Analyzing Variability in Product Families through Canonical Feature Diagrams

Jessie Carbonnel[1], Marianne Huchard[1], and Clémentine Nebut[1]

[1]LIRMM, CNRS & University of Montpellier, France,
{jcarbonnel, huchard, nebut}@lirmm.fr

Abstract

Product line engineering aims to reduce the cost and effort to develop new related softwares, while increasing the software quality and the software scope. Variability analysis and modeling is a key issue in this approach. Several representations were proposed, including feature models (FMs) and product comparison matrices (PCMs). While PCMs are useful for presenting products in a tabular form, for their understanding and manipulation, it helps to switch to a graphical view. FMs are graphical views, but they are not canonical (i.e., several equivalent FMs can represent a same PCM) and user intervention is necessary to ensure the extraction of a meaningful FM from PCMs. In this paper, we investigate the benefits of a new structure, which captures variability in a canonical graphical representation. We outline its construction and we give insights about its shape and use when it is used as an alternative representation of wikipedia PCMs in the domain of software.

Keywords*: Product lines, Product Comparison Matrix, Feature Model, Formal Concept Analysis*

1. Introduction

Software product line engineering [8] is a software development approach which aims to produce families of similar software systems, while reducing the cost and effort, and increasing the software quality and scope. Variability analysis is a key issue in this approach that includes the organization of the main characteristics (called features) depending on the software products composing the product line. Several representations were proposed, the main one being feature models (FMs) [7]. An FM models the variability as a tree of features: a feature may introduce sub-features, and the feature is linked to a sub-feature with relation *mandatory* or *optional*, or to groups of sub-features with relation *at least one* or *exactly one* amongst the sub-features. Constraints can also be added to an FM. Other ways of modeling such as product comparison matrices (PCMs), decision models or logic formulas were also proposed, and all these representations have many variants and address specific issues. For example, FMs are appropriate representations to graphically show variability and derive product selection tools, logic formulas are relevant for automated reasoning tools and PCMs are widely used for enumerating existing representative products. While PCMs are relevant for presenting products in a tabular form, switching to a graphical point of view for their understanding and manipulation may be useful. FMs can be used in this task, but several equivalent FMs can represent a same PCM, thus their construction includes interpretation and modeling choices. Therefore, it is difficult to automate their generation from product description [1].

DOI reference number: 10.18293/SEKE2017-087

In this paper, we present a new structure, the Equivalence Class Feature Diagram (ECFD), which captures variability in a canonical graphical representation, and thus does not require any human interpretation to be built. The ECFD represents and structures a configuration set, and any FM depicting this configuration set conforms with the ECFD. It can be considered an interim representation between the catalog-like PCMs and the meaningful FMs. We outline the ECFD construction and we give insights about its shape and use when it is used as an alternative representation of wikipedia PCMs in the domain of software.

Next section presents the ECFD. We outline the ECFD construction in Section 3. Section 4 develops the wikipedia PCMs study. Related work is reported in Section 5. We conclude the paper with a few perspectives in Section 6.

2. Equivalence Class Feature Diagram (ECFD)

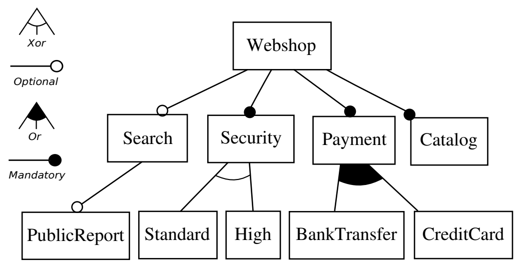
We introduce in this section Equivalence Class Feature Diagrams as an alternative way to represent variability, in a graphical and canonical way. The objective of the ECFD is to make explicit co-occurring features (features which are always present or absent together), exclusion constraints and other logical relations between features. In the following, we illustrate the ECFD with an example of e-shops configurations taken from an FM of SPLOT[1], shown in Figure 1. The configuration set is shown in Table 1 and the corresponding ECFD is given in Figure 2.

