Modelling and Correcting Duplication in Evolving Software Product Lines

Amal Khtira[1], Anissa Benlarabi[2] and Bouchra El Asri[3]

**IMS Team, ADMIR Laboratory, Rabat IT Center, ENSIAS, Mohammed V University
Rabat, Morocco**

Abstract

Software Product Lines (SPLs) are long-living systems that require inevitably continuous changes to product line models. Many studies in the literature have dealt with different challenges related to the evolution of software product lines. Among these challenges, the detection and correction of model defects have received a great interest. In this vein, our work addresses this challenge and focuses on a specific model defect, which is feature duplication. The main objectives of this paper are to propose a formal definition of feature duplication, to provide a meta-model that describes the dependencies between duplication-related concepts, and finally to present in details our solution to detect this defect in evolving software product lines. In order to illustrate our approach, we use an open source SPL called FeatureAMP.

Keywords: *Software Product Line, Feature Models, Software Evolution, Duplication, Natural Language Processing.*

1. Introduction

Software Product Line Engineering is an approach that aims at creating specific products for different customers while reducing development cost and enhancing product quality [1]. Feature-Oriented SPLs are centered on the notion of Feature. Indeed, the behavior and the functions of a system are described through the definition of features, then these features are used to determine the variability and commonality of the system, which enables the generation of a panoply of specific products that responds to different customers' needs.

Software Product Lines are long-living systems that require inevitably permanent evolution. This evolution is generally caused by new technologies, new customer requirements or new business strategies. Researchers have carried out many studies that deal with evolution-related issues in software product lines. These studies can be classified into four main categories: Evolution traceability [2][3][4], evolution modelling [5][6][7], co-evolution analysis [8][9][10], and finally change impact analysis [11][12][13]. In the context of the last category, we deal particularly with model defects caused by SPL evolution. A literature review about model defects in software product lines has shown that some defects such as ambiguity and inconsistency have generated considerable interest [7][11][12][15][20][27][28], while other defects such as obsolescence, omission and duplication have not been thoroughly treated. In our work, we focus on Feature duplication which occurs when two or more features of the same semantics co-exist in a feature model of a software product line.

Thus, the objective of this paper is to provide definition and modelling for all the concepts related to feature duplication and to describe in details our framework proposed to optimize the evolution of SPLs through the correction of feature duplication.

The remainder of the paper is structured as follows. Section 2 gives an overview of the background of our work, namely software product line evolution, model defects in SPLs and feature duplication. Section 3 proposes a definition of feature duplication and all the underlying concepts, and provides a meta-model that relates these concepts. In Section 4, we present the details of the framework proposed to detect and correct feature duplication when evolving software product lines. The different processes of the framework are illustrated through the FeatureAMP product line. Section 5 presents some studies in the literature that address model defects in software product lines. Finally, Section 6 concludes the paper and describes future work.

2. Background and Objective

In this section, we present the background of our work. First, we address the software product line evolution and discuss the challenges related to this issue. Then, we list the different model defects discussed in the literature, and finally we highlight the concept of duplication in software product lines.

2.1. Software Product Line Evolution

Software evolution has always been one of the issues most addressed in literature. Software product lines are no exception, since they are long lived systems that incur significant evolution throughout their service life due to

new business strategies, new customers' requirements or new technology challenges. Many studies in the literature have dealt with issues related to SPL evolution. These issues can be classified into four categories: Evolution traceability, evolution modelling, co-evolution analysis, change impact analysis.

Evolution Traceability [2][3][4]: The traceability is a mechanism that helps identify and trace links between the artefacts of a SPL or between its different versions. The approaches dealing with traceability address in general the evaluation of change history, the analysis of relationships between interrelated artefacts and the detection of potential inconsistencies, which enables the anticipation of future decisions and the estimation of evolution cost.

Evolution Modelling [5][6][7][16]: In the case of software product lines, the evolution impacts different types of assets, namely requirements, architecture and code. In order to preserve the integrity of these assets, several works have proposed strategies for change management and defined systematic and controlled stages of evolution, which simplifies the evolution process.

Co-evolution Analysis [8][9][10]: In software product lines, two kinds of co-evolution are discussed. The first type concerns co-evolution between artefacts like in [8] where co-evolution is captured between the variability model, the makefiles and the source code in a specific Linux kernel release, or in [9] that analyzed the co-evolution between feature models and code. The second type is the co-evolution between the core platform and the derived products, which was discussed in [10]. According to this study, the products of a software product line could evolve independently of the domain, which leads to a set of single applications instead of applications belonging to the same platform. Thus, an approach is proposed in this paper as a solution to this problem.

Change Impact Analysis [11][12][13][17][40]: The evolution of complex and large scale systems is a difficult task since any change can have adverse effects on all parts of a system. The analysis of change impact helps estimate the maintenance effort, define evolution-related tasks and take the right decisions concerning the change implementation. It also enhances the product quality and ensures the system integrity by detecting potential defects.

Within this context, our work aims at the verification of SPL models during evolution through the detection and correction of model defects.

2.2. Model Defects in SPLs

As a result of software product lines evolution, some defects can be introduced in the different artefacts of the product line (i.e. requirements, features, design and code). Based on a systematic review on model defects in software product lines, we could identify the different defects addressed in the literature and determine the different solutions dealing with these defects. Table 1 contains the definitions of these defects.

