

## 31) Feature Models and MDA for Product Lines

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

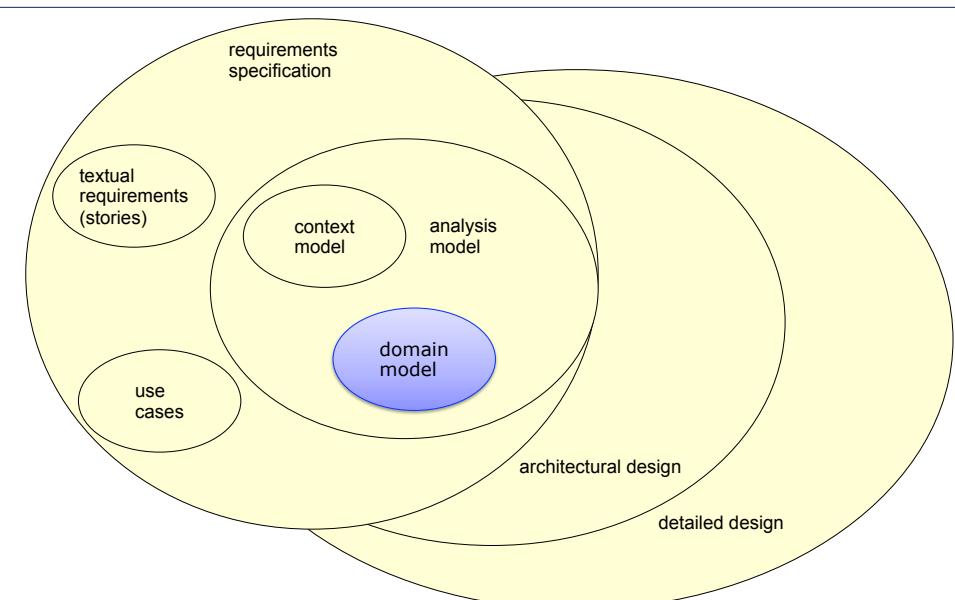
➤ Prof. Dr. U. Aßmann  
 ➤ Florian Heidenreich  
 ➤ Technische Universität Dresden  
 ➤ Institut für Software- und Multimedia-technik  
 ➤ Gruppe Softwaretechnologie  
<http://st.inf.tu-dresden.de>  
 ➤ Version 11-0-2, Januar 25, 2012

### References

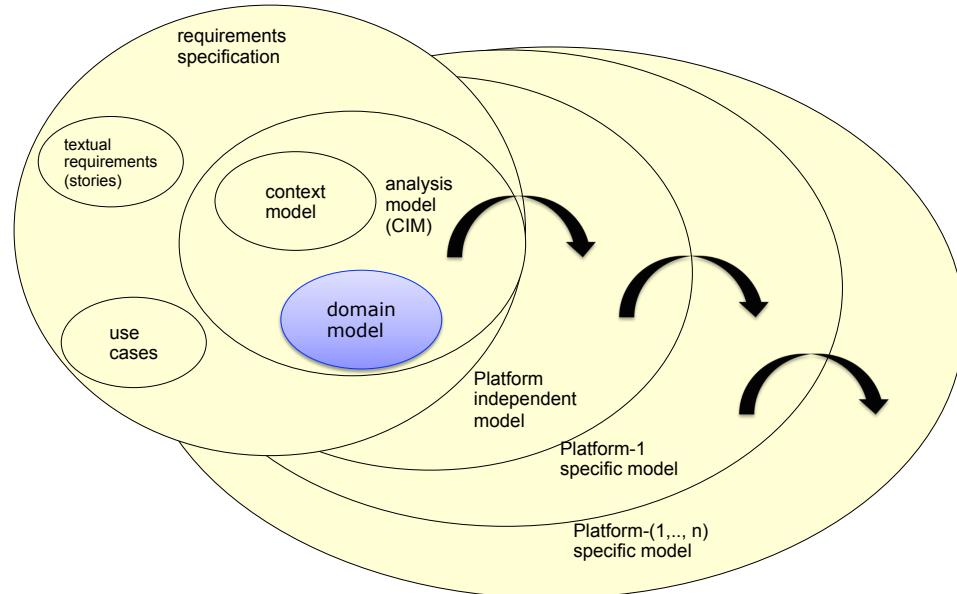
- [Aßm03] U. Aßmann. *Invasive Software Composition*. Springer, 2003.
- [Cza05] K. Czarnecki and M. Antkiewicz. *Mapping Features to Models: A Template Approach Based on Superimposed Variants*. In R. Glück and M. Lowry, editors, *Proceedings of the 4th International Conference on Generative Programming and Component Engineering (GPCE'05)*, volume 3676 of LNCS, pages 422-437. Springer, 2005.
- [Cza06] K. Czarnecki and K. Pietroszek. *Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints*. In *Proceedings of the 5th International Conference on Generative Programming and Component Engineering (GPCE'06)*, pages 211-220, New York, NY, USA, 2006. ACM.
- [Hei08a] F. Heidenreich, J. Kopcsek, and C. Wende. *FeatureMapper: Mapping Features to Models*. In *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, pages 943-944, New York, NY, USA, May 2008. ACM.
- [Hei08b] Florian Heidenreich, Ilie Šavga and Christian Wende. *On Controlled Visualisations in Software Product Line Engineering*. In *Proc. of the 2nd Int'l Workshop on Visualisation in Software Product Line Engineering (ViSPL 2008)*, collocated with the 12th Int'l Software Product Line Conference (SPLC 2008), Limerick, Ireland, September 2008.
- [Hei09] Florian Heidenreich. *Towards Systematic Ensuring Well-Formedness of Software Product Lines*. In *Proceedings of the 1st Workshop on Feature-Oriented Software Development (FOSD 2009)* collocated with MODELS/GPCE/SLE 2009, Denver, Colorado, USA, October 2009. ACM Press

- Florian Heidenreich, Jan Kopcsek, and Christian Wende. *FeatureMapper: Mapping Features to Models*. In *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, Leipzig, Germany, May 2008.
  - <http://fheidenreich.de/work/files/ICSE08-FeatureMapper--Mapping-Features-to-Models.pdf>

