# Towards Duplication-Free Feature Models when Evolving Software Product Lines

Amal Khtira, Anissa Benlarabi, Bouchra El Asri
IMS Team, SIME Laboratory, ENSIAS, Mohammed V Souissi University
Rabat, Morocco
amalkhtira@gmail.com, a.benlarabi@gmail.com, elasri@ensias.ma

***Abstract*—Since the emergence of Software Product Line Engineering, the requirements evolution issue has been addressed by many researchers and many approaches have been proposed. However, most studies focused on evolution in domain engineering while application engineering has not received the same attention. During the evolution of a derived product, new features are added or modified in the application model, which may cause many model defects, such as inconsistency and duplication, both in application model and between the latter and the domain model. The aim of this paper is to propose a framework that enables to avoid duplication when evolving software product lines.**

***Keywords*—*Software Product Line; Requirements Evolution; Domain Engineering; Application Engineering; Duplication.***

## I. INTRODUCTION

The Software Product Line Engineering (SPLE) has emerged as a paradigm whose main objective is to develop software applications based on a core platform. The adoption of this approach by companies enables them to reduce time to market, to reduce cost and to produce high quality applications. Another major advantage of the PLE is the reuse of core assets to generate specific applications according to the need of customers.

The SPLE approach consists of two processes, namely, domain engineering and application engineering [1]. During these processes, a number of artefacts are produced which encompass requirements, architecture, components and tests. Domain engineering involves identifying the common and distinct features of all the product line members, creating the design of the system and implementing the reusable components. During application engineering, individual products are derived based on the artefacts of the first process, using some techniques of derivation.

Many issues related to SPLE have been addressed both by researchers and practitioners, such as reusability, product derivation, variability management, etc. The focus of our study will be on SPL evolution. Evolution is defined by Madhavji et al. [2] as "*a process of progressive change and cyclic adaptation over time in terms of the attributes, behavioral properties and relational configuration of some material, abstract, natural or artificial entity or system*". This definition applies to different domains, including software engineering.

In the literature, several studies have dealt with evolution in Software Product Lines (SPLs). Xue et al. [3] presented a method to detect changes that occurred to product features in a family of product variants. In order to support agile SPL evolution, Urli et al. [4] introduces the Composite Feature Model (CFM), which consists of creating small Feature Models (FMs) that corresponds each to a precise domain. Other approaches, such Ahmad et al.'s [5], focused on the extraction of architecture knowledge in order to assess the evolutionary capabilities of a system and to estimate the cost of evolution. Some papers focused on the co-evolution of different elements of SPLs [6].

Based on the literature, we have found that most of the studies addressing software evolution focus on domain engineering, while application engineering has not received the same interest. However, the experience has proven in many industrial contexts that systems continue to change even after the product derivation. This change can be the source of many problems in the product line such as inconsistency and duplication. Indeed, the core assets of the product line and the artefacts of derived products are most of the time maintained by different teams. Moreover, developers under time pressure can forget to refer to the domain model before starting to implement the changes. For these reasons and others, duplication in SPL can easily happen. We consider Duplication the fact of adding to the application model features of the same semantics, which means that they satisfy the same functionality. In this paper, we propose a framework that deals specifically with the problem of duplication when evolving products in application engineering.

The remainder of the paper is structured as follows. Section 2 gives an overview of the background of our study and describes the problem we are dealing with. In Section 3, we present the basic concepts and the overview of the proposed framework. In Section 4, we provide a formalization of the basic concepts before describing the algorithm of deduplication. An application of the framework on a case study is presented in Section 5. Section 6 positions our approach with related works. The paper is concluded in Section 7.

## II. BACKGROUND

In this section, we introduce the background of our study. First, we present the SPLE paradigm, then we give an insight on the problem of duplication when evolving products in application engineering.

### A. Software Product Line Engineering

A SPL is defined by Clements and Northop [7] as "*a set of intensive-software systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*". The main goals of a SPL are to reduce the cost of developing software products, to enhance quality and to promote reusability.