The main construct of an ECFD are boxes that represent a (maximal) set of features occurring always together. In the example of Figure 2, all the boxes are of size 1 except the one at the top, that is composed of four features. Indeed, those four features always occur together in the configuration set. There are several kinds of relations that can be added in an ECFD:

- The implications between boxes, denoted by a simple edge from a box b_1 to another box b_2, meaning that when the features of b_1 are present, the features of b_2 are also present in a configuration. In the example, we have such an implication between `PublicReport` and `Search`. Indeed in the configurations, when `PublicReport` is present, `Search` is also present.
- Or-groups can be denoted by grouped edges from several boxes b_i to another box b, to indicate that when features of b are present, there is at least the features of one box b_i. In the example, `BankTranfer` and `CreditCard` form an Or-group below the top box.
- Xor-groups are denoted like Or-groups, but with a cross on the grouped edges from several boxes b_i to another box b, to indicate that when features of b are present, there are exactly the features of one box b_i. In the example, `High` and `Standard` form a Xor-group below the top box.
- Mutex are denoted with a line linking boxes, with a cross on it. Two boxes in mutex relationship indicate that features of one box cannot appear at the same time as features of the other box. E.g., `High` and `PublicReport` are mutually exclusive.

A more detailed description of ECFD is presented in [3]. The ECFD is a canonical structure: for a given set of configurations, there is one and only one corresponding ECFD. On the contrary, the feature model is not canonical, several different feature models can represent the same set of configurations: e.g., the two feature models of Figures 1 and 3 represent the same set of configurations depicted in Table 1.

[1]http://www.splot-research.org/



$High \rightarrow\ !PublicReport\ ;\ High \rightarrow CreditCard$

Figure 1. FM1 for Tang-Eshops

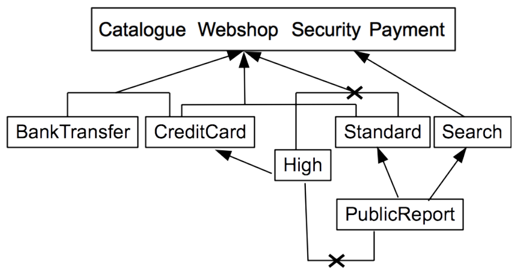
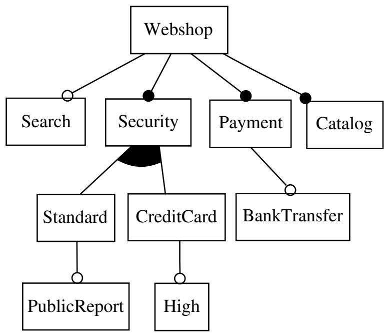


Figure 2. ECFD for Tang-Eshops

3. ECFD construction

The ECFD is built in two main steps: (1) AC-poset construction and (2) Xor-groups, Or-groups and Mutex computation. The AC-poset produces the feature equivalence classes (boxes/nodes of the ECFD) and the hierarchical structure between the nodes (Section 3.1). The Xor-groups, Or-groups and Mutex are then added to the structure (Section 3.2). This construction shares several steps with the feature model extraction proposed in [9]. However, our objectives partly diverge, since we want to preserve a canonical structure, while they aim to build one possible feature model amongst those that our canonical structure represents.



$High \rightarrow\ !PublicReport\ ;\ PublicReport \rightarrow Search$
$Payment \rightarrow BankTransfer \mid CreditCard\ ;$