### Object-Oriented Analysis vs Object-Oriented Design

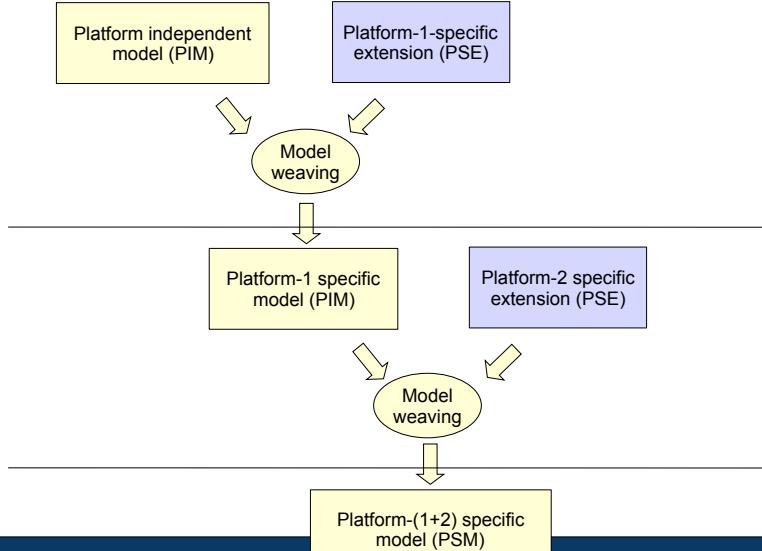


## Extended to Model-Driven Architecture (MDA)

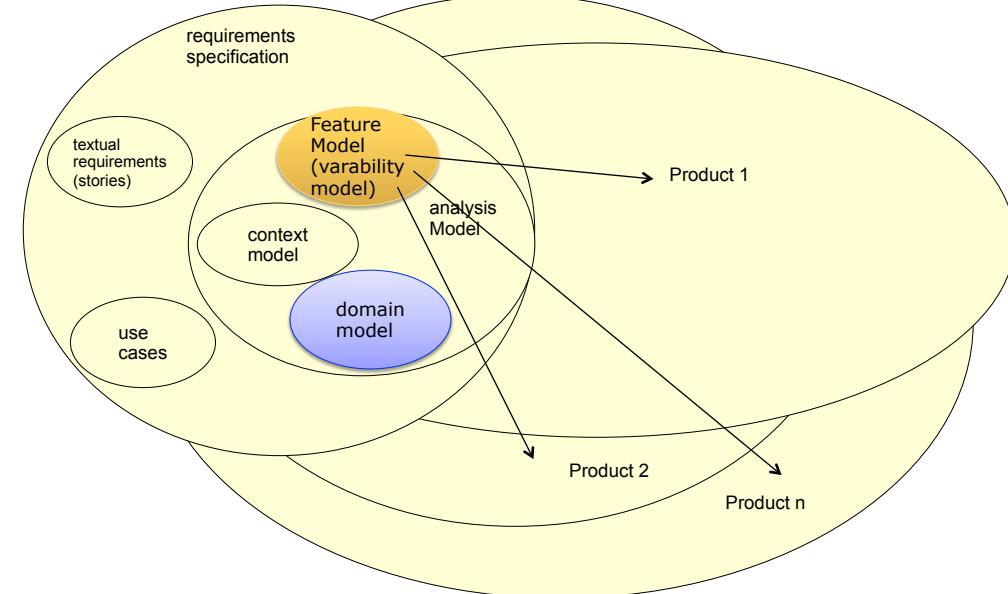


## Adding Extensions to Abstract Models in the MDA

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

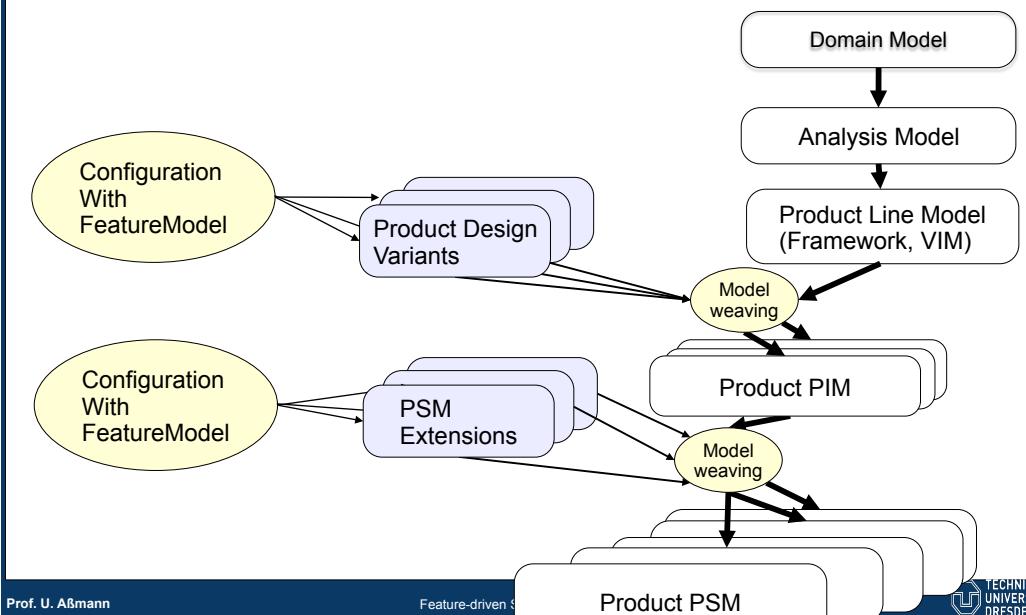


## Product Lines (Product Families)



## Configuration of Variabilities in Product Lines (MDA for Product Lines)

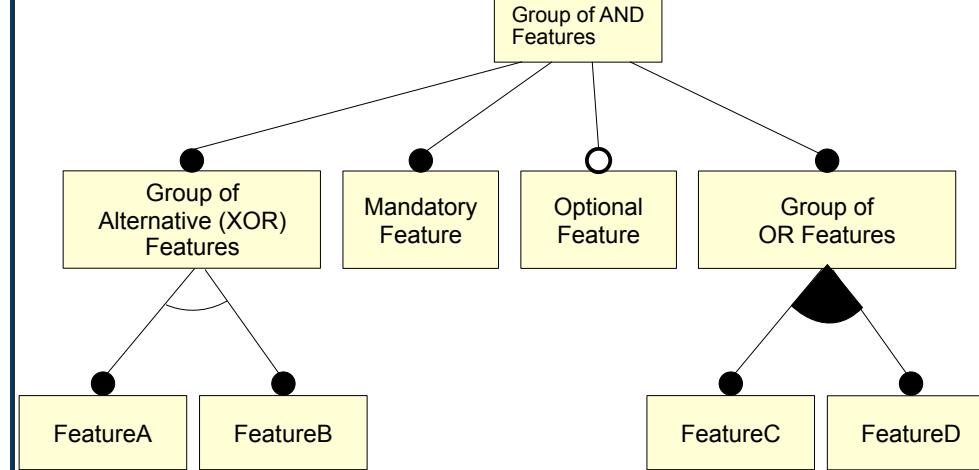
- The VIM (variant independent model) is the common model of the product family



