimization
- ...
- Many introduced in here in Oslo
Conclusion

A modeling language deserves to be supported.

Price per model realized

Number of models

No tool support

Dedicated tool support made easy
Conclusion

Dedicated tool support made easy with many concepts beautiful well checked generating many languages high degree of optimization different concrete syntaxes easy to learn able to solve many problems
Thank you!

More or less questions than before lunch?