TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Software- und Multimediatechnik, Lehrstuhl für Softwaretechnologie

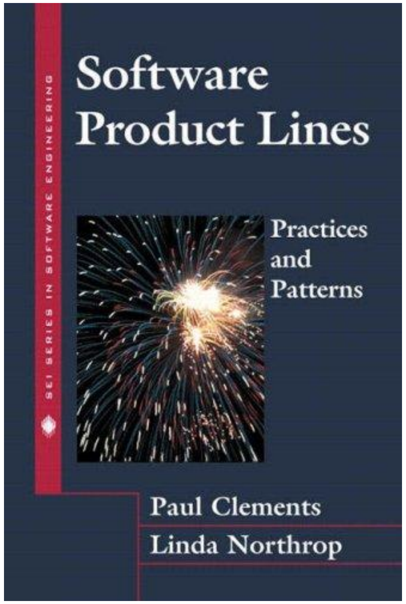
31. Feature Models and MDA for Product Lines

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
- Wintersemester 2014/15, 25. Januar 2015

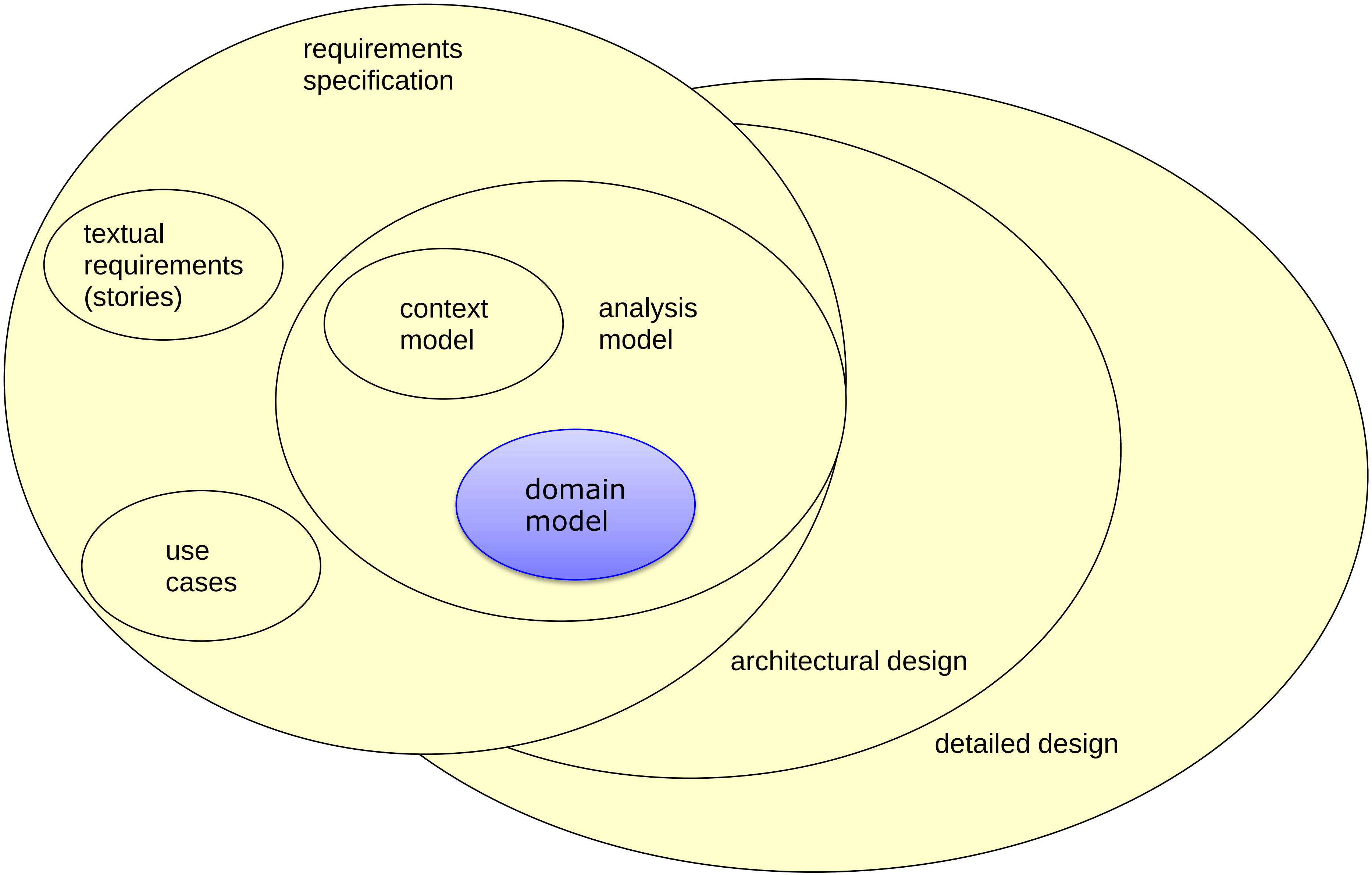
- **Lecturer**: Dr. Sebastian Götz

- 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://featuremapper.org/files/ICSE08-FeatureMapper--Mapping-Features-to-Models.pdf
- Paul C. Clements, Linda M. Northrop: **Software Product Lines: Practices and Patterns.** Addison-Wesley Professional, 2001.
- Sven Apel, Don Batory, Christian Kästner, Gunter Saake: **Feature-Oriented Software Product Lines – Concepts and Implementation.** Springer, 2013.



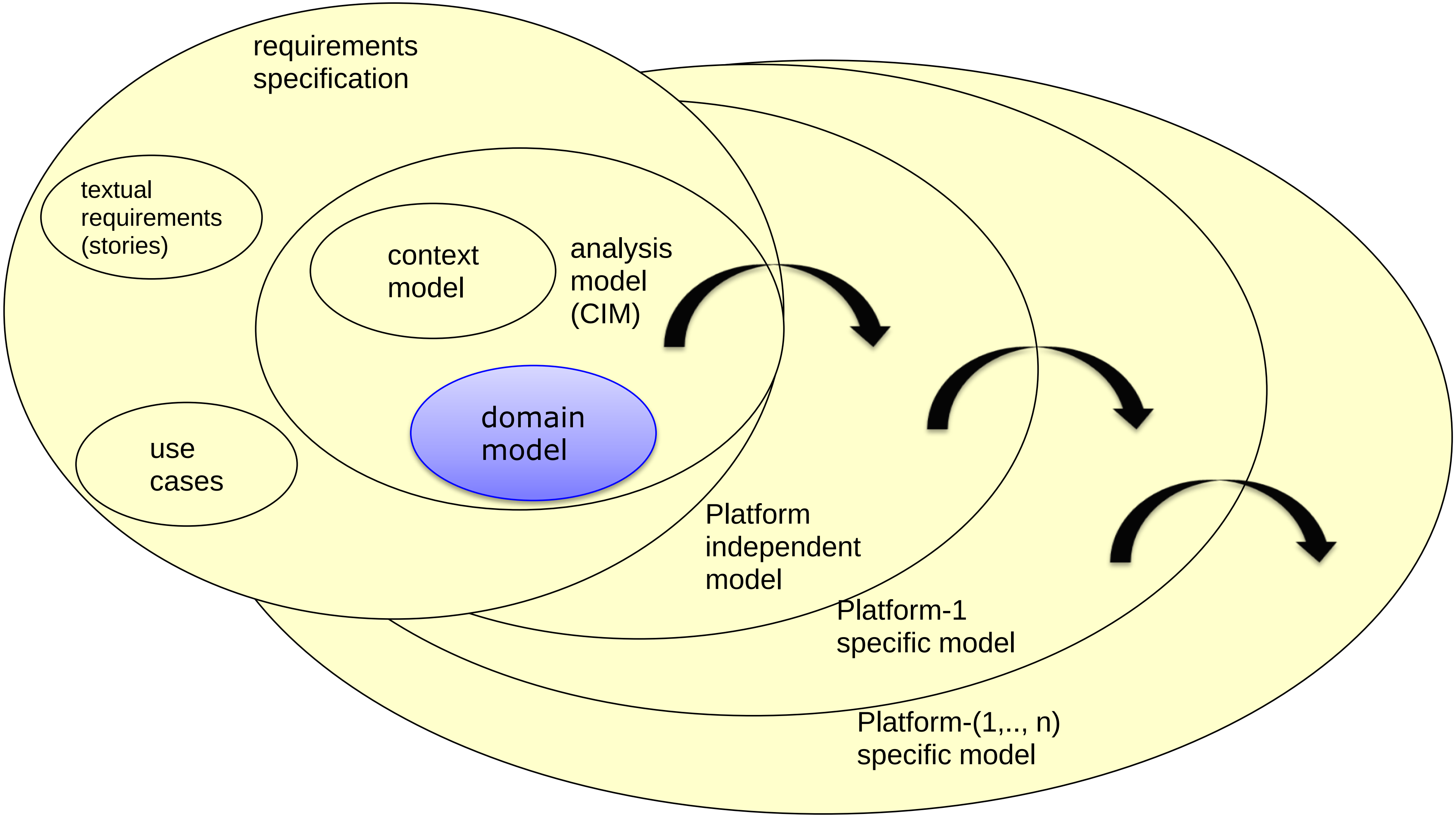
- [Aßm03] U. Aßmann. Invasive Software Composition. Springer, 2003.
- [Cza05] K. Czarnecki and M. Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In R. Glück and M. Lowry, editors, Proceedings of the 4th International Conference on Generative Programming and 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 Şavga 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 (ViSPLE 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

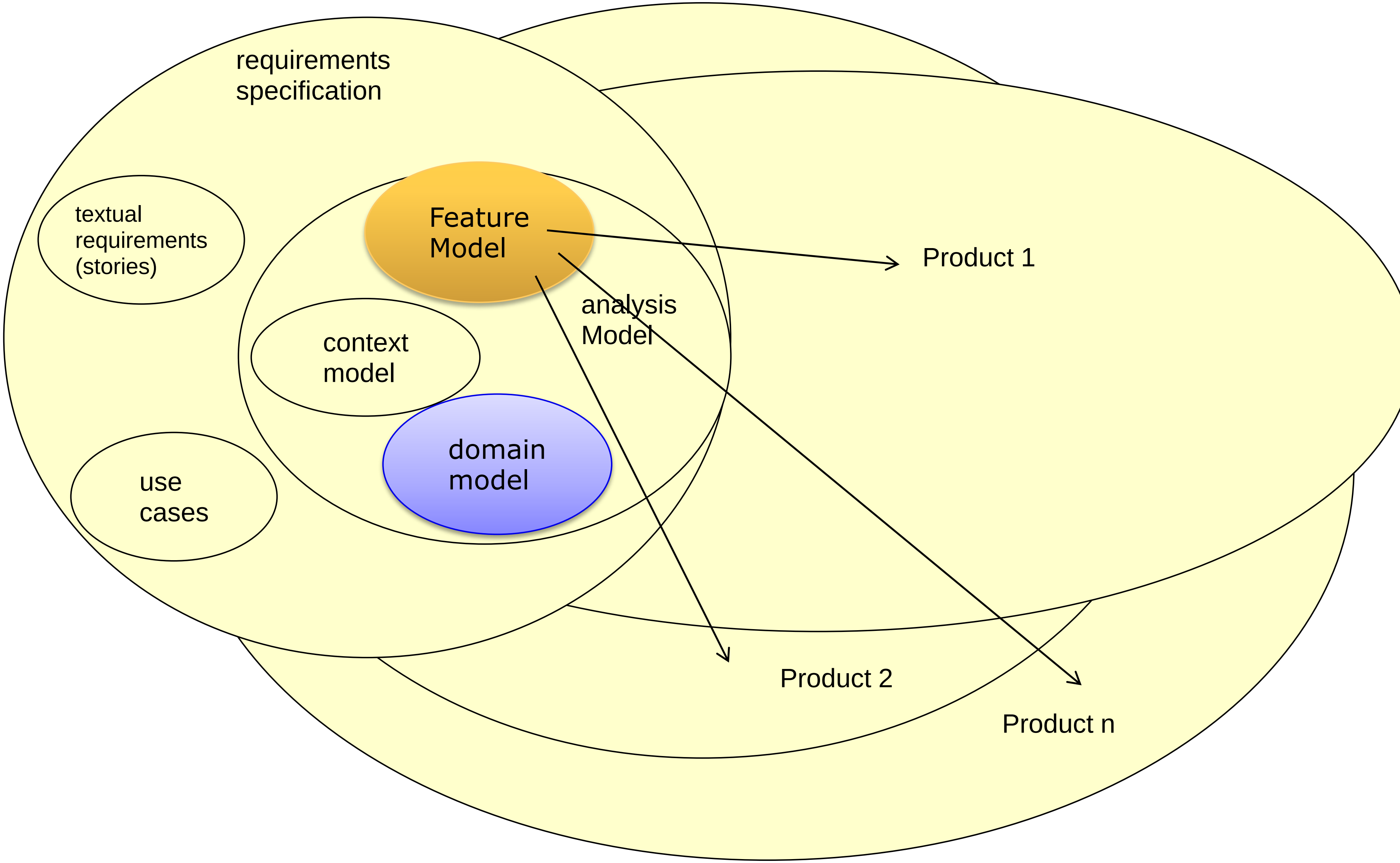
Object-Oriented Analysis vs Object-Oriented Design



