

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

L.F. Rincón <sup>a,1</sup>, G.L. Giraldo <sup>a,1</sup>, R. Mazo <sup>b,2</sup> and  
C. Salinesi <sup>b,2</sup>

<sup>a</sup> *Departamento de Ciencias de la Computación y de la Decisión, National University of Colombia, Medellín, Colombia*

<sup>b</sup> *CRI, Panthéon Sorbonne University, Paris, France*

---

## Abstract

Feature models are a common way to represent variability requirements of software product lines by expressing the set of feature combinations that software products can have. Assuring quality of feature models is thus of paramount importance for assuring quality in software product line engineering. However, feature models can have several types of defects that diminish benefits of software product line engineering. Two of such defects are *dead features* and *false optional features*. Several state-of-the-art techniques identify these defects, but only few of them tackle the problem of identifying their causes. Besides, the explanations they provide are cumbersome and hard to understand by humans. In this paper, we propose an ontological rule-based approach to: (a) identify dead and false optional features; (b) identify certain causes of these defects; and (c) explain these causes in natural language helping modelers to correct found defects. We represent our approach with a feature model taken from literature. A preliminary empirical evaluation of our approach over 31 FMs shows that our proposal is effective, accurate and scalable to 150 features.

**Keywords:** Feature Models, Defects, Ontologies, Software Engineering

---

## 1 Introduction

A Software Product Line (SPL) is a family of related software systems with common and variable functions whose first objective is reusability [5]. 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 [6]. SPLE usually uses Feature Models (FMs) to represent the correct combination of features that software products can have. In particular, FMs describe the features and their

---

<sup>1</sup> Email: {luftrinconpe, glgiraldog}@unal.edu.co

<sup>2</sup> Email: {raul.mazo, camille.salinesi}@univ-paris1.fr

dependencies for creating valid products of a product line [13]. 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 feature models that correctly represent the domain it is intended to represent is not trivial [3]. 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 semantics of FMs. A feature is dead if it cannot appear in any product of the product line [13]. A feature is false optional if it is declared as optional, but it appears in all products of the product line [42]. 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 [19].

Numerous researches focus on identifying dead and false optional features in FMs [13,42,8,28,40,34]. Others approaches focus on identifying dead and false optional features, and identifying the causes that produce these defects [38,35]. Some others works propose even using ontologies to represent FMs [9,30,14] and others propose using ontologies for identifying defects in FMs [1,23,41]. 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.

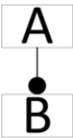
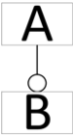
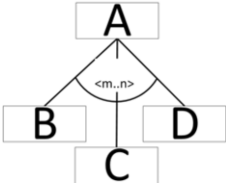
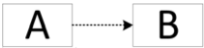

Our general goal is to find a generic technique that will point out the cause of various kinds of defects on product line models specified with different notations. In this paper, we propose a first step to achieve this goal. In particular, we propose an ontological rule-based approach to analyze dead 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 [35,33].

Our original contribution can be summarized as follows:

- (i) We propose a framework that: Identifies dead and false optional features in FMs, identifies the causes of these defects, and creates explanations in natural language about each detected cause.
- (ii) We construct the *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



Table 1  
Types of dependencies in Feature Models

Notation	Type of Dependency
	<b>Mandatory</b> [13] 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 [40].
	<b>Optional</b> [13] 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.
	<b>Group cardinality</b> [7] 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 m of the child features are included into a product, the father feature should be included too.
	<b>Requires</b> [13] Feature B should be included in valid products with feature A. This dependency is unidirectional.
	<b>Excludes</b> [13] Features A and B cannot be in valid products at same time. This dependency is bidirectional.

2.2 Running Example

In this paper we use as running example an adapted version of the Graph Product-Line (GPL) [16]. Figure 1 presents the resulting model. We used this example because it is well-known among the product line community, and it was proposed to be a standard case for evaluating product line methodologies [16].

In order to illustrate our approach, we intentionally introduced eight dead features (cf., AF2, AF7, AF11, AF12, AF13, AF14, AF15, and Connected) and three 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.

Graphs derived from the GPL 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 the **StronglyCon** algorithm requires **Directed** graphs (cf., OD4) and the DFS search algorithm (cf., OD17).

