

An Ontological Rule-Based Approach for Analyzing Dead and False Optional Features in Feature Models

Luisa Rincón, Gloria Lucia Giraldo, Raúl Mazo, Camille Salinesi

▶ **To cite this version:**

Luisa Rincón, Gloria Lucia Giraldo, Raúl Mazo, Camille Salinesi. An Ontological Rule-Based Approach for Analyzing Dead and False Optional Features in Feature Models. XXXIX Latin American Computing Conference (CLEI), Oct 2013, Naiguatá, Venezuela. hal-00913944

HAL Id: hal-00913944

https://paris1.hal.science/hal-00913944

Submitted on 4 Dec 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

An Ontological Rule-Based Approach for Analyzing Dead and False Optional Features in Feature Models

L. Rincón*, G. Giraldo**, R. Mazo*** and C. Salinesi***
* Universidad Nacional de Colombia, Medellín, Colombia
** Departamento de Ciencias de la Computación y de la Decisión, Universidad Nacional de Colombia, Medellín, Colombia
***Centre de Recherche en Informatique CRI, Université Panthéon Sorbonne, Paris, France
{lufrinconpe, glgiraldog}@unal.edu.co
{raul.mazo, camille.salinesi}@univ-paris1.fr

***Abstract*- Product lines engineering uses Feature Models (FMs) as a notation to represent variability and commonality in families of products. One of the well-known issues of FMs is that they may have defects that can drastically diminish the benefits of the product line approach. Two of these defects are dead features and false optional features. Dead features are features absent from any valid product of the product line. False optional features are features declared as optional but actually required in all valid products. These two types of defects are undesirable in FMs because they give a wrong idea of domain that represents the FM. Several techniques documented in literature help to identify dead and false optional features. However, only few of them tackle the problem of identifying the causes of these defects. Besides, the explanations they provide are cumbersome and hard to understand by humans. In this paper, we propose an ontological rule-based approach to (i) identify dead and false optional features in FMs; (ii) identify certain causes of these defects; and (iii) explain these causes in natural language. Moreover, we propose a collection of rules that (i) formalize some cases that produce dead and false optional features; (ii) find the FM's elements that causes each defect; and (iii) explain why a feature is dead or false optional. This collection of rules helps modelers to correct the defects found in FMs and helps prevent the occurrence of new ones. We illustrate our approach in a reference model from literature. A preliminary empirical evaluation of our approach, using a benchmark composed of 31 FMs of sizes up to 150 features, shows that the proposal is effective, accurate and scalable.**

Keywords

Feature Models, Defects, Ontologies, Software Engineering

I. INTRODUCTION

A Software Product Line (SPL) is a family of related software systems with common and variable functions whose first objective is reusability [1]. Extensive research and industrial experience have widely proven the significant benefits of Software Product Line Engineering (SPLE) practices. Among them are: reduced time to market, increased asset reuse and increased software quality [2]. SPLE usually uses Product Line Models (PLMs) to represent the correct combination of features that represent valid products.

Feature Models (FMs) are a common language to represent PLMs in order to describe the features and their dependencies for creating valid products [3]. FMs have also proven useful to communicate effectively with customers and other stakeholders such as marketing representatives, managers, production engineers, system architects, etc. Consequently, having FMs that correctly represent the domain of the product line is of paramount importance to the success with the SPLE production approach.

However, creating models with features that correctly represent the domain described by the model is not trivial [4]. In fact, when a FM is constructed, defects may be unintentionally introduced. Dead and false optional features are two types of defects directly related to the semantic of FMs. A feature is dead if it cannot appear in any product of the product line [3]. A feature is false optional if it is declared as optional, but it appears in all products of the product line [5]. Due to the ability of FMs to derive a potentially large number of products, any defect in a FM will inevitably affect many products of the product line [6].

Numerous researches focus on identifying dead and false optional features in FMs [3], [5], [7–10]. Others approaches focus on identifying dead and false optional features and identifying the causes that produce these defects [11], [12]. Some others works propose even using ontologies to represent FMs [13–15] and others propose using ontologies for identifying defects in FMs [16–19]. However, few researchers have addressed the problem of identifying the causes that produce these defects and explain them in a human understandable language. This means that once defects are found it is necessary to manually inspect models to look for why the defects occurred. Once engineers know why defects occurred, they can try to fix them. Our observation is that this is a cumbersome task. Indeed, looking for the causes of defects is about as complicated as looking for defects themselves even when the defect is already known. Therefore, we believe that it is of paramount importance to solve this key problem if we really want FMs verification methods to be effective in an industry context.

Our general goal is to find a generic technique that will point out the cause of various kinds of defects on product line models specified with different notations. In this paper, we propose a first step to achieve this goal. In particular, we propose an ontological rule-based approach to analyze dead and false optional features in FMs, that is: identify features of a FM that are dead or false optional, identify the causes of these defects, and explain each cause in natural language.

We hope this information helps product line engineers to avoid same mistakes in future work, and to understand why dead and false optional features occur [12], [20].

Our original contribution can be summarized as follows:

1. We propose a framework that (i) identifies dead and false optional features in FMs; (ii) identifies the causes of these defects; and (iii) creates explanations in natural language about each detected cause.
2. We construct a *Feature Model Ontology* and we formalize—using first-order logic—six rules for identifying dead features and three rules for identifying false optional features. Each rule defines a case in which a feature is dead or false optional. In that way, we know the causes that origin each defect, and we build the corresponding explanation. We defined these rules based on our experience and on the rules found in literature [9].
3. We developed an automated tool to implement our approach. The results of our validation show that our approach is effective and scalable until FMs of 150 features.

The remainder of the paper is as follows. Section II gives a brief overview of the necessary concepts for understanding the framework presented in Section III. Section IV presents the implementation details. Section V presents the evaluation of the precision, scalability and usability of our approach. Section VI presents related research. Finally, Section VII presents the conclusions and suggests future research directions.

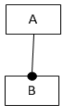
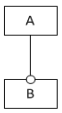
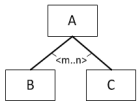
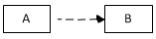
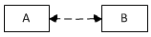
II. General Concepts

A. Features Models

Feature modeling is a notation proposed in [3] as part of their method for performing a domain analysis of possible products of a target domain. In the SPLE, feature-oriented domain engineers use Feature Models (FMs) to represent commonality and variability of a target domain. FMs show how the domain features are related and show some trade-off decisions that must be made for creating a valid product in the domain of interest [3].

Under this notation, a feature is a distinctive element that directly affects final users. Each feature is a node in a tree structure, and the model dependencies are directed arcs. The tree structure represents hierarchical organization of the features. The tree's root of the FM represents whole product line and therefore it is part of all valid products of the product line. Each feature represented by a non-root node can be associated with a product only if the feature represented by the father node is associated with the product too. The elements of the feature notation that we use in this paper are presented in Table 1. An example of FM using this notation is presented in Figure 1.