The domain engineering phase of the SPLE framework is responsible for defining the commonality and variability of the applications of the product line. Capturing the common

 ISBN: 978-1-61208-367-4 

features of all the applications increases the reusability of the system, and determining the variant features allows the production of a large number of specific applications that satisfy different needs of customers. In order to document and model variability, many approaches have been proposed. Some of them proposed to integrate the variability in the existing models, such as UML models or feature models (FORM [8]). Pohl et al. [1] preferred to define it separately in a dedicated model, i.e., the orthogonal variability model. Another approach proposed by Salinesi et al. [9] used a constraint-based product line language. When the model is ready, the next step consists of creating the design of the system which contains the software components and their relationships. Those components are then implemented and the code of the product line is generated.

The process of creating a specific product based on a SPL is referred to as product derivation or product instantiation. Product derivation consists of taking a snapshot of the product line by binding variability already defined in the domain engineering and using it as a starting point to develop an individual product. This process is applied during application engineering phase and is responsible for instantiating all the artefacts of the product line, i.e., model, design, components, etc.

### B. Duplication of Features during SPL Evolution

The goal of SPLE is to make an up-front investment to create the platform. Indeed, during domain engineering, the requirements of all the potential applications are captured, and as far as possible, the scenarios of the possible changes have to be predicted and anticipated. The evolution and maintenance of the product line are conducted through several iterations until the platform becomes as stable as possible. As new evolutions arise, the domain artefacts are adapted and refined.

On the one hand, the team responsible for developing and maintaining the product line studies the requirements of each customer and derives specific applications that respond to these requirements. On the other hand, a different team takes in charge the maintenance of each application. Following the logic of SPLE, the derived applications are not supposed to change much, but the experience has shown that this assumption is not always true. In fact, even after the derivation of a specific product, new demands can be received from the customer, either changes to existing features or addition of new ones.

During the maintenance of a product, duplication of knowledge can easily happen when evolving the model, the design or the code. In [10], four categories of duplication are distinguished:

- Imposed duplication: Developers cannot avoid duplication because the technology or the environment seems to impose it.
- Inadvertent duplication: This type of duplication comes about as a result of mistakes in the design. In this case, the developers are not aware of the duplication.
- Impatient duplication: When the time is pressing and deadlines are looming, developers get impatient and tend to take shortcuts by implementing as quick as possible the requirements of customers. In these conditions, duplication is very likely to happen.
- Inter-developer duplication: Different people working on one product can easily duplicate information.

In the context of SPLE, at least the three last categories might occur. Indeed, when a derived application is shipped, developers responsible for maintaining it do not have a clear visibility of the domain model because another team conceived it. Thus, developers of the application may add features which are already satisfied in the domain model and have only to be derived or configured. In addition, under time pressure, developers do not refer to the application model and might add features which are already implemented. To the best of our knowledge, few attempts have dealt with duplication in the application engineering. The aim and contribution of this paper is to provide a framework that helps developers avoid duplication in a SPL when evolving a specific product.

## III. A Framework to Avoid Duplication when Evolving Derived Products

In this section, we first provide a short definition of the basic concepts used in the framework, then we present the overview of the framework.

### A. Basic Concepts

Before going any further, we will give an insight of the basic concepts used in the framework.

**Domain Model:** A domain is a family of related products, and the domain model is the representation of all the different and common features of these products. There are many types of domain models, but the most interesting are the feature model [8] and the variability model [1].

**Application Model:** 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 [1].

**Feature:** A feature is the abstraction of functional or non-functional requirements that help characterize the system and must be implemented, tested, delivered, and maintained [8][11]. A feature is either:

- *Mandatory*: it exists in all products.
- *Optional*: it is not present in all products.
- *Alternative* (One Of): it specializes more general feature; only one option can be chosen from a set of features.
- *Or*: One or more features may be included in the product.

**Variation Point:** Variation points are places in a design or implementation that identify the locations at which variation occurs [12].

**Variant:** It is a single option of a variation point and is related to the latter using a variability dependency [13].

 ISBN: 978-1-61208-367-4 

**Specification:** Requirements specification is a description of the intended behavior of a software product. It contains the details of all the features that have to be implemented during an evolution of the system.

