Assisting Configurations-based Feature Model Composition: Union, Intersection and Approximate Intersection

Jessie Carbonnel, Marianne Huchard, André Miralles, Clémentine Nebut

Assisting Configurations-based Feature Model Composition:
Union, Intersection and Approximate Intersection

Jessie Carbonnel[1], Marianne Huchard[1],
André Miralles[2], and Clémentine Nebut[1]

[1] LIRMM, CNRS and Université de Montpellier,
161 rue Ada, 34095, Montpellier Cedex 5, France
{jessie.carbonnel,marianne.huchard,clementine.nebut}@lirmm.fr
[2] TETIS, IRSTEA, 500 rue Jean-François Breton,
34093, Montpellier Cedex 5, France
{andre.miralles}@teledetection.fr

Abstract. Feature Models (FMs) have been introduced in the domain of software product lines to model and represent product variability. They have become a de facto standard, based on a logical tree structure accompanied by textual cross-tree constraints. Other representations are: (product) configuration sets from concrete software product lines, logical representations, constraint programming, or conceptual structures, coming from the Formal Concept Analysis (FCA) framework. Modeling variability through FMs may consist in extracting them from configuration sets (namely, doing FM synthesis), or designing them in several steps potentially involving several teams with different concerns. FM composition is useful in this design activity as it may assist FM iterative building. In this paper, we describe an approach, based on a configuration set and focusing on two main composition semantics (union, intersection), to assist designers in FM composition. We also introduce an approximate intersection notion. FCA is used to represent, for a product family, all the FMs that have the same configuration set through a canonical form. The approach is able to take into account cross-tree constraints and FMs with different feature sets and tree structure, thus it lets the expert free of choosing a different ontological interpretation. We describe the implementation of our approach and we present a set of concrete examples.

Keywords: Software Product Line, Feature Model, Feature Model Composition, Feature Model Merging, Formal Concept Analysis

1 Introduction

Software Product Line Engineering (SPLE) is a development paradigm which aims to develop a set of related and similar software systems as a single entity rather than developing individually each software system [30]. From a development point of view, the core of this methodology is a generic architecture

where off-the-shelf reusable artifacts can be plugged depending a given set of requirements, and from which can easily be derived a set of software variants. SPLE is composed of two phases. *Domain engineering* consists in analyzing and representing the domain, developing the off-the-shelf artifacts and implementing the generic architecture. *Application engineering* consists in giving the final user the opportunity to select the characteristics she/he wants in her/his software product, and then to derive the corresponding software variant composed with the matching artifacts.

Variability modeling is a task that takes place during domain representation. It consists in modeling what varies in the software variants, and how it varies. It is central to SPLE paradigm, as a substantial part of the method is based on the variability representation, as for instance designing the generic architecture, or guiding the user to select characteristics. The most common approaches model variability in terms of *features*, where a feature is a distinguishable characteristic which is relevant to one or several stakeholders. *Feature models* (FMs) are considered the standard to model variability with these approaches. They are a family of visual/graphical languages which depict a set of features and dependencies between these features. FMs are used, amongst others, to derive selection tools for the end product designer.

Nowadays, practitioners have to cope with product lines which are more and more complex. Managing one, huge feature model representing the whole product line is unrealistic. A solution to ease the application of the SPLE approach in these cases is to divide the product line according to various concerns and to manage a separate and specific feature model for each concern. However, even though it is easier to manage separate FMs, for some design activities it can be useful to merge these FMs, as for commonalities analysis between different concerns. Therefore, defining operations that enable feature model composition is necessary. Feature model composition also has other purposes, in the context of Software Product Line *reengineering*, for feature model reuse and adaptation.

Several approaches for FM composition have been proposed in the past, that are reported in [1]. The main directions for feature model composition in the literature are using operators on the feature model structure, or propositional logic computation. Although these approaches have their advantages, either they tend to confine the designer in a predefined ontological view, or they produce approximate results, or they need a significant work to build a feature model from the result (when the result is a logic formula). Besides, operators on feature model structure hardly take into account the textual cross-tree constraints.

In this paper, we investigate feature model composition in the contexts where the product configuration set is known (or can be obtained) and where the entities to be composed are either several feature models, or a feature model and a product configuration set, or several product configuration sets. The paper extends previous research presented in [15]. The approach uses the framework of Formal Concept Analysis [22] which provides relevant tools for variability representation. This framework ensures the production of sound and complete compositions, taking into account the cross-tree constraints. Our approach ex-

ploits Formal Concept Analysis properties to produce intermediate canonical and graphical representations (Equivalence Class Feature Diagrams, or ECFDs) which give assistance to a designer to manually derive a feature model. The ECFD contains all the possible ontological links and avoids confining the designer in a specific ontological view. Two main composition operations are defined (union and intersection), and, in this paper, we also study the problem of common sub-configuration extraction (approximate intersection), which arises when the intersection is empty, but the feature models have some similarities. Except in extreme cases, the approximate intersection is not empty. We present a prototype tool which computes union, intersection, and approximate intersection, and we conduct an evaluation on real feature models. The results allow us to show concrete situations where the approach is scalable, to draw its scope of applicability and to compare the different operations.

Next Section 2 defines feature models and gives an overview of the approach. Section 3 introduces the main composition operations (union and intersection). Section 4 introduces Formal Concept Analysis and shows how the framework helps to build an intermediate canonical and graphical representation with the aim to assist a designer in feature model composition. The section also proposes an assisting approach for the extraction of common sub-configurations (approximate intersection) which is based on the conceptual structure. The prototype tool and the evaluation are presented in Section 5. Related work and a discussion are developed in Section 6. Section 7 summarizes the approach and provides perspectives for this research.

2 Context and overview

In this section, we define feature models (Section 2.1), then we provide an overview of the compositional approach (Section 2.2).

2.1 Feature Models

The most common SPLE approaches model variability in terms of features, where a feature is a distinguishable characteristic which is relevant to some of the involved stakeholders. Feature Models (FMs) are considered the standard to model variability with these approaches [25]. They are a family of visual languages which depict a set of features and dependencies between these features [5]. In this way, they define the legal combinations of these features, namely the possible software variants of the product line (also called product configurations or simply configurations). In FMs, the features are organized in a hierarchy, where each feature represents a characteristic or a concept at several levels of increasing details, and where each edge represents an ontological relation as "is-a-kind-of", "refines", "is-a-part-of", "implements", etc. Dependencies are expressed on the edges of the tree with graphical decorations, or in textual cross-tree constraints. Figure 1 gives an example of an FM representing an SPL about e-commerce applications, in the most common formalism (FODA [25]). The example states that

`e_commerce` (root feature) requires a `catalog`. This mandatory relation is indicated through an edge ending with a black disk. Also, it shows that `e_commerce` optionally possesses a `basket`, and this is indicated by an edge ending with a white circle. In FMs, the children of a feature may also be grouped into *xor groups* (meaning that if the parent feature belongs to a configuration, exactly one child feature of the group is also present) or into *or groups* (meaning that if the parent feature belongs to a configuration, one or more child features of the group are also present). An *xor group* is indicated by a black line connecting the edges going from the parent to the children of this group. In the example, one can see that an `e_commerce` application proposes a `catalog` presentation as a `grid` or as a `list` (but not both). An *or group* is indicated by a black filled zone connecting the edges going from the parent to the children of this group. We can see in the example that the proposed `payment_method` may be `credit_card`, or `check`, or both. In Figure 1, two cross-tree constraints, are added to the feature tree, to indicate a mutual dependency between `payment_method` and `basket`.

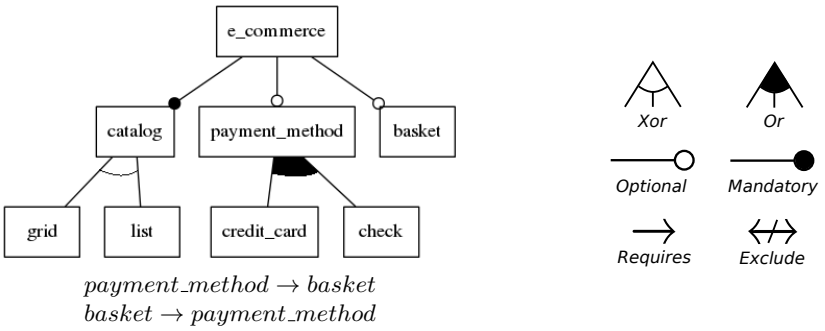


Fig. 1. Left-hand side: An FM describing the variability in a family of e_commerce applications in FODA formalism. Right-hand side: Main feature tree annotations.