Table 1: Definitions of Model Defects

Model Defect	*Definition*	*Source*
Ambiguity	Some Information from the feature model is not clear, allowing multiple interpretations for the specified domain.	[14]
Duplication	To have the same thing expressed in two or more places; duplication can happen in specifications, processes and programs.	[18]
Erosion	Erosion means that realization artifacts become overly complex due to unforeseeable changes.	[21]
Non-attainable domains	A non-attainable value of a domain is the value of an element that never appears in any product of the product line.	[19]
Uncertainty	Requirements uncertainty refers to changes that occur to requirements during the development of software.	[22]
Incompleteness	The lack of necessary information related to a feature or requirement.	[23]
Inconsistency	Some feature model element is not consistent with another element from the same feature model.	[14]
Incorrectness	Some information or behavior from the feature model contradicts its domain specification.	[14]
Extraneous information	Some Information in the feature model is outside the domain scope.	[14]
Unsafety	This happens when the behavior of existing products is affected by a new evolution.	[17]
Redundancy	Redundancy in a PLM is the presence of reusable elements and variability constraints among them that can be omitted from the PLM without loss of semantic on the PLM.	[19]
Non-conformance	Given a feature f, and a (FSMd, FSMr) pair corresponding to f, we say that the design of f conforms to the requirements of f, if every variant of the FSMd has a corresponding FSMr variant.	[25]
Obsolescence	An obsolete software requirement is a software requirement, implemented or not, that is no longer required for the current release or future releases.	[26]
Omission	Some information from the domain was not properly included in the feature model.	[14]

An analysis of the different papers concerned by the systematic review has shown that the model defect most discussed in the literature is inconsistency [7][12][13][14][20][27][28], while other defects are not thoroughly treated, especially duplication.

2.2. The Notion of Duplication

Duplication as described by [18] is the fact of having the same thing expressed in two or more places and can happen in specifications, processes and programs. Based on the systematic review of model defects in SPLs and a complementary review of duplication in software, we found out that the majority of approaches dealing with duplication focus on code cloning [29][46][47]. However, software product line evolution includes also a change in functional specifications and in system models due to the addition of new requirements and the modification or the removal of existing ones. This change may cause duplications both in the SPL domain and the derived applications.

Studies working on code cloning overlook the fact that the detection of defects in the implementation phase could be an expensive and time-consuming task. Hence, the activity of feature deduplication must always be carried out in an early stage of the development lifecycle in order to avoid the propagation of duplication in the next steps of the project, to achieve a satisfactory level of quality and to reduce the implementation cost.

In this vein, we decided to deal with the problem of duplication in the feature level. Our objective is to detect and correct potential duplications introduced in feature models during the evolution of software product lines.

3. Duplication Modelling

In this section, we provide definition and modelling for all the concepts related to duplication in feature-oriented software product lines.

3.1. SPL-Related Concepts

In feature-oriented SPLs, domain and application models are expressed in feature models that represent the SPL variability, while the requirements of new evolutions are generally expressed in natural language specifications. A definition of all these concepts is provided bellow.

Definition 1: Variation Point [30]

A variation point p_i represents one or more locations at which variation occurs.

Definition 2: Variant [31]

A variant v_{ij} is a unique option of a variation point that represents a possible realisation of variability.

Definition 3: Feature

A Feature $\boldsymbol{F}$ is a tuple $(\boldsymbol{R}, \boldsymbol{C}, \boldsymbol{A}, \boldsymbol{V})$ where:

- $\boldsymbol{R}$ is the root element
- $\boldsymbol{C}$ is the cardinality of the feature
- $\boldsymbol{A}$ is a finite set of annotations
- $\boldsymbol{V}$ is the feature type

A feature is the main constituent of the feature model and represents an indivisible function of the system. A feature has two cardinalities, a minimal and a maximal and could be related to a set of annotations that clarifies its semantics. According to the mapping proposed in [1], we consider that a feature can be either a variation point or a variant.

Definition 4: Domain Model

A domain is a family of related products, and the domain model $\boldsymbol{D}$ is the representation of all the different and common features of these products.

$\boldsymbol{PD}$ is the set of variation points of $\boldsymbol{D}$ and $\boldsymbol{VD}$ is the set of variants of *D*.

- $PD = \{pd_i \mid i \leq m \text{ and } i \in \mathbb{N}\}$
- $\forall pd_i \in PD \quad \exists VD_i \text{ where } VD_i = \{vd_{ij} \mid j \in \mathbb{N}\}$
- $VD = \bigcup_{i=1}^{m} VD_i$

This definition involves two assumptions:

- A variation point $\boldsymbol{pd_i}$ from the set $\boldsymbol{PD}$ is associated with a set $\boldsymbol{VD_i}$ that contains the variants $\boldsymbol{vd_{ij}}$.
- $\boldsymbol{VD}$ is the union of all the sets $\boldsymbol{VD_i}$. It represents the set of all the variants of the domain model.

Definition 5: Application Model

An application model $\boldsymbol{A}$ is the model corresponding to an individual application. It is generated by binding the variability of the domain model in a way that satisfies the needs of a specific customer.

$\boldsymbol{PA}$ is the set of variation points of $\boldsymbol{A}$ and $\boldsymbol{VA}$ is the set of variants of **A**.

- $PA = \{pa_i \mid i \leq n \text{ and } i \in \mathbb{N}\}$
- $\forall pa_i \in PA \quad \exists VA_i \text{ where } VA_i = \{va_{ij} \mid j \in \mathbb{N}\}$
- $VA = \bigcup_{i=1}^{m} VA_i$

The association between the domain model and the application model can be expressed as follows:

- $PA \subseteq PD$: The set of variation points of an application is a subset of the set of variation points of the SPL domain.

- $VA \subseteq VD$: The set of variants of an application is a subset of the set of variants of the SPL domain.

Definition 6: Specification

A Specification ***S*** is a description of the intended behavior of a software product. ***PS*** is the set of variation points of ***S*** and ***VS*** is the set of variants of ***S***.

- $PS = \{ps_i \mid i \leq q \text{ and } i \in \mathbb{N}\}$
- $\forall ps_i \in PS \;\; \exists VS_i \text{ where } VS_i = \{vs_{ij} \mid j \in \mathbb{N}\}$
- $VS = \bigcup_{i=1}^{m} VS_i$

In our case, a specification contains the details of all the features that have to be implemented during an evolution of the system and it's expressed in natural language.

3.2. Formalizing Duplication

After defining the concepts related to the evolution of feature-oriented software product lines, we propose a definition for the notion of equivalence and duplication.

Definition 7: Equivalence

Two elements E_1 and E_2 are said to be equivalents if they have the same semantics and represents the same function:

$$E_1 \equiv E_2$$

In our work, this notion is applied both to variation points and variants. Based on equivalence, we introduce the notion of feature duplication. Duplication occurs when we implement independently two requirements that seem to be different when in fact they refer to the same business need.

