Composing your Compositions of Variability Models

Mathieu Acher, Benoît Combemale, Philippe Collet, Olivier Barais, Philippe Lahire, Robert France

Composing your Compositions of Variability Models

Mathieu Acher[1], Benoit Combemale[1], Philippe Collet[2], Olivier Barais[1], Philippe Lahire[2], and Robert B. France[3]

[1] University of Rennes 1, Inria/Irisa, France
firstname.lastname@irisa.fr
[2] I3S laboratory CNRS, University of Nice Sophia Antipolis, France
firstname.lastname@i3s.unice.fr
[3] Colorado State University, USA
france@cs.colostate.edu

Abstract. Modeling and managing variability is a key activity in a growing number of software engineering contexts. Support for composing variability models is arising in many engineering scenarios, for instance, when several subsystems or modeling artifacts, each coming with their own variability and possibly developed by different stakeholders, should be combined together. In this paper, we consider the problem of composing feature models (FMs), a widely used formalism for representing and reasoning about a set of variability choices. We show that several composition operators can actually be defined, depending on both matching/merging strategies and semantic properties expected in the composed FM. We present four alternative forms and their implementations. We discuss their relative trade-offs w.r.t. reasoning, customizability, traceability, composability and quality of the resulting feature diagram. We summarize these findings in a reading grid which is validated by revisiting some relevant existing works. Our contribution should assist developers in choosing and implementing the right composition operators.

1 Introduction

Designing, developing and maintaining software systems for one customer, one hardware device, one operating system, one user interface or one execution context is no longer an option. Numerous organizations rather need to efficiently produce a large variety of similar software products, for satisfying the requirements of a particular domain. Variability, defined as "*the ability of a software system or artifact to be efficiently extended, changed, customized or configured for use in a particular context*" [1], is pervasive in a growing number of systems, from software product lines (SPLs) [2] to dynamic adaptive systems [3]. When properly managed, variability can lead to order-of-magnitude improvements in cost, time-to-market, and productivity of products.

Models are traditionally employed to formally identify, organize and configure features of a system, automate the generation of products as well as their

verification. A variety of models may be used for different development activities and artifacts – ranging from requirements, source codes, certifications and tests to user interfaces. In an increasing number of scenarios, support for composing models and their variability is becoming more and more crucial [4–16].

Multiple systems. When a multitude of subsystems (modular systems such as software components or services) or artifacts must be combined, several variability descriptions are to be related, organized and finally composed to form a consistent result. This context of use is broad, with first needs on organizing several software product lines with shared variabilities [5], evolving to compositional software product lines [4], in which a complex domain is captured and organized [14] into multiple product lines [8,11] with relations between input product lines' variability models. Handling these relations really lead to both reasoning on the represented configuration sets and maintaining a understandable organization (i.e. a feature hierarchy) for the organizations. The same situation arises when extracting FMs from different software artifacts [12, 17–19]. However these various usages necessitate different interpretations of the FM composition operation to reflect the captured variable assets.

Multiple stakeholders. Together with multiple product lines comes the need to handle different stakeholders on one or several SPLs. Researchers developed techniques for FMs that reflect organisational structures and tasks. For example, Reiser et al. [6] address the problem of representing and managing FMs in SPLs that are developed by several companies in the automotive domain. Several FMs are used and structured hierarchically, so that they can be managed separately by suppliers. The FM composition is then concerned with the propagation of local changes through the hierarchy. In a similar situation, Hartmann et al. [7] used an FM in the context of a multiple SPL supporting several dimensions. It requires the definition of a merging process of FMs during their pre-configurations.

Multiple perspectives. The need for reasoning on FM compositions while manipulating a consistent FM hierarchy is also emphasized by the separation of concerns on variability models. With their increasing complexities and usages, practitioners may define different viewpoints according to different criteria or concerns. The most used viewpoints are the ones defining the user-oriented view (external variability) from the technical features (internal one) [2]. These views have many usages [10, 20, 21], i.e. defining abstraction layers, reflecting organizational structure with specific stakeholders [22], supporting collaborative design [23] or multi-level staged configurations [24]. Separation of these views also means that some relations and compositions must be done at some point to reason over the whole SPL, with references, constraints [25], a reduced form of composite model, and even in a semi-automatic way to synthesize an integrated model [11, 15].

As a result, several modeling artifacts, each coming with their own variability and possibly developed by different stakeholders, should be combined together. In this paper, we first consider the problem of composing feature models (FMs),

a widely used formalism (see Section 2) for representing and reasoning about a set of variability choices (a.k.a. features). We show that several salient variants of composition operators can actually be defined, depending on the semantic properties expected in the composed FM (Section 3). We present four variants with their respective implementations using the FAMILIAR language [15] (Section 4). We also study the different realized trade-offs w.r.t. reasoning, customizability, traceability and composability capacities, as well as quality of the resulting feature diagram (Section 5). We show that existing works [6, 8, 26] and our past attempts [15, 25] can benefit from the new proposed techniques when reasoning, synthesizing feature diagrams, aligning FMs or simply devising new composition-based operators. As a result, the contributions of this paper are:

- the identification of composition mechanisms and semantic properties for building more complex composition-based operators on FMs.
- the survey of four possible variant implementations of such composition-based operators including two new realizations in comparison with previous work [15, 25].
- a reading framework to help on selecting the right composition according to their respective qualities.
- its validation by instantiating some representative existing works.

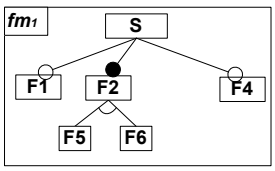
Our contribution should both assist developers in *i)* choosing or devising composition-based operators for FMs and *ii)* choosing the most adequate tool-supported technique to realize these operators.

2 Background

Feature Models (FMs) are a widely used formalism for modeling and reasoning about commonality and variability of a system [27]. A recent survey of variability modeling showed that FMs are by far the most frequently reported notation in industry [28].