An FM owns an *ontological semantics*. Closeness and correspondences between FMs and ontologies are studied in [18]. The ontological semantics is the domain knowledge expressed by the feature tree along with the other feature dependencies (groups, mutex and constraints). For instance, we can read in the e-commerce FM that `catalog` is a mandatory characteristic that refines the concept of `e_commerce`, and that (pay with) `check` implements the concept of `payment_method`. In our work, we also are interested in another semantics of feature models, the *configuration semantics*. It is given by the set of valid configurations, which are the combinations of features (feature sets) which respect all the dependencies given by the FM. The set of configurations of an FM f is denoted by $[\![f]\!]$. Our e-commerce FM here has 8 valid configurations, which correspond to the description of the 8 possible software variants of the product

line. Equation 1 shows ⟦ `e_commerce` ⟧.

$$\begin{array}{rr} Ec, Ca, G & Ec, Ca, L \\ Ec, Ca, G, Pm, Cc, B & Ec, Ca, L, Pm, Cc, B \\ Ec, Ca, G, Pm, Ch, B & Ec, Ca, L, Pm, Ch, B \\ Ec, Ca, G, Pm, Cc, Ch, B & Ec, Ca, L, Pm, Cc, Ch, B \end{array} \quad (1)$$

2.2 Compositional Approach Overview

Figure 2 illustrates the proposed composition operations. The input can be: two feature models (top left), one feature model and one configuration set (bottom left), or two configuration sets (not illustrated). The configuration sets are computed for each input feature model, and then represented in the form of formal contexts. The composition operations (union and intersection) are made on the formal contexts. A conceptual structure (here an AC-poset, namely a structure only containing the concepts introducing the features) is built for the union (resp. intersection) formal context. The ECFD, which is a canonical and graphical structure is then extracted from the AC-poset; it supports the designer to compose a new FM. All these notions are explained in the next sections.

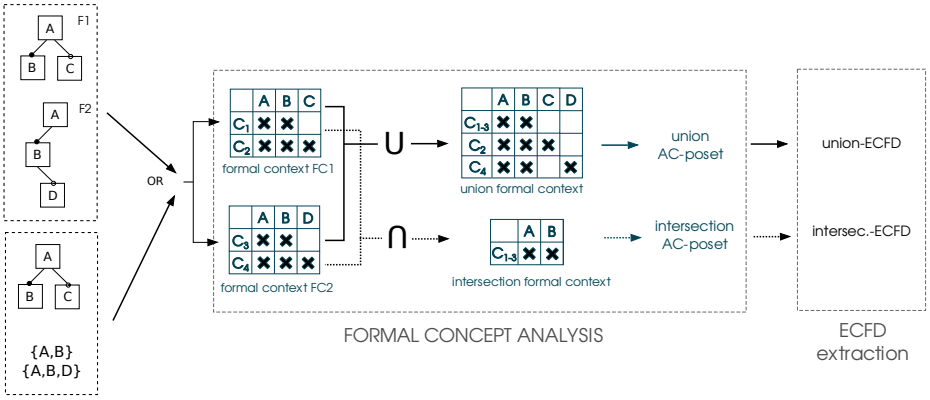


Fig. 2. An overview of the FM composition process. The FMs F_1 and F_2 (top left) have resp. FC1 and FC2 as associated configuration sets / formal contexts.

3 Feature Model Composition

In this section, we define the union and intersection of feature models (Section 3.1), then we discuss the main existing approaches to implement them in order to motivate ours (Section 3.2).

3.1 Intersection/union based composition

Nowadays, practitioners have to cope with product lines which are more and more complex, and managing one, huge feature model representing the whole product line is unrealistic. To ease the application of the SPL approach in these cases, a solution is to divide the product line according to separate concerns and to manage a distinct and more specific feature model for each one of these concerns. However, even though it is easier to manage, for some design activities it can be necessary to merge these feature models (or part of them), and therefore we need operations that enable feature model composition [2]. Among the various composition operations shown in [2], merge-union and merge-intersection have a special place for managing FMs that give different views of a system. Merge-union is an integrated view, while merge-intersection allows to highlight the common core. They are defined using the configuration semantics as follows.

Definition 1 (Merge-intersection [1]).
The merge-intersection operation, denoted by $\cap$*, builds a feature model* FM_3 *from two feature models* FM_1 *and* FM_2 *such that* $[\![FM_3]\!]=[\![FM_1]\!]\cap [\![FM_2]\!]$.

Definition 2 (Merge-union [1]).
The merge-union operation, denoted by $\cup^{\sim}$*, builds a feature model* FM_3 *from two feature models* FM_1 *and* FM_2 *such that* $[\![FM_3]\!]\supseteq[\![FM_1]\!]\cup [\![FM_2]\!]$*. This is an approximate union.*

Definition 3 (Merge-strict-union [1]).
The merge-strict-union operation, denoted by $\cup$*, builds a feature model* FM_3 *from two feature models* FM_1 *and* FM_2 *such that* $[\![FM_3]\!]=[\![FM_1]\!]\cup [\![FM_2]\!]$.

By definition, the merge-strict-union is a restriction of the merge-union. Figure 2 illustrates merge-intersection and merge-strict-union on a simple example, with $[\![F_1]\!]= \{\{A, B\}, \{A, B, C\}\}$ and $[\![F_2]\!]= \{\{A, B\}, \{A, B, D\}\}$. Thus intersection and strict union are as follows: $[\![F_1]\!]\cap [\![F_2]\!]= \{\{A, B\}\}$ and $[\![F_1]\!]\cup [\![F_2]\!]= \{\{A, B\}, \{A, B, C\}, \{A, B, D\}\}$. An example of a merge-union is given in next Section 3.2.

3.2 Comparing main implementations of composition operations

Several methods have been proposed for implementing merge-union and merge-intersection. The two main approaches are based on the feature tree structure or on the logic formula associated with the FMs [2]. Both take as input two feature models.

The approach based on logic formulas consists in using the logic formulas that are equivalent to the FMs to be merged. In our case, a formula for F_1 can be $(A \wedge B) \vee (A \wedge B \wedge C)$, while a formula for F_2 can be $(A \wedge B) \vee (A \wedge B \wedge D)$. In [2], the proposed formula for the merge-intersection (resp. merge-strict-union) is given by Equation 2 (resp. Equation 3). While the approach is sound and complete, and can be implemented using the FM to derive the logic formula

as defined in [11], it needs to be completed by a second step consisting in FM extraction from the logic formula, for example, using the approach given in [19].

$$(((A \wedge B) \vee (A \wedge B \wedge C)) \wedge (\neg D)) \wedge (((A \wedge B) \vee (A \wedge B \wedge D)) \wedge (\neg C)) \quad (2)$$

$$(((A \wedge B) \vee (A \wedge B \wedge C)) \wedge (\neg D)) \vee (((A \wedge B) \vee (A \wedge B \wedge D)) \wedge (\neg C)) \quad (3)$$

For discussing the structural approach based on the feature tree, we need to introduce a slightly more complicated example, and we use a follow up of the e-commerce example. Figure 3 presents two feature models representing e-commerce applications and that are to be merged. Table 1 presents their respective sets of valid configurations. These configurations are given an identifier (such as FM_1C_1) for later use in the paper. The structural approach is based on a set of composition rules which compute the merge-intersection and the merge-union. These rules are listed in [2]. Their result is shown in Figure 4. For example, a rule for merge-union composition establishes that the *xor group* below `Catalog` of FM_2, when merged with the mandatory `grid` feature of `Catalog` in FM_1, gives an *or group* below `Catalog` in the merge-union (see right-hand side of Figure 4). An underlying hypothesis in this approach is that the same set of features is shared by the two FMs to be merged. In our case, this is not the case and the rules sometimes produce a non-strict merge-union, as for example configuration $\{Ec, Ca, G, L, Pm, Ch\}$ is not in the merge-strict-union of the configurations appearing in Table 1: this configuration indeed contains L which is not available in FM_1, and Ch which is not available in FM_2. A main characteristic of this approach is that the rules do not reconsider all the ontological semantics and especially the child-parent relationships. In our example, two different solutions for attaching `payment_method` may be considered: below `e_commerce`, as it is preserved by the structural rule or below `catalog`, which is an alternative that can be considered by a designer (with a "part-of" semantics in the associated software components), but is not proposed by the rule. Besides, it is important to underline that this approach does not take into account the cross-tree constraints, if some exist.

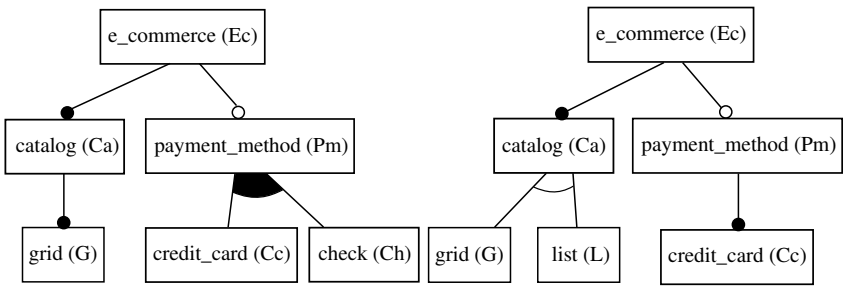


Fig. 3. Two feature models (left-hand, FM_1 and right-hand, FM_2) representing e-commerce applications and that are to be merged.