Definition 8: Duplication

Let p_i be a variation point and v_{ij} one of its variants:

$$\exists p \text{ where } p_i \equiv p \text{ and } \exists v \text{ where } v_{ij} \equiv v \Rightarrow Duplication$$

We distinguish two types of duplication, internal duplication and external duplication.

Definition 8.1: Internal Duplication

Internal duplication is detected between the features of the model or between the features of the specification.

For models: Let $(p_i, v_{ij}) \in PD \times VD_i$

$$\exists p \in PD \text{ where } p_i \equiv p \text{ and } \exists v \in VD \text{ where } v_{ij} \equiv v \Rightarrow InternalDuplication$$

For specifications: Let $(p_i, v_{ij}) \in PS \times VS_i$

$$\exists p \in PS \text{ where } p_i \equiv p \text{ and } \exists v \in VS \text{ where } v_{ij} \equiv v \Rightarrow InternalDuplication$$

Definition 8.2: External Duplication

External duplication is detected between the features of the model and those of the specification.

Let $(p_i, v_{ij}) \in PS \times VS_i$

$$\exists p \in PD \text{ where } p_i \equiv p \text{ and } \exists v \in VD \text{ where } v_{ij} \equiv v \Rightarrow ExternalDuplication$$

3.3. Meta-Modelling

In order to present the relations between the different concepts defined previously, we propose the meta-model depicted in ***Fig. 1***.

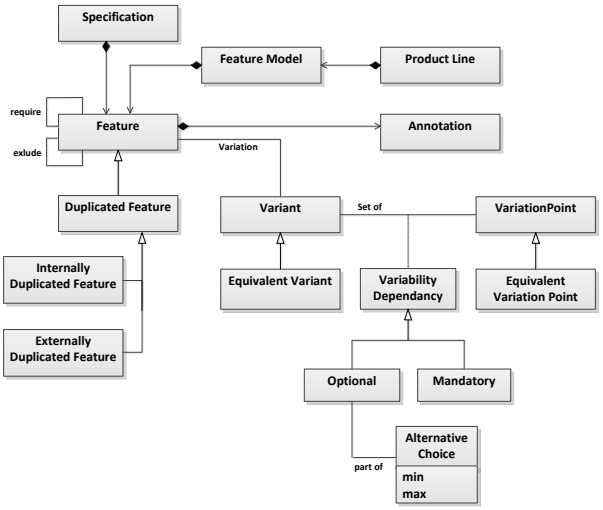

Fig. 1 Meta-model of Duplication in FO-SPL

According to Fig. 1, a feature is either a component of a feature model or associated to a specification. To add more semantics to a feature, annotations can be defined. The variability of features is defined using variation points and variants. The dependency between these two classes is represented with an abstract class, and we distinguish two types of dependencies, mandatory and optional. The dependency is mandatory when a variant of a variation point must be selected in every derived application. The dependency is optional when a variant of a variation point may or may not be selected in an application. In addition to dependencies, transversal constraints can be added to features, such as « *Require* » et « *Exclude* ». The constraint « *Require* » means that the selection of a feature requires the selection of the other. As for « *Exclude* », it means that two features can't co-exist in the same application.

In the meta-model, the notion of equivalence is presented using two classes « *Equivalent Variant* » and « *Equivalent Variation Point* » that inherit respectively from the classes « *Variant* » and « *Variation Point* ».

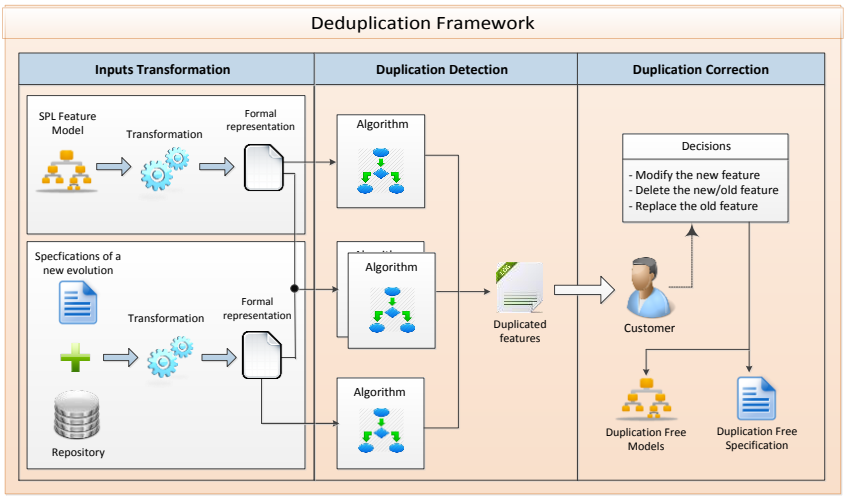

Fig. 2 The overview of the proposed framework.

Similarly, the notion of duplication is expressed via the class « Duplicated Feature » that inherits from the class « Feature ». We distinguish the two types of duplication via the classes « Internally Duplicated Feature » and « Externally Duplicated Feature » that inherit both from « Duplicated Feature ».

4. A Framework for Feature Deduplication in SPLs

This section presents the details of the framework proposed in [32] as a solution for detecting and correcting feature duplication in evolving SPLs. As depicted in Fig. 2, this framework is based on three main processes: Inputs Transformation, duplication detection and duplication correction.

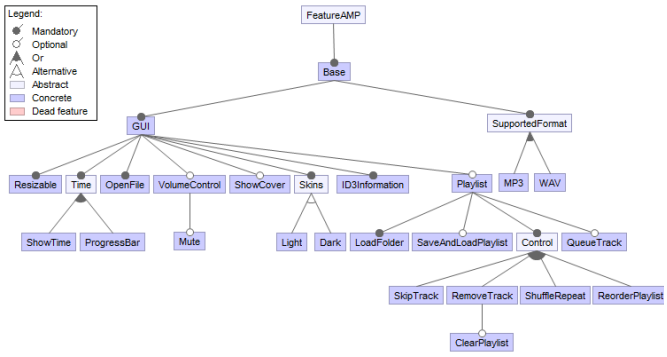

Fig. 3 The Domain Model of FeatureAMP

The first process consists of transforming the framework inputs into a more formal representation. These inputs are: the domain model of the SPL, the application model of a derived product and the specification of a specific evolution. In the second process, a set of algorithms are applied do detect internal and external duplications. The last process is responsible for analyzing the potential duplications in order to take the rights decisions concerning their correction.