# 31.1 PRODUCT LINES WITH FEATURE TREES AND FEATURE MODELS

## Feature Models

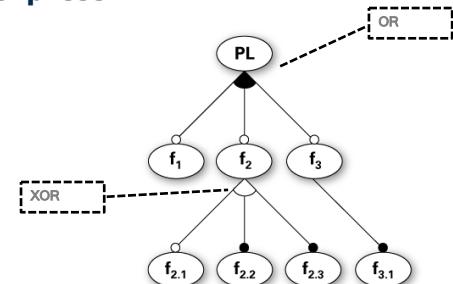
➤ Feature Tree Notation



PhD Thesis, Czarnecki (1998)  
based on FODA-Notation by Kang et al. (1990)

➤ Feature models are used to express variability in Product Lines

- alternative,
- mandatory,
- optional features, and
- their relations

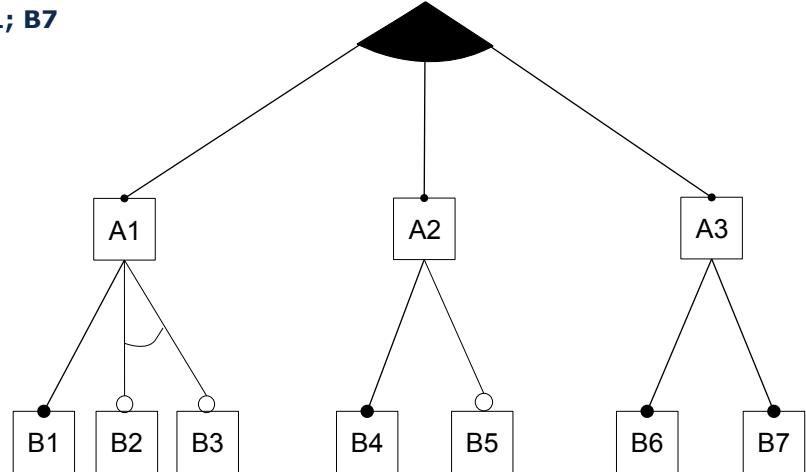


➤ A variant model represents a concrete product from the product line

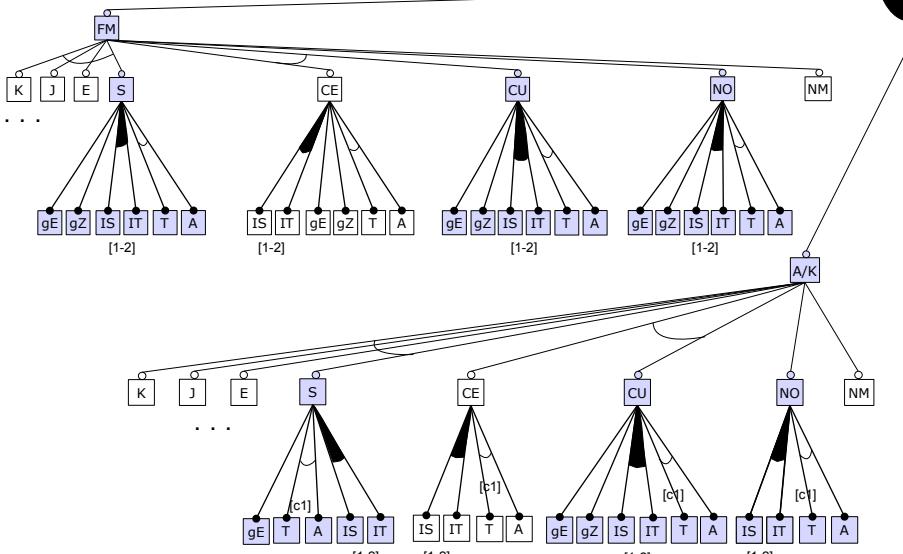
- The variant model results from a selection of a subgraph of the feature model
- The variant model can be used to parameterize and drive the product instantiation process

## Beispiel

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



## ➤ [K. Lehmann, Diplomarbeit]



Ex: Plugins have Features (in Eclipse)

## Plug-ins and Fragments

Plug-ins and Fragments

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

org.eclipse.core.filesystem.linux.ppc (1.0.100.v20080604-1400)
org.eclipse.core.filesystem.linux.x86 (1.0.20.v20080604-1400)
org.eclipse.core.filesystem.linux.x86_64 (1.0.100.v20080604-1400)
org.eclipse.core.filesystem.macosx (1.1.0.v20090112)
org.eclipse.core.filesystem.solaris.sparc (1.0.100.v20080604-1400)
org.eclipse.core.filesystem.win32.x86 (1.1.0.v20080604-1400)
org.eclipse.core.resources.win32.x86 (3.5.0.v20081020)
org.eclipse.equinox.launcher.carbon.macosx (1.0.200.v20090520-1835)
org.eclipse.equinox.launcher.gtk.linux.ppc (1.0.200.v20090519)
org.eclipse.equinox.launcher.gtk.linux.x86 (1.0.200.v20090520)
org.eclipse.equinox.launcher.gtk.linux.x86_64 (1.0.200.v20090519)
org.eclipse.equinox.launcher.gtk.solaris.sparc (1.0.200.v20090519)
org.eclipse.equinox.launcher.win32.win32.x86 (1.0.200.v20090519)
org.eclipse.equinox.security.macosx (1.100.0.v20090520-1800)
org.eclipse.equinox.security.win32.x86 (1.0.100.v20090520-1800)
org.eclipse.swt.carbon.macosx (3.5.0.v3550b)
org.eclipse.swt.gtk.linux.ppc (3.5.0.v3550b)
org.eclipse.swt.gtk.linux.x86 (3.5.0.v3550b)
org.eclipse.swt.gtk.linux.x86_64 (3.5.0.v3550b)
org.eclipse.swt.gtk.solaris.sparc (3.5.0.v3550b)
org.eclipse.swt.win32.win32.x86 (3.5.0.v3550b)
org.eclipse.ui.carbon (4.0.0.12009090525-2000)
org.eclipse.ui.win32 (3.2.100.v20090429-1800)
org.eclipse.update.core.linux (3.2.100.v20081008)
org.eclipse.update.core.win32 (3.2.100.v20080107)

Total: 25

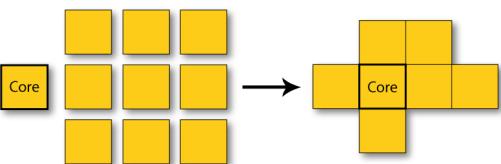
Overview | Information | Plug-ins | Included Features | Dependencies | Installation | Build | feature.xml | build.properties

- 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... ?