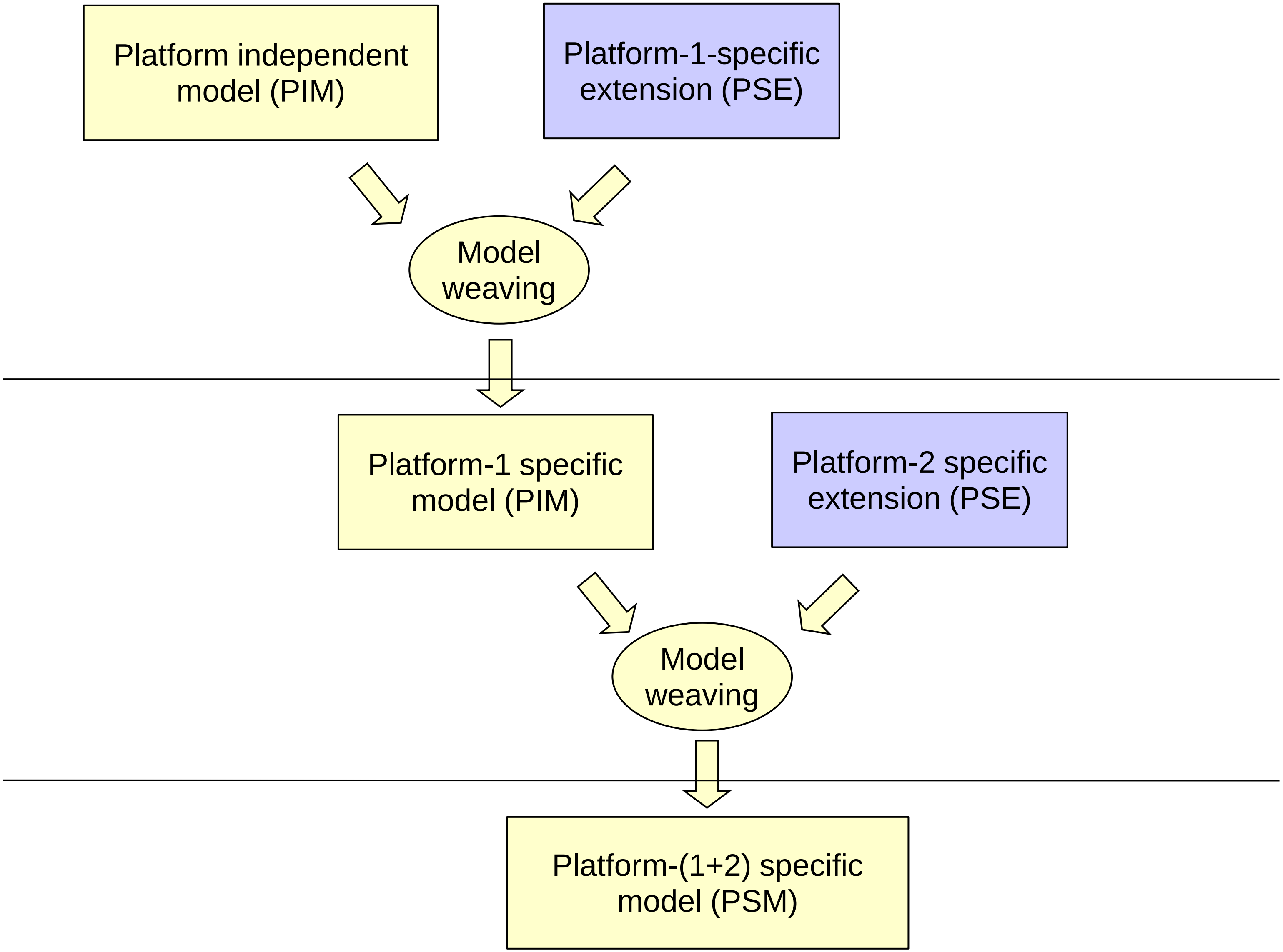
Extended to Model-Driven Architecture (MDA)

- **Horizontal product line: one product idea in several markets**

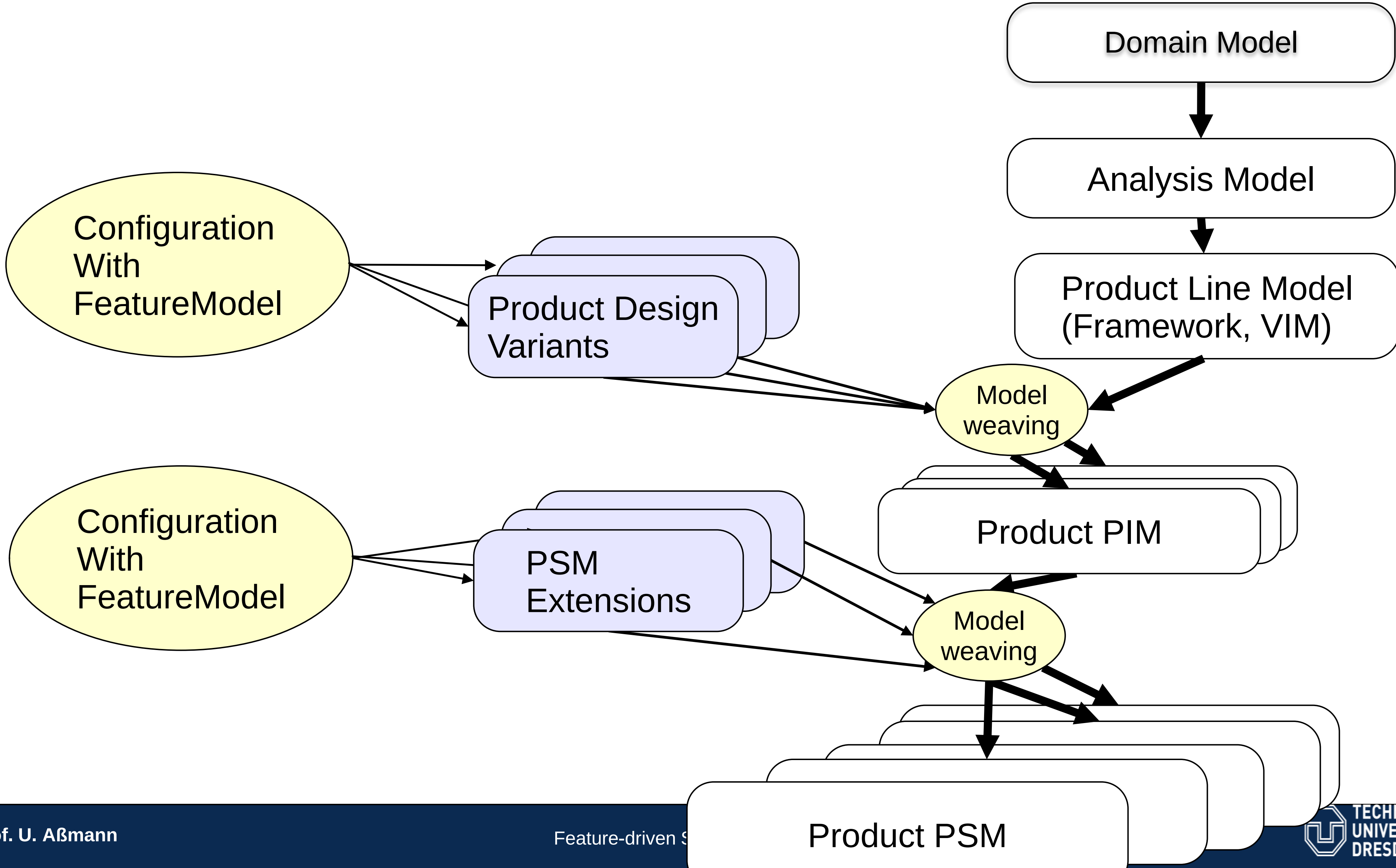



requirements
specification
textual
requirements
(stories)
Feature
Model
context
model
analysis
Model
domain
model
use
cases
Product 1
Product 2
Product n

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

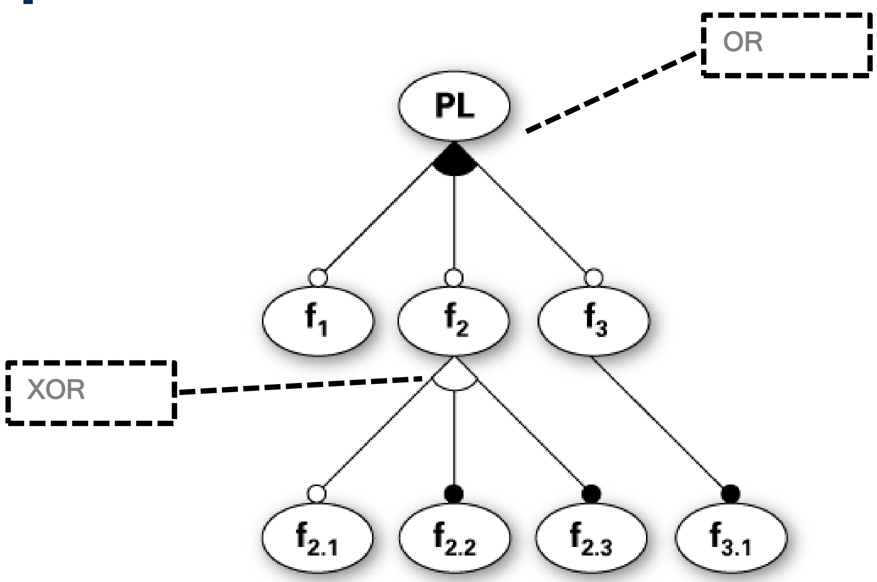


- **Vertical product line: several products in one or several markets**
  - The VIM (variant independent model) is the common model of the product family



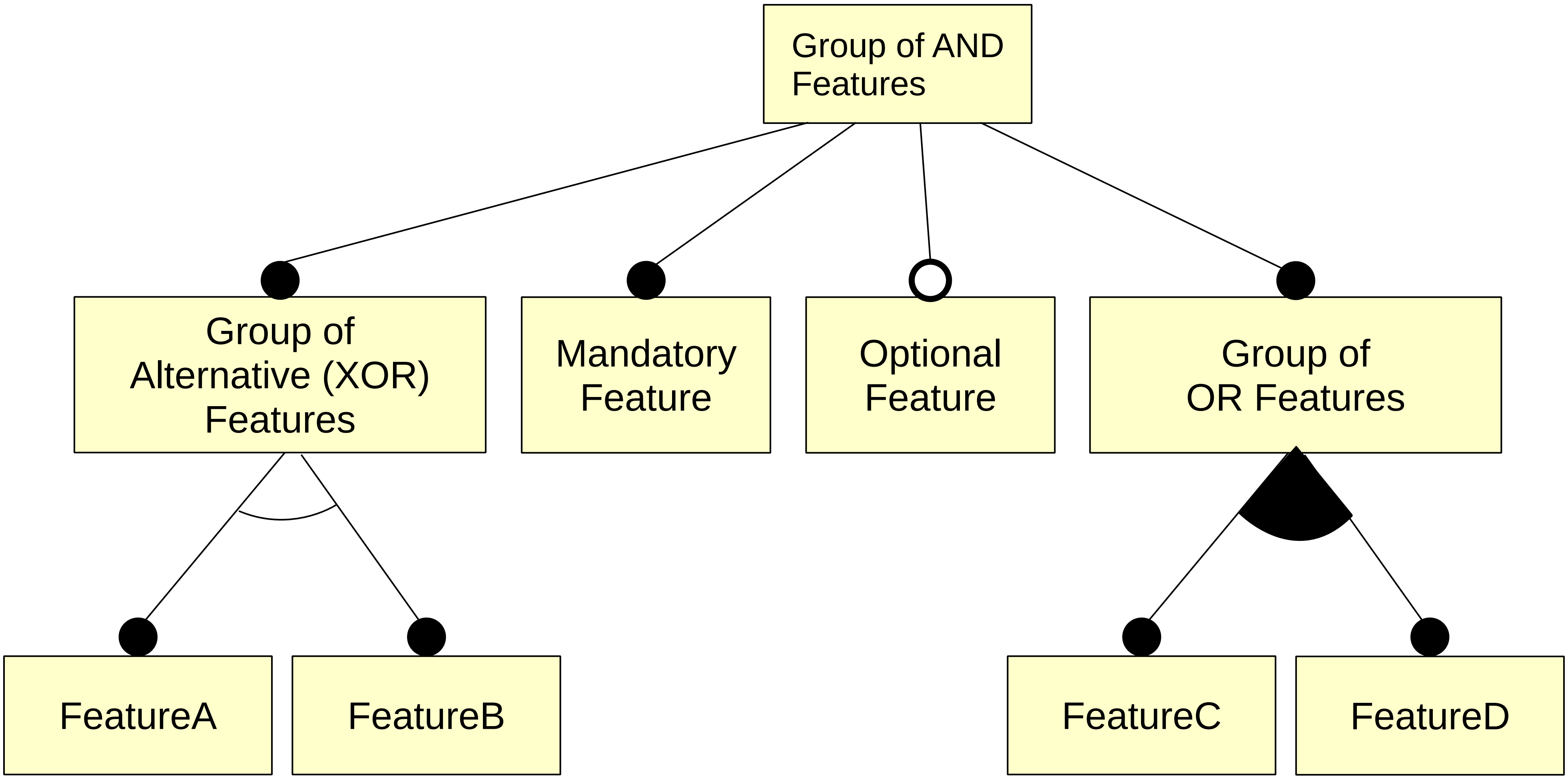
31.1 PRODUCT LINES WITH FEATURE TREES AND FEATURE MODELS

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



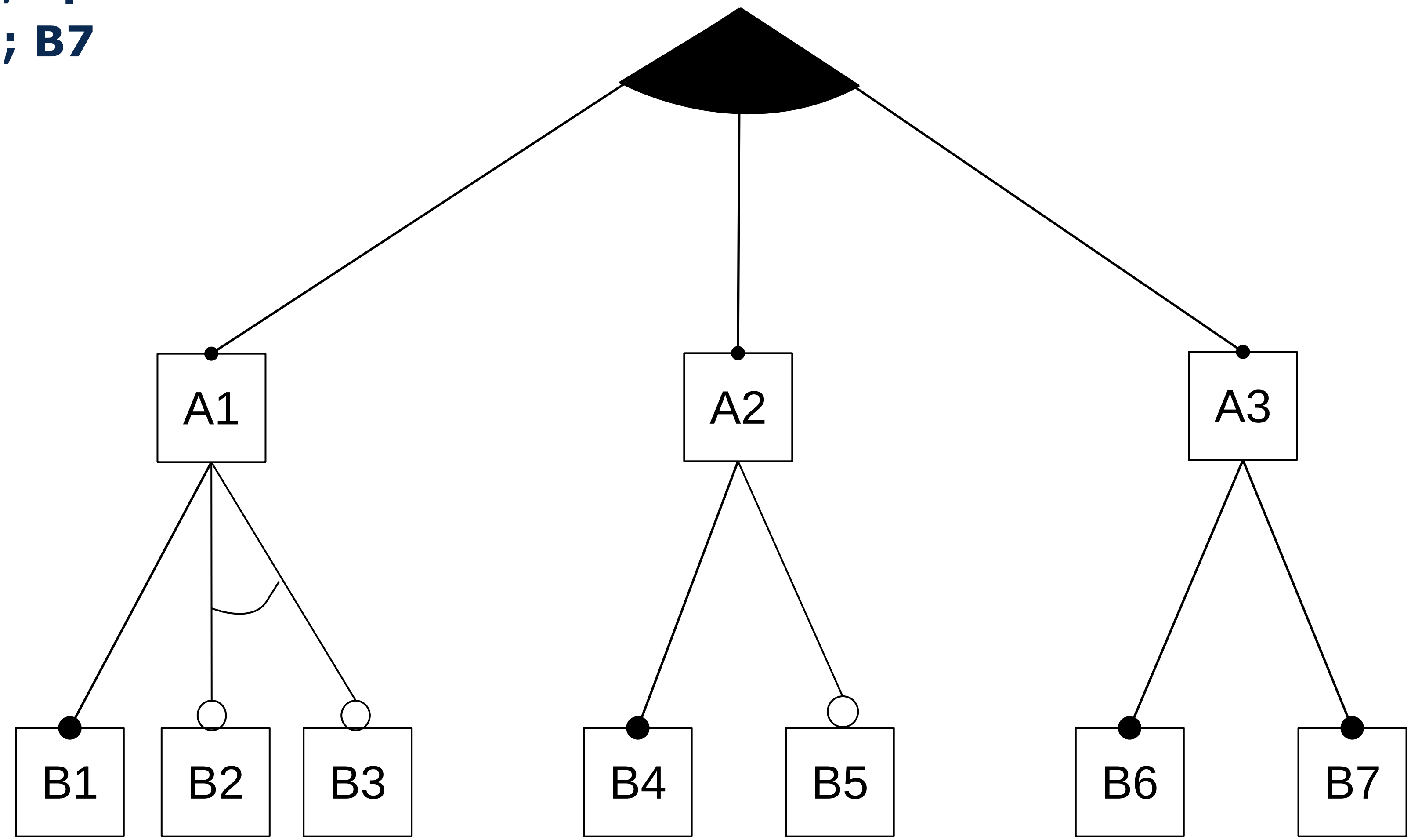
- **A *variant model* represents a concrete product (variant) from the product line**
  - The variant model results from a selection of a subgraph of the feature model
  - The variant model can be used to parameterize and drive the product instantiation process

