TECHNISCHE UNIVERSITÄT DRESDEN

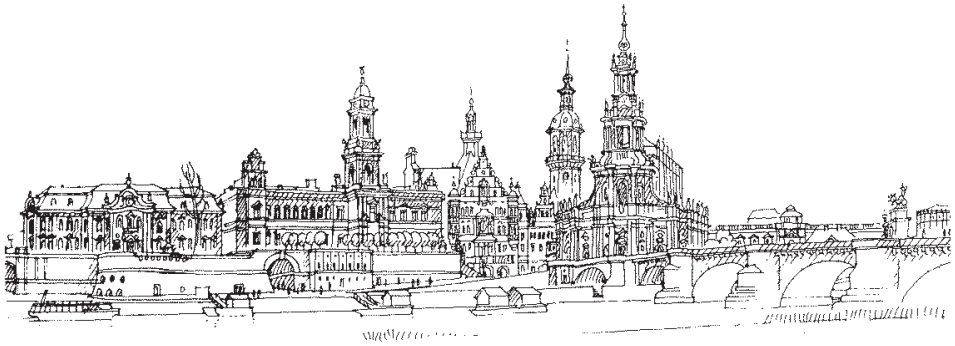
Fakultät Informatik

Technische Berichte
Technical Reports
ISSN 1430-211X

TUD-FI11-07-Dezember 2011

Julia Schroeter[1], **Malte Lochau**[2], **Tim Winkelmann**[2]
[1]Institut für Software- und Multimediatechnik, TU Dresden, [2]Institut für Programmierung und Reaktive Systeme, TU Braunschweig

Extended Version of Multi-Perspectives on Feature Models



Technische Universität Dresden
Fakultät Informatik
D–01062 Dresden
Germany
URL: http://www.inf.tu–dresden.de/

Extended Version of Multi-Perspectives on Feature Models

Julia Schroeter[1], Malte Lochau[2] and Tim Winkelmann[2]

[1] TU Dresden
Institute for Software- and Multimedia-Technology
julia.schroeter@tu-dresden.de
[2] TU Braunschweig
Institute for Programming and Reactive Systems
{m.lochau,t.winkelmann}@tu-bs.de

Abstract. Domain feature models concisely express commonality and variability among variants of a software product line. For separation of concerns, e.g., due to legal restrictions, technical considerations, and business requirements, multi-view approaches restrict the configuration choices on feature models for different stakeholders. However, recent approaches lack a formalization for precise, yet flexible specifications of views that ensure every derivable configuration perspective to obey feature model semantics. Here, we introduce a novel approach for clustering feature models to create multi-perspectives. Such customized perspectives result from composition of multiple concern-relevant views. A structured view model is used to organize feature groups, whereat a feature can be contained in multiple views. We provide formalizations for view composition and guaranteed consistency of the resulting perspectives w.r.t. feature model semantics. Thereupon, an efficient algorithm to verify consistency for entire clusterings is provided. We present an implementation and evaluate our concepts on two case studies.

1 Introduction

In software product line (SPL) engineering, the variability and commonality among product variants of the same domain are expressed in a domain feature model [13, 25, 32]. It organizes features in a hierarchical structure as well as dependencies and constraints between them. In general, the entire domain feature model is used to derive product variants. However, there are various case scenarios which require the variant space defined by the domain feature model to be further restricted. Reasons for those restrictions are driven by business or legal concerns, e.g., to enable a variable pricing strategy for offering features as packages to various stakeholders [22]. Other concerns may be of technical nature, e.g., to restrict the overall variant space to a representative subset for efficiently testing complete SPLs (cf e.g. [21]).

It seams promising to express these concerns by grouping features in a separate model orthogonally to the domain feature model. Prior the derivation of a

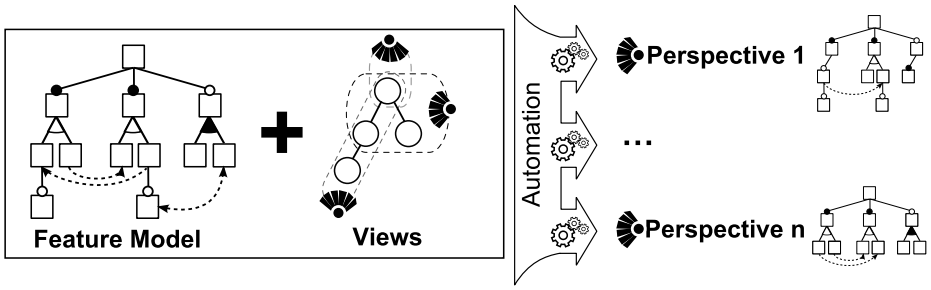


Fig. 1. Perspectives are created by joining multiple views on a domain feature model. The result is a filtered feature model, that is subsequently used to derive variants.

product by a specific stakeholder, concern-related groups are selected and the domain feature model will be filtered accordingly to ensure that restricted features are not available for selection. Those groups are perceived as overlapping views on the feature model. According to the ISO/IEC/IEEE 42010:2011, *Systems and software engineering*[3] standard, "a view is a representation of the whole system from the perspective of a related set of concerns". In other words, a view shows only features that belong to concerns a stakeholder is interested in. Multiple approaches to create views on feature models exist [1, 4, 11, 24]. Though, these approaches focus on the multi-dimensional separation of concerns (MDSoC) and a particular view is not intended to derive a complete product variant, but rather to allow for specific configuration decisions only. To the best as our knowledge, there are neither approaches to tailor the variant space using views on the domain feature model nor is there any work on aggregating and integrating multiple views to achieve that.

To tackle these challenges, we propose multi-perspectives on feature models. Therefore, in a perspective we aggregate multiple views to refine the variant space of the original domain feature model, as shown in Fig. 1. We define further consistency requirements to guarantee soundness of the potential multi-perspectives and introduce viewpoints to explicitly define allowed view combinations. Every viewpoint requires to incorporate a feature model perspective that states a specialization, i.e., a refinement of the original feature model semantics. A domain feature model and a view model are unified in a cluster model, which imposes a conservative extension to the domain feature model.

Ensuring cluster model consistency is, in general, hard to maintain due to the crosscutting nature w.r.t. the feature model and the potential overlapping of feature groups in a view model. Therefore, besides a comprehensive brute-force approach, we also provide an incremental heuristic for verifying cluster model consistency efficiently. Furthermore, our concepts support customization on feature model level in the way that stakeholder-specific features added to the domain feature model are restricted to that particular stakeholder's perspective.

[3] http://www.iso-architecture.org/ieee-1471/

The structure of this report is as follows. We explain preliminaries in Sect. 2. In Sect. 3, we formalize our concepts of multi-perspectives on feature models, we outline cluster model consistency requirements and provide an efficient algorithm for their verification. Subsequently, in Sect. 4 we show how to apply those concepts in SPL engineering. We present two case studies to evaluate the concepts of our approach and the performance of consistency insurance in Sect. 5 and document our technical realization. Finally, we present related work in Sect. 6 and conclude our work in Sect. 7.

2 Preliminaries

In this section, we review some basic notions concerning syntax and semantics of *feature models* A multitude of feature modeling variants exists in the literature [7]. Here, we refer to the approach introduced in the feature oriented domain analysis (FODA) study by Kang et al. [20].

2.1 Concrete Syntax of Feature Models

The concrete syntax of a feature model is usually given in a graphical layout, i.e., *feature diagrams* capturing hierarchies, dependencies, and constraints between domain features. Feature diagrams organize features in a tree structure. Four kinds of edges represent different hierarchical decomposition relations between a parent feature and its groups of child features. These concepts are graphically represented in the sample feature model shown on the left hand side of Fig. 2.

The intuitive semantics of the four kinds of edges are explained as follows:

1. a *mandatory* feature will be contained in every variant, in which it's parent feature is contained,
2. an *optional* feature may be included in a variant, where it's parent feature is contained,
3. if a group of child features is *alternative*, exactly one of its features is present in every variant, in which the parent feature is included, and
4. if a group of child features is *or related*, at least one of its features is present in every variant, in which the parent feature is contained.

For a feature diagram language of a domain feature model to be conceptional complete, i.e., *fully expressive*, further *constraints* are to be provided [28], e.g.,*requires* and *exclude* cross-tree edges. Those constraints are often represented as additional propositional formulas over features that cross-cut the feature diagram tree [8] in arbitrary ways. Further constructs for enhancing feature models are mentioned in the literature, e.g., feature cardinalities, feature attributed, and feature references which are out of scope of this report.

2.2 Abstract Syntax of Feature Models

Please note, that we consider the two notions *feature model* and *feature diagram* as synonyms in the following. Therefore, we introduce an abstract syntax for cardinality-based feature models.

Definition 1. *(Feature Model)*
A feature model is a 4-tuple $FM = (F, \prec, \lambda, , \Phi)$*, where* F *is a finite set of feature nodes,* $\prec \subseteq F \times F$ *is a decomposition relation on* F*,* $\lambda : \mathcal{P}(F) \rightharpoonup \mathbb{N}_0 \times \mathbb{N}_0$ *is a partial cardinality function assigning intervals to feature groups, and* Φ *is a set of propositional formulas over* F*.*

For feature model *FM* being *well-formed*, it must satisfy the following rules:

1. Relation $\prec$ forms a *rooted tree* on F, i.e., the reflexive, transitive closure $\preceq^*$ defines a *partial order* on F, and for every node $f \in F$ despite the unique root node $f_r \in F$, there is exactly one predecessor node $f' \in F$ with $f' \prec f$.
2. In feature groups $F' \in dom(\lambda)$, except the singleton group F_r solely containing the root feature, all features have the same parent node, thus:

$$\forall F' \in dom(\lambda) \setminus \{F_r\} : \exists f'' \in F : \forall f' \in F' : f'' \prec f'$$

3. Function λ partitions F, i.e.:

$$\text{(a) } \forall F', F'' \in dom(\lambda) : F' \neq F'' \Rightarrow F' \cap F'' = \emptyset \text{ , and (b) } \bigcup_{F' \in dom(\lambda)} F' = F.$$

4. Feature groups are non-empty: $\emptyset \notin dom(\lambda)$
5. Cardinalities $\lambda(F') = (k, l)$ of feature groups $F' \subseteq F$ define reasonable intervals for child features, i.e, $k \leq l$ and $l \leq \mid F' \mid$ holds.