In the rest of this section, we present the different processes in details and we illustrate them through an open source SPL called FeatureAMP [33] whose domain model is presented in Fig. 3.

4.1. Model Transformation

In order to model the domain and application models of the proposed SPL, we use FeatureIDE [34]. FeatureIDE is an open source framework based on Eclipse that supports all steps of the SPL development cycle, especially domain analysis and feature modeling. Indeed, it provides the possibility to present graphically the SPL features and the dependencies between them, to configure the application models from a domain model and to generate automatically an XML file for the feature models. This file is structured using the following tags:

- <and> with the option « Mandatory »: for mandatory features.
- <and> without the option « Mandatory »: for optional features.
- <or> : For features related by the OR-relation.
- <alt> : For features related by the XOR-relation.
- <feature> : For the features existing in the bottom of the tree.

This representation of models is centered on dependencies and doesn't take into account the notion of variability. Consequently, we need a supplementary step to transform this tree into a new representation that focuses both on variability and semantics. For this, we propose the following mapping rules:

- The tags <and>, <or> and <alt> correspond to variation points.
- The tag <feature> correspond to variants.

In the case of different levels of variation points, we take the lowest level because we consider the higher levels as abstract. By applying the mapping rules on the FeatureAMP domain model, we obtain the arborescence illustrated in Fig. 4.

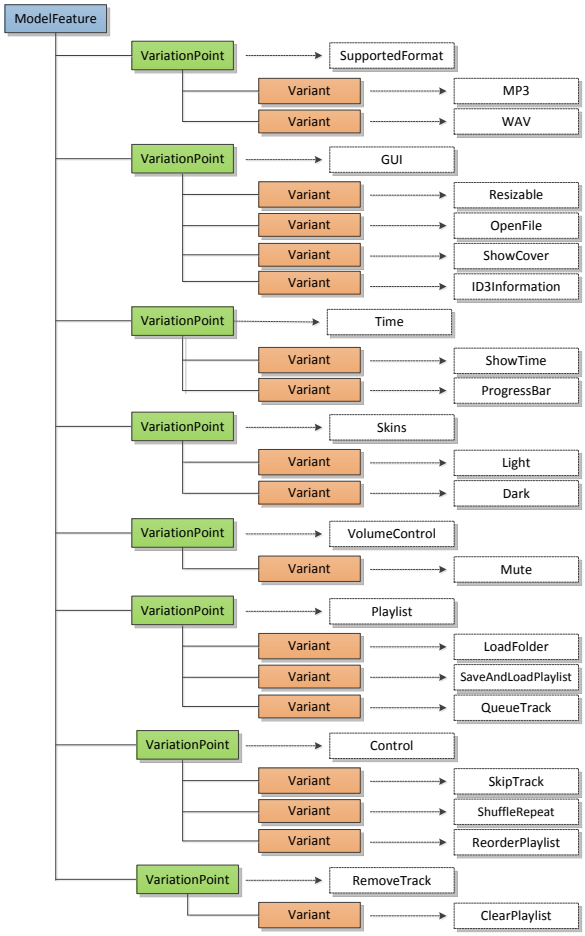

Fig. 4 The new representation of FeatureAMP domain model

4.2. Specification Transformation

The objective of this section is to transform specifications into a simple representation that allows the detection of duplications inside the specifications and duplications against feature models. In Section 4.1., we explained the method followed to transform a feature model to a new arborescence that contains two tags "Variation Point" et "Variant". In order to unify the framework inputs and facilitate the comparison between then, we chose to transform the specification into the same structure.

The transformation of textual specifications consists of analyzing syntactically and semantically the sentences of the specification at the aim of understanding the requirements described by the client and extracting the potential variation points and variants. So that the detection of entities from a specification is performed automatically, the machine needs a repository of entities built upon the SPL domain. The management of this repository is presented in Section 4.3. The proposed approach is based essentially on the notion of Machine Learning [35]. Thus, to enhance the activity of entity recognition, the repository must be updated constantly by following the SPL evolutions.

In the rest of this section, we describe in details the different steps of the specification transformation, namely syntactic and semantic analysis. To illustrate this activity, we consider the specification depicted in Fig. 5 that represents an evolution of FeatureAMP.

> The new version of the application supports the WAV and OGG formats. The user can play a track, stop it, pause it, skip forward or skip rewind. He can also add and remove tracks from the playlist. In addition, the application must give the possibility to repeat the playlist in the order set by the user.

Fig. 5 The Evolution Specification

- **Syntactic Analysis**

The activity of syntactic analysis is composed of three main actions: The detection of sentences, the tokenization and the parsing.

- **Detection of sentences**: Since the input specification is a textual document composed of sentences, this first operation consists of detecting the punctuation marks that indicate the end of a sentence then write each sentence in a separate line. The result of this step is a document that contains one sentence per line.
- **Tokenization**: This action is responsible for segmenting sentences into tokens. A token can be a word, a punctuation, a number, etc. At the end of this action, all of the tokens of the specification are separated using whitespace characters, such as a space or line break, or by punctuation characters.
- **Parsing**: The objective of this action is to analyze each word in the specification and determine its role in the sentence to which it belongs, based on the rules of a specific language grammar. In our case, the language used in the specifications is English. A parser marks all the words of a sentence using a POS tagger (part-of-speech tagger) and converts the sentence into a tree that represents the syntactic structure of the sentence. This action enables us to verify for example whether a sentence is affirmative or negative or whether a requirement is mandatory or optional, etc.

- **Semantic Analysis**

This activity consists of extracting semantic information from a sentence. In our approach, we are interested especially in the parts of sentences considered as variation points or variants. In order to accomplish this task, we must feed the base model of the repository with tagged variants from the domain model. The variants are not necessarily named entities; they can also be a part of a sentence.

During this activity, the content of each sentence is compared against the repository of entities (variation points, variants) already created, in order to detect the variants that potentially exist in the specification. This operation is performed automatically.

