On Extracting Feature Models From Product Descriptions

Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans[†], Charles Vanbeneden
PReCISE Research Centre
Faculty of Computer Science
University of Namur, Belgium
INRIA Lille-Nord Europe[†]
Université Lille 1 – LIFL – CNRS , France[†]
{mac, acl, gpe, phe}@info.fundp.ac.be

Philippe Collet, Philippe Lahire
University of Nice Sophia Antipolis
I3S Laboratory (CNRS UMR 6070)
France
{collet, lahire}@i3s.unice.fr

ABSTRACT

In product line engineering, domain analysis is the process of analyzing related products to identify their common and variable features. This process is generally carried out by experts on the basis of existing product descriptions, which are expressed in a more or less structured way. Modeling and reasoning about product descriptions are error-prone and time consuming tasks. Feature models (FMs) constitute popular means to specify product commonalities and variabilities in a compact way, and to provide automated support to the domain analysis process. This paper aims at easing the transition from product descriptions expressed in a tabular format to FMs accurately representing them. This process is parameterized through a dedicated language and high-level directives (e.g., products/features scoping). We guarantee that the resulting FM represents the set of legal feature combinations supported by the considered products and has a readable tree hierarchy together with variability information. We report on our experiments based on public data and characterize the properties of the derived FMs.

1. INTRODUCTION

In product line engineering, domain analysis consists of identifying commonalities and differences among the members of a family of products [17]. It generally follows *domain scoping* [15, 16], where product line information is gathered and organised from informal documentation. The explicit identification of common and variable assets helps in analyzing competing products of suppliers or direct competitors and is the starting point of the actual product line development. A popular way to reason about these assets is to describe them in terms of features, which are domain abstractions relevant to stakeholders and are typically increments in program functionality [6]. These features are organised hierarchically in *Feature Models* (FMs) [17]. The latter are now widely used to compactly define all features as well as their valid combinations. An FM basically consists of an AND-OR graph with propositional constraints.

Yet, the task of building FMs may prove very arduous for stakeholders, especially when they have to collect and model them from such inputs as unstructured product descriptions, tabular data or product matrices. Indeed, they then have to deal with a large amount of information on the domain and potential sub-domains [15, 16]. Furthermore, such descriptions do not help in identifying constraints among features, which are essential to undertake economical and technical decisions. When performed manually, this activity is both time-consuming and error-prone (see Section 2).

In this paper, we extract FMs from several tabular data files documenting a set of products along different perspectives. The procedure is semi-automated since fully automating the process is neither realistic nor desirable in practice: not all data (products and features) are relevant for a practitioner; some information can be explicitly interpreted in terms of variability (e.g., a feature of a product may be optional); the features can be hierarchically organised in different ways, etc. We therefore propose a dedicated language that can be used by a practitioner to parameterize the extraction process (see Section 3). The language supports scoping activities allowing to ignore some features or some products. It also enables practitioners to specify the interpretation of data in terms of variability and to set a feature hierarchy if needs be.

The second step of our approach is to synthesize an FM characterizing the valid combinations of features (configurations) supported by the set of products (see Section 4). Several FMs, representing the same set of configurations but according to different feature hierarchies, can be derived [8, 24]. We adapt previous techniques [10, 3, 1, 24]: we exploit the fact that product descriptions are similarly structured in order to determine, *a priori*, the adequate feature hierarchy. We describe a specific merging algorithm that first computes the feature hierarchy and then synthesizes the variability information (mandatory and optional features, Mutex-, Xor- and Or-groups, (bi-)implies and excludes constraints) using propositional logic techniques.

We validate our approach through experiments on various public data and characterize the properties of the extracted FMs (see Section 5). Our evaluation shows that, although many feature groups, implies and excludes constraints are recovered, a large amount of constraints is still needed to correctly represent the valid combinations of features sup-

ported by the products.

The contributions of this paper are three-fold: *i)* an extraction process parameterized by a dedicated language, *ii)* an automated procedure that synthesizes an FM based on product descriptions and *iii)* a report on experimenting the procedure and language on public data.

2. MOTIVATION

In this section, we analyze the difficulties of modeling variability from a product catalog, and we identify several related challenges.

2.1 Exploiting Product Descriptions

The domain analysis process analyzes software systems and documents to identify, classify and represent variability [17]. A significant portion of this domain knowledge is more and more available in digital form. For example, tabular data, such as spreadsheets, are widely used during many professional activities and are likely to contain a wealth of implicit knowledge such as *product descriptions*.

Let us consider, for instance, the domain of Wiki engines. The list of features supported/offered by a set of Wiki engines can be documented using a semi-structured specification. We use an excerpt of data from Wikimatrix[1], which provides information about different wiki engines. Figure 1(a) gives a visual representation of semi-structured data that provide a comparison of different wiki solutions. The tabular data can be specified, for instance, using comma-separated values or an XML format. Each row of the table documents a product, i.e., a Wiki engine, and the list of features it supports. For example, the *Confluence* wiki has a Commercial license that costs 10 USD and it supports RSS feeds, whereas the *MoinMoin* wiki has a GPL licence; *DokuWiki*, *PmWiki*, *DrupalWiki* and*MediaWiki* engines are all written in PHP, etc.

Based on such product descriptions, a practitioner might consider that the eight Wiki engines form a family of products and might want to build a *model* that represents the commonalities and variabilities of those eight products. This model should provide a compact representation of all valid combinations of features described by the product catalog of Fig. 1(a). This representation should allow for both structuring and reasoning.

A logical encoding of the set of products is usually produced to perform reasoning operations. For example, when a practitioner wants to determine if a certain combination of features, say f_1 and f_2, does lead to at least one product, off-the-shelf SAT solvers or BDD libraries can be used to perform reasoning operations in the logical space. Hence consistency checking can be applied on $\phi_{catalog} \wedge f_1 \wedge f_2$, where $\phi_{catalog}$ is the Boolean formula that represents valid combinations of catalog features. In addition, valid domains can be calculated to infer some variability choices.

Beyond the logical representation, a feature hierarchy and precise variability information are needed, especially when more complex management activities are performed. FMs constitute ideal candidate formalisms in this context, as they have been intensively used as a means to specify variability and pilot configuration. Formalization (e.g., [23]), automated reasoning operations [8] and tools (e.g., [19]) have been developed to support FMs management.

[1]see Section 5 for more details.

