A Comparison of Tool Support for Textual Domain-Specific Languages

Michael Pfeiffer and Josef Pichler

Dr. Josef Pichler
+43 7236 3343 867
josef.pichler@scch.at
www.scch.at
Motivation

- Domain-specific languages are languages tailored to a specific application domain
- Notation (textual, graphical, tabular) depends on the application domain
- Focus on textual languages following source-to-source transformation; host languages are Java or C#
- Successful application of DSL depends on provided tool support
- Tool support by *language workbenches* (Fowler)
  - oAW, MPS, MontiCore, IMP, TCS, TEF, CodeWorker, ...
- Feature model for expressing variations on DSL and DSL tools in general (not only textual languages) (Langlois et. al 2007)
- Reuse of an existing criteria catalog facilitates comparison of results
Overview of Tools

- openArchitectureWare (oAW, 4.3)
  - Eclipse, modular MDA/MDD generator framework, xText
- Meta Programming System (MPS, early access)
  - JetBrains
- MontiCore (1.1.5)
  - Academic, TU Braunschweig
- IDE Meta-Tooling Platform (IMP, 0.1.74)
  - Eclipse
- Textual Concrete Syntax (TCS, 0.0.1)
  - Eclipse, very similar to oAW
- Textual Editing Framework (TEF, 1.0.3)
  - Academic, Humboldt-Universität zu Berlin, very similar to oAW
- CodeWorker (3.5)
  - parsing tool and source code generator, no editor
Example (finite state machine) implemented with 4 tools

Grammar productions for textual FSM

FSM = "inputAlphabet" string "outputAlphabet" string {State}.
State = ["start"] "state" id {Transition}.
Transition = "transition" char ["/" char] "->" id.

Example: determine if a binary number has an odd or even number of zero digits

inputAlphabet "01" // Digits of a binary number
outputAlphabet "eo" // Char 'e' for even and 'o' for odd

start state Even
transition 0 / o -> Odd
transition 1 / e -> Even
state Odd
transition 0 / e -> Even
transition 1 / o -> Odd
openArchitectureWare (xText)

- Open source project
- Model Driven Development
- Eclipse, EMF
- Subcomponent xText for textual DSL

- Grammar defines ASG in form of dynamic EMF model
- Validation language
- Template language for xPand
openArchitectureWare (xText)

Full featured editors
- syntax highlighting
- code completion
- folding, error decoration
- JetBrains (IntelliJ), no release yet, early access program
- Cell-based editing model (no free text!)
- Powerful but complex

- Abstract syntax is specified by concepts (instead of production rules)
- All concepts are the structure of the language

- Concrete syntax is static text (editor layout) for every concept (AS)
Meta Programming System$_2$

Aggregation of concepts

Static text

Editable cell

```
concept declaration FSM

extends: BaseConcept

implements
INamedConcept

is root: true

properties:
inputAlphabet : string
outputAlphabet : string

children:
State, state 0..n spe
```

```
inputAlphabet "01"
outputAlphabet "0e"

state Even
start true
transition 0 / o -> Odd
transition 1 / e -> Even
transition <no input> / <no output> -> <no state>

state Odd
start false
transition 0 / e -> Even
transition 1 / o -> Odd
```
- Academic project, TU Braunschweig
- Eclipse based

- Grammar for concrete syntax and abstract syntax (similar to input format of ANTLR)

- Generated AST
- Generated compiler-frontend (ANTLR)

- Transformation
  - Visitors on AST
  - Template engine
MontiCore\textsubscript{2}

Editors

- syntax highlighting
- folding
- error decoration
IDE Meta-Tooling Platform

- Open source, IBM Watson Research
- Eclipse
- Wizard to generate code skeletons for a large range of IDE features

- Grammar for concrete syntax
- LPG generates scanner, parser, and data structures for AST

- Transformation by visitors, no template engine
IDE Meta-Tooling Platform

Editors

- syntax highlighting
- folding, formatting
- code completion
Criteria for our comparison are a subset derived from feature model Groups: language (LA), transformation (TR), and tools (TO)

Language (abstract and concrete syntax)
- LA-AS1. abstract syntax tree or abstract syntax graph
- LA-AS2. grammar or meta-model
- LA-AS3. can be composed