Cardinalities emulate the four decomposition types for feature groups $F' \in dom(\lambda)$, where $n = \mid F' \mid$ as follows:

1. $\lambda(F') = (n, n)$ for *mandatory* features/groups,
2. $\lambda(F') = (0, n)$ for *optional* features/groups,
3. $\lambda(F') = (1, 1)$ for *alternative* groups, and
4. $\lambda(F') = (1, n)$ for *or* groups.

For the group F_r of the root feature, we assume $\lambda(F_r) = (1, 1)$ by conventions. Propositional formulas $\phi \in \Phi$ are boolean formulas $\phi \in \mathbb{B}(F)$ expressing cross tree constraints on *FM*. In particular, we consider:

1. *requires* edges leading from feature f to feature f' introducing implications $\phi_{rq} = f \rightarrow f'$, and
2. *excludes* edges introducing implications $\phi_{ex} = f \rightarrow \overline{f'}$.

The set Φ is interpreted as the conjunction $\bigwedge_{\phi \in \Phi} \phi$ of all constraints.

By $\mathcal{FM}(F)$ we refer to the set of all t'*syntactically well-formed* feature models over feature names F. Please note, that for the sake of simplicity of later discussions we also assume well-formed feature model FM' built only over a sub set $F' \subseteq F$ of features to be also part of the syntactical domain $\mathcal{FM}(F)$.

2.3 Semantics of Feature Models

The semantics of a feature model *FM* defines the *variant space*, i.e., the set of *valid* product configurations. A *product configuration* is given as a subset $F_{pc} \subseteq F$ of features selected for a concrete product variant. Hence, the semantical function

$$[\![\cdot]\!] : \mathcal{FM}(F) \rightarrow \mathcal{P}(\mathcal{P}(F))$$

maps feature models $FM \in \mathcal{FM}(F)$ built over features F into the domain of sets of *valid* product configurations obeying the decomposition types and constraints, i.e.,

$$[\![FM]\!] \in \mathcal{P}(\mathcal{P}(F)), \text{ where } FM \in \mathcal{FM}(F)$$

The semantical evaluation function defines the *maximum set* of valid product configurations such that:

$$\begin{aligned}[\![FM]\!] = \{&F_{pc} \in \mathcal{P}(F) \mid f_r \in F_{pc} \wedge \\ &(f \in F_{pc} \wedge f \prec F' \wedge \lambda(F') = (k,l) \Rightarrow k \leq | \{f' \in F' \cap F_{pc}\} | \leq l) \wedge \\ &(f'' \in F_{pc} \wedge f''' \prec f'' \Rightarrow f''' \in F_{pc}) \wedge F_{pc} \models \bigwedge_{\phi \in \Phi} \phi\}\end{aligned}$$

where $f \prec F' \Leftrightarrow : \forall f' \in F' : f \prec f'$.

Thus, validity of configurations $F_{pc} \in \mathcal{P}(F)$ requires:

1. the root node f_r to be contained in every configuration,
2. satisfaction of group cardinalities concerning features f' in all groups F' decomposing selected nodes f,
3. justification of a selected feature f'' by means of the presences of its parent feature f''', and
4. satisfaction of global constraints in Φ on F_{pc}.

Although, a given feature model is syntactically well-formed, it potentially lacks a useful configuration semantics. We use the semantics introduced above to review the notion of feature model satisfiability (cf. [17]).

Definition 2. *(Satisfiable Feature Model)*
A feature model FM is satisfiable, if $[\![FM]\!] \neq \emptyset$.

Thus, satisfiability requires the existence of at least one non-empty feature selection $F_{pc} \subseteq F$, that satisfies the conditions over features as imposed in the feature model. In other words, a feature model is satisfiable, if the configuration space is non-empty. We assume any given feature model under consideration to be satisfiable in the following.

A further semantical property crucial for the approach presented in this report, is the notion of *refinement*.

Definition 3. *(Feature Model Refinement)*
A feature model FM' *is a refinement of a feature model FM, if* $[\![FM']\!] \subseteq [\![FM]\!]$.

Thus, a refined feature model semantics defines a sub space of the configuration space of the original feature model, i.e., refinement restricts the variability [31].

For explicitly reasoning about (staged) configuration processes on multi-perspectives on feature models in future work, we also provide an alternative semantical representation. In the next section, we provide a formalized staged configuration semantics based on our feature model definition.

2.4 Staged Configuration Semantics for Feature Models

Feature Models are commonly used:

1. to model valid feature combinations for product families during domain engineering, and
2. to support the process of configuring (and assembling) valid product variants during application engineering [9, 18].

Considering 2), staged configuration semantics describes the incremental process of deriving product configurations in an operational way [14, 12, 27].

Configuration processes are carried out in stages/steps, where stages may be associated with different phases of the configuration process and/or with views dedicated to different stakeholders. In each configuration step C_{op}, a configuration operation *op* is performed on the feature model. Therefore, a *configuration step* transforms a feature model $FM \in \mathcal{FM}(F)$ to a modified feature model $FM' \in \mathcal{FM}(F)$.

Here, we consider *positive*, as well as *negative* configuration steps. This means, that configuration *choices* made by a stakeholder w.r.t. feature parameters, is either a selection, or a deselection of a feature for a product configuration. Thus, when conducting a sequence of configuration steps, the variability is incrementally removed from the feature model until reaching a fully specified product configuration.

Formally, this kind of configuration operation has the following signature:

$$C_{op} : \mathcal{FM}(F) \times F \rightarrow \mathcal{FM}(F)$$

where we write $op \in \{+, -\}$ to either denote a selection operation $C_+(FM, f)$, or a deselection operation $C_-(FM, f)$ of feature f on feature model FM. The semantics of a configuration step:

$$C_{op}(FM, f) = FM' = (F, \prec, \lambda'_{op}, \Phi)$$

considering feature $f \in F$, where $f \in F' \in dom(\lambda)$ and for $\lambda(F') = (k, l)$, transforms feature model $FM = (F, \prec, \lambda, \Phi)$ to feature model FM'. The adaption of λ to λ'_{op} depends on *op* and is defined as follows:

- if $\mid F' \mid > 1$, then:
  1. feature f is moved from group F' into a fresh singleton group:

$$dom(\lambda'_{op}) := (dom(\lambda) \setminus F') \cup F'' \cup \{f\}, \text{ where } F'' = F' \setminus \{f\}$$

2. group cardinalities are adjusted:
   (a) the interval of the former group of f is decremented:

$$\lambda'_{op}(F'') := \begin{cases} (k-1, l-1), & \text{if } op = \text{`}+\text{`} \ \wedge \ k > 0 \\ (k, l-1), & \text{else} \end{cases}$$

   (b) the singleton group of f is set to either mandatory, or exclusion:

$$\lambda'_{op}(\{f\}) := \begin{cases} (1,1), & \text{if } op = \text{`}+\text{`} \\ (0,0), & \text{if } op = \text{`}-\text{`} \end{cases}$$

– else, if $\mid F' = \{f\} \mid = 1$ and $\lambda(F') = (k, l)$, then:

$$\lambda'_{op}(F') := \begin{cases} (1,1), & \text{if } op = \text{`}+\text{`} \wedge l = 1 \\ (0,0), & \text{if } op = \text{`}-\text{`} \wedge k = 0 \end{cases}$$

  i.e., either single optional features are (de-)selected, or feature (de-)selections already performed are preserved.
– feature constraints in Φ relevant for application of $C_{op}(FM, f) = FM' = (F, \prec, \lambda'_{op}, \Phi)$ are recursively applied:
  1. features f' required for selected features f are also selected:

$$\forall f \to f' \in \Phi : \lambda'_+ := \lambda''_+, \text{ where } (F, \prec, \lambda''_+, \Phi) = C_+((F, \prec, \lambda'_+, \Phi), f')$$

  2. features f' requiring a deselected feature f are also deselected:

$$\forall f' \to f \in \Phi : \lambda'_- := \lambda''_-, \text{ where } (F, \prec, \lambda''_-, \Phi) = C_-((F, \prec, \lambda'_-, \Phi), f')$$

  3. features f' excluded by selected features f are deselected:

$$\forall f \to \overline{f'}, f' \to \overline{f} \in \Phi : \lambda'_+ \quad := \quad \lambda''_-, \\ \text{where } \ (F, \prec, \lambda''_-, \Phi) = C_-((F, \prec, \lambda'_+, \Phi), f')$$

Hence, in a configuration step, two transformation phases take place:

1. the selected/deselected feature is moved from its feature group into a sibling singleton group, whose cardinality imposes the configuration choice, and
2. the recursive selection/deselection of further features is triggered according to constraints potentially affected by the configuration choice and/or previous recursion steps.