Deelstra et al. [11] notably report that product *configuration* is a time-consuming and expensive activity. They show that the lack of appropriate clustering of variation points are severe impediments to efficient product configuration. Hence, a proper hierarchy may be helpful to understand complex relationships between features. Similarly, the *maintenance* of an existing product line is a difficult activity and the extracted FM can serve as a starting point for further refactoring or evolving by practitioners [14, 26]. During the development of a software product line, the extracted FM can be associated to other artefacts, e.g., source code, models [6]. Finally, when stakeholders, from an internal division of a company or external suppliers, *communicate* about feature planning and reuse opportunities, it is preferable to rely on an explicit documentation of variability, that is what an FM provides.

Several approaches tackle the problem of extracting variability models of existing systems and products. Several kinds of artifacts can be considered, including legacy system documentation [15], requirements [28, 5, 27, 18, 12], source code [9] or the combination of system artefacts [24, 2]. Most of these approaches either focus on logical structures representing constraints and dependencies between features, or retrieve the feature hierarchy when those constraints are provided. To ease domain analysis, we believe that both are equally important. We therefore address the synthesis of a comprehensive FM in the remainder of the paper.

2.2 Feature Models

FMs hierarchically structure application features into multiple levels of increasing detail. When decomposing a feature into subfeatures, the subfeatures may be optional or mandatory or may form *Mutex*, *Xor*, or *Or* groups (see Figure 1(b) for a visual representation of an FM). The terms FM and *feature diagram* are employed in the literature, usually to denote the same concept. In this paper, we consider that a feature diagram (see Definition 1) includes a feature hierarchy (tree), a set of feature groups, as well as human readable constraints (bi-implies, implies, excludes).

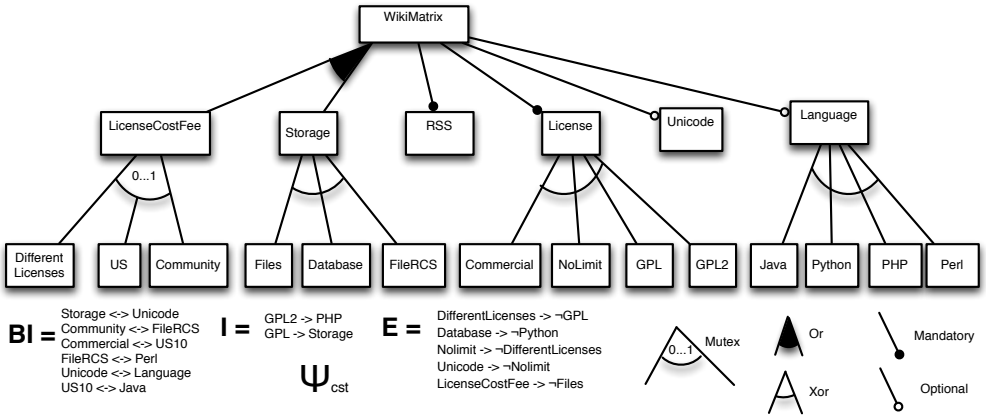
Definition 1 (Feature Diagram). *A feature diagram* $FD = \langle G, r, E_{MAND}, G_{MUTEX}, G_{XOR}, G_{OR}, BI, I, EX \rangle$ *is defined as follows:*

- $G = (\mathcal{F}, E, r)$ *is a rooted tree where* $\mathcal{F}$ *is a finite set of features,* $E \subseteq \mathcal{F} \times \mathcal{F}$ *is a finite set of edges and* $r \in \mathcal{F}$ *is the root feature ;*
- $E_{MAND} \subseteq E$ *is a set of edges that define mandatory features with their parents ;*
- $G_{MUTEX} \subseteq \mathcal{P}(\mathcal{F}) \times \mathcal{F}$, $G_{XOR} \subseteq \mathcal{P}(\mathcal{F}) \times \mathcal{F}$ *and* $G_{OR} \subseteq \mathcal{P}(\mathcal{F}) \times \mathcal{F}$ *define feature groups and are sets of pairs of child features together with their common parent feature ;*
- *a set of bi-implies constraints* BI *whose form is* $A \Leftrightarrow B$*, a set of implies constraints* I *whose form is* $A \Rightarrow B$*, a set of excludes constraints* EX *whose form is* $A \Rightarrow \neg B$ $(A \in \mathcal{F}$ *and* $B \in \mathcal{F})$*.*

Features that are neither mandatory features nor involved in a feature group are optional features. A parent feature can have several feature groups but a feature must belong to only one feature group. Similar to [24], we consider that an

Identifier	License	Language	Storage	LicenseCostFee	RSS	Unicode
Confluence	Commercial	Java	Database	US10	Yes	Yes
PBwiki	Nolimit	No	No	Yes	Yes	No
MoinMoin	GPL	Python	Files	No	Yes	Yes
DokuWiki	GPL2	PHP	Files	No	Yes	Yes
PmWiki	GPL2	PHP	Files	No	Yes	Yes
DrupalWiki	GPL2	PHP	Database	Different Licences	Yes	Yes
TWiki	GPL	Perl	FilesRCS	Community	Yes	Yes
MediaWiki	GPL	PHP	Database	No	Yes	Yes

(a) Tabular data for Wiki engines comparison



(b) Extracted Feature Model from Product Description of Figure 1(a)

Figure 1: Tabular data and extracted feature model for Wiki engines

FM is composed of a feature diagram *plus* a propositional formula (see Definition 2).

Definition 2 (Feature Model). *An FM is a tuple* $\langle FD, \psi_{cst} \rangle$ *where* FD *is a feature diagram and* ψ_{cst} *is a propositional formula over the set of features* $\mathcal{F}$

Not all combinations of features (*configurations*) are authorized by an FM. For example, Xor-groups require that at least one feature of the group is selected when the parent feature is selected, whereas in Mutex-groups no feature can be selected. An FM thus defines a set of *valid* (or *legal*) feature *configurations*.

Definition 3 (Configuration Semantics). *A configuration of an FM* g *is defined as a set of selected features.* $[\![g]\!]$ *denotes the set of valid configurations of* g*, that is a set of sets of features.*

For example, {WikiSPL, License, GPL, Language, Python, Storage, Files, RSS, Unicode } is a valid configuration of the FM shown in Figure 1(b). The set of configurations represented by an FM can be compactly described by a propositional formula defined over a set of Boolean variables, where each variable corresponds to a feature [10].

FM and Products. The relationship between a family of products and an FM can be summarized as follows: each valid configuration of an FM should correspond to at least one product of the family. In Figure 1(a), 8 products are documented but the corresponding FM represents 7 valid configurations. The reason is that products *DokuWiki* and *PmWiki* actually correspond to the same configuration {WikiSPL, License, GPL2, Language, PHP, Storage, Files, RSS, Unicode }.