## 31.2 PRODUCT-LINE CONFIGURATION WITH FEATURE MODELS

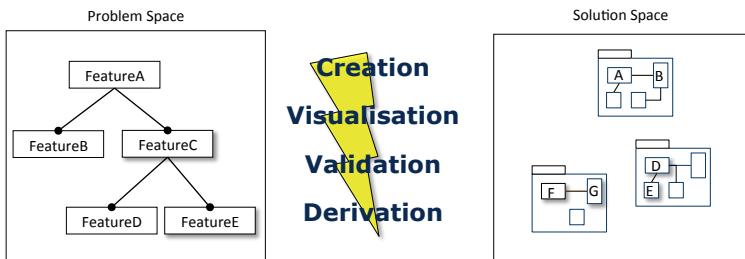
## Different Approaches of Variant Selection Additive approach

- 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



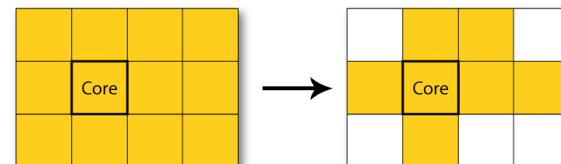
- Pros:
  - conflicting variants can be modelled correctly
  - strong per-feature decomposition
- Cons:
  - traceability problems
  - increased overhead in linking the different fragments

## The Mapping Problem between Features and Solution Elements



## Different Approaches of Variant Selection (2) Subtractive approach

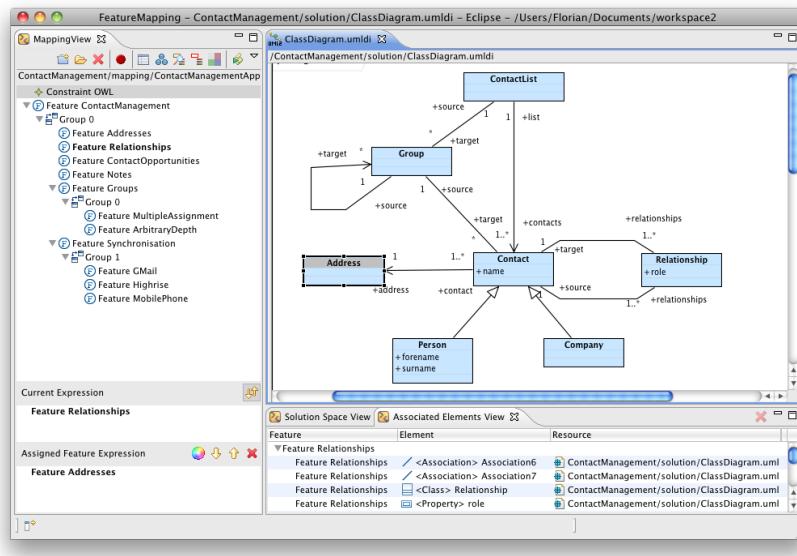
- Model all features in one model
- Remove elements based on absence of the feature in the variant model



- Pros:
  - no need for redundant links between artifacts
  - short cognitive distance
- Cons:
  - conflicting variants can't be 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>
- 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

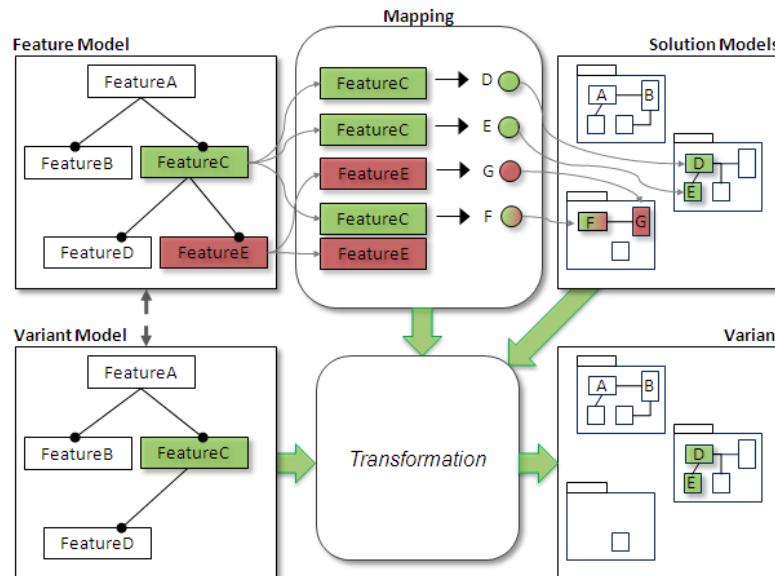


Prof. U. Aßmann

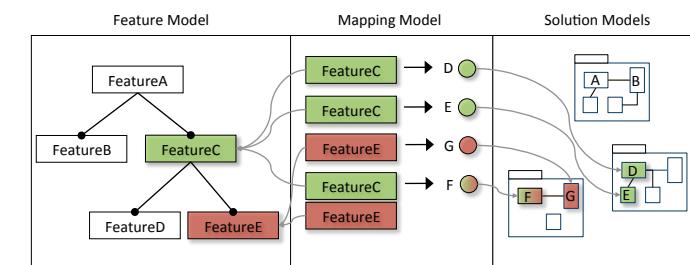
Feature-driven SPLE

Slide 21

## From Feature Mappings to Model Transformations



- We chose an explicit **Mapping Representation** in our tool **FeatureMapper**
- Mappings are stored in a mapping model that is based on a mapping metamodel



