Comparing Approaches to Implement Feature Model Composition

Mathieu Acher[1], Philippe Collet[1], Philippe Lahire[1], and Robert France[2]

[1] I3S Laboratory (CNRS UMR 6070)
University of Nice Sophia Antipolis, France
{acher,collet,lahire}@i3s.unice.fr
[2] Computer Science Department,
Colorado State University, USA
france@cs.colostate.edu

Abstract. The use of Feature Models (FMs) to define the valid combinations of features in Software Product Lines (SPL) is becoming commonplace. To enhance the scalability of FMs, support for composing FMs describing different SPL aspects is needed. Some composition operators, with interesting property preservation capabilities, have already been defined but a comprehensive and efficient implementation is still to be proposed. In this paper, we systematically compare strengths and weaknesses of different implementation approaches. The study provides some evidence that using generic model composition frameworks are not helping much in the realization, whereas a specific solution is finally necessary and clearly stands out by its qualities.

1 Introduction

The concept of Software Product Line (SPL) [1] is based upon an appealing idea: instead of considering applications individually, the co-development of a family of related programs is planned from the beginning. The family's common features are collected in reusable assets that can be later adapted to derive and fit the requirements of an individual product. In domain and application engineering, *feature models* [2, 3, 4] are widely used to describe a family (e.g., an SPL) in terms of common and variable features. A feature model represents a set of valid combination of features, each one corresponding to an actual product of a family.

Current feature modeling techniques often do not scale up to SPLs with a large number of features and a high degree of variability [5,6]. In these situations, the techniques produce large feature models that are too complex to be easily understood by engineers or analyzed by reasoning tools. Applying separation of concerns principles and providing support for modularising and composing feature models can improve scalability. Yet a study of the literature about SPL engineering demonstrates that providing automated support for composing feature models still remains an open challenge [7,8,5,9,10]. In previous work [11], we designed a set of *composition* operators for feature models and defined semantic properties that must be preserved during composition.

There are several ways to implement the composition operators. On the one hand, previous work in the feature modeling community can be revisited to implement the composition operators. On the other hand, Model-Based Engineering (MBE) and Aspect-Oriented Modeling (AOM) communities have developed a set of model composition techniques and tools. Therefore, there is an interest in determining how these techniques perform with feature model composition and which techniques are the most suitable. The intended audience of this paper are *i)* SPL researchers working on feature modeling techniques or developers of feature modeling tools ; *ii)* researchers involved in the AOM community or more generally dealing with model transformation.

The remainder of this paper is organized as follows. In Section 2, we give an overview of feature models, motivate the need to support a set of composition operators and present their semantic properties. We then discuss the properties we expect in a good implementation of the composition operators (Section 3) so that we can set up an experimental comparison to systematically evaluate and compare the considered implementation techniques (Section 4). Results are reported and interpreted while most suitable approaches are determined and discussed (Section 5).

2 Background and Motivation

2.1 Feature Models

A Feature Model (FM) is a representation of a family, e.g., a family of medical images, in terms of features [4, 3]. Let us consider FM_{ep3} depicted in the right part of Figure 1: A medical image has two *mandatory* features, Modality and Format, which implies that each valid configuration of a medical image should include these two features. There are two alternatives for Modality acquisition: SPEC and PET features form an *Xor*-group (i.e., at least and at most one feature must be selected). An *optional* feature is Anonymized, which states whether all patients metadata of the medical image are included or not. Finally, a medical image Header supports either the format DICOM or Nifti or both of them: DICOM and Nifti form an *Or*-group. A FM thus describes the set of valid feature combinations. Every member of a family is represented by a unique combination of features. In the remainder of the paper, *a combination of selected features is called a configuration of a FM and is represented as a set of features.* In Figure 1, a valid configuration of FM_{ep3} is {MedicalImage, Modality, SPEC, Format, Anonymized, Header, DICOM}.

2.2 Composition Operators

In realistic SPL development, large and monolithic FMs must be built, evolved and analyzed. These tasks are cumbersome, error-prone and costly owing to the large amount of features to be considered by (different) stakeholders [5]. To manage complexity, FMs can be separated and *composed*, with then the crucial need to ensure that relevant properties are preserved during composition. In

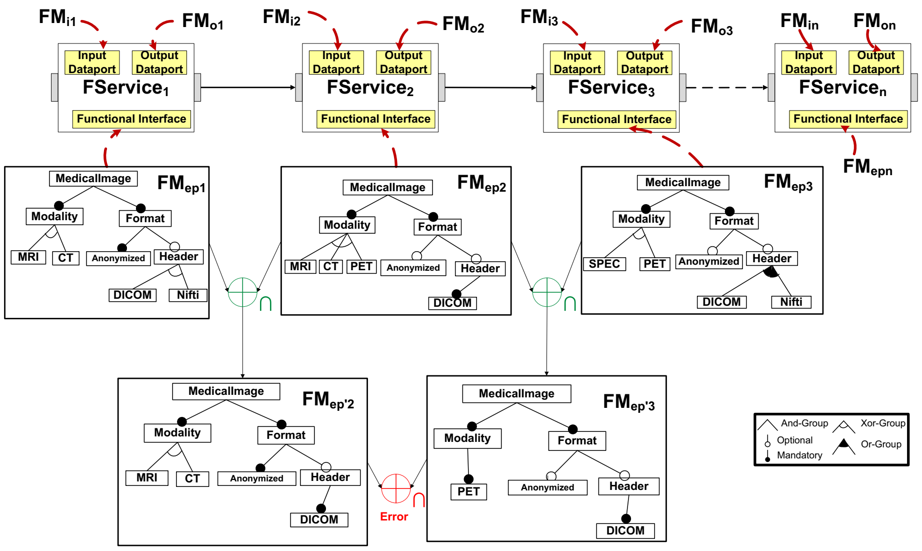


Fig. 1. Chaining Merge of Feature Models

prior work [11], we promote the use of multiple FMs, each one focusing on a well-identified concern and we define a set of composition operators for FMs. Two main composition operators, *insert* and *merge*, were proposed. For each operator, the semantics is given in terms of the expressed configurations and implementation feasibility is demonstrated. Here, we focus on the *merge* operator and detail the preserved properties when two FMs are merged.