### 2.3 Defects in Feature Models

Defects in product line models are undesirable properties that adversely affect the quality of the model [28,20]. 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 [13,40,36,26]. When a FM has dead features, the model is not an accurate representation of the domain [3]. 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 [3]. A feature is false optional if it is declared as optional in the FM, but it is present in all valid configurations [42,40,38,35]. 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 [17] or when the FM has a misuse among the dependencies that relate its features [3,42,40]. For instance, if a full mandatory feature (a feature that appears in all the products of the product line) requires an optional feature, this optional feature became false optional [40].

Ontologies have proven to be useful for dealing with defects in FMs. For instance, in [1] authors use the semantic relationships between the ontology concepts to define a set of rules to identify defects related to the conformance checking [19] 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 [23]. In particular, they use the Pellet [32] reasoner for identifying defects in FMs represented with description logic.

## 3 Proposed solution

We present our approach through two Sub-sections. The first one presents how we construct the *Feature Model Ontology*, which is an ontology that represents concepts of a meta-model of FMs. The second one presents the *Defect analyzer framework*: our approach that uses the *Feature Model Ontology* for identifying dead and false optional features in FMs, identifying its causes and explaining in natural language why these defects occur.

### 3.1 Feature Model Ontology: How to built it

An ontology is a formal explicit specification for a shared conceptualization [4,11]. 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 [12]. Classes are the main concepts related to the ontology domain. Individuals represent objects in the domain of interest. 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. Finally, constraints describe the restrictions that individuals must satisfy to belong to a class. In this paper, we use ontologies to build our *Feature Model Ontology* (cf. Figure 2).

The *Feature Model Ontology* represents the FMs concepts in the form of an ontology. This representation allows us to exploit 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 dependencies at the same time.

We constructed the *Feature Model Ontology* using the guide to construct ontologies proposed by Noy and McGuinness [24], and adapting the UML-based FM meta-model proposed by Mazo *et al.* [19](cf. Figure 3). 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

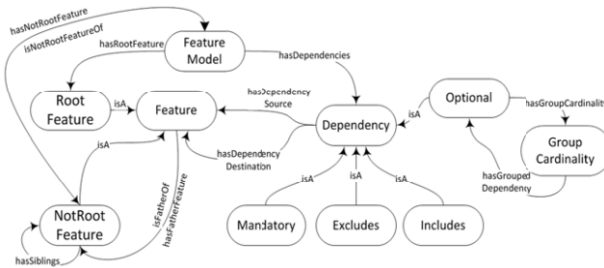


Fig. 2. Proposed ontology to represent feature models

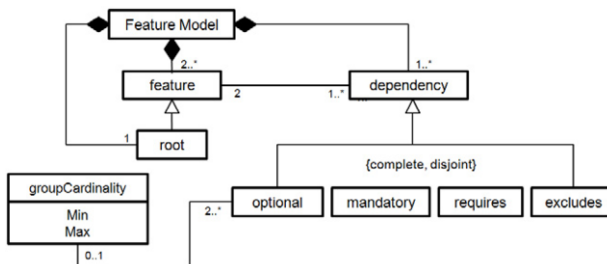


Fig. 3. Feature models meta-model based on the one proposed by Mazo *et al.* [19]

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. However, Mazo *et al.* [19] and Noorian *et al.* [23] provide further details about this topic).

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 [9,30]. 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 [23]. Even others are motivated by the fact that the ontological representation allows inferring interesting information regarding the FMs; for instance, obtain sibling features [1].

### 3.2 Defect analyzer framework

Our *Defect analyzer framework* is formed by two general inputs: (i) the *Feature Model Ontology* and the (ii) FM to analyze; three main parts: (i) the *Transformer*, (ii) the *Identifier of defects* and (iii) the *Explainer*; and two outputs: (i) dead features, their causes, and their explanations, and (ii) false optional features, their causes and their explanations. Figure 4 presents an overview of the proposed framework.