Table 1. Configuration sets of FM_1 and FM_2 from Figure 3.

	Ec	Ca	G	Pm	Cc	Ch
FM_1C_1	x	x	x			
FM_1C_2	x	x	x	x	x	
FM_1C_3	x	x	x	x		x
FM_1C_4	x	x	x	x	x	x

	Ec	Ca	G	L	Pm	Cc
FM_2C_1	x	x	x			
FM_2C_2	x	x		x		
FM_2C_3	x	x	x		x	x
FM_2C_4	x	x		x	x	x

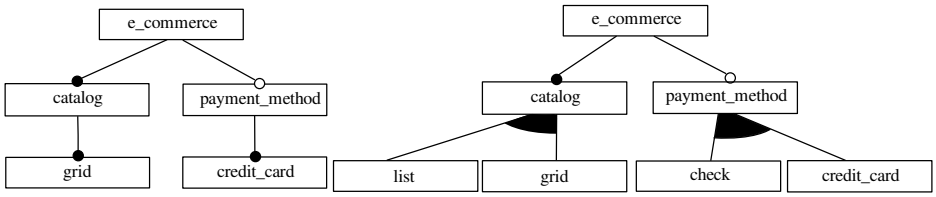


Fig. 4. Merge-intersection (left-hand side) and merge-union (right-hand side) of FM_1 and FM_2 from Figure 3 using the structural approach of [2].

This is why, despite the qualities of these approaches, it is useful to have a complementary point of view, based on the configuration set, which ensures the soundness and completeness of merge-intersection and merge-strict-union operations, is able to take into account cross-tree constraints and FMs with different feature sets, and does not confine the designer in a specific ontological view (if a FM is badly designed, but its configuration set is correct, our approach produces a correct result), while assisting her/him in the FM construction. We propose such a solution in the following section. The solution can be used for merging two feature models, or one feature model and a configuration set, or two configuration sets.

4 Formal Concept Analysis for Feature Model Composition

In Section 4.1, we briefly present the notion of *formal context*. A formal context is an input data for Formal Concept Analysis (FCA). We define (merge) intersection and strict union in terms of operations on formal contexts and illustrate the definitions in the context of variability modeling. The *conceptual structures* that are built by FCA are presented in Section 4.2. Then, we introduce an intermediate structure, the *Equivalence Class Feature Diagram (ECFD)* in Section 4.3. The ECFD associated with a configuration set is a canonical, graphical representation of variability: all FMs having the same configuration set have a projection in the ECFD (and can be extracted from it). We show how the ECFD can assist the designer in building feature models which are consistent with the

domain. At last, we extend the scope of the study by an approach for extracting the *common sub-configurations* in Section 4.4.

4.1 Formal Contexts for intersection and strict union

Formal Concept Analysis (FCA) is a mathematical data analysis framework for hierarchical clustering and rule extraction [22]. In its basic form, it concentrates on a restricted application of Galois connection and Galois lattice theory to binary relationships [12, 10, 20]. As input, it takes a set of objects described by a set of attributes, arranged in a tabular form called a formal context.

Definition 4 (Formal context). *A formal context K is a 3-tuple (G, M, I), where G is an object (configuration) set, M is an attribute (feature) set, and $I \subseteq G \times M$ is a binary relation which associates objects (configurations) with attributes (features) they own. Given a context $K = (G, M, I)$, for $g \in G$ we will denote by $I(g)$ the set of features of g, i.e. the set $\{m \in M | (g, m) \in I\}$.*

The two binary relationships of Table 1, which presents the configuration sets of FM_1 and FM_2, are formal contexts. Each row corresponds to an object (a configuration) and each column corresponds to an attribute (a feature). The left-hand side formal context indicates that configuration FM_1C_1 comprises the attributes Ec, Ca, G. We present in the next Section 4.2 the conceptual structures that are extracted from a formal context.

For defining the intersection and strict union formal contexts, we first introduce the notion of equality of objects (configurations), denoted by $\triangleq$, as objects having the same set of attributes. In tables and figures, which are generated by tools, when applicable, $\triangleq$ is denoted by "=".

Definition 5 (Equality of objects, $\triangleq$).

$$g_1 \triangleq g_2 \Leftrightarrow g_1 \in G_1, g2 \in G_2 \text{ and } I_1(g_1) = I_2(g_2)$$

We then define the formal context associated with intersection as the rows that are present in the two initial formal contexts (Definition 6). A labeling of rows is added to indicate their origin. Table 2 shows the formal context associated with the intersection of FM_1 and FM_2 formal contexts from Table 1.

Definition 6 (Intersection formal context). *The formal context of intersection $Inter(K_1, K_2)$ is*
$K_{Inter(K_1,K_2)} = (G_{Inter(K_1,K_2)}, M_{Inter(K_1,K_2)}, I_{Inter(K_1,K_2)})$ such that:

- $G_{Inter(K_1,K_2)} = \{g_{g_1 \triangleq g_2} \mid \exists (g_1, g_2) \in G_1 \times G_2, g_1 \triangleq g_2\}$
- $M_{Inter(K_1,K_2)} = M_1 \cap M_2$
- $I_{Inter(K_1,K_2)} = \{(g_{g1 \triangleq g2}, m) \mid m \in M_{Inter(K_1,K_2)}, g_{g1 \triangleq g2} \in G_{Inter(K_1,K_2)}, (g1, m) \in I_1 (or\ equivalently\ (g2, m) \in I_2)\}$

Definition 7 introduces the formal context associated with strict union. Table 3 shows the formal context associated with the strict union of FM_1 and FM_2 formal contexts from Table 1. It highlights the two common configurations (first two rows) and the configurations that are specific to one FM (next four rows).

Table 2. Formal context associated with the intersection of FM_1 and FM_2 formal contexts from Table 1.

	Ec	Ca	G	Pm	Cc
$FM_1C_1 = FM_2C_1$	x	x	x		
$FM_1C_2 = FM_2C_3$	x	x	x	x	x

Definition 7 (Strict union formal context). *Let us consider:*

- *the set of common configurations (from Def. 6)* $G_{Inter(K_1,K_2)}$ *and the corresponding relation* $I_{Inter(K_1,K_2)}$
- *the set of configurations specific to* G_1*:* $SPE(G_1) = \{g1 \mid g1 \in G_1 \text{ and } \nexists g2 \in G_2, \text{with } g_{g1 \triangleq g2} \in G_{Inter(K_1,K_2)}\}$
- *the set of configurations specific to* G_2*:* $SPE(G_2) = \{g2 \mid g2 \in G_2 \text{ and } \nexists g1 \in G_1, \text{with } g_{g1 \triangleq g2} \in G_{Inter(K_1,K_2)}\}$

The formal context of strict union $Union(K_1, K_2)$ *is:*
$K_{Union(K_1,K_2)} = (G_{Union(K_1,K_2)}, M_{Union(K_1,K_2)}, I_{Union(K_1,K_2)})$ *such that:*

- $G_{Union(K_1,K_2)} = G_{Inter(K_1,K_2)} \cup SPE(G_1) \cup SPE(G_2)$
- $M_{Union(K_1,K_2)} = M_1 \cup M_2$
- $I_{Union(K_1,K_2)} = I_{Inter(K_1,K_2)}$
  $\cup \{(g,m) \mid g \in SPE(G_1), m \in M_{Union(K_1,K_2)}, (g,m) \in I_1\}$
  $\cup \{(g,m) \mid g \in SPE(G_2), m \in M_{Union(K_1,K_2)}, (g,m) \in I_2\}$

Table 3. Formal context associated with the strict union of FM_1 and FM_2 formal contexts from Table 1.

	Ec	Ca	G	L	Pm	Cc	Ch
$FM_1C_1 = FM_2C_1$	x	x	x				
$FM_1C_2 = FM_2C_3$	x	x	x		x	x	
FM_1C_3	x	x	x		x		x
FM_1C_4	x	x	x		x	x	x
FM_2C_2	x	x		x			
FM_2C_4	x	x		x	x	x	

4.2 Conceptual structures

From a formal context, specialized algorithms of the FCA framework build *formal concepts*. A formal concept is a maximal group of objects associated with the maximal group of attributes they share. It can be read in the table of the context as a maximal rectangle of crosses (modulo permutations of rows and columns).

