Model-driven Compiler Construction

Prof. Andreas Prinz

Meta-Introduction, Meta-Languages
DSL, Compilers, Modelling
Examples
Meta-models vs. Grammars
Summary
Meta-Introduction A

• Say your name
• Go to the right side
• Explain the sequence-rule
• Explain the skip-rule
• Go to the left side
• Explain the meaning of life
• Select one participant
• Pass the word to the selected person, Switch to the next slide and Sit down on your seat
Meta-Introduction B

• Say your name
• Go to the right side
• Draw a circle on the blackboard
• Explain the term “Meta”
• Explain
• Go to the left side
• Extrapolate positively
• Select one participant
• Pass the word to the selected person, Switch to the next slide and Sit down on your seat
Meta-Introduction C

• *Say* your name
• *Read* the following text aloud
  • The term “meta” means *transcending* or *above*.
  • In our context, “meta” can be replaced by the following phrases:
    • is a description of
    • is a model of
    • is an abstraction of
• *Select* Prof. Andreas Prinz
• *Pass* the word to the selected person, *Switch* to the next slide and *Sit down* on your seat
Meta-lecture

• description of a lecture
• if it is
  • formal (formulated in a formal language)
  • complete (on some level of abstraction)
  • executable (has semantics)
• then we can execute it (on a computer)
Meta-language

- description of a language
- if it is
  - formal (formulated in a formal language)
  - complete (on some level of abstraction)
  - executable (has semantics)
- then we can execute it (on a computer)
What describes a language?

- **Language structure**
  - construct: concepts and their relations
  - restrict: conditions, constraints
- **Presentation**
  - textual: text that presents that structure
  - graphical: graphics for the structure
- **Semantics (Meaning)**
  - transform: translate to another language
  - execute: run the statements
Aspects of a language & tools

- Build a tool from this info
- Idea: The meta-model IS the tool.
Language tools: compilers

- Graphical Editor
- Parser
- Text editor
- Static checks
- Internal format
- Exchange Format (XMI, ASN.1)
- Test case derivation
- Code generation
- Simulation

- Solved: many input/output formats
- Graphical / Domain specific languages, many transformations
- Platform dependent code generation
- Combination of tools
- Internal format based on: abstract syntax, meta-model, MOF-structure
Why to describe Languages?

- graphical languages / combined languages
- domain specific languages
  - small languages
  - higher abstraction levels – use of models
  - fast production of compilers
- Needs good language design!
- Less focus on optimization because of high-level output languages
A rise in productivity is overdue

- "The entire history of software engineering is that of the rise in levels of abstraction"

- New general-purpose programming languages have not increased productivity

- Abstraction of development can be raised above current level...

- ... without losing control or accepting substandard results

*Software Productivity Research & Capers Jones, 2002
Productivity increase from DSM

Comparing to earlier practice (typically compared to coding)

Percent Increase

- Embedded UI applications: 500%
- Mobile phone software: 1000%
- Phone switch features: 750%
- Call processing services: 600%
- Heart rate monitor: 900%
- J2EE web application: 500%
- Home automation: 600%

© JPT/2011 MetaCase
Models and systems

• A model is an abstraction of a (part of a) system.
  • one model describes several systems, one system can have several models
  • simplified view of a system with respect to criteria
  • needs a representation, e.g. using a language

• Models on different abstraction levels: Modelling language, Programming Language, Assembler, Machine code, Bits, Electricity, Atoms, ...

• Meta-model = high-level description of a language
  • narrow view: concepts of the language
  • wider view: all important aspects of the language, i.e. concepts, presentation, static and dynamic semantics

• Language descriptions use also DSLs and have aspects.
A meta-modelling architecture

<table>
<thead>
<tr>
<th>OMG Level</th>
<th>Examples</th>
<th>Grammar example</th>
<th>OCL example</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 = meta meta model</td>
<td>MOF</td>
<td>EBNF</td>
<td>MOF</td>
</tr>
<tr>
<td>2 = meta model</td>
<td>UML MM</td>
<td>Java grammar</td>
<td>OCL language</td>
</tr>
<tr>
<td>1 = model</td>
<td>UML Model</td>
<td>a program</td>
<td>a formula</td>
</tr>
<tr>
<td>0 = instances</td>
<td>real objects</td>
<td>A run</td>
<td>a truth value</td>
</tr>
</tbody>
</table>
Meta-Languages in MDA and Eclipse

MDA

- Structure
- Constraints
- OCL
- Semantics
- execute
- action
- transform
- QVT

Presentation
- graphical
- HUGN
- textual
- HUTN

eclipse (oaw)

- Structure
- Constraints
- OCL
- Semantics
- execute
- Java
- transform
- xtend/xpand

Presentation
- graphical
- GEF/GMF
- textual
- xtext

Prepared by Andreas Prinz
Simple sample structure (EMF)

