Composing Feature Models

Mathieu Acher, Ph. Collet, Philippe Lahire, Robert France

Composing Feature Models*

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

[1] University of Nice Sophia Antipolis,
I3S Laboratory (CNRS UMR 6070),
06903 Sophia Antipolis Cedex, France
{acher,collet,lahire}@i3s.unice.fr
[2] Computer Science Department,
Colorado State University,
Fort Collins, CO 80523, USA
france@cs.colostate.edu

Abstract. Feature modeling is a widely used technique in Software Product Line development. Feature models allow stakeholders to describe domain concepts in terms of commonalities and differences within a family of software systems. Developing a complex monolithic feature model can require significant effort and restrict the reusability of a set of features already modeled. We advocate using modeling techniques that support separating and composing concerns to better manage the complexity of developing large feature models. In this paper, we propose a set of composition operators dedicated to feature models. These composition operators enable the development of large feature models by composing smaller feature models which address well-defined concerns. The operators are notably distinguished by their documented capabilities to preserve some significant properties.

1 Introduction

Clements et al. define a software product line (SPL) as "*a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*" [1]. SPL engineering involves managing common and variable features of the family during different development phases (requirements, architecture, implementation), to ensure that family instances are correctly configured and derived [2]. In this context, Model-Driven Engineering is gaining more attention as a provider of techniques and tools that can be used to manage the complexity of SPL development.

In model-based development of SPLs, feature models (FMs) [3,4] are widely used to capture SPL requirements in terms of common and variable features. From an early stage (e.g. requirements elicitation) to components and platform modeling, FMs can be applied to any kind of artefacts (code, documentation, models) and at any level of abstraction. As a result, FMs can play a central role in managing variability and product derivation of SPLs (e.g., see [5,6,7]).

* This work was partially funded by the French ANR TL FAROS project.

Like other model-based approaches, SPL engineering now faces major scalability problems and FMs with thousands of features are not uncommon [8, 9]. Creating and maintaining such large FMs can then be a very complex activity [10, 11, 12, 13, 14, 15]. This problem indicates a need for tools that developers can use to better manage complexity. One way that this can be done is to provide the means to separate the concerns or the business domains in an SPL. Our work focuses on an approach that puts FMs at the center of SPL management. The separation of concerns approach we propose enables stakeholders to manage and maintain FMs that are specific to a business domain, a technological platform or a crosscutting concern.

In this paper, we propose generic composition operators to compose FMs in order to produce a new FM. The proposed operators have been determined through a classification of possible manipulations when composing elements of two FMs. This classification is inspired by the similar distinctions made when composing models (introduction, merging, modification, extension) [16]. The proposed *insert* operator supports different ways of inserting features from a crosscutting FM into a base FM. Depending on the inserted and targeted feature nodes, we determine whether the insertion preserves the set of configurations determined by the input FMs. This preservation property is called the *generalization* property. We also propose a *merge* operator that is capable of combining matching features in two input FMs. This operator is defined using the *insert* operator and similar properties are also determined.

The remainder of this paper is organized as follows. Section 2 describes the motivation for separating and composing FMs through an example. Section 3 sets out the rationale behind the design of the proposed composition operators and discusses properties that are used to characterize the provided operators. Section 4 and Section 5 detail the insert and merge operators and illustrate their use on the example presented in Section 2. Section 6 discusses related work. Section 7 describes future work and concludes this paper.

2 Motivation

The plethora of *feature* definitions [17] suggests that FMs can be used at different stages of the SPL development, from high-level requirements to code implementation. In this paper, FMs are considered from a general perspective in that FMs are not restricted to a specific development phase. As a result, a FM can just as well describe a family of software programs, a family of requirements or a family of models.

2.1 Feature Model

FMs organize a hierarchy of features while explicitly specifying the variability [18]. Features of a FM are nodes of a tree represented by strings and related by various types of edges [19]. The edges are used to progressively decompose features into more detailed subfeatures. (The tree structure starts from the *root*

feature, which is then the parent of its child features and so on.) Some mechanisms are also used to express variabilities in a FM. Hence, a group of child features can form an *And*-, *Xor*-, or *Or*-groups. Features in an *And*-group can be either mandatory or optional subfeatures of the parent feature. There are some rules to determine whether a FM is well-formed or not. For example, there cannot be an *And*, *Or* or *Xor*-group with only a single child. In Fig. 1, the concept of person is represented as a FM, whose root feature is Person. Information associated to a person includes housing, transport and telephone, which are mandatory features. The transport feature consists of either a car or an other kind of transport. These child features are mutually exclusive and thus are organized in a Xor-group. The housing feature is composed of any combination of an address, a street name or a street number feature.

Since their original definition by Kang et al. [3], several FM notations have been proposed [19]. The FM language used throughout this paper supports standard structures previously described, but we do not consider directed acyclic graph structures and do not deal with constraints defined across features, whether they are internal or between several FMs. Nevertheless, taking into account constraints on FM is part of our future work (see Section 7).

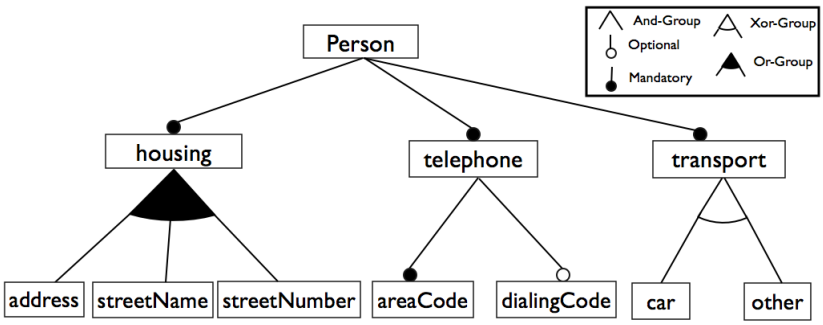


Fig. 1. A feature model representing the concept of person

A FM is a representation of a family and describes the set of valid feature combinations. Every member of a family is thus represented by a unique combination of features[3]. In the remainder of the paper, a combination of selected features is called a *configuration* of a FM. A configuration is valid if all features contained in the configuration and the deselection of all other features is allowed by the FM. The validity of a configuration is determined by the semantics of FM that prevents the derivation of illegal configurations. A FM is a characterization of a set of valid configurations.