Once all of the variants are tagged, the corresponding tree is generated. Since we follow a machine learning approach, the initial model is updated continuously to improve the operation of variants detection. Indeed, the more the model is full with tagged variants, the more the recognition of variants from the specification is accurate.

At the end, the result of the specification transformation is a tree whose nodes are tagged either with <VariationPoint> or <Variant>. For the example of FeatureAMP, the tree corresponding to the specification is illustrated in Fig. 6.

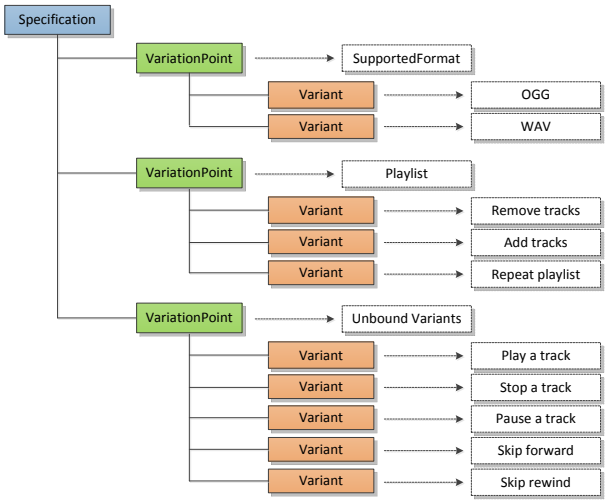

Fig. 6 The Specification Tree

4.3. Repository Management

The repository is a central element in the proposed framework, because it is used by both the first and the second processes. It consists of two main components, the model and the dictionary.

- **The model**

The model (or the glossary) is used basically in the transformation of natural language specifications. Indeed, the model stores the general specifications of the SPL domain previously annotated in a way that distinguishes specific entities. In our work, we are interested especially in variation points and variants. The construction of the model is an up-front investment that must be performed in the phase of requirements engineering of a software product line. Hence, the activity of specification tagging has to be done in parallel with the conception of the domain model.

So that the model can detect entities in a specification with a satisfying level of precision, it must keep up with the evolution of the SPL platform and its derived products. Thus, our approach is based on machine learning, because during each new evolution, the specification is verified based on the model, and when the analysis is done, the new generated specification is annotated and added to the initial model in order to enrich it and enhance the precision of entity recognition.

- **The dictionary**

Since the proposed approach is based on a semantic comparison, a dictionary is thus necessary to compare the new features against the new ones and to detect the potential duplications. The dictionary, as its name suggests, contains the definition of all the features of the product line, their description and their synonyms, which helps detect both internal and external duplications. A dictionary is initially built based on the domain features, but should be constantly updated and refined to improve the activity of detecting duplications.

4.4. Duplication Detection

The second process of the framework consists of detecting duplications introduced into a SPL during a new evolution. This process includes two main activities [32]: i) Detection of Internal Duplication and ii) Detection of External Duplication.

- **Detection of Internal Duplication**

Internal duplication corresponds to duplication in one of the framework inputs, the feature models or the specification. Since both inputs are transformed to a unified representation, the algorithm used to detect duplication is the same and is composed of the following steps:

- Define a key synonym for each set of synonyms based on the dictionary.
- Update of all the variation points and variants by their synonyms in the specification or in the model in question.
- Put in alphabetical order the variation points and the variants.

- For every variation point, delete the duplicated variants (For every variation point, compare the first variant with the second. If the two variants are equivalents, delete the first, else move to the next comparison and repeat the same action until all the variants of a variation point are deleted).
- Detect the duplicated pairs (variation point, variant).

- **Detection of External Duplication**

External duplication corresponds to duplication between the feature models and the specification. We distinguish six possible cases of a new pair (variation point, variant) or (p_i, v_{ij}) [36].

Case 1. The variation point p_i has an equivalent in $\boldsymbol{PA}$ and the variant $\boldsymbol{v_{ij}}$ has an equivalent in $\boldsymbol{VA_i}$. In this case, duplication occurs against the application model. Consequently, the two elements must be removed from the specification, but the domain model and the application model do not change.

$$\exists p \in PA \; where \; p_i \equiv p \; and \; \exists v \in VA_i \; where \; v_{ij} \equiv v \Rightarrow Duplication$$

Case 2. The variation point p_i has an equivalent in $\boldsymbol{PA}$ and the variant $\boldsymbol{v_{ij}}$ has an equivalent in $\boldsymbol{VD_i}$. In this case, duplication occurs against the domain model. Thus, a derivation of the variant from the domain model is sufficient.

$$\exists p \in PA \; where \; p_i \equiv p \; and \; \exists v \in VD_i \backslash VA_i \; where \; v_{ij} \equiv v \; \Rightarrow Duplication$$

Case 3. The variation point p_i has an equivalent in $\boldsymbol{PA}$ and the variant $\boldsymbol{v_{ij}}$ has no equivalents in $\boldsymbol{VD_i}$. In this case, there is no duplication; the pair is thus new and must be implemented.

$$\exists p \in PA \; where \; p_i \equiv p \; and \; \nexists v \in VD_i \; where \; v_{ij} \equiv v \nRightarrow Duplication$$

Case 4. The variation point p_i has an equivalent in $\boldsymbol{PD}$ but not in $\boldsymbol{PA}$ and the variant $\boldsymbol{v_{ij}}$ has an equivalent in $\boldsymbol{VD_i}$ but not in $\boldsymbol{VA_i}$. In this case, duplication occurs against the domain model. Thus, a derivation of the variant from the domain model is sufficient.

$$\exists p \in PD \backslash PA \; where \; p_i \equiv p \; and \; \exists v \in VD_i \backslash VA_i \; where \; v_{ij} \equiv v \; \Rightarrow Duplication$$

Case 5. The variation point p_i has an equivalent in $\boldsymbol{PD}$ but not in $\boldsymbol{PA}$ and the variant $\boldsymbol{v_{ij}}$ has no equivalents in $\boldsymbol{VD_i}$. In this case, there is no duplication; the pair is thus new and must be implemented.

$$\exists p \in PD \backslash PA \; where \; p_i \equiv p \; and \; \nexists v \in VD_i \; where \; v_{ij} \equiv v \nRightarrow Duplication$$

Case 6. The variation point p_i has no equivalents in $\boldsymbol{PD}$ and the variant $\boldsymbol{v_{ij}}$ has no equivalents in $\boldsymbol{VD_i}$. Dans ce cas, on conclut qu'il n'y pas de duplication ni par rapport au domaine ni par rapport à l'application. In this case, there is no duplication; the pair is thus new and must be implemented.

$$\nexists p \in PD \; where \; p_i \equiv p \; and \; \nexists v \in VD_i \; where \; v_{ij} \equiv v \nRightarrow Duplication$$

Based on the identified cases, two algorithms were proposed for the detection of external duplication, one for the comparison between specifications and domain models, and the second for the comparison between specifications and application models. Even if the two algorithms are similar, we chose to separate the two verifications, because the decision taken in each case is different.