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



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

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



Mapping Features to Model Fragments (Model Snippets)

- **Bridging the gap between configuration and solution space**

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

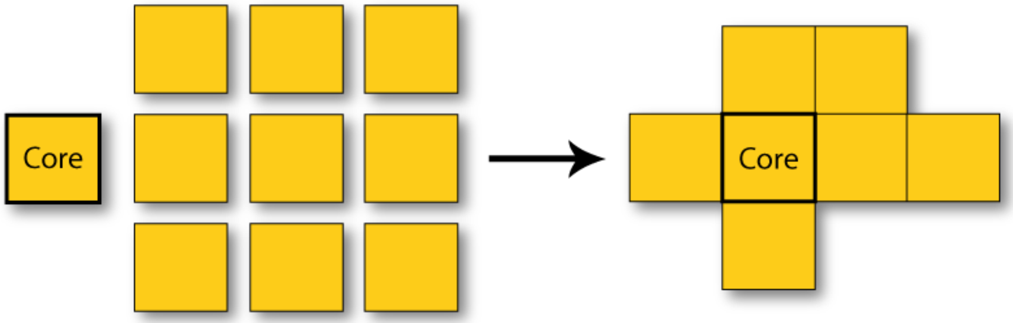
- **Possible artifacts**
  - Models defined in DSLs
  - Model fragments (snippets)
  - Architectural artifacts (components, connectors, aspects)
  - Source code
  - Files

- **But how can we achieve the mapping… ?**

31.2 PRODUCT-LINE CONFIGURATION WITH FEATURE MODELS

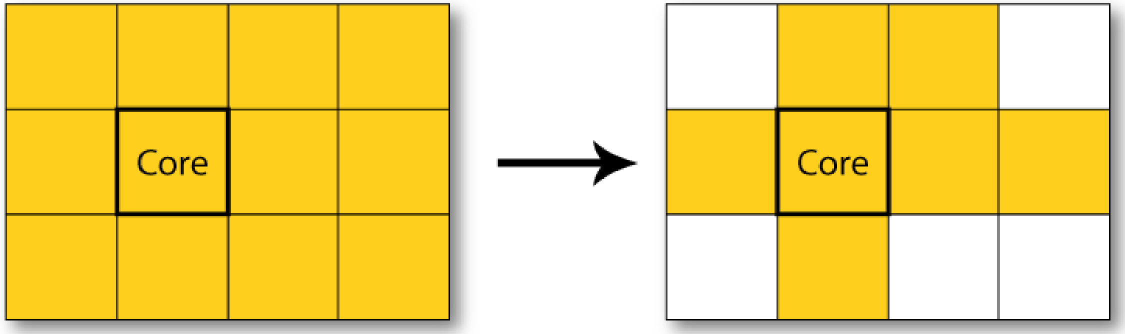
Additive approach

- **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**

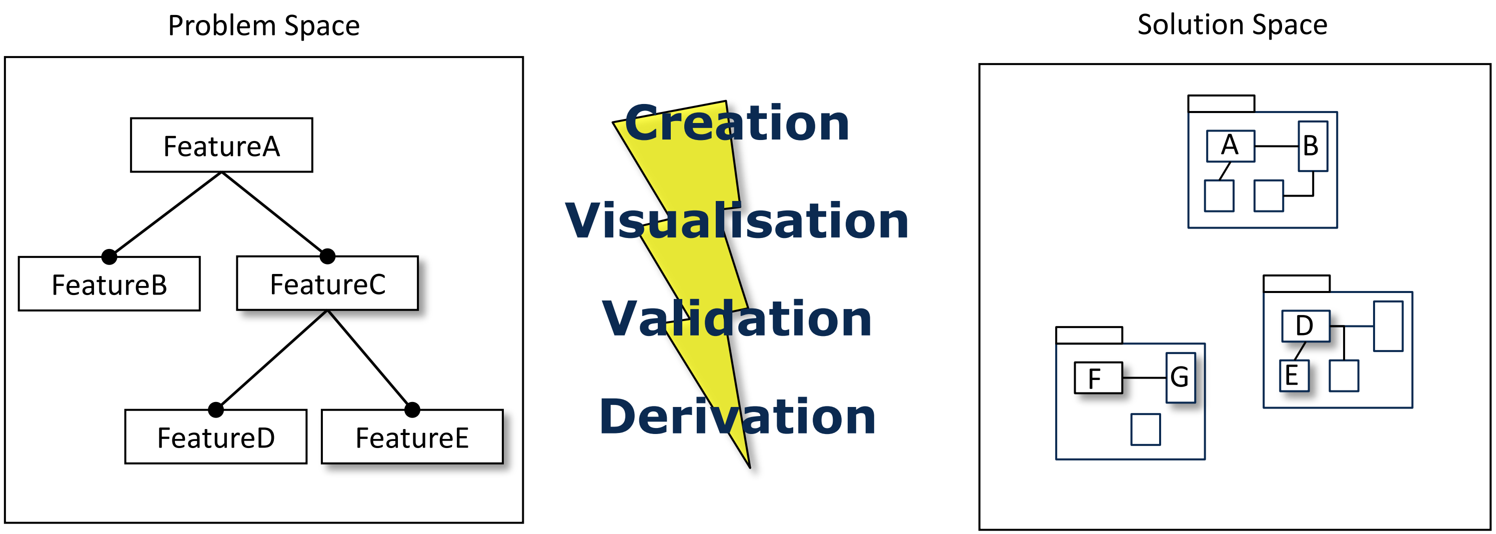


- Pros:
  - conflicting variants can be modeled correctly
  - strong per-feature decomposition
- Cons:
  - traceability problems
  - increased overhead in linking the different fragments

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



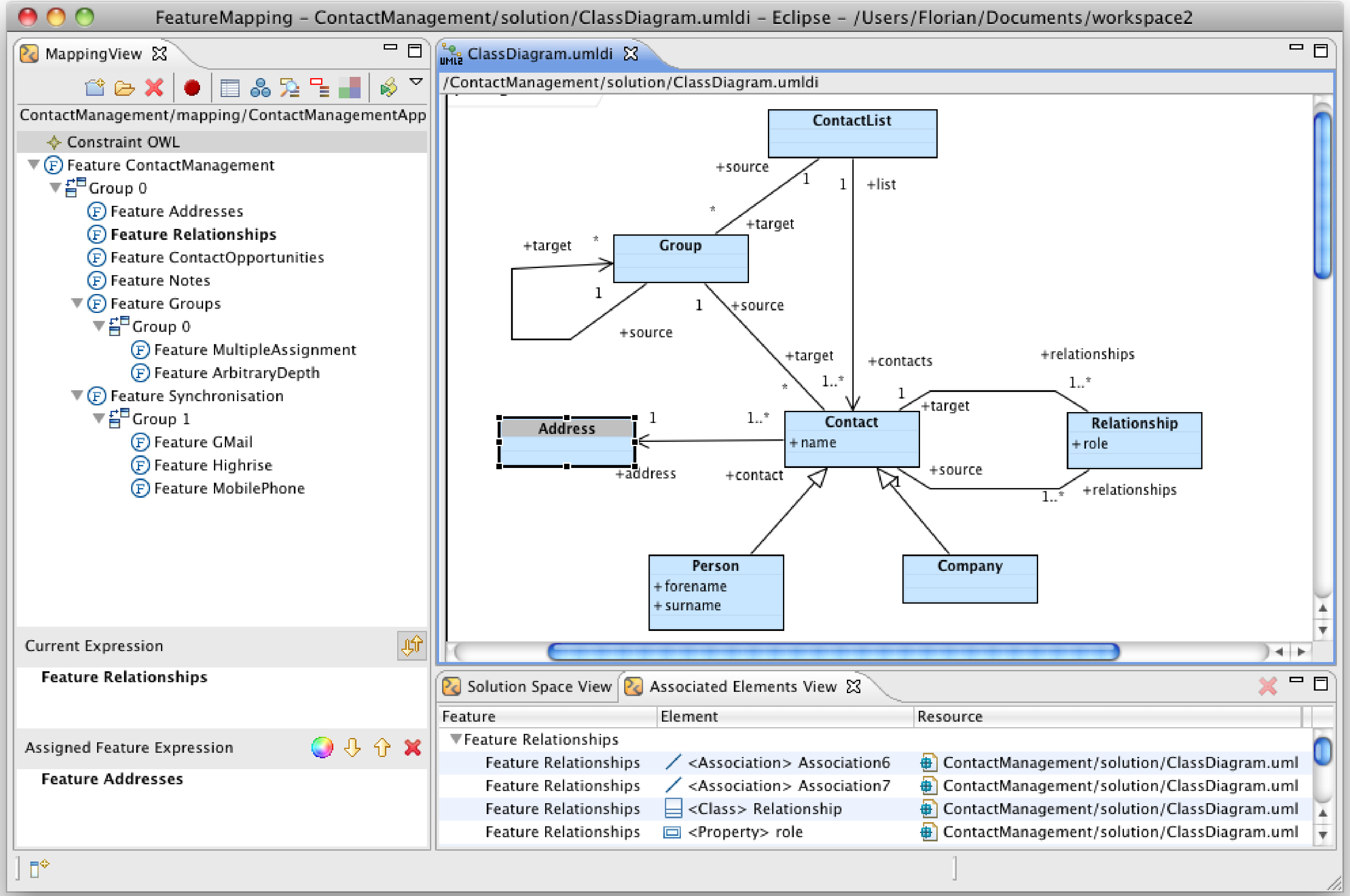
- Pros:
  - no need for redundant links between artifacts
  - short cognitive distance
- Cons:
  - conflicting variants can't be modeled correctly
  - huge and non-concise models


Problem Space
FeatureA
FeatureB
FeatureC
FeatureD
FeatureE
Creation
Visualisation
Validation
Derivation
Solution Space
A
B
D
E
F
G

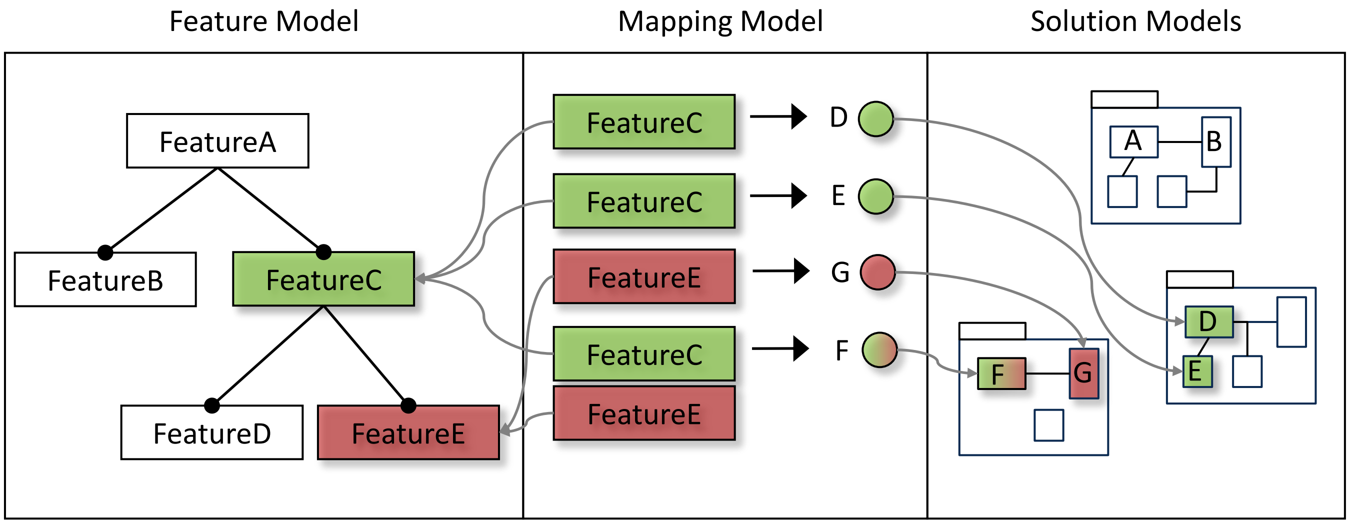
- **FeatureMapper** - a tool for mapping of feature models to modeling artifacts developed at the ST Group
- Screencast and paper available at **http://featuremapper.org**