An FM is a hierarchical organization of features that aims to represent the constraints under which features occur together in products configurations. When decomposing a feature into subfeatures, the subfeatures may be optional or mandatory or may form *Xor*- or *Or*-groups (see Fig. 1a for a visual representation of an FM). Not all combinations of features (*configurations*) are authorized by an FM. Importantly, the hierarchy imposes some constraints: the presence of a *child* feature in a configuration logically implies the presence of its *parent* (e.g., the selection of F5 implies the selection of F2). The hierarchy also helps to conceptually organize the features into different levels of increasing detail, thus defining an *ontological semantics*.

A *valid* (or legal) configuration is obtained by selecting features in a manner that respects the hierarchy and the following rules: *i)* If a parent is selected, the following features must also be selected - all the mandatory subfeatures, exactly one subfeature in each of its Xor-groups, and at least one of its subfeatures in each of its Or-groups; *ii)* propositional constraints must hold. An FM defines a



(a) fm_1

$[\![fm_1]\!] = \{\{S, F2, F6\},$
$\{S, F2, F5, F1\},$
$\{S, F2, F5, F4\},$
$\{S, F2, F5, F1, F4\},$
$\{S, F2, F6, F1\},$
$\{S, F2, F6, F4\}\}$

(b) Configurations

$\phi_{fm_1} = S \quad // \text{ root}$
$\wedge \quad F2 \Leftrightarrow S \quad // \text{ mandatory}$
$\wedge \quad F1 \Rightarrow S \quad // \text{ optional}$
$\wedge \quad F4 \Rightarrow S \quad // \text{ optional}$
$// \text{ Xor-group}$
$\wedge \quad F5 \Rightarrow F2 \wedge \ F6 \Rightarrow F2$
$\wedge \quad F2 \Rightarrow F5 \vee \ F6$
$\wedge \quad \neg F5 \vee \neg F6$

(c) Boolean formula

Fig. 1: FM, set of configurations and Boolean logic encoding

set of valid feature configurations (see Definition 1). Fig. 1b displays the set of valid configurations characterized by the FM of Fig. 1a.

Definition 1 (Configuration Semantics) *A configuration of an FM fm_1 is defined as a set of selected features. $[\![fm_1]\!]$ denotes the set of valid configurations of fm_1 and is a set of sets of features.*

An FM is usually encoded as a propositional formula, denoted ϕ, and defined over a set of Boolean variables, where each variable corresponds to a feature [29] (see Fig. 1c for the propositional formula corresponding to the FM of Fig. 1a). The terms FM and *feature diagram* are employed in the literature, usually to denote the same concept. In this paper, we make a distinction. We consider that a feature diagram (see Definition 2) includes a feature hierarchy (tree), a set of feature groups, as well as human readable constraints (implies, excludes). The syntactical constructs offered by such feature diagrams are not expressively complete w.r.t propositional logics. Similar to [17], we thus consider that an FM is composed of a feature diagram *plus* a propositional formula ψ (see Definition 3).

Definition 2 (Feature Diagram) *A feature diagram $FD = \langle G, E_{MAND}, G_{XOR}, G_{OR}, I, EX \rangle$ is defined as follows: $G = (\mathcal{F}, E, r)$ is a rooted, labeled tree where $\mathcal{F}$ is a finite set of features, $E \subseteq \mathcal{F} \times \mathcal{F}$ is a finite set of edges and $r \in \mathcal{F}$ is the root feature ; $E_{MAND} \subseteq E$ is a set of edges that define mandatory features with their parents ; $G_{XOR} \subseteq \mathcal{P}(\mathcal{F}) \times \mathcal{F}$ and $G_{OR} \subseteq \mathcal{P}(\mathcal{F}) \times \mathcal{F}$ define feature groups and are sets of pairs of child features together with their common parent feature ; I a set of implies constraints whose form is $A \Rightarrow B$, EX is a set of excludes constraints whose form is $A \Rightarrow \neg B$ ($A \in \mathcal{F}$ and $B \in \mathcal{F}$).*

Definition 3 (Feature Model) *An FM is a tuple $\langle FD, \psi \rangle$ where FD is a feature diagram and ψ is a propositional formula over the set of features $\mathcal{F}$.*

3 Meanings of Composition-based Operators

In an increasing number of contexts, a multiplicity of FMs have somehow to be combined, merged or confronted (i.e., *composed*), for instance, to synthesize an integrated view or reason globally about a system.

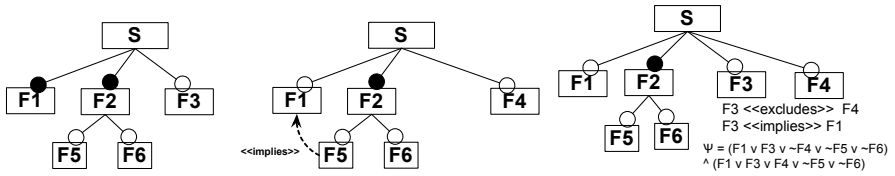
A first illustrative example Let us consider the composition of fm_1, fm_2 and fm_3 (see Fig. 1a, Fig. 2a and Fig. 2b). We denote by $\circ$ a composition operator over FM that computes a new FM. In our specific example, we consider that the composed FM, denoted $fm_{m_{union}}$, should represent the *union* of input sets of configurations of fm_1, fm_2 and fm_3, that is: $[\![fm_{m_{union}}]\!] = [\![fm_1]\!] \cup [\![fm_2]\!] \cup [\![fm_3]\!]$. Such a composition is typically used to build a new SPL offering all the possible configurations supported in at least one of the products or SPLs of an organization or a supplier. Two possible resulting FMs are depicted in Fig. 2c and Fig. 3. Intuitively, when features are selected in the composed FM, it means that the selection of corresponding features (i.e., with the same names) are also valid and both supported in either fm_1 or fm_2 or fm_3. For instance, a partial configuration involving the selections of features F1, F2, and F3 is valid in $fm_{m_{union}}$ since the combination of features F1, F2, and F3 is also valid in fm_2. However it is not possible to both select features F3 and F4 in $fm_{m_{union}}$ since no valid configurations of fm_1, fm_2 and fm_3 are supporting this combination.