Prof. U. Aßmann

Feature-driven SPLE

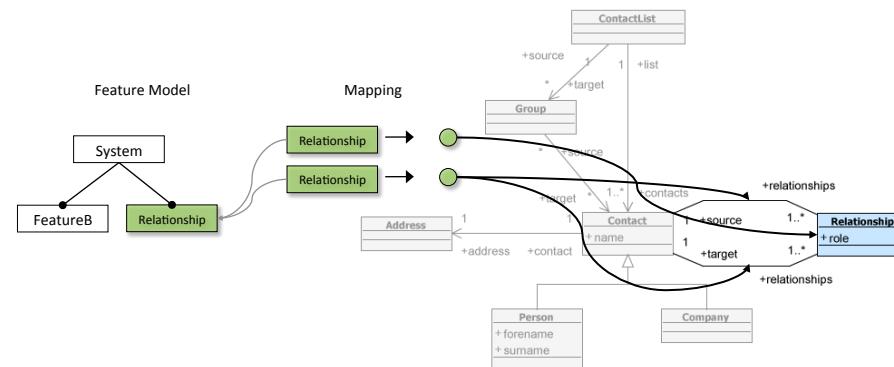
Slide 22

## Visualisation of Mappings (1)

- Visualisations play a crucial role in Software Engineering
  - It's hard/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

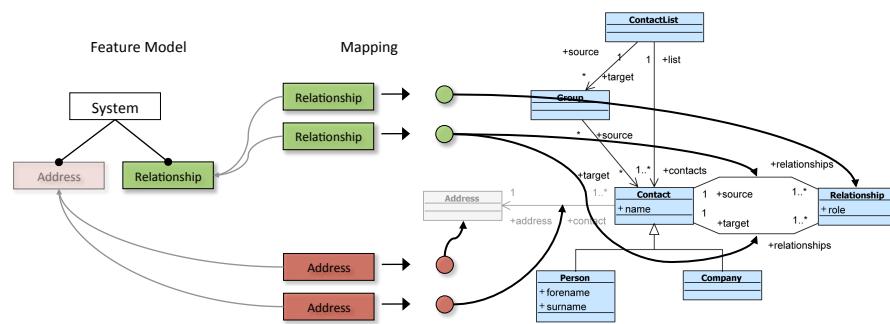
## Realisation View

- For one Variant Model, the realisation in the solution space is shown



## Variant View

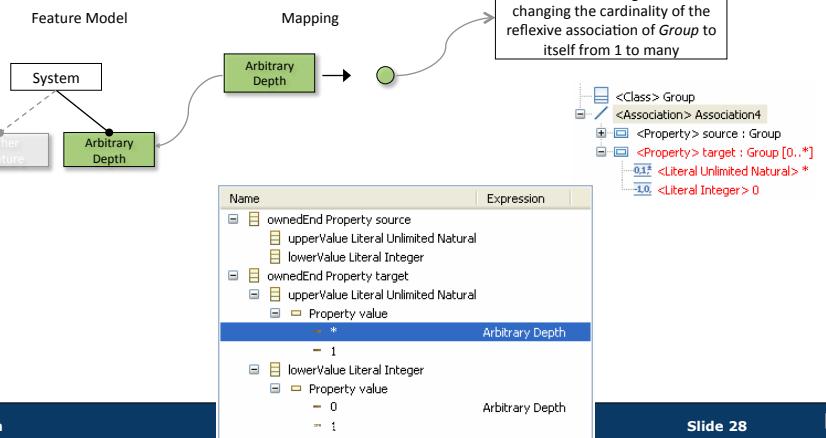
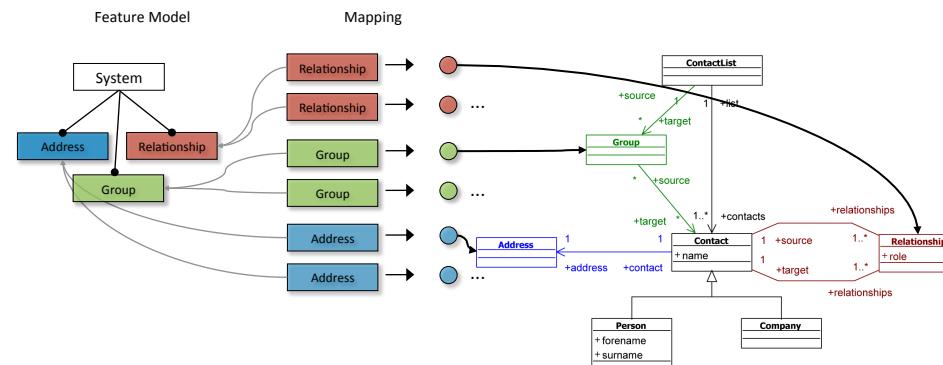
- The variant view shows different realisations in parallel



## Property-Changes View

- The Context View draws the variants with different colors

- Aspect-separation: each variant forms an aspect

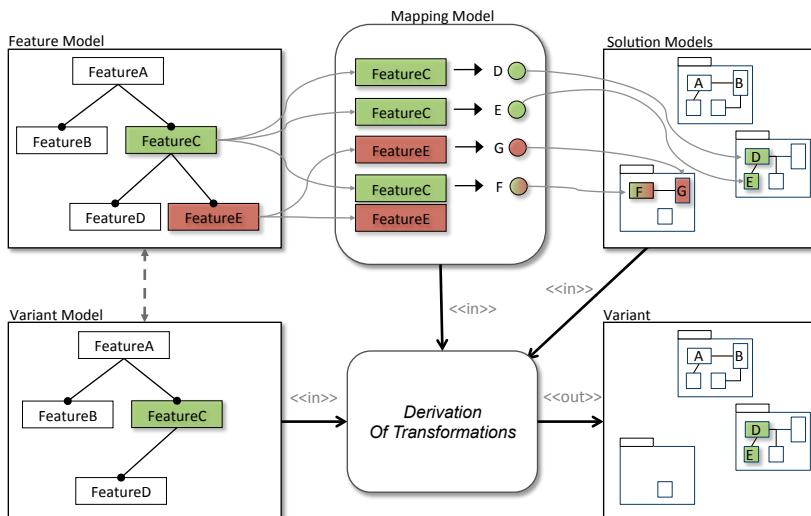


- Unified handling of modelling languages and textual languages by lifting textual languages to the modelling level with the help of EMFText
- All >80 languages from the EMFText Syntax Zoo are supported, including Java 5
- <http://emftext.org>