The semantics of a FM can be expressed in terms of the following rules: *i)* if a feature is selected, its parent must also be selected. The root feature is thus

[3] A member of a family can be an "instance", a "product", a "program", etc. All these terms are equivalent. Their uses depend on the kind of family represented.

always included in any configuration; *ii)* If a parent is selected, all the mandatory features of its And group are selected; *iii)* If a parent is selected, exactly one feature of its Xor-group must be selected; *iv)* If a parent is selected, at least one feature of its Or-group must be selected (it is also possible to select more than one feature in its Or-group). A valid configuration of the FM depicted in Fig. 1 follows:
$\{Person, housing, telephone, transport, address, streetName, areaCode, car\}$

2.2 A running Example

We use the following example to illustrate the FM composition operators described in this paper. The example is complex enough to illustrate composition needs.

In Fig. 2, the concept of person is designed from a general perspective and described as a FM. It acts as a base or primary model that may not provide all the elements required by an application or system of a person, that is, it may be augmented with other features describing different aspects of a person.

We explain how this base model can be composed incrementally with other FMs describing different aspects of features in the base model. These other FMs are called *aspects*. Let us take a first aspect called *Service Provided*, which deals with the services that may be offered to a person and another aspect called *Transport*, which addresses the kinds of transport that may be used by a person. These two aspects are orthogonal to the concept of person. Furthermore, they are not particularly applied to the concept of person and thus can be composed with other base models, e.g., representing an hotel or a nursing home.

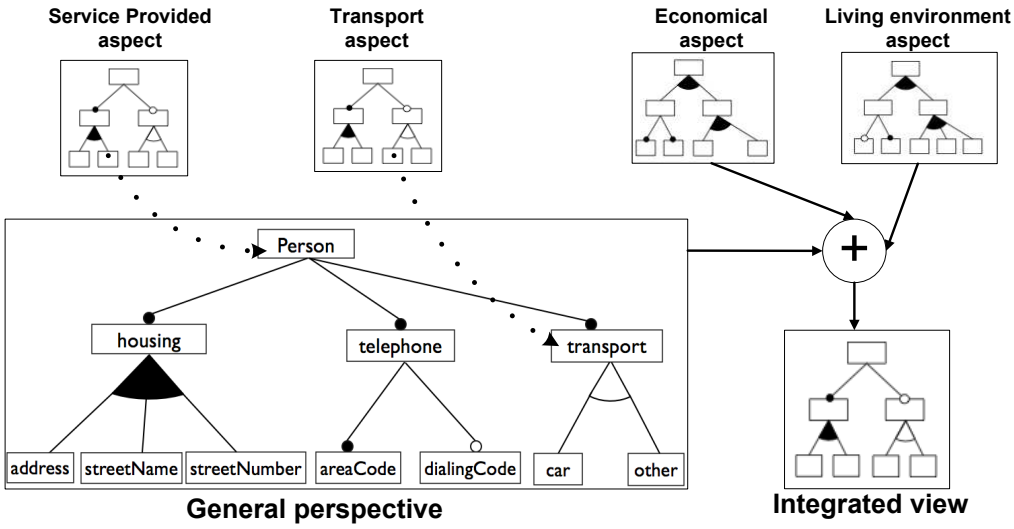


Fig. 2. Integrating several feature models

Additionally, the concept of person is enriched using two other aspects. The first aspect describes features that provide information about the living environment of a person while the second aspect describes features that defined its economic characteristics. These aspects may be considered as different viewpoints that represent the concept of person from the perspective of stakeholder's interests. Fig. 2 shows the four aspects to be composed with the base FM depicted in Fig. 1. The *Service Provided* and *Transport* aspects are orthogonal to the concept of person whereas the *Economical* and *Living Environment* aspects are additional facets of the concept of person.

2.3 Requirements

The example presented above highlights the need for compositional operators that can *i)* add information (e.g. subset features of a FM) to an existing feature, *ii)* refine some features with more detailed information, and *iii)* merge the contents of several features. The operators should work at the feature level to enable a modeler to compose only part of a FM with another FM. This should also enable reuse of part of an input FM when creating a larger composed FM.

Additionally one may need to reuse more than one part of a FM or the same part several times. One should also be able to preselect some of the features of one aspect before the composition is performed. The running example shown in Fig. 2 illustrates a sequence of introduction and merging of features.

These requirements mean that composing two models can correspond to a wide range of situations, from the single use of one operator on the root of two models to be merged, to multiple uses of one or several operators on various features of these aspects. In addition, taking into account the expressiveness of FMs, there are several ways to introduce one feature into another one or to merge them.

Previous work has pointed out that dealing with large, monolithic FMs is problematic, in particular, FM maintenance is a difficult process [11, 12, 10]. As in our running example, an appealing approach is rather to use multiple FMs during the SPL development. A first challenge is to allow different stakeholders or software suppliers, at different stages of the software development, to focus on their expertise and integrate their specific concerns. Another challenge is to manage the evolution of FMs [13, 14, 15]. In order to ensure that software products are well maintained, some relevant properties of the models have to be preserved during time.

The primary issue of all this work is to define some compositional mechanisms. But, to the best of our knowledge, they do not *i)* provide a set of composition operators, *ii)* define the semantics of these operators according to the expressed configurations, *iii)* propose a systematic technique to implement them.

3 Rationale

In order to meet the requirements above, we first identify some relevant semantic properties regarding composition operators. Then we discuss our main design

choices regarding the proposed operators. These operators aim to compose two concerns represented in two FMs. We then distinguish the aspect concern from the base concern. The result of the composition is described according to the set of configurations of the base concern.

3.1 Characterizing the Result of a Compositional Operator

Let f and f' be FMs and $[\![f]\!]$ and $[\![f']\!]$ denote their respective set of configurations. Let *op* be the operator which transforms a base FM f into f' using an aspect FM g. The semantics of the operator *op* is expressed in terms of the relationship between the configuration sets of the input models (f and g) and the resulting model f' (i.e. in terms of the relationship between the configuration sets of f, f' and g).