Meanings Obviously, the semantics of the previous composition can be in contradiction with the intentions, requirements or simply modeling objectives of a practitioner. First there are different ways of interpreting the way features *match* and are related to each other (e.g., the mapping is not necessarily one-to-one). Second the *configuration semantics* expressed in the composed FM may differ (stakeholders may want to compute the intersection, the reduced product, the difference, etc. of configuration sets instead of the union). Finally the conceptual organization of the features in the resulting FM is another variation. Due to the variety of compositional scenarios exposed in the introduction, there is no one-size-fits-all interpretation when FM have to be composed. In order to address the variations' meanings, we identify common mechanisms and present a generic framework to devise (new) composition-based operators.

3.1 Different Strategies for Matching and Merging

The composition process exposed in the previous example is in line with many works on model composition that consists in two main phases [30, 31]: *i)* the **matching** phase identifies model elements that describe the same concepts in the input models to be composed; *ii)* the **merging** phase where matched elements are grouped together (i.e., merged) to create new elements in the resulting model.

The previous strategy for matching/merging FMs is rather basic and straightforward: features match if they have the same names while the merging consists



(a) fm_2 (b) fm_3 (c) $fm_{m_{union}}$

Fig. 2: A possible composition ($fm_{m_{union}}$) of fm_1, fm_2, and fm_3

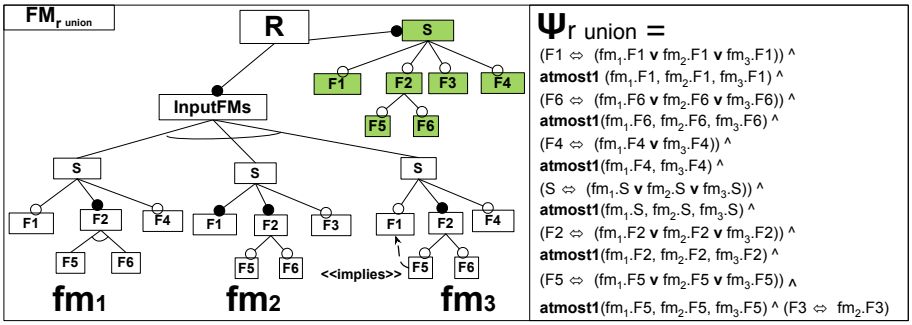


Fig. 3: Composition of fm_1, fm_2, and fm_3, somehow equivalent to $fm_{m_{union}}$. The term atmost1 $(F_1, \ldots, F_n)$ is equivalent to $\wedge_{i<j}(\neg F_i \vee \neg F_j)$

in simply creating new features with the same names S, $F1$, ..., $F6$. However more sophisticated matching and merging mechanisms are needed especially when input FMs are coming from different sources (e.g., suppliers) or when the composed FM should reflect a *view* of the system that does not necessarily include all the original details or feature names.

We give an example in Fig. 4 ($\psi_{r_{other}}$ will be explained in detail in the next section). Firstly, F56 is mapped to features F5 and F6 of input FMs. The intuition is that either selecting F5 or F6 is sufficient to realize the feature F56. In a sense, F56 *abstracts* features F5 and F6 since no distinction is made between F5 and F6 at the level of abstraction of the view (coloured features). Secondly, F1 is no longer present in the composed view. It is another form of abstraction: unnecessary details are removed. Thirdly another feature, named F8, is present in the view and aims to better structure the FM, considering that features F3 and F4 are ontologically closed.

3.2 Different Semantic Properties

The matching and merging mechanisms are the basics for devising a composition operator. However they do not state what are the properties of the composed FM in terms of *configuration semantics* and *ontological semantics*. Let us consider once again the composition of fm_1, fm_2, fm_3 and assume that features F3

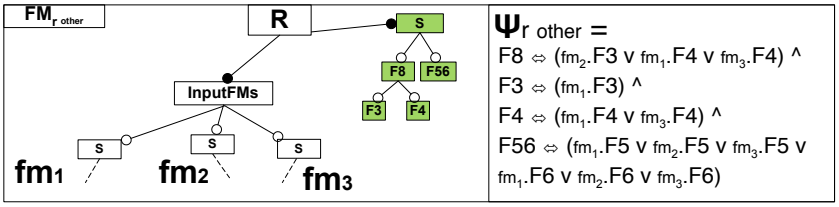


Fig. 4: Another composition of fm_1, fm_2, and fm_3 with different matching/merging strategies and semantic properties

match in the three FMs and are merged as a new feature F3 in the composed FMs. There is still need to establish the meaning of the new feature F3 in terms of configuration, i.e., what is the impact of a selection and deselection of F3 in the composed FM?

Configuration semantics A first interpretation is that the selection of F3 in the composed FM involves the selection of F3 in *one and only one* input FM. (It corresponds to the *union* of configuration sets as considered in the first illustrative example.) The direct impact of this specific semantics is that the selection of F3 induces in turn the selection of F1 (see Fig. 2c and Fig. 3), since there is no SPL that supports F3 without F1. Another more restrictive interpretation is that the selection of F3 in the composed FM forces the selection of *all* features named F3 in input FMs. If this interpretation is applied on all features, the composition intuitively corresponds to the *intersection* of configuration sets. Yet another (less restrictive) interpretation is that the selection of F3 in the composed FM forces the selection of *at least one* features named F3 in input FMs, etc.