- Sudoku
- dimension
- Field
- 1..*
- Column
- Row
- Box
- cells
- cellsB
- cellsR
- cellsC
- cellValue
- field
- column
- row
- box
Simple sample constraints (OCL)

context Field inv uniqueICellValues:
self.cells->forAll(c1,c2 : Cell | c1<>c2 implies c1.iCellValue <> c2.iCellValue)

context Cell inv rowFromCell:
self.row -> size()=1

context Puzzle inv numberOfBoxes:
self.Elements->select(f : Field | f.oclIsTypeOf(Box)) -> size()=9
Simple sample text syntax (xtext)

grammar my.pack.Sudoku
   with org.eclipse.xtext.common.Terminals
generate sudoku "http://www.eclipse.org/sudoku"

Puzzle :
   `puzzle` dimension=INT `;` Row+

Row :
   `row` `(` (Cell `,`)+ Cell `)` ;

Cell :
   `cellValue` = INT;
Simple sample graphics

Puzzle → Diagram

Row → Container

Cell → Rectangle

rows subsets diaContents

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

cells subsets contents

Rectangle → [5]
Simple sample transformation

transformation swap1and6 (source, target: Sudoku) {
    source Cell { cellValue = 1 }
    -> target Cell { cellValue = 6 };
    source Cell { cellValue = 6 }
    -> target Cell { cellValue = 1 };
    source Cell { cellValue = value }
    -> target Cell { cellValue = value };
    when{ cellValue <> 1 and cellValue <> 6; }
}

• *declarative versus operational*
Simple sample execution

Run(s:Sudoku) =  
   def 
      forall f in self.field do RunF(f)

Runf(f:Field) =  
   def 
      choose c in self.cell with c.value=null  
      and c.possible.size = 1  
      choose v in c.possible do c.value:= v  
      choose c in self.cell with c.value<>null  
      forall cc in self.cell do  
      delete c.value from cc.possible
# Problem area execution

<table>
<thead>
<tr>
<th>Meta-model</th>
<th>Syntax</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell</td>
<td>-</td>
<td>RTCell</td>
</tr>
<tr>
<td>X:Cell</td>
<td>A: RTCell</td>
<td>B: RTCell</td>
</tr>
</tbody>
</table>

e.g. history, possibilities
Problem area Presentation

- There are usually several representations for the same meta-model instances.
- Tools and theory exist only for the case 1:1.
- A representation is a separate model that is related to the meta-model.

```
signal Sig1, Sig2;
```

```
Sig1: SignalDefinition
Sig2: SignalDefinition
```

```
signal Sig1;
signal Sig2;
signal Sig1;
```

```
signal Sig2;
```
Semantic Analysis (text2as)

- Transformation from concrete syntax to abstract syntax: connect definitions with uses
  - flow-of-control checks, e.g. join/break labels
  - name-related checks, e.g. begin/end construct names

- Mapping patterns (syntax:semantics)
  - Direct mapping (1:1) – direct match
  - Merge mapping (1:n) – shorthand notations, e.g. int a,b;
  - Partial description mapping (n:1) – several descriptions of the same thing
Meta-models versus grammars

• Advantages of grammars
  • Strong mathematical basis
  • Tree-based
  • Trees can be extended into general graphs
  • Several advanced tools available
  • Easily understandable

• Advantages of meta-models
  • Direct representation of graphs (graphics!)
  • Namespaces and relations between language elements (in particular for language transformations and combinations)
  • Object-oriented definition of oo languages
  • More problem-oriented
  • Reuse and inheritance
  • Tools allow direct handling of models (repositories)
  • Structuring possible (e.g. packages)
Grammars → meta-models

1. Every symbol is represented with a class.
2. A rule with a single symbol on the rhs is represented with an association between the class representing the lhs and the rhs.
3. A rule with a composition on the rhs is represented with an association for every sub-expression.
4. A rule with an alternative on the rhs is represented with a generalization for every sub-expression.
5. A sub-expression consisting of just one symbol is represented with the symbol’s class.
6. A sub-expression being a composition or an alternative is represented with a new class with new name. The composition is then handled like a rule.
Using the transformation for SDL

- Introduction of abstract concepts
  - General: namespace, namedElement, typedElement
  - Specific: parametrizedElement, bodiedElement
- Introduction of relations
  - Procedure name versus procedure definition
- Deletion of grammar artefacts
  - Referencing: identifier, qualifier
  - Names in general
  - Superfluous structuring
Summary

• Future compilers based on language descriptions.
  • A description of something can be executed on a computer if it is formal, complete and executable.
  • describe languages instead of compiler writing
  • need also agreement (standard)
  • definition of good languages is difficult: use patterns

• A formal language description includes three aspects: structure, syntax, semantics

• A formal language description allows tool generation on a computer.
  • Model access & exchange, front-end and back-end
  • Easy exchange of representation or several of them
  • Combination of tools handling the language
  • Description of relations between languages

• This leads to model-driven compiler technology.
Importance of DSL (abstract syntax)

- Presentation (html web page, xml check descr.)
- Model (web page model, checking model)
- Checking tools (access. checker, type, consistency)
- Transformation tool (checks compiler)
- Generated tools (python code, access. checker, code model)