Slicing Feature Models

Mathieu Acher, Philippe Collet, Philippe Lahire
I3S – CNRS UMR 6070
Université Nice Sophia Antipolis, France
{acher,collet,lahire}@i3s.unice.fr

Robert B. France
Computer Science Department
Colorado State University, USA
france@cs.colostate.edu

***Abstract*—Feature models (FMs) are a popular formalism for describing the commonality and variability of software product lines (SPLs) in terms of features. As SPL development increasingly involves numerous large FMs, scalable modular techniques are required to manage their complexity. In this paper, we present a novel *slicing* technique that produces a projection of an FM, including constraints. The slicing allows SPL practitioners to find semantically meaningful decompositions of FMs and has been integrated into the FAMILIAR language.**

***Keywords*-Feature Models; Software Product Lines; Slicing;**

I. Introduction

The goal of software product line (SPL) engineering is to produce a family of related program variants for a domain [1]. SPL development starts with an analysis of the domain to identify commonalities and differences between the members of the family. A common way is to describe variabilities of an SPL in terms of features which are domain abstractions relevant to stakeholders [2]. A *Feature Model* (FM) is used to compactly represent all features in an SPL and define their valid combinations [3], [4].

FMs are becoming increasingly complex. There are a number of factors that contribute to their growing complexity. A contributing factor is that FMs are being used not only to describe variability in software designs, but also variability in wider system contexts [5], [6], [7]. For example, features may refer to high-level requirements as well as to properties of the software platform. Another contributing factor is the use of multiple FMs to describe SPLs. It has been observed that maintaining a single large FM for the entire system may not be feasible [8], [5], [1], [9]. Following a model-based approach, several FMs are usually designed to describe software features at various levels of abstraction and to manage variability of artifacts that are produced in different development phases [1], [2]. In addition, organizations are increasingly faced with the challenge of managing variability in product parts provided by external suppliers [6], [10]. The need to support multiple SPLs (also called product populations) makes developing SPLs challenging [1]. Product populations with FMs consisting of hundreds to thousands of features have been observed [11], [12], [13]. Finally, automated extraction of FMs from large implemented software systems [14], can produce FMs with thousands of features.

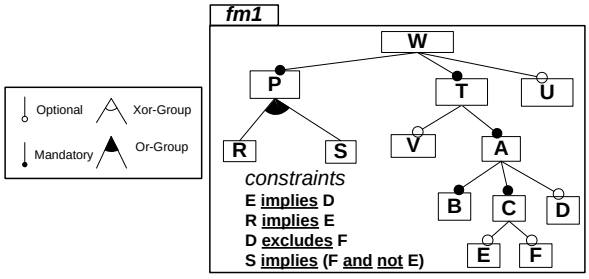
Managing the complexity of building FMs with a large number of features that are related in a variety of ways is a two-fold challenge. On the one hand FM decomposition techniques must be provided to better manage this complexity. On the other hand, automated rigorous reasoning techniques must be provided to relieve error-prone and tedious tasks associated with manually analyzing FMs [13]. Naturally, with FMs being handled by several stakeholders, or even different organizations, techniques that provide good support for the principles of *separation of concerns* seem particularly well-suited solution to the problem of effective decomposition of FMs. Two ways to support separation of concerns is to provide mechanisms for extracting views from large FMs [15], and to provide mechanisms for synthetizing large FMs from smaller composable FMs. In previous work [17], we designed a set of *composition* operators for FMs (insertion, merging, aggregation) and defined semantic properties that must be preserved during composition. In this paper we present a novel *slicing* technique that produces a projection of an FM (a slice) with respect to a set of selected features (slicing criterion). It allows SPL developers to automatically obtain semantically meaningful *decompositions* of FMs.

II. Background: Feature Models

FMs were first introduced in the FODA method [8], which also provided a graphical representation through *Feature Diagrams*. The two essential components of an FM are hierarchy and variability. A FM *hierarchy* (typically a tree) structures application features into multiple levels of increasing detail. The *variability* aspect of an FM restricts the legal combination of features and is expressed through a number of mechanisms. The features may be *optional* or *mandatory* or may form *Xor* or *Or*-groups. In addition, implies or excludes constraints that cut across the hierarchy can be specified to express more complex dependencies between features. We consider that an FM is composed of a feature diagram (see Definition 1) plus a set of constraints expressed in propositional logic (see Definition 2). Figure 1a shows an example of an FM.