In [14], the authors distinguish and classify four FM adaptations[4]:

a refactoring : no new configurations are added and no existing configurations are removed : $[\![f]\!] = [\![f']\!]$;
a specialization : some existing configurations are removed and no new configurations are added : $[\![f']\!] \subset [\![f]\!]$;
a generalization : new configurations are added and no existing configurations removed : $[\![f]\!] \subset [\![f']\!]$;
an arbitrary edit : a change that is not a refactoring, a specialization or a generalization.

The classification proposed in [14] covers all the changes a designer can produce on a FM and the formalization provided in [14] is a sound basis for reasoning about these changes. We rely on these four categories of FM adaptations in order to characterize the semantics of the insert and merge operators (see Section 4 and 5).

3.2 Main Design Choices

The composition of an aspect and a base concern may correspond either to the single use of the two proposed compositional operators (*insert* or *merge*), or to any combination of these two operators. Any of the two compositional operators ensure that the result of a successful composition is a well-formed FM (see Section 2).

Scope of an operator. An operator specifies **what** feature(s) g of the aspect concern is to be composed with features in the base concern, and **where** (i.e. which feature in the base model f) it is going to be inserted or merged with[5]. All features of the aspect concern not included in the hierarchy starting with g are not involved in the composition process and are not included in its result.

[4] The author use the term "edits" because the focus seems to be on local edits on FM. An example of edit given in the paper is "moving a feature from one branch to another".

[5] To choose the root feature is equivalent to consider the whole FM.

An aspect concern is either strongly or loosely related to the base concern. It can participate to the description of the same concept but can consider another facet of the information (another viewpoint), or its purpose is orthogonal to the concept described in the base concern. For example, the concern dealing with the economical information of a person corresponds to the first case whereas the kind of transport that may be offered in general (i.e. not only to a person) corresponds to the second case.

Let us now address **how** to compose FMs g with f and let us emphasize why both *insert* and *merge* are needed. The *insert* operator makes it possible to specify any applicable FM operators (i.e. And-, Xor-, or Or-groups) to compose g and f. It is more suited to the case of loosely connected aspects. *Merge* determines the FM operator to be used and it corresponds to the composition of two views of the same concept. *Merge* is higher level and we show that it may be implemented thanks to the *insert* operator (see Section 4 and 5).

Renaming. When two features are merged, two typical cases may occur: two features with the same name (*resp.* different names) in both the base and aspect model may not address the same meaning (*resp.* correspond to the same meaning). We provide an operator rename that allows the user to align the two FMs before composition. For the sake of brevity the renaming operator is not detailed in this paper.

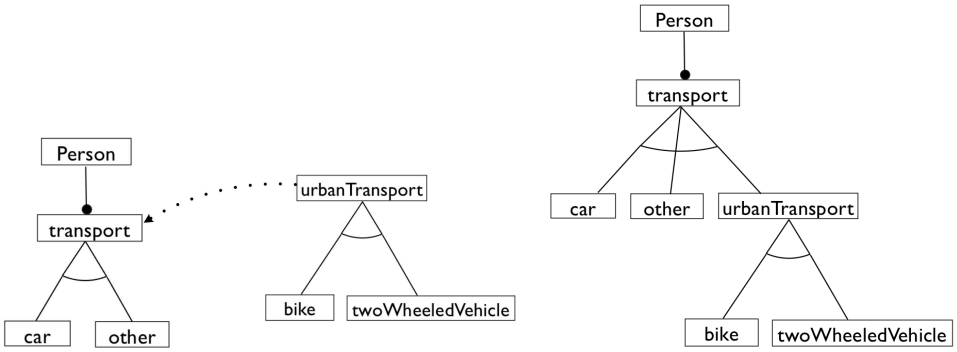
Limits. We might have included more operators as it is proposed in several approaches coming from the Aspect-Oriented Modeling community [20]. Mainly they deal with two other kinds of operators : *replace* and *delete*. We choose not to do so but not for the same reasons. Instead of proposing a new operator for deleting features in the base model[6], we propose that *i)* the semantics of *merge* may rely either on the semantics of the intersection (to only keep the common features) or union (to keep all features) and *ii)* more generally an operator may perform some deletion according to its semantics and to guarantee that the resulting FM is well-formed. We consider *replace* only as a special case of *merge* with some possible renamings before composition.

4 Insert Operator

The insert operator aims at introducing newly created elements into any base element or inserting elements from the aspect model into the base model. For example, a stakeholder can extend the transport feature associated to a Person (left part of Fig. 3(a)) by including the urban transport information, represented in an aspect FM (right part of Fig. 3(a)).

The dotted arrow indicates that the feature urbanTransport is inserted below the feature transport; it does not indicate how the feature tree will be inserted (e.g. which variability information will be associated to the feature tree). The stakeholder needs syntactic mechanisms to define precisely how the insertion is achieved.

[6] according to what had been said at the beginning of the section, there is no need to use such operators for the aspect concern.



(a) Insertion of the Urban transport aspect (b) A possible resulting FM

Fig. 3. Example of insertion of FM

4.1 Syntactic Definition

The insert operator is syntactically defined as follows:

insert (aspectFeature: Feature, joinpointFeature: Feature, operator: Operator)

It takes three arguments: the feature to be inserted (a feature in the aspect model), the targeted feature (a feature in the base model) where the insertion needs to be done, and the operator (e.g. *Xor*-group) specified by the user. The precondition of the insert operator requires that the intersection between the set of features of the base FM and the one of the aspect FM is empty. This condition preserves the well-formed property of the composed FM which states that each feature's name is unique.

The insert's parameters allow the stakeholder to control the insertion addressing the three following issues:

Where will the aspect FM be inserted into the base FM? The $joinpointFeature$ is a feature of the base FM and describes where the $aspectFeature$ should be inserted into the base FM.