- LA-CS1. technique to map abstract syntax to concrete syntax
- LA-CS2. representation (text, graphic, ...) for the concrete syntax
- LA-CS3. declarative or imperative style
<table>
<thead>
<tr>
<th></th>
<th>oAW</th>
<th>MontiCore</th>
<th>MPS</th>
<th>IMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA-AS1</td>
<td>Graph</td>
<td>Tree</td>
<td>Graph</td>
<td>Tree</td>
</tr>
<tr>
<td>LA-AS2</td>
<td>Meta-model (ECore)</td>
<td>Java Classes</td>
<td>Meta-model</td>
<td>Java Classes</td>
</tr>
<tr>
<td>LA-AS3</td>
<td>Composition</td>
<td>Composition</td>
<td>Composition</td>
<td>No built-in support</td>
</tr>
<tr>
<td>LA-CS1</td>
<td>Explicit. (1)</td>
<td>Explicit. (1)</td>
<td>Explicit. (2)</td>
<td>Implicit. (3)</td>
</tr>
<tr>
<td>LA-CS2</td>
<td>Text</td>
<td>Text</td>
<td>Text/XML</td>
<td>Text</td>
</tr>
</tbody>
</table>

- Abstract syntax: graph and trees; generated Java classes or models
- Mapping between abstract syntax and concrete syntax
  - Def. of concrete syntax is mixed with def. of abstract syntax (1)
  - Explicit definition of editor layout (CS) for each AS elements (2)
  - Implicit definition of abstract syntax by concrete syntax (3)
- Textual representation (XML only in MPS)
Transformation (target asset and operational translation)
- TR-TA1. target asset (model, text, graphic, binary)
- TR-TA2. destructive or incremental update
- TR-TA3. kind of support for integration of target assets

- TR-OT1. transformation techniques (M→M, M→T, T→T, T→M)
- TR-OT2. transformation by compilation or interpretation
- TR-OT3. internal or external environment for transformation
- TR-OT4. implicit or explicit scheduling
- TR-OT5. internal or external location
- TR-OT6. automation level (manual or automated)
Comparison - Transformation

<table>
<thead>
<tr>
<th></th>
<th>oAW</th>
<th>MontiCore</th>
<th>MPS</th>
<th>IMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR-TA1</td>
<td>Text</td>
<td>Text</td>
<td>Model or Text</td>
<td>No built-in support.</td>
</tr>
<tr>
<td>TR-TA2</td>
<td>Destructive</td>
<td>Destructive</td>
<td>Destructive</td>
<td>No built-in support</td>
</tr>
<tr>
<td>TR-TA3</td>
<td>No support for integration with target assets available.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Different levels of tool support for generating target assets (TA)
- Template engines for model-to-text transformation
  - outstanding (oAW), rudimentary (MontiCore)
  - MPS requires generation of target model that is transformed to text
- All tools (except IMP) provide destructive update (overwrites target assets)
- No support for integration of generated target assets
### Comparison - Transformation

<table>
<thead>
<tr>
<th></th>
<th>oAW</th>
<th>MontiCore</th>
<th>MPS</th>
<th>IMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR-OT1</td>
<td>M2T (xPand)</td>
<td>M2M (visitor)</td>
<td>M2M (generator)</td>
<td>No built-in support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M2T (visitor/template engine)</td>
<td>M2T (indirect)</td>
<td></td>
</tr>
<tr>
<td>TR-OT2</td>
<td>Interpretation. Templates filled at runtime.</td>
<td></td>
<td></td>
<td>Compilation.</td>
</tr>
<tr>
<td>TR-OT5</td>
<td>Internal. Runs in runtime workbench (Eclipse) or in MPS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Concerning operational translation, tools are very similar
Tool (respect of abstraction, assistance)
- TO-RA1. respect of abstraction (intrusive or seamless)
- TO-AS1. kind of assistance (static or adaptive)
- TO-AS2. process guidance (step or workflow)
- TO-AS3. checking (completeness or consistency)

Omitted
- Quality factors
- Process features
### Comparison - Tool

<table>
<thead>
<tr>
<th></th>
<th>oAW</th>
<th>MontiCore</th>
<th>MPS</th>
<th>IMP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TO-AS2</strong></td>
<td>Neither step nor workflow process guidance is supported.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- High variance ranging from
  - plain text editor for CS definition (IMP)
  - to an editor with syntax coloring (MontiCore)
  - code completion and validation while typing (oAW and MPS)
Conclusion

- Language workbenches are driven by existing IDE
  - JetBrains (IntelliJ) for MPS
  - Eclipse Platform (JDT) for all others
- Eclipse technology (platform, EMF, ...) predominant for textual DSLs
- Free text editing including features like code completion but missing refactoring, searching, references, ...
- MPS with unique cell based editing model
- Reuse of compiler generators (ANTLR, LGP) to generate scanner/parser for text-to-model transformation
- Template engine or visitor pattern for model-to-text