Ontological semantics Another important aspect of FMs is the way features are conceptually organized in the tree-based hierarchy. Given a set of configurations, there still exists different candidate FMs yet with different hierarchies [17]. Therefore what the most appropriate feature hierarchy is should be part of the composition. For instance, a practitioner may consider that the feature F3 is more appropriately located below the feature F1 than below the root S in Fig. 3.

4 Variations in the Compositions of Feature Models

A composition operator $\circ$ takes as input a set of FMs and can be customized for supporting different matching/merging strategies and semantic properties (being related to configuration or ontological aspects) in the resulting FM. The following section addresses another important and related problem: How to implement these compositions? Different variants are indeed worth to consider, each having strengths and weaknesses.

4.1 Denotational-based Composition (Logic-based)

The logic-based implementation consists in *i)* encoding the expected configuration set of the composed FM as a Boolean formula ϕ_c *ii)* synthesizing the feature diagram from ϕ_c. Fig. 5a summarizes the process. The first step is to compute ϕ_c. All input FMs (resp. fm_1 and fm_2) are encoded as Boolean formula (resp. ϕ_1 and ϕ_2). Then the composition operator is *denoted* (or translated) in the Boolean logic. If we consider the case of *union* (see the first illustrative example), the denotational operator roughly corresponds to a *disjunction* of formulae (details have been given in [25]). Similar denotations can be applied for computing the intersection, diff, reduced product, etc. of configurations sets. The second step determines an appropriate hierarchy and synthesizes variability information. First we compute the binary implication graph of ϕ_c. It is a directed graph $BIG_c = (V, E)$, V being the set of nodes corresponding to variables of the formula, while the set of edges is formally defined as $E = \{(f_i, f_j) \mid \phi \wedge f_i \Rightarrow f_j\}$. BIG_c is a representation of all logical implications between two variables in ϕ_c and corresponds intuitively to all possible hierarchies of fm_c. Second we compute a directed *minimum spanning tree (MST)* of BIG_c that maximises the parent-child relationships of input FM hierarchies. Finally, other components of the feature diagrams can be synthesized [19, 32]. In Fig. 2c, the resulting synthesized FM corresponds to the first illustrative composition of fm_1, fm_2 and fm_3 (union mode, name-based matching strategy).

4.2 Operational-based Composition (Reference-based)

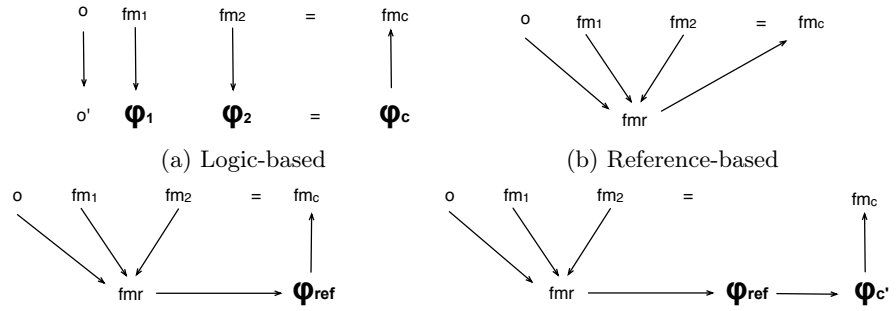
Another radically different implementation is to *reference* input FMs. The key idea is to build a separated FM (i.e., a *view*) that typically contains features with the same names of the input FMs. The features of the view are then related to input features through a set of logical constraints. The result is an FM that both aggregates the input FMs, the view, and the constraints. Fig. 3 depicts the resulting FM on the same kind of composition (union) than previously considered. Other kinds of configuration semantics (e.g., intersection) can be realized by defining another view and logical mapping.

The main difference is that features of input FMs are still present (i.e, the merging strategy differs compared to the denotation-based implementation). Yet it is worth to observe that the configuration semantics expressed in $fm_{r_{union}}$ (see Fig. 3) is *equivalent* to $fm_{m_{union}}$ (see Fig. 2c). The equivalence is defined as follows:

$[\![fm_{m_{union}}]\!] = [\![fm_{r_{union}}]\!]_{|\mathcal{F}_{r_{view}}}$

where $\mathcal{F}_{r_{view}}$ is the set of features in the view (coloured features in Fig. 3) and $A_{|B}$ denotes the projection of for two given sets A and B such that: $A_{|B} \triangleq \{a' \mid a \in A \wedge a' = a \cap B\} = \{a \cap B \mid a \in A\}$. Intuitively it means that the exact same combinations of $S, F1, \ldots, F6$ are authorized in $fm_{r_{union}}$ and $fm_{m_{union}}$. This is due to $\psi_{r_{union}}$ that constraints the way features $S, F1, \ldots, F6$ of $fm_{r_{union}}$ can be combined. For instance, $\psi_{r_{union}}$ states that the selection of F2 should correspond

to at least and at most one of the following features: $fm1.\mathsf{F2}$, $fm2.\mathsf{F2}$, or $fm3.\mathsf{F2}$. Therefore $\mathsf{F2}$ is actually mandatory in $\psi_{r_{union}}$ (as in $\psi_{m_{union}}$).



(a) Logic-based (b) Reference-based

(c) With local synthesis (d) With slicing

Fig. 5: Variants of composition-based operator implementation

4.3 Hybrid

The semantic equivalence of the denotational and operational-based implementations and the last remarks give the idea of going further by *correcting* the view of the reference-based FM. Two equivalent solutions are considered. In both cases, the principle is to *i)* denote the reference-based FM as a formula ϕ_{ref} and then *ii)* synthesize a new feature diagram and FM (see Fig. 5c and Fig. 5d).

Reference-based and Local Synthesis Our goal is to synthesize a new FM that only contains features of $\mathcal{F}_{r_{view}}$. However ϕ_{ref} contains many Boolean variables that may disturb the algorithm. In particular the computation of the implication graph is likely to contain nodes and edges that are not relevant. Furthermore considering all variables of ϕ_{ref} will dramatically increase the computation time. We thus adapt the synthesis procedure so that reasoning operations are only applied *over relevant variables*. For instance, the computation of the implication graph can be realized by checking possible implications only between features of interest. The synthesis of the variability information leads to the same exact feature diagram depicted in Fig. 2c on the previous example.