**Specific Variant or Variation point:** We consider a variant or a variation point as specific when they concern a particular need of an application that belongs to the product line (e.g., features related to confidential data, features that need legal authorization).

**Generic Variant or Variation point:** We consider a variant or a variation point as generic if they can be demanded by many applications of the product line (e.g., ergonomic or utility features, non-functional features).

### B. The Framework in a Nutshell

With the large number of features in the SPLs, the manual checking of duplication becomes a complicated and an error-prone task. In order to deal efficiently with the problem of duplication during the evolution of derived products, we propose the framework depicted in Figure 1 as an attempt to set an automated deduplication tool.

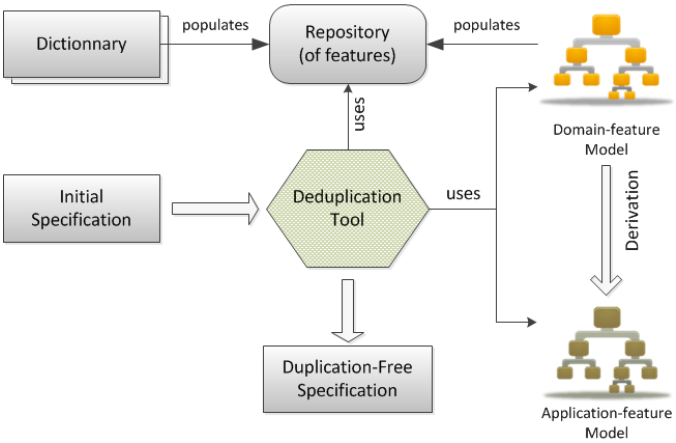


Figure 1. The overview of the framework.

**Initial Specification:** In this framework, we take as an input the specification of a new evolution related to a derived product. This specification contains the requirements that have to be implemented in this specific product. To use these requirements, we need to express them as features using the FODA feature model [14]. In the context of our framework, we consider that a feature is the association of a variation point and a variant. To create the feature model, we opt for FeatureIDE [15]. This tool enables to graphically create the feature model and the associated XML is generated automatically.

**Domain and Application Models:** The main prerequisites of the framework are the domain model and the application model. To create these models, we use the FeatureIDE tool in order to generate the sources in the form of XML files.

**Repository:** The repository contains the features of the domain model and also the set of all the possible synonyms and alternatives for the concepts used in the product line. The elements of the repository are defined using the Resource Description Framework (RDF). RDF [16] is a W3C recommendation that supports semantic interoperability between different resources on the web.

**Deduplication Tool:** This tool contains a set of algorithms of features verification. In this paper, we focus on the algorithm of deduplication. Before describing the algorithm, we need to define some predicates.

*Equivalence:* We consider that a variation point (resp. a variant) is equivalent to another variation point (resp. variant) if they both implement the same functionality, which means that they have the same semantics. We define the function *Equiv* which can take three values:

$$Equiv(x) = x_0 \Rightarrow \begin{cases} x_0 = x \\ x_0 \text{ is equivalent to } x \text{ and } x_0 \neq x \\ x_0 \in \emptyset \end{cases}$$

Example: The variant "On-line Sales" associated to the variation point "Sales" is equivalent to the new variant "e-sales" (cf. Section 5).

*Duplication:* We consider that a feature of the specification is duplicated if the associated variation point and variant have equivalents in the application model or the domain model.

The aim of the algorithm is thus to verify the non-duplication of all the features of the initial specification in order to generate a new correct specification. Indeed, for each feature of the initial specification, the algorithm verifies whether the associated variation point and variant have equivalents in the domain model and the application model. The detection of equivalence is carried out based on the *Repository* content. The steps of the algorithm are explained in details in Section 4.

**Duplication-Free Specification:** The output of the framework is a specification that does not contain features causing duplication in the SPL.

## IV. An Algorithm for Duplication-Free SPL

In this section, we provide the formalization of the basic concepts used in the framework, then we describe the deduplication algorithm.