What feature(s) of the aspect FM will be inserted into the base FM? The $aspectFeature$ feature is inserted and comes with its child features. If the $aspectFeature$ feature is the root of an aspect FM, the aspect FM is entirely inserted into the base FM. Otherwise only the subtree starting at $aspectFeature$ is inserted.

How will the insertion be done? What are the effects on the properties of the composed model? According to the third argument *operator* (e.g. *Xor*-group) and the group (e.g. *Or*) of $joinpointFeature$ in the base FM , it can change the group of the $aspectFeature$ to be inserted. The remainder of this section defines the semantics and the rules to implement it.

4.2 Semantics

The semantics of the insert operator is represented by the relationship that exists between the new composed model and the base/primary model, so that it refers to the properties preserved or not by the composed model according to its set of configurations. The insert operator should respect one (or more) properties defined in Section 3.1 (generalization, specialization, refactoring or none of these) considering the composed model and the base model. A stakeholder can thus anticipate the changes to the base model while applying the insertion.

Intuitively, if an aspect model is added somewhere in a base model $Base$, the set of configurations of $Base$ should grow. The new version of $Base$ which results from applying the insert operation can produce a generalization: new configurations are added and no existing configurations are removed. But the situation corresponding to an arbitrary edit may also happen depending on the operator that is passed as parameter of *insert*: some new configurations are added while some others are removed. The refinement of a FM can indeed alter the existing configurations such as they become deprecated. According to their definition (see Section 3.1), specialization and refactoring are not possible because they correspond to situations that are not compatible with the meaning of an insertion. This simply follows the rationale behind the insert operator, which is to add details and to populate the base model with additional information. In the remainder of this section, $Base$ FM corresponds to the (sub-)tree of the base FM whose root is $joinpointFeature$ while $Aspect$ FM corresponds to the (sub-)tree of the aspect FM whose root is $aspectFeature$.

More formally the semantics of *insert* is defined as follows:

- The set of configurations of the FM after insertion (*Result*) is at least the set of configurations of $Base$ FM. This can be expressed as follows:

$$[\![Base]\!] \subset [\![Result]\!] \qquad (I_1)$$

- or the set of configurations of *Result* is at least the set of configurations of the cross product of $Base$ and $Aspect$. This can be expressed as follows:

$$[\![Base]\!] \otimes [\![Aspect]\!] \subseteq [\![Result]\!] \qquad (I_2)$$

  where the cross product is defined as (A and B being a set of sets):

$$A \otimes B = \{a \cup b \mid a \in A, b \in B\}$$

The two relations (I_1) and (I_2) define the semantics. The former states that *Result* FM is a generalization of $Base$ FM. The latter ensures that each configuration of $Base$ FM is supplemented by the features of $Aspect$ FM. The insert operator may, in some situations, respect *i)* only one of the relation (i.e. (I_1) or (I_2)) or *ii)* both of them (i.e. (I_1) and (I_2)). A supporting tool can easily exploit this information to produce appropriate warnings when an insertion only preserves one relation and thus assist modelers in reasoning during composition.

As an example, let us consider the set of configurations of the base FM included in the left part of Fig. 3(a), $[\![Base]\!]$,

$$[\![Base]\!] = \{\{Person, transport, car\}, \{Person, transport, other\}\}$$

the set of configurations of the aspect FM included in the right part of Fig. 3(a), $[\![Aspect]\!]$,

$$[\![Aspect]\!] = \{\{urbanTransport, bike\}, \{urbanTransport, twoWheeledVehicle\}\}$$

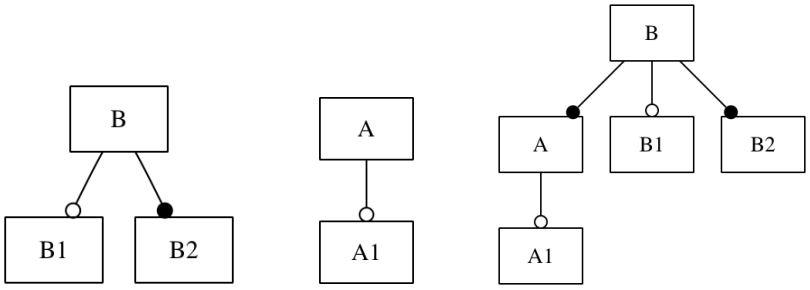
and the set of configurations of the composed FM corresponding to an insertion using the Xor operator is described in Fig. 3(b), $[\![Result]\!]$:

$$[\![Result]\!] = \{\{Person, transport, car\},$$
$$\{Person, transport, other\},$$
$$\{Person, transport, urbanTransport, bike\},$$
$$\{Person, transport, urbanTransport, twoWheeledVehicle\}\}$$

The relationships between $[\![Base]\!]$, $[\![Aspect]\!]$ and $[\![Result]\!]$ respect only the relation (I_1). As a result, the composed FM of Fig. 3(b) is a generalization of the base FM from the left part of Fig. 3(a).

4.3 Rules

In this subsection, we describe rules associated with an insertion. They define *when* and *how* the operator passed as an argument preserves (or not) the previously described properties on the base FM. The rules are given on a base model called *Base*, which has a root feature B and one or several children B1, B2, ..., Bn. The model to be inserted has a root feature A and its child features are A1, A2, ..., An and is called *Aspect*.



(a) Base FM (b) Aspect FM (c) One possible resulting FM

Fig. 4. Rule for insertion of FM

Let us consider the insertion of *Aspect* (Fig. 4(b)) into the *Base* (Fig. 4(a)). If the operator passed to *insert* is an "*And* with the mandatory status", the feature A is inserted as a child feature of B with the mandatory status (Fig. 4(c)). For this example, the sets of configurations of *Base*, *Aspect*, and *Result* are:

$[\![Base]\!] = \{\{B, B1, B2\}, \{B, B2\}\}$
$[\![Aspect]\!] = \{\{A, A1\}, \{A\}\}$
$[\![Result]\!] = \{\{B, A, B1, B2, A1\}, \{B, A, B2, A1\}, \{B, A, B1, B2\}, \{B, A, B2\}\}$

Consequently, the relation (I_1) does not hold. For instance, $\{B, B1, B2\}$ is not a member of $[\![Result]\!]$. Nevertheless, the relation (I_2) is satisfied and the resulting FM is an arbitrary edit to the Base FM. On the contrary, if the stakeholder wants to preserve the (I_1) property, the feature A should be inserted as a child feature of B with the optional status.