Reference-based and Slicing Another variant is to eliminate disturbing variables in ϕ_{ref} and obtain a new formula $\phi_{c'}$. Intuitively, non relevant variables are removed by existential quantification in ϕ_{ref}.

Definition 4 (Existential Quantification) *Let v be a Boolean variable occurring in ϕ. $\phi_{|_v}$ (resp. $\phi_{|_{\bar{v}}}$) is ϕ where variable v is assigned the value True (resp. False). Existential quantification is then defined as $\exists v\ \phi =_{def} \phi_{|_v} \vee \phi_{|_{\bar{v}}}$.*

In case of union, intersection, etc., $\phi_{c'}$ is equal to ϕ_c (the formula obtained with a denotational-based approach), i.e., the formula logically represents the exact same valid configurations and the set of variables is exactly the same. Therefore $\phi_{c'}$ can be used afterwards to synthesize an FM: the feature diagram obtained is the same as Fig. 2c.

4.4 Tooling Support

We rely on FAMILIAR (for FeAture Model scrIpt Language for manIpulation and Automatic Reasoning) [15]. The language already includes facilities for importing/exporting, editing, reasoning about FMs and their configurations. Two reasoning back-ends (SAT solvers using SAT4J and BDDs using JavaBDD) are internally used and perform over propositional formulae. Compared to our previous effort [15,19], we extend the language and integrate the new compositional techniques developed in the paper through the form of operations over FMs (aggregateMerge, ksynthesis "over" , etc.). We adapt the Tarjan's algorithm based on corrections reported in [33] to compute the directed MST of binary implication graphs. The code snippet below illustrates how to use the four implementation variants on the illustrative example of the paper. The reference [34] provides a comprehensive tutorial and numerous examples.

```
fm1 = FM (S : ..) fm2 = FM (S : ..) fm3 = FM (S : ..) // input feature models

fmMUnion = merge union { fm1 fm2 fm3 } // logic-based
fmRUnion = aggregateMerge union { fm1 fm2 fm3 } // reference-based

fm6 = extract fmRUnion.S // basic extraction (features are all optionals)
fm7 = slice fmRUnion including fmRUnion.S* // slicing (same FD + formula than fmMUnion)
fm8 = ksynthesis fmRUnion over fm5.S* // local synthesis (same FD but formula differs)
```

5 A Framework for Composing your Compositions

Users of composition operators for FMs have to define a specific semantics (or reuse an existing one, see left part of Fig. 6) and then select an appropriate implementation (see right part of Fig. 6). In this section, we provide a *reading grid* and *practical illustrations* in order to assist users in customizing a composition adapted to their requirements.

5.1 Comparison Framework and Reading Grid

We first discuss and compare the pros and cons of each implementation variant.

Quality of the Feature Diagram The feature diagram (see Definition 2) can be seen as a syntactical view of the configuration set that practitioners or automated tools usually exploit in a forward engineering phase. Given a set of configurations (say s), there may not exist a feature diagram FD such that $[\![FD]\!] = s$. In both cases, as much information as possible should be represented in the resulting feature diagram to approximate or fully represent s. It is known as the property of *maximality* [29]. A violation of maximality can have severe

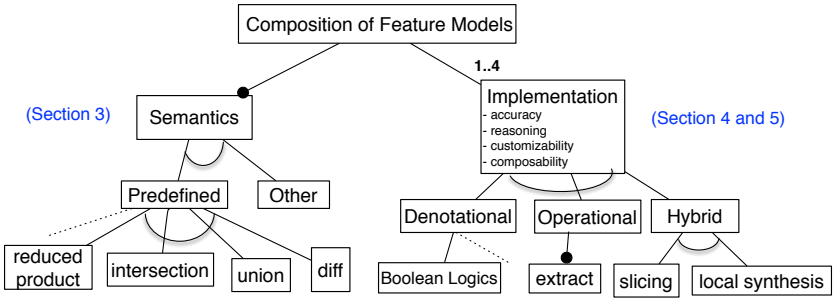


Fig. 6: Composing your Compositions

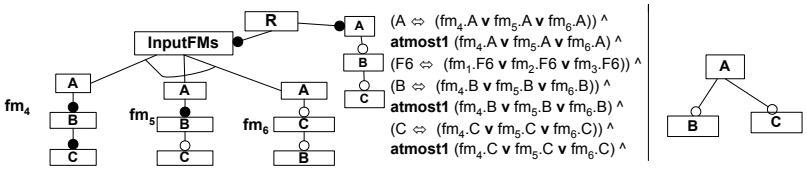


Fig. 7: Composition of fm_4, fm_5, and fm_6 (union): in left-part, the hierarchy leads to an incomplete FM ; in the right-part, a complete and sound FM.

consequences, since in this case the syntactical information may contradict the actual meaning of the FM. For instance, the operational-based composition has the worst maximality since the resulting feature diagram is a super-set of all combinations of features and is a very rough over-approximation of s. In particular, the feature F2 is optional in the feature diagram whereas it is always included in every configuration. The other variants have the best possible maximality since they all rely on the logical synthesis technique that is known to produce a maximal feature diagram [32].

Another expected quality of a feature diagram is its *soundness* and *completeness*. In the reference-based FM, the feature hierarchy of the view is chosen without *a priori* considering the configuration set. Therefore it may happen that the retained hierarchy is not a spanning tree of the implication graph, with the consequence of either precluding some valid configurations (incomplete) or all possible configuration (unsound). We give an example in Fig. 7 (the FM is incomplete). Hybrid techniques (i.e., local synthesis and slicing), that rely on the reference-based FM, could be adapted to fix the problem. The idea is to first set a basic and very flattened hierarchy (i.e., all features are child features of the root) that could not violate any configurations. Then a safe hierarchy could be determined from the implication graph and replaced afterwards.