Please note, that the termination of recursive constraint evaluations that imply cyclic dependencies on a feature model is ensured via case 2). Further note, that case 2) also ensures that no (syntactically ill-formed) empty groups arise during a configuration step, as singleton groups are preserved.

In general, the semantics allows features $f \in F$ to be (de-)selectable in $FM \in \mathcal{FM}(F)$ in any stage of the configuration process even though contradicting

feature model semantics. For instance, attempting to deselect a mandatory feature should, by intuition yield an erroneous configuration result $FM' \notin \mathcal{FM}(F)$. The same holds for selecting a feature that was already deselected in a previous configuration step.

Correspondingly, a configuration step $C_{op}(FM, f)$ for f on feature model FM is *invalid*, written $C_{op}(FM, f) = \perp$, if it yields an ill-formed feature model FM'.

Semantically, a configuration process defines a sequence of configuration choices, where each step imposes a a feature model *refinement*:

$$FM' = C_{op}(FM, f) \;\Rightarrow\; [\![FM']\!] \subseteq [\![FM]\!]$$

The resulting specialized product configuration space of FM' is either *partially specified*, if $\mid [\![FM']\!] \mid > 1$, or it is *fully specified* if $\mid [\![FM']\!] \mid = 1$. For a configuration process that ends up in fully specified product configuration, the concrete product configuration results from the sequence of (explicit and implicit) feature selection operations.

3 Formalization of Model-Based Multi-Perspectives

In this section, we identify requirements an approach for multi-perspectives on feature models must address. Subsequently we introduce our concepts to address these requirements and give a formalization of our approach.

3.1 Requirements of Multi-Perspectives on Feature Models

From the case scenarios described in Sect. 1, we obtain the following requirements an approach for multi-perspectives on feature models has to satisfy.

Requirement 1 *Aggregation of multiple views for creating perspectives on the domain feature model.*

Requirement 2 *An explicit concept constitutes aggregations of views, that are allowed to form perspectives.*

Requirement 3 *Preservation of feature model semantics in perspectives is efficiently decidable.*

Requirement 4 *Group features according to concerns that are crosscutting to the domain feature model hierarchy in a separate view model. Thereto, use a hierarchical structure to enable step-wise refinement of groups and an overlapping structure to express crosscutting concerns.*

Requirement 5 *The view model contains a common group, that references all features which are not explicitly mapped to other groups of the view model. Those core features are available per default in every perspective.*

Requirement 6 *Express customization on feature model level.*

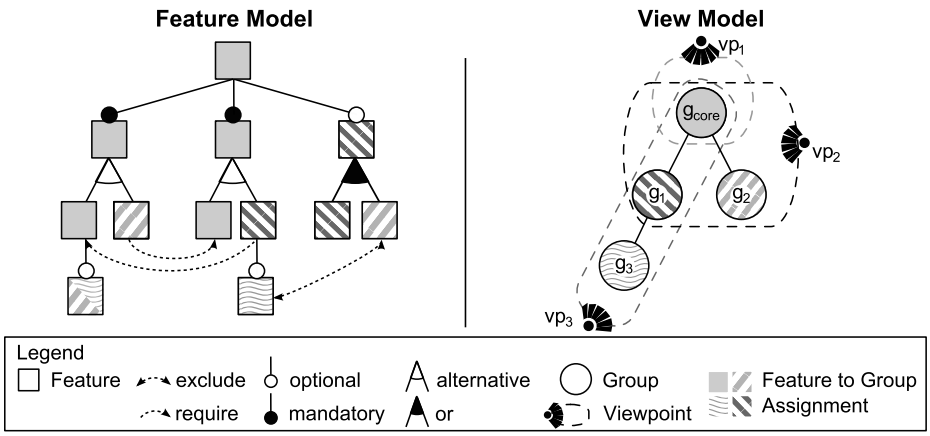


Fig. 2. The cluster model consists of a feature model, a view model and an assignment of features to groups.

To address the identified requirements, we introduce the concept of clustering feature models. It means, that features of the domain feature model are grouped according to multiple concerns, which crosscuts the feature model hierarchy. Therefore, a *cluster model* consists of the *domain feature model*, a *view model* and a mapping between both, as we show in Fig. 2. Features are assigned to *group*s of the view model, whereat a feature may be assigned to multiple groups. In turn, each group forms a partial *view* on the domain feature model, whereat views may overlap. The view model addresses Req. 4.

Nuseibeh et al. use the concept of viewpoints to describe a concrete perspective of a system [15, 23]. Hence, in our approach, a *viewpoint* is an explicit definition of groups of the view model that are aggregated to form a valid *perspective*. That addresses Req. 1 and Req. 2.

A perspective represents a filtered domain feature model or in other words, is a specialization of a feature model [31]. Variants that can be derived from a perspective are a valid sub set of the variants derivable from the domain feature model. A viewpoint defines which features are available in the according perspective and the view model structure implies, that all subgroups of a group include the viewpoint of that group as well.

In addition to purely hierarchical inclusions, modeling arbitrary overlappings among groups allows for capturing further interrelations between concerns of corresponding views addresses Req. 4. A unique root group contains every viewpoint and is therefore called *core group*. This addresses Reg. 5 as the core group references all features that are not explicitly assigned to other groups. In addition, Req. 3 is addressed by an efficient algorithm for the cluster model consistency verification that is explained in Sect. 3.5. Req. 6 is addressed by introducing a special kind of groups which is discussed in Sect. 3.3.

3.2 Views and Perspectives on Feature Models

Views use a sub set of configuration parameters to restrict the access to a given domain feature model. Formally, a view projects from a feature model $FM \in \mathcal{FM}(F)$ a sub set of features $F' \subseteq F$ and related constraints.

Definition 4. *(Feature Model View)*
A feature model view $V_{FM} = (F_V, \Phi_V)$ of a feature model $FM \in \mathcal{FM}(F)$ consists of a subset $F_V \subseteq F$ of selectable features, and a subset $\Phi_V \subseteq \Phi$ of constraints such that $\phi_V \in \mathbb{B}(F_V)$ for each $\phi_V \in \Phi_V$.

By $\mathcal{V}_{FM}$, we refer to the set of all views of a feature model *FM*. In general, a view $V_{FM} \in \mathcal{V}_{FM}$ contains an arbitrary selection of features $F_V \subseteq F$ and corresponding constraints $\phi_V \in \Phi_V$.

Example 1. In Fig. 2, four views on the feature models are highlighted via different hatchings marking feature groups selected into the same view. Note, that the lower most feature on the left is selected into two views.

Views restrict domain feature models, thus each view is associated with a *perspective* that interprets a view as a variability-reduced feature model, i.e., a partial tree of the original feature tree. For the sake of simplicity, we assume

$$\mathcal{FM}(F_V) \subseteq \mathcal{FM}(F), \text{ where } F_V \subseteq F$$

for the following discussions, i.e., $\mathcal{FM}(F)$ to also contain feature models built over only a sub set of features F.

Perspectives. A perspective $FM_V \in \mathcal{FM}(F)$ for a view $V_{FM} \in \mathcal{V}_{FM}$ is defined via a projection function:

$$p_{FM} : \mathcal{V}_{FM} \rightharpoonup \mathcal{FM}(F)$$

where

$$p_{FM}(V_{FM}) = (F_V, \prec_V, \lambda_V, \Phi_V)$$

for a view V such that:

1. $\prec_V \subseteq F_V \times F_V \subseteq \prec$, i.e., the *restriction* of $\prec$ onto F_V, and
2. λ_V is reduced to F_V as follows:
   - if $F' \in dom(\lambda)$, then $F' \cap F_V \in dom(\lambda_V)$, if $F' \cap F_V \neq \emptyset$
   - if $\lambda(F') = (k, l)$, then $\lambda_V(F' \cap F_V) = (k, l- \mid F' \setminus \{F_V \cap F'\} \mid)$

Note, that p_{FM} is a partial function, because views $V_{FM} \in \mathcal{V}_{FM}$ exist, whose projection application would yield an *ill-formed* feature model $p_{FM}(V_{FM}) \notin \mathcal{FM}(F)$. Furthermore, even if $p_{FM}(V_{FM}) \in \mathcal{FM}(F)$ holds, the perspective $p_{FM}(V_{FM})$ is not necessarily semantically refining *FM*, i.e., $[\![p_{FM}(V_{FM})]\!] \not\subseteq [\![FM]\!]$. Therefore, we introduce the notion of *FM-consistent views*.

Definition 5. *(FM-consistent View)*
A view $V_{FM} \in \mathcal{FM}(F)$ is FM-consistent if:

1. $p_{FM}(V_{FM}) \in \mathcal{FM}(F)$,
2. $[\![p_{FM}(V_{FM})]\!] \subseteq [\![FM]\!]$, *and*
3. $p_{FM}(V_{FM})$ *is satisfiable.*

The first property holds, if the selected features in a view preserve the feature tree structure of *FM* and obey feature group constraints. For the second property, constraints are to be considered. In general, for each constraint $\phi \in \Phi \setminus \Phi_v$ and $F' \subseteq F$ to be the sub set of features appearing in ϕ, we require $F' \cap F_v = \emptyset$. We weaken this property as we focus only on feature models with binary require and exclude constraints. Thus, the feature selection must solely support feature implications to be satisfiable, as exclude constraints are either fully supported, or they cannot be invalidated in a view, because one of the features is not selectable.