TABLE I. Types of dependencies in FMs

Notation	Type of Dependency
	Mandatory [3] Child feature B should be included in all valid products containing the parent feature A and vice versa. It a feature is mandatory and all its ancestors are also mandatory, then, this feature is a full mandatory feature [21].
	Optional [3] Child feature B may or may not be included in valid products containing parent feature A. However, if feature B is included in a product, its father A should be included too.
	Group cardinality [22] Represents the minimum (m) and the maximum (n) number of child features (B...C) grouped in a cardinality (<m..n>) that a product can have when the father feature (A) is included in the product. If at least one of the child features is included into a product, the father feature should be included too.
	Requires [3] Feature B should be included in valid products with feature A. This dependency is unidirectional.
	Excludes [3] Features A and B cannot be in valid product at same time. This dependency is bidirectional.

This paper uses an adapted version the Graph Product-Line (GPL) [23] as running example. The resulting model is presented in Figure 1. We used this example because it is well-known in the product line community, and it was proposed to be a standard case for evaluating product line methodologies [23]. In order to illustrate our approach, we intentionally introduced 8 dead features (cf., AF2, AF7, AF11, AF12, AF13, AF14, AF15, Connected) and 3 false optional features (cf., AF1, AF9, AF10) into the original model. We used 15 artificial features and 25 dependencies to produce these defects. All features and dependencies have a name for easier identification. We identified artificial features with a capital AF, and artificial dependencies with a capital AD. In addition, we identified original features of the model with their names, and we used a capital OD to build the name of original dependencies.

The members of the GPL are graphs either *Directed* or *Undirected*, their edges are *Weighted* or *Unweighted*, and their search algorithms are breadth-first search (*BFS*) or depth-first search (*DFS*). All products of this FM implement one or more of the following search algorithms: Vertex Numbering (*Number*), Connected Components (*Connected*), Strongly Connected Components (*StronglyCon*), Cycle Checking (*Cycle*), Minimum Spanning Tree (*MST*) and Single-Source Shortest Path (*Shortest*). Moreover, this FM has dependencies that limit the valid combination of features previously described. For instance, the *MST* algorithm requires *Undirected* graphs (cf., OD21) and *Weighted* edges (cf., OD20), and *StronglyCon* algorithm requires *Directed* graphs (cf., OD4) and the *DFS* search algorithm (cf., OD17).

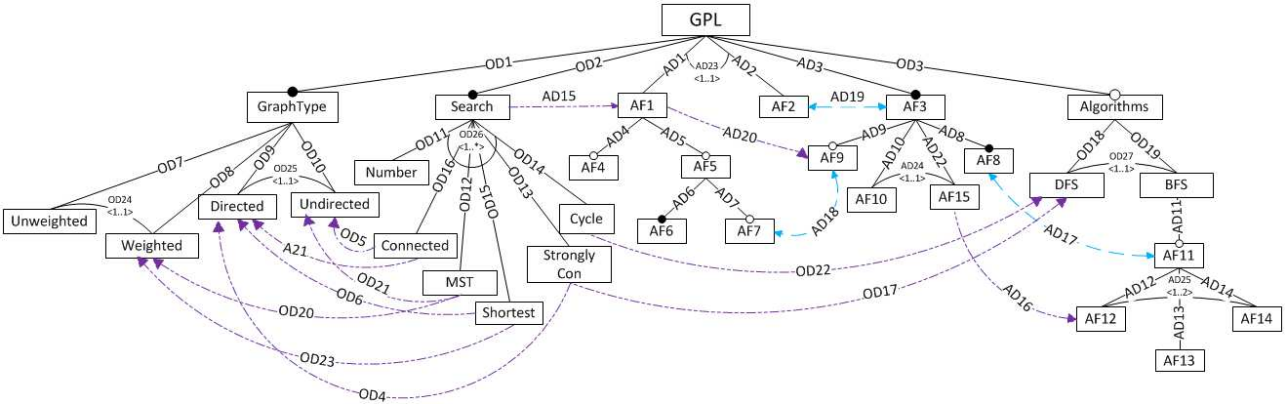


Figure 1. GLP Feature Model based on the one proposed in [23]

B. *Defects in Feature Models*

Defects in PLMs are undesirable properties that adversely affect the quality of the model [8], [24]. In this paper, we are interested in two common types of defects on FMs: Dead features and false optional features.

A feature is dead when it is not present in any valid product of the product line [3], [9], [25], [26]. When a FM has dead features, the model is not an accurate representation of the domain [4]. In fact, if a feature belongs to a FM, the feature is important in the domain that domain analysts want to represent. Therefore, it should be possible to incorporate that feature in at least one product of the product line [4].

|A feature is false optional if it is declared as optional in the FM, but it is required in all valid configurations [5], [9], [11], [12]. This defect also gives a wrong idea of domain that represents the FM.

Generally, dead and false optional features arise when a group cardinality is wrong defined [27] or when the FM has a misuse among the dependencies that relate its features [4], [5], [9]. For instance, if a full mandatory feature requires an optional feature, this optional feature became false optional [9].

Ontologies have proven to be useful for dealing with defects in FMs. For instance, in [16], [17] authors use the semantic relationships between the ontology concepts to define a set of rules to identify defects related to the conformance checking [6] of the FMs (e.g., identify if a feature is required and excluded at the same time for another feature). These rules allow authors to classify the ontology individuals (features) that cause each defect. Noorian *et al* also use ontologies to identify and fix defects related to the conformance checking of the FM [18]. In particular, they use the Pellet[28] reasoner for identifying defects in FMs represented with description logic.

III. PROPOSED SOLUTION

The framework proposed in this paper is presented in this section through two sub-sections. The first one presents how we construct the *Feature Model Ontology*. The *Feature Model Ontology* represents concepts of a meta-model of FMs. The second one presents the tool that we call *Defect analyzer*. This tool identifies dead and false optional features in FMs, identifies its causes and explains in natural language why these defects occur. An overview of our proposal framework is presented in Figure 2.