Merge Operator Semantics. When two FMs share several features and are different viewpoints of a concern, the goal of the merge operator is to merge the overlapping parts of the two FMs to obtain an integrated model of the system. Two modes are defined for the merge operator. The *intersection* mode is the most restrictive option: the merged FM, $FM_{ep'2}$, expresses the common valid configurations of FM_{ep1} and FM_{ep2}. The *union* mode is the most conservative option: the merged FM, can express either valid configuration of first input FM or second input FM. The variability information associated to features in the merged FM is different according to the merge mode and the properties that one want to preserve. The properties of the merged FM is formalized with respect to the sets of configurations of input FMs. Let f be a FM and $[\![f]\!]$ denotes its set of configurations. The relationship between a merged FM *Result* in intersection mode and two input FMs *Base* and *Aspect* can be expressed as follows:

$$[\![Base]\!] \bigcap [\![Aspect]\!] = [\![Result]\!] \qquad (M_1)$$

The merge operator in the intersection mode is noted: $Base \oplus_{\cap} Aspect = Result$. In the intersection mode, a valid configuration of the merged FM, *Result*, is valid in *Base* and in *Aspect* at the same time. In Figure 1, the DICOM feature is always part of any valid configuration of FM_{ep2} whereas the Nifti feature cannot be part of any valid configuration of FM_{ep2}. As a result, DICOM feature is a mandatory feature of the merged FM $FM_{ep'2}$ while the Nifti feature is not part of the merged FM $FM_{ep'2}$. The reader can check that the following relations hold: $[\![FM_{ep1}]\!] \bigcap [\![FM_{ep2}]\!] = [\![FM_{ep'2}]\!]$ and $[\![FM_{ep2}]\!] \bigcap [\![FM_{ep3}]\!] = [\![FM_{ep'3}]\!]$.

In the union mode, we want to obtain a merged FM that represents the set of configurations of *Base* and *Aspect*. The union of two FMs, Base and Aspect, is a new FM where each configuration that is valid *either* in Base *or* Aspect, is also valid:

$$[\![Base]\!] \bigcup [\![Aspect]\!] \subseteq [\![Result]\!] \qquad (M_2)$$

A more restrictive property in union mode, called *strict union*, is defined as follows:

$$[\![Base]\!] \bigcup [\![Aspect]\!] = [\![Result]\!] \qquad (M_3)$$

2.3 Motivating Scenario

In the grid-based medical imaging community, scientists compose a wide variety of parameterized image services to create processing pipelines, and the lack of variability management mechanisms causes major issues in provisioning and composing such services [12, 13].

We illustrate here how the merge operator can be used. Figure 1 shows three services $FService_1$, $FService_2$ and $FService_3$ connected in sequence. The connection between services implies that some of their entities are dependent in some way. For instance, we consider that the functional interfaces of $FService_i$ which is connected to $FService_{i+1}$ has to be compatible for $i \in 1...n$. In particular, the medical image associated to $FService_i$ must be compatible with the one of $FService_{i+1}$. This implies to check that *i)* FM_{ep1} and FM_{ep2} are consistent and also that *ii)* FM_{ep2} and FM_{ep3} are consistent. It is necessary to check if, e.g., the set of configurations of FM_{ep1} is equal or included in the set of configurations of FM_{ep2} (and vice versa). In this case, the use of the merge operator occurs: The technique is to compute the merge in intersection mode of two FMs. If the merged FM should not represent an empty set of configurations, then there should be at least one configuration that is valid in the former *and* latter FM. The consistency checking can thus be achieved: In the example, such an FM exists when merging FM_{ep1} and FM_{ep2} (see $FM_{ep'2}$) and also when merging FM_{ep2} and FM_{ep3} (see $FM_{ep'3}$). Nevertheless, there is no solution when merging $FM_{ep'2}$ and $FM_{ep'3}$. It implies that $FService_1$, $FService_2$ and $FService_3$ are not compatible.

2.4 Related Work

In the literature, several papers suggest the design and implementation of a merge operator, as in [7], in which separate FMs are used to model decisions taken by different stakeholders and the need to compose and merge FMs is identified. In [8], Hartmann and Trew dealt with multiple product lines and identified several compositional issues, especially the significance of the merging activity. Recently, Hartmann et al. propose a Supplier Independent Feature Model (SIFM) which contains the "super-set of the features from all the FMs of suppliers" [14], corresponding to property (M_2) in union mode. The creation of the SIFM relies on the work described in [10] and further considered in Section 4.1. Reiser and Weber propose to use multi-level feature trees consisting of a tree of FMs in which the parent model serves as a reference FM for its children [5]. Their purpose is mostly to cope with large diagrams and large-scale organizations, rather than different concerns. They thus do not provide operators to merge FMs. A few approaches use multiple FMs during the SPL development (e.g., see [15]). Such contributions do not consider FMs that are sharing some features, whereas this can happen when FMs interact, when multiple perspectives or views on a FM needs to be managed or when SPLs are composed with SPLs.

In [16], an algorithm is designed to automatically determine the kind of relations between two FMs in terms of sets of configuration. In [17], the case of *synchronizing* existing configurations of a FM that have evolved over time (e.g., some features are added) is considered and can be seen as a merge. However the properties preserved by the synchronization are not formalized and the authors consider FMs with attributes and cardinality. The composition operators previously defined are restricted to basic [18] FMs and do not consider such FM formalism.