## Mapping-based Derivation of Transformations

- Transformations in the solution space build the product



- Aspect-related color markup of the code

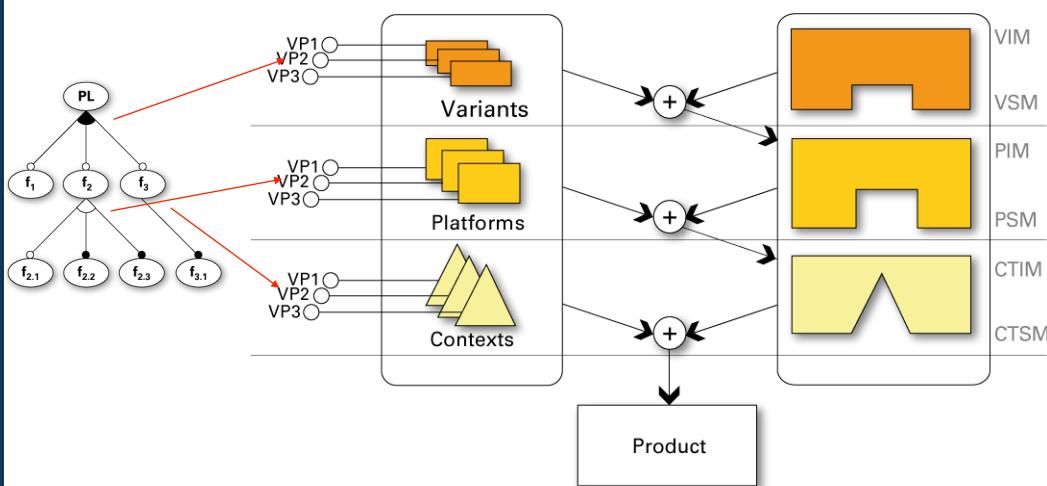
The screenshot shows the Eclipse Platform interface with the following components:

- Project Explorer:** Shows the "ContactManagementJava" project with its package structure and files.
- Mapping View:** Displays the "Contact.java" file with code related to Contact management.
- Outline View:** Shows the class structure of "Contact" with relationships and methods.
- Current Expression:** Shows the expression "Feature Relationships" selected.
- Assigned Feature Expression:** Shows the expression "Feature Relationships" assigned to the code.
- Solution Space View:** Shows the assigned elements for "Feature Relationships".

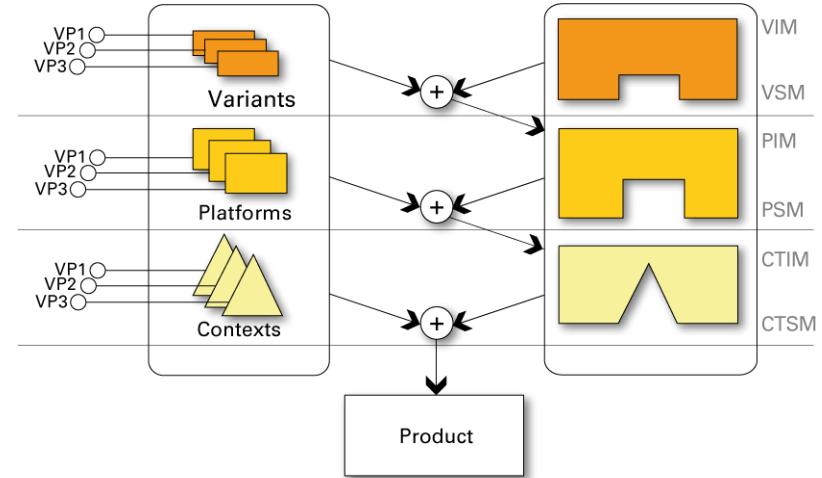
```
*Contact.java
package org.featuremapper.examples.contactmanagement;
import java.util.LinkedHashSet;
import java.util.Set;
public class Contact {
    private String name;
    private Set<Relationship> relationships;
    public Contact(String name) {
        this.name = name;
        this.relationships = 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();
    }
}
```

## 31.3 MULTI-STAGE CONFIGURATION

- Choose one variant on each level
- Feature Tree as input for the configuration of the model weavings

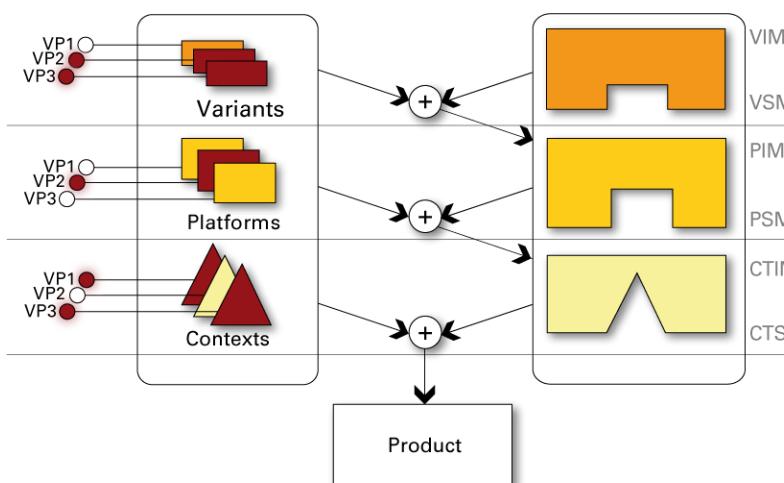


- Goal: a staged MDSD-framework for PLE where each stage produces the software artefacts used for the next stage