Definition 8 (Formal concept). *Given a formal context $K = (G, M, I)$, a formal concept associates a maximal set of objects with the maximal set of attributes they share, yielding a set pair $C = (Extent(C), Intent(C))$ such that:*

- $Extent(C) = \{g \in G | \forall m \in Intent(C), (g, m) \in I\}$ *is the extent of the concept (objects covered by the concept).*
- $Intent(C) = \{m \in M | \forall g \in Extent(C), (g, m) \in I\}$ *is the intent of the concept (shared attributes).*

For example, $(\{FM_1C_1 = FM_2C_1, FM_1C_2 = FM_2C_3, FM_1C_3, FM_1C_4\}, \{Ec, Ca, G\})$ is a formal concept in strict union of Table 3.

The formal concepts are ordered using inclusion of their extent (or reverse inclusion of their intent). Given two formal concepts $C_1 = (E_1, I_1)$ and $C_2 = (E_2, I_2)$ of K, the concept specialization/generalization order $\preceq_C$ is defined by $C_2 \preceq_C C_1$ if and only if $E_2 \subseteq E_1$ (and equivalently $I_1 \subseteq I_2$). C_2 is a specialization (a subconcept) of C_1. C_1 is a generalization (a superconcept) of C_2. Due to these definitions, C_2 intent inherits (contains) the attributes from C_1 intent, while C_1 extent inherits the objects from C_2 extent. For example, concept $(\{FM_1C_1 = FM_2C_1, FM_1C_2 = FM_2C_3, FM_1C_3, FM_1C_4\}, \{Ec, Ca, G\})$ is a superconcept of concept $(\{FM_1C_3, FM_1C_4\}, \{Ec, Ca, G, Pm, Ch\})$ in strict union of Table 3.

Definition 9 (Concept lattice). *If we denote by $\mathcal{C}_K$ the set of all concepts of K, $\mathcal{L}_K$=($\mathcal{C}_K$, $\preceq_C$), is the concept lattice associated with K.*

The graphical representation of the conceptual structures (as concept lattices) exploits the inclusion property to avoid representing in the concepts the top-down inherited attributes (features) and the bottom-up inherited objects (configurations). An attribute appears in the highest concept that possesses this attribute. We say that this concept introduces the attribute, and it is then an Attribute-Concept. An object appears in the lowest concept that possesses this object. We say that this concept introduces the object, and it is then an Object-Concept. A concept is represented in this document by a three-parts box. The upper part is the concept name; the middle part contains the simplified intent (deprived of the top-down inherited attributes); the bottom part contains the simplified extent (deprived of the bottom-up inherited objects).

Specific suborders, that contain only some concepts, can be isolated in the concept lattice. In these structures, configurations are organized depending on the features they share, and dually, the features are structured depending on the configurations in which they are. Thus, these structures permit to emphasize and extract information about variability. The difference is that some of them keep only some of this variability information. The AOC-poset (Attribute Object Concept partially ordered set) contains only the concepts introducing at least one object (configuration), or at least one attribute (feature), or both. In the AOC-poset (as in the concept lattice) a configuration (resp. a feature) appears only once, thus we have a maximal factorization of configurations and features. Another interesting conceptual structure to address our problem is the AC-poset (Attribute-Concept poset) where one configuration may appear several times (and be introduced by several lowest concepts), but features remain

maximally factorized revealing an even more simple structure, focusing on the representation of the feature hierarchy. The AC-poset is the minimal conceptual structure necessary to extract logical dependencies between features. The four structures: formal context, concept lattice, AOC-poset and AC-poset are equivalent, in the sense that each one can be rebuilt from any other one, without ambiguity.

Left-hand side of Figure 5 shows the AC-poset associated with the formal context of Table 3. It emphasizes: *co-occuring features*, e.g. `e_commerce` and `catalog` always appear together in any configuration; *implication between features*, e.g. when a configuration has the feature `list` it always has the feature `catalog`; *mutually exclusive features*, e.g. `list` and `grid` never appear together in any configuration; and *feature groups*, e.g. when `payment_method` is in a configuration, there is at least `check` or `credit_card`, or they are both present. As this kind of information is rather difficult to read in an AC-poset, in the next section, we propose an equivalent diagrammatic representation that we have called the Equivalence Class Feature Diagram (ECFD) and which is closer to the FM.

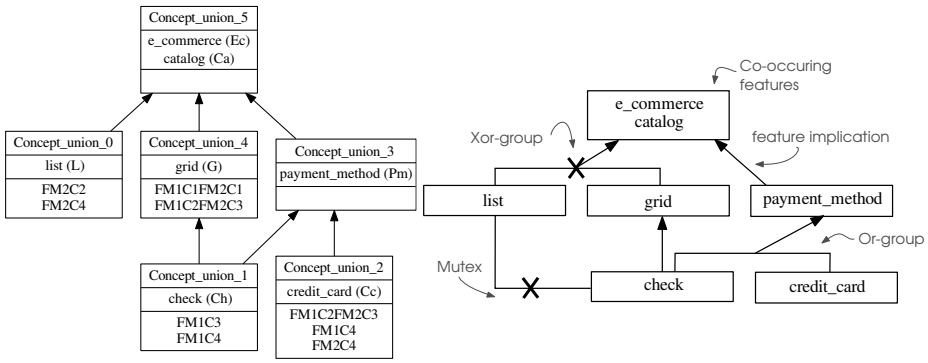


Fig. 5. Left-hand side: AC-poset associated with the strict union formal context of Table 3. Right-hand side: ECFD extracted from the AC-poset.

4.3 Equivalence Class Feature Diagram (ECFD)

The ECFD seeks to be more intuitive than the AC-poset to read variability information. It depicts the feature dependencies extracted from the initial set of configurations that are summarized in the AC-poset, in a representation close to a feature model but without explicit ontological semantics. Therefore it includes all equivalent feature models, hence its name.

Figure 5 shows the ECFD (right-hand side) extracted from the AC-poset (left-hand side). Co-occuring features (as `e_commerce` and `catalog`) are in a same box. Arrows between boxes represent feature implications (like `check` implies

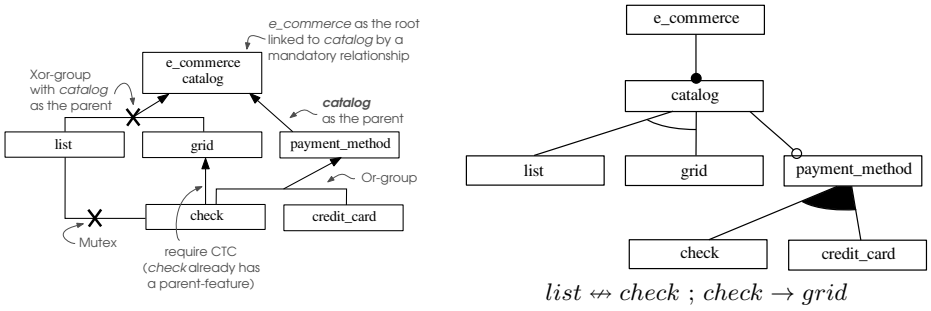


Fig. 6. Left-hand side: ECFD for the strict union formal context of Table 3 annotated with designer choices. Right-hand side: First extracted FM.

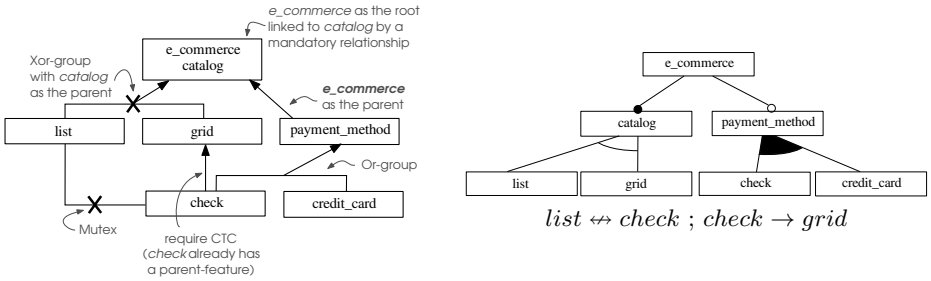


Fig. 7. Left-hand side: Reminder of the ECFD for the strict union formal context of Table 3 annotated with designer choices. Right-hand side: Second extracted FM.

`grid`). Groups of boxes connected by horizontal lines rooted in an upper feature summarize feature groups (like `list` and `grid` rooted in box `e_commerce-catalog`, or `check` and `credit_card` rooted in box `payment_method`). *Xor groups* are marked with a cross. A cross also represents mutually exclusive features, also called *mutex* (like `list` and `check`) when they do not belong to a group. The constructs and the semantics of the ECFD are more generally given in Table 4 and a construction algorithm is available in [16]. If we consider an AC-poset corresponding to a set of feature models with the same configuration set, all these FMs *conform* to the AC-poset. This means that each dependency expressed in these feature models matches a dependency expressed in the corresponding AC-poset. For instance, if there is a child-parent (f_c, f_p) in one FM, it belongs to the AC-poset in this way: let C_c be the concept introducing f_c and let C_p be the concept introducing f_p, we have $C_c \preceq_C C_p$.

