Graphical Concrete Syntax Design for Domain-specific Languages

Dániel Varró, Zoltán Ujhelyi
Model Driven Software Development
Lecture 7
Structure of DSMs

Graphical syntax

Abstract syntax

Well-formedness constraints

Behavioural semantics, simulation

Code generation

Mapping

Textual syntax

View

Code (documentation, configuration)
DSM aspects

- Abstract syntax
- Concrete syntax
- Views, translations, mappings
- Behavioural (dynamic) semantics
- Well-formedness constraints
Concrete Syntax Design

- User-managed parts of a modeling language
  - Performance
  - Robustness
  - Usability issues

- Creating model editors
  - Similar problems at programming languages
  - IDE extensions needed
Approaches

- **Textual syntax**
  - Character-based edit operations
  - Abstract syntax: traditional AST

- **Graphical syntax**
  - Editing operations: translated to abstract syntax
  - Abstract syntax: based on metamodel

- **Form-based entry**
  - Less common
  - Behaves similar to graphical syntax
Advanced features

High level editing support

• Outline view
• Documentation display (e.g. Javadoc)
• Templates/snippets/examples
• Content assist
• Validation, automatic fixes

Project-level integration

• Code generation
• Wizards to create projects/files
• Integration with manually written code in GPL
Technology

- **Eclipse Modeling Tools**
  - Several related subprojects
  - Each supports a single aspect
  - Examples of today

- **Microsoft Visual Studio 2010 Visualization & Modeling SDK**
  - DSL modeling framework from Microsoft
  - Own metamodeling core
  - Focuses on graphical modeling
Graphical Editors
Graphical Editor Techniques

- EuGENia
- Zest
- GEF3D
- GEF
  - Draw2D
- GMF
- Sirius
- Spray
- Graphiti
- EMF
  - EMF.Edit
Graphical Modeling

- **Model**
  - Typically graph-based modeling (Edges, Nodes)
  - In our case EMF

- **Idea**
  - Display and editing as a graph model
Example: Petri net editor

Tree-based outline view
Example: Social Network editor

- Project Explorer
- Outline view
- Graphical view
- Properties view
- Extensions
Implementation

- **Presentation**
  - Based on a Canvas
  - Using vector-graphic libraries (GEF/Draw2d)

- **Model manipulation**
  - EMF Edit model manipulation commands
    - Atomic operations: create/modify/remove node/edge
  - Transactional modifications
    - Undo/redo support
    - Replayability
Implementation 2.

- **View models**
  - Modeling for view-specific information
    - Coordinates
    - Size
    - Colors and fonts
    - ...
  - Generic implementation in GMF and Graphiti
  - Often stored in external files
    - Separation of concerns
    - E.g. code generator not interested in view information
Technologies 1. - GEF

- Graphical Editing Framework (GEF)
  - “Low level” editor framework
  - Not EMF-specific
- Model-View-Controller approach
- Generic graph-based editor framework
  - Including undo/redo support
  - Graphical outlines
- Manual coding for every possible element
- GEF4 FX – JavaFX-based replacement of the core
Technologies 2. – GMF

- Graphical Modeling Framework
- Based on GEF and EMF
- Well-separated view and domain models
  - Generic view model
  - Synchronization provided by GMF framework
- Relatively old technology
  - Widely used
  - Very complex to start
Technologies 2. – GMF

- Model-driven development environment
  - Common model for graphical editors, using
    - Figure definition model
      - Basic symbol definition of the graphical language
    - Tooling model
      - Defining model manipulation commands
    - Mapping model
      - Mapping figures and tools to domain model
  - Fully functional editor can be generated
    - Problematic manual modifications
- Or a high-level editor framework
  - Manual coding
Technologies 3. - Graphiti

- Newer high level graphical editor framework
  - Based on EMF and GEF
  - But: different approach then GMF
    - Simplified programmatic API
    - Manual coding
  - Idea
    - All Graphiti based editors should
      - Look similar
      - Behave similar
Technologies 3. - Graphiti

- Development methodology
  - Coding over a high-level Java framework
    - Much simpler than GMF
    - Repetitive code needed

- Spray project
  - Textual modeling environment for graphical editors
  - Generates code over the Graphiti framework
Technologies 4. - Sirius

- New modeling project
  - Since 2013 on eclipse.org
  - Previously Obeo Designer – commercial tool

- How stable is it?
  - Old projects are to be migrated
  - Version history
    - 0.9: 2013-12-10
    - 1.0: 2014-06-25 (Kepler release train)
    - 2.0: 2014-10-24, (Luna release train)
    - 3.0: 2015-06-25, planned (Mars release train)
Sirius Viewpoints

- Base concept:
  - Every diagram is a view of the model
  - With a defined syntax
    - Graphical
    - Table/Tree syntax
    - Xtext-based textual syntax

- Viewpoint definition
  - Viewpoint specification model
Viewpoint Specification Model

- Viewpoint
- Feature Provider registration
- Mappings
- Creation tools
Node & Edge Mapping

- **Domain class**: petrinet.Transition
- **Filter settings**: feature:transitions
- **Edge class**: petrinet.TPArc
- **Source features**: feature:source
- **Target features**: feature:target
Feature Selection

- Interpreted expressions
  - Special interpreters
    - **var**: accessing specification model variables
    - **feature**: accessing EMF model features
    - **service**: accessing service methods
  - Acceleo
    - Acceleo expressions
      - Basic operations
      - Comparison with single ‘=' symbols
    - Syntax: `[theExpression/]`
  - Raw OCL
    - Not recommended, Acceleo provides superset features
  - Custom interpreter
Node & Edge Tool