Feature Diagram and Over-approximation. In Figure 1(b), the feature diagram (i.e., without ψ_{cst}) represents 11 configurations and is an *over-approximation* of the expected set of configurations. This means that set of valid configurations of the feature model is actually larger than the actual set of products. E.g., the configuration {WikiMatrix, License, GPL, Language, PHP, Storage, Files, RSS, Unicode } is valid w.r.t. the feature diagram but its does not correspond to any product. The reason is that the use of bi-implies, implies and excludes constraints is not expressively sufficient, thus justifying the presence of ψ_{cst} in Definition 2.

2.3 From Products to Feature Models

The main issue we identified is that manually creating an FM that contains all the variability of the set of products (e.g., described by Figure 1(a)) is a tedious and error-prone task. The feature hierarchy is not the only element of the FM to consider when accomplishing this task. Determining whether a feature is mandatory, optional or part of a group is far from trivial. In addition, complex constraints relating features of the feature hierarchy may occur and are hard to derive for a human. For example, the constraint "Java implies Database" is *deduced* from product specifications since there is no Wiki engine that uses Java and that supports another storage strategy than a database.

This becomes even more challenging when products are documented from different perspectives or *concerns*. For example, the specification of a Wiki engine includes such features as Datastorage, Hosting, Security, Development support,.... In this case, the user has to consider different, potentially heterogeneous, sources of information.

Facilitating the extraction of an FM from a product cata-

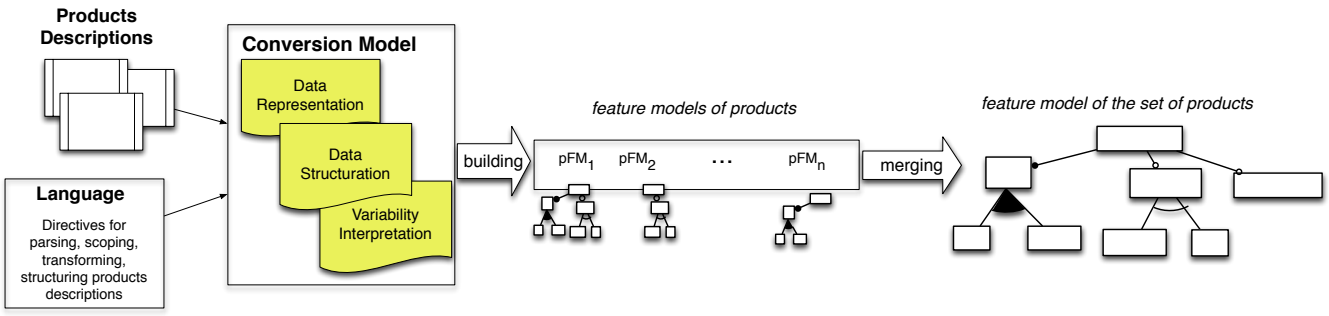


Figure 2: Modeling variability of products descriptions and synthesizing FMs

log is desirable but faces the three following main challenges.

Challenge 1: Automation of the extraction. To reduce both the manual effort and time needed to build the FM, the extraction should be as automated and scalable as possible.

Challenge 2: Putting the user in the extraction process. Although the procedure may be fully automated, the user should be able to play a role in the extraction. As it is the case in product line and domain engineering, a key issue is to provide *scoping* facilities, for example, to ignore some products or features in the resulting FM. In addition, the way data is interpreted in terms of variability should be easily overridden, if needs be, to better fit user intention.

Challenge 3: Quality of the extracted FMs. Finally, a required property of the extracted FM is that it accurately represents the valid combinations of features supported by the product set. This is not the only criterion since several, yet different, FMs can be produced while representing the same set of configurations. A meaningful feature hierarchy, the restitution of feature groups and (bi-)implies/excludes constraints, are needed from a modeler perspective.

3. MODELING VARIABILITY OF PRODUCT DESCRIPTIONS

In this paper, we consider that product descriptions are organized through semi-structured data, typically tabular data where each row of the table specifies a product. In their current form, such data are not semantically rich enough to directly translate as an FM. A key issue is the *transformation* of data in terms of feature hierarchy and variability.

We propose a generic *conversion metamodel* in order to normalize the input data and facilitate its interpretation. In particular, we reify the feature modeling concepts necessary for the conversion and we define transformation rules to build FMs from product descriptions. On top of the conversion metamodel, we provide a dedicated *language*, called VariCell, so that practitioners can specify high-level directives and parameterize the data transformation.

Figure 2 gives an overview of the extraction process. A set of product descriptions and directives expressed in VariCell are interpreted to build as much FMs as there are products. Finally, the FMs of the products are *merged*, producing a new FM that compactly represents valid combinations of features supported by the set of products.

Representing and Structuring Data.

We rely on the Comma Separated Values (CSV) format to represent product descriptions. We chose CSV since it is widely supported by business and scientific applications. Furthermore there exist several tools to export data (e.g., from spreadsheets or relational databases) to CSV and many of them are used to document catalogs of products. Using a CSV representation, we expect to apply our techniques to a large class of product descriptions. CSV is a basic format consisting of a text file containing lines and values. We naturally include CSV concepts in the conversion metamodel.

For sake of brevity, we do not detail the conversion metamodel. Its important concepts are specifically formatted hereafter. We consider that a ***ProductsDescription*** contains ***Line***s and ***Column***s. A ***Cell*** is the content placed at the intersection of a column and a line (e.g., Files, Yes, RSS). The first line of the ***ProductsDescription*** is the ***Head***. The cells of the head are named ***Label***s (e.g., License, Language). Each label must be unique in the conversion model. A ***Row*** is a line other than the head. A specific label is called ***Key*** (e.g., Identifier) to indicate that the column uniquely identifies each row in a table. These values are called ***Identifier***s (e.g., *Confluence*, *PBwiki*). The ***Value*** is the content of the cell for a given row and for a given non-identifier column (e.g., PHP, Database, Yes, No). In the general case, the different values within a product description correspond to ***Features*** of a ***Product*** (see below for more details). When describing a product, different perspectives, concerns, domains/sub-domains – called *views* hereafter – can be considered. As a result, the information about a product can be stored in several sources of data. To organize data, facilitate their manipulation and reuse, we associate a source of data to a ***View*** in the conversion metamodel.

Variability Interpretation.

Based on our observation on various data publicly available (see Section 5), we notice that some ***Value***s of a tabular data have more semantics than a simple textual value, especially in terms of variability (see Figure 3 for an example).