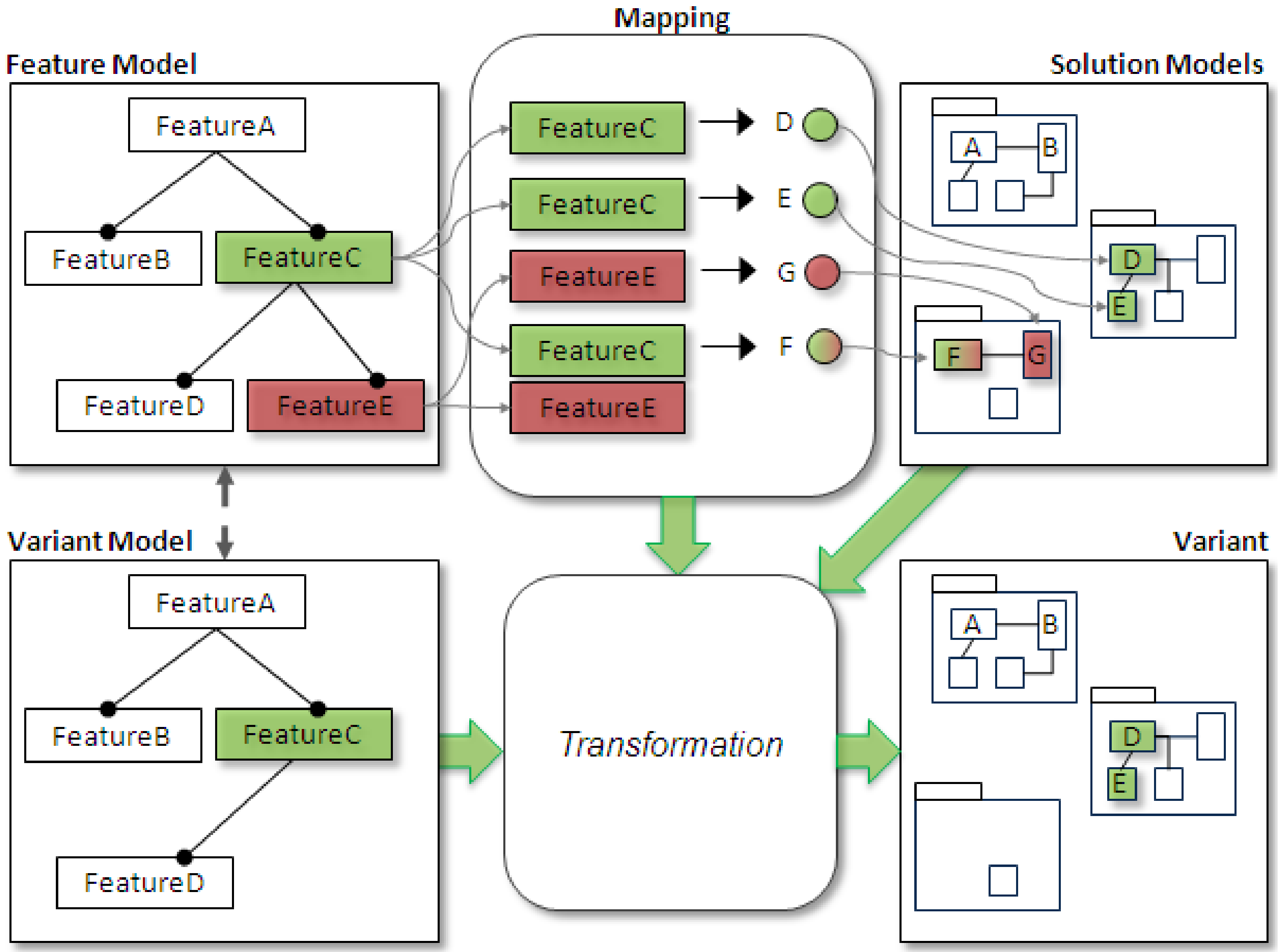
- **Advantages:**
  - Explicit representation of mappings
  - Configuration of large product lines from selection of variants in feature trees
    - Customers understand
  - Consistency of each product in the line is simple to check
  - Model and code snippets can be traced to requirements

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**

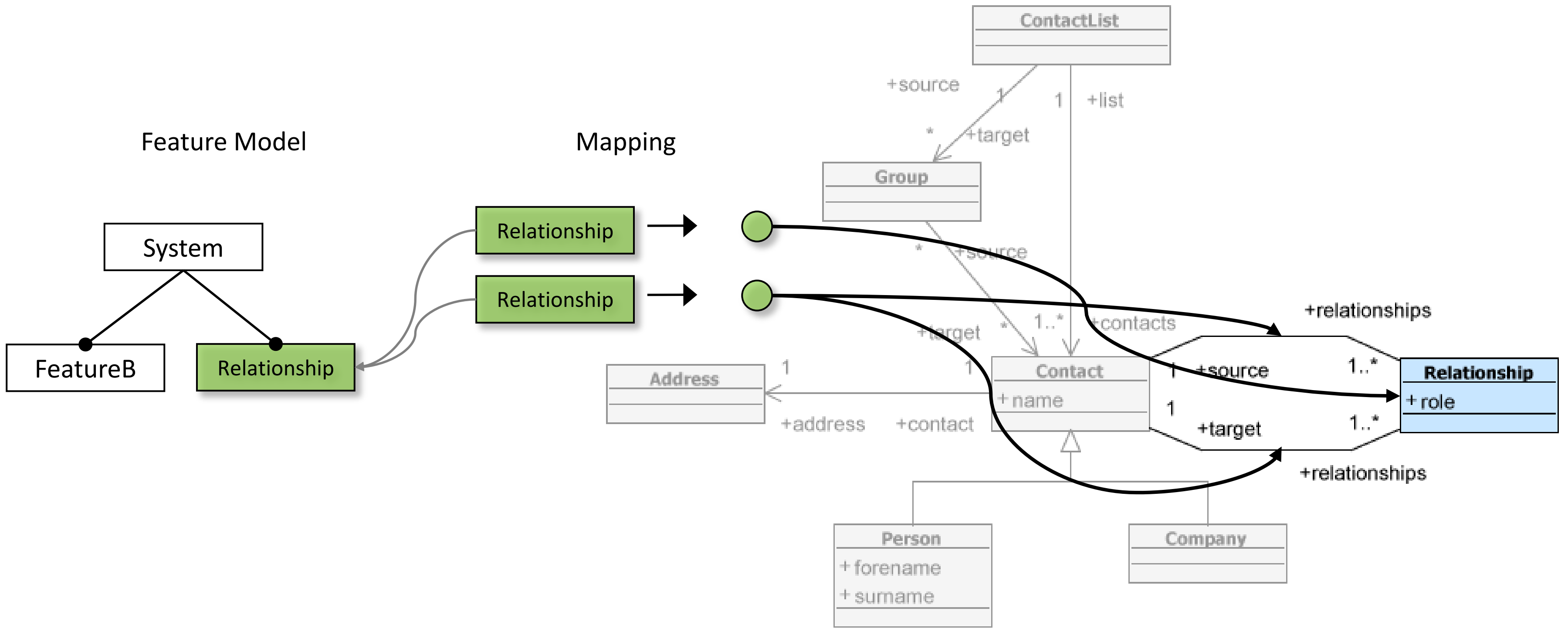




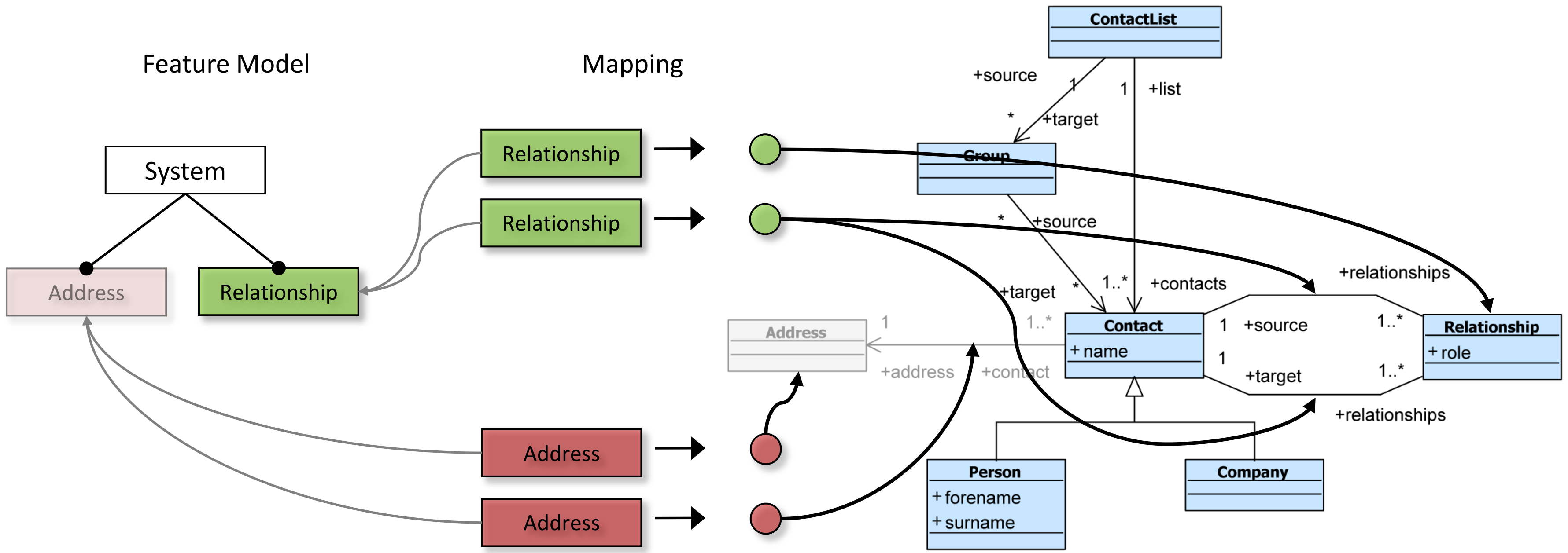
Mapping
Feature Model
Solution Models
FeatureA
FeatureB
FeatureC
FeatureD
FeatureE
FeatureC
FeatureC
FeatureE
FeatureC
FeatureE
D
E
G
F
A
B
D
E
F
G
Variant Model
Variant
FeatureA
FeatureB
FeatureC
FeatureD
Transformation
A
B
D
E

- **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

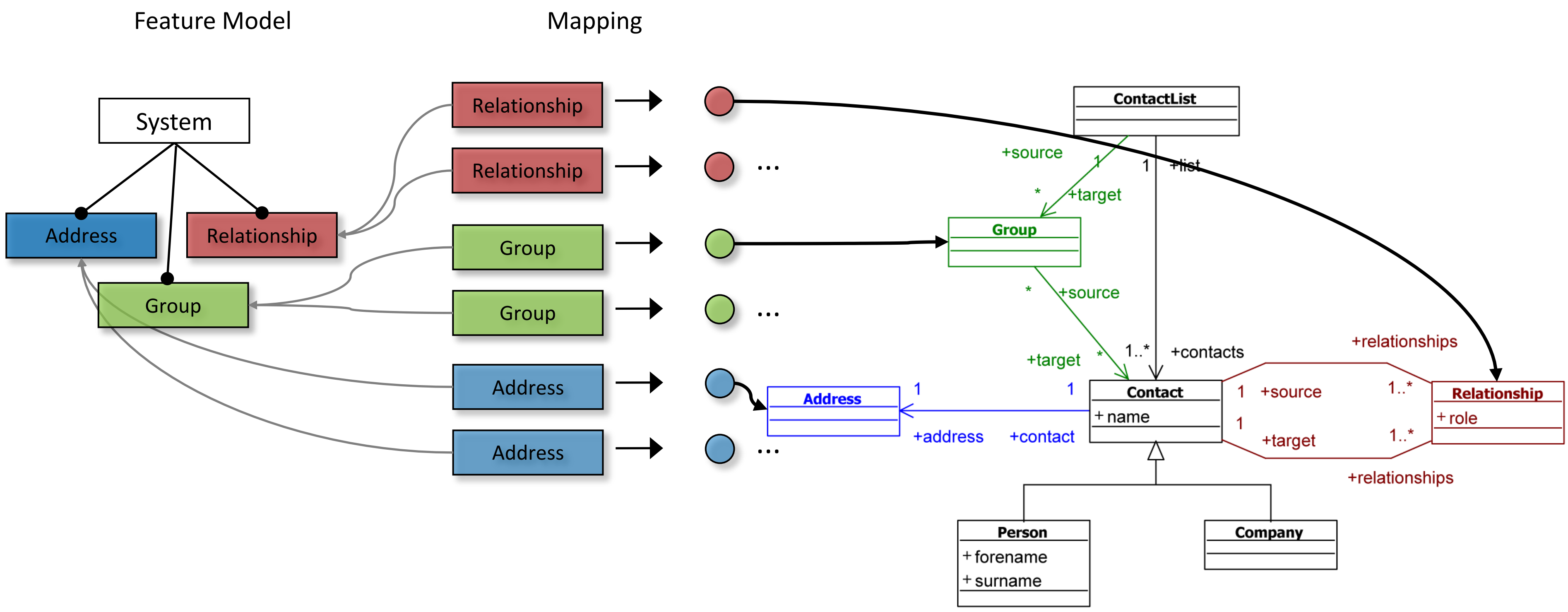
- For one Variant Model, the realisation in the solution space is shown



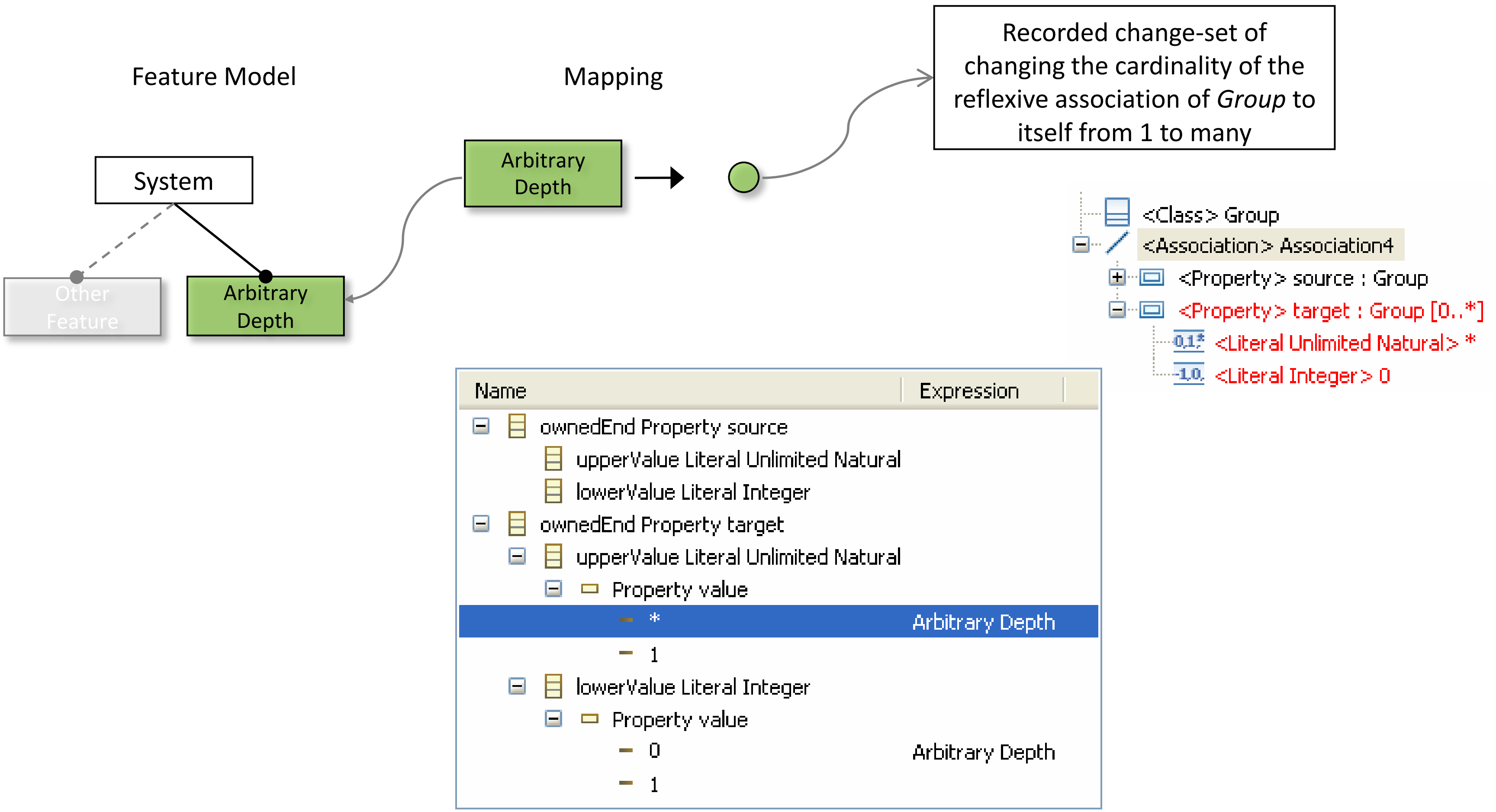
- **The variant view shows different variant realisations (variant models) in parallel**