Definition 1 (Feature Diagram): A feature diagram $FD = \langle G, r, E_{MAND}, \mathcal{F}_{XOR}, \mathcal{F}_{OR}, Impl, Excl \rangle$ is defined as follows:

- $G = (\mathcal{F}, E)$ is a rooted tree where $\mathcal{F}$ is a finite set of features and $E \subseteq \mathcal{F} \times \mathcal{F}$ is a finite set of edges (edges represent top-down hierarchical decomposition of features, i.e., parent-child relations between them) ;
- $r \in \mathcal{F}$ is the root feature ;
- $E_{MAND} \subseteq E$ is a set of edges that defines mandatory features with their parents ;
- $\mathcal{F}_{XOR} \subseteq \mathcal{P}(\mathcal{F}) \times \mathcal{F}$ and $\mathcal{F}_{OR} \subseteq \mathcal{P}(\mathcal{F}) \times \mathcal{F}$ define feature groups and are sets of pairs of child features together with its common parent feature. The child features are either exclusive (Xor-groups) or inclusive (Or-groups) ;



(a) FM: FODA-like representation

$$[[fm1]] = \{$$
$$\{A, B, C, D, E, P, R, T, U, W\},$$
$$\{A, B, C, F, P, S, T, U, W\},$$
$$\{A, B, C, D, E, P, R, T, W\},$$
$$\{A, B, C, F, P, S, T, V, W\},$$
$$\{A, B, C, F, P, S, T, U, V, W\},$$
$$\{A, B, C, F, P, S, T, W\},$$
$$\{A, B, C, D, E, P, R, T, V, W\},$$
$$\{A, B, C, D, E, P, R, T, U, V, W\}$$
$$\}$$

(b) set of configurations

$$\begin{aligned}
&\phi_{fm_1} = W \text{ // root} \\
&\wedge\ W \Leftrightarrow P \wedge\ W \Leftrightarrow T \\
&\wedge\ U \Rightarrow W \text{ // optional} \\
&\wedge\ V \Rightarrow T \\
&\wedge\ A \Leftrightarrow T \text{ // mandatory} \\
&\wedge\ A \Leftrightarrow B \wedge\ A \Leftrightarrow C \\
&\wedge\ D \Rightarrow A \\
&\wedge\ E \Rightarrow C \wedge\ F \Rightarrow C \\
&\text{// Or-group} \\
&\wedge\ P \Rightarrow R \vee S \\
&\wedge\ R \Rightarrow P \wedge\ S \Rightarrow P \\
&\text{// constraints} \\
&\wedge\ E \Rightarrow D \wedge\ R \Rightarrow E \\
&\wedge\ D \Rightarrow \neg F \wedge\ S \Rightarrow (F \wedge \neg E)
\end{aligned}$$

(c) corresponding propositional formula

Figure 1. FM, set of configurations and propositional logic encoding

- a set of implies constraints $Impl$ (resp. excludes constraints $Excl$), each implies constraint (resp. excludes constraint) being a propositional formula whose form is $A \Rightarrow B$ (resp. $A \Rightarrow \neg B$) where $A \in \mathcal{F}$ and $B \in \mathcal{F}$.

Features that are neither mandatory features nor involved in a feature group are optional features. Furthermore, a parent feature can have several feature groups but a feature must belong to only one feature group.

Definition 2 (Feature Model): An FM is a tuple $\langle FD, \psi_{cst} \rangle$ where FD is a feature diagram and ψ_{cst} is a set of cross-tree constraints where each constraint is a propositional formula over the set of features $\mathcal{F}$ but neither an implies constraint nor an excludes constraint.