Lemma 1. *A feature model view $V_{FM} \in \mathcal{V}_{FM}$ is FM-consistent, if:*

1. $f_r \in F_V$,
2. *if* $f \in F_V$ *and* $f' \prec f$, *then* $f \in F_V$,
3. *if* $f \in F_V$ *and* $f \prec F'$ *with* $\lambda(F') = (k, l)$, *then* $\mid F' \cap F_V \mid \geq k$
4. *if* $f \in F_V$ *and* $f \rightarrow f' \in \Phi$, *then* $f' \in F_V$, *thus* $f \rightarrow f' \in \Phi_V$

Properties 1.) and 2.) enforce preservation of the feature model (partial) tree structure, property 3.) ensures group cardinalities to be satisfiable also by the remaining features in the view, and property 4.) ensures the view to be closed w.r.t. the transitive closure of feature implication constraints.

Example 2. In Fig. 2, only the view marked in solid gray is *FM-consistent*.

By $\mathcal{V}^c_{FM} \subseteq \mathcal{V}_{FM}$ we refer to the sub set of *FM-consistent* views on a feature model *FM*. Views $V_{FM} \notin \mathcal{V}^c_{FM}$ are called *partial* views.

View Composition. For the aggregation of multiple views, we introduce a *composition operator* on views:

$$\oplus : \mathcal{V}_{FM} \times \mathcal{V}_{FM} \rightarrow \mathcal{V}_{FM}$$

such that:

$$V_{FM} \oplus V'_{FM} = V''_{FM} = (F_V \cup F_{V'}, \Phi_{V''})$$

where:

- $\Phi_{V''} \subseteq \Phi$, and
- $\phi \in \Phi_{V''} \Leftrightarrow \phi \in \mathbb{B}(F_{V''})$.

Due to the definition of view composition via set union and conjunction, we obtain the following properties.

Lemma 2. *The view composition operator is commutative and associative.*

The proof follows directly from the operator definition. Due to the cross cutting nature of constraints in feature models, feature model semantics is, in general, not compositional [2]. Accordingly, view composition does, in general, not commute with feature model semantics.

Proposition 1. *(Non-commutativity of View Composition)*
For two feature model views $V_{FM}, V'_{FM} \in \mathcal{V}_{FM}$

$$[\![p_{FM}(V_{FM} \oplus V_{FM'})]\!] \neq [\![p_{FM}(V_{FM})]\!] \cup [\![p_{FM}(V'_{FM})]\!]$$

holds.

The proof follows directly from the usual counter-examples addressing non-compositional feature model semantics due to cross-cuttings of constraints. In particular, from $\Phi_{V''} \neq \Phi_V \wedge \Phi_{V'}$, it follows that constraints $\phi \in \Phi_{F_{V''}} \cap \Phi_F$ with $\phi \notin \Phi_{F_V} \cup \Phi_{F_{V'}}$ may exist.

Fortunately, view composition is closed under *FM-consistent* views, i.e., joining two *FM-consistent* views yields, again, an *FM-consistent* view.

Proposition 2. *(Closedness of FM-consistent view composition)*
For two FM-consistent views $V_{FM}, V'_{FM} \in \mathcal{V}^c_{FM}$, $V_{FM} \oplus V'_{FM} \in \mathcal{V}^c_{FM}$ *holds.*

Proof: Well-formedness of the tree structure and group constraints is preserved, as both views are *FM-consistent* for the same feature model *FM* and composition is monotone on F. For imply constraints $\phi = f \rightarrow f'$ with $f, f' \in F_{V''}$, $\phi \in \Phi_{V''}$ is guaranteed as ϕ must be already contained in V and/or V'. Otherwise, one view must have contained f without f' which would contradict the *FM-consistency* assumption. Summarizing, the relation between the different syntactical and semantical domain discussed in this section, are illustrated in Fig. 3. The subset

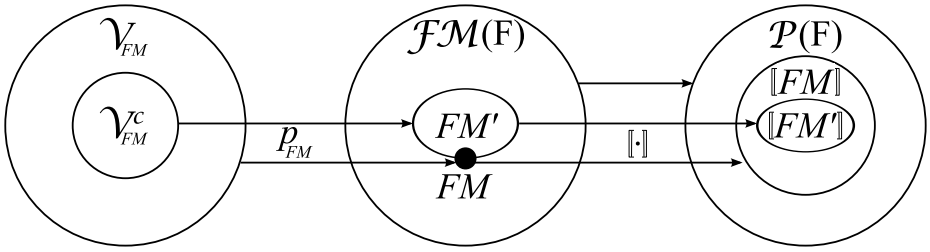


Fig. 3. A visualization of the formalized concepts on feature models, views, and variants.

$\mathcal{V}^c_{FM} \subseteq \mathcal{V}_{FM}$ of *FM-consistent* views of a feature model $FM \in \mathcal{FM}(F)$ are mapped to perspectives FM' that semantically refine FM by restricting the set of variants to a subset.

Based on these results, we use view composition to join multiple views for the derivation of viewpoint-specific perspectives on feature models. The aggregation of those perspectives for particular stakeholders depend on the organization of their viewpoints in *View Models*.

3.3 View Models

Recent multi-view approaches on feature models usually assume a one-to-one correspondence between a set of views on a feature model and a set of *disjoint* stakeholders [12, 33, 18, 19, 4, 26, 27]. Hence, views are considered to encapsulate a particular concern and multi-view approaches on feature models aim at a *separation of concerns*.

Here, we also assume views to modularize feature models for certain *concerns*. But, as concerns are potentially interrelated, we suppose views to overlap in arbitrary ways and to be organized in groups. In addition, depending on the concerns relevant for a stakeholder, various (abstract) views are aggregated into concrete well-defined *viewpoints* for deriving a tailored feature model *perspective* [23]. For capturing the relationships between views and viewpoints, we introduce a *view model*.

Definition 6. *(View Model)*
A view model is a pair $VM = (VP, G)$*, where* $VP = \{vp_1, vp_2, \ldots, vp_m\}$ *is a finite set of* m *viewpoints and* $G = \{g_1, g_2, \ldots, g_n\}$ *is a finite set of* n *groups, i.e., a collection of predicates* $g_i \subseteq VP$ *over viewpoints.*

Group predicates $g_i \in G$ indicate the corresponding sub set $g_i \subseteq VP$ of viewpoints as *members* of that group, thus sharing common concerns dedicated to that group. Furthermore, sub sets introduce an implicit *group hierarchy relation* $<_G \subseteq G \times G$, as follows:

$$g <_G g' :\Leftrightarrow g \subset g'$$

thus defining a predecessor relation among groups via inclusion of their viewpoints. Relation $<_G$ is a *strict* partial order, because we allow groups with equal predicates $g_i = g_j$ to be distinguished in G by their indices i and j. Correspondingly, two groups g_i and g_j are either related under $<_G$, or they are incomparable, i.e., either (1) disjoint, or (2) overlapping (including set equality). We define the *overlapping group relation* $\sqcap_G \subseteq G \times G$ to be:

$$g_i \sqcap_G g_j :\Leftrightarrow i \neq j \wedge g_i \cap g_j \neq \emptyset \wedge g_i \not<_G g_j \wedge g_j \not<_G g_i$$

therefore being irreflexive, symmetric, and not transitive. For *well-formed* view models, we require $<_G$ to be upwards closed in G, i.e., there exists a unique *core group* $g_{core} \in G$ with $g_{core} = VP$, thus $g <_G g_{core}$ for each $g \in G$. Furthermore, for (optional) singleton groups $g \in G$, where $g = \{vp_i\}$, used to exclusively assign customization properties to particular viewpoints $vp \in VP$, we also require uniqueness in G. We use the following notations:

- a group $g' \in G$ is a *direct predecessor* of $g \in G$, if $g <_G g'$ and there is no $g'' \in G$ such that $g <_G g'' <_G g'$.
- a group $g \in G$ is *most specific* for a viewpoint $vp \in VP$, if (1) $vp \in G$, and there is no $g' \in G$ with $vp \in G$ and $g' <_G g$.

The core group has no (direct) predecessors. Without any further restrictions, due to overlapping groups, a view model VM allows for any other groups to have

multiple direct predecessors and viewpoints with multiple most specific groups. We further distinguish between *abstract* and *concrete* groups: a group $g \in G$ is concrete, if it is *most specific* to at least one viewpoint $vp \in VP$, otherwise it is *abstract*.

Example 3. A sample graphical representation of a view model is presented in Fig 2. Here, circles denote groups, where a tree-like structure is used to visualize inclusion hierarchies among groups. Accordingly, $g_3 <_G g_1 <_G g_{core}$ and $g_2 <_G g_{core}$ holds, whereas g_2 and g_2 are unrelated under $<_G$ Viewpoints are denoted by eye-like symbols and dashed lines mark the sets of groups the viewpoint is part of. Therefore, $g_1 \sqcap_G g_2$ holds, because $vp_2 \in g_1 \cap g_2$, and vp_2 has both g_1 and g_2 as its most specific groups. Group g_3 is a singleton group solely containing viewpoint vp_3.

Each viewpoint in a view model aggregates the views assigned to groups of that viewpoint to build a *perspective*. The potential multi-perspectives on the feature model are specified in a *cluster model*.

3.4 Cluster Models

The integration of feature model views and view models imposes a *cluster model* applied to feature models by inferring multiple perspectives from joined views of viewpoints.

Definition 7. *(Cluster Model)*
A cluster model is a triple $CM = (FM, VM, \sigma)$*, where* $FM \in \mathcal{FM}(F)$ *is a feature model,* $VM = (VP, G)$ *is a view model, and* $\sigma : G \to \mathcal{V}_{FM}$ *is mapping function.*

We require every feature of feature model FM to be mapped to at least one view, i.e., for each $f \in F$, there is some $g \in G$ with $\sigma(g) = (F_g, \Phi_g)$ such that $f \in F_G$.