We identified five variability patterns[2] based on the values of a table:

mandatory meaning that the product necessarily exhibits the characteristic. Values like "Yes", "1", "True" are typically used;

[2]It should be noted that some values are empty or the uncertainty about the value is high (e.g., value "Partial"). In those cases, we consider that this is up to the user to precisely define the interpretation of the value in terms of variability.

optional meaning that the product may exhibit the characteristic. Values like "Opt", "Optional", "Plugin" are typically used;

dead feature meaning that the product does not support the characteristic. Values like "No", "0", "False" are typically used;

multi value a data value can be multi-valued (e.g., Windows; Linux). In terms of variability, it can be interpreted in various ways. For example, Windows and Linux can be considered either as mutually exclusive, both as optional features, etc.;

real value a textual value does not have a particular semantics other than the textual value itself.

We capture the five kinds of information and translate them in terms of features and variability. Algorithm 1 informally describes the different cases and steps to produce an FM of a product description.

Algorithm 1 *buildFM* $(S, pid, rootName)$

Require: a products description S, a product identifier pid and the name of the root feature $rootName$
Ensure: a feature model FM_{pid} representing the variability of the product

$r_{pid} \leftarrow rootName$
$\mathcal{F}_{pid} \leftarrow \mathcal{F}_{pid} \cup r_{pid}$
$cells \leftarrow extractCells(S, pid)$
for all $cell \in cells$ {we can ignore identifier column} **do**
  $labelName \leftarrow label(cell)$
  $value \leftarrow value(cell)$
  if $value$ is a dead value **then**
    {skip and continue.}
  end if
  $\mathcal{F}_{pid} \leftarrow \mathcal{F}_{pid} \cup labelName$
  $eLabel \leftarrow (labelName, r_{pid})$
  $E_{pid} \leftarrow E_{pid} \cup eLabel$
  if $value$ is a mandatory value **then**
    $E_{MAND_{pid}} \leftarrow E_{MAND_{pid}} \cup eLabel$
  else if $value$ is an optional value **then**
    {nothing to do, continue.}
  else if if $value$ is multi-valued **then**
    {interpret the value in terms of variability and set either G_{MUTEX}, G_{XOR}, G_{OR} or $E_{MAND_{pid}}$}
  else
    $E_{MAND_{pid}} \leftarrow E_{MAND_{pid}} \cup eLabel$
    $eValue \leftarrow (labelName, value)$
    $E_{pid} \leftarrow E_{pid} \cup eValue$
    $E_{MAND_{pid}} \leftarrow E_{MAND_{pid}} \cup eValue$
  end if
end for

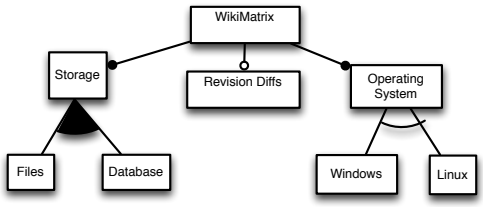
Let us consider the tabular data of Figure 3(a). The value Windows; Linux is multi-valued and is interpreted as two mutually exclusive features, thus forming a Xor-group. The value Files; Database is multi-valued and is translated as an Or-group. Revision Diffs is considered as an optional feature, due to the value "Plugin" in the cell. Finally, the feature Open is not included in the resulting FM (see Figure 3). In this example, we choose not to include the identifier "FooWiki" (it can be parameterized using our tool, see next section).

Modeling in Practice.

A raw transformation of data is likely to produce unuseful, unexploitable and/or unprecise FMs of product descriptions. Clearly, a user must be involved in the extraction process and should be able to specify directives to transform data

ID	Revision Diffs	Operating System	Storage	Open
...	...	...	...	...
FooWiki	Plugin	Windows; Linux	Files ; Database	No
...	...	...	...	...

(a) A wiki engine description



(b) Corresponding FM

Figure 3: Translating a product description into an FM

(*Challenge 2* of Section 2.3). It notably includes scoping and variability interpretation directives. VariCell, the language we developed, provides a practical solution to parse, scope, organize and transform product descriptions into an FM, possibly from different data sources. Users can programmatically parameterize the extraction process (see below for an example).

```
import: "general.csv" as general
        "datastorage.csv" as datastorage
        "hosting.csv" as hosting
        "security.csv" as security

name: "SPLWiki"

structure:
        security below hosting
        hosting below general
       // datastorage and general are inserted below SPLWiki
default:
        parsing:
                key: "brand" "range" // composite key
                variability:
                        optional: "Optional" "Opt" "Partial"
                        mandatory: "Yes" "true" "1"
                        dead: "No" "false" "0"
                separator: ","
                multivalued_separator: ";"
        // scoping (e.g., filter some products for *all* views)
        only_products: "Moinmoin" "DokuWiki" "PmWiki"

// scoping (e.g., features, for each view)
view: general
        rootname: "General"
        // override parsing instructions
        parsing:
                variability:
                        optional: "Plugin" "Optional" "Opt" "Partial"
        except_features: "Version" "Author"
        rewriting: "License Cost/ Fee" => "LicenseCost"
        multivalues:
                "Other Limits" => Alternatives
                "Intended Audience" => OR-Alternatives

view: security
        rootname: "Security"
        only_features: "Mail Encryption" "Blacklist"

view: datastorage
        structure:
                  "SQL" replace "MySQL" or "PostgreSQL" or "SQLite"
        multivalues:
                "Storage Quota" => Alternatives
```

The language provides some facilities for:

Parsing A practitioner can import several sources of data (i.e., CSV files) that are then associated to a view (see lines **1-4**). Some parsing instructions, specific to the CSV format, can be specified including the definition of the separator or multivalued_separator (see lines **19-20**). The parsing instructions can be specified for all views (default part, see lines **12-21**) or for a specific view ; Products/entities are identified by keys (which can be composite keys) and the practitioner can define, for example, which column of the CSV file the keys correspond to (see line **14**). We assume that the same key is used to identify a product in the different data sources. Therefore a key can be specified only in the default part and cannot be redefined in a view.

Scoping A practitioner may scope the data in various ways and for many purposes. For instance, not all products have to be integrated in the family of products since some products are considered as too basic in terms of supported features or not competitive enough. Another example is that not all features have to be considered, for example, the version number of the product may not be a relevant information. The language directives except_products and only_products are specified to (not) consider products for all imported views (see line **22**). Note that the directives except_features and only_features can be specified within a specific view (see line **31** and **39**).