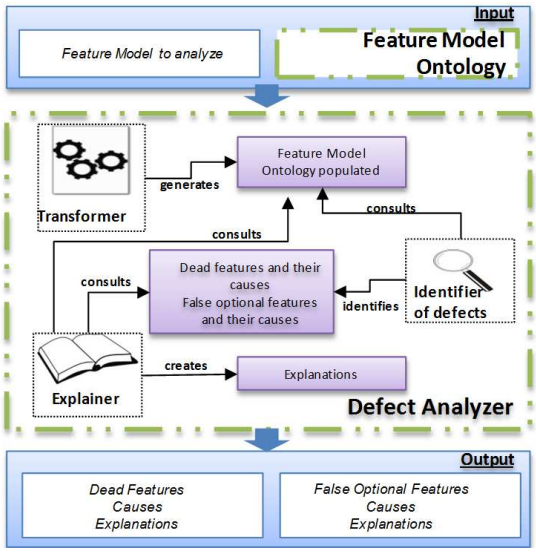


Figure 2. Proposed framework overview

A. *Feature Model Ontology: How to built it*

An ontology is a formal explicit specification for a shared conceptualization [29], [30]. In the same way that FMs, ontologies help to identify and define the domain basic concepts and the dependencies among them.

Ontologies comprises classes, properties, constraints, and individuals [31]. Classes are the main concepts related to the ontology domain. Properties are the data-type properties or object properties. Object properties relate ontology individuals among them, whereas data-type properties relate ontology individuals with concrete values, for example, an integer value. Constraints describe the restrictions that individuals must satisfy to belong to a class; and individuals represent objects in the domain of interest. In this paper, we use ontologies to build our *Feature Model Ontology (*cf. Figure 3).

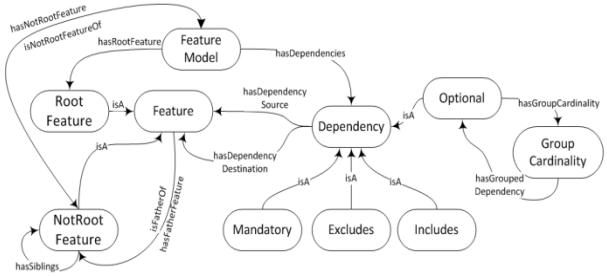


Figure 3. Proposed ontology to represent FM

The *Feature Model Ontology* represents the FMs concepts in the form of ontology. This representation allows us to exploit g the semantic relationships among the concepts involved in FMs. For instance, we can ask for features that have the same father, or features that are related by mandatory and exclude depethe same cies at same time.

We constructed the *Feature Model Ontology* using the guide to construct ontologies proposed in [32], and adapting the UML-based FM meta-model proposed in [6] (cf. Figure 4). We separate the meta-model class *Feature* in the ontology classes *NotRootFeature* and *RootFeature* with the aim of representing in the ontology that a FM only has one root feature. In addition, in the *Feature Model Ontology* the meta-model classes correspond to classes of the ontology; the dependencies between meta-model classes are represented as ontology object properties; and the attributes of the *groupCardinality* meta-model classes are represented as ontology datatype properties. Besides, the inheritance dependencies among the meta-model classes are represented as *isA* dependencies. It is worth noting that we do not consider feature attributes in our ontology (nor in our FM meta-model) since attributes are not involved in the FM defects in which we are interested in this paper.

Since we use ontology classes and properties to represent the FM meta-model, if an individual violates the conditions defined for the classes or properties during the population process, the ontology becomes inconsistent (this issue is beyond the scope of this paper, but is covered in [6] and [18]).

The use of ontologies to represent FMs is not new; in fact, there are several works that use ontologies to represent FMs because ontologies increase the expressiveness level provided by FMs [13], [14]. Other authors are motivated by the fact that the ontological representation of FMs makes possible to verify consistency between the feature model and its meta-model [18]. Even others are motivated by the fact that the ontological representation allows inferring interesting information regarding the FMs; for instance, obtain sibling features [17].

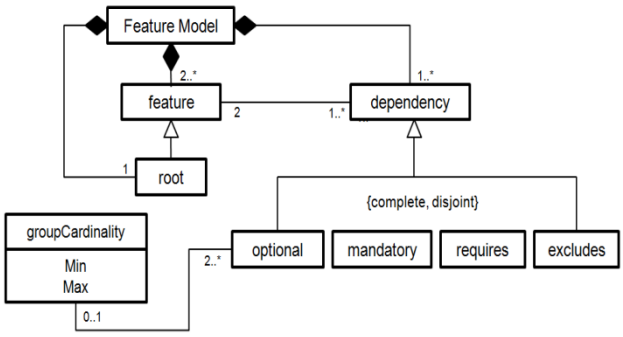


Figure 4. FM meta-model based on the one proposed in [6]

B. Defect analyzer

Defect analyzer is the tool of our proposal that identifies and explains in natural language dead and false optional features. In particular, it receives as input the *Feature Model Ontology* and the FM to analyze, and produces as result the identified dead and false optional features (if any), the causes of each defect and one explanation in natural language for each identified cause. The *Defect analyzer* is composed of three parts: the *transformer*, the *identifier of defects* and the *explainer*. Following sub-sections explain and give details of each one of these modules.

1) Transformer

Transformer module is the responsible of populating the *Feature Model Ontology* with the elements of the FM to analyze. Populate an ontology consists in creating individuals in the classes of the ontology. First, the *Transformer* reads each element of the input FM, and second, it creates one individual in the corresponding ontology class and fills the properties of each individual.

In our populated *Feature Model Ontology*, FM dependencies are individuals of one of the following ontology classes: `Optional`, `Mandatory`, `Requires` and `Excludes`. The class in which the *Transformer* creates each individual depends of the type of dependency in the FM. For instance, dependency OD2 (cf., Figure 1) is an individual of the `Mandatory` ontology class.

All features of the FM are individuals by inheritance of the `Feature` class. Moreover, the FM root is an individual of the `RootFeature` ontology class, and all other features are individuals of the `NotRootFeature` ontology class. For instance, in our running example, `GPL` is an individual of the `RootFeature` ontology class, and `Search` is an individual of the `NoRootFeature` ontology class.

Transformer fills the properties of each individual that it created using information obtained from the input. For instance, according to our *Feature Model Ontology*, individuals of ontology class `Dependency` (e.g., `R3` in our running example) have the properties `hasDependencySource` (`GPL`) and `hasDependencyDestination` (`Search`).

Henceforth other components of the *Defect analyzer* use *Feature Model Ontology* populated with the information of the FM for analyzing the FM.

2) Identifier of defects

Identifier of defects is the module that identifies dead and false optional features and their causes in the populated *Feature Model Ontology*.

We define a set of rules that represent six specific cases of misuse among the FM dependencies that cause dead features, and three specific cases that cause false optional features. Thus, when the *Identifier of defects* applies these rules on the populated *Feature Model Ontology*, the identified features are considered as dead or false optional, and each used rule is a cause that originates the identified dead or false optional features.

The use of rules to detect dead and false optional features is not new. For instance, Van der Massen and Lichter [9] define six rules to identify defects in FODA models (using a feature notation with group cardinalities, as we do, these six rules become two because boolean dependencies can be represented using group cardinalities). However, the approach presented in this paper considers these two rules and seven more that we identified through our academic and industrial experience working with FMs. For each rule, we (i) specify the cause as a general explanation about the defect; (ii) specify the rule in first-order logic; (iii) present an explanation

template; and (iv) present an example based in our running example (cf., Figure 1).