An FM defines a set of valid feature *configurations*. A valid configuration is obtained by selecting features in a manner that respects the following rules: *i)* If a feature is selected, its parent must also be selected; *ii)* If a parent is selected, the following features must also be selected - all the mandatory subfeatures, exactly one subfeature in each of its Xor-groups, and at least one of its subfeatures in each of its Or groups; *iii)* Constraints (implies, excludes or any propositional constraints) relating features in different subtrees must hold. For example, the set of valid configurations characterized by the FM of Figure 1a is enumerated in Figure 1b.

Definition 3 (Configuration Semantics): A configuration of an FM fm is defined as a set of selected features. $[\![fm]\!]$ denotes the set of valid configurations of the FM fm and is thus a set of sets of features.

III. Slicing Operator: Semantics and Algorithm

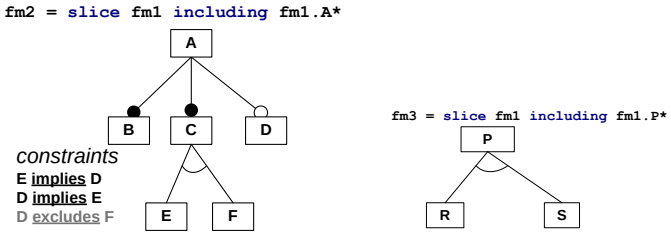
Managing a large number of features, governed by many and often complex rules, is obviously a problem per se for an SPL practitioner. Dividing FMs into localized and separated parts seems a particularly well-suited solution to the problem, because SPL practitioners can focus their attention on one part of a given FM at a time. A first basic mechanism is to "copy" a sub-tree of an FM, including cross-tree constraints involving features of the subtree. This mechanism has two important limitations. First, the mechanism is purely syntactical and ignores cross-tree constraints that involve features not present in the sub-tree. However, these constraints may have an impact on the set of configurations. For example, considering the example of Figure 1a, the basic extraction fails to infer the transitivity of implications (e.g., D implies E) or that features E and F are mutually exclusive. Second, features must belong to the same sub-tree whereas a more generic approach would be to consider any features of an FM wherever they are located in the FM.

To address these limitations, we propose an automated technique, called *slicing*, that produces an FM that contains an arbitrary subset of features. The overall idea behind FM slicing is similar to program slicing [19]. Program slicing techniques usually proceed in two steps: the subset of elements of interest (e.g., a set of variables of interest and a program location), called the slicing *criterion*, is first identified ; then, a *slice* (e.g., a subset of the source code) is computed. In the context of FMs, we define the slicing criterion as a set of features considered to be pertinent by an SPL practitioner while the slice is a new FM.

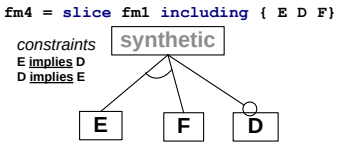
A. Examples of Slicing

In the rest of this section, we use the FM shown in Figure 1a to illustrate the semantics properties of the slice operator.

A first example is given in Figure 2a where the slicing criterion corresponds to the set of features A, B, C, D, E and F. The hierarchy of fm_2 does not alter the structure (i.e., parent-child relationships) of the original FM fm_1. It corresponds to a subtree of the tree of fm_1 whose root is feature A. The valid configurations characterized by fm_2 corresponds to the valid configurations of the original FM fm_1, when looking only at the specific features of the criterion. It can be seen as a *projection* on $[\![fm1]\!]$ (see Figure 1b) when the features not included in the criterion (W, P, ..., U) are discarded. The variability of fm_2 is set to accurately represent $[\![fm_2]\!]$. We can notice that: *(1)* features E and F form an Xor-group in fm_2 whereas they are optional features in fm_1. Indeed, the constraints require that features E and F *are* mutually exclusive in fm_1 even though their presentation in the feature diagram does not make this explicit ; *(2)* the constraint D implies E has been added to fm_2. The reason is that though the constraint is not part of the original FM, it is logically entailed by fm_1 ; *(3)* the constraint D excludes F is not added to fm_2 since the constraint is redundant (i.e., does not alter $[\![fm2]\!]$).



