31. Feature Models and MDA for Product Lines

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediantechnik
Gruppe Softwaretechnologie

http://st.inf.tu-dresden.de/teaching/swt2
Version 16-0.1, 14. Januar 2017

1. Feature Models
2. Product Linie Configuration with Feature Models
3. Multi-Stage Configuration
4. Multi-Stage Configuration of Transformations
Literature


References

The most simple software development process is the V-model, but for software product lines (SPL) we need other processes. *Software Product Line Engineering (SPLE)* is the discipline of creating product lines.
Object-Oriented Analysis vs Object-Oriented Design

- Requirements specification
  - Textual requirements (stories)
  - Use cases
  - Context model
  - Analysis model
  - Domain model

- Architectural design
- Detailed design

Prof. U. Aßmann
Extended to Model-Driven Architecture (MDA)

- Horizontal product line: one product idea in several markets
Software Product Lines (Software Product Families)

- **Requirements Specification**: Use cases, textual requirements (stories), context model, domain model, Feature Model (variability model), analysis model.

- Products:
  - Product 1
  - Product 2
  - Product n
Adding Extensions to Abstract Models in the MDA

➢ In the following, we extend the MDA (below) with configuration

- Platform independent model (PIM)
- Platform-1-specific extension (PSE)
- Model weaving
- Platform-1 specific model (PIM)
- Platform-2 specific extension (PSE)
- Model weaving
- Platform-(1+2) specific model (PSM)
Vertical product line: several products (different feature sets) in one or several markets
- The VIM (variant independent model) is the common model of the product family
31.1 PRODUCT LINES WITH FEATURE TREES AND FEATURE MODELS
Feature Models for Product Configuration

- Feature models are used to express variability in Product Lines
  - alternative,
  - mandatory,
  - optional features, and
  - their relations

- A *variant model* represents a concrete product (variant) from the product line
  - The variant model results from a selection of a subgraph of the feature model
  - The variant model can be used to parameterize and drive the product instantiation process
Feature Models

- The Feature Tree Notation is derived from And-Or-Trees

based on FODA-Notation by Kang et al. (1990)
Example

- A1 or A2 or A3
- B1; B2 xor B3
- B4; optional B5
- B1; B7
A Feature Model for Computer-Aided Cognitive Rehabilitation

[K. Lehmann-Siegemund, Diplomarbeit]
Feature Models form a Specific Concern space

- Remember: Aspect-oriented modeling means to separate a **concern space** from a **component space**
- In SPLE, the feature space forms a concern space; the component space is usually called the solution space
Ex: Plugins have Features (in Eclipse)

Softwaretechnologie II

Plug-ins and Fragments

Select plug-ins and fragments that should be packaged in this feature.

[Table of plug-ins and fragments]

Plug-in Details

Specify installation details for the selected plug-in.

Name: Core File System for Linux PPC
Version: 1.0.130.v20090604-1400
Download Size (MB): 0
Installation Size (MB): 0

Unpack the plug-in archive after the installation

Specify environment combinations in which the selected plug-in can be installed. Leave blank if the plug-in does not contain platform-specific code:

Operating Systems: Linux

Window Systems:

Languages:

Architecture: ppc

Total: 25

Overview  Information  Plug-ins  Included Features  Dependencies  Installation  Build  Feature.xml  build.properties
31.1.2 MODEL-BASED PRODUCT LINES
Mapping Features to Model Fragments (Model Snippets)

- **Feature mapping** bridges the gap between configuration space and solution space.
- Possible artefacts in the solution space:
  - Models defined in DSLs; Model fragments (snippets)
  - Architectural artefacts (components, connectors, aspects)
  - Source code, Files
- A **Model-based Product Line** is a SPL with models in the component space.
31.2 PRODUCT-LINE CONFIGURATION WITH FEATURE MODELS
Different Approaches of Variant Selection in an SPL Additive approach

- Map all features to model fragments (model snippets)
- Compose them with a core model based on the presence of the feature in the variant model

Pros:
- conflicting variants can be modelled correctly
- strong per-feature decomposition