Figure 3. FM2 for Tang-Eshops

3.1. AC-poset construction

The AC-poset is a structure which is defined in Formal Concept Analysis (FCA) as a sub-order of the concept lattice [5]. We briefly recall the needed definitions. Data of FCA are encoded in a *formal context* which is a triple $K = (O, A, R)$ where O and A are sets (objects and attributes, respectively) and R is a binary relation, i.e., $R \subseteq O \times A$. In our case, the objects are the configurations, and the attributes are the features (see Table 1). A *formal concept* C is a pair (E, I) composed of a maximal object set (E) and of the maximal set of attributes shared by these objects (I). $E = Extent(C) = \{o \in O | \forall a \in I, (o, a) \in R\}$ is the extent of the concept, $I = Intent(C) = \{a \in A | \forall o \in E, (o, a) \in R\}$ is the intent of the concept. The *concept lattice* is the concept set provided with a specialization order $\leqslant_s$ defined as follows: given two formal concepts $C_1 = (E_1, I_1)$ and $C_2 = (E_2, I_2)$ of K, $C_1 = (E_1, I_1) \leqslant_s C_2 = (E_2, I_2)$ if and only if $E_1 \subseteq E_2$ (and equivalently $I_2 \subseteq I_1$). C_1 is called a subconcept of C_2. C_2 is called a super-concept of C_1.

Table 1. Formal Context for Tang-Eshops

`TangEshop`	**BankTransfer**	**Catalogue**	**CreditCard**	**High**	**Payment**	**PublicReport**	**Search**	**Security**	**Standard**	**Webshop**
c0	×	×	×	×	×			×		×
c1		×	×		×	×	×	×	×	×
c2	×	×			×			×	×	×
c3		×	×		×		×	×	×	×
c4	×	×			×		×	×	×	×
c5	×	×	×		×		×	×	×	×
c6	×	×			×	×	×	×	×	×
c7		×	×		×			×	×	×
c8		×	×	×	×			×		×
c9	×	×	×	×	×		×	×		×
c10		×	×	×	×		×	×		×
c11	×	×	×		×			×	×	×
c12	×	×	×		×	×	×	×	×	×

Given a formal context $K = (O, A, R)$, an *attribute-concept* C is a concept such that some of the attributes of its intent are not present in any intent of any super-concept of C (these attributes are not inherited): $\exists a \in Intent(C), \forall C_{sup}, C \leqslant_s C_{sup}, a \notin Intent(C_{sup})$.

Let $\mathcal{AC}_K$ be the set of all *attribute-concepts* of a formal context K. This set of concepts provided with the specialization order $(\mathcal{AC}_K, \leqslant_s)$ is the AC-poset (for Attribute-Object-Concept poset) associated with K.

Algorithm 1 is a simple algorithm to build the Hasse diagram of the AC-poset. In this algorithm, we use complementary standard FCA notations: for any object set $S_o \subseteq O$, the set of shared attributes is $S'_o = \{a \in A | \forall o \in S_o, (o, a) \in R\}$, and for any attribute set $S_a \subseteq A$, the set of owners is $S'_a = \{o \in O | \forall a \in S_a, (o, a) \in R\}$. Figure 4 shows the AC-poset for the context of Table 1.

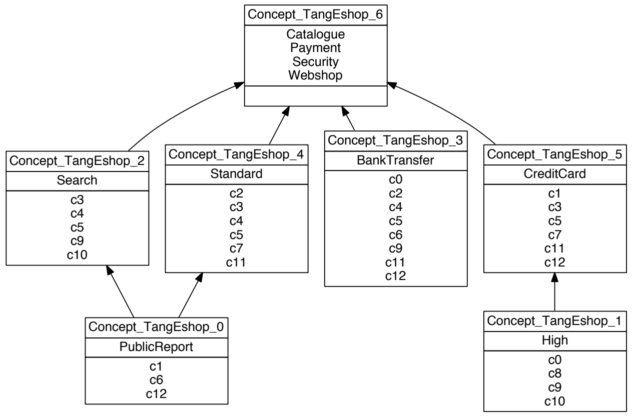


Figure 4. AC-poset for Tang-Eshops

Algorithm 1: ComputeACposet(K)

```
Data: $K$: a formal context
Result: $(AC_K, \leqslant_s)$: the AC-poset associated with $K$
/* attribute concepts */
1 $AC_K \leftarrow \varnothing$
2 foreach $a \in A$ do
3     $AC_K \leftarrow AC_K \cup (\{a\}', \{a\}'')$ //that are the objects that
      share the attribute $a$, with all the attributes they share
4 end
/* specialization order */
5 Compute the transitive reduction of $\leqslant_s$ by comparing the
  concept extents in $AC_K$
```

While a concept lattice may have $2^{min(|O|,|A|)}$ concepts, the number of concepts in an AC-poset is bounded by $|A|$. Reference [9] details the complexity of construction of the AC-poset (concept enumeration and transitive reduction computation) which is in $\mathcal{O}(|A|^2 \times |O|^2 + |A|^3)$.