In order to implement the two algorithms, we are working on a tool support that we introduced in [37]. By verifying the specification against the corresponding application model of FeatureAMP, we found the results presented in Table 2.

Table 2: Cases Detected in the Specification

(VariationPoint, Variant)	Case
(SupportedFormat, OGG)	Case 3
(SupportedFormat, WAV)	Case 1
(Playlist, RemoveTracks)	Case 3
(Playlist, AddTracks)	Case 3
(Playlist, RepeatPlaylist)	Case 3
(UnboundVariants, PlayATrack)	Case 6
(UnboundVariants, StopATrack)	Case 6
(UnboundVariants, PauseATrack)	Case 6
(UnboundVariants, SkipForward)	Case 6
(UnboundVariants, SkipRewind)	Case 6

4.5. Feature Deduplication

As depicted in Fig. 7, the process of feature deduplication involves two main activities, the analysis of detected duplications and the generation of a correct specification (or a feature model).

The principal inputs of the first activity are: i) the feature model and ii) the log containing the potential duplications generated in the duplication detection process. The analyst, with the help of the customer, analyses the duplications to assess their relevance and validate or not their removal.

A number of decisions may be taken:

- **The removal of the new feature**: If the analyst chooses this decision, the sentences containing the

duplications are deleted automatically from the specification and a new correct specification is generated.

- **The modification of the new feature**: A demanded feature may be badly expressed and must be modified to respond exactly to the client's need. In this case, after the addition of the new feature, a new verification can be carried out to verify that a new duplication wasn't introduced.
- **The replacement of the old feature**: The analyst can decide to delete an old feature and replace it by a new feature. In this case, the specification doesn't change, but work must be done to delete the old feature.

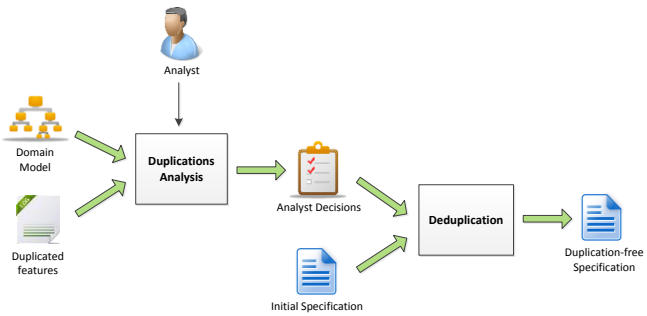

Fig. 7 The Process of specification deduplication

The second activity takes as inputs the decisions of the analyst and the initial specification of the evolution. In the output, it provides a duplication-free specification that can be used to implement the demanded evolution. It should be noted that the first activity can't be performed automatically because the analyst's intervention is mandatory. In contrast, the second activity is automatic.

5. Related Work

A plethora of papers have dealt with model defects in software product lines. In this section, we present these papers according to the artefact they address (Requirements, models, architecture and code).

5.1. Requirement Verification

In the studies dealing with requirements, many papers followed an approach based on natural language processing (NLP) to verify the textual specifications of a software product line. For instance, [38] carried out a systematic literature review to investigate the applications of NLP in the context of Software Requirements Engineering (SRE) between 2002 and 2016. Hajri et al. [27] propose an NLP-based tool for the verification of use cases and the associated models, whose variability is defined with the method PUM (Product line Use case modeling Method). Ali et al. [39] intend to verify the Software Requirement Specification (SRS) document by proposing a methodology of four processes i.e. Parsing Requirement (PR), Requirement Mapping using Matrix (RMM), Addition of Requirements in SRS template and Third Party Inspection. The objective of this paper is basically to minimize ambiguities and incorrectness inside the SRS. In addition, many tools of requirements verification have been proposed, such as RSLingo [40], TRIC [4] and Marama [41].

5.2. Model Verification

The majority of papers addressing the verification of domain models focus on feature models [12][15][20][28] [41], which is logical given that software product lines in literature are most of the time feature-oriented. Several solutions have been proposed in this sense, namely tools such as VML4RE [42], VCC [20][28] and SPLEnD [25], extensions of the DOPLER tool [16][43], frameworks such as SPLEMMA [7][47] and techniques such as FMCheck [14].

5.3. Architecture Verification

The studies concerning architecture in software product lines deal with architecture documents or UML models such as class diagrams or components diagrams. Dam et al. [24] focus on the merging of artifacts in software product lines. For this, it presents an approach to automatically merge consistent artifacts, inform users of the potential inconsistent/conflicting artifacts and propose ways to resolve them. Shumaiev and Bhat [44] retrieved different types of uncertainties based on the analysis of three real-world software architecture documents. Then discussed how existing NLP techniques could help authors of software architecture documents to detect various kinds of uncertainty. Farias et al. [45] present an exploratory study that evaluates empirically the impact of stability on the effort of model composition. In this study, some composition heuristics were applied to 18 versions of design models related to three product lines. The main finding was that stable models tend to reduce the inconsistency rate and to lower the model composition effort.

5.4. Code Verification

A review of the papers addressing code verification has shown that the defect that has received most attention is code cloning. For example, [46] proposes a conceptual framework based on machine learning to detect code clones. The authors use summaries generated by deep neural networks as metrics to measure similarities between code snippets. Schmorleiz and Lämmel [29] describe a process for similarity management of cloned variants during software evolution. This process uses annotations

to record developers' intentions and to anticipate automatic change propagation. Hellebrand et al. [9] address the coevolution between feature models and code. More specifically, it proposes metrics that allow the detection of variability erosion between the two artefacts during SPL evolution. Rubin et al. [47] focus on the management of software product variants realized via cloning. For this purpose, the authors present a framework that consists of seven conceptual operators and validate their efficiency through three case studies from the automotive industry.