Cons:
- traceability problems
- increased overhead in linking the different fragments
Different Approaches of Variant Selection (2)

**Subtractive approach**

- Model all features in one model
- Remove model elements based on absence of the feature in the variant model

**Pros:**
- no need for redundant links between artifacts
- short cognitive distance

**Cons:**
- conflicting variants can't be modelled correctly
- huge and inconcise models
The Mapping Problem between Features and Solution Elements

Creation
Visualisation
Validation
Derivation

Problem Space

Solution Space
Mapping Features to Models

- FeatureMapper - a tool for mapping of feature models to modelling artefacts developed at the ST Group
- Screencast and paper available at http://featuremapper.org

- Advantages:
  - Explicit representation of mappings
  - Configuration of large product lines from selection of variants in feature trees
    - Customers understand
  - Consistency of each product in the line is simple to check
  - Model and code snippets can be traced to requirements
FeatureMapper
We chose an explicit Mapping Representation in our tool FeatureMapper.
Mappings are stored in a mapping model that is based on a mapping metamodel.
From Feature Mappings to Model Transformations

Diagram showing the process from feature models to solution models through transformations.
Visualisation of Mappings (1)

- Visualisations of Feature Mappings by colors (as in AOP)
- **View construction:** In many cases, developers are interested only in a particular aspect of the connection between a feature model and realising artefacts
  - How a particular feature is realised?
  - Which features communicate or interact in their realisation?
  - Which artefacts may be effectively used in a variant?
- Solution of the FeatureMapper: MappingViews, a visualisation technique that provides four basic visualisations
  - Realisation View
  - Variant View
  - Context View
  - Property-Changes View
For one Variant Model, the realisation in the solution space is shown.
The variant view shows different variant realisations (variant models) in parallel.
The Context View draws the variants with different colors
- Aspect-separation: each variant forms an aspect
Property-Changes View

Recorded change-set of changing the cardinality of the reflexive association of Group to itself from 1 to many

<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownedEnd Property source</td>
<td>upperValue Literal Unlimited Natural</td>
</tr>
<tr>
<td>lowerValue Literal Integer</td>
<td></td>
</tr>
<tr>
<td>ownedEnd Property target</td>
<td>upperValue Literal Unlimited Natural</td>
</tr>
<tr>
<td>Property value</td>
<td>1</td>
</tr>
<tr>
<td>lowerValue Literal Integer</td>
<td></td>
</tr>
<tr>
<td>Property value</td>
<td>0</td>
</tr>
<tr>
<td>Property value</td>
<td>1</td>
</tr>
</tbody>
</table>

Arbitrary Depth
Textual Languages Support (1)

- Unified handling of modelling languages and textual languages by lifting textual languages to the modelling level with the help of EMFText

- All >80 languages from the EMFText Syntax Zoo are supported, including Java 5

- [http://emftext.org](http://emftext.org)
Textual Languages Support (2)

- Aspect-related color markup of the code

```
package org.featuremapper.examples.contactmanagement;
import java.util.LinkedHashSet;
import java.util.Set;

public class Contact {
    private String name;
    private Set<Relationship> relationships;

    public Contact(String name) {
        this.name = name;
        this.relationships = new LinkedHashSet<Relationship>();
    }

    public void addRelationship(Relationship relationship) {
        this.relationships.add(relationship);
    }

    public String toString() {
        final StringBuffer result = new StringBuffer();
        result.append("Name: " + name);
        result.append("Relationships: " + relationships);
        return result.toString();
    }
```
Transformations in the solution space build the product.
31.3 MULTI-STAGE CONFIGURATION
FEASIPLE: A Multi-Stage Process Architecture for PLE

- Chose one variant on each level
- Feature Tree as input for the configuration of the model weavings
FEASIBLE: A Multi-Stage Process Architecture for PLE

Goal: a staged MDSD-framework for PLE where each stage produces the software artefacts used for the next stage
Advantages of FEASIPLE

- Characteristic feature 1:
- Variability on each stage
Advantages of FEASIPLE

Characteristic feature 2:
- Different modelling languages, component systems and composition languages per stage
Advantages of FEASIPLE

- Characteristic feature 3:
- Different composition mechanisms per stage
Advantages of FEASIPLE

- Characteristic feature 4:
- Composition mechanisms are driven by variant selection
31.4 MULTI-STAGE CONFIGURATION OF TRANSFORMATIONS
How do we compose transformations? Between different stages?