- **The Context View draws the variants with different colors**
  - Aspect-separation: each variant forms an aspect



Property-Changes View



Textual Languages Support (1)

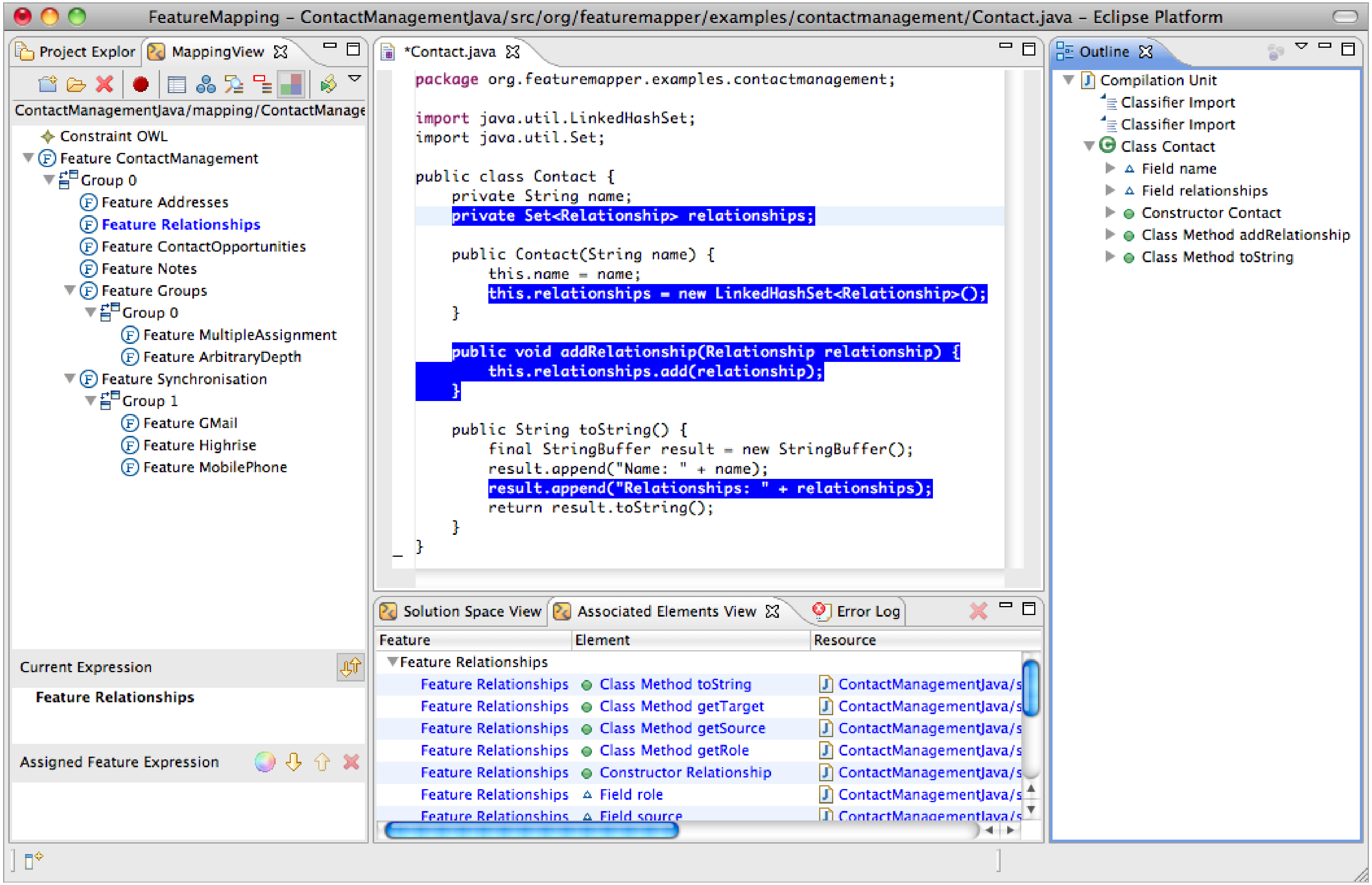
- **Unified handling of modeling languages and textual languages by lifting textual languages to the modelling level with the help of EMFText**