6. Conclusion and Future Work

Many studies in the literature proposed solutions to optimize the evolution of software product lines. The challenges addressed in these studies concern in particular the evolution traceability, evolution modelling, co-evolution and change impact. In our work, we focused on the last category and especially on the model defects caused by the SPL evolution. Based on a systematic review, we found out that the problem of feature duplication hasn't been given a big interest in the literature. Thus, in this paper, we proposed a formal definition of all the duplication-related concepts and a meta-model that describes the relations between them. Then, we described a solution to detect duplication in feature-oriented software product lines. The different processes of the proposed framework were illustrated through the SPL FeatureAMP. Currently, we are working on a tool support for feature deduplication based on the proposed framework. In future work, we intend to provide the details of this tool and to describe the results of the application of our approach on an industrial product line.

References

[1] K. Pohl, G. Böckle, and F. Van Der Linden, Software Product Line Engineering Foundations, Principles, and Techniques, Berlin, Germany: Springer-Verlag, 2005.

[2] D. Yu, P. Geng, and W. Wu, "Constructing traceability between features and requirements for software product line engineering", in 19th Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2012, pp. 27-34.

[3] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J. C. Royer, A. Rummler, and A. Sousa, "A model-driven traceability framework for software product lines", Software and Systems Modeling, Vol. 9, No. 4, 2010, pp. 427-451.

[4] A. Goknil, I. Kurtev, K. van den Berg, and J. Veldhuis, "Semantics of trace relations in requirements models for consistency checking and inferencing, Software Systems Modeling", Springer, Vol. 10, No. 1, 2011, pp. 31-54.

[5] S. Lity, S. Nahrendorf, T. Thüm, et al., "175% Modeling for Product-Line Evolution of Domain Artifacts", in 12th International Workshop on Variability Modelling of Software-Intensive Systems, ACM, 2018, pp. 27-34.

[6] A. Pleuss, G. Botterweck, D. Dhungana, et al., "Model-driven support for product line evolution on feature level", Journal of Systems and Software, Vol. 85, No. 10, 2012, pp. 2261-2274.

[7] D. Romero, S. Urli, C. Quinton, et al., "SPLEMMA: A generic framework for controlled-evolution of software product lines", in 17th International Software Product Line Conference, 2013, pp. 59-66.

[8] L. Passos, K. Czarnecki, S. Apel, et al., "Feature-oriented software evolution", in 7th International Workshop on Variability Modelling of Software-intensive Systems, ACM, 2013, p. 17.

[9] R. Hellebrand, A. Silva, M. Becker, et al., "Coevolution of variability models and code: an industrial case study", in 18th International Software Product Line Conference, ACM, Sept. 2014, Vol. 1, pp. 274-283.

[10] A. Benlarabi, A. Khtira, and B. El Asri, "A Co-Evolution Analysis for Software Product Lines: An Approach based on Evolutionary Trees", International Journal of Applied Evolutionary Computation (IJAEC), Vol. 6, No. 3, 2015, pp. 9-32.

[11] M. Bhushan, S. Goel, and K. Kaur, "Analyzing inconsistencies in software product lines using an ontological rule-based approach", Journal of Systems and Software, 2017.

[12] A. O. Elfaki, "A rule - based approach to detect and prevent inconsistency in the domain - engineering process", Expert Systems, Vol. 33, No. 1, 2016, pp. 3-13.

[13] A. Goknil, I. Kurtev, K. Van Den Berg, et al., "Change impact analysis for requirements: A metamodeling approach", Information and Software Technology, Vol. 56, No. 8, 2014, pp. 950-972.

[14] R. M. de Mello, E. Nogueira, M. Schots, et al., "Verification of Software Product Line Artefacts: A Checklist to Support Feature Model Inspections", Journal of Universal Computer Science, Vol. 20, No. 5, 2014, pp. 720-745.

[15] L. Neves, P. Borba, V. Alves, et al., "Safe evolution templates for software product lines", Journal of Systems and Software, Vol. 106, 2015, pp. 42-58.

[16] D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer, "Structuring the modeling space and supporting evolution in software product line engineering", Journal of Systems and Software, Vol. 83, No. 7, 2010, pp. 1108-1122.

[17] L. Neves, L. Teixeira, D. Sena, V. Alves, U. Kulezsa, and P. Borba, "Investigating the safe evolution of software product lines", ACM SIGPLAN Notices, Vol. 47, No. 3, 2012, pp. 33-42.

[18] A. Hunt, and D.Thomas, "The pragmatic programmer: from journeyman to master", Addison-Wesley Professional, 2000.

[19] C. Salinesi and R. Mazo, Defects in Product Line Models and how to Identify them, InTech editions, 2012, p. 50.

[20] S. Apel, D. Batory, C. Kästner, and G. Saake, "Analysis of Software Product Lines", in Feature-Oriented Software Product Lines, Berlin: Springer, 2013, pp. 243-282.

[21] B. Zhang, M. Becker, T. Patzke, et al., "Variability evolution and erosion in industrial product lines: a case study", in 17th International Software Product Line Conference, ACM, 2013, pp. 168-177.

[22] Z. Stephenson, K. Attwood, and J. McDermid, "Product-Line Models to Address Requirements Uncertainty, Volatility and Risk", in Relating Software Requirements and Architectures, Springer Berlin Heidelberg, 2011, pp. 111-131.