Example 4. The mapping σ between the feature model and the view model in Fig. 2 is denoted by the same hatchings of features and groups. For instance, the core group maps to the gray features, whereas singleton group g_3 maps to a customization feature as well as to a feature shared with group g_2.

For accessing a clustered feature model, a stakeholder chooses a viewpoint according to the corresponding concerns of relevance. Viewpoints $vp \in VP$ refer to aggregated views V_{vp}, i.e., views that result from joining all views mapped to groups of that viewpoint:

$$V_{vp} = \sigma(g_{core}) \oplus \sigma(g_1) \oplus \sigma(g_2) \cdots \oplus \sigma(g_k)$$

where $vp \in g_i$, $1 \leq i \leq k$. By $\mathcal{V}_{CM} \subseteq \mathcal{V}_{FM}$, we denote the set of all views of viewpoints in a cluster model CM on FM. The clustering of feature models into sets of views $\mathcal{V}_{CM}$ of all viewpoints introduces multi-perspectives, i.e., a set $FM_{vp} = p(V_{vp})$ of feature models. Clusterings should preserve the constraints of the original domain feature model, i.e., we require the semantics of multi-perspectives to be consistent with the feature model.

3.5 Consistency of Cluster Models

Despite multi-views, multi-perspectives on feature models do not enforce all views to obey consistency properties, but only those being non-partial, i.e., visible to at least one viewpoint. Therefore, for a cluster model to be *consistent* we require all derivable perspectives to be projected from viewpoints of *FM-consistent* views.

Lemma 3. *A cluster model* $CM = (FM, VM, \sigma)$ *is consistent, if* $\mathcal{V}_{CM} \subseteq \mathcal{V}^C_{FM}$.

Brute-Force Algorithm. A corresponding intuitive brute-force approach for verifying consistency of a cluster model $CM = (FM, VM, \sigma)$ is outlined in Algorithm 1. The algorithm iterates over every view point $vp \in VP$ defined in the

Algorithm 1 Brute-Force Cluster Model Consistency Check

```
input: cluster model CM
for all viewpoints vp ∈ VP do
  for all groups g ∈ G where vp ∈ G do
    V_vp := V_vp ⊕ σ(g)
  end for
  FM_vp := p(V_vp)
  check(FM, FM_vp)
end for
```

view model VM of CM. The set of view $\sigma(g)$ of (partial) groups $g \in G$ containing viewpoint vp are composed as described previously. The restricted feature model perspective FM_{vp} for viewpoint vp is projected from FM and procedure *check* performs the *FM-consistency* checks according to Lemma 1. However, ensuring consistency this way, i.e., viewpoint by viewpoint, is in general not efficiently computable. Even for the simpler case that no equal groups $g_i = g_j, i \neq j$ exist in CM, the maximum number of groups is $\mathcal{O}(2^{|VP|})$, and the maximum number of group views to be composed for a viewpoint $vp \in VP$ is $\mathcal{O}(|G|)$. Checking *FM-consistency* of viewpoints includes (1) traversal of the feature tree in $\mathcal{O}(|F|)$ to check well-formedness and refinement properties (cf. Sect. 3.2), and (2) to decide *satisfiability* of the resulting perspective, which is known to be NP-complete, i.e., reducible to SAT [6].

However, because of inclusions and overlapping within the group hierarchy, many redundant checks are performed that way due commonality between views of viewpoints within related groups. As a solution, we propose a more conservative criterion for cluster model consistency imposing a sufficient, but not necessary requirement, that is verifiable in an efficient, incremental way. Based on the the closedness property of the view composition operator (cf. Prop. 2, we make the following assumptions: (1) the original feature model FM is *satisfiable*, and (2) the consistency of all potential group views in separate ensures consistency of all joined views of every viewpoint.

Incremental Heuristic Algorithm. To check large-scale cluster models, we propose a more conservative criterion imposing a sufficient, but not necessary requirement, that is verifiable in an efficient, incremental way by iterating over groups instead of viewpoints. Therefore, we interpret view models $VM = (VP, G)$ as graphs $(G_c, \rightarrow)$ where:

- nodes $g \in g_c$ refer to *concrete* groups $G_c \subseteq G$, and
- edges $g \rightarrow g'$ connect groups g and g' if $g' <^*_G g$ and each $g'' \in G$ with $g' <^*_G g'' <^*_G g$ is *abstract*, hence $g'' \notin G_c$. In other words, there is no further *concrete* group in the group hierarchy between g' and g.

Starting from the core group, Algorithm 2 incrementally checks for every edge $g \rightarrow g'$ the preservation of *FM-consistency* by considering sets of features added via partial views of abstract groups passed from g to g' (denoted by $F_{g \rightarrow g'}$) and those added by g. Thus, a depth-first-traversal on $(G_c, \rightarrow)$ is performed, where each path segment $g \rightarrow g'$ is checked separately based on previous steps. This incremental heuristic is *reliable* as it ensures *consistency* of a given cluster model.

Algorithm 2 Incremental Heuristic for Cluster Model Consistency Check

Input: FM, $(G_c, \rightarrow)$, σ
Require: $g_{core} \in G_c$
$\forall g \in G_c : g.F = \sigma(g)$ {feature sets mapped and aggregated to groups}
$\forall g \in G_c : g.cons =$ **true** {flag for group views FM-consistency}
$g_{core}.cons := check(g_{core}.F, FM)$ {consistency checks, cf. Lemma 1}
$\forall g \in G_c : g.done =$ **false** {predecessor nodes of node completely checked}
$g_{core}.done :=$ **true**
for all $g \in G_c$ **where** $g.done =$ **true do**
  for all $g' \in G_c$ **where** $g \rightarrow g'$ **do**
    $g'.F := g'.F \cup g.F \cup F_{g \rightarrow g'}$ {add features from predecessors between g and g'}
    $g'.cons := check(g'.F, FM) \wedge g'.cons$ {check consistency preservation}
    if $\forall g'' \in G_c$ **where** $g'' \rightarrow g' : g''.done =$ **true then**
      $g'done :=$ **true** {all predecessors of g' checked}
    end if
  end for
  $G_c := G_c \setminus g$ {check of g done}
end for
return true if $\forall g \in G_c : g.cons =$ **true**

Summarizing, the algorithm uses the following data structures:

- $G_C \subseteq G$ – the sub set of concrete groups, i.e., groups being most specific to at least one viewpoint. Note that we require G_C to always contain the core group, even though it might be abstract.
- $g \rightarrow g'$ – the group hierarchy relation lifted to G_C.
- $g.cons$ – flag for group consistency. The flag is set to false (and stays false) as soon as one potentially inconsistent group view is detected.

- $g.done$ – flag is set to true if all predecessor groups of that group are completely checked. Thus, the traversal can continue at this group.
- $g.F$ – set of features in views of that group incrementally collected from all predecessor group views of that group.
- function $check(F, FM)$ is defined according to the requirements of lemma 1.
- $F_{g \to g'}$ – the union of features mapped into views of abstract groups g'' between g and g'.

Theorem 1. *(Cluster Model Consistency)*
If a cluster model CM passes Algorithm 2 successfully, then CM is consistent.

Proof: First, we have to show that algorithm terminates. According to the definition of the view model hierarchy, the construction of $(G_c, \to)$ results in a connected, directed, and acyclic graph. Therefore, (1) predecessor nodes always exists for the traversal and (2) no cyclic traversals may arise. The preservation of *FM-consistency* can be shown by induction over the traversal of paths in $(G_c, \to$). The induction starts by ensuring *FM-consistency* of the core group view, which is the predecessor of any other group. Then, the algorithm incrementally ensures in every step, i.e., for every *edge* $g \to g'$ to be *consistency preserving* by (1) assuming the view of g to be already checked as *FM-consistent* (induction hypothesis), and (2) the aggregated views for the set $F_{g \to g'}$ to preserve *FM-consistency*. Thus, the incremental traversal ensures concrete views aggregated from views of groups via hierarchical inclusions to preserve *FM-consistency*, if all its predecessors under $<_G$ are *FM-consistent*. Finally, consistency of overlapping group views is given as follows. According to Prop. 2, closedness of *FM-consistent* view composition implicitly ensures views arbitrarily joined for groups $g \sqcap_G g'$ to also preserve *FM-consistency*, even though never explicitly checked.

The opposite direction of Theorem 1 does not hold, i.e., the algorithm may produce false negatives as it may mark groups to be inconsistent, even though all its potential viewpoints have consistent views after aggregation of the overlapping views. For instance, even if the core group is not *concrete*, the algorithm requires its view to be *FM-consistent*.

4 Multi-Perspective SPL Engineering

In SPL engineering, we distinguish the processes of *domain engineering* and *application engineering* [13, 25, 32]. In this section we describe, how to extend these processes with the concepts formalized in the section before to support multi-perspectives on feature models and customization.

4.1 Domain Engineering

In addition to modeling commonality and variability among products in a domain feature model, we propose to model a cluster to support the creation of perspectives in the application engineering process. The domain feature model, as defined in Sect. 2, is created independently. As our cluster approach is a

conservative extension to the domain feature model, it can also be applied to existing feature models. The cluster combines the domain feature model with a view model as formalized in Sect. 3.4. After the view model is created and features are assigned to groups of the view model, viewpoints can be identified to create perspectives in the application engineering process.