### A. Formalizing the Basic Concepts

Prior to explaining the algorithm, a certain number of predicates must be defined. We denote by D the domain model. PD is the set of variation points of D, and VD is the set of variants of D.

$$PD = \{PD_1, PD_2, \ldots, PD_p\}$$
$$VD = \{VD_1, VD_2, \ldots, VD_q\}$$

Similarly, we denote by A the application model of a derived application. PA is the set of variation points of A, and VA is the set of variants of A.

$$PA = \{PA_1, PA_2, \ldots, PA_s\} \text{ with } s \leq p$$
$$VA = \{VA_1, VA_2, \ldots, VA_t\} \text{ with } t \leq q$$

Thus:

$$PA \subseteq PD \quad \text{and} \quad VA \subseteq VD$$

 ISBN: 978-1-61208-367-4 

We denote by $S_0$ the specification of an evolution, i.e., the set of new features to implement.

$$S_0 = \{F_1, F_2, \ldots, F_n\}$$

P and V are, respectively, the sets of variation points and variants, which correspond to the features defined in $S_0$.

$$P = \{P_1, P_2, \ldots, P_v\},\ V = \{V_1, V_2, \ldots, V_u\}$$

It has to be noted that P and V are not subsets of PA and VA. In our framework, we consider that a feature in $S_0$ can be defined as follows: $F_k = (P_i, V_j)$.

### B. The Deduplication Algorithm

To implement a new evolution, we propose an evolutionary framework that verifies whether a feature of the specification is duplicated in the model and generates in the end a new verified specification. Figure 2 shows the relationship between the specification of an iteration k-1, the feature $F_k$ and the resulting specification.

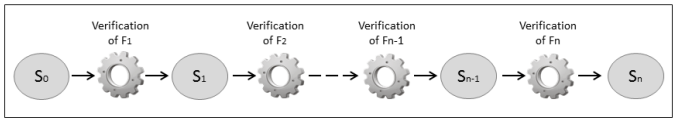


Figure 2. Relationship between $S_{k-1}$, $F_k$ and $S_k$.

To verify whether a feature $F_k$ is duplicated or not, the algorithm distinguishes six different cases in each iteration (k-1). These cases are represented in Figure 3.

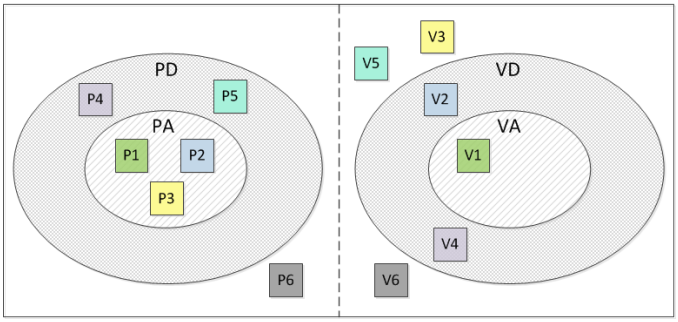


Figure 3. The figure shows the different cases of variants and variation points: Every pair ($P_i$,$V_i$) corresponds to a case i.

**Case 1:** The variation point associated to the feature $F_k$ has an equivalent in $PA$ and the variant has an equivalent in $VA$. Consequently, the feature is duplicated and must be removed from the specification, but the domain model and the application model do not change.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PA) \wedge (Equiv(V_j) \in VA)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

**Case 2:** The feature $F_k$ consists of adding a new variant to a variation point that has an equivalent in $PA$, where an equivalent of the variant exists already in $VD$. Consequently, the feature must be removed from the specification.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PA) \wedge (Equiv(V_j) \in VD \setminus VA)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

The variant $V_j$ must be added to the application model.

**Case 3:** The feature $F_k$ requires adding a new variant to a variation point that has an equivalent in $PA$, and the variant does not have an equivalent in $VD$. In this case, we assume that $V = V_S \cup V_G$ where $V_S$ is the set of variants of V that are specific to the business of the application, and $V_G$ is the set of variants of V that are generic. We distinguish two sub-cases:

*Case 3.1:* If the variant in question belongs to $V_S$. In this case, the specification does not change and the feature is added directly to the application model.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PA) \wedge\ (Equiv(V_j) = \emptyset) \wedge (V_j \in V_S)$$
$$\Rightarrow \quad S_k = S_{k-1}$$

*Case 3.2:* If the variant in question belongs to $V_G$. In this case, the feature is removed from the specification.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PA) \wedge\ (Equiv(V_j) = \emptyset) \wedge (V_j \in V_G)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

The variant $V_j$ is added to the domain model and then to the application model.