(a) *[[fm2]] = {{A, B, C, D, E},{A, B, C, F}}* (b) *[[fm3]] = {{P, S}, {P, R}}*



(c) *[[fm4]]={{F}, {D, E}}*

Figure 2. Example of slice operations applied on the FM of Figure. 1a.

A second example is shown in Figure 2b. The resulting hierarchy of fm_3 preserves the structure of fm_1 and features R and S form an Xor-group whereas they originally form an Or-group in fm_1. The slicing operation infers that it is really mutual exclusion. $[\![fm_3]\!]$ is equal to the projection onto features P, R and S of $[\![fm_1]\!]$. The third example (see Figure 2c) is less straightforward as the slicing criterion involves features from different locations of the original FM. The slicing operation tries to preserve as much as possible the original feature hierarchy (see below for an explanation on the presence of a synthetic root).

B. Semantics

We define slicing as a unary operation on FM, denoted $\Pi_{\mathcal{F}_{slice}}(fm)$ where $\mathcal{F}_{slice} = \{ft_1, ft_2, ..., ft_n\} \subseteq \mathcal{F}$ is a set of features.

Definition 4 (Slicing Properties): The result of the slicing operation is a new FM, fm_{slice}, such that:

- **configuration semantics** $[\![fm_{slice}]\!] = \{ x \in [\![fm]\!] \mid x \cap \mathcal{F}_{slice} \}$ (called the *projected* set of configurations) ;
- **hierarchy** Intuitively, the hierarchy of fm_{slice} is such that features are connected to their closest ancestor if their parent is not part of the slicing criterion. Formally: $G_{slice} = (\mathcal{F}_{fm_{slice}}, E_{slice})$ with $\mathcal{F}_{fm_{slice}} = ((\mathcal{F}_{slice} \setminus deads(fm)) \cup synthetic_s)$ and $E_{slice} \subseteq E$ such that $E_{slice} = \{e = (v, v') \mid e \in E' \wedge \nexists\, v'' \in E' : ((v, v'') \in E' \wedge (v', v'') \in E')\}$ where $G' = (\mathcal{F}', E')$ is the transitive closure of G_{slice}.

About the synthetic root. We consider that the synthetic root is *not* part of $[\![fm_{slice}]\!]$ and is only here to ensure the well-formedness of the hierarchy. The synthetic root can be removed from G_{slice} if and only if one or more than one of its child feature is a *core* feature (see Definition 5):

- in case the synthetic root has two or more than two child features that are core features, a procedure should choose one – deciding which feature to choose from amongs the core features is left as open – to replace the synthetic root ;
- in case there is exactly one core child feature f_{core}, the root feature of fm_{slice} becomes f_{core}. For example, feature A is the root feature of the sliced FM of Figure 2a and feature P is the root feature of the sliced FM of Figure 2b.

The synthetic root cannot be removed from G_{slice} and is necessary (e.g., for the purpose of visualization) if all its child features are not core features. It is the case in Figure 2c. Indeed, given the expected set of configurations represented by fm_4 (see $[\![fm_4]\!]$), neither feature D, E, nor F can be the root feature.

Definition 5 (Dead and Core features): The set of dead features of an FM fm is noted $deads(fm) = \{f \in \mathcal{F} \mid \forall c \in [\![fm]\!], f \notin c\}$. The set of core features of an FM fm is noted $cores(fm) = \{f \in \mathcal{F} \mid \forall c \in [\![fm]\!], f \in c\}$

C. Algorithm

Our previous experience in the merging of FMs has shown that *syntactical* strategies have severe limitations with respect to accurately representing the expected set of configurations, especially in the presence of cross-tree constraints [17]. The same observation applies to the slicing operation so that reasoning directly at the *semantic* level is required. FMs have been semantically related to propositional logic [4]. The set of configurations represented by an FM can be described by a propositional formula ϕ defined over a set of Boolean variables, where each variable corresponds to a feature (see Figure 1c for the propositional formula corresponding to the FM of Figure 1a). The key ideas of the proposed algorithm are to *i)* compute the propositional formula representing the projected set of configurations and *ii)* to apply propositional logic reasoning techniques to construct an FM (including its hierarchy, variability information and cross-tree constraints) from the propositional formula.