Transforming Data It includes renaming or rewriting facilities (see line **32** or **43**) or more complex mapping[3] Importantly, the interpretation of data in terms of variability (e.g., to define when a feature is considered to be optional) can be specified either for all data sources and views (see lines **16-19**) or for a specific view (see lines **29-30**).

Specifying Structure Views are possibly related to each other (e.g., to describe a sub-domain) and this structuring information is usually not explicit in the format. By default, the FM of a view is inserted below the root feature (e.g., it is the case for the view *datastorage* and *general*). A practitioner may want to impose a specific hierarchy. For example, *hosting below general* means that the FM of the *hosting* view is inserted below the FM of the *general* view (see lines **8-10**). This information has an impact on the hierarchy of the resulting FM and is used by the extraction procedure (see next section).

4. AUTOMATED SYNTHESIS OF FEATURE MODELS

The execution of a VariCell specification should produce an FM representing the combinations of features actually supported by the products set.

4.1 Putting All Together

[3]Note that, more generally, data can be pre-treated using a more sophisticated *"extract, transform, and load"* process and fed to VariCell. We chose to integrate only an excerpt of these facilities into the language, considered as relevant to product line engineering and feature modeling.

Firstly, we need to combine the different descriptions of a given product. Steps 1 to 10 of Algorithm 2 produce an integrated FM for each product specification. It simply consists in *aggregating* the different FMs produced in each view.

Algorithm 2 $SynthesisFM\ (S_1, S_2, ..., S_m)$

Require: a set of products descriptions $pds = \{S_1, S_2, ..., S_m\}$ and the name of the root feature $rootName$
Ensure: a feature model G_{fm} representing the family of products
1: $mfms \leftarrow \emptyset$
2: {Let n be the number of products. We assume that each product is described in all products descriptions}
3: **for all** $pid \in 0...n$ **do**
4: $\quad pfms \leftarrow \emptyset$
5: $\quad$ **for all** $pd \in pds$ **do**
6: $\quad\quad pfms \leftarrow pFMs \cup\ buildFM\ (pid, pd, rootName))$
7: $\quad$ **end for**
8: $\quad pfm_{pid} \leftarrow aggregate\ (pfms)$
9: $\quad mfms \leftarrow mfms \cup pfm_{pid}$
10: **end for**
11: $G_{fm} \leftarrow merge\ (UNION, mfms)$

It should be noted that steps 1 to 10 of Algorithm 2 are an oversimplification of the reality, especially the aggregation part. Obviously, directives of VariCell (e.g., for scoping data, structuring the different views and imposing a hierarchy) are taken into account within the implementation.

Implementation.
To realize Algorithm 2, we need to perform various operations on FMs. We rely on FAMILIAR, a domain specific language for manipulating FMs [4]. For instance, the merging technique (see below for more details) is integrated in the language. The interpreter of VariCell generates FAMILIAR instructions for building FMs of a product (see step 6), for aggregating FMs (see step 8) or for merging FMs (see step 11). The FAMILIAR instructions are finally executed to synthesize G_{fm}.

4.2 Automated Merging of Feature Models

At the end of step 10 of Algorithm 2, we have an integrated FM for each product. We now need to *merge* the overlapping parts of the FMs to obtain a new FM representing the set of products. In previous work [3], we developed a merge operator that takes as input a set of FMs and produces as output an FM. The merge operator uses a name-based matching: two features match if and only if they have the same name. The properties of a merged FM are formalized in terms of the hierarchies and sets of configurations of input FMs. Here, we want to compute the merged FM that represents the *union* of the sets of configurations and preserve as much as possible the feature hierarchy of input FMs.

Our previous experience in the merging of FMs has shown that *syntactical* strategies have severe limitations to correctly restitute variability information and that a semantic-aware technique based on propositional logic is needed [3]. The same observation applies to the merging of products description. The key ideas of the proposed algorithm are to *i)* compute the feature hierarchy of the merged FM, *ii)* compute the propositional formula representing the set of configurations, and *iii)* to apply propositional logic reasoning techniques to construct a feature diagram based on the propositional formula and the hierarchy.

Hierarchy Computation.

In [1], we developed a generic technique that can deal with input FMs having different hierarchies. We formulated the problem of choosing a hierarchy from amongst a set of hierarchies as a minimum spanning tree problem (see details in [1]). In our specific context, though, we can observe that products description do have a similar hierarchy. Therefore, we can adapt the merging technique and remove the cost of choosing a hierarchy. It is straightforward to determine the hierarchy of the merged FM. Let be r_1, r_2, ..., r_n the roots and $G_1 = (\mathcal{F}_1, E_1)$, $G_2 = (\mathcal{F}_2, E_2)$, ..., $G_n = (\mathcal{F}_n, E_n)$ the hierarchies of FMs to be merged. The root of the merged FM is r_{Result} (equals to r_1, r_2, ..., and r_n). The hierarchy of the merged FM is $G_{Result} = (\mathcal{F}_{Result}, E_{Result})$ with $\mathcal{F}_{Result} = \mathcal{F}_1 \cup \mathcal{F}_2 \cup ... \cup \mathcal{F}_n$ and $E_{Result} = E_1 \cup E_2 \cup ... \cup E_n$.

Formula Computation.

The propositional formula representing the union of two sets of configurations represented by two FMs, FM_1, and FM_2, is computed as follows. First, FM_1 (resp. FM_2) FMs are encoded into a propositional formula ϕ_{FM_1} (resp. ϕ_{FM_2}). Then, the following formula is computed:

$$\begin{aligned}\phi_{Result} \quad &= (\phi_{FM_1} \ \wedge \ not\ (\mathcal{F}_{FM_2} \setminus \mathcal{F}_{FM_1})) \vee \\ &\quad (\phi_{FM_2} \ \wedge \ not\ (\mathcal{F}_{FM_1} \setminus \mathcal{F}_{FM_2})) \qquad (1)\end{aligned}$$

with $\mathcal{F}_{FM_1}$ (resp. $\mathcal{F}_{FM_2}$) the set of features of FM_1 (resp. FM_2) FM.

$\mathcal{F}_{FM_2} \setminus \mathcal{F}_{FM_1}$ denotes the complement or difference of $\mathcal{F}_{FM_2}$ with respect to $\mathcal{F}_{FM_1}$. *not* is a function that, given a non-empty set of features, returns the Boolean conjunction of all negated variables corresponding to features:

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

The presence of negated variables is required since we need to emulate the deselection of features that are in FM_1 (resp. FM_2) but not in FM_2 (resp. FM_1). Otherwise, two features, say $f \in \mathcal{F}_{FM_1}$ and $g \in \mathcal{F}_{FM_2}$ such that $f \neq g$, can be combined to form a configuration, thereby violating the semantics configuration of the merge in union mode. The computation of the formula is generalizable to more than two FMs.