Tool parameter variables

Model creation sequence

Different variables

More complex creation steps

Section createTools
  - Container Creation createPlace
    - Node Creation Variable container
    - Container View Variable containerView
  - Begin
    - Change Context var:container
    - Create Instance petrinet.Place
      - Set name

Edge Creation createArc
  - Source Edge Creation Variable source
  - Target Edge Creation Variable target
  - Source Edge View Creation Variable sourceView
  - Target Edge View Creation Variable targetView
  - Begin
    - Switch
      - Case [source.eClass().name = 'Transition']
        - Create Instance TPArc
          - Set source
          - Set target
        - Case [source.eClass().name = 'Place']
Interpreted Modeler Development

Viewpoint specification

View model using the interpreted specification
<table>
<thead>
<tr>
<th></th>
<th>GEF</th>
<th>GMF</th>
<th>Graphiti</th>
<th>Sirius</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Arbitrary</td>
<td>EMF</td>
<td>EMF</td>
<td>EMF</td>
</tr>
<tr>
<td>Non graph-based</td>
<td>Manageable</td>
<td>Large amount of</td>
<td>Not supported</td>
<td>Tree, Table</td>
</tr>
<tr>
<td>presentation</td>
<td></td>
<td>customization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>needed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code size</td>
<td>Large, repetitive code</td>
<td>Mostly modeling, some coding</td>
<td>Smaller amount, but repetitive code</td>
<td>Negligible</td>
</tr>
<tr>
<td>Development</td>
<td>Only coding</td>
<td>Modeling and</td>
<td>Coding</td>
<td>Modeling</td>
</tr>
<tr>
<td>workflow</td>
<td></td>
<td>coding</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Advanced issues

- Cumbersome editing
  - E.g., reorganization to insert a node to the middle

- Handling large models
  - 20+ nodes on a diagram:
    - Logical structure, readability possible
    - But needs human support
  - 100-1000+ nodes on a diagram
    - Technological limitations
    - Usability limitations
Example: Layouting
Example: Layouting
Layouting Support for Graphical Editors

- Computation of the position of nodes
  - Possible to do automatically
  - For a given metamodel
    - No unified visual requirements possible
    - We have to decide what is important to show

![Diagram showing minimum edge length and minimum amount of edge crossings](image.png)
Layouting Support for Graphical Editors

- **GraphViz** - [http://graphviz.org](http://graphviz.org)
  - Layouting project with high quality layout algorithm
  - Hard to integrate into Eclipse applications

  - Graph widgets for SWT applications
  - Easily integratable into Eclipse applications
  - Not the best layout algorithms

- **KIELER** - [http://rtsys.informatik.uni-kiel.de/trac/kieler/](http://rtsys.informatik.uni-kiel.de/trac/kieler/)
  - Eclipse based tools
  - Built-in support for GMF layouting
Textual or graphical?
## Comparison

<table>
<thead>
<tr>
<th>Textual Languages</th>
<th>Graphical Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick and simple editing</td>
<td>More cumbersome editing</td>
</tr>
<tr>
<td>References described as <em>string identifiers</em></td>
<td>References displayed visually</td>
</tr>
<tr>
<td>Inconsistent models during editing</td>
<td>Models always syntactically correct</td>
</tr>
<tr>
<td>Automatic formatting</td>
<td>Automatic layouting</td>
</tr>
<tr>
<td>Content assist</td>
<td>Tool list to add nodes/edges</td>
</tr>
<tr>
<td>Displaying validation errors, offering quick fixes</td>
<td></td>
</tr>
<tr>
<td>Both are supported with EMF-based technologies</td>
<td></td>
</tr>
</tbody>
</table>
Textual Editors:
- Insert character(s)
- Delete character(s)
- Replace character(s)

Graphical editors
- Insert model element
- Remove model element
- Insert reference
- Remove reference
- Modify attribute
Question: textual or graphical?

- No clear choice
- Rules of thumb
  - Behaviour description is usually simpler in textual
  - For structural information graphical is often better
- For simple languages
  - Form-based editing might also be an alternative
Xtext and GMF on the same instance model

```plaintext
SocialNetwork {
    Person Ujhelyi {
        male
        memberships BME, VVEC
    }
    Person Horvath {
        male
        memberships FTSRG
    }
    Community BME {
        Community FTSRG {
            Community test
        }
    }
    Person Test {
        female
        memberships test
    }
    Community VVEC
    Person Proba {
        male
    }
    Community Pr2
    Person valaki [{
        male
    }
    Ujhelyi is friend of Horvath
    Test is married to Ujhelyi
}
```
Derived Graphical viewer support

- Xtext Generic Viewer component
  - Created by Xtext developers
  - Independent from the main Xtext development
  - Requies an extra language
    - to define uni-directional mapping
    - to define format

- See “A fresh look at graphical modeling” for details
  - [http://www.slideshare.net/schwurbel/a-fresh-look-at-graphical-editing-10068461](http://www.slideshare.net/schwurbel/a-fresh-look-at-graphical-editing-10068461)
Concrete Syntax Design

Conclusion
Concrete Syntax Design

- Multiple approaches
  - Textual and/or graphical syntaxes
  - Combinable

- Large amount of development work needed
  - Directly used by users
  - Usability issues

- Not everything is coded in an editor
  - Editor + corresponding views form the interface