Other relevant works [3, 19, 11, 9, 10, 18] are discussed and compared in the rest of the paper.

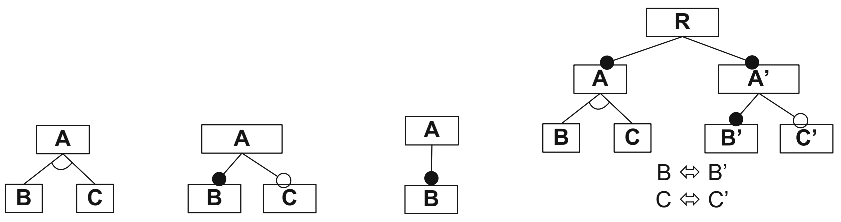
3 Comparison Framework

In this section, we describe the properties we expect in a good implementation (see Section 3.2) of the merge operators and outline how we evaluate different implementation approaches.

3.1 An Illustrative Approach

We use the following implementation of the merge operator, inspired from [3] and [19], to discuss the properties considered to evaluate approaches. The overall idea is that intersection or (strict) union can be realized by maintaining *separate* input FMs and inter-relating them with constraints. In intersection mode, the merged FM consists of a root feature R which joins *Base* and *Aspect* FMs, the roots of *Base* and *Aspect* being child mandatory features of R. Then, features are renamed so that they are disjoint in *Aspect* and *Base* (e.g., priming them in *Aspect*). Finally, constraints are added: P requires P' and P' requires P for each feature P (P' is the renaming of P in *Aspect*).

The merge in intersection mode between *Base* FM of Figure 2a and *Aspect* FM of Figure 2b computes the *Separate* FM shown in Figure 2d. The resulting FM respects the property (M_1) given in Section 2.2 assuming that the primed features A', B', C' and the root feature R are removed in each set of features



(a) *Base* (b) *Aspect* (c) *Expected* (d) *Separate*

Fig. 2. Merging FMs in intersection mode

belonging to $[\![Separate]\!]$. Based on this assumption, *Separate* FM represents exactly the set of configurations of *Expected* FM (see Figure 2c). It is straightforward to check the following equality:

$$\{u \in [\![Separate]\!] \mid u \setminus \{A', B', C', R\}\} = \{\{A, B\}\} = [\![Base]\!] \bigcap [\![Aspect]\!] = [\![Expected]\!]$$

3.2 Properties of a Good Implementation

Quality of the Result. A good implementation of the merge operators should possess Semantics Properties defined in Section 2.2. This is truly the case in our illustration, even if additional effort is required to remove primed features from the set of configurations of the resulting FM. Although the semantics properties are correctly preserved in the resulting FM, the implementation is deficient from several perspectives. In [3, 19], the authors precisely recognize that "the resulting FM should probably be simplified for readability." As this *readability* criterion is too general, we define specific factors that affect success in reading and understanding FMs: Hierarchy Respect, Number of Features and FM Errors.

Hierarchy Respect requires that the resulting FM preserves the hierarchy used in the input models. The essence of FMs have often been defined as feature hierarchy and variability [20]. The hierarchy indeed helps to organize features with increasing detail [20] and loosing the initial hierarchy of input FMs affects the understandability of the model and complicates selections and deselections of features. In Figure 2d, the resulting FM clearly illustrates these issues, with a root feature different from the root features of *Base* and *Aspect*, a new sub-tree and some additional constraints making it confusing.

An interesting property of the merge operator is its ability to reduce the set of features to be considered (i.e., merging two features with the same name into one feature). For example, $FM_{ep'2}$ has only 8 features while input FMs FM_{ep1} and FM_{ep2} have 9 features each in Figure 1. In the illustrative approach, there is no such benefit: The entire set of features of input FMs is included in the resulting FM (see Figure 2d). This becomes worse when merge calls are chained (e.g., when FM_{ep1}, FM_{ep2} and FM_{ep3} are merged, see Figure 1) since the number of features increases and large FMs are produced. We draw the conclusion that a good implementation of the merge operators should produce a composed FM that contains the minimum Number of Features needed to express the desired set of configurations.

With some final observations on Figure 2d one can note that features C and C' are not included in any configuration. Trinidad et al. identify *dead features* and *full-mandatory features* as FM errors [21]. A dead feature is a non-instantiable feature, i.e., a feature that despite being defined in a FM, it appears in no product in the SPL. C and C' are dead features. A child feature in a non-mandatory relationship is a full-mandatory feature if it has to be instantiated whenever its parent feature is, i.e., it is neither an optional nor an alternative feature. C is a full-mandatory feature since it belongs to an *Xor*-group but appears in every configuration. The presence of dead or full-mandatory features introduces incorrect relationships between features and should be avoided [21].

Error Handling. In intersection mode, if the condition $[\![Base]\!] \bigcap [\![Aspect]\!] = \emptyset$ holds, the FM *Result* then defines no configuration at all and is considered as an unsatisfiable or *void* FM [22, 21]. When two input FMs cannot be merged (see $FM_{ep'2}$ and $FM_{ep'3}$, in Figure 1), we consider that there is an error to be *detected* by the merge operator. Error Detection can be done during the merge computation or *a priori*. In the illustrative approach, there is no *a priori* detection. The only way to detect an error is to determine whether the resulting FM of Figure 2d is void or not.