We describe each rule with one or more Horn Clauses [33]. In our Horn Clauses, antecedents are conditions that must occur together for producing the analyzed defect, and the consequent is that a feature is dead or false optional.

Our collection of 9 rules intends to find and explain the causes of dead and false optional features. The first rule is about optional features that become false optional when they are required by full mandatory features. The second rule is about optional features that become false optional when they make part of a group cardinality (with a full mandatory father) having one or several dead features into the bundle. The third rule is about optional features that become false optional when they are required by another false optional feature. The fourth rule refers to optional features that become dead when they are excluded by full mandatory features. The fifth rule is about optional features that become dead when they are excluded by false optional features. The sixth rule deals with optional features that become dead when one of their ancestors is also dead. The seventh rule is about optional features that become dead when they require dead features. The eighth rule is about optional features that become dead when they make part of a group cardinality that has one or several false optional features into the bundle. Finally, the ninth rule refers to optional features that become dead when they require features that make part of a group cardinality (with a full mandatory father), but the number of required features exceeds the upper bound of the group cardinality.

In the rest of this section (i) we formalize in first-order logic each one of these rules; (ii) we present the template in natural language that each rule use to explain the cause of the particular defect that raise the rule—we call that "explanation template"; and (iii) we show how this rule can be used in our running example (cf. Figure 1). We use the following first-order logic predicates, functions and sets to formalize the rules as Horn Clauses:

- `requires(x,y)`: This predicate indicates that feature x requires feature y. In our running example `requires (Cicle,DFS)`.
- `excludes(x,y)`: This predicate indicates that feature x and feature y are mutually exclusives. In our running example `excludes(BFS,F2)`.
- `ancestor(x,y)`: This predicate indicates that feature `x` is an ancestor of feature `y`. In our running example `ancestor (GPL,Weighted)` and `ancestor (GPL,Search)`.
- `nameDependency(x,y)`: This function returns the name of a given dependency that relates feature `x` with feature `y`. In our running example `nameDependency(GPL,Search)` returns `OD2.`
- `ModelFeaturesSet`: This set represents the collection of all features of a feature model.
- `OpSet`: This set represents the collection of all optional features of a feature model.
- `FMSet`: This set represents the collection of all full mandatory features of a feature model.
- `DeadSet`: This set represents the collection of all dead features of a feature model.
- `FalseOptionalSet`: This set represents the collection of all false optional features of a FM.

Where

$$\texttt{OpSet} \wedge \texttt{FMSet} \wedge \texttt{DeadSet} \wedge \texttt{FalseOptionalSet} \subseteq \texttt{ModelFeaturesSet}$$

For the sake of presentation of rules, false optional features with the acronym `FO` and dead features will be referred with the acronym `DF`.

Rule FO1: an optional feature becomes false optional when a full mandatory feature requires an optional feature.

*Formalization***:**

$$\forall x \in \texttt{FMSet}\ ,\quad \forall y \in \texttt{OpSet}:$$

$$\texttt{requires}(x,y) \rightarrow y \in \texttt{FalseOptionalSet}$$

Explanation template: Feature `y` is false optional because it is required for the full mandatory feature `x` in the dependency `nameDependency(x,y)`.

Application to the running example: Feature `AF1` is false optional because it is required for the full mandatory feature `Search` in the dependency `AD15`.

Rule FO2: an optional feature becomes false optional when it is grouped by a group cardinality (with a full-mandatory father) having dead features. The feature must be selected to satisfy the lower group cardinality.

Formalization:

`z= group cardinality (with father feature being full mandatory) of the FM at hand`
`m =Lower bound of z`
`DFGroupSet= {Dead features that belong to z}`
`NotDFGroupSet={Features not dead that belongs to z}`
`GroupFeaturesSet= {Features grouped by the group cardinality z}`

Where,

$$\texttt{GroupFeaturesSet} \subseteq \texttt{ModelFeaturesSet} \wedge \texttt{NotDFGroupSet} = \texttt{GroupFeaturesSet} \setminus \texttt{DFGroupSet}$$

Then,

$$|\texttt{NotDFGroupSet} \models \texttt{m} \rightarrow \texttt{NotDFGroupSet} \subseteq \texttt{FalseOptionalSet}$$

Explanation template: Feature `y` is false optional because it must be selected to satisfy the lower bound `m` of the group cardinality `z` to which it belongs.

Application to the running example: Feature `AF10` is false optional because it must be selected to satisfy the lower bound `1` of the group cardinality `AD24` to which it belongs.

Rule FO3: an optional feature becomes false optional when it is required by another false optional feature.

Formalization:

$\forall x \in FalseOptionalSet, \forall y \in OpSet:$

$requires(x,y) \rightarrow y \in FalseOptionalSet$

Explanation template: Feature y is false optional because it is required by the false optional feature x through the dependency nameDependency(x,y).
Application to the running example: Feature AF9 is false optional because it is required by the false optional feature AF1 through dependency AD20

Rule DF1: an optional feature becomes dead when it is excluded by a full mandatory feature.
Formalization:

$\forall x \in FMSet, \forall y \in OpSet:$

$excludes(x,y) \rightarrow y \in DeadSet$

Explanation template: Optional feature y is dead because it is excluded by the full mandatory feature x through the dependency nameDependency(x,y).
Application to the running example: Optional feature AF11 is dead because it is excluded by the full mandatory feature AF8 through dependency AD17.

Rule DF2: an optional feature becomes dead when it is excluded by a false optional feature.
Formalization:

$\forall x \in FalseOptional, \forall y \in OpSet :$

$excludes(x,y) \rightarrow y \in DeadSet$

Explanation template: Optional feature y is dead because it is excluded by the false optional feature x through the dependency nameDependency(x,y).
Application to the running example: Optional feature AF7 is dead because it is excluded by the false optional feature AF9 through dependency AD18.

Rule DF3: a feature becomes dead when one of its ancestors is dead.
Formalization:

$\forall x \in DeadSet, \forall y \in ModelFeaturesSet:$

$ancestor(x,y) \rightarrow y \in DeadSet$

Explanation template: Feature y is dead because x, its ancestor feature, is a dead feature too.
Application to the running example: Feature AF14 is dead because AF11, its ancestor feature, is a dead feature too. Features AF12 and AF13 are also identified as dead features for this rule.

Rule DF4: a feature becomes dead when it requires another dead feature.
Formalization:

$\forall x \in ModelFeaturesSet, \forall y \in DeadSet:$

$requires(x,y) \rightarrow x \in DeadSet$

Explanation template: Feature x is dead because it requires the dead feature y. The name of the requires dependency is nameDependency(x,y).
Application to the running example: Feature AF15 is dead because it requires the dead feature AF12. The name of the requires-type dependency is AD16.