Reasoning A composition-based operator computes a FM that can be exploited afterwards for *reasoning*, for example, when performing assisted configurations (decision verification and propagation, auto-completion, scheduling of configuration tasks, etc.), when automating analysis over the FMs (e.g., debugging of FMs, comparison of two FMs) [35, 36]. The question we address here is: how to reason about the configurations *once* the resulting FM has been synthe-

sized? The drawback of a reference-based approach is that the reasoning should be performed over (a large amount of) features that are sometimes not relevant. For instance, if we want to perform a configuration over the features F1, F2, ..., F6, it necessarily involves considering the referenced features fm_1.F1, fm_1.F2, ..., fm_3.F6. As a result, the relevant *view* (coloured features of Fig. 3) of the composition is not independent of the other FMs. Furthermore, reasoning operations, usually implemented with SAT solvers or BDD, are not directly usable as such and rather have to be adapted to deal with unnecessary Boolean variables. On the contrary, the denotational-based technique or the use of slicing overcome such limitations since the computed formula only contains relevant Boolean variables and can be exploited independently. The local synthesis is not adequate for simplifying the formula since it calculates a feature diagram that is likely to express an over-approximation of the actual formula. For example, the local synthesis will generate the same feature diagram of Fig. 2c but not $\psi_{m_{union}}$, thus precluding its use for a correct reasoning.

Traceability Features are usually mapped to development artefacts, such as components, models and user documentation . The preservation of the *traceability* between the FM and the artefacts is essential for automatic derivation of products from the configuration of the composed FM. In the case of a denotational-based technique, the mapping between the input FMs is not kept intact because they are replaced by a merged FM. As a result, the selections of features in the composed model may correspond to as many corresponding features in the input FMs. In the case of reference-based FM, the traceability is kept intact so that it is straightforward and immediate to determine the impact of a selection or a deselection on inputs.

Customizability In the previous section, we have shown that there are different mechanisms that can be customized to specify the meaning of a composition. The denotational-based strategy is the most rigid since the matching strategy is assumed to be one-to-one and based on feature names while the merging process creates a new feature with the same name. It can be argued that some pre-processing steps and post-processing steps (renaming, removal of unnecessary features, etc.) can be applied to implicitly implement a matching and merging strategy. However the user effort can be very arduous and error-prone. The task is even more complex when the configuration semantics should be defined. The reference-based techniques are more general since any kinds of logical mappings between *i)* the features planned to be present in the composed FM and *ii)* the features in the input FMs can be defined. A last aspect is the customization of the ontological semantics. Denotational or hybrid techniques provide to users the means to select a sound feature hierarchy through the implication graph. The operational-based approach does not permit such scenarios and therefore the specification of the hierarchy is more error-prone.

Composability Let us consider the composition in union mode and a matching strategy based on feature names (as the example explained in Section 3). The reference-based technique is neither associative nor commutative, e.g., $\circ(\circ(fm_1, fm_2), fm_3) \neq \circ(\circ(fm_1, fm_3), fm_2) \neq \circ(fm_1, fm_2, fm_3)$. Though the configu-

ration set represented is the same, the feature diagrams are different. On the contrary the denotational-based and hybrid techniques are associative and commutative (in the case of union) since the Boolean formulas obtained are the same as previously and the logical operations do have the properties. Finally, it should be noted that a reference-based composition is hardly composable with a denotational-based composition since they are not operating over the same set of features, leading to counter intuitive results. In this case it is needed to slice the reference-based FM in order to align their domains.

	Denotational	Operational	Local Synthesis	Slicing
Diagram quality	*A*	*C*	*A*	*A*
Reasoning	*A*	*C*	*C*	*A*
Customizability	*C*	*B*	*A*	*A*
Traceability	*C*	*A*	*A*	*A*
Composability	*A*	*C*	*B*	*A*

Fig. 8: Comparison of approaches (A: best ; C: worst)

Table 8 summarizes the discussions and results by classifying the *best* and the *worst* solution in a given dimension. Some implementation variants are equivalent for some criteria (e.g., denotational and hybrid techniques compute the same feature diagram). The slicing-based technique fulfils all the criterion and, as such, can be considered as the most suited in the general case. Yet, its *performance* has to be confronted to other composition variants in practical settings (with different kinds of input FMs, matching and semantic properties, etc.). We leave it as future work since it is a *knowledge compilation problem* [37] that deserves a focused and careful attention.

5.2 Instantiating the Framework

We revisit some existing works that target different variability modeling scenarios. The goal is to illustrate the tradeoffs and validate the reading grid.

Devising web configurators from product descriptions In [12], we extract FMs from product descriptions with the ultimate goal of devising product configurators. In this scenario the requirements are as follows. First, the reasoning facilities are crucial to assist end-users in configuring the products. Second, there are no alignment issues since the product descriptions are semi-structured in a tabular data that defines the vocabulary. Third, the FM has to be transformed (e.g., into widgets such as check boxes, lists, images, etc.). The transformation strategy is both automatic (mandatory features are hidden while Xor-groups are transformed as lists of configuration options) and manual (an expert overrides or defines some specific strategies to transform features into widgets). Given all these requirements, the best solution is to rely on a denotational-based implementation that has good reasoning capabilities, computes a high-quality feature diagram, while other criteria (e.g., composability) are not as important.

Modular model checking In order to implement parallel composition of feature transition systems, Classen et al. proposed to compute the *intersection* of two FMs [38]. The composition consists in computing a FM characterizing the intersection of the two configuration sets. (The matching strategy is based

on feature names while the merging strategy is to create a new feature with a same name.) The denotational-based strategy is again the best solution since reasoning is crucial – model checking techniques based on the formula of the composed FM are applied afterwards – the matching strategy is basic while the semantic properties (intersection) can be easily denoted in Boolean logic.