If an error is detected, providing the causes why the two input FMs cannot be merged can assist users to diagnose and repair variability contradictions. The source of error can be a feature or a variability information associated to a feature. For instance, the observation that the (mandatory) feature PET of $FM_{ep'3}$ is not included in $FM_{ep'2}$ can be a conceivable Explanation. In our example, locating the source of errors *during* the computation of the resulting FM is not possible. Automated error-analysis techniques presented in [4, 21, 23] can be applied *once* the FM of Figure 2d is computed but the primed features may disturb the understandability of the diagnosis.

Assumption on Input FMs. The interest here is to determine the degrees of difficulties arising from the handling of several kinds of input FMs (FMs with Constraints, Different Sets of features or Hierarchy mismatch) by an implementation of the merge operator.

Basic FMs support Constraints between features such as *implies* or *excludes.* Constraints crosscut the hierarchy of features (the feature tree) and can be arbitrary propositional formulas [4]. In previous work [11], we intentionally do not consider constraints. Nevertheless handling constraints in FMs can be useful. The presence of constraints alters the set of valid combinations of features but does not change the semantics of the merge operator that still remains to preserve properties (see Section 2.2) in terms of sets of configurations represented by input FMs.

Given the open nature of software architecture or domains, the assumption that FMs to be merged have the same granularity may no longer be valid. The merge operator should be able to deal with input FMs defined on Different Sets of features. Input FMs can also have different hierarchies, e.g., the depth of a feature B in the *Base* FM can be equal to 2 whereas the depth of a feature B in the *Aspect* FM can be equal to 4. Supporting Hierarchy mismatch between input FMs is an interesting quality of a merge operator implementation. As for our illustrative approach, it supports hierarchy mismatch, since there is no assumption made about the hierarchy of input FMs, as well as different set of features or constraints.

Aspects of the Implementation. Finally, additional properties are defined to evaluate some qualities of the implementation. The Ease of Implementation attempts to capture how much effort is required to implement the approach, looking at how built-in mechanisms of considered tools help in the implementation. In the illustration, the implementation is trivial. The Testing Effort property concerns evidences of the respect of the semantics properties, e.g., tests or proof that the implementation is sound, or additional effort to get more confidence in the implementation. Finally, there is need to evaluate the Computational Complexity since the number of calls to the merge operator can be dramatically important (e.g., when a large number n of services are connected in the motivating scenario). In the illustrative approach, the computation of the result FM is solved in linear time [3].

3.3 Comparison Set Up

In order to compare the other approaches according to the defined properties, we set up a comparison protocol described in Figure 3.

The first step is to generate two FMs, *aspect* and *base*. Then merge operator provided by a given approach (see ①) is used to compute the merged FM (R_1 corresponds to the merged FM computed by $Approach_1$, R_2 corresponds to the merged FM computed by $Approach_2$, etc.). The generation process of FMs is manually or randomly performed ②. The way FMs are generated depends on the assumptions made on input FMs by an approach. For example, if an approach is known to *not* support hierarchical mismatch of input FMs, then only input FMs with the same hierarchy are generated. The generation process controls the

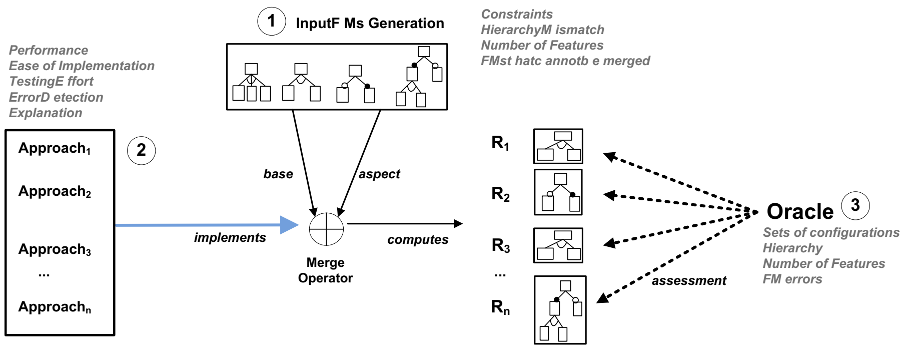


Fig. 3. Comparison Protocol

number of features of input FMs and may propose input FMs that cannot be merged to evaluate the ability of the approach to detect errors. Pre-conditions of the merge operator can be tuned to determine how approaches deal with several kinds of input FMs. In addition, once the merge operator has computed a FM, an oracle (see ③) states whether the result is correct, i.e., in terms of sets of configuration, hierarchy respect, FM errors, etc. For most of the properties, the oracle can be automated and post-conditions of the merge operator be evaluated. The algorithm presented in [16] allows us to reason on the relationship between input FMs and the output FM in terms of sets of configurations. We make use of the tree edit distance metric [24], a common similarity measure for rooted ordered trees, to evaluate the hierarchy respect of the output FMs.

4 Systematic Comparison

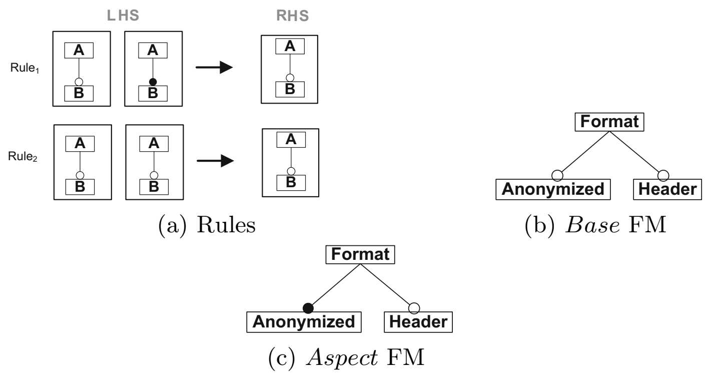
Our selection of approaches for establishing the comparison covers a large spectrum of paradigm and technology. We do not claim to cover all possible solutions but we choose, for each paradigm, at least one possible technique, i.e., AGG and Kermeta for model transformation, Kompose for model composition and an FM-specific solution. For each candidate approach, we report our experience and experimental results considering the set of criteria and the comparison protocol previously described.