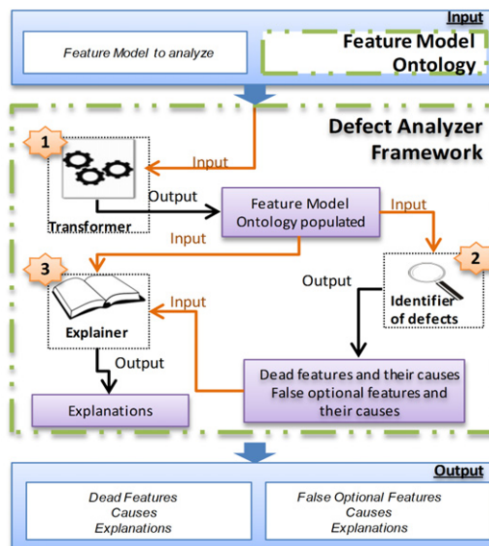


Fig. 4. Proposal framework overview



First, the *Transformer* receives as input the *Feature Model Ontology* explained in Sub-section 3.1, and the FM for which we want to identify dead and false optional features, their causes, and their explanations. With this information, the *Transformer* generates as output a new *Feature Model Ontology* with the elements of the FM to analyze (see Sub-section 3.2.1).

Second, with the populated *Feature Model Ontology* as input, the *Identifier of defects* identifies the dead and false optional features of the FM to analyze and the causes of each defect (see Sub-section 3.2.2).

Finally, the *Explainer* uses the populated *Feature Model Ontology*, the identified defects, and their causes to create explanations for each cause in natural language (see Sub-section 3.2.3).

With all this information, the framework produces as output the identified dead and false optional features (if any), the causes of each defect and one explanation in natural language for each identified cause. Following Sub-sections explain and give details of each main part of our framework.

### 3.2.1 Transformer

Transformer part 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., Fig. 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 framework* use *Feature Model Ontology* populated with the information of the FM for analyzing the FM.

### 3.2.2 Identifier of defects

The *Identifier of defects* is the part 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 [40] 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 [29,2,10,22,18].

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
- (iv) present an example based in our running example (cf., Figure 1).

We describe each rule with one or more Horn Clauses [39]. 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 nine rules intends to identify dead and false optional features and to identify their causes.

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.

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

For the sake of presentation of rules, false optional features will be referred 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:*

$$\forall x \in \text{FMSet}, \forall y \in \text{OpSet} : \text{requires}(x, y) \rightarrow y \in \text{FalseOptionalSet}$$

*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,

$$\begin{aligned} \text{GroupFeaturesSet} &\subseteq \text{ModelFeaturesSet} \wedge \\ \text{NotDFGroupSet} &= \text{GroupFeaturesSet} \setminus \text{DFGroupSet} \end{aligned}$$

Then,

$$| \text{NotDFGroupSet} | = m \rightarrow \text{NotDFGroupSet} \subseteq \text{FalseOptionalSet}$$

*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:*

$$\forall x \in \text{FalseOptionalSet}, \forall y \in \text{OpSet} : \text{requires}(x, y) \rightarrow y \in \text{FalseOptionalSet}$$

*Explanation template:* Feature  $y$  is false optional because it is required for the false optional feature  $x$  through the dependency  $\text{nameDependency}(x, y)$ .

*Application to the running example:* Feature AF9 is false optional because it is required for the false optional AF1 through the dependency AD20.

### Rule DF1

An optional feature becomes dead when it is excluded by a full mandatory feature.

*Formalization:*

$$\forall x \in \text{FMSet}, \forall y \in \text{OpSet} : \text{excludes}(x, y) \rightarrow y \in \text{DeadSet}$$

*Explanation template:* Optional feature  $y$  is dead because the full mandatory feature  $x$  excludes it through the dependency  $\text{nameDependency}(x, y)$ .

*Application to the running example:* Optional feature AF11 is dead because the full mandatory feature AF8 excludes it through the dependency AD17.

### Rule DF2

An optional feature becomes dead when it is excluded by a false optional feature.

*Formalization:*

$$\forall x \in \text{FalseOptional}, \forall y \in \text{OpSet} : \text{excludes}(x, y) \rightarrow y \in \text{DeadSet}$$

*Explanation template:* Optional feature  $y$  is dead because the false optional feature  $x$  excludes it through the dependency  $\text{nameDependency}(x, y)$ .

*Application to the running example:* Optional feature AF7 is dead because the false optional feature AF9 excludes it through dependency AD18.

### Rule DF3

A feature becomes dead when one of its ancestors is dead.

*Formalization:*

$$\forall x \in \text{DeadSet}, \forall y \in \text{ModelFeaturesSet} : \text{ancestor}(x, y) \rightarrow y \in \text{DeadSet}$$

*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.