Overview of the table of rules. The result of an insertion of a given feature only depends on *i)* the operator passed as argument of *insert* and *ii)* the operator associated to the feature where the insertion is made. All combinations are given in Table 1. We distinguish the cases where no FM operator is associated to a feature of the base FM (it is a leaf) and those where there is either *And*, *Or* or *Xor* operators. *Insert* may accept the following operators : *And* with mandatory (resp. optional) sub-features, *Or* and *Xor*. The table summarizes the properties that are verified by *Result* FM for each combination. When "=" is set, this means that the set of configurations of *Result* FM is strictly equal to $[\![Base]\!] \otimes [\![Aspect]\!]$.

Note that the insertion of one single feature with an *Or* or *Xor* operator into a leaf feature is forbidden, as it would generate badly-formed FMs. Nevertheless, this is possible when insertion deals with a set of features of the aspect model (*i.e.* parameter *aspectFeature* is a set and not a single feature).

Base / Operator	And-Mandatory	And-Optional	Xor	Or
Leaf	$= I_2$	I_1 and I_2	I_1	I_1 and I_2
And	$= I_2$	I_2 and I_1	I_1	I_1 and I_2
Xor	$= I_2$	I_1 and I_2	I_1	I_1 and I_2
Or	$= I_2$	I_1 and I_2	I_1	I_1 and I_2

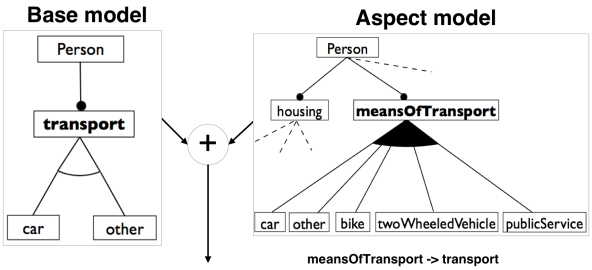
Table 1. Insertion rules

5 Merge Operator

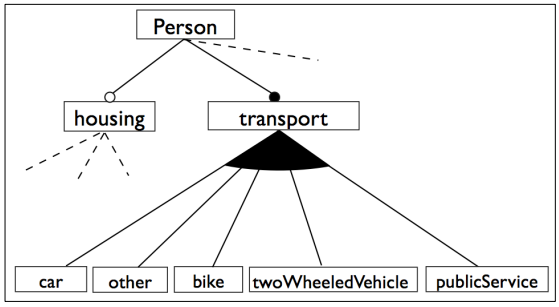
When two FMs share several features and are different views of an aspect of a system, it is necessary to merge the overlapping parts of the two FMs to obtain a single model that presents an integrated view of the system.

Let us consider the example of a base FM (left part of Fig. 5(a)). The root feature is the Person feature which has a child feature transport with two alternatives features car and other. The aspect FM (right part of Fig. 5(a)) describes the concept of Person from another perspective. In that case, a person has also the feature meansOfTransport but the set of alternatives is structured in an *Or*-group, addressing also additional features such as bike, publicService and twoWheeledVehicle. The merge operator can then be used to unify the two viewpoints from the FMs. A mapping can be specified by the stakeholder (*e.g.* to relate the

feature transport of the base FM and the feature meansOfTransport of the aspect FM). More important, the merged FM should verify some properties such as the preservation of configurations. This requires to solve some of the variability issues in each FM. For example, in Fig. 5(a), features car and other cannot be concurrently selected in the *Base* FM whereas the selection of both of them is allowed by the *Aspect* FM.



(a) Base and Aspect FMs to be merged



(b) Merged FM

Fig. 5. Merging of two FMs

5.1 Syntactic Definition

The merge operator is syntactically defined as follows:

merge (aspectFeature: Feature, baseFeature: Feature, mode: Mode)

It takes three arguments: the feature to be merged (a feature of the aspect model), the feature in the base model where the merge is done, and the mode specified by the user. This mode indicates how the merge has to be done in terms of *union* or *intersection* of configurations (see below).

Like for the insert operator, the merge's parameters allow the stakeholder to answer the three same questions:

Where are the features of the aspect FM and the base FM such as the two FMs match? To merge FMs we thus need to first identify match points (similar to joinpoints in aspect terminology). The stakeholder can thus specify the

feature $aspectFeature$ of the aspect FM and the feature $baseFeature$ of the base FM. They are not necessary the root of the FMs.

What are the features of the aspect FM and base FM that will appear in the merged model? Two FMs are merged by applying the operator recursively to their subtrees, starting from the match points ($aspectFeature$ and $baseFeature$). If two features have been merged, the whole process proceeds with their children features. If not, they are inserted as separate child features. The variability information associated to features in the merged model should also be set.

How features are merged by the operator? It uses a name-based matching: two features match if and only if they have the same name. If so, they are merged to form a new feature. Features with different names can be bound to each other thanks to an explicit renaming (see Section 3). Finally, a set of rules resolves possible variability mismatches between features of the two FMs according to the mode (i.e. the third argument of the merge operator).

5.2 Semantics

Like for the operator *insert*, the semantics of *merge* is defined according to the relationship which exists between the FM resulting from the merging and the two input FMs. It is based on the *union* or the *intersection* of the two configuration sets.

Union: When transport is merged with meansOfTransport (see Fig. 5(a)), original information from the base model must be preserved while adding information from the aspect model. The set of configurations of the base and aspect FMs should then be preserved in the merged FM.

> 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.

