Meta-models and Grammars
Prof. Andreas Prinz

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

- graphical languages / combined languages
- fast production of compilers:
  - domain specific languages
  - small languages
- platform dependent code generation
- combination of tools
- language design!

- but also: less focus on optimization because of high-level output languages
Solution 1: abstract syntax

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

- Solved: many input/output formats
- Graphical / Domain specific languages, many transformations
- Internal format based on: abstract syntax, metamodel, MOF-structure
Importance of abstract syntax

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

- Model
  (checking model)

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

- transformation tool
  (checks compiler)

- generated tool
  (access. checker)

- generated
  code
  (python code)

- generated
  Model
  (code model)

Prepared by Andreas Prinz
Problem speed / many languages

• Why do we need many languages?
  • Higher abstraction levels – use of models
• 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, ...
Solution: Language Description

• Do not write compilers, but describe languages
• 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
• Meta-models (language descriptions) are also languages and have aspects.
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
- Checker
- Transformer
- Access interface
- Repository
- Exchange format
- Structure
- Execution
- Semantics
- Transform

- Build a tool from this info
- Idea: The meta-model IS the tool.
Aspects for SDL and UML

SDL

- Structure
  - formal EBNF
- Semantics
  - formal ASM
- Execution
  - transform
- Representation
  - informal EBNF
  - textual

UML

- Structure
  - formal MOF
- Semantics
  - informal text
  - formal OCL
- Execution
  - transform
- Representation
  - informal text
  - textual

Prepared by Andreas Prinz
Language support in MDA and Eclipse

MDA

- Structure
  - MOF
- Semantics
  - OCL
- Constraints
  - OCL
- Representation
  - graphical
    - HUHN
  - textual
    - HUTN

Eclipse (oaw)

- Structure
  - EMF
- Semantics
  - transform
    - QVT
- Constraints
  - OCL
- Representation
  - textual
    - xtext
  - graphical
    - GEF/GMF

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, separator ",", last false;
    ")";
  }
  element PuzzleTpl for Puzzle{
    "Puzzle"; "("; single for iDimension, with INTEGER; ")"; ";"; 
    sequence for Elements, with @FieldTpl, separator ",", last false;
  }
  choice FieldTpl for Field{ @RowTpl }
}

Prepared by Andreas Prinz
Simple sample graphics

Sudoku → Diagram

Row → Container

Cell → Rectangle

rows subsets diaContents

cells subsets contents
Simple sample transformation (QVT)

transformation theOne (source : sudoku, target: sudoku){
    top relation change1to16 {
        checkonly domain source sudoku:Cell { iCellValue = 1 }; enfore domain target sudoku:Cell { iCellValue = 16 };}
    top relation change6to11 {
        checkonly domain source newstructure:Cell { iCellValue = 6 }; enfore domain target newstructure:Cell { iCellValue = 11 };}
    top relation nochange { value: Integer;
        checkonly domain source newstructure:Cell { iCellValue = value }; enfore domain target newstructure:Cell { iCellValue = value }; when{ iCellValue <> 1 or iCellValue <> 6; }
    }
}

Prepared by Andreas Prinz
Simple sample execution

\[
\text{Run}(s: \text{Sudoku}) = \text{def}
\]
\[
\text{forall } f \text{ in self.field do RunF}(f)
\]

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

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

- **Model**
  - X:Cell
- **Meta-model**
  - Cell
  - RTCell

Note: e.g. history, possibilities
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 \(\rightarrow\) 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

Prepared by Andreas Prinz
Conclusions / Summary

- Future language definitions based on 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 + constraints, syntax, semantics
- Formal 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

- This leads to model-driven compiler technology.
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 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>
Instances on several levels

Class
- name : String
- isAbstract : Boolean
  name = "Class"
  isAbstract = false

Property
- name : String
- owned-Attribute

Class
- name : String
- isAbstract : Boolean
  name = "Class"
  isAbstract = false

Person
- :Property
  name = "isAbstract"
  name = "name"

Prepared by Andreas Prinz