**Case 4:** The variation point related to $F_k$ has an equivalent in $PD$ but not in $PA$, and the variant has an equivalent in $VD$ but not in $VA$. Consequently, the feature must be removed from the specification.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PD \backslash PA) \wedge (Equiv(V_j) \in VD \backslash VA)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

The variation point and the variant must be derived from the domain model and added to the application model.

**Case 5:** The variation point related to $F_k$ has an equivalent in $PD$ but the variant is new. In this case, the feature is removed from the specification.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) \in PD \setminus PA) \wedge (Equiv(V_j) = \emptyset)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

The variant is added to the domain model then to the application model.

**Case 6:** The variation point related to $F_k$ does not have an equivalent in $PD$, and the variant does not have an equivalent in $VD$. In this case, we assume that $P = P_S \cup P_G$ where $P_S$ is the set of variation points of P that are specific to the business of the application, and $P_G$ is the set of variation points of P that are generic. We distinguish two sub-cases:

*Case 6.1:* If the variation point belongs to $P_S$, the specification does not change and the feature is added directly to the application model.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) = \emptyset) \wedge (Equiv(P_i) \in P_S)$$
$$\Rightarrow \quad S_k = S_{k-1}$$

*Case 6.2:* If the variation point belongs to $P_G$, the feature is removed from the specification, added to the domain model then to the application model.

$$(F_k = (P_i, V_j)) \wedge (Equiv(P_i) = \emptyset) \wedge (Equiv(P_i) \in P_G)$$
$$\Rightarrow \quad S_k = S_{k-1} \setminus \{F_k\}$$

 ISBN: 978-1-61208-367-4 

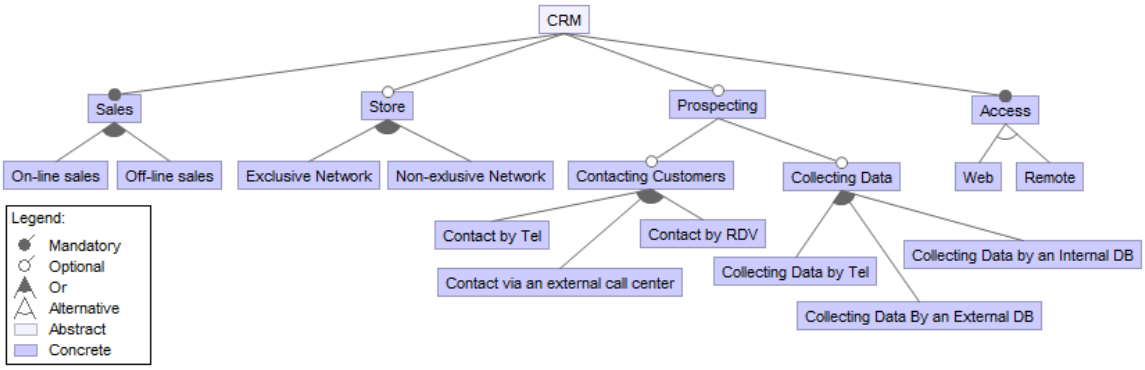


Figure 4. The Domain Feature Model of the CRM.

**Result:**

In the end, when all the verifications are carried out for all the features of $S_0$, we obtain $S_N$. This new specification is a duplication-free specification which contains only the features that have to be implemented directly in the application model.

## V. Case Study

To illustrate our framework, we propose a part of a feature model of a CRM (Customer Relationship Management). The Figure 4 depicts the domain feature model of the CRM. The feature model is created using FeatureIDE as mentioned in Section 3.