More formally, the result of a merge in the union mode has the following properties:

- The set of configurations of the FM after merging (*Result*) is at least the set of configurations of Base FM (i.e. *Result* FM is a generalization or a refactoring of *Base* FM). This can be expressed as follows:

$$[\![Base]\!] \subseteq [\![Result]\!] \qquad (M_1)$$

- The set of configurations of *Result* is at least the set of configurations of the Aspect FM (i.e. *Result* FM is a generalization or a refactoring of *Aspect* FM). This can be expressed as follows:

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

Note that if the relations (M_1) and (M_2) are met, the following relationship holds:

$$[\![Base]\!] \cup [\![Aspect]\!] \subseteq [\![Result]\!]$$

This means, the merged FM may allow some configurations that are *not* included in the set of configurations of the base or in the one of the aspect FMs. In order to restrict these configurations, we propose to reinforce the constraints on the merged FM with an additional property (see (M_3)). It states that the set of configurations of *Result* is at least the set of configurations of the cross product of *Base* and *Aspect*. This can be expressed as follows:

$$[\![Base]\!] \otimes [\![Aspect]\!] \subseteq [\![Result]\!] \qquad (M_3)$$

(M_3) can hold concurrently with (M_1) and (M_2), individually or not at all.

Intersection When transport is merged with meansOfTransport (see Fig. 5(a)), only common information of the base model and the aspect model is retained:

> The intersection of two FMs, Base and Aspect, is a new FM where each configuration that is valid *both* in Base *and* Aspect, is also valid.

In the intersection mode, the relationship between the merged FM *Result*, the base FM *Base* and the aspect FM *Aspect* can be expressed as follows:

$$[\![Base]\!] \bigcap [\![Aspect]\!] = [\![Result]\!] \qquad (M_4)$$

Besides, if the following condition holds:

$$[\![Base]\!] \bigcap [\![Aspect]\!] = \emptyset \qquad (M_5)$$

the FM *Result* then defines no configuration at all and can be considered as an inconsistent or an unsatisfiable FM [8].

5.3 Merging Rules

We now describe rules for merging FMs. These rules aim at resolving variabilities in each FM such as the expected properties are met. For example, in Fig. 5(a), features car and other do not exhibit the same variability as they belong to a Xor-group in the base FM whereas they belong to an Or-group in the aspect FM. Not surprisingly, the sets of configurations of the base FM and the aspect FM are not the same, and some configurations are valid in the base FM but not valid in the aspect FM. For example, $\{Person, meansOfTransport, car, housing\}$ is only valid in the aspect FM (since the feature housing is included in all its configurations). Yet, the merged FM should be able to express the set of configurations of both FMs.

To tackle this issue, we propose *i)* to make an explicit difference between common and non common features of the two FMs and *ii)* to (re-)use the insert

Base / Aspect	And-Mandatory	And-Optional	Xor	Or
And-Mandatory	And-Mandatory	And-Optional	Or	Or
And-Optional	And-Optional	And-Optional	And-Optional	And-Optional
Xor	Or	And-Optional	Xor	Or
Or	Or	And-Optional	Or	Or

Table 2. Merge in union mode - relations (M_1) and (M_2) are satisfied.

operator at each step of the merge. As the common features of the two FMs can belong to a different group, a new variability operator has to be chosen in accordance with the intended semantics properties (i.e. merge in the union or intersection mode). We thus propose to organize rules to compute the variability operator into predominance tables. Tables 2 and 3 make the assumption that the same set of features are shared by the base and aspect FMs.

Base / Aspect	And-Mandatory	And-Optional	Xor	Or
And-Mandatory	And-Mandatory	And-Mandatory	And-Mandatory	And-Mandatory
And-Optional	And-Mandatory	And-Optional	Xor	Or
Xor	And-Mandatory	Xor	Xor	Xor
Or	And-mandatory	Or	Xor	Or

Table 3. Merge in intersection mode - relation (M_4) is satisfied.

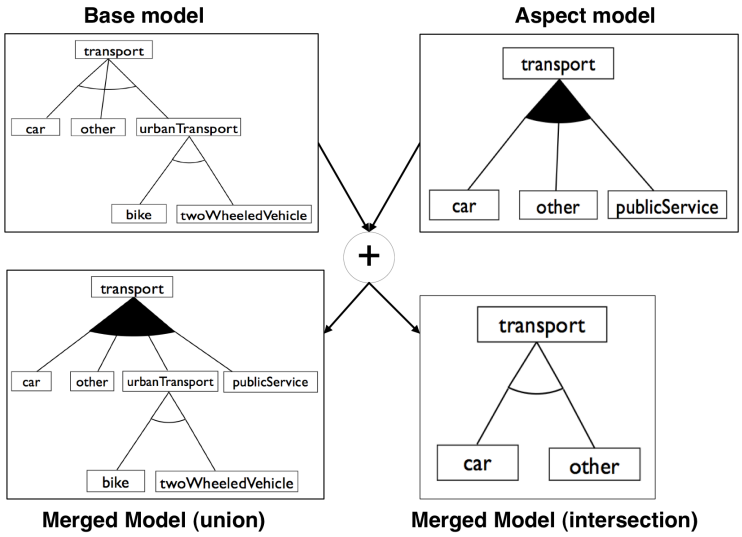


Fig. 6. Merging example

In Fig. 6, features car and other are child features of transport. They belong either to a Xor-group in *Base* FM or to an Or-group in *Aspect* FM. In this case, the predominant operator is an Or-group, that is, the features car and other can

be both selected at the same time (i.e. (M_1) is respected), or car and other can be selected alone (i.e. (M_2) is respected). As a result, the relations (M_1) and (M_2) truly hold for the merged FM depicted in the left bottom part of Fig. 6. Moreover, the relation (M_3) holds too.

Merging in the intersection mode the features car and other of the aspect FM (which belong to an Or-group) with the features car and other of the base FM (which belong to a Xor-group) gives the predominant operator Xor (see right bottom part of Fig. 6). The relation (M_4) truly holds.