4.1 Catalogue Rules

We first consider the work of Segura et al. [10] who propose a catalogue of visual rules to merge FMs using AGG technology [25].

In AGG, a transformation rule is composed mainly of a source graph or Left-Hand Side (LHS) and a target graph or Right-Hand Side (RHS). For each merge rule of the catalogue, LHS consists of two input FM patterns (pre-conditions) and an output FM pattern representing the merging result (post-conditions). In Figure 4a, two rule samples are given. LHS patterns are searched iteratively into the FMs to be merged. Let us show how the catalogue rules apply for the merge in *union* mode with *Base* the FM of Figure 4b and *Aspect* the FM of Figure 4c. The expected merged FM is *Base* FM. The reader can check that the property (M_2) defined in Section 2.2 holds. Rule 1 applies for Anonymized features such that Anonymized is optional in the merged FM. Rule 2 applies for Header features such that the Header feature is optional in the merged FM.

The implementation turned out to be time-consuming and error-prone. The catalogue rules should be modified and maintained according to properties expected in union or intersection mode. The number of rules to specify in the union mode is around 30. Validating the catalogue of rules such that the semantics properties are preserved for any input FMs is still missing. A brute force testing strategy, which consists in generating randomized input FMs and then ensuring each output FM as correct, is not sufficient to cover all cases. Interestingly, AGG implements the mechanism of critical pair analysis which can be used to check consistency of catalogue rules. However, there is no proof about



(a) Rules (b) *Base* FM (c) *Aspect* FM

Fig. 4. Rules to merge in union mode

the *completeness* of the rules. Studying theorem provers and model checkers, as done in [26] *for* refactoring rules (the starting point of [10]), is still to be done and requires intensive research.

The semantics properties currently implemented are limited to the merge in union mode (see property (M_2) in Section 2.2). The intersection mode remains particularly challenging to be implemented. Considering the merge in intersection mode of *Base* (see Figure 5a) and *Aspect* (see Figure 5b), it is hard to specify, in the general case, a rule and an associated pattern that deduce the removal of the feature B. Indeed, the expressiveness of AGG is limited to non recursive-patterns (thus prohibiting traversal of multi-level parent-child relationships) and does not support multi-objects. Handling constraints largely disturbs the strategy based on graph patterns since the presence of constraints may lead to the removal of a feature which may be located elsewhere in the FM.

The elements not mentioned in any of the patterns remain unchanged by default [10]. Then, considering the number of features in the merged FM, there is a risk to unnecessarily adding features and FM errors. Moreover, additional rules are needed to deal with different sets of features of input FMs. As the approach is based on graphs, the hierarchy of the resulting FM is well restored assuming that "The parental relationship between features is equal in all the FMs. That is, a feature must have the same parent feature in all the models in which it appears." [10]. It seems hardly conceivable to deal with hierarchy mismatch. Finally, the approach can detect that two input FMs cannot be merged in intersection mode *during* the iterative application of rules but not *a priori*. Interestingly, negative application conditions (NAC) can provide explanations and precisely locate the source of errors.

4.2 Compositional Approach

The second approach considers the use of Kompose [27, 28] which implements a generic structural composition operator that can be specialized to a particular

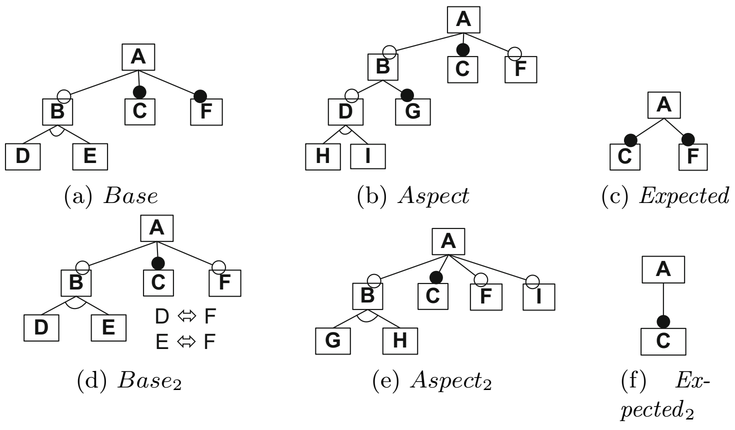


Fig. 5. Non-trivial example of merging FMs in intersection mode

modeling language. We implement the merging rules and strategy proposed in [11] using as much as possible the composition facilities of Kompose.

In Kompose, the composition mechanism is structured in two major phases: (1) The **Matching** phase identifies model elements that describe the same concepts in the input models to be composed; (2) The **Merging** phase where matched elements are merged to create new elements in the resulting model.

Each element type has a *signature* that determines the uniqueness of elements, i.e, two elements with equivalent signatures are merged. A signature is a set of syntactic properties associated with an element type. To achieve our goal we first define the signature of type *Feature* as the name of the feature. The hardest issue is to specify the various types *Operator* (i.e., Xor-, Or-, And-) associated to features. Such operators are likely to be in conflict : two features having the same name may be associated to different operators. The decision to merge them or not and the nature of the resulting operator depends on the intended semantics properties (e.g., as defined in [11] the merging of an *Or*-group with an *Or*-Group gives an *Or*-Group in union mode).