Rule DF5: a feature becomes dead if it belongs to a group cardinality and the number of false optional features is equal to the cardinality upper bound.
Formalization:

Z: group cardinality of the FM at hand
n: Upper cardinality of z
FOGroupSet: set of false optional features that belong to z
NotFOGroupSet: set of features not false optional that belongs to z}
GroupFeaturesSet= set of features grouped by the group cardinality z
Where,
$FOGroupSet \subseteq GroupFeaturesSet \subseteq ModelFeaturesSet \wedge$
$NotFOGroupSet = GroupFeaturesSet \setminus FOGroupSet$
Then,
$|FOGroupSet| = n \rightarrow NotFOGroupSet \subseteq DeadSet$

Explanation template: Feature y is dead because it cannot be selected from its group cardinality z, since the upper bound n of the group cardinality z is attained with the following false optional features FOGroupSet.
Application to the running example: Feature AF2 is dead because it cannot be selected in its group cardinality AD23, since the upper group cardinality 1 of AD23 is satisfied with the following false optional features: F1.

Rule DF6: an optional feature becomes dead if it requires features that belongs to group cardinality, but the number of required features is greater than the upper bound of the group cardinality.
Formalization:

z: group cardinality (with father feature being full mandatory) of the FM at hand
n: upper cardinality of z
DFGroupSet: set of dead features that belong to z
IncludesFeaturesSet: set of features that belong to z and are includes by another feature of the FM
GroupFeaturesSet: set of features grouped by z
Where,
$IncludesFeaturesSet \subseteq GroupFeaturesSet \subseteq ModelFeaturesSet$

Then,

$\forall y \in OpSet, \forall x \in GroupFeaturesSet:$

$includes(y,x) \rightarrow x \in IncludesFeaturesSet$

$|IncludesFeaturesSet| \geq n \rightarrow y \in deadSet$

Explanation template: Feature `y` is dead because it requires the feature(s) `IncludesFeaturesSet` that belong(s) to the group cardinality `z`. Required feature(s) exceed(s) the upper bound `n` of the group cardinality `z`.

Application to the running example: Feature `Connected` is dead because it requires the feature(s) `Directed, Undirected` that belong(s) to the group cardinality `OD26`. Required feature(s) exceed(s) the upper bound `1` of the group cardinality `OD26`.

It is worth noting that aforementioned rules are interrelated. These relationships are presented in Figure 5. In this figure, identification process begins with the dead features found by rule DF1 and false optional features found by rule FO1. Then, rules DF2, DF5 and DF6 receive as input the identified false optional features, and identify dead features. Inversely, rule FO2 receives as input dead features and identifies false optional features. Rule FO3 receives false optional features as input and identifies new false optional features, and rules DF3 and DF4 receive dead features as input and identify new dead features. The process ends when the *Identifier of defects* executes all rules and it does not find new dead or false optional features. On the contrary, if new dead and false optional features appear, the *Identifier of defects* runs again all rules using false optional and dead features as input to find new ones.

3) Explainer

Once the *Identifier of defects* identifies dead and false optional features and their causes, the *Explainer* constructs explanations in natural language according to the rule used to find each defect. In the explanation process, the *Explainer* executes the following tasks:

- It obtains the rule used to identify each false optional or dead feature.
- It takes the explanation template associated with the rule identified in the previous task.
- It fills the explanation template at hand with the corresponding instances from the populated *Feature Model Ontology*.

It is worth noting that if a feature is involved in more than one rule, the *Identifier of defect* identifies all different rules used to identify this dead or false optional feature. Consequently, the *Explainer* makes for each rule a different explanation. This is the case of `F2` in our example: (i) rule DF1 identifies that feature `F2` is dead because it is excluded by the full mandatory feature `F3`; and (ii) rule DF5 identifies that feature `F2` is dead because it belongs to a group cardinality `<1..1>` where one the features of the bundle (i.e., the children of `F1`) is a false optional feature (due to the dependency `A15`). In that case, the *Explainer* provides an explanation corresponding to (i) and another one corresponding to (ii).

IV. Implementation Details

The method, ontology and rules presented above were implemented into the prototype tool called *Defect analyzer* using Java, and the JESS (Java Expert System Shell)[1] reasoner to execute queries in SQWRL [34]. The tool was tested with the Graph Product Line case study, and with 30 random FMs generated with the BEnchmarking and TesTing on the analYsis (BeTTy) [35] tool. Our approach was implemented in two stages. In the first stage, we used Protégé 3.4.8 for creating the *Feature Model Ontology* to represent concepts of the FMs meta-model. In the second stage, we developed the *Defect analyzer*.

Broadly, each component of the *Defect analyzer* works as follows:

(i) *Transformer*: It uses a library available in the SPLOT website[2], for reading FMs in the Simple XML Feature Model (SXFM) format. Then, this component uses Jena[3] to manipulate the ontology inside Java for creating individuals in the *Feature Model Ontology* with the information of the analyzed FM. When the *Transformer* ends populating the ontology, it creates a new OWL[4] file with the *Feature Model Ontology* populated with the information of the analyzed FM. The OWL file of our *Feature Model Ontology* populated with the running example is available online[5].

(ii) Identifier of defects: It uses SQWRL to implement the rules proposed in the Section III. A SQWRL query comprises an antecedent and a consequent expressed in terms of OWL classes and properties. The antecedent defines the criteria that individuals must satisfy to be selected, and the consequent specifies the individuals to select in the query results. In our approach, SQWRL use classes and properties defined in the *Feature Model Ontology* to query for information of the FM represented as ontology individuals. *Identifier of defects* executes and manipulates all rules from Java.