The AC-poset (as the whole concept lattice) is a canonical structure *per se*, i.e., it is unique for a given formal context (set of configurations). However, the knowledge contained in the attribute-concept extents is not easy to capture, e.g., the exclusion constraints are present, but are not emphasized. This is why we propose the ECFD, which only keeps the intents of the attribute-concepts, and adds the additional information as Xor-groups, Or-groups and Mutex in a more intuitive way.

3.2. Groups and Mutex computation

Algorithm 2 is a schematic algorithm which specifies the computation of the Xor- (alternative) and the Or-groups with some similarities with the algorithm proposed by [9]. Let us define by construction the Xor- and Or-groups which we consider in Algorithm 2. We consider all the AC-poset antichains (sets of pairwise non comparable attribute-concepts) of growing size from 2 to $|\mathcal{AC}_K|$, which is the number of attribute-concepts (lines 2-3). For each candidate antichain A, we compute the set $Parents(A)$ containing the lowest attribute-concepts that are greater than all elements of A. Then for each $p \in Parents(A)$, we check if the configurations that have some $a \in A$ cover exactly the configurations that have p (line 5). If the configuration sets associated with each a form a partition (pairwise disjoint), A is a Xor-group with parent p of size $|A|$ (lines 7-8). If they do not form a partition, and A does not include a smaller Xor- or Or-group (lines 11-12), A is an Or-group with parent p of size $|A|$ (lines 13-14).

Algorithm 2: ComputeXorOrGroups(K)

```
   Data: Formal context $K$, AC-poset $(\mathcal{AC}_K, \leqslant_s)$
   Result: XorGroups, OrGroups
1  $XorGroups = \varnothing$; $OrGroups = \varnothing$;
2  foreach $i \in \{2..|\mathcal{AC}_K|\}$ do
3      foreach Antichain $A$ of size $i$ in $(\mathcal{AC}_K, \leqslant_s)$ do
4          foreach $p \in Parents(A)$ do
5              if $p' = \bigcup_{a \in A} a'$ then
6                  if $\forall a1, a2 \in A, a1' \cap a2' = \varnothing$ then
7                      $XorGroups += (A, p)$;
8                  end
9                  else
10                     if $\forall xor \in XorGroups, xor \not\subseteq A$,
11                     and $\forall or \in OrGroups, or \not\subseteq A$ then
12                         $OrGroups += (A, p)$;
13                     end
14                 end
15             end
16         end
17     end
18 end
```

In contrast with the proposal of [9], we do not apply heuristics to extract a tree structure and implication rules to conform with the FM formalism. This part of their work (which suits their objective) creates structures that are no more canonical. We keep the whole directed acyclic graph.

The last part of our approach takes another track and consists in building the Mutex (Algorithm 3), that will encode the missing knowledge about the exclusions between features. A Mutex is a set of mutually exclusive features, whose associated configurations do not cover the configurations of a specific feature (i.e. they have no parent). Mutex are extracted by growing size (line 2-3). A feature set is a Mutex if the features have no configuration in common, the set does not contain a Xor-group, is not included in a Xor-group (line 4) and, either its size is 2, or it does not contain any smaller Mutex (line 5).