4.2 Application Engineering

We extend the application engineering process by creating a valid perspective on the domain feature model before deriving variants for a stakeholder. A stakeholder can choose from groups of the view model. Those groups form the stakeholder's viewpoint.

Due to the fact that the view model is hierarchically structured, all ancestor groups of the groups the stakeholder selects are contained in the viewpoint as well including the core group. A selection of groups forms a viewpoint only, if a valid perspective can be created. Deriving a perspective from a viewpoint is an automated task. This perspective is then used to derive variants in the application engineering process.

4.3 Customization

Stakeholders may have requirements that do not match the features currently available in the domain feature model. To decide about adding new features, a cost-benefit analysis is performed. If the stakeholder's requirements lead to stakeholder-specific features that must only be available for that particular stakeholder, those features could only be added to the domain feature model, if it is possible to restrict their access.

Our multi-perspective approach supports this customization on feature level by introducing the concept of a stakeholder's most specific group. Such a group is only accessible by a single stakeholder and will only be contained in this stakeholder's perspective. Therefore, features that are referenced by this group will not be available in other perspectives than of this particular stakeholder. Thus, stakeholder-specific features are added to the domain feature model without polluting it.

In addition, a feature of the domain feature model can be replaced by a stakeholder-specific feature in a stakeholder's perspective. Therefore, both features, the original and the customized one, must have the same ancestor feature in the domain feature model. Furthermore, the customized feature must be references by the stakeholder's singleton group and the original feature must be referenced by a group that is not contained in the stakeholder's perspective.

4.4 Good Modeling Practices

In our research we identified some good practices in modeling the cluster. The following good modeling practices help to create meaningful models and reduce the computation complexity:

- *Core Features* Features that will be contained in every derived variant are called core features [7]. As the core group of the view model will be included in every perspective, core features must be contained in this group.
- *Optional Features* Features that should not be contained in all perspectives must be declared as optional features in the feature model. This can also be used to make whole sub trees optional. Those sub trees can then be explicitly excluded from perspectives.
- *Potential Dead Features* Due to grouping features in the view model, it is possible to create dead features. A feature is dead, when it is excluded from any perspective [7]. To prevent this, all created groups must be used in at least one perspective.
- *Hierarchical Restriction* Features that are in the lower tree level of the feature model hierarchy should not be referenced by higher level groups in the view model.
- *Constraint Relation* In the case that a feature requires another feature in the feature model and a group in the view model references one of those features, then this group must reference the other feature as well.
- *Exclude a Feature from a Perspective* To restrict the selection of a feature, this feature will excluded from a perspective. That can be achieved by a group that references the feature, but is not contained in the perspective.
- *Replace a Feature in a Perspective* A feature of the domain feature model can be replaced by a customized feature in the stakeholder's perspective. Therefore, both features, must have the same ancestor feature and the customized feature must be references by the stakeholder's most specific group, whereas the original feature must not be contained in the perspective.

5 Application of Multi-Perspective SPL Engineering

In this section we show the applicability of our approach and explain it's technical realization. In addition, we use two case studies to evaluate the generality and scalability of our multi-perspective approach. The first case study describes an SPL for document management systems and consists of 22 features. The second case study describes an SPL for crisis management and consists of 84 features.

5.1 Document Management Case Study

Fig. 4 shows the domain feature model of a document management SPL, a view model and a valid perspective. The domain feature model contains 22 features. The root node `DocumentManagementSystem` represents the document management application.

The application is capable of handling documents of different `DocumentType`s, which are `UnicodeTextType`, `TextType`, `PDFType` and `ImageType`. At least one document format must be handled by the document management. Text from images can be extracted by using `OCR`, which is an optional functionality. Two

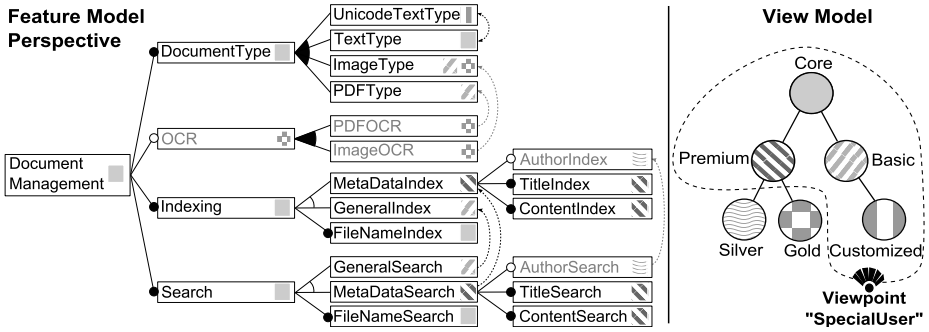


Fig. 4. A perspective on the domain feature model of our document management system case study created by a viewpoint in the view model. Black-colored features and constraints are visible in the perspective, whereas gray-colored ones are not available.

OCR-extractors are provided, the `PDFOCR` and the `ImageOCR`. Each of them requires the according document type. The `Indexing` mechanism is needed to analyze the documents content and prepare it for an efficient search. A `FileNameIndex` is a mandatory feature. Two further index mechanisms could be selected alternatively, `MetaDataIndex` and `GeneralIndex`. The latter one analyzes the document's plain content without considering the structure of the document. In contrast, the `MetaDataIndex` allows for fine grained document analysis and implies the `TitleIndex` and the `ContentIndex`. Optionally, if `AuthorIndex` is selected, the author of a document will be indexed. Hence, the document management system provides `Search` mechanisms. Whereas, each search capability implies that the according index is available.

We group the features according to business concerns in the view model shown on the right side of Fig. 4. Beside the `Core` group, the groups `Premium`, `Silver`, `Gold`, `Basic` and `Customized` are hierarchically ordered in the view model. Features of the feature model are assigned to these groups, whereat the *ImageType* feature is assigned to two groups (`Basic` and `Gold`). A viewpoint of the stakeholder "SpecialUser", references the groups (`Customized`, `Basic`, `Premium` and `Core`). Note, that the group `Customized` represents the most specific group of the stakeholder and the assigned feature `UnicodeTextType` and the according constraint are only available in this stakeholder's perspective. The perspective defined by the viewpoint is visualized on the left side of Fig. 4, whereat black-colored features and constrains are visible and gray-colored ones are not included.

5.2 Crisis Management Case Study

The crisis management SPL allows to derive variants for multiple crisis scenarios, e.g., car or flood crisis. Its feature model contains 84 features and is explained in detail in [16]. We use this case study to show the scalability of our approach by conducting performance measurements.

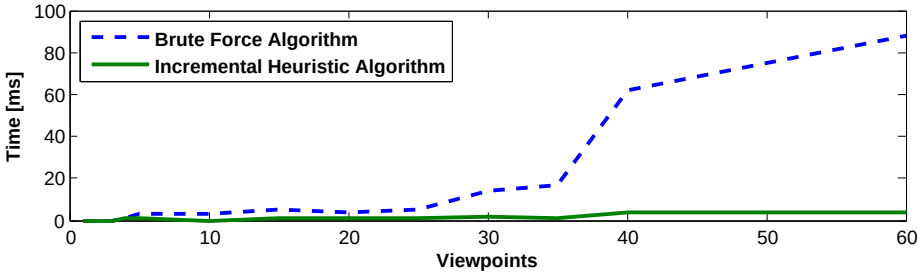


Fig. 5. The performance of the heuristic algorithm is compared to the performance of the brute force algorithm for checking the cluster model consistency.

We implemented the algorithms presented in Sect. 3.5 to check the consistency of the cluster model and measured their performance according to different numbers of viewpoints. We show the results in Fig. 5. As the number of viewpoints increases, the number of groups assigned to a viewpoint increases, accordingly. We evaluate the *incremental heuristic algorithm* (cf. Algorithm 2) against the *brute force algorithm* (cf. Algorithm 1) and show the results in Fig. 5. Comparing both algorithms, for a small number of viewpoints the computing time is almost the same, but for larger numbers the curves detach. The curve of the brute force algorithm start to ascend rapidly at about 35 viewpoints, but slows down at 40 viewpoints, whereas the incremental heuristic algorithm increases only slightly at about 40 viewpoints. The measurements confirm our assumptions from Sect. 3.5 in terms of scalability.

5.3 Technical Realization of Multi-Perspectives

We implemented our concepts of multi-perspectives as plug-ins for the Eclipse[4] integrated development environment (IDE) and combined them with the existing feature modeling and mapping environment *FeatureMapper*[5]. Further information, the source code and a screencast are provided online[6].

Conceptual Design To realize our cluster approach as described in Sect. 3, we need to implement the concepts of view models, feature models and mappings between them. In addition, mechanisms to derive consistent perspectives are needed.

View Model A *View Model* captures the relationship between views and viewpoints. We implement this concept using the Eclipse modeling framework (EMF). We decided on EMF, because it allows to define own structured meta-models in

[4] http://www.eclipse.org
[5] http://featuremapper.org
[6] https://github.com/multi-perspectives/cluster/wiki

Ecore and provides a comprehensive application programming interface (API) to handle meta-models and models [30]. Furthermore, EMF based models seamless integrate with the FeatureMapper and allow to reuse those software artifacts.

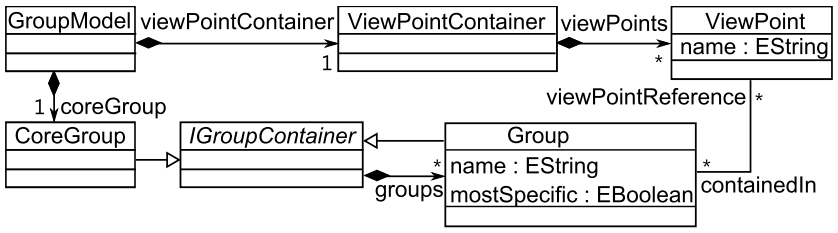


Fig. 6. The view meta-model.