- **functional Feature Model**
- **context Feature Model**
- **platform Feature Model**

- **VIM Mapping**
- **PIM Mapping**
- **PSM Mapping**

- **Variant Independent Model**
- **Platform Independent Models**
- **Platform Specific Models**

- **M2M Transformations**
- **M2C Generators**

- **Platform Specific Code**
TraCo: A Framework for Safe Multi-Stage Composition of Transformations

TraCo encapsulates transformations into composable components [TraCo]

- Arranges them with composition programs of parallel and sequential transformation steps (multi-threaded transformation)

**Diagram:**

- **Functional variant**
  - $V_1 \rightarrow M_1 \rightarrow T_1 \rightarrow S\text{A}_1$

- **Platform variant**
  - $V_2 \rightarrow M_2 \rightarrow T_2 \rightarrow S\text{A}_2 \rightarrow S\text{A}_1^*$
  - $T_1^* \rightarrow M_1^* \rightarrow V_1^*$

- **Context variant**
  - $V_n \rightarrow M_n \rightarrow T_n \rightarrow S\text{A}_n$

**Labels:**

- $V$ Feature Selection
- $M$ Mapping
- $S\text{A}$ Solution Artefact
- $T$ Transformation

**Symbol Key:**

- $\rightarrow$: Composition step
- $\rightarrow^*$: Multi-threaded transformation step
Steps in Multi-Staged Derivation of Transformations

1. In TraCo, Transformations are represented as composable components

2. **Definition and Composition of Transformation Steps**
   - A *Composition System* is needed (course CBSE): Allows for reuse of arbitrary existing transformation techniques

3. **Validation of each transformation and composition step**
   - Type-checking
   - Invariant- and constraint-checking
   - Correctness of port and parameter binding
   - Static and dynamic analysis

4. **Execution of composition program**
Multi-Staged Derivation of Transformations

- Implemented in our tool TraCo
Composition Programs can be Configured (Metacomposition)

„Anything you can do, I do meta“ (Charles Simonyi)

- The composition program shown in the last slide can be subject to transformation and composition

- If we build a product line with TraCo, platform variability can be realised by different transformation steps

- A TraCo composition program can be used with FeatureMapper
  - Multi-Staged transformation steps
  - Even of composition programs

- More about metacomposition in CBSE course
The final frontier: Ensuring Well-formedness of SPLs

- Motivation: Make sure that well-formedness is ensured
  - Feature Model
  - Mapping Model
  - Solution Models

- Well-formedness rules are described using OCL

- Constraints are enforced during mapping time
Case Studies with FeatureMapper, TraCo, and FEASIPLE

- **Simple Contact Management Application** Software Product Line
  - FeatureMapper used to map features to UML2 model elements
  - Both static and dynamic modelling

- **Simple Time Sheet Application** Software Product Line
  - FeatureMapper used to tailor ISC composition programs
  - ISC used as a universal variability mechanism in SPLE
  - Meta Transformation

- **SalesScenario** Software Product Line
  - FeatureMapper used to tailor models expressed in Ecore-based DSLs
  - was developed in project feasiPLe (http://www.feasiple.de)

- **TAOSD AOM Crisis Management System**
Summary

- Configuration of product lines with mapping of feature models to solution spaces
- Mapping of Features to models in Ecore-based languages using FeatureMapper
- Visualisations of those mappings using MappingViews
  - Realisation View
  - Variant View
  - Context View
  - Property-Changes View
- Derivation of solution models based on variant selection and mapping
- Multi-Staged derivation using TraCo
- Ensuring well-formedness of SPLs

http://featuremapper.org
The End

- Many slides are courtesy Florian Heidenreich