Algorithm 2 is exponential, as explained in [9], but the

Algorithm 3: ComputeMutex(K)

```
   Data: Formal context $K$, AC-poset $(\mathcal{AC}_K, \leqslant_s)$
   Result: Mutex
1  $Mutex = \varnothing$;
2  foreach $i \in \{2..|\mathcal{AC}_K|\}$ do
3      foreach Antichain $A$ of size $i \in (\mathcal{AC}_K, \leqslant_s)$ s.t.
       $\bigcap_{a \in A} a' = \varnothing$ do
4          if $\forall xor \in XorGroups, (A \not\subseteq xor$ and $xor \not\subseteq A)$
5          and $(|A| = 2$ or $\forall m \in Mutex, m \not\subseteq A)$ then
6              $Mutex += A$;
7          end
8      end
9  end
```

authors also mention that input size is $|O| \times |A|$, and they argue that the computation can be reasonable in most cases. The same complexity arguments apply to Algorithm 3. Besides, they can be implemented efficiently (although still with exponential complexity) by avoiding generating useless antichains, namely, those that contain previously recognized groups of some categories (lines 11-12 in Algorithm 2, lines 4-5 in Algorithm 3). In Algorithm 2, this corresponds to the min-set-cover problem solving as mentioned in [9].

4. Study of PCMs variability structure

In this section, we show the application of our approach to a set of real PCMs and a few examples of the roles the ECFD can play for data summary, for FM extraction or for product selection.

Dataset description We selected 21 PCMs from wikipedia, that illustrate typical cases. These PCMs describe software products such as: wikis, data-base editors, media players, licenses or web browsers. They have been cleaned as described in [2] and transformed into Formal Contexts[2] with scaling operations [5]. Pages of wikipedia product descriptions are sometimes split in several PCMs. Some products even do not appear in all PCMs. In these cases, we kept this splitting, rather than concatenating the PCMs, because we think that it represents some domain knowledge.

ECFD computation and description The AC-posets are built with RCAexplore[3], and the groups with a specific Java program. Table 2 presents figures about the selected PCMs. Due to the scaling operations, a feature is a pair (property, value). The number of described products (`#conf`) varies between 9 and 90. The number of (property, value) pairs (`#feat`) varies between 5 and 67.

[2]Available with the corresponding AC-posets at: `www.lirmm.fr/recherche/equipes/marel/datasets/fca-and-pcm`

[3]http://dolques.free.fr/rcaexplore/

Table 2. Figures for TangEshop FM and for the selected 21 PCMs from our dataset. The PCMs are organized by size and topic. The number placed against the PCM name is its identifier in the dataset.

PCM name (id)	$\|\#conf.\| \times \|\#feat.\|$	ECFD Groups				ECFD structure	
		#xor (nb,size)*	**#or (nb,size)***	**#mutex (nb,size)***	**#mutex** of size 2	**#nodes**	**#multi-parents (nb,size)***
TangEshop	13 × 10	1 (1:2)	2 (2:2)	1 (1:2)	1	7	1 (1:2)
MathSoft (19)	9 × 8	0	1 (1:2)	0	0	5	1 (1:2)
Dbedition (5)	22 × 5	0	0	0	0	4	0
PublicLicense (27)	12 × 16	1 (1:3)	1 (1:3)	7 (6:2)(1:3)	6	10	2 (2:2)
Wiki (2)	40 × 7	0	1 (1:2)	0	0	7	1 (1:4)
Wiki (3)	38 × 6	0	4 (3:2)(1:3)	0	0	5	0
3Dsoft (7)	36 × 14	0	0	18 (18:2)	18	11	4 (3:2)(1:3)
Player (11)	33 × 9	0	0	13 (13:2)	13	9	0
Music (12)	25 × 8	0	0	0	0	8	2 (2:2)
Mail (39)	16 × 16	1 (1:2)	24 (13:2)(11:3)	7 (7:2)	7	14	5 (3:3)(1:4)(1:5)
PublicLicense (26)	51 × 6	0	0	0	0	6	0
Texteditor (17)	66 × 5	0	0	0	0	5	0
Web browser (21)	43 × 9	0	0	0	0	9	4 (1:4)
Download (15)	34 × 27	1 (2:1)	0	180 (171:2)(9:3)	171	24	9 (2:2)(7:3)
LinuxAdr (20)	34 × 16	0	0	5 (5:2)	5	13	6(2:3)(4:2)
Webbrowser (23)	36 × 12	0	0	2 (2:3)	0	12	2 (1:4)(1:5)
Webbrowser (22)	41 × 13	1 (1:2)	0	3 (3:2)	3	12	4 (1:4)
Prog. Languages (31)	56 × 25	0	0	212 (93:2) (88:3)(27:4)(4:5)	93	25	10 (4:2)(3:3) (2:4)(1:5)
Prog. Languages (32)	40 × 10	0	0	26 (8:2)(18:3)	8	10	1(1:3)
Prog. Languages (33)	90 × 10	0	0	17 (15:2)(2:3)	15	10	1 (1:4)
SocialNetJour (29)	11 × 67	70 (11:2)(1:3)(2:4)(6:5) (20:6)(21:7)(8:8)(1:9)	95 (1:4)(4:5)(43:6) (40:7)(7:8)	104 (96:2)(8:3)	96	25	11 (6:2)(5:3)
HtmlRendering (24)	15 × 61	15 (4:2)(1:3)(3:4) (4:5)(1:6)(1:7)(1:8)	3 (1:4)(2:5)	135 (135:2)	135	22	8 (8:2)