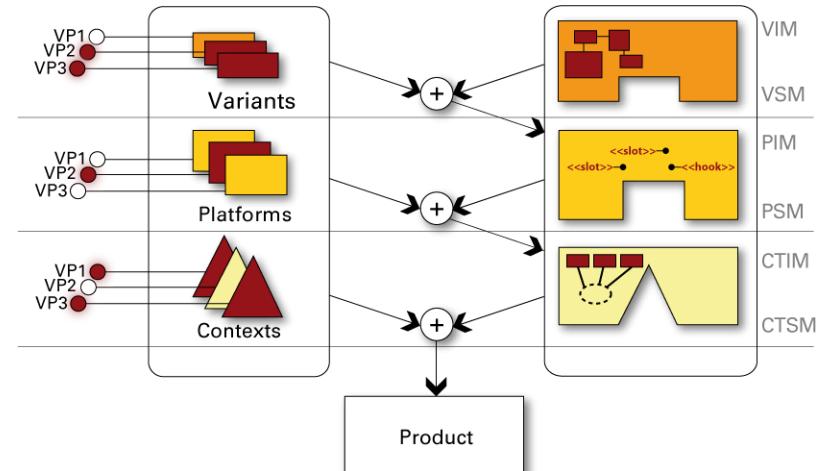


## Advantages of FEASIPLE

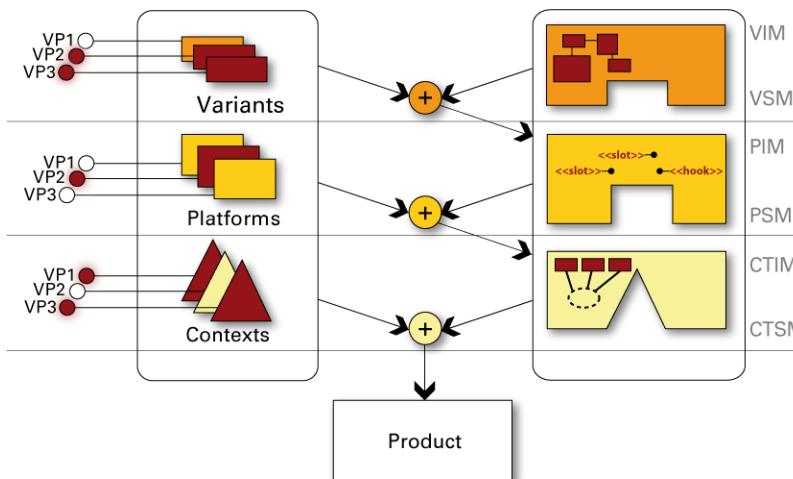
- Characteristic feature 1:
- Variability on each stage



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

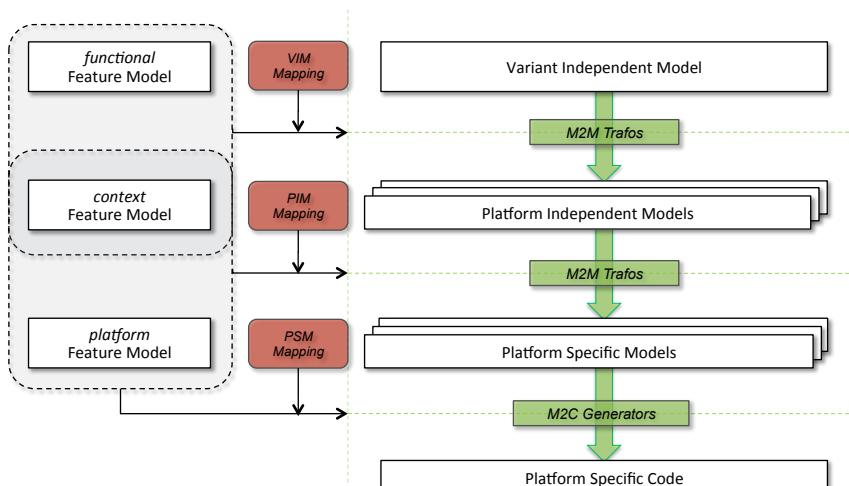


- Characteristic feature 3:
- Different composition mechanisms per stage

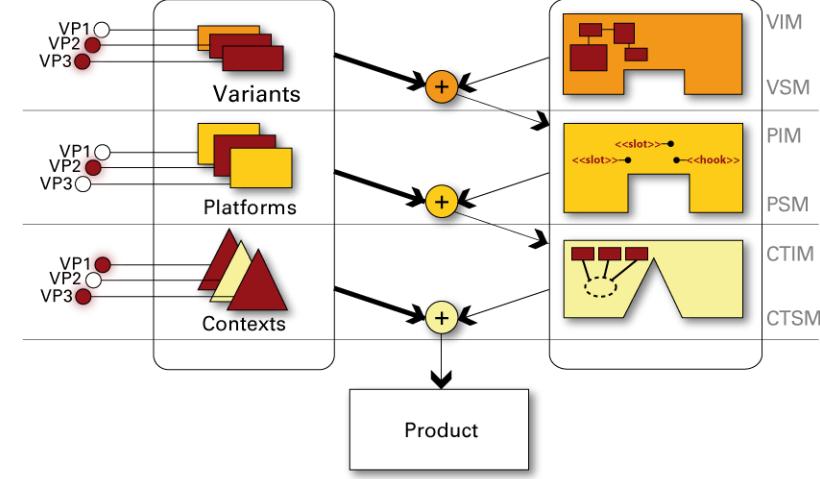


## Multi-Staged Derivation of Transformations

- How do we compose transformations? Between different stages?



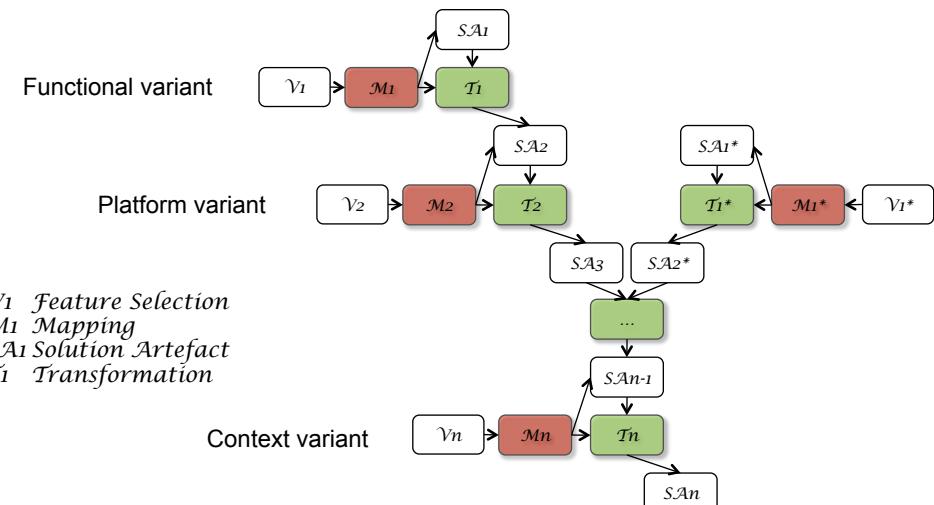
- Characteristic feature 4:
- Composition mechanisms are driven by variant selection