The ECFD structures the variability information extracted from the configuration set, and it can guide the expert in assigning ontological semantics on its logical dependencies. Figures 6 and 7 show the guidance process. The two FMs at the right-hand side of the figure have the same configuration-semantics. To obtain them, first the designer has to choose between `e_commerce` and `catalog`.

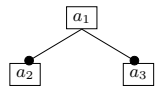
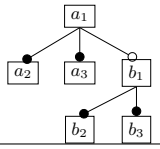
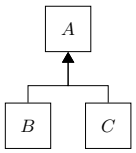
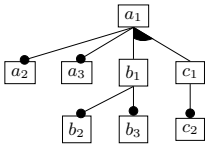
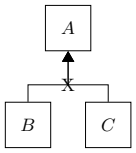
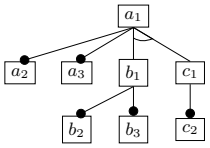
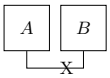
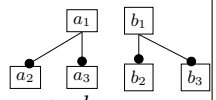
construct	semantics	ex. of conform FM
$A = \{a_1, \cdots, a_{n_A}\}$	features a_i are always present together (or absent)	
$A = \{a_1, \cdots, a_{n_A}\}$ $B = \{b_1, \cdots, b_{n_B}\}$	When b_i features are present, all a_i features are present as well	
	or-group: when a_i features are present, either the b_i are present, or the c_i are present, or the b_i and the c_i are present	
	xor-group: when a_i features are present, either the b_i are present, or the c_i are present (not both)	
	mutex: features a_i and features b_i are nether present together	 $a_1 \rightarrow \neg b_1$ $b_1 \rightarrow \neg a_1$

Table 4. Equivalence class feature diagram (ECFD): constructs and semantics [15]. The third column is an example of conform feature model with $n_A = n_B = 3$ and $n_C = 2$.

One is the root (e.g. `e_commerce`), and the other (e.g. `catalog`) is a mandatory feature connected to the root. The *xor group* `list` and `grid` has to be connected to `e_commerce` or to `catalog`. The designer here chooses `catalog` as the parent of the group. Then `payment_method` is connected either to `catalog` (Figure 6) or to `e_commerce` (Figure 7). Feature `check` can be a child of `grid`, or it can belong to the *or group* (`check, credit_card`, rooted in `payment_method`). The second choice is made. The cross-tree constraints `list` $\nleftrightarrow$ `check` and `check` $\rightarrow$ `grid` are added to the FMs.

The left-hand side of Figure 8 shows the ECFD extracted from the intersection AC-poset, built from Table 2. In this very simple case, the difference between the AC-poset and the ECFD is only that in the AC-poset, the nodes (concepts) also contain the list of configurations. The right-hand side shows a possible FM extracted from the ECFD. The top box of co-occurring features gives rise to mandatory feature `grid` refining mandatory feature `catalog` refining root `e_commerce`. With the bottom box, the designer chooses to insert `payment_method` as an optional sub-feature of `catalog`, and `credit_card` as a mandatory feature refining `payment_method`.

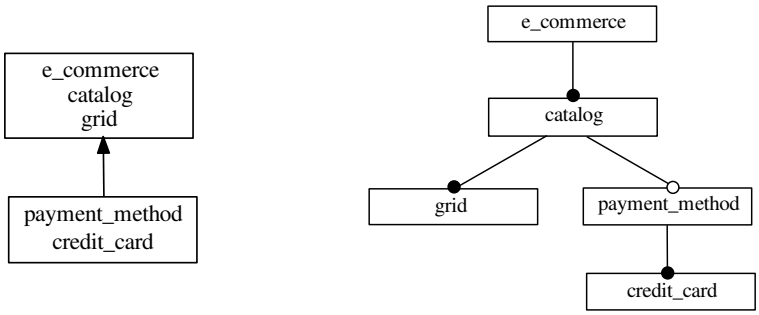


Fig. 8. Left-hand side: ECFD for the intersection formal context of Table 2. Right-hand side: An extracted FM.

4.4 Extraction of common sub-configurations

As we noticed during our evaluation (reported in the next section), while FM strict union is always informative, FM intersection is often empty, even when the initial FMs have similarities. We illustrate this issue with a slight modification of the e-commerce example. To the configurations of FM_1, we add `UserManagement (Um)` as a mandatory sub-feature of `e_commerce (Ec)` (the new FM is denoted by FM_{1_e}). To the configurations of FM_2, we simply add `Paypal (Pp)` as a mandatory sub-feature of `Credit_Card (Cc)` (the new FM is denoted by FM_{2_e}). After these additions, there is no more common configuration to FM_{1_e} and FM_{2_e}. Table 5 shows the extended formal contexts for FM_{1_e}, FM_{2_e} and the

strict union formal context. Figure 9 shows the AC-poset built from the union formal context.

Table 5. Top: Configuration sets of FM_{1_e} and FM_{2_e} from FM_1 and FM_2 of Figure 3 extended with `UserManagement (Um)` and `Paypal (Pp)`. Bottom: Strict union formal context $FM_{1_e} \bigcup FM_{2_e}$.

FM_{1_e}	Ec	Ca	G	Pm	Cc	Ch	Um
FM_1eC_1	x	x	x				x
FM_1eC_2	x	x	x	x	x		x
FM_1eC_3	x	x	x	x		x	x
FM_1eC_4	x	x	x	x	x	x	x

FM_{2_e}	Ec	Ca	G	L	Pm	Cc	Pp
FM_2eC_1	x	x	x				
FM_2eC_2	x	x		x			
FM_2eC_3	x	x	x		x	x	x
FM_2eC_4	x	x		x	x	x	x

$UnionExt$	Ec	Ca	G	L	Pm	Cc	Ch	Um	Pp
$FM1eC1$	x	x	x					x	
$FM1eC2$	x	x	x		x	x		x	
$FM1eC3$	x	x	x		x		x	x	
$FM1eC4$	x	x	x		x	x	x	x	
$FM2eC1$	x	x	x						
$FM2eC2$	x	x		x					
$FM2eC3$	x	x	x		x	x			x
$FM2eC4$	x	x		x	x	x			x

Concepts in the AC-poset highlight different types of information on common parts and differences between the FMs. Their study allows us to determine a common core in feature combinations and to categorize the sub-configurations:

- *(Specific sub-configuration)* When the (complete) extent only contains configurations from one feature model, the intent is a sub-configuration or a valid configuration for this feature model only. In both cases, it is specific to this feature model and does not belong to a common core.
- *(Core sub-configuration)* When the (complete) extent contains configurations from both feature models, the intent is a partial common configuration (in a broad meaning, namely it can be a valid configuration) and:
  - *(Configuration)* If the simplified extent contains one configuration of both feature models, the intent is a valid configuration for both and it is in the intersection which is not empty (it was the case for `Concept_Union_4` in Figure 5).
  - *(Strict semi-partial sub-configuration)* If the simplified extent only contains configurations from one feature model (as for `Concept_UnionExt_6` in Figure 9, whose simplified extent only contains configurations from FM_{2_e}), the intent is a strict partial configuration for the feature model which has no configuration in the simplified extent (here FM_{1_e}) and a valid configuration for the other (here FM_{2_e}).
  - *(Strict partial sub-configuration)* If the simplified extent is empty (as for `Concept_UnionExt _7` and `Concept_ UnionExt_5` in Figure 9), the intent

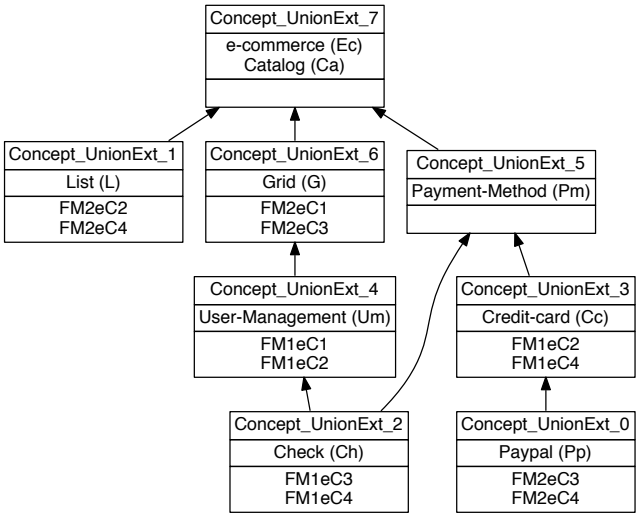


Fig. 9. AC-poset associated with the strict union of the configuration sets of FM_{1_e} and FM_{2_e}.

is a strict partial configuration. It is not valid for neither of the feature models, but it is contained inside some of the configurations of both feature models and it highlights a similarity between them.