Managing variability of independent suppliers More and more organizations are developing software based on commercially available components from the marketplace and implemented by external suppliers. In such supply chains, variability coming from different sources has to be integrated (see, e.g., [8,24]). Specifically, Hartman et al. [8] presented the problem in the domain of wireless solutions. They introduced the Supplier Independent FM (SIFM) in order to select products among the set of products described by several Supplier Specific FMs (SSFM). The key benefits, already given [8], are as follows *i)* the traceability with suppliers is kept intact ; *ii)* the mappings with suppliers' features can be easily customized. This corresponds exactly to the use of a reference-based FM that exhibits such property.

Moreover our tool-supported proposal can raise two limitations. First, the choice of the feature hierarchy in the SIFM is ad-hoc with the risk of being unsafe (precluding some valid configurations). Second, all features in the SIFM are optionals. In both cases, the hybrid techniques can be used to synthesize a better feature diagram (maximal and sound by construction).

Variability modeling in large-scale organizations Reiser and Weber presented an approach to cope with large diagrams and large-scale organizations in the car industry [6]. The hierarchical organization of product sublines leads naturally to have an integrated view of the system referring to other features. The traceability with the different departments of the organization is crucial. The mappings can be arbitrarily complex since some features of input FMs are either not referenced by the view features (abstraction) or related through complex logical relationships. A denotational-based approach is therefore too rigid. The reference-based approach is the most appropriate solution while the local synthesis or the slicing techniques can be used for correcting the view.

Impact of FM composition on modeling assets An FM is usually associated to an asset (e.g., models) [2,38]. Based on a selection of desired features, a customized model product can be automatically obtained through transformations. Composing such model-based SPLs is naturally emerging (e.g., [4,38]). Given FMs (e.g., fm_1 and fm_2), their respective (sets of) assets (resp. $A1$ and $A2$), and their bindings (materialized as arrows in Fig. 9), the challenge is to compute a new model-based SPL (fm_c, Ac, and a new binding). The major difficulty is that that the resulting composed triplet should be consistent with $[\![fm_c]\!]$. Mirroring the semantics of the composition-based operators on the triplet raises two main challenges (see Fig. 9): *i)* the composition (Ob) of the bindings (see [39] for the underlying challenges) ; *ii)* the composition (Oa) of the assets.

The semantics and implementation is obvious if the rules of the binding are simple (e.g., one-to-one mapping), and the composition operator used to assemble the assets is the law of a mathematical group composed of the assets (closure,

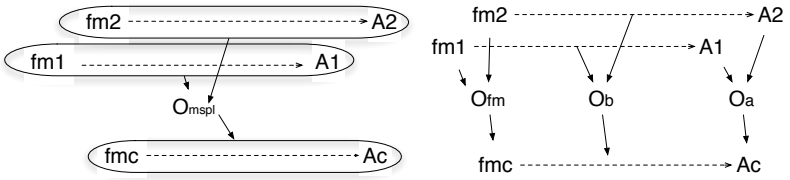


Fig. 9: Composing model-based SPL (left-part) is mirroring the semantics of the composition-based operators on the bindings and the assets (right-part)

associativity, identity and invertibility). Unfortunately, this is in practice seldom the case, e.g., the Common Variability Language provides a powerful action language to express the binding [40], and can be arbitrarily complex. Moreover, most of the composition operators used to derive concrete products by assembling assets do not ensure the properties of the law of a group (e.g., invertibility). Though numerous approaches to model composition have been proposed [30,31], the problem of composing model-based SPLs has not yet deserved enough attention. The trade-offs discussed in the paper are a first step towards automatically mirroring the semantics of compositions operators for model-based SPLs.

6 Conclusion and Perspectives

Composing different variability descriptions from different sources is now a strong need in many engineering contexts. In this paper we have studied the different forms of feature model (FM) compositions, establishing the differences in feature matching and relations, as well as in the expressed configuration and ontological semantics. We have also detailed four different implementations of the composition operation, being based either on the underlying logic or some references between composed FMs. Two implementations are revisited versions from [25] while the two others are new and use forms of slicing and local synthesis over the FMs. We discussed the benefits and drawbacks of each variant using different criteria: the quality of the resulting FM, its customizability, as well as the provided capability of reasoning over the FM and of composing different implementations together. Different practical scenarios of use [6, 8, 12, 24, 38] were presented and a reading grid synthesizes these findings, in the aim of assisting developers choosing and implementing the right compositions.

Our immediate concern is to address one of the challenges opened by our contribution: the impact of the FM composition over the related modeling assets. The tradeoffs made explicit in the proposed reading grid should be reused to identify how to automate a mirroring of the FM composition semantics for model-based product lines.

Acknowledgements. This work was developed in the VaryMDE project (a bilateral collaboration between the Triskell team at INRIA and the Thales Research & Technology) and the CNRS PICS project MBSAR.