From Hierarchy and Formula to Feature Diagram.

We reuse and adapt techniques presented in [10]. The authors propose an algorithm to construct a feature diagram from a Boolean formula. Propositional logics techniques are developed to detect features logically implied, *Mutex-*, *Xor-* and *Or-* groups. Furthermore the authors propose a generalized notation, roughly, a directed acyclic graph with additional nodes for feature groups. A major difference is that, in our work, we already *know* the resulting hierarchy, i.e., a tree in which a feature has only one parent. We thus exploit this information to streamline the algorithm. It proceeds as follows. We first compute G_{Result}, the hierarchy of the merged FM, as previously explained. At this step, all features, except root, are optional.

Mandatory and Feature Groups. We compute the implication graph, noted I_{Result}, of the formula ϕ_{Result} over $\mathcal{F}_{Result}$ the set of features of FM_{Result}. I_{Result} is a directed graph $G = (V, E)$ formally defined as:

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

I_{Result} is used to identify biimplications and thus set mandatory features together with their parents (i.e., setting $E_{MAND_{Result}}$). For feature groups, we reuse the method proposed in [10] and thus set $G_{MUTEX_{Result}}$, then $G_{XOR_{Result}}$ and finally $G_{OR_{Result}}$. It must be noted that a feature may be candidate to several feature groups (which is not allowed by our formalism). For example, a feature may be candidate to be part of three Or-groups. It is the case in Figure 1 where the feature LicenseCostFee can form an Or-group with the feature Storage, the feature Language, and the feature Unicode. It is difficult, in the general case, to determine whether a feature group must predominate over another feature group. A plausible strategy is to present users with all detected groups so that they can decide which to maintain in the feature diagram (as proposed in [24]). To reduce the amount of effort and time, we use an automated procedure that selects the larger candidate group (in case the candidate groups have the same size, we randomly select one). Nevertheless, we admit that a tradeoff between time, effort and reliability should be found.

Constraints. The set of biimplies and implies constraints, if any, can be deduced by removing edges of I_{Result} that are already expressed in the feature diagram (e.g., parent-child relations). Similarly, excludes constraints that were not chosen to be represented as a Mutex- or Xor-group are added. We incrementally add constraints (first biimplies, then implies, finally excludes).

From Feature Diagram to Feature Model.

The feature diagram, including the biimplies/ implies /excludes constraints, may still be an over-approximation of ϕ_{Result}. It is detected by checking the logical equality between ϕ_{Result} and $\phi_{diagram_{Result}}$, the encoding of the computed feature diagram as a propositional formula. The relative complement of ϕ_{Result} with respect to $\phi_{diagram_{Result}}$ corresponds to $\psi_{cst_{Result}}$ and can be computed using standard propositional logic techniques. $\psi_{cst_{Result}}$ can be restituted in different forms (e.g., conjunctive or disjunctive normal forms) but are usually not shown to the user, since many features can be related to each other in a complex manner.

BDD-based Implementation.

Currently, the handling of logical operations relies on Binary Decision Diagrams (BDDs). Computing the disjunction, conjunction and negation of BDDs can be performed in at most polynomial time with respect to the size of the BDD involved. As argued in [10], the cost of feature diagram construction is polynomial regarding the size of the BDD – the most expensive step being the computation of prime implicants to compute Or-groups. We reuse the heuristics developed in [20] (i.e., Pre-CL-MinSpan) to reduce the size of the BDD and that are known to scale up to 2000 features. Recently, She et al. proposed techniques to reverse engineering very large FMs (i.e., with more than 5000 features) using SAT solvers [24]. For this order of complexity, BDDs do not scale. Nevertheless, two challenges remain open for a SAT-based implementation of merging: *i)* the formula is in disjunctive normal form (see Equation 1) while SAT solvers require a conjunctive normal form ; *ii)* support for identifying Or-groups is currently missing.

5. PRELIMINARY EVALUATION

In Section 2.3, we have identified three challenges. *Challenge 1* is related to the automation of the synthesis procedure and its scalability. As previously described, our BDD-

based implementation can scale up to 2000 features. We did not encounter scalability issues for the data considered so far. Yet, scalability is a concern that will be addressed in future work. *Challenge 1* and *Challenge 2* are related to the effort and time spent by a practitioner using the proposed language. Qualitative properties of the language like learnability, productivity and expressiveness can be considered but have not yet been evaluated.

For the evaluation of the extraction techniques previously described, we focus on *Challenge 3*. We conduct experiments to characterize the properties of the extracted FMs.

Data Corpus.

Some existing works (see Section 6) consider products description to produce an FM. But only a few of these data are publicly available, except for Hartmann et al. [13]. For the experiments, we thus essentially use public data of products description: *i)* [13] describing an excerpt of a suppliers' offering, documenting variability of the products in a table (H); *ii)* catalog of products (laptops, bikes), stored in CSV files and available from SPLOT [19] (L, B, B_1, B_2) ; *iii)* a comparative table, including variability information, provided by an external person (C) ; *iv)* Wikimatrix website that provides access to information of a large set of wiki engines (WM_0, WM_1, WM_2,
WM_3). 154 wiki engines are documented. Each wiki engine describes several characteristics classified in 16 categories (General, Media and Files, Usability, etc.). We convert the XML file located in `http://www.wikimatrix.org/api/byproduct` into 16 CSV different files (one file per category).

For each data set, we write an VariCell script that includes some of the products described in the data and ignores some features, considered as not relevant. In all cases, we choose to not include identifiers (e.g., name of the product) in the resulting FM. Reagrding the bikes' catalog (B, B_1, B_2) and Wikimatrix (WM_1, WM_2, WM_3), we wrote three different scripts to change the number of configurations and features. Note that WM_0 is the example given in Figure 1(a). The data corpus, the VariCell scripts, the FAMILIAR scripts generated as well as the synthesized FMs are available online[4].

Measurements and results.

To characterize the properties of the synthesized FMs, we perform some measurements. In Table 4, we report on respectively the number of features, the number of leaves in the feature hierarchy, the number of mandatory features, the number of Mutex-, Xor- and Or-groups, the number of biimplies, implies and excludes constraints. We also measure $|[\![FD]\!]|$ and $|[\![FM]\!]|$ to compare the number of configurations characterized by the feature diagram (see Definition 1) with the number of configurations characterized by the merged propositional formula. As an example, the feature diagram synthesized in Figure 1(b) represents 11 configurations while there are actually 7 configurations expressed by the products set of WM_0.