This rule also identifies two other dead features in our running example: **AF12** and **AF13**.

#### Rule DF4

A feature becomes dead when it requires another dead feature.

*Formalization:*

$$\forall x \in \text{ModelFeaturesSet}, \forall y \in \text{DeadSet} : \text{requires}(x, y) \rightarrow x \in \text{DeadSet}$$

*Explanation template:* Feature  $x$  is dead because it requires the dead feature  $y$ . The name of the requires-type dependency is  $\text{nameDependency}(x, y)$ .

*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  $z$

$\text{FOGroupSet}$  = Set of false optional features that belong to  $z$

$\text{NotFOGroupSet}$  = Set of features not false optional that belongs to  $z$

$\text{GroupFeaturesSet}$  = Set of features grouped by the group cardinality  $z$

Where,

$$\begin{aligned} \text{FOGroupSet} &\subseteq \text{GroupFeaturesSet} \subseteq \text{ModelFeaturesSet} \wedge \\ \text{NotFOGroupSet} &= \text{GroupFeaturesSet} \setminus \text{FOGroupSet} \end{aligned}$$

Then,

$$|\text{FOGroupSet}| = n \rightarrow \text{NotFOGroupSet} \subseteq \text{DeadSet}$$

*Explanation template:* Feature  $y$  is dead because it cannot be selected from its group cardinality  $z$ .

*Application to the running example:* Feature **AF2** is dead because it cannot be selected from its group cardinality **AD23**.

#### 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  $z$

$\text{DFGroupSet}$  = Set of dead features that belong to  $z$

$\text{IncludesFeaturesSet}$  = Set of features that belong to  $z$  and included by another

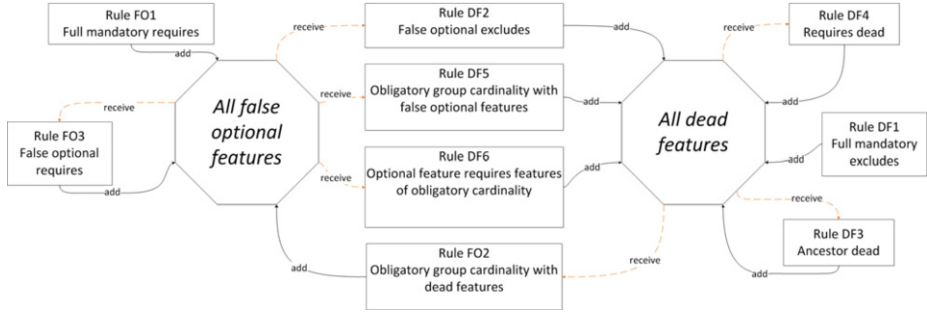


Fig. 5. Relationship among our collection of rules

feature of the FM

*GroupFeaturesSet* = Set of features grouped by  $z$

Where,

$$\text{IncludesFeaturesSet} \subseteq \text{GroupFeaturesSet} \subseteq \text{ModelFeaturesSet}$$

Then,

$$\forall y \in \text{OpSet}, \forall x \in \text{GroupFeaturesSet} : \text{includes}(y, x) \rightarrow x \in \text{IncludesFeaturesSet} \\ \wedge | \text{IncludesFeaturesSet} | > n \rightarrow y \in \text{DeadSet}$$

*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 F01. Then, rules DF2, DF5 and DF6 receive as input the identified false optional features, and identify dead features. Inversely, rule F02 receives as input dead features and identifies false optional features. Rule F03 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.

### 3.2.3 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).

## 4 Implementation details

The ontology and the framework presented above were implemented into a prototype tool using Java, and the JESS (Java Expert System Shell)<sup>3</sup> reasoner to execute queries in SQWRL [25]. 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 implemented each part of the *Defect analyzer framework* as a component using Java.

Broadly, each component works as follows:

### 4.1 Transformer

This part uses a library available in the SPLOT website, for reading FMs in the Simple XML Feature Model (SXFM) format. Then, this component uses Jena<sup>4</sup> 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 of populating the ontology, it creates a new OWL<sup>5</sup> file with the *Feature Model Ontology* populated with information of the analyzed FM. The OWL file of our *Feature Model Ontology* populated with the running example is available online<sup>6</sup>.

