31. Feature Models and MDA for Product Lines

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

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und MultimEDIATEchnik
Lehrstuhl Softwaretechnologie
http://st.inf.tu-dresden.de

Lecturer: Dr. Sebastian Götz


Object-Oriented Analysis vs Object-Oriented Design

- requirements specification
- textual requirements (stories)
- use cases
- context model
- analysis model
- domain model
- architectural design
- detailed design

Object-Oriented Analysis vs Object-Oriented Design

Prof. U. Aßmann

Feature-driven SPLE
Horizontal product line: one product idea in several markets

- Extended to Model-Driven Architecture (MDA)

- Feature-driven SPLE

- requirements specification

- textual requirements (stories)

- use cases

- context model

- analysis model (CIM)

- domain model

- Platform independent model

- Platform-(1,.., n) specific model
Product Lines (Product Families)

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

Product 1

Product 2

Product n

Prof. U. Aßmann

Feature-driven SPLE
In the following, we extend the MDA (below) with configuration.

- Platform independent model (PIM)
- Platform-1 specific extension (PSE)
- Platform-2 specific extension (PSE)
- Platform-(1+2) specific model (PSM)
**Vertical product line: several products 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 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
The Feature Tree Notation is derived from And-Or-Trees

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

- FeatureA
- FeatureB
- FeatureC
- FeatureD

- A1 or A2 or A3
- B1; B2 xor B3
- B4; optional B5
- B1; B7
Bridging the gap between configuration and solution space

Need for mapping of features from feature models to artifacts of the solution space

Possible artifacts
- Models defined in DSLs
- Model fragments (snippets)
- Architectural artifacts (components, connectors, aspects)
- Source code
- Files

But how can we achieve the mapping...?
31.2 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 modeled 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 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 modeled correctly
- huge and non-concise models
The Mapping Problem between Features and Solution Elements

Feature-driven SPLE

Problem Space

FeatureA

FeatureB

FeatureC

FeatureD

FeatureE

Solution Space

Creation

Visualisation

Validation

Derivation
- **FeatureMapper** - a tool for mapping of feature models to modeling artifacts 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
Feature-driven SPLE
- 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
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 variant realisations (variant models) in parallel.
The Context View draws the variants with different colors

- Aspect-separation: each variant forms an aspect
Recorded change-set of changing the cardinality of the reflexive association of Group to itself from 1 to many

```
<Class> Group
  <Association> Association4
    <Property> source : Group
      <Property> target : Group [0..*]
        <Literal Unlimited Natural> *
        <Literal Integer> 0
        <Literal Unlimited Natural> *

 ownedEnd Property source
  upperValue Literal Unlimited Natural
  lowerValue Literal Integer

 ownedEnd Property target
  upperValue Literal Unlimited Natural
  Property value

 Arbitrary Depth
  - 1
  lowerValue Literal Integer
    Property value
    - 0
    - 1
```

Feature-driven SPLE
Unified handling of modeling 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
➢ Aspect-related color markup of the code
Transformations in the solution space build the product
31.3 MULTI-STAGE CONFIGURATION
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 artifacts used for the next stage.
Advantages of FEASIPLE

- **Characteristic feature 1:**
- **Variability on each stage**

![Diagram of FEASIPLE features](image.png)
Advantages of FEASIPLE

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

- **Characteristic feature 3:**
- **Different composition mechanisms per stage**

![Diagram of composition mechanisms per stage](image)
Advantages of FEASIPLE

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

- **functional Feature Model**
  - VIM Mapping
  - Variant Independent Model
  - M2M Trafos
  - Platform Independent Models
  - Platform Independent Models
  - Platform Independent Models
- **context Feature Model**
  - PIM Mapping
  - Platform Specific Models
  - M2M Trafos
- **platform Feature Model**
  - PSM Mapping
  - Platform Specific Models
  - M2C Generators
  - Platform Specific Code
TraCo encapsulates transformations into composable components

- Arranges them with *composition programs of* parallel and sequential transformation steps (multi-threaded transformation)
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
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 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
Many slides are courtesy of Florian Heidenreich