`#xor`, `#or` and `#mutex` are respectively the number of Xor-groups, Or-groups and Mutex. The number of Mutex of size 2 is isolated in one column to show the mutex that are commonly used in FM formalisms. `#nodes` is the number of ECFD nodes, and also the number of AC-poset nodes (and the number of feature equivalence classes). `#multi-parents` is the number of ECFD nodes that have more than one direct parent. `#xor`, `#or`, `#mutex` and `#multi-parents` are followed by a list of `(group-size:group-number)` pairs. For example, `24 (13:2)(11:3)`, in PCM Mail-39 `#or` groups means that in the corresponding ECFD, there are 24 Or-groups, including 13 groups of size 2, and 11 groups of size 3.

ECFD as a data summary The ECFD helps to identify the situations where many products have the same description and are grouped together (PCMs 2, 3, 23, 33); or inversely where product groups are smaller (PCM 15). The AC-poset shows feature sets that few products have (such as in PCM 7), revealing possibilities for increasing the product range, or balancing the product offer (if the products are developed together or belong to a same brand). A complex ECFD (PCMs 24, 29) should alert a designer to simplify, or refine the product offer. She/he also should check the cases where the ECFD is not complex, except for the number of Mutex (PCM 15, 31). We observe that beyond 20 (property, value) pairs, the number of Mutex becomes large (up to 212) which is specific to ECFD coming from PCMs (it does not appear for ECFD built upon the configuration set of an existing FM, as we studied in [3] in the context of FM composition).

ECFD for FM extraction We believe that the ECFD is a good candidate for FM extraction. When the multi-parent number is null or low (such as in PCMs 26, 17, 32, 33), the FM tree should be easy to extract. Furthermore, if there are no Mutex (such as in PCMs 26, 17), the FM will not include additional constraints and will be more intuitive. PCM 29, in the opposite case, has many multi-parents and many Mutex and will be difficult to transform into an FM. When the AC-poset has several roots (as this is the case for PCMs 24 or 29), they can be used to structurally decompose the description into several FMs, using also information about the semantics of the features, as recommended by methodologies like OVM [8]. Besides, the ECFD, which contains all the possible FMs, is a relevant structure to support a methodology which guides a designer through the FM extraction process. Mutex of size 2 can easily be used in a translation from the ECFD to an FM. In the studied PCMs, they cover the largest part of the Mutex set. During the FM extraction process, the FM should cover the products present in the PCM, but should not be too strict, thus it may accept products designed in (or suggested for) the future, and e.g., may not take into account the numerous Mutex.

ECFD for product selection Conceptual structures are good candidates for free datasets navigation, as shown in [11, 6]. In [11, 6], the conceptual structures contain both objects and properties, while the ECFD only contains the properties (here features). However, the ECFD still is a navigation structure, constrained by the groups, which allows to select products and, while doing this selection, be able to switch to close products, more easily than in a tree structure, by using the AC-poset relationships, especially in complex cases like PCMs 24, 29.