[23] G. Lami, S. Gnesi, F. Fabbrini, et al., "An automatic tool for the analysis of natural language requirements", Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Sept. 2004.
[24] H. K. Dam, A. Egyed, M. Winikoff, et al., "Consistent merging of model versions", Journal of Systems and Software, Vol. 112, 2016, pp. 137-155.
[25] J. V. Millo, S. Ramesh, S. N. Krishna, and G. K. Narwane, "Compositional verification of software product lines", in: Johnsen E.B., Petre L. (eds) Integrated Formal Methods (IFM 2013), Lecture Notes in Computer Science, Springer Berlin Heidelberg, Vol. 7940, June 2013, pp 109-123.
[26] K. Wnuk, T. Gorschek, and S. Zahda, "Obsolete software requirements", Information and Software Technology, Vol. 55, No. 6, 2013, pp. 921-940.
[27] I. Hajri, A. Goknil, L. C. Briand, and T. Stephany, "Applying product line use case modeling in an industrial automotive embedded system: Lessons learned and a refined approach", in 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, Sept. 2015, pp. 338-347.
[28] M. Alférez, R. E. Lopez-Herrejon, A. Moreira, et al., "Consistency Checking in Early Software Product Line Specifications-The VCC Approach", Journal of Universal Computer Science, Vol. 20, No. 5, 2014, pp. 640-665.
[29] T. Schmorleiz and R. Lämmel, "Similarity management of 'cloned and owned' variants", in 31st Annual ACM Symposium on Applied Computing, ACM, Apr. 2016, pp. 1466-1471.
[30] I. Jacobson, M. Griss, and P. Jonsson, Software Reuse. Architecture, Process and Organization for Business Success, Addison-Wesley, ISBN: 0-201-92476-5, 1997.
[31] S. Creff, "Une modélisation de la variabilité multidimensionnelle pour une évolution incrémentale des lignes de produits", Doctoral dissertation, University of Rennes 1, 2003.
[32] A. Khtira, A. Benlarabi, and B. El Asri, "Duplication Detection when evolving Feature Models of Software Product Lines", Information Science Journal (ISJ), Vol. 6, No. 4, Oct. 2015, pp. 592-612,
[33] SPL2go, "FeatureAMP", spl2go.cs.ovgu.de/projects/59 [retrieved: December, 2016].
[34] C. Kästner, T. Thüm, G. Saake, et al., "Featureide: A tool framework for feature- oriented software development", in 31st International Conference on Software Engineering (ICSE'09), IEEE, Washington, DC, USA, 2009, pp. 611–614.
[35] E. Alpaydin, Introduction to Machine Learning, London: The MIT Press, 2010. ISBN 978-0-262-01243-0.
[36] A. Khtira, A. Benlarabi, and B. El Asri, "Towards Duplication-Free Feature Models when Evolving Software Product Lines", in 9th International Conference on Software Engineering Advances (ICSEA'14), Oct. 2014, pp. 107-113.
[37] A. Khtira, A. Benlarabi, and B. El Asri, "A Tool Support for Automatic Detection of Duplicate Features during Software Product Lines Evolution", IJCSI International Journal of Computer Science Issues, Vol. 12, No. 4, July 2015, pp. 1-10.
[38] F. Nazir, W. H. Butt, M. W. Anwar, and M. A. K. Khattak, "The applications of natural language processing (NLP) for software requirement engineering-a systematic literature review", in International Conference on Information Science and Applications, Springer, Singapore. March 2017, pp. 485-493.
[39] S. W. Ali, Q. A. Ahmed, I. Shafi,. "Process to enhance the quality of software requirement specification document", in International Conference on Engineering and Emerging Technologies (ICEET), Feb. 2018, pp. 1-7.
[40] D. A. Ferreira and A. R. da Silva, "RSLingo: An information extraction approach toward formal requirements specifications", in Model-Driven Requirements Engineering Workshop (MoDRE), IEEE, Sept. 2012, pp. 39-48.
[41] M. Kamalrudin, J. Grundy, and J. Hosking, "Managing consistency between textual requirements, abstract interactions and Essential Use Cases", in 34th Computer Software and Applications Conference (COMPSAC), IEEE, July 2010, pp. 327-336.
[42] M. Alférez, R. E. Lopez-Herrejon, A. Moreira, et al., "Supporting consistency checking between features and software product line use scenarios", in: Schmid K. (eds) Top Productivity through Software Reuse (ICSR 2011), Lecture Notes in Computer Science, Springer Berlin Heidelberg, Vol. 6727, June 2011, pp. 20-35.
[43] M. Vierhauser, P. Grünbacher, W. Heider, et al., "Applying a consistency checking framework for heterogeneous models and artifacts in industrial product lines", in: France R.B., Kazmeier J., Breu R., Atkinson C. (eds) Model Driven Engineering Languages and Systems (MODELS 2012), Lecture Notes in Computer Science, Springer Berlin Heidelberg, Vol. 7590, 2012, pp. 531-545.
[44] K. Shumaiev and M. Bhat, "Automatic Uncertainty Detection in Software Architecture Documentation", in International Conference on Software Architecture Workshops (ICSAW), April 2017, pp. 216-219.
[45] K. Farias, A. Garcia, and C. Lucena, "Effects of stability on model composition effort: an exploratory study", Software & Systems Modeling, Vol. 13, No. 4, 2014, pp. 1473-1494.
[46] J. Ghofrani, M. Mohseni, A. Bozorgmehr, "A conceptual framework for clone detection using machine learning", in 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI), Dec. 2017, pp. 0810-0817.
[47] J. Rubin, K. Czarnecki, and M. Chechik, "Cloned product variants: from ad-hoc to managed software product lines", International Journal on Software Tools for Technology Transfer, Vol. 17, No. 5, 2015, pp. 627-646.

Amal Khtira received a degree in software engineering from National High School of Computer Science and Systems Analysis (ENSIAS), Mohamed V University, Rabat, in 2008. She is currently a PhD student in the IMS (Models and Systems Engineering) Team of ADMIR Laboratory at ENSIAS. Her research interests include Software Product Line Engineering, Requirements Engineering, Feature Modeling and Software Evolution.

Anissa Benlarabi has a Phd in Software product line evolution issues. She worked with the IMS Team, ADMIR Laboratory at ENSIAS, Mohamed V University, Rabat on many challenges related to software product lines.

Bouchra El Asri is a Professor in the Software Engineering Department and a member of the IMS Team of ADMIR Laboratory at ENSIAS, Mohamed V University, Rabat. Her research interests include Service-Oriented Computing, Model-Driven Engineering, Cloud Computing, Component-Based Systems and Software Product Line Engineering.