A Reference Structure for Metamodels of Quality-Aware Domain-Specific Languages

Young Researcher Forum, 05.04.2016
Misha Strittmatter, Robert Heinrich
Why use Modeling / Metamodels?

- Model-driven Engineering (MDE)
  - Use domain-specific models
  - Raise the abstraction level
  - „Hide“ complexity behind tooling
  - Use models for analysis, simulation, code generation

- Metamodel
  - Abstract syntax of a (modelling) language
  - Defines set of its instances (models)

- Our focus
  - Quality-aware domain specific languages
  - EMOF based or similar
Metamodel Evolution

Problem:

Integration of new features into the PCM

PCM

Design Decisions

Patterns

Security

Maintainability

Requirements

PCM = Metamodel
Metamodel Evolution

Solution:
- Build monolith!?
Metamodel Evolution

Solution:
- Branch the Metamodel for each concept?

PCM
+Requirements

PCM
+Maintainability

PCM
+Patterns

PCM
= Metamodell
Metamodel Evolution

Solution
- Externally extend the PCM

Design Decisions
Patterns
Maintainability
Base for extension should be well designed

Security
Requirements

PCM

= Metamodel
Problems

- Monolithic metamodels
  - All or nothing reuse
  - Misleads to intrusive extension
    \[ \Rightarrow \text{Degradation of structure over time} \]

- Internal structural problems
  - Unnecessary dependencies
  - Uns suited for extension
  - Improper modularization
  - Inconsistent structuring

- Impeding factors
  - Much dependent code
  - High coupling

Increased complexity
\[ \Rightarrow \text{impaired understandability and maintainability} \]
Solution

- Use external extension
  - No structural degradation
  - No feature overload
    - User only chooses features he needs
    - Ease of reuse

- Use a reference structure
  - Enforce modularity
  - Restrict dependencies
  - Provides explicit structure
    - Consistency
    - Guides placement of new extensions
    - Less prone to degradation
Reference Structure

- Concerns encapsulated in Modules (own Metamodels)
- Module dependencies form cycle free, directed graph
- Modules are organized in layers
- Specific types of dependencies between layers

Diagram:
- Paradigm
- Domain
- Quality
- Analysis

Specific for quality-aware ADLs
Further Contributions

- **Process support**
  - Modes
    - Design from scratch
    - Refactoring legacy metamodels
  - Tool support, guidance
  - Full automation not possible

- **Extension mechanisms**
  - Evaluate for ecore
  - MDSD Profiles have drawbacks
  - Conceptually valid alternatives
Metamodell-based Tools

Editors

Simulators

Validators

Metamodel

Analyzers, Solvers

Generators, Transformations

*
Modular Extensible Tooling

- Tools should be able to support extensibility of the metamodel
- Different types of tooling
  - Metamodel-oriented
    - Editors, generators
    - Support modularity of the metamodel
  - Extrinsic functionality
    - Simulators
    - Only support predefined/foreseen extension
Evaluation

- Applicability
  - PCM
  - Smart grid topology
  - Automation control systems

- Benefit
  - Compare effort for extension, modification
  - (Metrics)
State of Research

- Almost done
  - Reference structure *
  - PCM case study *
  - Extension mechanisms
  - Extensible modular graphical editors

- Todo
  - Process support
  - General tooling concept
  - Automation control case study

* [ModComp’15]
Summary

- Problem: metamodel evolution
  - Intrusive extensions
  - Erosion of structure over time
- Solution: reference structure
  - Enforce external extensions
  - Provide explicit structure
  - Classification guidance
  - Tooling concept, extension mechanisms

Thank you for your attention!
Application: Starting Point

- Subset of PCM and some extensions

PCM

- Performance Results

- Static Dependencies

- Modifications

- Artifacts and Staff

Component Architecture, Behavior, Performance, Reliability, Infrastructure, Events, Resources, Deployment, Usage, ...

KAMP Maintainability Prediction

Subset of PCM and some extensions
Application: Result

Core Entities

Composition

Software Components

Static Dependencies

Development Artifacts

Modifications

Behavior

Failure Types

Performance Metrics

KAMP Input

KAMP Result

Performance Results

Performance Configuration

Application: Result

Misha Strittmatter, Robert Heinrich
A Reference Structure for Metamodels of Quality-Aware DSLs
A Reference Structure for Metamodels of Quality-Aware DSLs
Related Work

- Structuring:
  - Orthographic Software Modeling [Atkinson10]
  - Deep modeling [Atkinson12]
  - UML archetypes [Coad99]
  - Software “blood types” [Siedersleben04]

- Extensibility:
  - JetBrains MPS [Voelter12]
  - Arch Studio [Dashofy05]

- Modularity:
  - Generators [Jung15]
  - Transformations [Rentschler14]
## Dependencies within the Layering

<table>
<thead>
<tr>
<th></th>
<th>Π</th>
<th>Ω</th>
<th>Σ</th>
<th>Δ</th>
<th>Ύ</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>Y</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>X</td>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram showing dependencies among layers with symbols and values indicating relationships.
Extension Mechanisms

- Implementation of extends relation
  - Simple reference
  - Simple inheritance
  - Abstract reference
  - Decorator
  - Stereotyping [Kramer12]
  - Aspect(-oriented) extension [Jung14]
  - Roles
  - Completions [Kapova13]
  - MIRs
  - Model weaving