53) Feature Models, Domain Models and Product Lines

1. Feature Models
2. Product Line Configuration with Feature Models
3. Multi-Stage Configuration

Obligatory Literature

References


Extended to Model-Driven Architecture (MDA)

Product Lines (Product Families)

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

Platform-independent model
Platform-1 specific model
Platform-(1,.., n) specific model

Feature Model (variability model)

Product 1
Product 2
Product n
In the following, we extend the MDA (below) with configuration:

- Platform independent model (PIM)
- Platform-1 specific extension (PSE)
- Platform-1 specific model (PIM)
- Platform-2 specific extension (PSE)
- Platform-(1+2) specific model (PSM)

Configuration of Variabilities in Product Lines:

- Als Obermodelle des Analysemodells: Analyseklassen erben von Domänenklassen,

- Domain Model
- Analysis Model
- Product Line Model (Framework, VIM)
- Product PIM
- Product PSM
53.1 PRODUCT LINES WITH FEATURE TREES AND FEATURE MODELS

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

A variant model represents a concrete product 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

Feature Tree Notation

- Group of AND Features
- Group of Alternative (XOR) Features
- Mandatory Feature
- Optional Feature
- Group of OR Features

Example:

- A1 or A2 or A3
- B1; B2 xor B3
- B4; optional B5
- B1; B7

based on FODA-Notation by Kang et al. (1990)
Mapping Features to Model Fragments (Model Snippets)

- Bridging the gap between configuration and solution space
- Need for mapping of features from feature models to artefacts of the solution space
- Possible artefacts
  - Models defined in DSLs
  - Model fragments (snippets)
  - Architectural artefacts (components, connectors, aspects)
  - Source code
  - Files

- But how can we achieve the mapping...?
**53.2 PRODUCT-LINE CONFIGURATION WITH FEATURE MODELS**
Different Approaches of Variant Selection

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

Subtractive approach

- Model all features in one model
- Remove 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
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](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
Mapping Features to Models

- We chose an explicit *Mapping Representation* in our tool *FeatureMapper*
- Mappings are stored in a mapping model that is based on a mapping metamodel
Visualisations play a crucial role in Software Engineering
- It’s hard to impossible to understand a complex system unless you look at it from different points of view

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 realisations 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</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td></td>
</tr>
<tr>
<td>source</td>
<td></td>
</tr>
<tr>
<td>lowerValue</td>
<td>Literal Unlimited Natural</td>
</tr>
<tr>
<td>upperValue</td>
<td>Literal Unlimited Natural</td>
</tr>
<tr>
<td>Property value</td>
<td></td>
</tr>
<tr>
<td>arbitrary</td>
<td></td>
</tr>
<tr>
<td>Depth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Arbitrary Depth</td>
</tr>
<tr>
<td></td>
<td>= 1</td>
</tr>
<tr>
<td></td>
<td>lowerValue</td>
</tr>
<tr>
<td></td>
<td>Property value</td>
</tr>
<tr>
<td></td>
<td>Arbitrary Depth</td>
</tr>
</tbody>
</table>
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

Textual Languages Support (2)

- Aspect-related color markup of the code
Transformations in the solution space build the product

53.3 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

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

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

Characteristic feature 3:
Different composition mechanisms per stage

Characteristic feature 4:
Composition mechanisms are driven by variant selection
How do we compose transformations? Between different stages?

TraCo: A Framework for Safe Multi-Stage Composition of Transformations

- TraCo encapsulates transformations into composable components
  - Arranges them with composition programs of parallel and sequential transformation steps (multi-threaded transformation)

Functional variant

Platform variant

Context variant

$V_1$ Feature Selection  
$M_1$ Mapping  
$SA_1$ Solution Artefact  
$T_1$ Transformation

$V_n$ Feature Selection  
$M_n$ Mapping  
$SA_n$ Solution Artefact  
$T_n$ Transformation
Steps in Multi-Staged Derivation of Transformations

1. 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

Implemented in our tool TraCo

- Component Model,
- Composition Language,
- Composition Technique
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 of all participating models is ensured
  - Feature Model
  - Mapping Model
  - Solution Models
- Well-formedness rules are described
- Constraints are enforced during mapping

Motivation: Make sure that well-formedness of all participating models is ensured
- Feature Model
- Mapping Model
- Solution Models

Well-formedness rules are described

Constraints are enforced during mapping

Constraint OWL
- Feature ContactManagement
  - Group 0
    - Feature Addresses
    - Feature Relationships
    - Feature ContactOpportunities
    - Feature Notes
  - Feature Groups
    - Group 0
      - Feature MultipleAssignmen
      - Feature ArbitraryDepth
- Feature Synchronisation
- FeatureInCrisis
  - status : VehicleStatus

Vehicle

<<enumeration>>

VehicleStatus

INVOLVED
LOWDAMAGED
DAMAGED

VehicleInCrisis

+ status : VehicleStatus

org.featuremapper.example

Vehicle

Properties

Value
Constrained Features
- Feature MultipleAssignment, Feature Highrise
Expression
Language
OWL
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