On results given in Table 4, we can first observe that feature groups, obtained by our procedures, are present in all cases. The larger proportion is composed of Mutex- and Xor-groups, the presence of Or-groups being less important. Second, a large number of constraints is present and the amount can be very important. As an extreme case, we report on 53 biimplies, 512 implies and 325 excludes (see $B2$). This number tends to significantly increase with the number of configurations and features (see B, B_1 and B_2 or WM_1, WM_2 and WM_3). Third, the configurations characterized by the synthesized feature diagram is always an over-approximation of the actual configurations of the products set (the only exception being B). This over-approximation may be very important (e.g., see WM_3).

Discussions.

We now discuss the results of our preliminary evaluation.

Over-approximation. We recall that an FM is composed of a feature diagram *plus* cross-tree constraints (see Definition 2). Reasoning operations, like configuration assistance, are obviously not performed using the feature diagram but the FM (in our case, using the merged propositional formula). The over-approximation is thus transparent for the user and not an issue *when reasoning* about FMs. Nevertheless, it questions the use of a feature diagram: what could be the impact of this over-approximation for the user in terms of *maintainability* or *understandability*?

Key factors when extracting FMs. Both data and directives specified by the practitioner have an impact on the properties of the synthesized FMs. For instance, directives for enforcing the structure of the FM (as permitted by the language) may reduce the number of biimplies/implies/excludes constraints or produce new feature groups. Further experiments are needed in this direction. The properties of the products descriptions should be considered as well, including the quality of the documentation, the relevance of considering some features or grouping together products.

Comparison with existing FMs. We can make the assumption that FMs produced in our context significantly differ from those encountered in the literature (e.g., those available in [19]). It is notably the case for feature hierarchy and constraints. We plan to use or adapt existing metrics [8, 7] to further characterize the properties of such FMs. Furthermore, this new class of FMs may be studied and used as a benchmark for reasoning techniques already developed in the literature.

Threats to Validity. Threats to external validity are conditions that limit our ability to generalize the results of our experiment to real practice (e.g., in the industry). An important concern is whether public data selected are representative of realistic usage. An internal threat concerns the correctness of the implementation. By construction, the correctness of the implementation depends on the correctness of FAMILIAR and VariCell interpreters. In particular, the merge operator must guarantee that some semantic properties are preserved. Our implementation is currently checked by a comprehensive set of unit tests, complemented by cross-checked testing with other operations provided by FAMILIAR. VariCell interpreter comes with unit tests as well. We also manually verified a large number of examples. Another internal threat is that we cannot guarantee that the extraction process does not depend on certain shapes of products description.

6. RELATED WORK

We now review some of previous works tackling the problem of extracting variability models from existing systems and products. Weston et al. [27] proposed a tool which creates FMs from requirements specifications. Clustering methods identify features and then a vocabulary lexicon and grammatical pattern matching identify feature variants, like

[4] `https://nyx.unice.fr/projects/familiar/wiki/VariCell`

Data	$\|\mathcal{F}\|$	$\|leaves\|$	$\|E_{MAND}\|$	$\|G_{MUTEX}\|$	$\|G_{XOR}\|$	$\|G_{OR}\|$	$\|BI\|$	$\|I\|$	$\|EX\|$	$\|[\![FD]\!]\|$	$\|[\![FM]\!]\|$
WM_0	22	16	4	1	3	1	6	2	5	11.0	7.0
H	9	7	2	1	0	0	0	1	3	52.0	40.0
L	113	99	14	0	12	0	37	192	91	57.0	33.0
B	28	22	6	0	4	0	5	18	10	26.0	26.0
B_1	61	54	7	0	5	0	6	98	63	89.0	79.0
B_2	190	179	7	4	5	0	53	512	325	964.0	263.0
C	45	39	4	1	3	1	1	45	63	899.0	535.0
WM_1	38	26	3	7	0	0	11	59	51	1570.0	999.0
WM_2	59	43	3	11	0	3	22	108	101	19449.0	11956.0
WM_3	92	79	3	10	0	0	21	242	676	11511.0	1544.0

Figure 4: Experimental results: properties of the synthesized FMs

alternative enumerations or conjunctive enumerations. In our work, product descriptions are much more structured, easing the identification of features. Another difference is that the modeler has to incorporate the new variability information manually into the resulting FM whereas, in our case, the variability information is automatically synthesized.

She et al. proposed a reverse engineering approach combining two distinct sources of information: textual feature descriptions and feature dependencies [24]. Our approach also benefits from the combination of two sources of information, namely the documentation about products' characteristics and the products' dependencies (encoded as a propositional formula, see Section 4.2). She et al. mostly focus on the retrieval of the feature hierarchy (heuristics to identify the most likely parent features of each feature). In our case, we exploit structurally similar product descriptions and/or practitioner instructions to derive a feature hierarchy.

Lora-Michiels et al. investigate the use of mining techniques such as the Apriori algorithm and independence tests in order to automate the construction of a product line model, starting from a collection of product models [18]. Based on those techniques, the method guides the identification of candidate features, group cardinalities, as well as excludes and requires constraints. In our context, we use propositional logic techniques to synthesize an FM. We guarantee that the resulting FM compactly represents the set of legal feature combinations supported by the considered products and has a readable tree hierarchy together with variability information, including feature groups and constraints. Dumitru et al. mine raw feature description and uses clustering to identify features within a large corpus of descriptions [12]. The method can be easily adapted to a variety of situations since they do not require any specific formatting. Their approach retrieve association rules that are then used to support recommendations during feature selection. Alves et al. investigate the suitability of information retrieval techniques for identifying commonalities and variabilities in requirement specifications [5]. They also propose a systematic framework to abstract existing requirements specifications into an FM. The latter approaches allow to identify individual features as well as dependencies between them, but they do not consider extracting the feature hierarchy as we do. Snelting applied formal concept analysis (FCA) to extract concept lattices from C's preprocessor directives like #ifdef statements [25]. Due to the formalism of FMs, we have to derive a single tree rather than visualizing entire lattice. We also detect feature groups, (bi-)implies and excludes constraints. Ryssel et al. develop optimized methods based on FCA and analyze incidence matrices containing matching relations [21]. In their context, there are neither preexisting FMs nor propositional formulas.