In Fig. 6 we show the Ecore meta-model that is used to instantiate view models according to the definition given in Sect. 3.3. The central root element is the `GroupModel`. It contains the `CoreGroup` and a `ViewPointContainer`. `CoreGroup` and `Group` implement the interface `IGroupContainer` and may contain multiple Sub-`Group`s. The `ViewPointContainer` collects all `ViewPoint`s. A single `ViewPoint` may be contained in multiple `Group`s, and in turn a `Group` may be referenced by multiple `Viewpoint`s. Both have a unique identifier represented by the attribute `name`. In addition, a `Group` may be a stakeholder's most specific group. That is expressed by specifying the attribute flag `mostSpecific`. Such a group is only referenced by the stakeholder's viewpoint and contains stakeholder-specific features only.

Feature Model The FeatureMapper offers a common feature meta-model that allows to specify feature models according to the preliminary explanation in Sect. 2. As this meta-model is EMF-based, we reuse it in our approach to create feature model instances.

Mapping between Feature Model and View Model The FeatureMapper defines mappings between feature models in the problem space and EMF-based solution space artifacts. We use this functionality to create the assignment between features of the feature model and groups of the view model. Therefore, we do not implement a new mapping strategy, but we reuse the one provided by the FeatureMapper.

Cluster Tooling The tooling for our cluster approach is fully integrated in the Eclipse IDE and is a non-invasive extension to the FeatureMapper tool. We provide various editors for focussing on different aspects of the cluster model.

Group View This view represents the view model as a tree, whereas the core group is the root node. Therefore, it is a top down view and is used during

domain engineering to create view groups and viewpoints. In combination with the mapping view provided by the FeatureMapper, features are assigned to view groups using this view. In In Fig. 7 we show the *Group View* used in the document management case study, that is explained above.

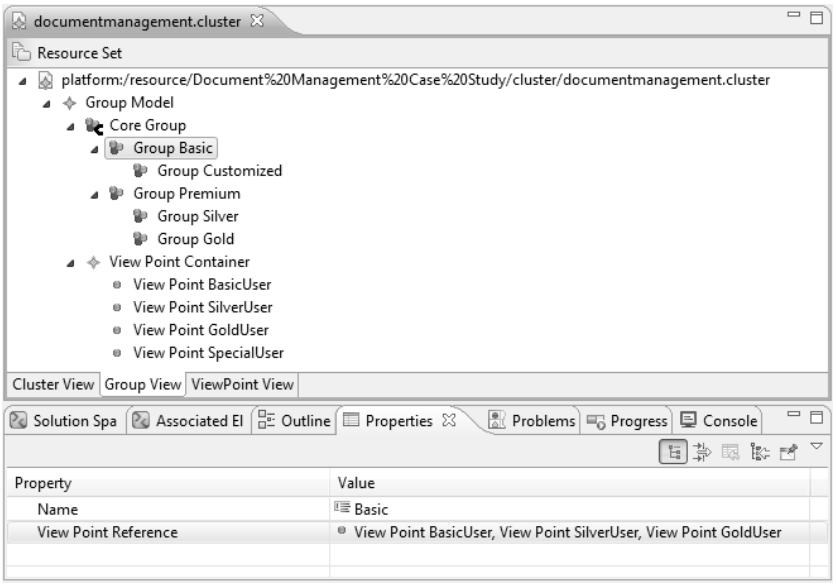


Fig. 7. The group view: Starting from a single viewpoint, its referring groups of views and their inclusion hierarchy are shown.

Viewpoint View This view complements the *Group View* as it represents the view model as a tree, whereas the root node is a selected viewpoint. Therefore it is a bottom up view. The purpose of this view is to show all direct and indirect referenced view groups that belong to a certain viewpoint. In Fig. 8 we show the *Viewpoint View* of the cluster model used in the document management case study described above. In this figure, the viewpoint for a stakeholder named "SpecialUser" is shown.

Cluster View The *Cluster View* visualizes the mapping between feature model and the view model. It shows how features are assigned to view groups and which features belong to a certain view point. In Fig. 9 we show the *Cluster View* of the document management case study described above. In this figure, the viewpoint for a stakeholder named "SpecialUser" is selected, and the referenced view groups and features are highlighted.

FeatureMapper Mapping View We use the *Mapping View* provided by the FeatureMapper to assign features to view groups.

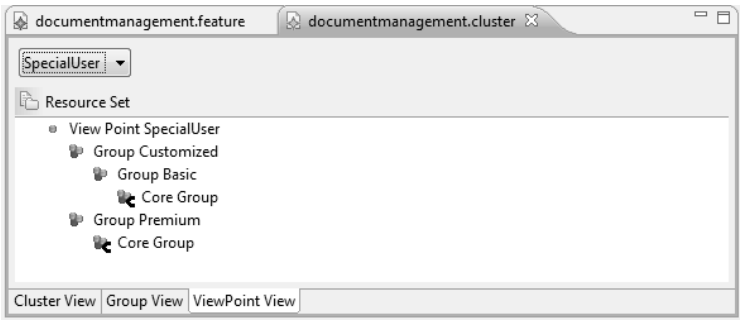


Fig. 8. The viewpoint view: Starting from a single viewpoint, its referring groups of views and their inclusion hierarchy are shown.

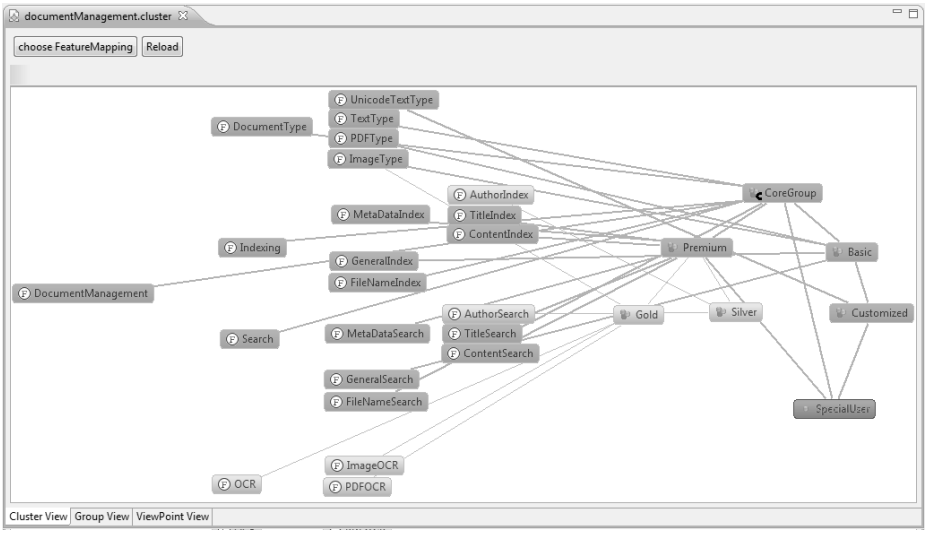


Fig. 9. The cluster view: Visualization of the mapping between view model and feature model. All features and groups that are referenced by the viewpoint "SpecialUser" are highlighted.

Cluster Consistency Check The *incremental heuristic algorithm* (cf. Algorithm 2) is used to check the consistency of the entire cluster. The algorithm is available as an action in the editor menu.

Create a Perspective By selecting a view point in one of the editors, it is possible to derive a perspective from. This action is available in the context menu. By executing it, a consistency check of the resulting filtered feature model is performed. As the check succeeds the perspective is persisted as a filtered feature model in the workspace. This perspective is subsequently used as input for the FeatureMappers variant editor to derive product variants from. Fig. 10 shows

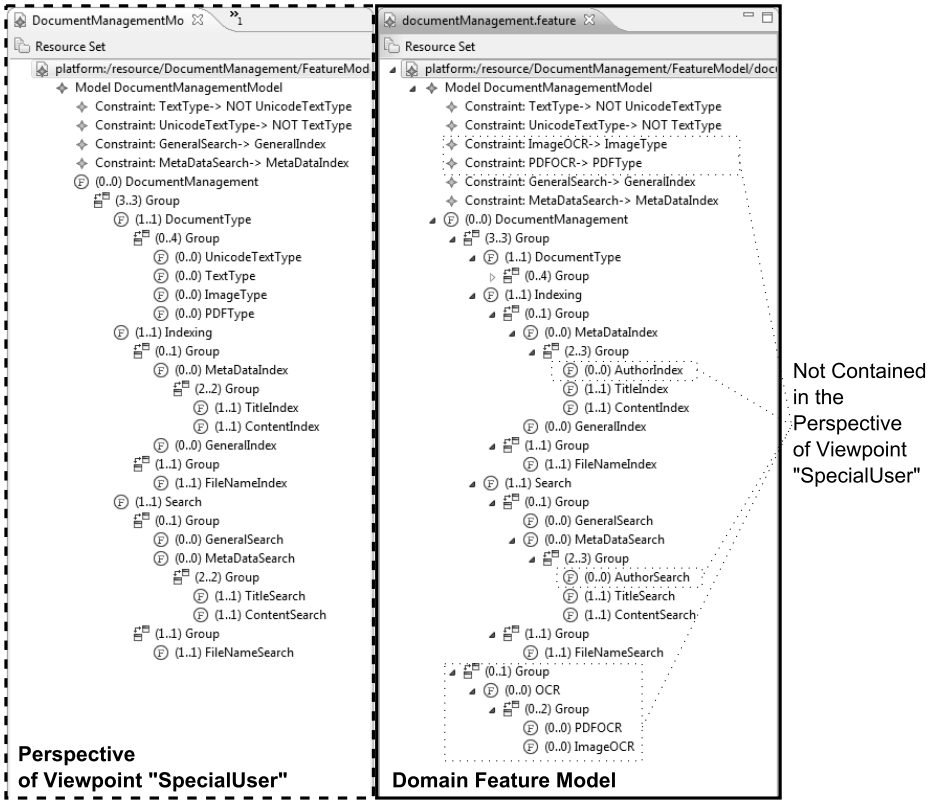


Fig. 10. Comparison between the SpecialUser's perspective (left) and the original domain feature model (right).

the perspective created by the viewpoint of stakeholder "SpecialUser" on the left hand side in comparison with the original domain feature model on the right hand side. Elements that are not contained in the perspective are highlighted by dotted lines in the domain feature model.

