Meta-models and Grammars

Prof. Andreas Prinz

Introduction, Compilers
Modelling & Meta-modelling
Examples
Meta-models vs. Grammars
Summary
Compilers

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

• Solved: many input/output formats
• Graphical / Domain specific languages, many transformations
• Internal representation: Meta-model vs. Abstract syntax

Prepared by Andreas Prinz
Importance of internal structure

- Representation
  - (html web page)
  - (xml check descr.)

- Model
  - (web page model)
  - (checking model)

- Tools
  - (access. checker)
  - checking
  - (type, consistency)

- Code
  - generated code
    - (python code)

- Model
  - generated Model
    - (code model)

- Transformation
  - transformation tool
    - (checks compiler)

- Generated
  - generated tool
    - (access. checker)
Aspects of Compilers/Languages

• Language structure: What are the concepts? How are they related?
• Static semantics: additional conditions, what is allowed?
• Representation: How are programs written? -> graphical vs. textual
• Dynamic semantics: What do the programs mean? How to generate code for them?
Aspects of a language & tools

- Graphical editor
- Textual editor
- Parser
- Access interface
- Repository
- Simulator
- Transformator
- Exchange format

- XMF Mosaic from Xactium as example tool set.
- Idea: The meta-model IS the tool.
Aspects for SDL and UML

SDL
- Structure: informal EBNF, formal ASM
- Behaviour: informal textual, formal EBNF
- Constraints: formal PC1, textual transform

UML
- Structure: informal textual, formal meta-model
- Behaviour: informal textual, formal EBNF
- Constraints: formal OCL, transform

Prepared by Andreas Prinz
What is a Model?

- 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
  - can answer questions about the system if related to the view
  - needs a representation, e.g. using a modelling language
- Models on different abstraction levels
  - Models of the real Bits: Assembler
  - Models of data storage: Database descriptions
  - Models of access: Interface languages
- What is the best model of a cat? → It is a cat. But it has to be the same cat!
- A model has aspects like a language.
What are Meta-Models?

- A description of a class of models
- Models / high-level descriptions of the modelling language
  - narrow view: structure of the modelling language
  - wider view: all important aspects of the language, i.e. structure, presentation, static and dynamic semantics
- Meta-models (languages) can have several aspects.
Language support MDA and Eclipse

MDA

Eclipse (oaw)

HUGN
HUTN
Constraint
OCL

Structure
MOF

Action
Behaviour

QVT

GEF/GMF

Xtext

Constraints
OCL’

Xpand

Xtend

Behaviour

---
# 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>

**Diagram:**

- **OMG Level:**
  - 3 = meta meta model
  - 2 = meta model
  - 1 = model
  - 0 = instances

- **Examples:**
  - MOF
  - UML MM
  - UML Model
  - real objects

- **Grammar example:**
  - EBNF
  - Java grammar
  - a program

- **OCL example:**
  - MOF
  - OCL language
  - a formula
  - a truth value

Prepared by Andreas Prinz
Instances on several levels

```
Class
  name : String
  isAbstract: Boolean

Property
  name : String
  owned-Attribute

Class
  name = "Class"
  isAbstract = false

Class
  name = "Class"
  isAbstract = false

Person
  :Property
    name = "isAbstract"

Person
  :Property
    name = "name"
```

Prepared by Andreas Prinz
Simple sample structure (EMF)
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 (TEF)

syntax toplevel PuzzleTpl, ecorepath "..." { 
  element CellTpl for Cell{ single for iCellValue, with INTEGER; } 
  element RowTpl for Row{ 
    "Row"; "("; 
    sequence for cellsInRow, with @CellTpl, seperator ",", last false; 
    ");"; 
  } 
  element PuzzleTpl for Puzzle{ 
    "Puzzle"; "("; single for iDimension, with INTEGER; ");"; "="; 
    sequence for Elements, with @FieldTpl, seperator ",", last false; 
  } 
  choice FieldTpl for Field{ @RowTpl } 
}
Simple sample graphics

Sudoku → Diagram

Row

Cell → Rectangle

Prepared by Andreas Prinz
Simple sample transformation (QVT)

class transformation theOne (source : sudoku, target: sudoku){
    top relation change1to16 {
        checkonly domain source sudoku:Cell { iCellValue = 1 }; enforce domain target sudoku:Cell { iCellValue = 16 };}
    top relation change6to11 {
        checkonly domain source newstructure:Cell { iCellValue = 6 }; enforce domain target newstructure:Cell { iCellValue = 11 };}
    top relation nochange { value: Integer;
        checkonly domain source newstructure:Cell { iCellValue = value }; enforce domain target newstructure:Cell { iCellValue = value }; when{ iCellValue <> 1 or iCellValue <> 6; } }
}
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></th>
<th>Syntax</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Meta-model</strong></td>
<td>Cell → RTCell</td>
<td></td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>X:Cell</td>
<td>A: RTCell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B: RTCell</td>
</tr>
</tbody>
</table>

-e.g. history
Problem area “representation”

• 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.
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
Conclusions / Summary

• Future languages will be defined using meta-models.
  • definition of good meta-models is difficult
  • need also agreement (standard)
  • patterns for good models needed, maybe joint concepts

• Meta-models / Languages have several aspects: structure, syntax, static and dynamic semantics

• Meta-model language definitions allow tool generation
  • Direct access to the models
  • Easy exchange of representation or several of them
  • Combination of tools handling the language
  • Description of relations between languages

• Meta-models are the models to be used in model-driven compiler technology.

Prepared by Andreas Prinz