In [13], the authors mention the need for integrating FMs that come from different suppliers. The competing products offered by suppliers are either documented as FMs or in tabular data. Our work can thus be applied in this context. Furthermore, the authors proposed a reference FM that is related to suppliers' FMs through constraints. In contrast, we propose to merge similar features which allows to reduce the number of features in the resulting FM.

In [9], probabilistic FMs (PFMs) are introduced to model and reason about preferences among the legal configurations. In this paper, we consider FMs in their basic, propositional forms. The use of an extended formalism, such as PFM or support for feature attributes, can be considered in the context of our work.

Domain scoping is an important activity in product line engineering. The techniques and tools proposed in this paper can be applied to support a scoping activity. On the other hand our work can benefit from existing scoping approaches and methodologies (see, e.g., [22, 16]).

7. CONCLUSION AND FUTURE WORK

In domain analysis and product line engineering, the manual construction of feature models to represent the variability of a family of products is neither desirable nor feasible. Several sources of information can be considered and treated by a (semi-)automatic procedure to extract a feature model. In this paper, we considered products specification, typically organized in tabular data. We proposed a semi-automated process, language and tool supported, to extract variability of the family of products. We developed an automated technique to synthesize a feature model based on the merging of a set of products description. The synthesized feature model represents valid combinations of features supported by the set of products and comes with a readable hierarchy and proper identification of feature groups. We reported on preliminary experiments on public data.

Throughout the paper, we identified open issues and discussed their possible impact on the presented work. First, regarding the automated merging of feature models, an effective solution for selecting overlapping groups and a SAT-based implementation need further research. Second, we plan to evaluate the extraction process including the tool support and the language. Third, we hope to apply our techniques on a larger scale, on different data, in different domains and contexts (e.g., industrial context). Fourth, syn-

ergies with other areas (database engineering, data mining, probabilistic feature models) will be considered.

Acknowledgments

We thank Clément Quinton for his helpful discussions.This work was partially funded by the Walloon Region under the NAPLES project, the IAP Belgian State, Belgian Science Policy under the MoVES project, the BNB, the FNRS, as well as the the french ANR SALTY project[5] under contract ANR-09-SEGI-012.

8. REFERENCES

[1] Mathieu Acher. *Managing Multiple Feature Models: Foundations, Language and Applications*. PhD thesis, University of Nice Sophia Antipolis, 2011.

[2] Mathieu Acher, Anthony Cleve, Philippe Collet, Philippe Merle, Laurence Duchien, and Philippe Lahire. Reverse Engineering Architectural Feature Models. In *Proc. of ECSA'11*, volume 6903 of *LNCS*, page 16. Springer, 2011.

[3] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. Comparing Approaches to Implement Feature Model Composition. In *Proc. of ECMFA'10*, volume 6138 of *LNCS*, pages 3–19. Springer, 2010.

[4] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. A Domain-Specific Language for Managing Feature Models. In *Proc. of SAC'11*, . ACM, 2011.

[5] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. An exploratory study of information retrieval techniques in domain analysis. In *SPLC'08*, pages 67–76. IEEE, 2008.

[6] S. Apel and C. Kästner. An overview of feature-oriented software development. *Journal of Object Technology (JOT)*, 8(5):49–84, July/August 2009.

[7] Ebrahim Bagheri and Dragan Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 19(3):579–612, 2011.

[8] D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated Analysis of Feature Models 20 years Later: a Literature Review. *Information Systems*, 2010.

[9] K. Czarnecki, S. She, and A. Wąsowski. Sample spaces and feature models: There and back again. In *SPLC'08*, pages 22–31, 2008.

[10] K. Czarnecki and A. Wąsowski. Feature diagrams and logics: There and back again. In *SPLC'07*, pages 23–34, 2007.

[11] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Product derivation in software product families: a case study. *Journal of Systems and Software*, 74(2):173–194, 2005.

[12] Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *Proc. of ICSE'11*, pages 181–190. ACM, 2011.

[5] `https://salty.unice.fr`

[13] Herman Hartmann, Tim Trew, and Aart Matsinger. Supplier independent feature modelling. In *SPLC'09*, pages 191–200. IEEE, 2009.

[14] Mikolás Janota, Victoria Kuzina, and Andrzej Wąsowski. Model construction with external constraints: An interactive journey from semantics to syntax. In *Proc. of MoDELS'08*, volume 5301 of *LNCS*, pages 431–445. Springer, 2008.

[15] I. John. Capturing product line information from legacy user documentation. In *Software Product Lines*, pages 127–159. Springer, 2006.

[16] Isabel John and Michael Eisenbarth. A decade of scoping: a survey. In *Proc. of SPLC'2009*, volume 446 of *ICPS*, pages 31–40. ACM, 2009.

[17] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA). Technical Report CMU/SEI-90-TR-21, SEI, November 1990.

[18] Alberto Lora-Michiels, Camille Salinesi, and Raúl Mazo. A method based on association rules to construct product line models. In *Proc. of VAMoS'2010*, volume 37 of *ICB-Research Report*, pages 147–150, 2010.

[19] Marcilio Mendonca, Moises Branco, and Donald Cowan. S.P.L.O.T.: software product lines online tools. In *Proc. of OOPSLA'09*, pages 761–762. ACM, 2009.

[20] Marcilio Mendonca, Andrzej Wąsowski, Krzysztof Czarnecki, and Donald Cowan. Efficient compilation techniques for large scale feature models. In *GPCE'08*, pages 13–22. ACM, 2008.

[21] Uwe Ryssel, Joern Ploennigs, and Klaus Kabitzsch. Extraction of feature models from formal contexts. In Ina Schaefer, Isabel John, and Klaus Schmid, editors, *SPLC Workshops*, page 4. ACM, 2011.

[22] Klaus Schmid. A comprehensive product line scoping approach and its validation. In *Proc. of ICSE'02*, pages 593–603. ACM, 2002.

[23] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, 2007.

[24] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki. Reverse engineering feature models. In *ICSE'11*. ACM, 2011.

[25] Gregor Snelting. Reengineering of configurations based on mathematical concept analysis. *ACM Trans. Softw. Eng. Methodol.*, 5:146–189, April 1996.

[26] Thomas Thüm, Don Batory, and Christian Kästner. Reasoning about edits to feature models. In *ICSE'09*, pages 254–264. IEEE, 2009.

[27] N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. In *SPLC'09*, volume 446 of *ICPS*, pages 211–220. ACM, 2009.

[28] Wei Zhang, Hong Mei, and Haiyan Zhao. A feature-oriented approach to modeling requirements dependencies. In *Proc. of RE'05*, pages 273–284. IEEE Computer Society, 2005.