Algorithm 1 Merging algorithm

```
merge (aspectFeature: Feature, baseFeature: Feature, mode: Mode)
begin
  if ¬matching(aspectFeature, baseFeature) then "error" fi
  new := newFM (newFeature (baseFeature.getName() ))
  predominanceOp := computeOperator (baseFeature, aspectFeature, mode)
  base := extractChild (baseFeature)
  aspect := extractChild (aspectFeature)
  foreach N ∈ (base ⋂ aspect) do
    res := merge (aspectFeature :: N, baseFeature :: N, mode) /* recursively */
    stackFeatures.push (res) /* pushes the merged feature */
  od
  /* insert the set of features of the stack */
  insert_multi (stackFeatures, new, predominanceOp)
  /* following loops are not executed in the intersection mode */
  foreach N ∈ (base ⧹ (base ⋂ aspect)) do
    insert (N, new.getRoot(), predominanceOp)
  od
  foreach N ∈ (aspect ⧹ (base ⋂ aspect)) do
    insert (N, new.getRoot(), predominanceOp)
  od
  return new
end
```

We define an algorithm for the merge that implements the principles above (see Algorithm 1). As an illustration, let us consider the merge of the Base Model and the Aspect Model depicted in top of Fig. 6. The merge operator is used with the first parameter "transport feature" of the base FM, the second parameter "transport feature" of the aspect FM and the third parameter being the union mode.

Algorithm for the merge First, a new FM is created with one single feature called "transport", which becomes its root, and acts as a temporary FM where the features of the base and aspect FMs will be incrementally inserted. The predominant operator is computed using the predominance table corresponding to the mode. In the example, we obtain an *Or*-group with the union table (see bottom left part of Fig. 6). The common features of the two FMs (i.e. car and other) are merged recursively. Then, they are inserted all together with the predominant operator. At this stage, the connection between the transport root

feature of the temporary FM and its group of children car and other is an Or-group. The next step is to insert the non common features urbanTransport and publicService with the Or-operator into the root feature of the temporary FM, transport. The insertion of a feature with an Or-operator into a feature which is connected to its group of children by an Or-group respects (I_1) and (I_2). As a result, urbanTransport and publicService also belong to an Or-group.

In the intersection mode, the algorithm is executed when the condition (M_5) does not hold. Only the set of common features are considered. In the example, only the features car and other are merged. The result is depicted in bottom right part of Fig. 6. The predominant operator is the Xor-group.

6 Related work

Several previous works consider some forms of composition for FMs. Alves et al. motivate the need to manage the evolution of FMs (or more generally of an SPL) and extend the notion of refactoring to FMs [13]. The authors provide a catalog of sound FM refactorings, which has been verified in Alloy by automatically checking properties of resulting FMs [21]. Although their work is focused on refactoring single FMs, they also suggest to use these rules to merge FMs. Our proposal goes further in this direction by providing mechanisms to implement the merge and by clarifying the semantics (as in [14], our terminology is to consider the unidirectional refactoring as a generalization and a bidirectional refactoring as a refactoring). Segura et al. provide a catalogue of visual rules to describe how to merge FMs [15]. The authors emphasize the need to provide a formal semantics to their approach. To the best of our knowledge, their rules implement the merge in the union mode while the the merge in the intersection is not taken into account. Schobbens et al. identify three operations to merge FMs – intersection, union (a.k.a. disjunction) or reduced product of two FMs [19] but do not provide mechanisms to implement the merging. Czarnecki et al. propose to construct FM from propositional formulas and suggest to use their algorithm to merge FMs, but without further detail [22]. Computing the intersection or union at the propositional logic level is not without problems. It is necessary to generate a FM from the new propositional formula and a major issue is then to take additional structuring information into account. In [23], a feature is represented by a FST (Feature Structure Tree), roughly a stripped-down abstract syntax tree. The authors propose to use superimposition to compose features. A FM is a "hierarchy of features with variability" [18] and can be seen as a FST *plus* variability. As a result, the superimposition mechanism has to be adapted to resolve variabilities mismatch.

In SPL engineering, reusable software assets must be composed to derive specific products according to a particular set of features. An approach is to use FMs to specify the variability and then to relate FMs to architectural or design models (e.g. UML models) [6, 24, 7, 5]. A configuration of the FM can correspond to the removal or the activation of some elements of a model [5, 6]. Another option is to associate each feature to some model artefacts which are

then inserted in a primary design model [7] or composed together [25,6,24]. Our work focuses strictly on the composition of the variability models, i.e. FMs. Our proposal is not incompatible with the approaches described as the composed FM can be related to other models and thus be used during the derivation process.

Aspect-Oriented Modeling (AOM) allows developers to isolate and address separately several aspects of a system by providing techniques to achieve separation and composition of concerns [20]. Existing AOM approaches notably focused on the composition of UML models such as UML class diagrams (e.g. [26]) or UML state and sequence diagrams (e.g. [27]). To the best of our knowledge, no existing approach proposes to compose FMs.

7 Conclusion and Future Work

In this paper, we proposed two main operators to compose feature models (FMs). Each operator is described by stating where it is applied, what features will be composed and how the composition is made. Each composition is defined by rules that formally describe the structure of the resulting FM. Depending on the composed and the targeted features, some properties regarding the expressed set of configurations are made explicit for each operator. A first *insert* operator enables developers to insert features from a crosscutting FM into a base FM. Each insertion can then be characterized by its ability to preserve or not the set of configurations expressed by the base FM. Building on this operator, the proposed *merge* operator makes possible to put together features from two separated FMs, when none of the two clearly crosscuts the other. The result is also characterized through the set of expressed configurations, and is parameterized to enable developers to choose between union or intersection of the configurations.

The two operators cover different use cases but always ensure the well-formedness of the resulting FM. When using the provided operators, developers can choose to make insertion or merge while preserving the expression of the original set of configurations. This enables them to compose FMs at a large scale. On the contrary, when the need to make more important changes appears, developers can then use all presented forms of insertion and merge, while being aware of whether the original semantics of the base FM is preserved or not.