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

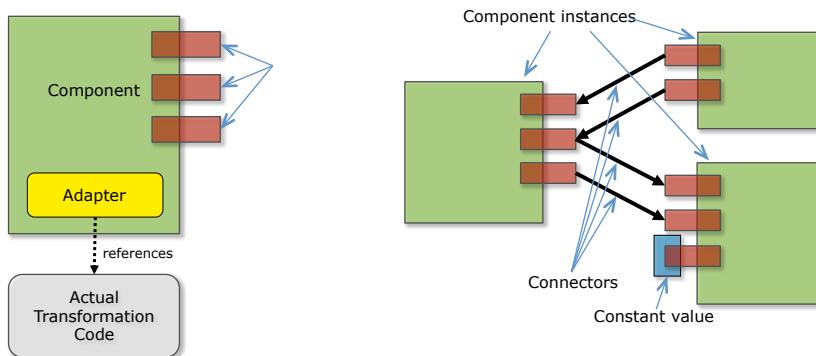
- TraCo encapsulates transformations into composable components

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



$\mathcal{V}_1$  Feature Selection  
 $M_1$  Mapping  
 $SA_1$  Solution Artefact  
 $T_1$  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

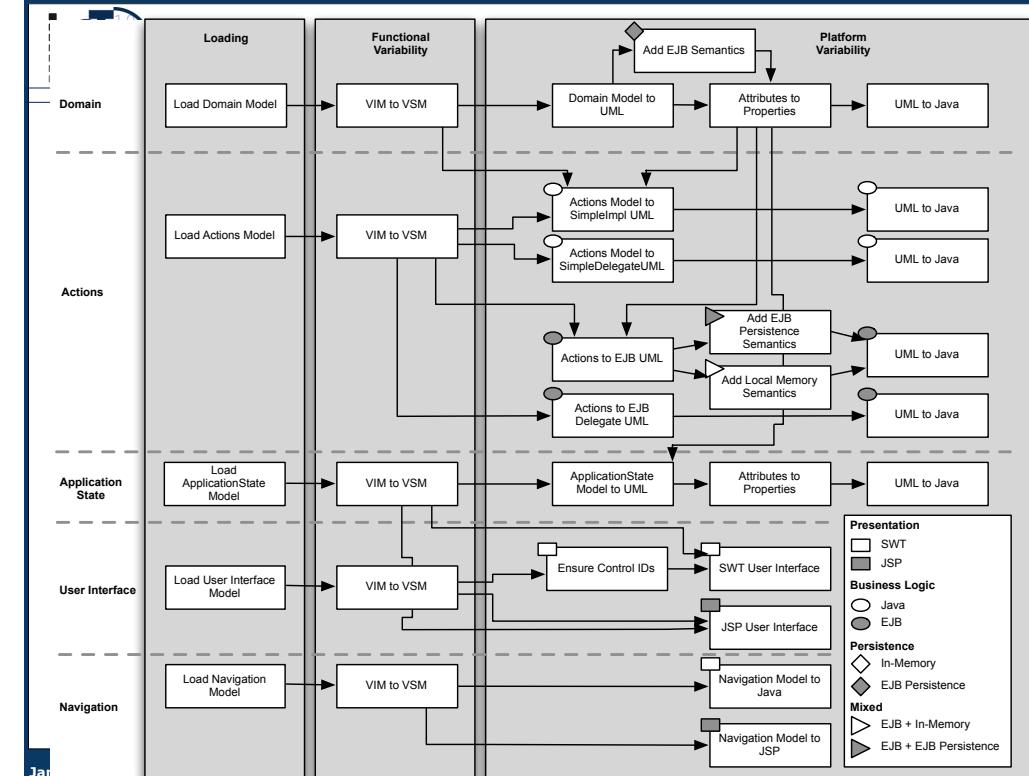
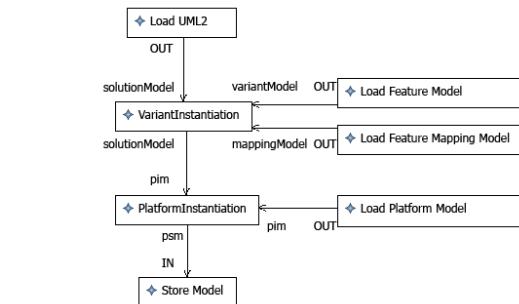
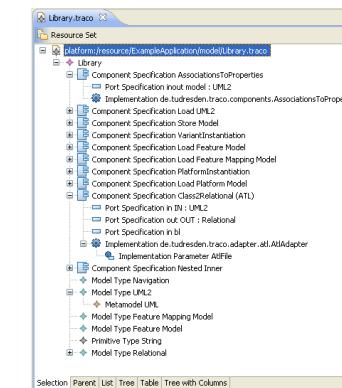


## Composition Programs can be Configured (Metacomposition)

„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

### ➤ Implemented in our tool TraCo

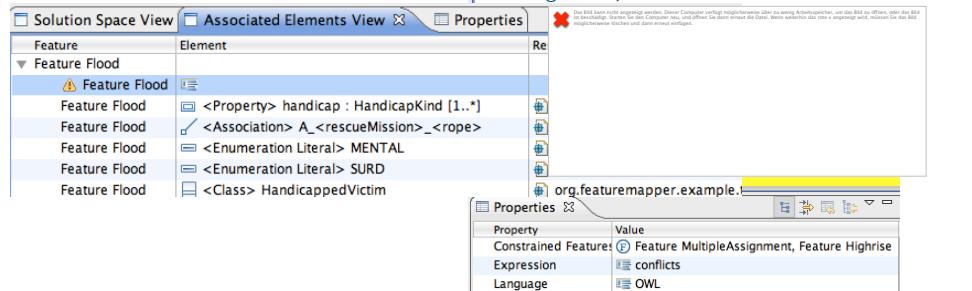


➤ Motivation: Make sure that well-formedness of all participating models is ensured

- Feature Model
- Mapping Model
- Solution Models

➤ Well-formedness rules are described

➤ Constraints are enforced during derivation



## 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

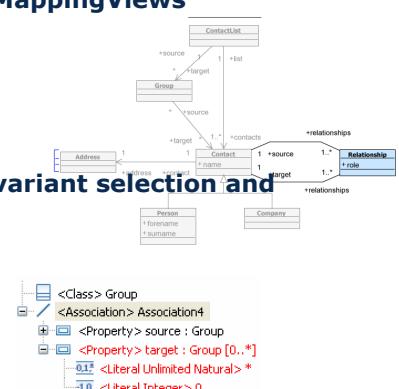
- Realisation View
- Variant View
- Context View
- Property-Changes View

➤ Derivation of solution models based on variant selection and mapping

➤ Multi-Staged derivation using TraCo

➤ Ensuring well-formedness of SPLs

<http://featuremapper.org>



➤ 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.feasible.de>)

➤ TAOSD AOM Crisis Management System

## The End