6 Related Work

In this section we present related work in the area of views on feature models and the composition of them.

Views on Feature Models Multiple approaches to create views on feature models already exist. Various authors propose to use views in a staged configuration process to partition the feature model [1, 4, 12, 14, 19, 27, 33]. Those approaches address MDSoC, whereas each view restricts the configuration space, dedicated to a stakeholder. Each stakeholder configures parts of the feature model in his own view until all variability is bound. [19] uses the term perspective to address such a stakeholder's view. Acher et al. [3] uses a technique called slicing to create restricted views on feature models. This is similar to the approaches discussed before. Clarke et al. provide a theoretical framework for feature model views that focuses on reasoning about compatibility and reconciliation of initially separated views [11], thus mainly addressing the integration of multiple SPLs. Another approach that addresses MDSoC is the concept of a hyperspace, presented by Ossher et al. [24]. The multi-dimensional hyperspace groups all concerns of system stakeholders in multiple dimensions, whereat a hyperslice encapsulate one concern. Conveying the hyperspace approach to feature modeling, a hyperslice can be considered as a view on a feature model. In contrast to these approaches, we compose multiple views to create a perspective, which we use to restrict the variant space of a feature model. Furthermore, we define explicitly which views form a valid perspective by using viewpoints.

Composition of Views on Feature Models Closely related to our work are approaches that compose views to create integrated views on feature models. Multiple approaches propose to combine disjoint views on multiple domain feature models of independent SPLs [5, 10, 26, 29]. An aspect-oriented approach to compose feature models is proposed by Acher et al. [2]. Thus, the domain feature model is modularized into a base feature model and multiple aspect feature models. Each of them is partial view on the domain and resulting feature models will have incomparable variant spaces. In those approaches, the set of derivable variants of the resulting feature model will contain all variants of the constituents. This is converse to our approach, as we create a perspective by composing views of the same feature model. Therefore, in our approach, variants derivable from a perspective are a subset of the variants of the domain feature model.

7 Conclusion

In this report we present our approach for extending SPL engineering with multi-perspectives on feature models. We give a sound formalization of the cluster concepts and apply them on a case study. In addition, we identified good modeling practices that help to create meaningful clusters. We propose an efficient algorithm to check the cluster consistency and evaluate its performance on another

case study. As the cluster approach is conservative, we could reuse the existing large-scale case study of a crisis management SPL therefore. In future work, we plan to apply our concepts to other scenarios and improve the implemented tooling. We will identify design pattern and propose a work flow for creating and using cluster models. Furthermore, we plan to apply satisfiability (SAT) solver to reason about the satisfiability of perspectives. As a long term goal, we will use our concepts in model-based testing to organize testing concerns, e.g., coverage criteria and technical prerequisites, in view models and to derive reduced representative variant spaces under test. Another long term goal is to use the concepts in a dynamic SPL to create customized perspectives and tailor the reconfiguration space. Therefore, we plan to extend the approach to support different forms of customizations. For instance, we will support the explicit removal of feature in groups to replace them by customized ones. In addition, we will define semantics to use perspectives for restricting edits on features. Concluding, the presented approach for multi-perspectives on feature models bears good prospects for future research in various case scenarios. We consider it a promising concept for tailoring the variant space of a domain feature model to multiple stakeholders.

Acknowledgements The presented work is co-funded by the European Social Fund, Federal State of Saxony and SAP AG within the project #080949335.

References

1. Abbasi, E., Hubaux, A., Heymans, P.: A toolset for feature-based configuration workflows. In: Proceedings of SPLC'11. pp. 65–69 (2011)
2. Acher, M., Collet, P., Lahire, P., France, R.: Composing feature models. In: Proceedings of SLE'09. pp. 62–81 (2009)
3. Acher, M., Collet, P., Lahire, P., France, R.: Slicing feature models. In: Proceedings of ASE'11 (2011)
4. Arnaud Hubaux, Patrick Heymans, P.Y.S.D.D.: Towards multi-view feature-based configuration. In: Proceedings of REFSQ'10. pp. 106–112 (2010)
5. Aydin, E.A., Oguztuzun, H., Dogru, A.H., Karatas, A.S.: Merging multi-view feature models by local rules. In: Proceedings of SERA'11. pp. 140–147 (2011)
6. Batory, D.S.: Feature models, grammars, and propositional formulas. In: Proceedings of SPLC'05. pp. 7–20 (2005)
7. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Information Systems 35, 615–636 (2010)
8. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated reasoning on feature models. In: Proceedings of CAiSE'05. pp. 381–390 (2005)
9. Botterweck, G., Nestor, D.: Towards supporting feature configuration by interactive visualisation. In: Proceedings of ViSPLE'07. pp. 77–86 (2007)
10. van den Broek, P., Galvão, I., Noppen, J.: Merging feature models. In: Proceedings of SPLC'10. pp. 83–89 (2010)
11. Clarke, D., Proença, J.: Towards a theory of views for feature models. In: Proceedings of FMSPLE'10 (2010)
12. Classen, A., Hubaux, A., Heymans, P.: A formal semantics for multi-level staged configuration. In: Proceedings of VaMoS'09. pp. 51–60 (2009)

13. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)
14. Czarnecki, K., Helsen, S., Ulrich, E.: Staged configuration using feature models. In: Proceedings of SPLC'04. pp. 266–283 (2004)
15. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: A framework for integrating multiple perspectives in system development. International Journal of Software Engineering and Knowledge Engineering 2 (1992)
16. Heidenreich, F., Sanchez, P., Santos, J., Zschaler, S., Alferez, M., Araujo, J., Fuentes, L., Kulesza, U., Moreira, A., Rashid, A.: Relating feature models to other models of a software product line: A comparative study of featuremapper and vml*. Transactions on Aspect-Oriented Software Development VII LNCS 6210, 69–114 (2010)
17. Heymans, P., Schobbens, P.Y., Trigaux, J.C., Bontemps, Y., Matulevicius, R., Classen, A.: Evaluating formal properties of feature diagram languages. IET Software 2(3), 281–302 (2008)
18. Hubaux, A., Classen, A., Heymans, P.: Formal modelling of feature configuration workflows. In: Proceedings of SPLC '09. pp. 221–230 (2009)
19. Hubaux, A., Heymans, P., Schobbens, P.Y.: Supporting multiple perspectives in feature-based configuration: Foundations. Tech. Rep. P-CS-TR MPFD-000001, PReCISE Research Centre, Univ. of Namur (2010)
20. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-21, Carnegie Mellon University Pittsburgh, Software Engineering Institute (1990)
21. Lochau, M., Oster, S., Goltz, U., Schürr, A.: Model-based pairwise testing for feature interaction coverage in software product line engineering. Software Quality Journal to appear, 1–38 (2011)
22. Nagle, T.T., Holden, R.K.: The strategy and tactics of pricing. Prentice Hall (2002)
23. Nuseibeh, B., Kramer, J., Finkelstein, A.: Viewpoints: meaningful relationships are difficult. In: Proceedings of ICSE'03. pp. 676–681 (2003)
24. Ossher, H., Tarr, P.: Multi-dimensional separation of concerns using hyperspaces. Tech. Rep. IBM Research Report 21452, IBM Research (1999)
25. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering - Foundations, Principles, and Techniques. Springer (2005)
26. Rosenmüller, M., Siegmund, N.: Automating the configuration of multi software product lines. In: Proceedings of VaMoS'10. pp. 123–130 (2010)
27. Rosenmüller, M., Siegmund, N., Thüm, T., Saake, G.: Multi-dimensional variability modeling. In: Proceedings of VaMoS '11. pp. 11–20 (2011)
28. Schobbens, P.Y., Heymans, P., Trigaux, J.C.: Feature diagrams: A survey and a formal semantics. In: Proceedings of RE'06. pp. 136–145 (2006)
29. Segura, S., Benavides, D., Ruiz-Cortés, A., Trinidad, P.: Automated merging of feature models using graph transformations. In: Post-Proceedings of GTTSE'07. pp. 489–505 (2008)
30. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley, Boston, MA, 2. edn. (2009)
31. Thüm, T., Batory, D., Kästner, C.: Reasoning about edits to feature models. In: Proceedings of ICSE '09. pp. 254–264 (2009)
32. Weiss, D.M., Lai, C.T.R.: Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Professional (1999)
33. White, J., Dougherty, B., Schmidt, D.C., Benavides, D.: Automated reasoning for multi-step feature model configuration problems. In: Proceedings of SPLC'09. pp. 11–20 (2009)