We consider a derived application with the feature model depicted in Figure 5. Based on the XML source, we consider that the tags "and" correspond to variation points and the tags "feature" correspond to variants.

```
<featureModel chosenLayoutAlgorithm="1">
    <struct>
        <and abstract="true" mandatory="true" name="CRM1">
            <or mandatory="true" name="Sales">
                <feature mandatory="true" name="On-line sales"/>
                <feature mandatory="true" name="Off-line sales"/>
            </or>
            <or name="Store">
                <feature mandatory="true" name="Exclusive network"/>
                <feature mandatory="true" name="Non-exclusive network"/>
            </or>
            <and name="Prospecting">
                <or mandatory="true" name="Collecting Data">
                    <feature mandatory="true" name="Collecting Data by Tel"/>
                    <feature mandatory="true" name="Collecting Data by an external DB"/>
                    <feature mandatory="true" name="Collecting Data by an internal DB"/>
                </or>
            </and>
            <and mandatory="true" name="Access">
                <feature name="Web"/>
            </and>
        </and>
    </struct>
    <constraints/>
    <calculations Auto="true" Constraints="true" Features="true" Redundant="true"/>
    <comments/>
    <featureOrder userDefined="false"/>
</featureModel>
```

Figure 5. The Application Feature Model in XML.

During a new evolution of this application, a number of requirements are demanded by the customer. In our case study, we will take into account only the following requirements:

1) Users can use the application in a disconnected mode.
2) The sector header can contact customers by setting up an appointment.
3) The application must enable users to follow the activity of competitors' shops.
4) The system must manage e-sales.
5) The sector header can generate summary reports in Excel.

We distinguish the following features:

- $F_1$= ($P_1$, $V_1$)= (Connexion, Disconnected mode)
- $F_2$= ($P_2$, $V_2$)= (Contacting Customers, Appointment)
- $F_3$= ($P_3$, $V_3$)= (Shop, Competitor's store)
- $F_4$= ($P_4$, $V_4$)= (Sales, e-sales)
- $F_5$= ($P_5$, $V_5$)= (Reporting, Excel Report)

with $S_0 = \{F_1, F_2, F_3, F_4, F_5\}$.

The list of equivalents of the variation points and variants related to these features is described in Table 1.

TABLE I. The Equivalents of Features

| x (VP or V) | Equiv(x) |
|---|---|
| $P_1$ | Access |
| $V_1$ | Remote |
| $P_2$ | Contacting Customers |
| $V_2$ | RDV |
| $P_3$ | Store |
| $V_3$ | $\emptyset$ |
| $P_4$ | Sales |
| $V_4$ | on-line sales |
| $P_5$ | $\emptyset$ |
| $V_5$ | $\emptyset$ |

After applying the verification algorithm to this specification, we came up with the results of Table 2.

*: $V_3$ is specific to this application, because following the activity of competitors' stores requires a legal authorization, which is not possible for all companies.

**: Generating reports (e.g., in Excel or Word) can be considered as a generic feature, because it can be demanded by other applications.

 ISBN: 978-1-61208-367-4 

TABLE II. The Results of the Verification Algorithm

| Feature | Verification of VP and V | Case | Result |
|---|---|---|---|
| $F_1$ | $(Equiv(P_1) \in PA) \wedge$ $(Equiv(V_1) \in VD \setminus VA)$ | Case 2 | $S_1 = S_0 \setminus \{F_1\}$ |
| $F_2$ | $(Equiv(P_2) \in PD \setminus PA) \wedge$ $(Equiv(V_2) \in VD \setminus VA)$ | Case 4 | $S_2 = S_1 \setminus \{F_2\}$ |
| $F_3$ | $(Equiv(P_3) \in PA) \wedge$ $(Equiv(V_3) = \emptyset) \wedge$ $(V_3 \in V_S)^*$ | Case 3.1 | $S_3 = S_2$ |
| $F_4$ | $(Equiv(P_4) \in PA)$ $(Equiv(V_4) \in VA)$ | Case 1 | $S_4 = S_3 \setminus \{F_4\}$ |
| $F_5$ | $(Equiv(P_5) = \emptyset) \wedge$ $(P_5 \in P_G)^{**}$ | Case 6.2 | $S_5 = S_4 \setminus \{F_5\}$ |

In the end, we obtain $S_4 = \{F_3\}$, which means that developers have to implement only the feature $F_3$ in the application model. The other features are all sent to the team maintaining the domain model in order to add the new features and bind the existing ones then re-derive the model.