For the sake of space, we only present the source code of the first rule (i.e., FO1), in which full mandatory features require optional features. Nevertheless, our nine rules have a similar structure. The whole code is available for download from Internet[5].

```
(1) Requires(?z) ^
(2) Optional(?w) ^
(3) hasDependencyDestination(?w, ?a)
(4) hasDependencySource(?z, COMODIN) ^
(5) hasDependencyDestination(?z, ?a) ^ ->
(6) sqwrl:selectDistinct(?a)
```

Lines 1 to 5 define conditions under which a feature can be considered false optional. Line 1 represents any instance of the ontology class `Requires` and line 2

[1] http://herzberg.ca.sandia.gov

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

[3] http://jena.apache.org

[4] The Ontology Web Language (OWL) is a language used to describe the classes and dependencies between ontologies. For more information, please visit http://www.w3.org/TR/owl-guide/

[5] https://sites.google.com/site/raulmazo/

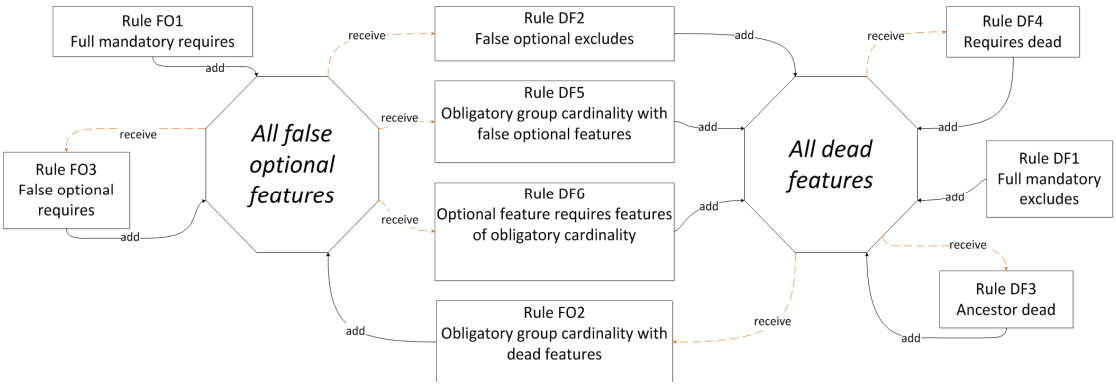


Figure 5. Relationship among our collection of rules

represents any instance of the ontology class `Optional`. Ontology classes `Requires` and `Optional` are subclasses of the ontology class `Dependency` in the *Feature Model Ontology* (cf. Figure 3). Lines 3 to 5 use properties `hasDependencyDestination` and `hasDependencySource` to link a dependency with its related features (cf. Figure 3).First argument of these properties is an individual of the class `Dependency` and the second is an individual of the class `Feature`. Word `COMODIN` in line 3 is an argument that takes the values of individuals identified as full mandatory features. The value of `COMODIN` depends of each rule (e.g., in rule DF2 `COMODIN` corresponds to false optional features, but in rule DF3, corresponds to dead features). Line 6 is the consequent of this query, which consists in selecting the feature `?a`. Note that the SQWRL rule to identify dead or false optional features only selects in the consequent the false optional feature `?a` that satisfy the rule, but it does not select the dependencies related to the defect. For each obtained defect, the *Explainer* executes another SQWRL query to get the necessary information to complete the explanation, as follows:

(iii) *Explainer*: Once the false optional or dead features are identified by the rules presented in (ii), the *Explainer* executes a new SQWRL query to get dependencies and other features related to the defect at hand and fill the explanation template of the corresponding rule. For instance, the following SQWRL obtains the dependency and the features related to each false optional feature obtained from rule FO1.

```
(1) Requires(?z) ^
(2) hasDependencyDestination(?z,COMODIN)^
(3) hasDependencySource(?z,?b) ->
(4) sqwrl:selectDistinct(?b) ^
(5) sqwrl:selectDistinct(?z)
```

Lines 1 to 3 define necessary conditions that must satisfy individuals `?b` and `?z` to be selected in the query. Line 1 represents any instance of the ontology class `Requires`. Lines 2 and 3 define the features source and destination of the ontology class `Requires`. Word `COMODIN` in line 2 is the false optional feature found with the query presented in (ii). The consequent of this SQWRL query consists in selecting feature `?b` requiring the false optional feature `COMODIN` and the requires-type dependency `?z` from `?b` to `COMODIN`. Thus, the explanation corresponding to the rule FO1 is as follows.

"Feature `COMODIN` is false optional because it is required for the full mandatory feature `?b` in the dependency `?z`."

V. PRELIMINARY EVALUATION

We assessed the precision, scalability and usability of our approach with 31 models clustered as presented in Table III.

Our preliminary evaluation was undertaken in the following environment: Laptop with Windows 7 Ultimate of 32 bits, processor Intel® Core™ i5-2410M, CPU 2.30 GHz, and RAM memory of 4,00 GB, of which 2.66 GB is usable by the operating system.

1) Precision

We tested our approach in three steps. First, we verified that it did not generate false positives. Second, we verified that the proposed solution identified 100% of dead and false optional features considered in our collection of rules. Finally, if the FMs had dead or false optional features, we manually validated that explanations corresponded to the case that produced the defect, and that the filled spaces in the explanation templates corresponded to real situation for each one of the models.

In the first stage, we compared the dead and false optional features with the results obtained using FaMa [36] and VariaMos [37]. We found that our proposal identified the 100% of the dead and false optional features that satisfied our rules, with 0% false positive. For the second and third stage, we made a manual inspection of correctness over the running example and two models(randomly selected) of each cluster. We found that our proposal constructed correct explanations; i.e., they corresponded to the cause(s) that originated each defect.

Figure 6 presents the number of dead and false optional features found in each analyzed FM.

TABLE II.
FEATURE MODELS COLLECTION BENCHMARK

Number of features	5	25	32	50	75	100	150
Number of models	5	5	1	5	5	5	5
% of requires and excludes relationship	40	40	18	40	40	40	40

2) Computational Scalability

In order to make performance measurement, we executed five times each of the 31 models, which means 155 (31x 5) queries.

The time measures presented in Figure 7 are the average of the five executions of each model. Y-axis corresponds to computation time in milliseconds (ms) that took the *Defect analyzer* to execute all the tasks of our approach, and X-axis corresponds to the number of features of each model. According to results, our approach took less than 5 sg (5000 ms) executing the *Defect analyzer* in FM up to 100 features and took about two minutes on models with 150 features.