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

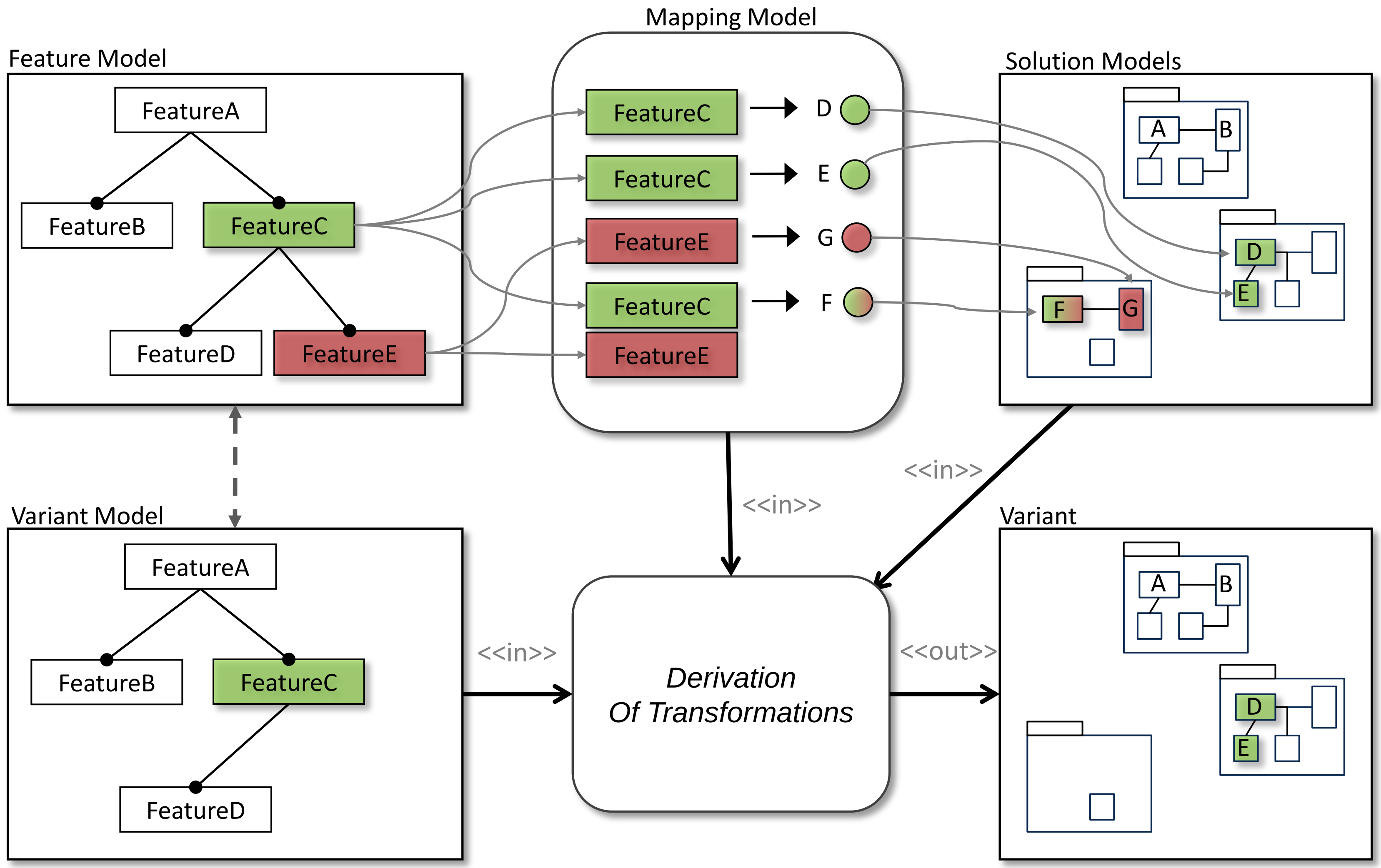
- **http://emftext.org**

emftext
concrete syntax mapper

- **Aspect-related color markup of the code**

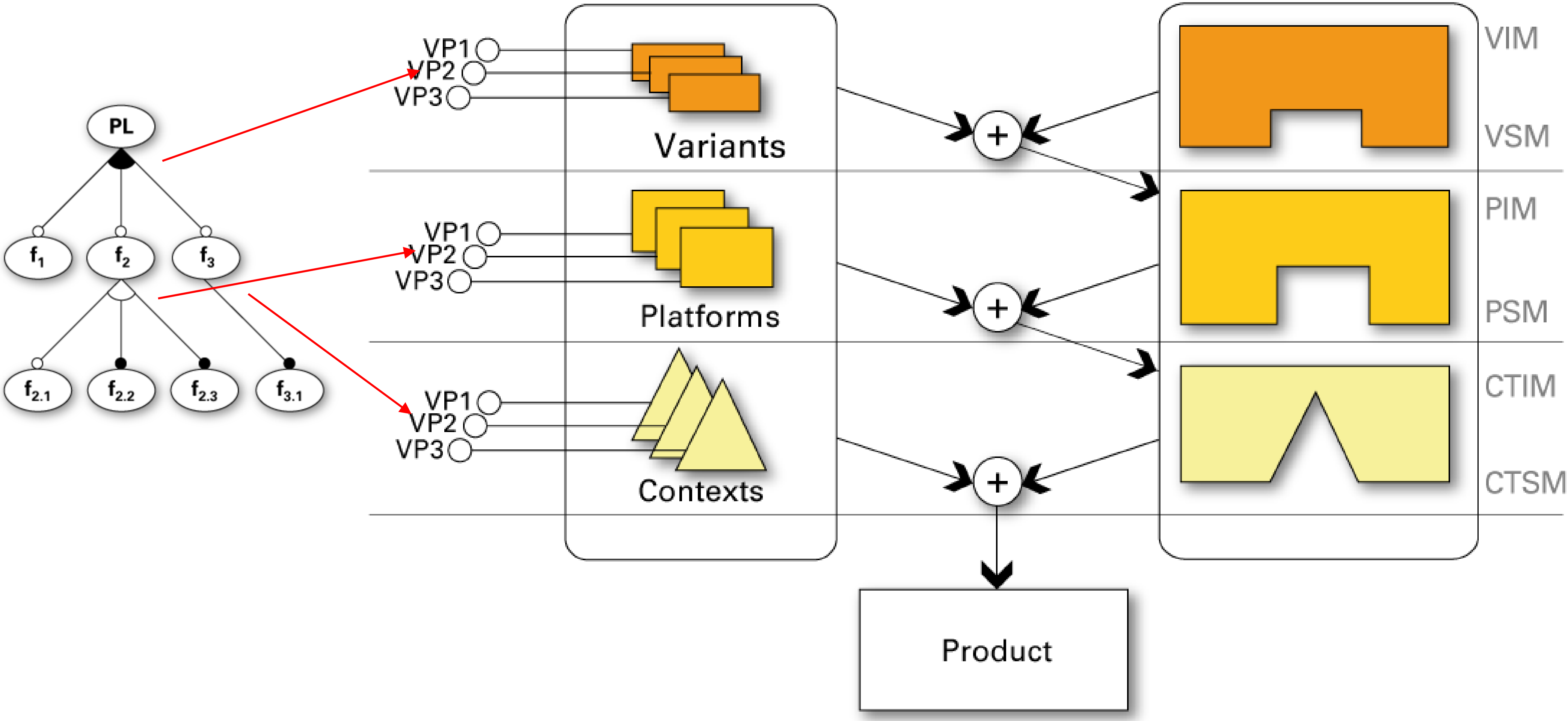


- **Transformations in the solution space build the product**

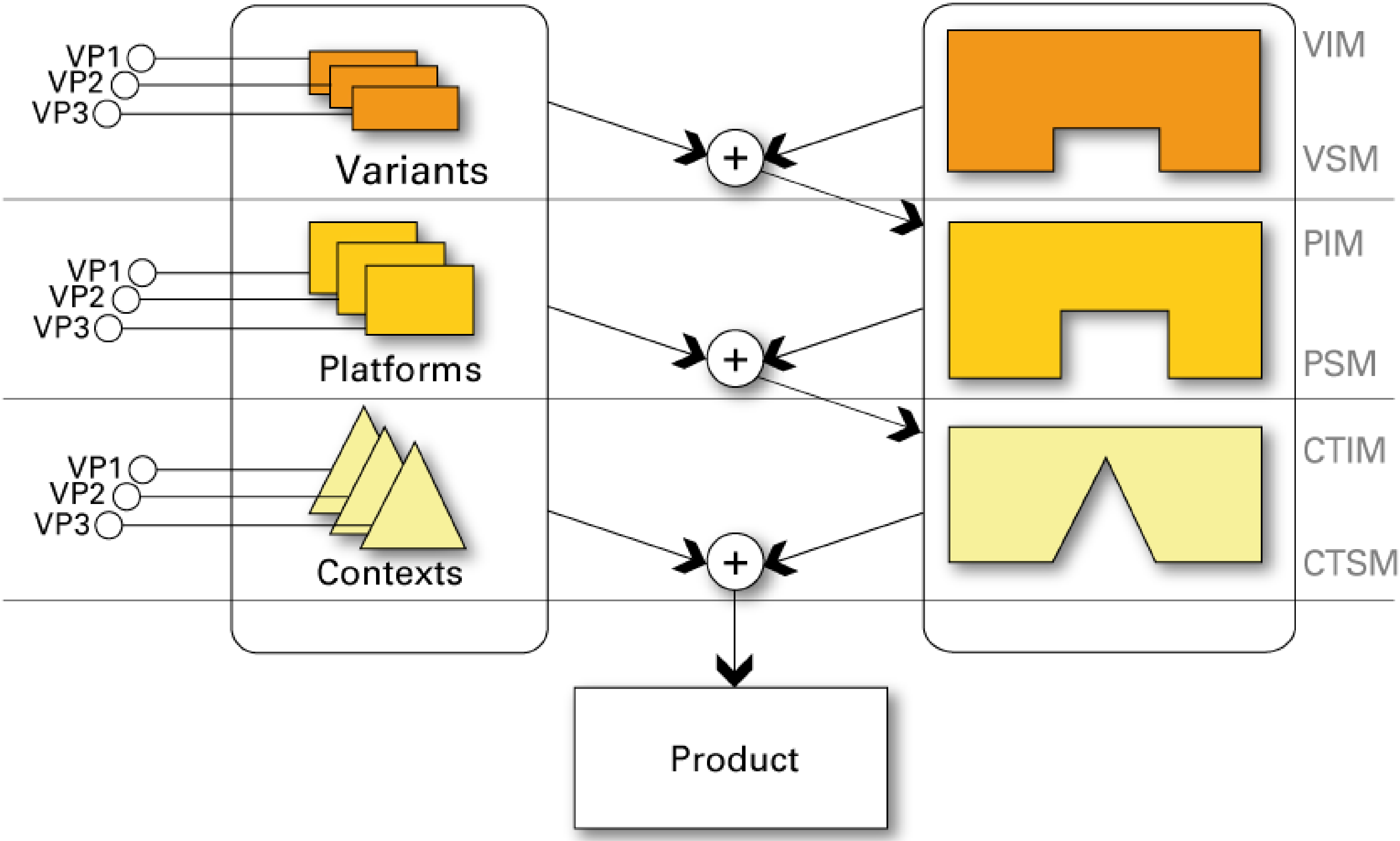


31.3 Multi-Stage Configuration

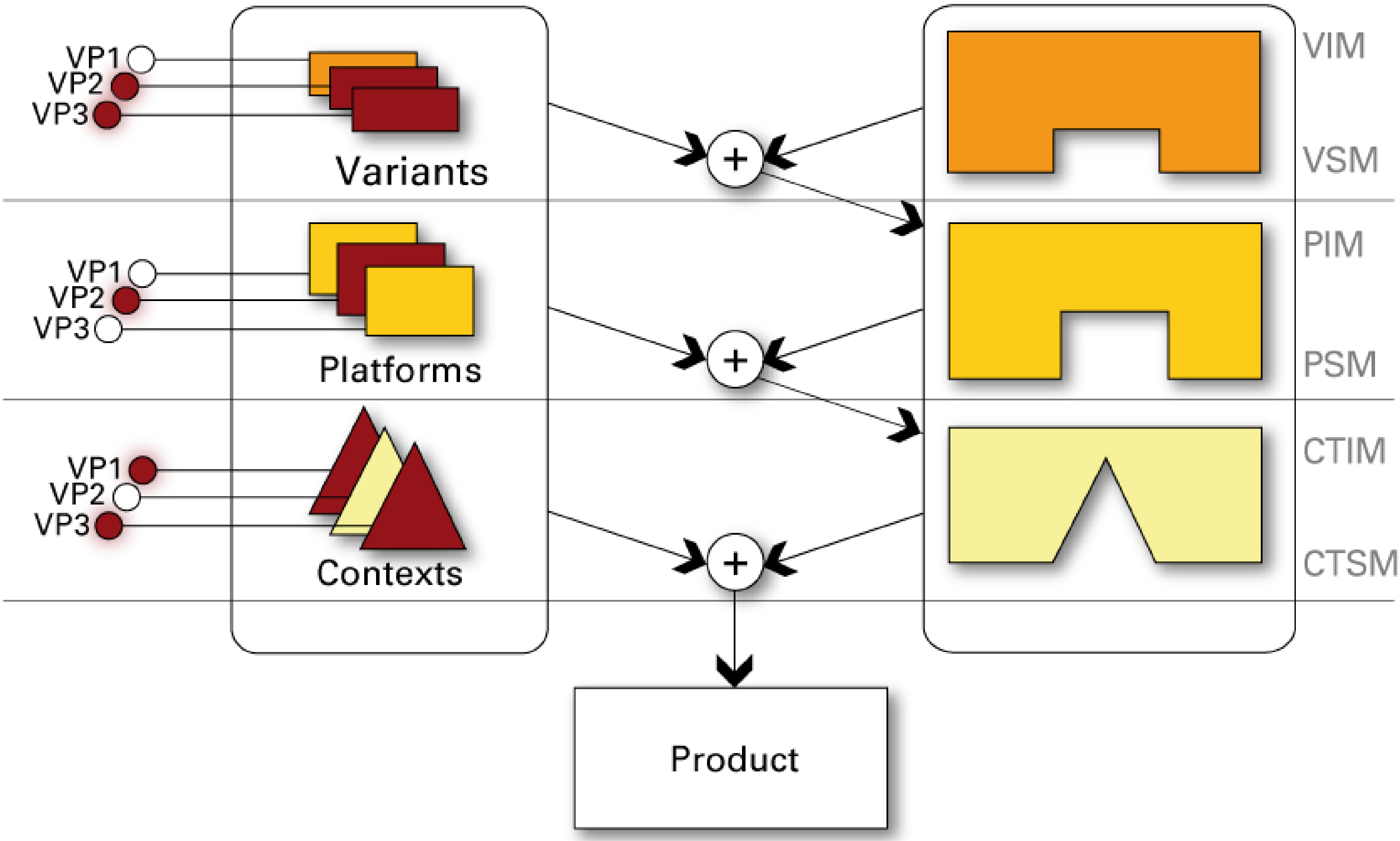
- **Chose one variant on each level**
- **Feature Tree as input for the configuration of the model weavings**



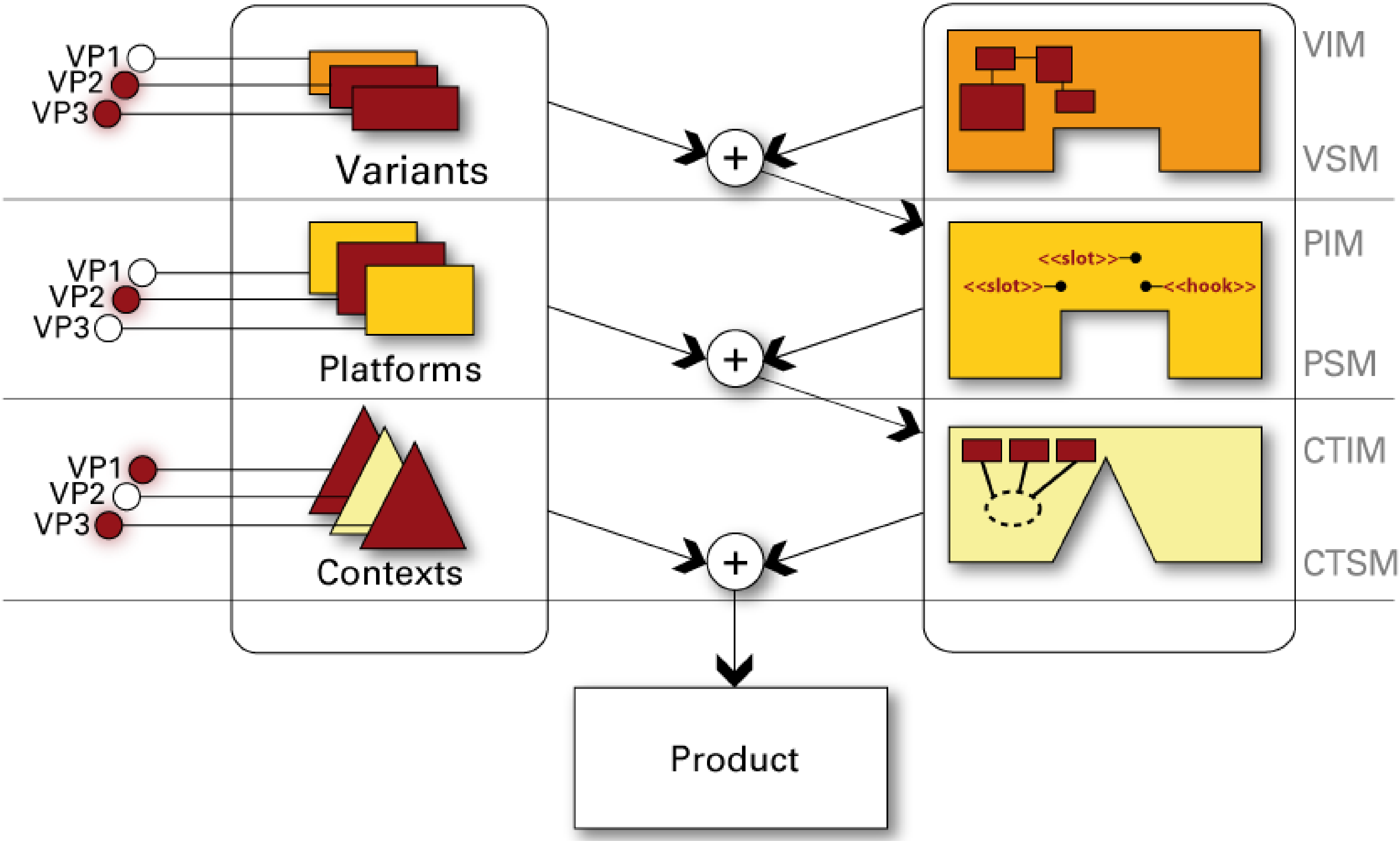
- **Goal: a staged MDSD-framework for PLE where each stage produces the software artifacts used for the next stage**



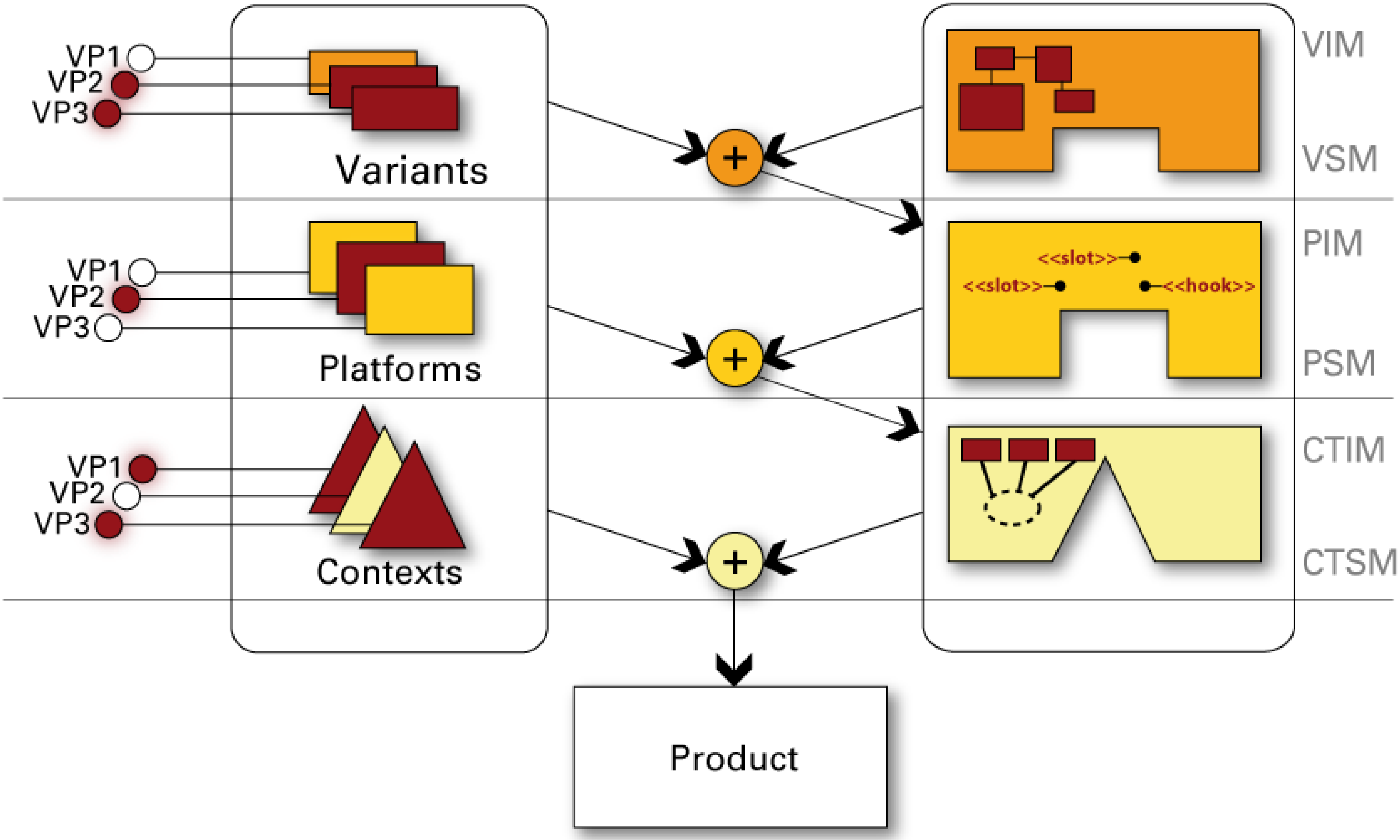
- **Characteristic feature 1:**
- **Variability on each stage**