## VI. Related Work

In this section, we provide an overview of the studies most relevant to our work by categorizing them according to the issues addressed in this paper.

*Evolution of feature and variability models:* In order to reduce complexity and improve the maintenance of variability in large-scale product lines, Dhungana et al. [17] proposes to organize product lines as a set of interrelated model fragments that define the variability of particular parts of the system, and presents a support to semi-automatically merge the different fragments into a complete variability model. The same approach is proposed by Pleuss et al. [18] for feature models. Voelter et al. [19] proposes an approach which consists of separating features in models and composing them by aspect-oriented composition techniques. Cordy et al. [20] defines two particular types of features, regulative features and conservative features, and explains how the addition of these features to the SPL can reduce the overhead of model-checking. The common denominator of the cited studies is that they all consider evolution in domain engineering, while our approach deals with evolution in application engineering.

*Model Defects in SPL:* Several papers in the literature have addressed model defects caused by SPL Evolution. For example, Guo and Wang [21] proposes to limit the consistency maintenance to the part of the feature model that is affected by the requested change instead of the whole feature model. Romero et al. [22] introduces SPLEmma, a generic evolution framework that enables the validation of controlled SPL evolution by following a Model Driven Engineering approach. This study focused on three main challenges: SPL consistency during evolution, the impact on the family of products and SPL heterogeneity. In [23], Mazo provides a classification of different verification criteria of the product line model that he categorizes into four families: expressiveness criteria, consistency criteria, error-prone criteria and redundancy-free criteria. Redundancy can easily be confused with Duplication, but it is completely different, because Mazo focuses on redundancy of dependencies and not redundancy of features. The same study defines also different conformance checking criteria, among which two features should not have the same name in the same model. This is also different from our approach which is based on equivalence and not only equality of features.

*Evolution in application engineering:* Carbon et al. [24] presents an empirical study which consists of adapting the planning game to the product line context in order to introduce a lightweight feedback process from application to family engineering at Testo, but it does not provide a general approach that is applicable to all SPLs. Hallsteinsen et al. [25] introduces the concept of Dynamic Software Product Lines (DSPL), which provide mechanisms for binding variation points at runtime in order to keep up with fluctuations in user needs. However, this approach does not explain in details how the variability is managed between application and domain engineering. Thao [26] proposes a versioning system to support the evolution of product line and change propagation between core assets and derived products. But this study also does not provide a method to manage features in application engineering. Our approach is different because it provides a feature-oriented approach to manage the evolution of derived products in a way that insures non-duplication in the SPL feature models.

## VII. Conclusion and Future Work

In the literature, many studies have addressed the evolution in SPLs, but the majority of them focused on the domain engineering phase, when application engineering has not been thoroughly discussed. Based on industrial experience, products are also likely to evolve even after their derivation, and this evolution can cause many problems especially duplication in the different artefacts of the product line. In this paper, we provided a framework that deals specifically with duplication in feature models. This framework uses a repository that contains the set of domain features and alternatives of the different concepts of the product line at the aim of verifying the non-duplication of all the features of a new specification. To illustrate the framework, we applied it to a case study from the CRM field. In a future work, we intend to initiate a tool based on the framework architecture, whose objective is to automatize the algorithm of verification and to generate automatically a duplication-free specification that contains only the relevant features to implement.

## 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] N. H. Madhavji, J. Fernandez-Ramil, and D. Perry, Software Evolution and Feedback: Theory and Practice, John Wiley & Sons, 2006, ISBN 978-0-470-87180-5.

[3] Y. Xue, Z. Xing, and S. Jarzabek, "Understanding feature evolution in a family of product variants," Proc. WCRE'10, IEEE, Oct. 2010, pp. 109-118.