### 4.2 Identifier of defects

This part uses SQWRL to implement the rules presented in the Section 3. A SQWRL query comprises an antecedent and a consequent expressed in terms of OWL classes and properties. The antecedent defines the criteria that individuals

<sup>3</sup> <http://herzberg.ca.sandia.gov/jess>

<sup>4</sup> <http://jena.apache.org>

<sup>5</sup> 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/>

<sup>6</sup> <https://sites.google.com/site/raulmazo/>

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. The *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., F01), in which full mandatory features require optional features. Nevertheless, our nine rules have a similar structure.

```
(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 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 2). Lines 3 to 5 use properties **hasDependencyDestination** and **hasDependencySource** to link a dependency with its related features (cf. Figure 2). 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**.

### 4.3 Explainer

Once the false optional or dead features are identified by the rules presented in the Sub-section 4.2, 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 F01.

```
(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



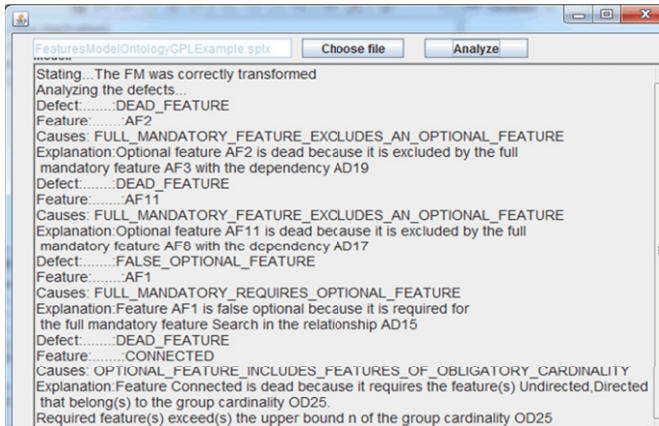


Fig. 6. Snapshot corresponding to a part of the results generated from analyzing our FM running example

with the query presented in the Sub-section 4.2. 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 *F01* is as follows:

“Feature *COMODIN* is false optional because it is required for the full mandatory feature *?b* in the dependency *?z*.”

#### 4.4 Defect Analyzer tool: Graphic presentation

We made a graphic presentation of the tool that implements the *Defect Analyzer Framework*. This tool receives as input a FM in SXFM format selected by the user with the “Choose file” button. Once the user presses the “Analyze” button, the *Transformer* module, reads the selected feature model and populates the *Transformer* with the features and the available FM dependencies. Then, the *Identifier of defects* and the *Explainer* execute the defined SQWRL rules on the populated *Feature Model Ontology*. Finally, the identified dead and false optional features, as well as their causes in natural language are presented to the user. Figure 6 corresponds to a snapshot of part of the feedback obtained from our tool when we analyzed our running example.

## 5 Preliminary evaluation

We assessed the precision and scalability of our approach with 31 models clustered as presented in Table 2. One of these models corresponds to the Graph Product Line model [16] and the others 30 are random FMs generated with the BEnchmarking and TesTing on the analYsis (BeTTy) tool [31]. 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.

Table 2  
Feature models collection of benchmarks

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

### 5.1 Precision

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

In the first stage, we manually compared the dead and false optional features with the results obtained from FaMa [37] and VariaMos [21]. 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 works well constructing correct explanations; i.e., they coincided to the cause(s) of each defect. Figure 7 presents the number of dead and false optional features found in each FM that we have used in our preliminary evaluation.

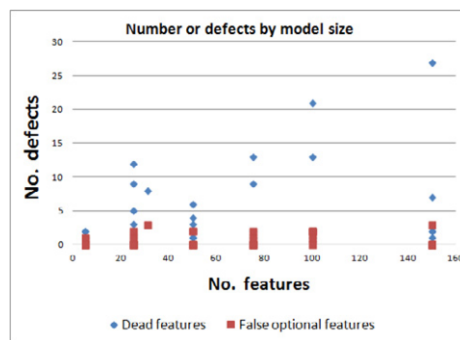


Fig. 7. Number of defects identified by model size

### 5.2 Computational scalability

Aiming to test the performance of our approach, we calculated the computation time in milliseconds (ms) that took the *Defect analyzer* to execute all the tasks of our approach. The time expended by our solution to analyze each of our 31 feature models is shown in Figure 8. In order to obtain the most accurate measure each

FM was tested 5 times, and the average of these values were used as the time on Y-axis in Figure 8. Conversely, the X-axis represents the number of features of each model.