References

1. Svahnberg, M., van Gurp, J., Bosch, J.: A taxonomy of variability realization techniques: Research articles. Softw. Pract. Exper. **35**(8) (2005) 705–754
2. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag (2005)
3. Morin, B., Barais, O., Nain, G., Jézéquel, J.M.: Taming dynamically adaptive systems using models and aspects. In: ICSE'09, IEEE (2009) 122–132
4. Bosch, J.: Toward compositional software product lines. IEEE Software **27** (2010) 29–34
5. Buhne, S., Lauenroth, K., Pohl, K.: Modelling requirements variability across product lines. In: RE '05: Proceedings of the 13th International Conference on Requirements Engineering, IEEE (2005) 41–52
6. Reiser, M.O., Weber, M.: Multi-level feature trees: A pragmatic approach to managing highly complex product families. Requir. Eng. **12**(2) (2007) 57–75
7. Hartmann, H., Trew, T.: Using feature diagrams with context variability to model multiple product lines for software supply chains. In: SPLC'08, IEEE (2008) 12–21
8. Hartmann, H., Trew, T., Matsinger, A.: Supplier independent feature modelling. In: SPLC'09, IEEE (2009) 191–200
9. Bošković, M., Mussbacher, G., Bagheri, E., Amyot, D., Gašević, D., Hatala, M.: Aspect-oriented feature models. In: Proceedings of MODELS'10 workshops. MODELS'10, Berlin, Heidelberg, Springer-Verlag (2011) 110–124
10. Hubaux, A., Heymans, P., Schobbens, P.Y., Deridder, D., Abbasi, E.K.: Supporting multiple perspectives in feature-based configuration. Software and Systems Modeling (2011) 1–23
11. Rosenmüller, M., Siegmund, N., Thüm, T., Saake, G.: Multi-dimensional variability modeling. In: VaMoS'11, ACM (2011) 11–20
12. Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P.: On extracting feature models from product descriptions. In: VaMoS'12, ACM (2012) 45–54
13. Clarke, D., Proença, J.: Towards a Theory of Views for Feature Models. In: Proceedings of the First Intl. Workshop on Formal Methods in Software Product Line Engineering (FMSPLE 2010). (2010) 91–100
14. Holl, G., Grünbacher, P., Rabiser, R.: A systematic review and an expert survey on capabilities supporting multi product lines. Information and Software Technology **54**(8) (August 2012) 828–852
15. Acher, M., Collet, P., Lahire, P., France, R.: Familiar: A domain-specific language for large scale management of feature models. Science of Computer Programming (SCP) Special issue on programming languages **78**(6) (2013) 657–681
16. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Netw. **51**(2) (2007) 456–479
17. She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K.: Reverse engineering feature models. In: ICSE'11, ACM (2011) 461–470
18. Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A.: On extracting feature models from sets of valid feature combinations. In: Proceedings of FASE'13. (2013) 53–67
19. Acher, M., Heymans, P., Cleve, A., Hainaut, J.L., Baudry, B.: Support for reverse engineering and maintaining feature models. In: VaMoS'13, ACM (2013)
20. Hubaux, A., Acher, M., Tun, T.T., Heymans, P., Collet, P., Lahire, P.: Separating Concerns in Feature Models: Retrospective and Multi-View Support. In: Domain Engineering: Product Lines, Conceptual Models, and Languages. Springer (2013)

21. Schroeter, J., Lochau, M., Winkelmann, T.: Multi-perspectives on feature models. In: MoDELS'12. Volume 7590 of LNCS. (2012) 252–268
22. Mannion, M., Savolainen, J., Asikainen, T.: Viewpoint-oriented variability modeling. In: Proceedings of the 33rd International Computer Software and Applications Conference (COMPSAC'09), IEEE (2009) 67–72
23. Mendonca, M., Cowan, D.: Decision-making coordination and efficient reasoning techniques for feature-based configuration. Science of Computer Programming **75**(5) (2010) 311 – 332
24. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration through specialization and multilevel configuration of feature models. Software Process: Improvement and Practice **10**(2) (2005) 143–169
25. Acher, M., Collet, P., Lahire, P., France, R.: Comparing approaches to implement feature model composition. In: ECMFA'10. Volume 6138 of LNCS. (2010) 3–19
26. Zaid, L.A., Kleinermann, F., Troyer, O.D.: Feature assembly: A new feature modeling technique. In: Conceptual Modeling (ER'10). (2010) 233–246
27. Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., Wąsowski, A.: Cool features and tough decisions: a comparison of variability modeling approaches. In: Proceedings of VaMoS'12, ACM (2012) 173–182
28. Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wąsowski, A.: A survey of variability modeling in industrial practice. In: Proceedings of VaMoS'13, ACM (2013)
29. Czarnecki, K., Wasowski, A.: Feature diagrams and logics: There and back again. In: SPLC'07, IEEE (2007) 23–34
30. Wimmer, M., Schauerhuber, A., Kappel, G., Retschitzegger, W., Schwinger, W., Kapsammer, E.: A survey on uml-based aspect-oriented design modeling. ACM Comput. Surv. **43**(4) (October 2011) 28:1–28:33
31. Jeanneret, C., France, R., Baudry, B.: A reference process for model composition. In: AOM '08: Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling, New York, NY, USA, ACM (2008) 1–6
32. Andersen, N., Czarnecki, K., She, S., Wasowski, A.: Efficient synthesis of feature models. In: Proceedings of SPLC'12, ACM Press (2012) 97–106
33. Camerini, P.M., Fratta, L., Maffioli, F.: A note on finding optimum branchings. Networks **9**(4) (1979) 309–312
34. Companion web page: `https://github.com/FAMILIAR-project/familiar-documentation/blob/master/manual/composition.md`
35. Benavides, D., Segura, S., Ruiz-Cortes, A.: Automated analysis of feature models 20 years later: a literature review. Information Systems **35**(6) (2010)
36. Thüm, T., Batory, D., Kästner, C.: Reasoning about edits to feature models. In: ICSE'09, ACM (2009) 254–264
37. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. (JAIR) **17** (2002) 229–264
38. Classen, A., Cordy, M., Schobbens, P.Y., Heymans, P., Legay, A., Raskin, J.F.: Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking. IEEE Trans Software Eng (TSE) (2012)
39. Diskin, Z., Maibaum, T., Czarnecki, K.: Intermodeling, queries, and kleisli categories. In: Fundamental Approaches to Software Engineering (FASE'12). Volume 7212 of LNCS., Springer (2012) 163–177
40. Filho, J.B.F., Barais, O., Acher, M., Le Noir, J., Baudry, B.: Generating counterexamples of model-based software product lines: An exploratory study. In: 17th International Conference on Software Product Lines (SPLC'13). (2013)