In Figure 5, the merge operator in intersection mode is applied on *Base* (Figure 5a) and *Aspect* (Figure 5b). *Result* (Figure 5c) is the expected FM according to the semantics of the merge operator defined in Section 2.2. The merge operator provided by Kompose has a behaviour which interferes with it. It is obvious that feature A of *Base* and A of *Aspect* must be merged and produces feature A of *Result* (this is exactly what Kompose does automatically). But Kompose applies recursively the same strategy to feature B and this is not what is expected according to *Result*. This shows that a compositional approach only structured in two-stages (matching and merging) is too restrictive for implementing an FM-specific merge operator. In particular, the recursive detection of matching elements is not sufficient since we need a more global vision to decide whether elements should be merged or not. To address this issue we could use the post-directives mechanism provided by Kompose. This would allow automatically removing feature B but this solution is practically hard to implement since it is specific to each composition.

As Kompose implies local reasoning, handling constraints is not conceivable as well (see the removal of feature F in Figure 5f). Moreover due to its recursive merging strategy, Kompose does not handle hierarchy mismatch. Consequently input FMs must be already aligned. Finally, the current approach cannot determine *a priori* when two FMs cannot be merged. Thanks to the case-based reasoning during the matching process, source of errors can be located and accurate explanations can be provided.

4.3 Transformational Approach

Due to the limits previously observed with AGG or Kompose, we decide to leverage the expressiveness of the model manipulation language used to implement the merge operator. We rely on Kermeta [29] an executable, imperative and object-oriented (meta-)modeling language which is designed to define both structures and behaviors of EMOF and Ecore (meta-)models.

We apply the same strategy as with Kompose but without strictly following the compositional approach which consists in match and merging phases. We gain some benefits, notably a better coverage of semantics properties. Now that global and more complex reasoning is possible, some features are not necessary added and less FM errors are generated.

Although the implementation is not obliged to apply a recursive reasoning and to strictly follow the hierarchy during traversal of input FMs, there is still an issue when dealing with different hierarchies. Finally, difficulties arise in constructing a merged FM that preserves properties with the presence of constraints.

4.4 Boolean Logic Based Composition

Enumerating all valid configurations of an FM is usually infeasible. Fortunately, the set of configurations represented by a FM can be compactly described by a propositional formula defined over a set of Boolean variables, where each variable corresponds to a feature. The intersection of two sets of configurations represented by two FMs, *Base*, and *Aspect*, is computed as follows. First, *Base* (resp. *Aspect*) FMs are encoded into a propositional formula ϕ_{base} (resp. ϕ_{aspect}) as defined in [4]. Then, the following formula is computed:

$$\phi_{Result} = (\phi_{base} \wedge \ not(\mathcal{F}_{aspect} \setminus \mathcal{F}_{base})) \ \wedge \ (\phi_{aspect} \wedge \ not(\mathcal{F}_{base} \setminus \mathcal{F}_{aspect}))$$

with $\mathcal{F}_{base}$ (resp. $\mathcal{F}_{aspect}$) the set of features of *Base* (resp. *Aspect*) FM. $\mathcal{F}_{aspect} \setminus \mathcal{F}_{base}$ denotes the complement (or difference) of $\mathcal{F}_{aspect}$ with respect to $\mathcal{F}_{base}$. If we consider *Base* FM of Figure 5a and *Aspect* FM of Figure 5b, then $\mathcal{F}_{aspect} \setminus \mathcal{F}_{base} = \{G, H, I\}$

not is a function that, given a non-empty set of features, returns the Boolean conjunction of all negated variables corresponding to features:

$$not(\{f_1, f_2, ..., f_n\}) = \bigwedge_{i=1..n} \neg f_i$$

Computing the strict union of two sets of configurations represented by two FMs, *Base*, and *Aspect*, follows the same principles and we obtain:

$$\phi_{Result} = (\phi_{base} \wedge \; not(\mathcal{F}_{aspect} \setminus \mathcal{F}_{base})) \; \vee \; (\phi_{aspect} \wedge \; not(\mathcal{F}_{base} \setminus \mathcal{F}_{aspect}))$$

Interestingly, ϕ_{Result} can be simplified. If $\phi_{Result} \wedge f$ is unsatisfiable, the feature F is dead and can be removed. Similarly, the feature F can be identified as a full mandatory feature if $\phi_{Result} \wedge \neg f$ is unsatisfiable. Moreover, the current approach can detect *a priori* that two FMs cannot be merged in intersection mode: In this case, ϕ_{Result} is unsatisfiable. Such operations on ϕ_{Result} can be realized using SAT solvers or BDD representation. The semantics properties are by construction respected. The technique does not introduce FM errors or does not increase unnecessarily the number of features. Constraints in FMs can be expressed using the full expressiveness of Boolean logic and different sets of features can be manipulated. At the moment, ϕ_{Result} is solely a compact representation of the sets of configurations of the expected FM. The hierarchy of the FM and the structuring information (e.g., parent-child relations between features) are still to be constructed. Czarnecki et al. propose an algorithm to construct a FM from Boolean formula [18]. More precisely, the algorithm constructs a tree with additional nodes for feature groups that can be translated into a basic FM. We first experiment their work on a set of input FMs sharing a same set of features and a same hierarchy. The simplifications of the formula ϕ_{Result} described above have been applied and then fed to the algorithm. Importantly, the algorithm indicates all parent-child relationships (mandatory features) and all possible optional subfeatures such that the hierarchy of the merged FM corresponds to hierarchies of input FMs. *And*-group, *Or*-group and *Xor*-group can be efficiently restored in the resulting FM when it was necessary.

The limitations come when different hierarchies of input FMs or different sets of features are proposed to the merge operator. Although the resulting FM is correct in terms of sets of configuration, determining the most suitable hierarchy for the resulting FM requires the intervention of the user since it can be the hierarchy of the *Base* FM, the hierarchy of the *Aspect* FM, or a combination of the two hierarchies. It comes even more challenging when several features are to be removed in intersection mode. As a result, there is need to impose a given FM hierarchy to the resulting FM and the current technique should be adapted.