[4] S. Urli, M. Blay-Fornarino, P. Collet, and S. Mosser, "Using composite feature models to support agile software product line evolution," Proc. 6th International Workshop on Models and Evolution, ACM, Oct. 2012, pp. 21-26.

[5] A. Ahmad, P. Jamshidi, and C. Pahl, "A Framework for Acquisition and Application of Software Architecture Evolution Knowledge," ACM SIGSOFT Software Engineering Notes, vol. 38, no. 5, Sept. 2013, pp. 65-71.

 ISBN: 978-1-61208-367-4 

[6] C. Seidl, F. Heidenreich, and U. Assmann, "Co-evolution of models and feature mapping in Software Product Lines," Proc. SPLC'12, ACM, New York, USA, 2012, Vol. 1, pp. 76-85.

[7] P. Clements and L. Northop, Software Product Lines - Practices and Patterns, Boston: Addison-Wesley, 2002.

[8] K. C. Kang et al., "FORM: A feature-oriented reuse method with domain-specific reference architectures," Annals of Software Engineering, vol. 5, no. 1, 1998, pp. 143-168.

[9] C. Salinesi, R. Mazo, O. Djebbi, D. Diaz, A. Lora-Michiels, "Constraints: the Core of Product Line Engineering," In. RCIS'11, IEEE, Guadeloupe-French West Indies, France, May 19-21, 2011, pp. 1-10.

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

[11] J. Bosch, Design and use of software architectures: adopting and evolving a product-line approach, New York, USA: ACM Press/Addison-Wesley, 2000.

[12] I. Jacobson, M. Griss, and P. Jonsson, Software Reuse. Architecture, Process and Organization for Business Success, Addison-Wesley, ISBN: 0-201-92476-5, 1997.

[13] S. Creff, "Une modélisation de la variabilité multidimensionnelle pour une évolution incrémentale des lignes de produits," Doctoral dissertation, University of Rennes 1, 2003.

[14] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Technical Report CMU/SEI-90-TR-21, Carnegie Mellon University, Software Engineering Institute, Nov. 1990.

[15] C. Kastner et al., "FeatureIDE: A Tool Framework for Feature-Oriented Software Development," Proc. The 31st International Conference on Software Engineering, 2009, pp. 611-614.

[16] O. Lassila, R. R. Swick, Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 Feb. 1999, http: //www.w3.org/TR/1999/REC-rdf-syntax-19990222/ [retrieved: August, 2014].

[17] 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.

[18] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski, "Model-driven support for product line evolution on feature level," Journal of Systems and Software, vol. 85, no. 10, 2012, pp. 2261-2274.

[19] M. Voelter and I. Groher, "Product line implementation using aspect-oriented and model-driven software development," Proc. SPLC'07, IEEE, Sept. 2007, pp. 233-242.

[20] M. Cordy, A. Classen, P. Y. Schobbens, P. Heymans, and A. Legay, "Managing evolution in software product lines: A model-checking perspective," Proc. 6th International Workshop on Variability Modeling of Software-Intensive Systems, ACM, Jan. 2012, pp. 183-191.

[21] J. Guo, and Y. Wang, "Towards consistent evolution of feature models," In. Software Product Lines: Going Beyond, Springer Berlin Heidelberg, 2010, pp. 451-455.

[22] D. Romero et al., "SPLEMMA: a generic framework for controlled-evolution of software product lines," Proc. 17th International Software Product Line Conference co-located workshops, ACM, 2013, pp. 59-66.

[23] R. Mazo, "A generic approach for automated verification of product line models," Ph.D. thesis, Pantheon-Sorbonne University, 2011.

[24] R. Carbon, J. Knodel, D. Muthig, and G. Meier, "Providing feedback from application to family engineering-the product line planning game at the testo ag," Proc. SPLC'08, IEEE, Sept. 2008, pp. 180-189.

[25] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," Computer, vol. 41, no. 4, 2008, pp. 93-95.

[26] C. Thao, "Managing evolution of software product line," Proc. 34th ICSE'12, IEEE, Jun. 2012, pp. 1619-1621.

 ISBN: 978-1-61208-367-4 