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
Importance of internal structure
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

- Structure
  - graphical
  - textual

- Constraints
  - run
  - transform

- Behaviour

- Representation
  - Graphical editor
  - Textual editor

- Access interface
  - Parser
  - Repository

- Checker

- Simulator
- Transformator
- Exchange format

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

SDL

- Structure
  - formal EBNF
- Constraints
  - formal PC1
- Behaviour
  - informal textual
  - run

UML

- Structure
  - formal meta-model
- Constraints
  - formal OCL
- Behaviour
  - informal textual
  - run

- Graphical Representation
- Textual Representation
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
Representation
HUTN

Structure
MOF

Constraints
OCL

Action
Behaviour
QVT

GEF/GMF
Representation
Xtext

Structure
EMF

Constraints
OCL'

---

Xtend
Xpand
## 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
- **«component» MOF** (M3)
- **«component» UML Metamodel** (M2)
- **«component» UML Model** (M1)
- **«component» User Data** (M0)
Instances on several levels

Slide 10
Simple sample structure

Grammar

ProductionRule 1..*

NonTerminal

left

Terminal

Alternative

Term 1..* right

TermSequence {ordered} 1..*

Optionality

Repetition

canBeEmpty: Boolean
Simple sample constraints

context = Repetition
inv not self.canBeEmpty implies self.exists(TermSequence)

context = NonTerminal
inv theGrammar.ProductionRule.exists (p|p.name = self.name)
Simple sample text syntax

Grammar : {rules=ProductionRule}*;
ProductionRule : left:NonTerminal "::="
  right:TermSequence ";";
NonTerminal : name=ID;
TermSequence : {term:Term}*
Term : NonTerminal | Terminal | Optionality
Terminal : name=ID;
Optionality : "[" opt:TermSequence "]"
Simple sample graphics

ProductionRule.contents = left - (::=) -> right
Simple sample transformation

removeAlternatives:
ProductionRule(nt, Alternative(set a))
--> set ProductionRule(nt, a)

removeOptional:
Optional(x)
--> Alternative({x,Nothing})
Simple sample execution

Run(a:NonTerminal) = def
  case a.rule of
    NonTerminal: Run(a.rule)
    Terminal: Print(a.rule.value)
  Repetition:
    choose n:Natural
    foreach x:1..n Run(a.rule.sequence)
  Optional:
    choose b:Boolean
    if b then Run(a.rule.term) else Skip
  TermSequence:
    foreach n:1..length(a.rule) Run(a.rule[i])
Problem area “runtime”

- Meta-model
- Model
- Running Model
- VM Model

- MM Code
- Code
- Running Code
- VM Code

Interpretation

Compilation

- Instances of the language: code
- Instances of the program: data / objects
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.