5 Results and Concluding Remarks

5.1 Results

Figure 6 summarizes our results. ++ is the highest score (i.e., the criteria is fully fulfilled by the approach) whereas −− is the lowest score (i.e., a non acceptable

	Separate App.	Catalog Rules	Composition	Transform.	Boolean Logic
Related Work Technology	[3,19] -	[10,9] AGG	[11] Kompose	[11] Kermeta	[18] SAT/BDD
Quality of the Result					
Semantics Properties	+	=	=	+	++
Hierarchy Respect	−−	++	++	++	+
Number of Features	−−	−	−	=	++
FM errors	−−	−	−	=	++
Aspects of the Implementation					
Ease of Implementation	++	−−	−	=	+
Computational Complexity	++	−	−	=	+
Testing Effort	++	−	−	−	++
Assumption on Input FMs					
Different Sets	++	+	+	+	++
Hierarchy mismatch	+	−−	−−	−	++
Constraints	++	−	−	=	++
Error Handling					
Error Detection	+	+	+	+	++
Explanation	−	+	+	+	−

Fig. 6. Comparison of approaches

solution). We can observe that only FM-specific solutions fully implement semantics properties. Current MBE or AOM solutions have issues related to the intersection or the strict union modes, especially when constraints are present. Strategies to avoid the adding of unnecessary features in the merged FM were difficult to implement. The confidence in modeling solutions appears to be too low (e.g., there is no proof that the set of rules in AGG is comprehensive such that semantics properties are preserved in all cases) and intensive testing effort is required. This is not the case with FM-specific solution which preserves, by construction, the sets of configurations.

Open Issues. Scalability. The manageable size (i.e., number of features) of input FMs is still to be determined. Using Boolean logic, preliminary experiments indicate that on typical propositional formula the algorithm presented in [18] scales up to 300 variables, e.g., the number of features commonly shared by input FMs should not exceed 300 features. Other approaches have scalability issues (100 features in each input FMs is the limit).

Explanation. When two inputs FMs cannot be merged, ϕ_{Result} is unsatisfiable and no FM can be synthesized from ϕ_{Result}. It is only possible to reason at the Boolean logic level (e.g., by computing a small unsatisfiable subset of the formula's clauses) and thus hard to provide the source of errors at the FM level. Rule-based approaches (AGG, Kompose) have better results. They provide precise explanations (e.g., NAC in AGG) when features' relations lead to FMs merging failure. Nevertheless, there is no evidence that the rules are sufficient to cover all merging failures.

Hierarchy mismatch. FM-specific solutions are more efficient to deal with different hierarchies of input FMs (no assumption is made about hierarchies) but the current proposals are not fully satisfying (see Section 4.4).

Revisiting Model-based Solutions. The study provides some evidence that MBE or AOM solutions considered in this paper are not suitable for implementing the merge of FMs. Below we give some possible reasons.

In AOM, many existing approaches to match and merge focus on *structural* similarities between models and on their *syntactical properties.* Most of these approaches treat models as graphical artifacts while (largely) ignoring their semantics. This treatment provides generalizable tools that can be applied to many different modeling notations. Our first intuition was to resolve every syntactical conflict and to reason recursively on the hierarchy of FMs – a classical approach in model composition. However, complex reasoning that takes into account the semantics of FMs is required to compute the combination of two or more FM elements into new FM elements. The experimentation of MBE techniques gives an insight to the characterization of FMs composition. A merging strategy mainly based on syntactical properties (as applied with AGG, Kompose and Kermeta) is likely to fail so that we can now consider that FMs composition is *not* purely structural. On the contrary, semantical transformations or semantics preserving model composition are needed to preserve the semantics properties of model. An open question in this area is how to achieve semantics preservation, both formally and practically. For instance, recent work on behavioural models has concentrated on establishing semantic relationships between models (e.g., see [30]). Merging FMs can be seen as a non-trivial case of semantics preserving model composition. Currently, model composition techniques are not necessary dedicated to support semantics preserving model composition: This is another way to interpret the difficulties of the modeling techniques considered in this paper. Nevertheless, the selection of approaches in the present study does not pretend to be comprehensive regarding MBE or AOM solutions. Other solutions based on different paradigms or technologies (e.g., QVT) are still conceivable and may successfully implement a merge operator. For instance, graph transformation tools with advanced transformation language constructs or supporting many-to-one transformations [31] may help to better cover semantics properties.

5.2 Future Work

The implementation of a merge operator for FMs is an interesting challenge for MBE and AOM techniques. Other modeling approaches and technologies can be considered and may emerge to outperform the solutions considered in this paper. Nevertheless, the use of Boolean logic turns out to fulfill most of the criteria expected from a merge operator. As future work, we plan to accurately determine for which amount of features the logic-based approach scales and to fully support different hierarchies of input FMs. The use of CSP solvers can also be considered in addition to SAT and BDD techniques.

A longer term perspective is to consider the implementation of *diff* and *refactoring* [9, 16] operations for FMs. These operators are commonly used in MBE for various kinds of models, but the specificity and the semantics properties of FMs should be taken into account. The efficiency of modeling techniques can be evaluated for diff and refactoring of FMs as similarly done for the merge operator. Another research direction is to consider other formalisms of FM including cardinality-based FMs and feature attributes. In this case, the sole use of Boolean logic is not sufficient to represent the semantics of FMs: MBE and AOM techniques may provide interesting support and built-in mechanisms to deal with such extended formalisms.

Acknowledgments. We thank Sergio Segura and Steven She for sharing with us their implementation related to work described resp. in [10] and [18].