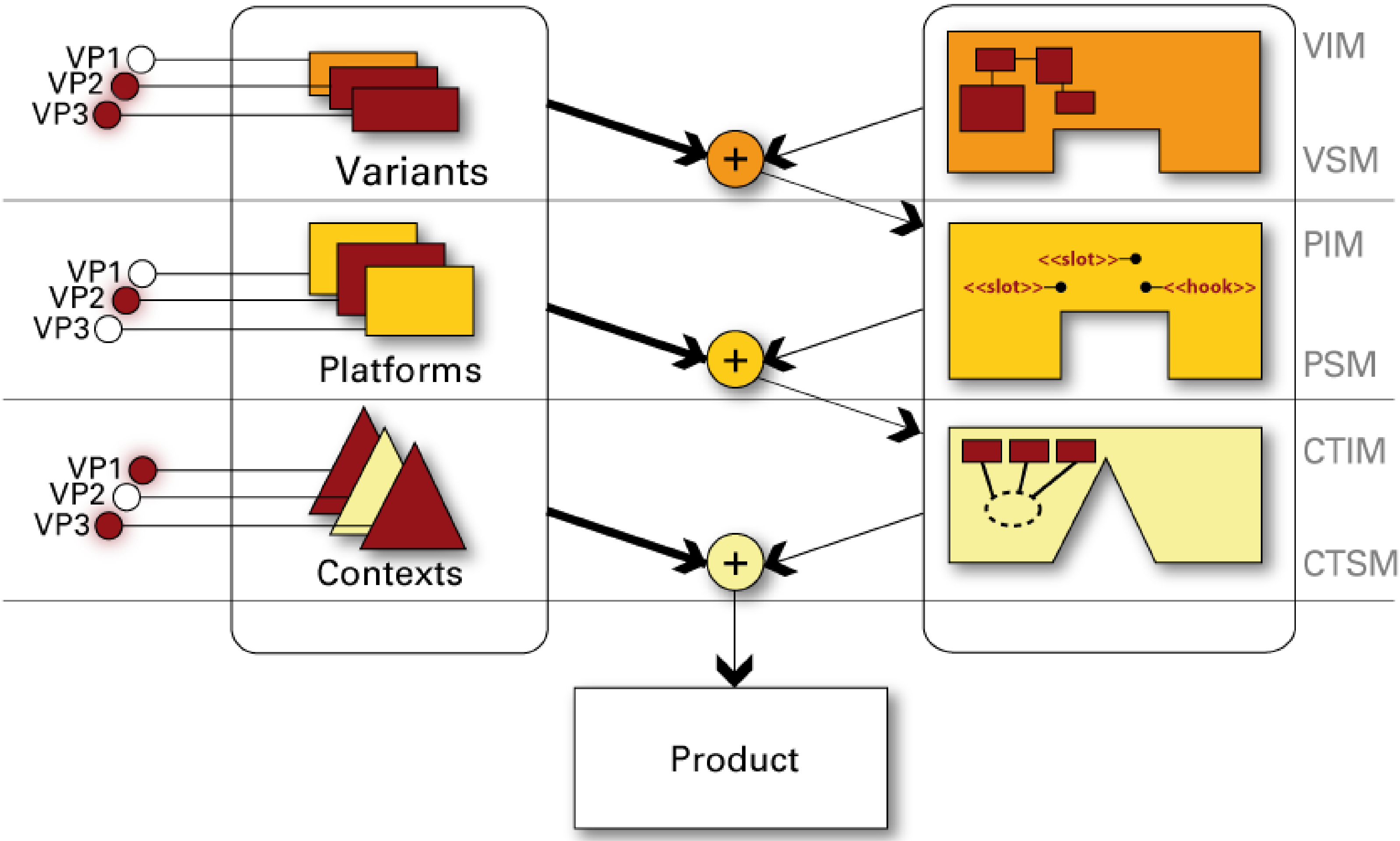
- **Characteristic feature 2:**
- **Different modeling languages, component systems and composition languages per stage**



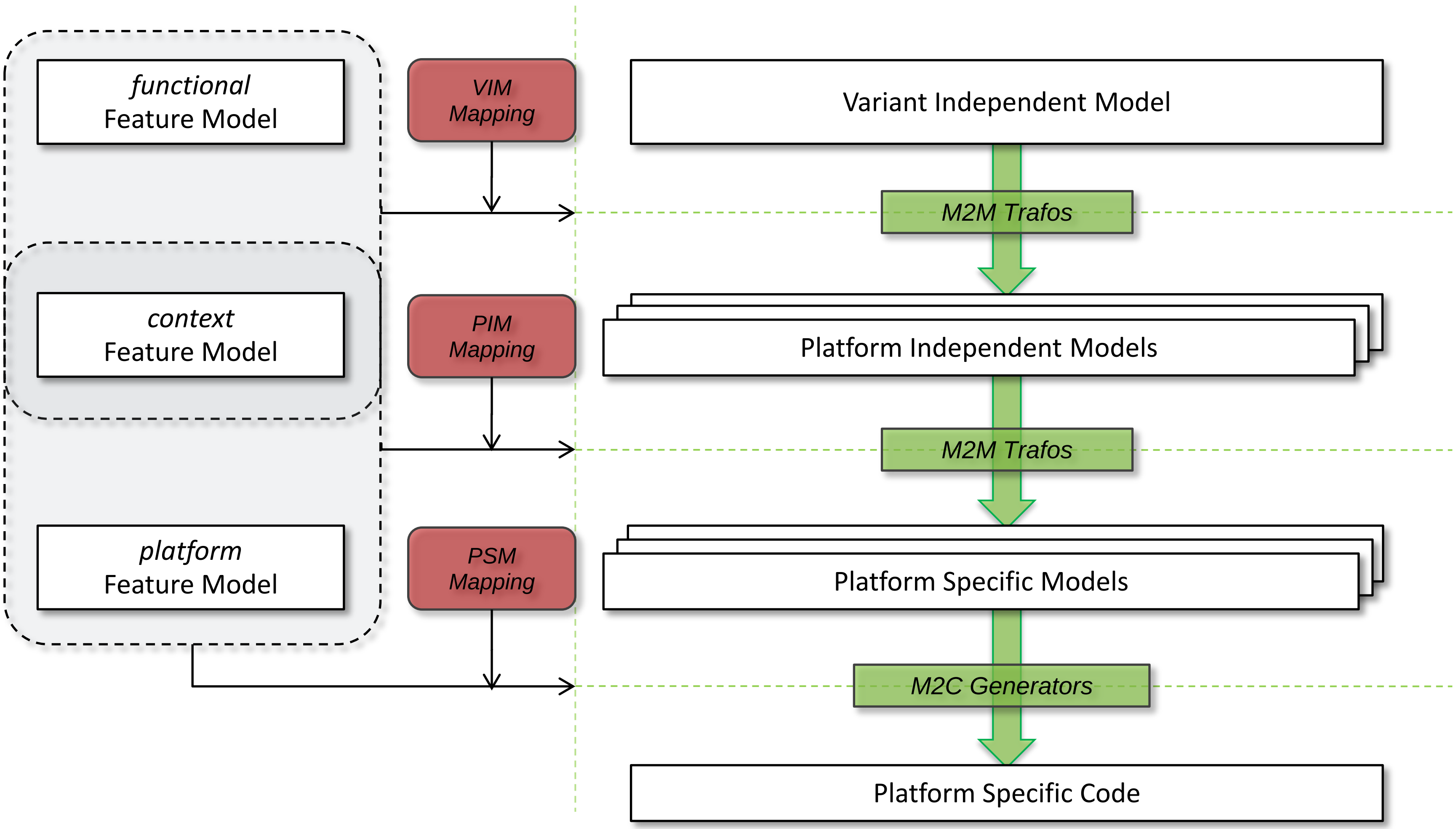
- **Characteristic feature 3:**
- **Different composition mechanisms per stage**



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

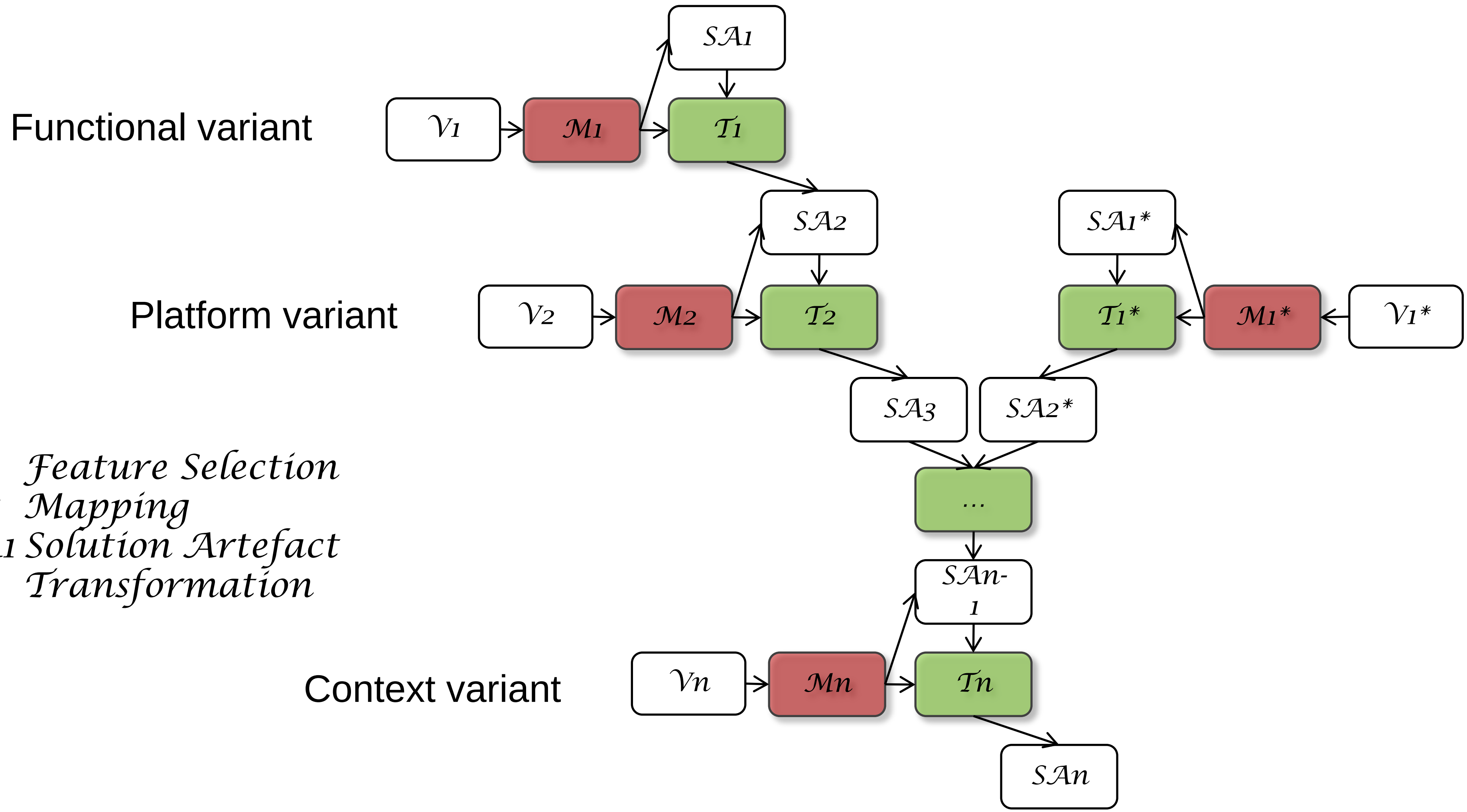


- **How do we compose transformations? Between different stages?**



➢ TraCo encapsulates transformations into composable components

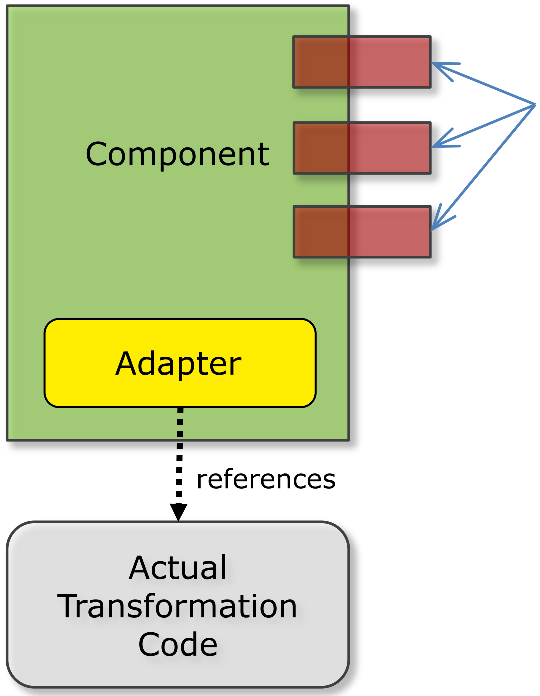
- Arranges them with *composition programs of* parallel and sequential transformation steps (multi-threaded transformation)

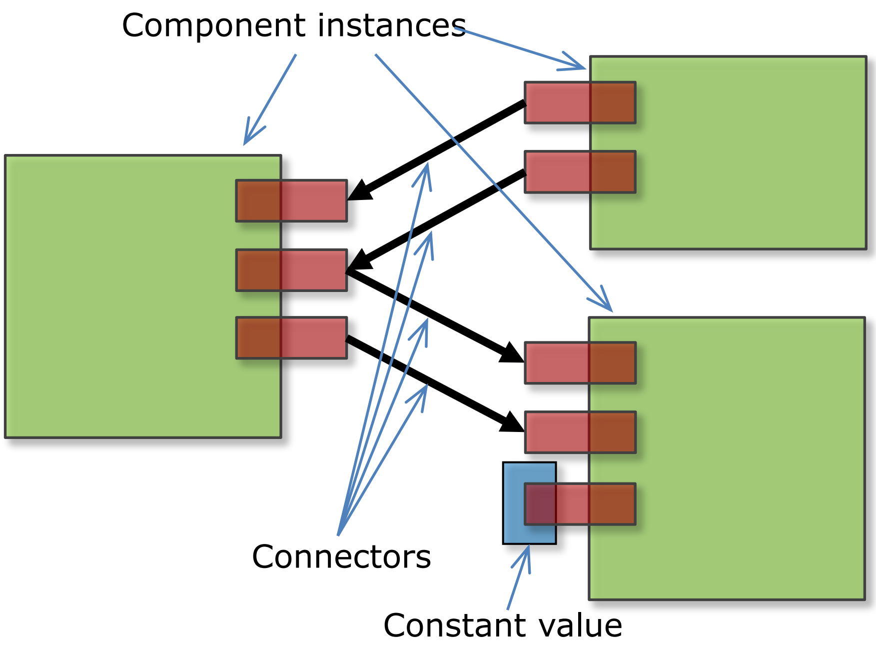


V1 Feature Selection
M1 Mapping
SA1 Solution Artefact
T1 Transformation

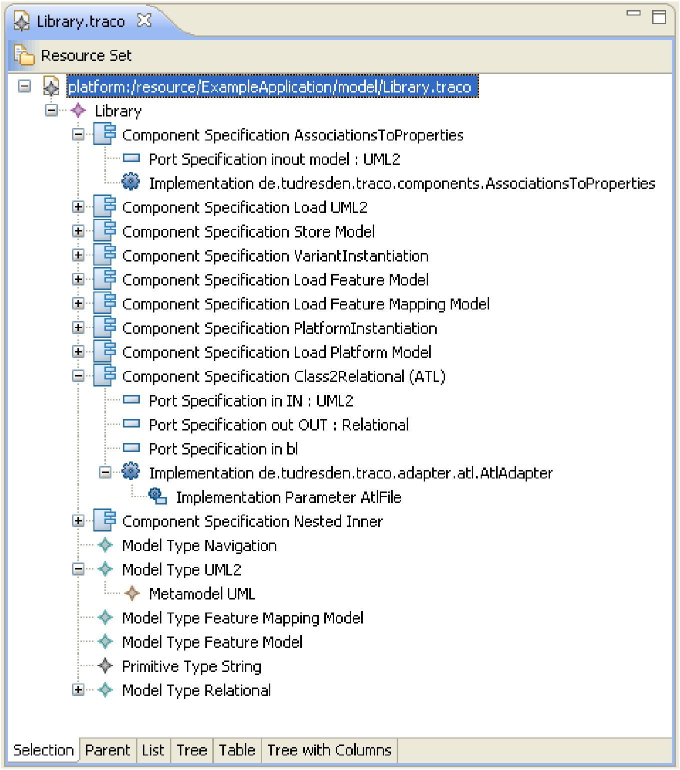
Steps in Multi-Staged Derivation of Transformations