Future work aims at tackling current restrictions and at getting validation of the scalability and usability of the proposed operators. These operators are currently under validation with the construction and usage of a large SPL which is dedicated to medical imaging services on the grid. The services are part of a service-oriented architecture in which data-intensive workflows are built to conduct numerous computations on very large set of images [28, 29]. This SPL is decomposed into several FMs, which are then to be composed using the proposed operators. Moreover, some of the designed FM are planned to be reused in another SPL that deals with video surveillance systems [30]. Some features related to QoS and imaging are likely to be common. The two case studies and SPLs are intended to be complementary and yet different to determine in what sense the merging operators can actually help to scale feature modeling (from

the users' perspective). They can also help to determine whether an arbitrarily decomposed FM can be relevant to all stakeholders or not. Another interest is to quantify the amount of information needed to apply merging operators in order to assess their easiness of use. To achieve these goals, we will raise the limitation on the hierarchy regularity of the composed FMs. Currently the considered FM cannot include any constraints between features, e.g. selecting a feature constrains that another one must be or not be selected. Taking into account such constraints will oblige us to tackle issues on how to reuse consistency checking in a modular way. But as a result, this should also solve some of the scalability issues that FM checking techniques currently face [8, 9].

References

1. Clements, P., Northrop, L.M.: Software Product Lines : Practices and Patterns. Addison-Wesley Professional (August 2001)
2. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag (2005)
3. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute (November 1990)
4. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley Professional (June 2000)
5. Czarnecki, K., Antkiewicz, M.: Mapping features to models: A template approach based on superimposed variants. In: Generative Programming and Component Engineering. Volume 3676/2005 of LNCS. (2005) 422–437
6. Sanchez, P., Loughran, N., Fuentes, L., Garcia, A.: Engineering languages for specifying Product-Derivation processes in software product lines. In: Software Language Engineering (SLE). (2008) 188–207
7. Voelter, M., Groher, I.: Product line implementation using aspect-oriented and model-driven software development. In: SPLC '07: Proceedings of the 11th International Software Product Line Conference, IEEE (2007) 233–242
8. Batory, D., Benavides, D., Ruiz-Cortés, A.: Automated analysis of feature models: Challenges ahead. Communications of the ACM **December** (2006)
9. Mendonca, M., Wasowski, A., Czarnecki, K., Cowan, D.: Efficient compilation techniques for large scale feature models. In: GPCE '08: Proceedings of the 7th international conference on Generative programming and component engineering, ACM (2008) 13–22
10. 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
11. 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
12. Hartmann, H., Trew, T.: Using feature diagrams with context variability to model multiple product lines for software supply chains. In: SPLC '08: Proceedings of the 2008 12th International Software Product Line Conference, IEEE (2008) 12–21
13. Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., Lucena, C.: Refactoring product lines. In: GPCE '06: Proceedings of the 5th international conference on Generative programming and component engineering, ACM (2006) 201–210
14. Thüm, T., Batory, D., Kästner, C.: Reasoning about edits to feature models. In: Proceedings of the 31th International Conference on Software Engineering (ICSE'09), IEEE Computer Society (May 2009)

15. Segura, S., Benavides, D., Ruiz-Cortés, A., Trinidad, P.: Automated merging of feature models using graph transformations. Post-proceedings of the Second Summer School on Generative and Transformational Techniques in Software Engineering **5235** (2008) 489–505
16. Lahire, P., Morin, B., Vanwormhoudt, G., Gaignard, A., Barais, O., Jézéquel, J.M.: Introducing Variability into Aspect-Oriented Modeling Approaches. In: ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MODELS). LNCS, Nashville, USA (October 2007) 498–513
17. Classen, A., Heymans, P., Schobbens, P. In: What's in a Feature : A Requirements Engineering Perspective. Fundamental Approaches to Software Engineering (FASE) (2008) 16–30
18. Czarnecki, K., Kim, C.H.P., Kalleberg, K.T.: Feature models are views on ontologies. In: SPLC '06: Proceedings of the 10th International on Software Product Line Conference, IEEE Computer Society (2006) 41–51
19. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Netw. **51**(2) (2007) 456–479
20. Aspect-Oriented Modeling Workshop Series: `http://www.aspect-modeling.org/`
21. Gheyi, R., Massoni, T., Borba, P.: A theory for feature models in alloy. In: Proceedings of First Alloy Workshop. (2006) 71–80
22. Czarnecki, K., Wasowski, A.: Feature diagrams and logics: There and back again. In: SPLC'07: Proceedings of the 11th International Software Product Line Conference. (2007) 23–34
23. Apel, S., Lengauer, C., Möller, B., Kästner, C.: An algebra for features and feature composition. In: AMAST'08: Proceedings of the 12th international conference on Algebraic Methodology and Software Technology, Springer-Verlag (2008) 36–50
24. Perrouin, G., Klein, J., Guelfi, N., Jézéquel, J.M.: Reconciling automation and flexibility in product derivation. In: SPLC '08: Proceedings of the 2008 12th International Software Product Line Conference, IEEE (2008) 339–348
25. Jayaraman, P.K., Whittle, J., Elkhodary, A.M., Gomaa, H.: Model composition in product lines and feature interaction detection using critical pair analysis. In Engels, G., Opdyke, B., Schmidt, D.C., Weil, F., eds.: MoDELS. Volume 4735 of Lecture Notes in Computer Science., Springer (2007) 151–165
26. 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. Transactions on Aspect-Oriented Software Development **3880** (2006) 75–105
27. Kienzle, J., Al Abed, W., Jacques, K.: Aspect-oriented multi-view modeling. In: AOSD '09: Proceedings of the 8th ACM international conference on Aspect-oriented software development, New York, NY, USA, ACM (2009) 87–98
28. Acher, M., Collet, P., Lahire, P.: Issues in Managing Variability of Medical Imaging Grid Services. In Olabarriaga, S., Lingrand, D., Montagnat, J., eds.: MICCAI-Grid Workshop (MICCAI-Grid), New York, NY, USA (September 2008)
29. 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 - Putting Both Together (SOAPL 2008), associated workshop issue of SPLC'08, IEEE (September 2008)
30. Acher, M., Lahire, P., Moisan, S., Rigault, J.P.: Tackling High Variability in Video Surveillance Systems through a Model Transformation Approach. In: MiSE '09: Proceedings of the International Workshop on Modeling in Software Engineering at ICSE'09, Vancouver, Canada, IEEE Computer Society (May 2009)