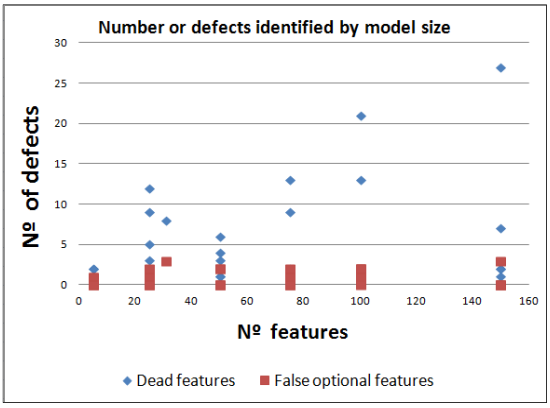


Figure 6. Number of defects identified by model size

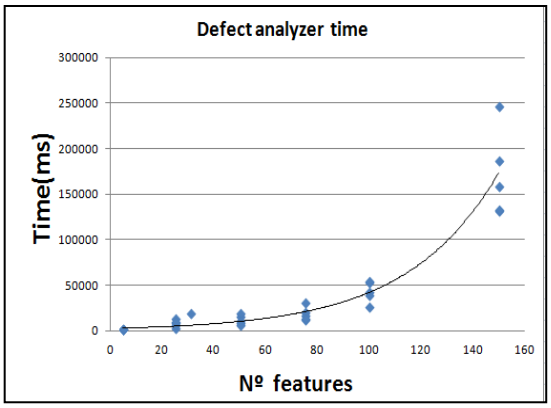


Figure 7. Number of defects identified by model size

3) Usability

In order to make more usable our approach, we developed a graphic presentation of our *Defect analyzer*. Our tool receives a FM, the one selected by the user with the "Choose file" button, in SXFM format. Then, when the user presses the "Analyze" button, the *Transformer* module, populating the *Feature Model Ontology* with the elements of the FM at hand. Then, the modules *Identifier of defects* and *Explainer* process the *Feature Model Ontology* with the individuals of the analyzed FM and present results to the user. Figure 8 corresponds to a snapshot of part of the feedback obtained from our tool when we analyzed dead and false optional features in our running example. For each found defect, our tool says what it corresponds to, the cause that origins it, and gives the corresponding explanation in natural language.

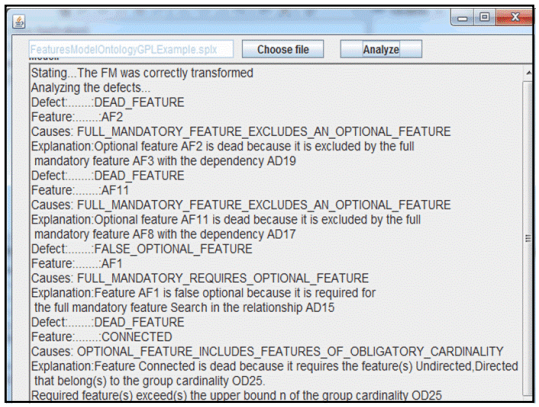


Figure 8. Snapshot corresponding to a part of the results generated from analyzing our FM running example

VI. Related Work

Two collections of approaches for FM defects cause identification can be met in the literature: those that use ontologies to represent and reason on FMs and the others.

From the first category, Wang *et al.* [19] propose representing FMs and their constraints in OWL ontology language. In their proposal, the authors represent each feature as an ontology class, and each dependency as an ontology property. Their study identifies inconsistencies in FMs configurations and provides explanations for inconsistencies. However, their approach does not analyze the FM itself to identify the shortcomings. Abo *et al.* [17] propose to use ontologies to represent FMs and facilitate their integration when they represent different views of a product line. Additionally, these authors describe SWRL (Semantic Web Rule Language) rules to validate model consistency. They define each situation that creates an inconsistency as an antecedent, and the elements involved as the consequent. However, their research aims at facilitating integration of different FMs, whereas that our approach focuses on identifying and explaining dead and false optional features and their causes. Moreover, Lee *et al.* [15] propose to use ontologies to represent FMs in order to analyze their variability and commonality. Even if they use ontologies to represent FM, their approach is different to ours. They use ontologies to analyze the semantic similarity of the FM, whereas our approach uses ontologies to identify dead features and explain their causes. Noorian *et al.* [18] propose to use descriptive logic to: (i) identify inconsistencies in FMs represented in SXFM; (ii) identify inconsistencies in products configured from the product line; and (iii) propose possible corrections. They implement their approach in a framework that uses OWL-DL to represent FMs and their configurations, and Pellet[28] as reasoner. We also use SXFM to represent FMs and description logic to represent our ontology. However, we focus on identifying and explaining dead and false optional features and not on conformance checking [6] as Noorian *et al.* do. Moreover, our approach could detect structural defects if we verify (using the corresponding Protégé's function) the consistency of the ontology after populating it.

Regarding the second category, several works were carried out to automatically identify dead features (and other defects) on FMs [3], [5], [7–10]. However, none of these works deals with identification of causes or explanations of dead and false optional features.

Trinidad *et al.* [12] present an automated method for identifying and explaining defects, such as dead features or false optional features in FMs. The authors transform FMs into a diagnostic problem and then into a constraint satisfaction problem. They automated their approach in FaMa [36], an Eclipse Plug-in for automatic analysis of FMs. Their proposal identifies the dead features and false optional features, and minimum set of dependencies necessary to create such features. However, their approach works like a black box, hard-coded in FaMa, where user cannot create new rules to interrogate the FM. Besides, explanations generated by FaMa are not in natural language, but they are rather a list of dependencies that modeler should modify to remove the defect. Thus, FaMa gives the dependencies participating in the defect, but it does not explain the defect itself, which our approach does.

In a more recent work, Trinidad *et al.* [11] use abductive reasoning to identify dead features and their causes. Unfortunately, authors do not provide any details or even an algorithm to implement their proposal.

It is worth noting that FaMa finds and explains other dead and false optional features that our approach did not identify. This is because we have not implemented all the cases to identify and explain all causes of dead features or false optional features. FaMa identifies all cases because it uses a constraint satisfaction approach to identify dead features, false optional features and other defects on FMs. However, our rule-based approach is extensible, it allows us to explain in natural language why defects occur, and it allows us to analyze dead and false optional features when FMs are void [3], three aspects that FaMa does not support.

VII. Conclusions and Discussion

In this paper, we proposed an ontological rule-base approach to analyze dead and false optional features. Our defect analysis consists in identifying dead and false optional features in FMs, identifying certain causes of these defects, and explaining these causes in natural language. To operationalize our proposal, we propose an OWL ontology for representing FM and we propose 9 rules that represent certain causes that produce dead or false optional features and have associate an explanation in natural language. These rules were formalized in first-order logic and implemented in SQWRL and Java. We validated our proposal with a well-known case study and with 30 random features models with until 150 features.

The approach developed in this paper represents an innovative alternative to the ones found in literature [3], [5], [7–12], [16–19], because we not only identify dead and false optional features, but we also identify their causes and build explanations in a human compressible language. We believe that this information could avoid modelers take the same mistakes in others FMs. However, there are other cases outside of the scope of this proposal (e.g. identifying dead features when they are produced for mandatory features whose predecessor is an optional feature). Indeed, it is necessary to continue extending our solution to identify with other rules dead and false optional features.

We are also interested in exploring dependency between dead features and void models, because we detected that many of our rules could identify void models if they are applied with mandatory and false optional features.

Acknowledgment

We perform this research under the stage of the Master of Engineering - Engineering Systems financed by the National University of Colombia and the Informatics Research Center (CRI) at University Paris I Pantheon Sorbonne in France.

References

[1] J. Bosch, *Design and Use of Software Architectures. Adopting and Evolving a Product-Line Approach*. Addison-Wesley Professional, 2000.

[2] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, 1st ed. Addison-Wesley Professional, 2001.