According to the test results, our approach took less than 5s to execute the *Defect analyzer* in FM until 100 features and about two minutes on models with 150 features. It is important to highlight that this time difference on results between FM of 100 features and FM of 150 features is due to our approach looks for dead and false optional features that satisfy the rules in the populated *Feature Model Ontology*; therefore if the FMs size increases, the search space and the execution time increase accordingly.

## 6 Related work

We divide the related research studies in two groups: First, studies related to using ontologies in product line models, and second, those related to identifying and explaining the causes of dead and false optional features.

For the first group, Wang *et al.* [41] 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 invalid FM configurations and explains, using ontological terms, the reason why that configuration is invalid. However, their approach does not analyze the FM to identify and explain dead features or false optional features.

Abo, Kleinermann and De Troyer [1] 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 our approach focuses on identifying and explaining dead and false optional features and their causes. Moreover, Lee *et al.* [14] 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

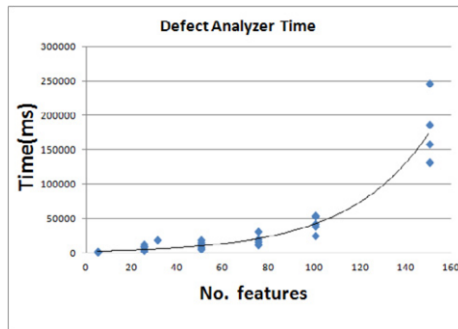


Fig. 8. Defect Analyzer Time (Number of Features vs Time)

dead and false optional features and explain their causes.

Noorian *et al.* [23] 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 [32] 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 [19] as Noorian *et al.* do. Moreover, our approach could detect structural defects if we verify (using the corresponding Protgs function) the consistency of the ontology after populating it.

For the second group, several works were carried out to automatically identify dead features (and other defects) on FMs [13,42,8,28,40,34]. However, none of these works deals with to identify causes and explain them in natural language for dead and false optional features.

Trinidad *et al.* [35] 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 [37], an Eclipse Plug-in for automatic analysis of FMs. Their proposal identifies dead features and false optional features, and the set of dependencies that entail in a FM the fewest changes to fix these defects. 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. In contrast to our approach, FaMa gives the dependencies participating in the defect, but it does not explain the defect itself.

In a more recent work, Trinidad and Ruiz-Cortés [38] 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 does not identify. It is explained by the fact that we have not implemented all the cases producing dead or false optional features. 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 [13], three aspects that FaMa does not support.

Rincón *et al.* [27] propose a method to explain why each dead feature occurs in FMs. In this approach, authors transform FMs into constraint satisfaction problem, and then they identify all the minimal corrections subsets (MCSes) [15] of dependencies that could be modified to correct each dead feature of the FM. This approach, identify the list of dependencies that entail the fewest changes to fix the defect, and also identify others set of dependencies that imply more changes to fix the defect.

Works proposed by Trinidad *et al.* [35], Trinidad and Ruiz-Cortés [38], and

Rincón *et al.* [27] compute the (minimal) correction subsets of constraints, which must be erased from the constraint program to correct the model. However, apart from showing the list of constraints to product line engineers, no supplementary information is offered to them about why these constraints are generating one or several defects on product line models. Constraint satisfaction techniques are not enough to provide these explanations because, for instance, the structure needed to provide these explanations, is lost when the models were transformed into constraint programs. Thus, we decide to transform both, the structure and the semantics of product line models into ontologies, because both properties are important to explain potential defects. For instance, we use the properties and the structure of the resulting ontologies to know the ancestors, brothers and children of features involved in defects. Based on that information, the approach presented in this paper gives explanations in natural language according to each defect situation, instead of only providing the list of features and constraints involved in the defect.

## 7 Conclusions and discussion

In this paper, we proposed an ontological rule-based approach to analyze dead and false optional features. Our defect analysis is 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 nine rules representing 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 [16] 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 [13,42,8,28,40,34,38,35,1,23,41], 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, by means of 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.

## Acknowledgement

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 Panthéon Sorbonne in France.