When the intersection is small or empty, the concepts of the core category are especially useful for exploring more deeply the commonalities between the two feature models. They represent possibly incomplete configurations, from which specific features present in only one FM (like `Um` to `Pp`), or specific combinations have been removed. From these concepts, we can build an approximate intersection (denoted by $\bigcap^{\sim}$). The corresponding formal context is built by keeping the intents of the core concepts and assigning them arbitrary configuration names (as common sub-configurations, possibly incomplete).

Figure 10 show the formal context (left-hand side) and the AC-poset (right-hand side) associated with $FM_1 \bigcap^{\sim} FM_2$ and $FM_{1_e} \bigcap^{\sim} FM_{2_e}$. They are identical. The difference lies in the fact that when considering $FM_1 \bigcap^{\sim} FM_2$, the formal context contains the configurations of $FM_1 \bigcap FM_2$ (which is not empty).

Figure 11 shows the ECFD extracted from this AC-poset. This is not appropriate in this case to build the groups and the mutex. For example, in the intents of the core concepts, features `G` and `Pm` never appear together, while they may appear together in complete valid configurations. Appropriate information that can be read is: co-occurring features, mandatory features, optional features and implications. In this example, two possible FMs that can be derived by an expert. In this specific case, she/he will preferably choose the FM where `e_commerce` is the root. Here, the approximate intersection is simple, but in the general case,

	Ec	Ca	G	Pm	Cc
c1	x	x			
c2	x	x	x		
c3	x	x		x	
c4	x	x		x	x

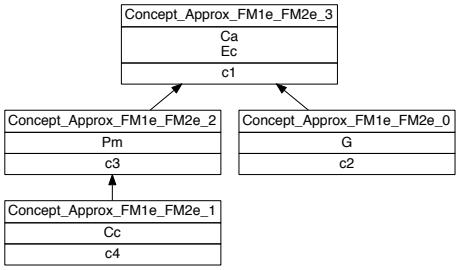


Fig. 10. Left-hand side: Formal context of $FM_1 \bigcap^{\sim} FM_2$ and $FM_{1_e} \bigcap^{\sim} FM_{2_e}$. Right-hand side: The corresponding AC-poset.

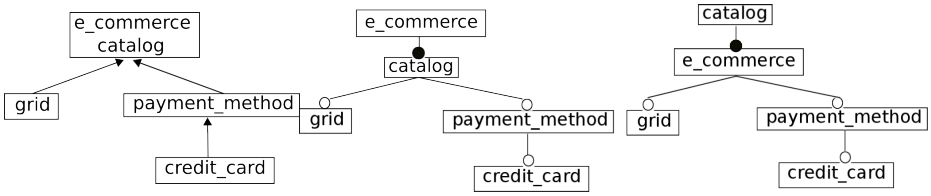


Fig. 11. Left-hand side: ECFD of the approximate intersection. Right-hand side: Two possible FMs extracted from the ECFD.

.

complex ECFDs can be found and the expert benefits from their full potential: compared to a logic-based approach, building an FM for an approximate intersection is guided; compared to a feature tree structure-based approach, no presupposition is made about the ontological relations.

5 Implementation and Assessment

The approach has been implemented as presented in Figure 12. It uses two existing tools. `Familiar`[3] [3] is an executable Domain Specific Language, provided with an environment allowing to create, modify and reason about FMs. In the current project, we use it to build the configuration set of an FM. `rcaexplore`[4] is a framework for Formal Concept Analysis which offers a variety of analysis kinds. It is used to build the AC-poset from which the ECFD structure (nodes and edges) is extracted. We also developed specific tools for this project: `ConfigSet2FormalContext` builds a formal context (within input format of `rcaexplore`) from a configuration set extracted with Familiar; `ComputeInterAndUnion` builds the intersection and strict union formal contexts; `ComputeGroupsAndMutex` computes the groups *Xor*, *Or* and the mutex of the

[3] https://nyx.unice.fr/projects/familiar/
[4] http://dolques.free.fr/rcaexplore/

ECFD. To obtain the approximate intersection, an additional tool, `ComputeApprox` computes the core concepts of the AC-poset and an ECFD without groups or mutex, which are not appropriate in this case.

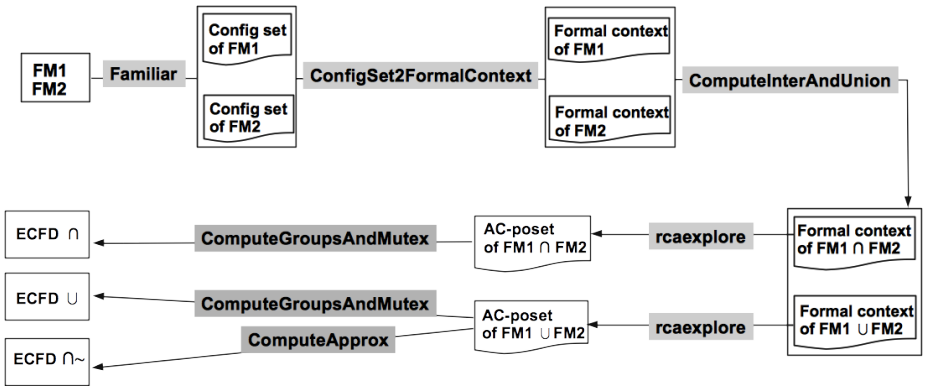


Fig. 12. The implemented process (extended from [15]).

We apply the approach on several feature models that own from 4 to 864 configurations, and from 6 to 26 features. Some are taken from the SPLOT repository[5] [27], from the Familiar[6] website, or from the literature, and we also made some variants of these feature models. In Table 6, we give the number of features, configurations, *xor groups*, *or groups* and constraints of each selected FM. We also compute the ECFD and indicate the number of *xor groups*, *or groups*, mutex, situations where a box in the ECFD has several direct parents (*multi-par.*), and nodes. The number of groups, e.g. *xor groups*, may vary between the FM and the ECFD. For example, one *xor group* of the ECFD may combine several *xor groups* of the FM when there are additional constraints, or the ECFD may reveal more possible *xor groups* than initially indicated in the FM.

Table 7 presents information about intersection, approximate intersection and strict union. The built ECFDs have reasonable size compared to the input FMs, with node number ranging from 2 to 22, mutex from 0 to 32, *xor* and *or groups* from 0 to 16, and very few multi-parent situations. This is encouraging if we consider that experts have to extract FMs, guided by the ECFDs.

The last two columns of Table 7 respectively show the difference between: intersection and approximate intersection; strict union and approximate intersection. For example, if we consider FM1 $\cap^{\sim}$ FM2, we can notice that union FM1 $\cup$ FM2 feature number (7) is 40% more than the FM1 $\cap^{\sim}$ FM2 feature number (5). FM1 $\cup$ FM2 sub-configurations (or AC-poset node number) (6) are 50% more than the FM1 $\cap^{\sim}$ FM2 feature number (4). Intersection FM1 $\cap$ FM2 feature number (5) is 0% less than the FM1 $\cap^{\sim}$ FM2 feature number

[5] http://www.splot-research.org/
[6] http://familiar.variability.io/

Table 6. FMs (and the corresponding ECFDs) used for testing the approach [15]. *var.* stands for *variant*. *Cst* stands for *Constraint*. e-commerce FMs are the examples of this paper, Eshop FMs come from SPLOT, Wiki FMs come from FAMILIAR documentation (or are variants), Bicycle FMs are variants of Mendonca SPLOT FMs.

FM	Feature Model					ECFD				
	#feat	#conf.	#Xor	#Or	# Cst	#Xor	#Or	#mutex	#multi-par.	#nodes
FM1 (e-com.)	6	4	0	1	0	0	1	0	0	3
FM2 (e-com.)	6	4	1	0	0	1	0	0	0	4
Martini Eshop	11	8	1	1	1	1	2	1	1	6
Tang Eshop	10	13	1	1	2	1	2	1	1	7
Toacy Eshop	12	48	1	2	0	1	2	0	0	9
Wiki-V1	14	10	4	0	4	3	2	5	2	9
Wiki-V2 (V1 var.)	17	50	4	1	4	6	13	1	1	13
Wiki-V3 (V1 var.)	18	120	3	2	6	2	2	1	0	13
Bicycle1	19	64	2	0	2	1	0	0	0	10
Bicycle2	22	192	5	0	1	6	1	6	0	17
Bicycle3	25	576	4	0	2	5	1	8	0	19
Bicycle4	26	864	5	0	2	6	1	8	0	21

(5). FM1 $\cap$ FM2 sub-configurations (or AC-poset node number) (2) are 50% less than the FM1 $\cap^{\sim}$ FM2 feature number (4). When intersection is empty, a relatively low difference between approximate intersection and strict union (like the Martini-Tang case, with 33%) indicates a good similarity between the FMs, not highlighted by the configuration-semantics. Reversely, when intersection is empty, but the difference between approximate intersection and union is high (like the WikiV1-WikiV3 case, with more than 150% for features, and 260% for common sub-configurations) reveals a low similarity. When approximate intersection is close to intersection (like the Bicycle3-Bicycle4 example), this means that the configuration-semantics is well captured by the common sub-configurations and features. When approximate intersection feature number is close to intersection feature number (like the Bicycle1-Bicycle2 example, with -6.7%), but this is not the case for node number (-40%), this means that there are many common features, but the configuration-semantics is not well captured by the common sub-configurations. This information also can guide an expert in her/his composition process.