1) Formula Computation: For a slicing $fm_{slice} = \Pi_{ft_1, ft_2, ..., ft_n}(fm)$, the propositional formula corresponding to fm_{slice} can be defined as follows:

$$\phi_{slice} \equiv \exists\, ftx_1, ftx_2, \dots ftx_{m'}\ \phi$$

where $ftx_1, ftx_2, \dots ftx_{m'} \in (\mathcal{F} \setminus \mathcal{F}_{slice}) = \mathcal{F}_{removed}$.

The propositional formula ϕ_{slice} is obtained from ϕ by *existentially quantifying* out variables in $\mathcal{F}_{removed}$. Intuitively, all occurrences of features that are not present in any configuration of fm_{slice} are removed by existential quantification in ϕ (in particular, dead features are removed).

2) From Formula to FM: We reuse and adapt techniques presented in [4], [14]. The authors propose an algorithm to construct a feature diagram from a propositional formula. Propositional logics techniques are developed to detect features logically implied, *Xor-* and *Or-* groups using the method of prime implicants. Furthermore the authors propose a generalized notation, roughly, a directed acyclic graph with additional nodes for feature groups from which different, yet valid, feature hierarchies can be chose. A major difference is that, in our work, we already *know* the resulting hierarchy. We thus exploit this information to streamline the algorithm. It proceeds as follows:

① **Hierarchy Computation.** Let G be the hierarchy of the input FM to be sliced, fm. The synthetic root is added to G

so that $synthetic_s$ is the new root of G. We obtain G_{slice}, the hierarchy of the resulting sliced FM, by incrementally removing all features of $\mathcal{F}_{removed}$ in G. In case the feature is a leaf, the feature and its associated edge are simply removed. In case the feature is not a leaf, the feature and all associated edges are removed while its children are connected to its parent feature by adding new edges.

② **Mandatory and Feature Groups.** At this step, all features, except root, are currently optional (see Definition 1). We compute the implication graph, noted I_{slice}, of the formula ϕ_{slice} over $\mathcal{F}_{slice}$.

I_{slice} is a directed graph $G = (V, E)$ formally defined as:

$$V = \mathcal{F}_{slice} \qquad E = \{(f_i, f_j) \mid \phi_{slice} \wedge f_i \Rightarrow f_j\}$$

We use I_{slice} to identify biimplications and thus set mandatory features together with their parents (i.e., setting E_{MAND}). For feature groups, we reuse the prime implications method proposed in [4] and thus set $\mathcal{F}_{XOR}$ and $\mathcal{F}_{OR}$. It should be noted that a feature may be candidate to several feature groups (which is not allowed in our formalism). We use information of the original FM to favor features that were initially grouped.

③ **Constraints.** The set of implies constraints can be deduced by removing edges of I_{slice} that are already expressed in the feature diagram (e.g., parent-child relations). Similarly, excludes constraints that were not chosen to be represented as an Xor-group are added. When adding constraints, we control that the constraint is not already induced by the FM. The feature diagram *plus* the implies/excludes constraints may still be an over approximation of ϕ_{slice}. The complement corresponds to $\psi_{slice_{cst}}$.

D. Implementation

The slicing technique has been integrated into FAMILIAR [18] a domain specific language for manipulating FMs (see Figure 2 for syntax examples). It can be used with a complete toolset and combined with other operators to support large scale management of FMs [21].

Currently, the handling of logical operations relies on Binary Decision Diagrams (BDDs). Using BDDs, computing the existential quantification of ϕ_{slice} can be performed in at most polynomial time with respect to the size of the BDD involved. The cost of feature diagram construction is polynomial regarding the size of the BDD (the most expensive step being the computation of prime implicants) [4]. We reuse the heuristics developed in [20] (i.e., Pre-CL-MinSpan) to reduce the size of the BDD and that are known to scale for up to 2000 features. Recently, She et al. proposed techniques to reverse engineering very large FMs (i.e., with more than 5000 features) using SAT solvers [14]. For this order of complexity, BDDs do not scale. We leave the SAT-based implementation of slicing as future work.