References

1. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
2. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA). Technical Report CMU/SEI-90-TR-21, SEI (November 1990)
3. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Netw. 51(2), 456–479 (2007)
4. Batory, D.S.: Feature models, grammars, and propositional formulas. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 7–20. Springer, Heidelberg (2005)
5. Reiser, M.O., Weber, M.: Multi-level feature trees: A pragmatic approach to managing highly complex product families. Requir. Eng. 12(2), 57–75 (2007)
6. Tun, T.T., Heymans, P.: Concerns and their separation in feature diagram languages - an informal survey. In: Proceedings of the Workshop on Scalable Modelling Techniques for Software Product Lines (SCALE@SPLC 2009), pp. 107–110 (2009)
7. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged Configuration through Specialization and Multilevel Configuration of Feature Models. Software Process: Improvement and Practice 10(2), 143–169 (2005)
8. Hartmann, H., Trew, T.: Using feature diagrams with context variability to model multiple product lines for software supply chains. In: SPLC 2008, pp. 12–21. IEEE, Los Alamitos (2008)
9. Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., Lucena, C.: Refactoring product lines. In: GPCE 2006, pp. 201–210. ACM, New York (2006)
10. Segura, S., Benavides, D., Ruiz-Cortés, A., Trinidad, P.: Automated merging of feature models using graph transformations. In: Lämmel, R., Visser, J., Saraiva, J. (eds.) Generative and Transformational Techniques in Software Engineering II. LNCS, vol. 5235, pp. 489–505. Springer, Heidelberg (2008)
11. Acher, M., Collet, P., Lahire, P., France, R.: Composing Feature Models. In: Gašević, D. (ed.) SLE 2009. LNCS, vol. 5969, pp. 62–81. Springer, Heidelberg (2010)
12. Acher, M., Collet, P., Lahire, P., Montagnat, J.: Imaging Services on the Grid as a Product Line: Requirements and Architecture. In: Service-Oriented Architectures and Software Product Lines (SOAPL 2008), at SPLC 2008, IEEE, Los Alamitos (2008)

13. Acher, M., Collet, P., Lahire, P., France, R.: Managing Variability in Workflow with Feature Model Composition Operators. In: 9th International Conference on Software Composition (SC 2010), June 2010. LNCS. Springer, Heidelberg (2010)
14. Hartmann, H., Trew, T., Matsinger, A.: Supplier independent feature modelling. In: SPLC 2009, pp. 191–200. IEEE Computer Society, Los Alamitos (2009)
15. Tun, T.T., Boucher, Q., Classen, A., Hubaux, A., Heymans, P.: Relating requirements and feature configurations: A systematic approach. In: SPLC 2009, pp. 201–210. IEEE Computer Society, Los Alamitos (2009)
16. Thüm, T., Batory, D., Kästner, C.: Reasoning about edits to feature models. In: ICSE 2009, pp. 254–264. IEEE, Los Alamitos (2009)
17. Kim, C.H.P., Czarnecki, K.: Synchronizing cardinality-based feature models and their specializations. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 331–348. Springer, Heidelberg (2005)
18. Czarnecki, K., Wasowski, A.: Feature diagrams and logics: There and back again. In: SPLC 2007, pp. 23–34 (2007)
19. Heymans, P., Schobbens, P.Y., Trigaux, J.C., Bontemps, Y., Matulevicius, R., Classen, A.: Evaluating formal properties of feature diagram languages. Software, IET 2(3), 281–302 (2008)
20. Czarnecki, K., Kim, C.H.P., Kalleberg, K.T.: Feature models are views on ontologies. In: SPLC 2006, pp. 41–51. IEEE, Los Alamitos (2006)
21. Trinidad, P., Benavides, D., Durán, A., Ruiz-Cortés, A., Toro, M.: Automated error analysis for the agilization of feature modeling. J. Syst. Softw. 81(6), 883–896 (2008)
22. Batory, D., Benavides, D., Ruiz-Cortés, A.: Automated analysis of feature models: Challenges ahead. Communications of the ACM (December 2006)
23. White, J., Schmidt, D.C., Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated diagnosis of product-line configuration errors in feature models. In: SPLC 2008, pp. 225–234. IEEE, Los Alamitos (2008)
24. Bille, P.: A survey on tree edit distance and related problems. Theor. Comput. Sci. 337(1-3), 217–239 (2005)
25. Taentzer, G.: AGG: A graph transformation environment for modeling and validation of software. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 446–453. Springer, Heidelberg (2004)
26. Gheyi, R., Massoni, T., Borba, P.: A theory for feature models in alloy. In: Proceedings of First Alloy Workshop, pp. 71–80 (2006)
27. Reddy, Y.R., Ghosh, S., France, R.B., Straw, G., Bieman, J.M., McEachen, N., Song, E., Georg, G.: Directives for composing aspect-oriented design class models. In: Rashid, A., Aksit, M. (eds.) Transactions on Aspect-Oriented Software Development I. LNCS, vol. 3880, pp. 75–105. Springer, Heidelberg (2006)
28. Fleurey, F., Baudry, B., France, R.B., Ghosh, S.: A generic approach for automatic model composition. In: Giese, H. (ed.) MODELS 2008. LNCS, vol. 5002, pp. 7–15. Springer, Heidelberg (2008)
29. Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving executability into object-oriented meta-languages. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 264–278. Springer, Heidelberg (2005)
30. Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., Zave, P.: Matching and merging of statecharts specifications. In: ICSE 2007, pp. 54–64. IEEE, Los Alamitos (2007)
31. Mens, T., Gorp, P.V., Varró, D., Karsai, G.: Applying a model transformation taxonomy to graph transformation technology, March 2006. Electronic Notes in Theoretical Computer Science, vol. 152, pp. 143–159 (2006)