6 Related Work and Discussion

Formal Concept Analysis has many applications in software engineering as was summarized in [36] for the period 1992-2003. Since this period new applications appeared, that range from fault localization [17] to bad smells and design patterns detection [9], suggest appropriate refactorings to correct some design defects [28], or analyzing software version control repositories [23]. In the domain of SPLE, FCA serves as a foundation for different approaches. Loesch and Ploederer [26] analyze the concept lattice between configurations and features to find variability information such as the co-occurring features, groups of features that are never present together, etc. This analysis helps extracting constraints

Table 7. Merge-intersection, approximate intersection, and merge-strict union ECFDs (extended from [15]). #conf. (resp. #subconf) is the number of different configurations for intersection and strict-union (resp. sub-configurations for approximate intersection). NA stands for "Non Applicable".

FM	Formal context		ECFD					% diff with $\cap^{\sim}$	
	#feat	#(sub)conf.	#Xor	#Or	#mutex	#multi-par.	#nodes	#feat.	#nodes
FM1 ∩ FM2	5	2	0	0	0	0	2	-0%	-50%
FM1 $\cap^{\sim}$ FM2	5	4	NA	NA	NA	0	4	.	.
FM1 ∪ FM2	7	6	1	1	1	1	6	+40%	+50%
Martini∩Tang	0	0	0	0	0	0	0	-100%	-100%
Martini $\cap^{\sim}$ Tang	9	6	NA	NA	NA	0	6	.	.
Martini∪Tang	12	21	1	2	3	1	8	+33%	+33%
Martini∩Toacy	0	0	0	0	0	0	0	-100%	-100%
Martini $\cap^{\sim}$ Toacy	9	6	NA	NA	NA	0	6	.	.
Martini∪Toacy	14	56	1	1	4	0	10	+56%	+67%
Tang∩Toacy	8	5	1	2	0	0	5	-11.1%	-16.7%
Tang $\cap^{\sim}$ Toacy	9	6	NA	NA	NA	0	6	.	.
Tang∪Toacy	13	56	1	1	4	1	10	+44%	+67%
WikiV1∩WikiV2	0	0	0	0	0	0	0	-100%	-100%
WikiV1 $\cap^{\sim}$ WikiV2	11	7	NA	NA	NA	0	7	.	.
WikiV1∪WikiV2	20	60	5	6	26	0	16	+82%	+129%
WikiV1∩WikiV3	0	0	0	0	0	0	0	-100%	-100%
WikiV1 $\cap^{\sim}$ WikiV3	9	5	NA	NA	NA	0	5	.	.
WikiV1∪WikiV3	23	130	3	4	42	0	18	+156%	+260%
WikiV2∩WikiV3	11	6	2	0	1	0	6	-0%	-45.5%
WikiV2 $\cap^{\sim}$ WikiV3	14	11	NA	NA	NA	0	11	.	.
WikiV2∪WikiV3	21	164	6	14	10	1	17	+50%	+55%
Bicycle1∩Bicycle2	14	8	1	0	0	0	6	-6.7%	-40%
Bicycle1 $\cap^{\sim}$ Bicycle2	15	10	NA	NA	NA	0	10	.	.
Bicycle1∪Bicycle2	26	248	6	1	32	2	21	+73%	+110%
Bicycle3∩Bicycle4	23	288	5	1	8	0	18	-4.2%	-5.3%
Bicycle3 $\cap^{\sim}$ Bicycle4	24	19	NA	NA	NA	0	19	.	.
Bicycle3∪Bicycle4	27	1152	6	1	8	0	22	+13%	+16%

or reorganizing features, e.g. by merging or removing some of them. These ideas are deepen and reused in feature model analysis or synthesis in [38, 31, 6, 35]. Another available tool in the framework of FCA is the notion of implicative systems, used in [31]. This is another logical encoding of the formula which is equivalent to a concept lattice (or to a feature model), which can be rather compact. The relationship between scenarios, functional requirements and quality requirements is studied in [29]. FCA-based identification of features in source code has been studied for software product line in [37, 7], where they use the description of software variants by source code elements. Finding traceability links between features and the code is more specifically studied in [32]. In [21], authors analyze source code parts and scenarios which execute them and use features, with the purpose to identify parts of the code which correspond to feature implementation. Carbonnel et al. analyze PCMs from Wikipedia or randomly generated to evaluate the scale up of FCA on this type of data in [14] and the associated ECFDs in [16].

Several approaches for FM composition are compared in [1] and [4]. In [33] and [24] the input feature models are maintained separately and links are established between them through constraints. The approach of [2] establishes, in a first phase, the matching between similar elements, then an algorithm recursively merges the feature models with structural rules. Catalogs of local transformation rules are proposed in [34, 8]. Other approaches encode the FMs into propositional formulas [11], then compute the formula representing the intersection (resp. the union), then synthesize a FM from the boolean formula [19]. The logic and structural approaches have been illustrated and discussed in Section 3.2 and our approach was illustrated with the example used for illustrating the structural approach.

Compared to the logic approach, our approach also is sound and complete, and we produce a structure (the ECFD with all feature groups and mutex) which assists the expert in the extraction of the composed FM. Compared to the structural approach, ours does not make any presupposition about which relations are ontological, allowing to fix possible mistakes in the initial FMs. In our approach, the configuration-semantics and the non-contradictory ontological child-parent edges are preserved. We accept FMs with different feature sets, and we take into account cross-tree constraints. Our approach computes the merge-strict-union, the merge-intersection, and we also compute an approximate intersection, which is useful when the configuration sets to be merged have an empty, or small, intersection, and in general, for having a core description of the two FMs. When there are hierarchy mismatches, the AC-poset manages this information but the vocabulary (feature names) has to be the same (it can be aligned before the merge operations).

Our approach needs to know the list of configurations, thus as such, the proposed solution is restricted to some contexts: FMs that have limited number of configurations; real-world product lines given with configuration sets. Many FMs have a very large configuration set, as Video player FM from SPLOT, with 71 features and more than 1 billion configurations. We do not address these cases, as we more specifically address the contexts where the FMs have a reasonable number of configurations, which corresponds in particular to FMs coming from real-world product lines. Concerning product lines inducing a number of configurations not tractable by FCA, our approach also could benefit from product line decomposition: dividing a feature model according to scopes, concerns or teams into less complex interdependent feature models. Besides, the paper [13] gives a procedure to derive (in a polynomial time) an implicative system directly from a feature model, thus without using the configuration set which may be an obstacle in some cases as noticed by [31]. The logical semantics is guaranteed by the FCA framework. The computational complexity is polynomial for AC-posets, in the size of the number of configurations and the number of features. Thus this is very different from the complexity of concept lattices, which may be exponential in worst cases. As detailed by [31], ECFD group and *mutex* computation might be exponential in the number of configurations or features but remains reasonable in typical situations, with an optimized implementation.

7 Conclusion

We have proposed an approach to assist designers in configurations-based FM composition. We focused on strict union, intersection and approximate intersection. FCA was used to represent all the FMs with the same configuration semantics through a canonical form, the ECFD (Equivalence Class Feature Diagram). Our approach may take into account different feature sets and structures, as well as cross-tree constraints. It allows to reset the ontological relationships. We have implemented our approach and we have tested it on concrete examples.

As future work, we would like to investigate more the approximate intersection. More specifically, from the intersection and union AC-posets, we would like to define similarity metrics, e.g. based on the size of intents and the number of concepts of each category (strict partial, strict semi-partial, configuration). We also would like to define a composition approach based on implicative systems, to discard the limit imposed by the current need to have the configuration set. Let us notice that having the configuration set is not always a limit, as in concrete product line, this is the standard data.