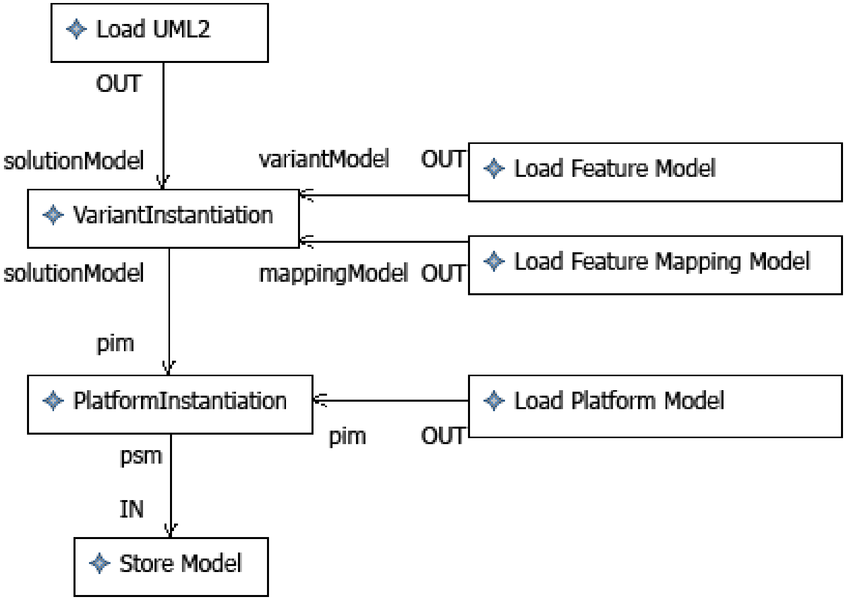
1. **Transformations are represented as composable components**
2. **Definition and Composition of Transformation Steps**
   - A *Composition System* is needed (course CBSE): Allows for reuse of arbitrary existing transformation techniques
3. **Validation of each transformation and composition step**
   - 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**

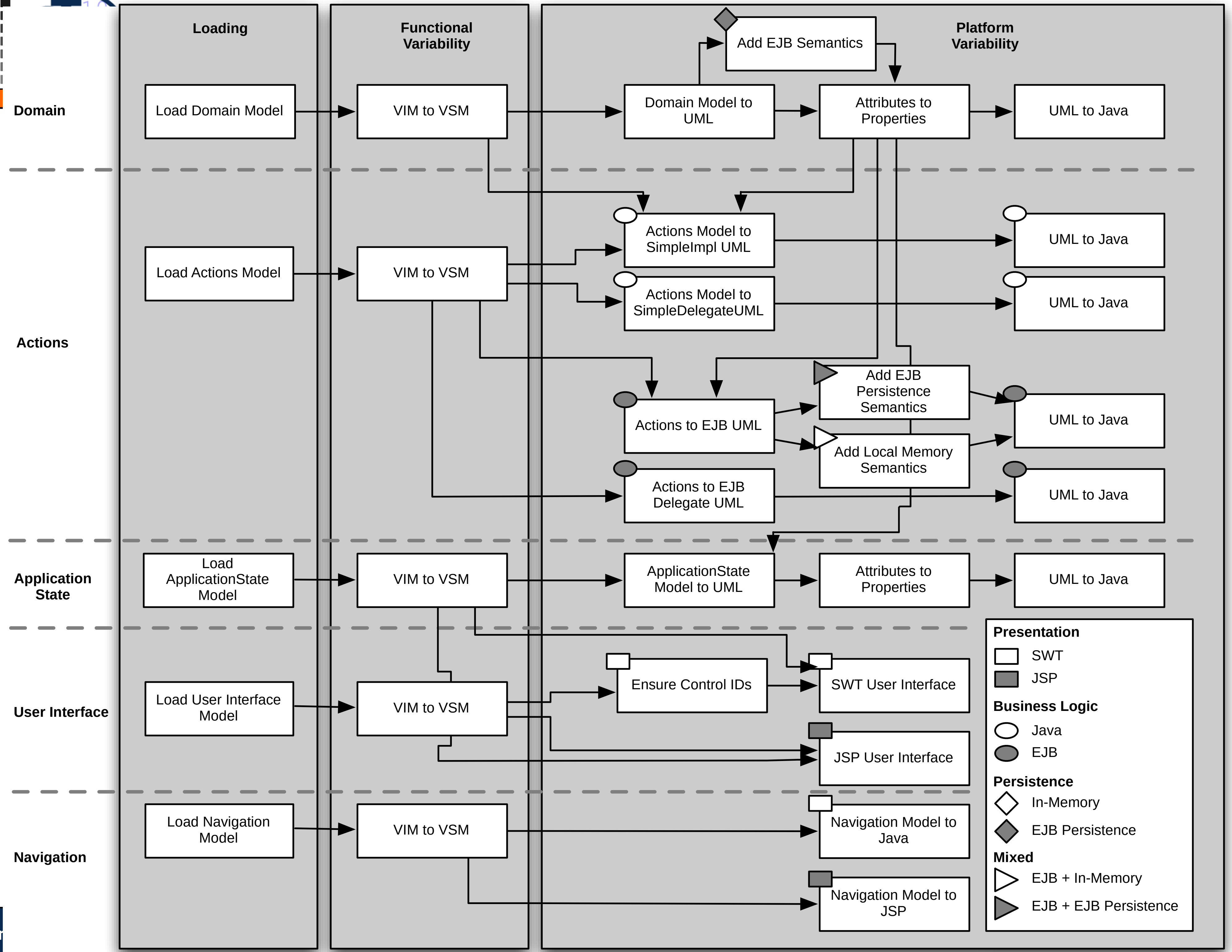




Composition Programs can be Configured (Metacomposition)

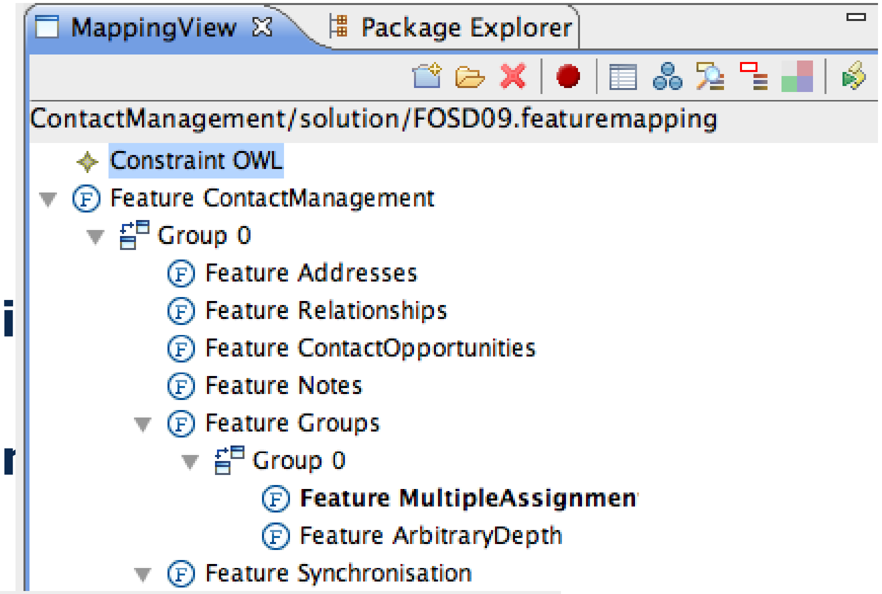
„Anything you can do, I do meta" **(Charles Simonyi)**

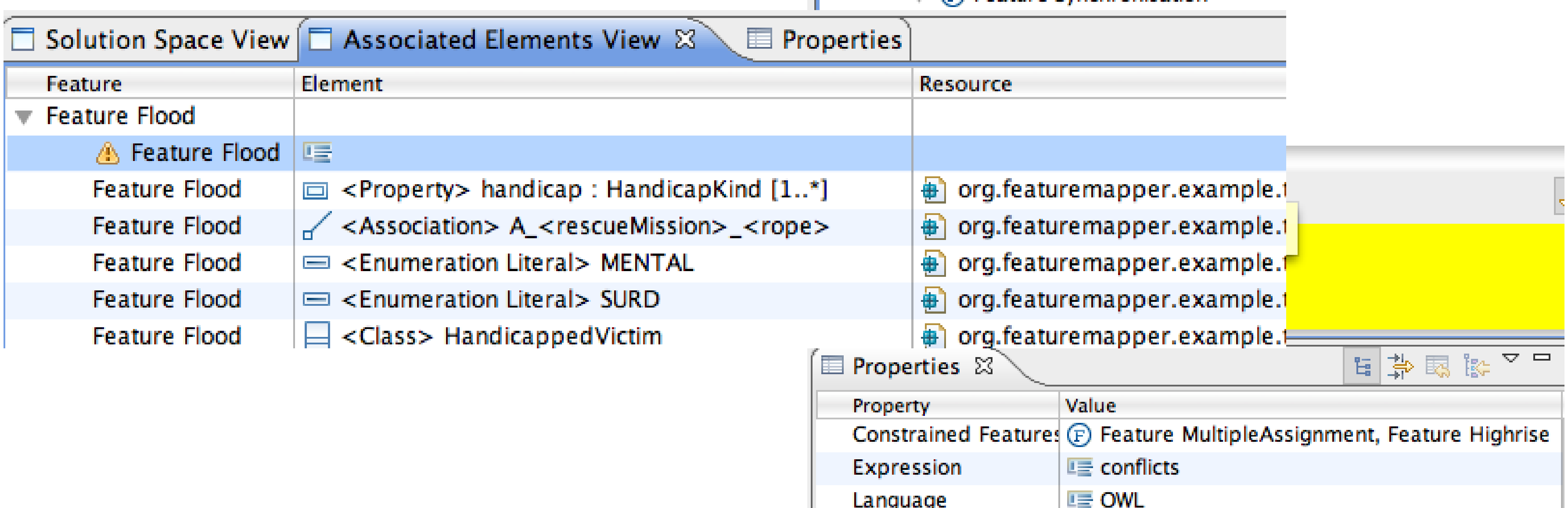
- **The composition program shown in the last slide can be subject to transformation and composition**
- **If we build a product line with TraCo, platform variability can be realised by different transformation steps**
- **A TraCo composition program can be used with FeatureMapper**
  - Multi-Staged transformation steps
  - Even of composition programs
- **More about *metacomposition* in CBSE course**



Loading
Functional Variability
Platform Variability
Domain
Actions
Application State
User Interface
Navigation
Load Domain Model
VIM to VSM
Domain Model to UML
Add EJB Semantics
Attributes to Properties
UML to Java
Actions Model to SimpleImpl UML
Actions Model to SimpleDelegateUML
Load Actions Model
Actions to EJB UML
Add EJB Persistence Semantics
Add Local Memory Semantics
Actions to EJB Delegate UML
Load ApplicationState Model
ApplicationState Model to UML
Ensure Control IDs
SWT User Interface
Load User Interface Model
JSP User Interface
Load Navigation Model
Navigation Model to Java
Navigation Model to JSP
Presentation
SWT
JSP
Business Logic
Java
EJB
Persistence
In-Memory
EJB Persistence
Mixed
EJB + In-Memory
EJB + EJB Persistence

- **Motivation: Make sure that well-formedness of all participating models is ensured**
  - Feature Model
  - Mapping Model
  - Solution Models
- **Well-formedness rules are descri**
- **Constraints are enforced during r**



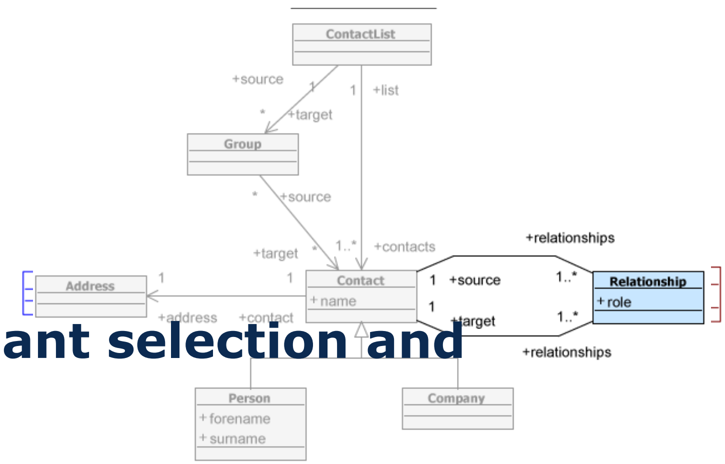


Case Studies with FeatureMapper, TraCo, and FEASIPLE

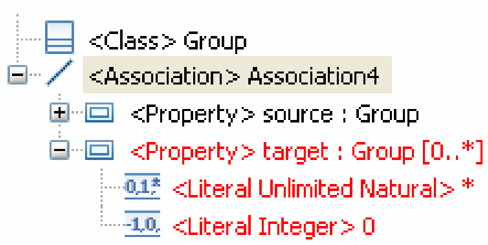
- **Simple Contact Management Application Software Product Line**
  - FeatureMapper used to map features to UML2 model elements
  - Both static and dynamic modelling
- **Simple Time Sheet Application Software Product Line**
  - FeatureMapper used to tailor ISC composition programs
  - ISC used as a universal 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**