5. Related Work

To our knowledge, besides FM, there are two other graphical representations which can depict the variability of a set of product descriptions. A *Feature Graph* [10] is a diagram-like representation of several FMs describing the same configurations. It is composed of a directed acyclic graph representing implications between features, and a set of textual lists of feature-groups (*Or*, *Xor* and *Mutex*). The feature graph transitive closure/reduction is canonical, but it may describe more configurations than in the initial set of products. Another possible graphical representation is the *Binary Implication Graph* [4], representing all feature implications that can be extracted from a set of product descriptions. The potential feature-groups are not documented. As for the feature graph, only the transitive closure/reduction is unique, and some sets of products cannot be strictly represented with this formalism. By comparison, the ECFD is a canonical and full graphical representation of the scope of a PCM. We made a recapitulative table of the different graphical representations[4].

As already mentioned in the paper, Ryssel et al. [9] have also worked on extracting variability information (e.g., FM hierarchy, feature groups) from a set of product descriptions using FCA. However, while they focus on building one FM, we keep all variability information extracted from conceptual structures to represent, in the ECFD, all possible FMs depicting the initial products. Finally, Acher et al. propose an approach to extract FMs from PCMs based on FM merged operations [1]. Here again, they focus on building one consistent FM that best depicts the PCM, and not a unique structure preserving all knowledge about the PCM variability. However, if building an FM is the last step of a design, the ECFD can be considered as an interim step.

[4]`http://www.lirmm.fr/recherche/equipes/marel/datasets/variability-representation`

6. Conclusion

In this paper, we proposed the ECFD structure which gives a graphical and canonical view on variability. We applied it to software PCMs taken from wikipedia and showed its use for data analysis, product selection or as a guide for building FMs. The experimentation results are heterogeneous and show the necessity to deepen the analysis of PCMs.

As future work, we would like to evaluate our approach against other existing ones. We plan to apply our ECFD construction to PCMs of other domains and to define a methodology for guiding the designer from an ECFD to its alternative FM representation. This should include a decomposition step, where the ECFD will be relevant. We also want to study the situations where several PCMs are connected, like a PCM of multimedia softwares connected to a PCM of databases.

References

[1] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. In *6th Int. VaMoS*, pages 45–54, 2012.

[2] J. Carbonnel, M. Huchard, and A. Gutierrez. Variability representation in product lines using concept lattices: Feasibility study with descriptions from wikipedia's product comparison matrices. In *Int. Works. FCA&A @ ICFCA*, pages 93–108, 2015.

[3] J. Carbonnel, M. Huchard, A. Miralles, and C. Nebut. Feature Model composition assisted by Formal Concept Analysis. In *12th Int. Conf. ENASE*, pages 27–37, 2017.

[4] K. Czarnecki and A. Wasowski. Feature Diagrams and Logics: There and Back Again. In *11th Int. Conf. on Soft. Product Lines (SPLC)*, pages 23–34, 2007.

[5] B. Ganter and R. Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.

[6] G. J. Greene and B. Fischer. Single-focus broadening navigation in concept lattices. In *Proceedings of the 3rd Works. CDUD @ (CLA 2016)*, pages 32–43, 2016.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA): Feasibility Study. *Technical Report CMU/SEI-90-TR-21 - ESD-90-TR-222*, 1990.

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

[9] U. Ryssel, J. Ploennigs, and K. Kabitzsch. Extraction of feature models from formal contexts. In *15th (SPLC) Workshop Proceedings (Vol. 2)*, page 4, 2011.

[10] S. She, U. Ryssel, N. Andersen, A. Wasowski, and K. Czarnecki. Efficient synthesis of feature models. *Information & Software Technology*, 56(9):1122–1143, 2014.

[11] T. Wray and P. W. Eklund. Exploring the information space of cultural collections using formal concept analysis. In *9th Int. Conf. ICFCA*, pages 251–266, 2011.