References

1. Acher, M., Collet, P., Lahire, P., France, R.B.: Comparing approaches to implement feature model composition. In: 6th Eur. Conf. on Modelling Foundations and Applications (ECMFA). pp. 3–19 (2010), http://dx.doi.org/10.1007/978-3-642-13595-8_3
2. Acher, M., Collet, P., Lahire, P., France, R.B.: Composing feature models. In: 2nd Int. Conf. Software Language Engineering (SLE), 2009, Revised Selected Papers. LNCS, vol. 5969, pp. 62–81. Springer (2010), http://dx.doi.org/10.1007/978-3-642-12107-4_6
3. Acher, M., Collet, P., Lahire, P., France, R.B.: Familiar: A domain-specific language for large scale management of feature models. Science of Computer Programming (SCP) 78(6), 657–681 (2013)
4. Acher, M., Combemale, B., Collet, P., Barais, O., Lahire, P., France, R.: Composing your Compositions of Variability Models. In: 16th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS). LNCS, vol. 8107, pp. 352–369 (2013), https://hal.inria.fr/hal-00859473
5. Achtaich, A., Roudies, O., Souissi, N., Salinesi, C.: Selecting SPL Modeling Languages: a Practical Guide. In: 3rd IEEE World Conf. on Complex Systems (WCCS). Marrakech, Morocco (2015), https://hal.archives-ouvertes.fr/hal-01527521
6. Al-Msie'deen, R., Huchard, M., Seriai, A., Urtado, C., Vauttier, S.: Reverse engineering feature models from software configurations using formal concept analysis. In: 11th International Conference on Concept Lattices and Their Applications (CLA). pp. 95–106 (2014), http://ceur-ws.org/Vol-1252/cla2014_submission_13.pdf
7. Al-Msie'deen, R., Seriai, A., Huchard, M., Urtado, C., Vauttier, S., Salman, H.E.: Mining Features from the Object-Oriented Source Code of a Collection of Software Variants Using Formal Concept Analysis and Latent Semantic Indexing. In: 25th

Conf. on Software Engineering and Knowledge Engineering (SEKE). pp. 244–249 (2013)
8. Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., de Lucena, C.J.P.: Refactoring product lines. In: 5th Int. Conf. on Generative Programming and Component Engineering (GPCE). pp. 201–210 (2006), http://doi.acm.org/10.1145/1173706.1173737
9. Arévalo, G., Ducasse, S., Gordillo, S., Nierstrasz, O.: Generating a catalog of unanticipated schemas in class hierarchies using formal concept analysis. Inf. Softw. Technol. 52, 1167–1187 (November 2010)
10. Barbut, M., Monjardet, B.: Ordre et Classification (volume 2). Hachette (1970)
11. Batory, D.S.: Feature Models, Grammars, and Propositional Formulas. In: 9th Int. Conf. on Software Product Lines (SPLC). pp. 7–20 (2005)
12. Birkhoff, G.: Lattice theory, Colloquium publications, vol. 25. American Mathematical Society (1940)
13. Carbonnel, J., Bertet, K., Huchard, M., Nebut, C.: FCA for software product lines representation: Mixing product and characteristic relationships in a unique canonical representation. In: 13th International Conference on Concept Lattices and Their Applications (CLA). pp. 109–122 (2016), http://ceur-ws.org/Vol-1624/paper9.pdf
14. Carbonnel, J., Huchard, M., Gutierrez, A.: Variability representation in product lines using concept lattices: Feasibility study with descriptions from wikipedia's product comparison matrices. In: 1st International Workshop on Formal Concept Analysis and Applications, FCA&A 2015, co-located with 13th International Conference on Formal Concept Analysis (ICFCA). pp. 93–108 (2015), http://ceur-ws.org/Vol-1434/paper7.pdf
15. Carbonnel, J., Huchard, M., Miralles, A., Nebut, C.: Feature model composition assisted by formal concept analysis. In: 12th Int. Conf. on Evaluation of Novel Approaches to Software Engineering (ENASE). pp. 27–37 (2017), https://doi.org/10.5220/0006276600270037
16. Carbonnel, J., Huchard, M., Nebut, C.: Analyzing Variability in Product Families through Canonical Feature Diagrams. In: 29th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE). pp. 185–190 (2017)
17. Cellier, P., Ducassé, M., Ferré, S., Ridoux, O.: Dellis: A data mining process for fault localization. In: 23rd Int. Conf. on Software Engineering and Knowledge Engineering (SEKE). pp. 432–437 (2009)
18. Czarnecki, K., Kim, C.H.P., Kalleberg, K.T.: Feature models are views on ontologies. In: 10th Int. Conf. on Software Product Lines (SPLC). pp. 41–51 (2006), https://doi.org/10.1109/SPLINE.2006.1691576
19. Czarnecki, K., Wasowski, A.: Feature Diagrams and Logics: There and Back Again. In: 11th Int. Conf. on Software Product Lines (SPLC). pp. 23–34 (2007)
20. Davey, B.A., Priestley, H.A.: Introduction to lattices and order. Cambridge University Press, Cambridge (1990), http://www.worldcat.org/search?qt=worldcat_org_all&q=0521367662
21. Eisenbarth, T., Koschke, R., Simon, D.: Locating features in source code. IEEE Trans. Softw. Eng. 29(3), 210–224 (2003)
22. Ganter, B., Wille, R.: Formal concept analysis - mathematical foundations. Springer (1999)
23. Greene, G.J., Esterhuizen, M., Fischer, B.: Visualizing and exploring software version control repositories using interactive tag clouds over formal concept lattices. Information & Software Technology 87, 223–241 (2017), https://doi.org/10.1016/j.infsof.2016.12.001

24. Heymans, P., Schobbens, P., Trigaux, J., Bontemps, Y., Matulevicius, R., Classen, A.: Evaluating formal properties of feature diagram languages. IET Software 2(3), 281–302 (2008), http://dx.doi.org/10.1049/iet-sen:20070055
25. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA): Feasibility Study. Technical Report CMU/SEI-90-TR-21 - ESD-90-TR-222 (1990)
26. Loesch, F., Ploedereder, E.: Restructuring Variability in Software Product Lines using Concept Analysis of Product Configurations. In: 11th Eur. Conf. on Soft. Maintenance and Reengineering (CSMR). pp. 159–170 (2007)
27. Mendonca, M., Branco, M., Cowan, D.: S.P.L.O.T.: Software Product Lines Online Tools. In: 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications. pp. 761–762. (OOPSLA), ACM (2009), http://doi.acm.org/10.1145/1639950.1640002
28. Moha, N., Hacene, A.R., Valtchev, P., Guéhéneuc, Y.: Refactorings of Design Defects Using Relational Concept Analysis. In: 6th Int. Conf. on Formal Concept Analysis (ICFCA). pp. 289–304 (2008), https://doi.org/10.1007/978-3-540-78137-0_21
29. Niu, N., Easterbrook, S.M.: Concept analysis for product line requirements. In: 8th Int. Conf. on Aspect-Oriented Software Development (AOSD). pp. 137–148 (2009)
30. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer Science & Business Media (2005)
31. Ryssel, U., Ploennigs, J., Kabitzsch, K.: Extraction of feature models from formal contexts. In: 15th Int. Conf. on Software Product Lines (SPLC) Workshop Proceedings (Vol. 2). p. 4 (2011)
32. Salman, H.E., Seriai, A., Dony, C.: Feature-to-code traceability in a collection of software variants: Combining formal concept analysis and information retrieval. In: 14th Conf. on Inf. Reuse and Integration (IRI). pp. 209–216 (2013)
33. Schobbens, P., Heymans, P., Trigaux, J., Bontemps, Y.: Generic semantics of feature diagrams. Computer Networks 51(2), 456–479 (2007), http://dx.doi.org/10.1016/j.comnet.2006.08.008
34. Segura, S., Benavides, D., Cortés, A.R., Trinidad, P.: Automated merging of feature models using graph transformations. In: Generative and Transformational Techniques in Software Engineering II, International Summer School (GTTSE 2007), Revised Papers. pp. 489–505 (2007), http://dx.doi.org/10.1007/978-3-540-88643-3_15
35. Shatnawi, A., Seriai, A.D., Sahraoui, H.: Recovering architectural variability of a family of product variants. In: 14th Int. Conf. on Soft. Reuse (ICSR). pp. 17–33 (2015)
36. Tilley, T., Cole, R., Becker, P., Eklund, P.: A survey of formal concept analysis support for software engineering activities. In: Formal Concept Analysis: Foundations and Applications, pp. 250–271. No. 3626 in LNCS–LNAI, Springer-Verlag (2005)
37. Xue, Y., Xing, Z., Jarzabek, S.: Feature location in a collection of product variants. In: 19th Working Conf. on Reverse Engineering (WCRE). pp. 145–154 (2012)
38. Yang, Y., Peng, X., Zhao, W.: Domain feature model recovery from multiple applications using data access semantics and formal concept analysis. In: 16th Working Conf. on Reverse Engineering (WCRE). pp. 215–224 (2009)