## References

- [1] Abo, L., G. Houben, O. De Troyer, and F. Kleinermann, An OWL- Based Approach for Integration in Collaborative Feature Modelling, in: *Proceedings of the 4th Workshop on Semantic Web Enabled Software Engineering*, Germany, 2008.
- [2] Alférez, G.H., V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz, *Dynamic adaptation of service compositions with variability models*, Journal of Systems and Software.(2013).
- [3] Benavides, D., and S. Segura, *Automated analysis of feature models 20 years later: A literature review* Information Systems **35** (2010), 615–636.
- [4] Borst, W., “Construction of Engineering Ontologies for Knowledge Sharing and Reuse,” Ph.D. thesis, University of Twente, 1998.
- [5] Bosch, J., Design and Use of Software Architectures. “Adopting and Evolving a Product-Line Approach,” Addison-Wesley Professional, 2000.
- [6] Clements, P., and L. Northrop, “Software Product Lines: Practices and Patterns,” 1st ed., Addison-Wesley Professional, 2001.
- [7] Czarnecki, K., S. Helsen, and U. Eisenecker, *Formalizing Cardinality-based Feature Models and their Specialization* Web Software Process: Improvement and Practice **10** (2005), 7–29.
- [8] Czarnecki, K., and C.H.P. Kim, Cardinality-based Feature Modeling and Constraints: A progress Report, in: *Proceedings of the International Workshop on Software Factories*, San Diego-California, 2005.
- [9] Czarnecki, K., C.H.P. Kim, and K.T. Kalleberg, Feature Models are Views on Ontologies, in: *Proceedings of the 10th International on Software Product Line Conference*, IEEE Computer Society, Washington, DC, USA, 2006: pp. 41–51.
- [10] Dumitrescu, C., R. Mazo, C. Salinesi, and A. Dauron, Bridging the gap between product lines and systems engineering, in: *Proceedings of the 17th International Software Product Line Conference (SPLC)*, ACM Press, Tokyo, Japan, 2013: pp. 254–263.
- [11] Gruber, T., Toward Principles for the Design of Ontologies Used for Knowledge Sharing, in: N. Guarino, R. Poli (Eds.) *International Workshop on Formal Ontology*, Padova, Italy, 1993.
- [12] Horridge, M., and H. Knublauch, A. Rector, R. Stevens, C. Wroe, “A Practical Guide To Building OWL Ontologies Using The Protege-OWL Plugin and CO-ODE Tools Edition 1.0,” 2001.
- [13] Kang, K.C., S.G. Cohen, J.A. Hess, W.E. Novak, and S.P. Peterson, “Feasibility Study Feature-Oriented Domain Analysis ( FODA ),” Technical Report, 1990.
- [14] Lee, S., 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*, IEEE, 2007: pp. 727–734.
- [15] Liffiton, M., and K. Sakallah, *Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints* Journal of Automated Reasoning **40** (2008), 1–33.
- [16] Lopez-Herrejon, R., and D. Batory, A Standard Problem for Evaluating Product-Line Methodologies, in: *Proceedings of the Third International Conference on Generative and Component-Based Software Engineering*, Springer-Verlag, London, UK, 2001: pp. 10–24.
- [17] Mazo, R., “A Generic Approach for Automated Verification of Product Line Models,” Ph.D. thesis, Paris 1 Panthéon Sorbonne University, Paris, France, 2011.
- [18] Mazo, R., P. Grnbacher, W. Heider, R. Rabiser, C. Salinesi, and D. Diaz, Using Constraint Programming to Verify DOPLER Variability Models, in: *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive System (VaMos11)*, Namur-Belgium, 2011: pp. 97–103.
- [19] Mazo, R., 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)*, IEEE Press. Best Paper Award, Munich-Germany, 2011: pp. 456–465.
- [20] Mazo, R., C. Salinesi, and D. Diaz, *Abstract Constraints: A General Framework for Solver-Independent Reasoning on Product-Line Models*, Journal of International Council on Systems Engineering (INCOS) **14** (2011), 22–24.