IV. Conclusion

Techniques to manage the complexity of increasingly larger and complex FMs are needed. In this paper, we presented a novel *slicing* technique that produces a projection of an FM, taking into account cross-tree constraints, with respect to a set of features acting as the slicing criterion. We defined the semantics of this operator in terms of configuration set and hierarchy. As a purely syntactic realization is not satisfactory, we proposed an algorithm to implement slicing. It is based on existing propositional logic techniques that are improved by taking into account the specificities of the operator. The slicing technique has been implemented and integrated into FAMILIAR, a language for manipulating FMs [21].

References

[1] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[2] S. Apel and C. Kästner, "An overview of feature-oriented software development," *Journal of Object Technology (JOT)*, vol. 8, no. 5, pp. 49–84, July/August 2009.

[3] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps, "Generic semantics of feature diagrams," *Comput. Netw.*, vol. 51, no. 2, pp. 456–479, 2007.

[4] K. Czarnecki and A. Wasowski, "Feature diagrams and logics: There and back again," in *SPLC'07*, 2007, pp. 23–34.

[5] A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, and G. Saval, "Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis," in *RE'07*, 2007, pp. 243–253.

[6] H. Hartmann and T. Trew, "Using feature diagrams with context variability to model multiple product lines for software supply chains," in *SPLC'08*, 2008, pp. 12–21.

[7] T. T. Tun, Q. Boucher, A. Classen, A. Hubaux, and P. Heymans, "Relating requirements and feature configurations: A systematic approach," in *SPLC'09*, 2009, pp. 201–210.

[8] K. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "Form: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, no. 1, pp. 143–168, 1998.

[9] D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer, "Structuring the modeling space and supporting evolution in software product line engineering," *Journal of Systems and Software*, vol. 83, no. 7, pp. 1108–1122, 2010.

[10] H. Hartmann, T. Trew, and A. Matsinger, "Supplier independent feature modelling," in *SPLC'09*, 2009, pp. 191–200.

[11] M.-O. Reiser and M. Weber, "Multi-level feature trees: A pragmatic approach to managing highly complex product families," *Requir. Eng.*, vol. 12, no. 2, pp. 57–75, 2007.

[12] M. Mendonca and D. Cowan, "Decision-making coordination and efficient reasoning techniques for feature-based configuration," *Science of Computer Programming*, vol. 75, no. 5, pp. 311 – 332, 2010.

[13] D. Benavides, S. Segura, and A. Ruiz-Cortes, "Automated analysis of feature models 20 years later: a literature review," *Information Systems*, vol. 35, no. 6, 2010.

[14] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Reverse engineering feature models," in *ICSE'11*, 2011.

[15] A. Hubaux, P. Heymans, and P.-Y. Schobbens, "Supporting multiple perspectives in feature-based configuration: Foundations," University of Namur, Tech. Rep. P-CS-TR PPFD-000001, Mar. 2010.

[16] M. Acher, P. Collet, P. Lahire, S. Moisan, and J.-P. Rigault, "Modeling Variability from Requirements to Runtime," in *ICECCS'11*, 2011.

[17] M. Acher, P. Collet, P. Lahire, and R. France, "Comparing Approaches to Implement Feature Model Composition," in *ECMFA'10*, LNCS, vol. 6138, 2010, pp. 3–19.

[18] M. Acher, P. Collet, P. Lahire, and R. France, "A Domain-Specific Language for Managing Feature Models," in *SAC'11*, ACM, 2011.

[19] M. Weiser, "Program slicing," in *ICSE '81*, 1981, pp. 439–449.

[20] M. Mendonca, A. Wasowski, K. Czarnecki, and D. Cowan, "Efficient compilation techniques for large scale feature models," in *GPCE'08*, 2008, pp. 13–22.

[21] M. Acher, P. Collet, P. Lahire, and R. France, "'Decomposing Feature Models: Language, Environment and Applications," in *ASE'11*, tool demonstration paper, 2011.