[3] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feasibility Study Feature-Oriented Domain Analysis (FODA). Technical Report," 1990.

[4] D. Benavides, S. Segura, and A. Ruiz-Cortes, "Automated analysis of feature models 20 years later: A literature review," *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010.

[5] W. Zhang and H. Zhao, "A Propositional Logic-Based Method for Verification of Feature Models," in *Proceedings of the 6th International Conference on Formal Engineering Methods (ICFEM'04)*, 2004, pp. 115–130.

[6] R. Mazo, R. Lopez-Herrejon, C. Salinesi, D. Diaz, and A. Egyed, "Conformance Checking with Constraint Logic Programming: The Case of Feature Models," in *Proceedings of the 35th Annual International Computer Software and Applications Conference (COMPSAC)*, 2011, pp. 456–465.

[7] K. Czarnecki and C. Kim, "Cardinality-based Feature Modeling and Constraints: A progress Report," in *Proceedings of the InternationalWorkshop on Software Factories (OOPSLA 2005)*, 2005.

[8] C. Salinesi and R. Mazo, "Defects in Product Line Models and how to Identify them," in *Software Product Line - Advanced Topic*, InTech., A. Elfaki, Ed. 2012, pp. 1–40.

[9] T. Von der Massen and H. Lichter, "Deficiencies in Feature Models," in *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, 2004.

[10] T. Thüm, C. Kastner, F. Benduhn, J. Meinicke, and Saak, "FeatureIDE: An extensible framework for feature-oriented software development," *Science of Computer Programming*, 2012.

[11] P. Trinidad and A. Ruiz-Cortes, "Abductive Reasoning and Automated Analysis of Feature models: How are they connected," in *Proceedings of the Third International Workshop on Variability Modelling of Software-Intensive Systems*, 2009, pp. 145–153.

[12] P. Trinidad, D. Benavides, A. Duran, A. Ruiz-Cortes, and M. Toro, "Automated Error Analysis for the Agilization of Feature Modeling," *Journal of Systems and Software*, vol. 81, no. 6, pp. 883–896, 2008.

[13] K. Czarnecki and K. T. Kalleberg, "Feature Models are Views on Ontologies," in *Proceedings of the 10th International on Software Product Line Conference (SPLC '06)*, 2006, pp. 41–51.

[14] K. Sandkuhl, C. Thörn, and W. Webers, "Enterprise Ontology and Feature Model Integration - Approach and Experiences from an Industrial Case," in *ICSOFT (PL/DPS/KE/MUSE)*, 2007, pp. 264–269.

[15] S. Lee, J. Kim, C. Song, and D. Baik, "An Approach to Analyzing Commonality and Variability of Features using Ontology in a Software Product Line Engineering," in *Proceedings of the Fifth International Conference on Software Engineering Research, Management and Applications*, 2007, pp. 727–734.

[16] L. Abo, G. Houben, O. De Troyer, and F. Kleinermann, "An OWL- Based Approach for Integration in Collaborative Feature Modelling," in *SWESE 2008. 4th Workshop on Semantic Web Enabled Software Engineering*, 2008.

[17] L. Abo, F. Kleinermann, and O. De Troyer, "Applying semantic web technology to feature modeling," in *Proceedings of*

the 2009 ACM symposium on Applied Computing (SAC '09), 2009, pp. 1252–1256.

[18] M. Noorian, A. Ensan, E. Bagheri, H. Boley, and Y. Biletskiy, "Feature Model Debugging based on Description Logic Reasoning," in *DMS'11*, 2011, pp. 158–164.

[19] H. H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan, "Verifying feature models using OWL," *Web Semant.*, vol. 5, no. 2, pp. 117–129, 2007.

[20] G. Spanoudakis and A. Zisman, "Inconsistency management in software engineering: Survey and open research issues," *Handbook of software engineering*, pp. 329–380, 2001.

[21] T. von der Maßen and H. Lichter, "Deficiencies in feature models," *workshop on software variability …*, 2004.

[22] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Formalizing Cardinality-based Feature Models and their Specialization," *Software Process: Improvement and Practice*, vol. 10, no. 1, pp. 7–29, 2005.

[23] R. E. Lopez-Herrejon and D. S. Batory, "A Standard Problem for Evaluating Product-Line Methodologies," in *Proceedings of the Third International Conference on Generative and Component-Based Software Engineering*, 2001, pp. 10–24.

[24] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated reasoning on feature models," in *Proceedings of the 17th international conference on Advanced Information Systems Engineering*, 2005, pp. 491–503.

[25] P. Trinidad, D. Benavides, and A. Ruiz-Cortés, "Isolated Features Detection in Feature Models," in *Proceedings of Conference on Advanced Information Systems Engineering (CAiSE 2006)*, 2006, vol. 01, pp. 1–4.

[26] A. Osman, S. Phon-Amnuaisuk, and C. Kuan Ho, "Knowledge Based Method to Validate Feature Models," in *First International Workshop on Analyses of Software Product Lines*, 2008, pp. 217–225.

[27] R. Mazo, "A Generic Approach for Automated Verification of Product Line Models," Ph.D.thesis.Paris 1 Panthéon – Sorbonne University, Paris, France, 2011.

[28] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semant.*, vol. 5, no. 2, pp. 51–53, 2007.

[29] W. Borst, "Construction of Engineering Ontologies for Knowledge Sharing and Reuse: Ph.D. Dissertation," University of Twente, 1998.

[30] T. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," in *International Workshop on Formal Ontology*, 1993.

[31] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, "A Practical Guide To Building OWL Ontologies Using The Protege-OWL Plugin and CO-ODE Tools Edition 1.0," 2004.

[32] N. Noy and D. McGuinness, "Ontology Development 101 : A Guide to Creating Your First Ontology," 2001.

[33] M. H. Van Emden and R. A. Kowalski, "The Semantics of Predicate Logic as a Programming Language," *J. ACM*, vol. 23, no. 4, pp. 733–742, 1976.

[34] M. O'Connor and A. Das, "SQWRL: a Query Language for OWL," in *Proceedings of the 6th International Workshop OWL: Experiences and Directions*, 2009.

[35] S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, and A. Ruiz-Cortés, "BeTTy: benchmarking and testing on the automated analysis of feature models," in *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, 2012, pp. 63–71.

[36] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez, "Fama framework," in *Software Product Line Conference, 2008. SPLC'08. 12th International*, 2008, vol. 81, no. 6, pp. 359–359.

[37] R. Mazo, C. Salinesi, and D. Diaz, "VariaMos : a Tool for Product Line Driven Systems," in *Proceedings of the 24th International Conference on Advanced Informa- tion Systems Engineering (CAiSE Forum'12)*, 2012, no. June, pp. 25–29.