- [21] Mazo, R., C. Salinesi, and D. Diaz, VariaMos: a Tool for Product Line Driven Systems, in: *Proceedings of the 24th International Conference on Advanced Information Systems Engineering (CAiSE Forum'12)*, Springer Press, Gdansk-Poland, 2012: pp. 25–29.
- [22] Mazo, R., C. Salinesi, D. Diaz, and O. Djebbi, A. Michiels, *Constraints: the Heart of Domain and Application Engineering in the Product Lines Engineering Strategy* Web International Journal of Information System Modeling and Design IJISMD **2** (2011), 33–68.
- [23] Noorian, M., A. Ensan, E. Bagheri, H. Boley, and Y. Biletskiy, Feature Model Debugging based on Description Logic Reasoning, in: *DMS'11*, 2011: pp. 158–164.
- [24] Noy, N., and D. McGuinness, “Ontology Development 101: A Guide to Creating Your First Ontology,” 2001.
- [25] O'Connor, M., and A. Das, SQWRL: a Query Language for OWL, in: *Proceedings of the 6th International Workshop OWL: Experiences and Directions*, Chantilly, 2009.
- [26] Osman, A., 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] Rincón, L.F., G.L. Giraldo, R. Mazo, C. Salinesi, and D. Diaz, Subconjuntos Mínimos de Corrección para explicar características muertas en Modelos de Líneas de Productos. El caso de los Modelos de Características, in: *Proceedings of the 8th Colombian Computer Conference (CCC)*, Armenia-Colombia, 2013.
- [28] Salinesi, C., and R. Mazo, “Defects in Product Line Models and how to identify them,” in: *Software Product Line - Advanced Topic*, InTech, 2012: pp. 97–122.
- [29] Salinesi, C., R. Mazo, and D. Diaz, Criteria for the verification of feature models, in: *Proceedings of the 28th INFORSID (INformatique Des ORganisations et Systèmes d'Information et de Décision)*, Marseille - France, 2010: pp. 293–308.
- [30] Sandkuhl, K., and C. Thörn, W. Webers, Enterprise Ontology and Feature Model Integration - Approach and Experiences from an Industrial Case, in: J. Filipe, B. Shishkov, M. Helfert (Eds.) *ICSOF (PL/DPS/KE/MUSE)*, INSTICC Press, 2007: pp. 264–269.
- [31] Segura, S., J.A. Galindo, D. Benavides, J.A. Parejo, and A. Ruiz-Corts, 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*, ACM, New York, NY, USA, 2012: pp. 63–71.
- [32] Sirin, E., B. Parsia, B.C. Grau, A. Kalyanpur, and Y. Katz, *Pellet: A practical OWL-DL reasoner*, Web Semantics: Science, Services and Agents on the World Wide Web **5** (2007), 51–53.
- [33] Spanoudakis, G., and A. Zisman, *Inconsistency management in software engineering: Survey and open research issues* Handbook of Software Engineering. (2001), 329–380.
- [34] Thüm, T., C. Kastner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, *FeatureIDE: An extensible framework for feature-oriented software development* Science of Computer Programming (2012).
- [35] Trinidad, P., D. Benavides, A. Duran, A. Ruiz-Cortés, and M. Toro, *Automated Error Analysis for the Agilization of Feature Modeling* Journal of Systems and Software **81** (2008), 883–896.
- [36] Trinidad, P., D. Benavides, and A. Ruiz-Corts, Isolated Features Detection in Feature Models, in: *Proceedings of Conference on Advanced Information Systems Engineering (CAiSE 2006)*, 2006: pp. 1–4.
- [37] Trinidad, P., D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez, FAMA Framework, in: *Proceedings of the 12th International Software Product Line Conference (SPLC'12)*, IEEE Computer Society, Washington, DC, USA, 2008: p. 359.
- [38] Trinidad, P., and A. Ruiz-Cortés, 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.
- [39] Van Emden, M.H., and R.A. Kowalski, *The Semantics of Predicate Logic as a Programming Language* Journal of the ACM **23** (1976), 733–742.
- [40] Von der Massen, T., and H. Lichter, Deficiencies in Feature Models, in: T. Mannisto, J. Bosch (Eds.), *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, 2004.
- [41] Wang, H., Y. Li, J. Sun, H. Zhang, and J. Pan, *Verifying feature models using OWL* Web Semantics: Science, Services and Agents on the World Wide Web **5** (2007), 117–129.
- [42] Zhang, W., and H. Zhao, A Propositional Logic-Based Method for Verification of Feature Models, in: J. Davies, W. Schulte, M. Barnett (Eds.), *Proceedings of the 6th International Conference on Formal Engineering Methods (ICFEM'04)*, Seattle-USA, 2004: pp. 115–130.