IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 28, 2018, accepted January 26, 2019, date of publication February 11, 2019, date of current version April 8, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2897405

A Method of Automatically Evolving Feature Models of Software Product Lines

JUNQI REN[1,2], LEI LIU[1], PENG ZHANG[1,2], AND WENBO ZHOU[1,2]

[1]College of Computer Science and Technology, Jilin University, Changchun 130012, China
[2]Key Laboratory of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun 130012, China

Corresponding author: Peng Zhang (zhangpengccst@jlu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61300049, Grant 61502197, and Grant61503044, in part by the Natural Science Research Foundation of Jilin Province of China under Grant 20190201193JC, Grant 20180101053JC, and Grant 20150101054JC, in part by the China Postdoctoral Science Foundation under Grant 2016M591482 and Grant 2016M590254, and in part by the Graduate Innovation Fund of Jilin University under Grant 101832018C025.

ABSTRACT A software product line is a kind of software, which generates software products with similar functionality by reusing assets. A feature model is extracted from requirements documents to represent each functional module of a software product line and the relationship among functionality. During the evolution of a software product line, a feature model always needs to be rebuilt. The rebuilding process cannot guarantee the correctness of the reconstructed feature model. Therefore, we propose a method of automatically evolving feature models of software product lines with evolutionary requirements to solve the repeated reconstruction and reduce errors in reconstruction. In the method, a formal model of feature models is constructed by communication membrane calculus, then the formal evolutionary requirements can change the formal model automatically, finally the changed formal model is mapped to the reconstructed feature model. The several experiments are carried out by open feature models, and the results of experiments not only verify the effectiveness of the proposed method but also show the method can be used to test whether the software product line evolves according to the requirements. The method can automatically change feature models of software product lines to avoid mistakes when people modify the feature model.

INDEX TERMS Software product line, feature model, communication membrane calculus, evolutionary requirement.

I. INTRODUCTION

A software product line (SPL) is a kind of software that generates software products family through the reuse of assets according to the customers' needs [9], [29], [35]. The software product family is a series of software products produced by SPL having the same basic functionality and similar other functionality. This approach of developing software through reuse of asset saves development time while reducing the upfront cost of the product and reducing development risk [1], [3], [7], [21], [33]. The design of SPL in industrial needs to meet the increasing needs of users, which requires SPL not only to have a long life cycle but also to have a stable architecture and a mechanism to respond to changes [15]. However, with the maturity of SPL technology, the scale of SPL and its functionality is increasing gradually, at the same time the difficulty and risk of evolving SPL are rising gradually. Compare with the traditional software evolution, SPL also has three ways of evolution: adding, deleting and modifying. But there are some differences between the SPL evolution and the evolution of traditional software. In the SPL evolution, we can add or delete a software with some functionality related to SPL into or from the original product lines, instead of adding or deleting some functionality in traditional software [22]. This unique way of adding and deleting makes the evolution of SPL more massive and difficult.

A feature model is a tool that can describe software functionality and functionality structure [14]. A feature model has a wide range of applications in SPL. It can not only describe the functional structure of SPL but also describe the structure of a software product generated by SPL. Feature models play an irreplaceable role in SPL and an integral role in the SPL evolution. Heradio et al. through access to 175 articles about the SPL from 1995 to 2014, get an important conclusion about the feature model: Feature modeling has been the most important topic in the whole SPL research area [18].

The associate editor coordinating the review of this manuscript and approving it for publication was Hailong Sun.

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

And Montalvillo et al. through access to 107 articles about the evolution of the SPL, get another critical conclusion about the feature model: in the past 15 years, the feature model is the most critical topic in the research of SPL evolution [25].

At present, most of the articles about SPL evolution are using the feature model of SPL to determine the influence scope of evolution. And then ensuring the correctness of evolution [25]. In order to ensure the availability of feature models for the next evolution after using a feature model to evolve SPL, it needs to change the feature model accordingly after each of the evolving SPL assets. When the SPL evolves, it needs domain experts to reconstruct the feature model by the existing modeling standards. This process relies on the experience of modelers and their understanding of requirements. It cannot guarantee the correctness of the reconstructed feature model. Therefore, we present a method of automatically evolving feature models of SPL (AutoEvoSPL) by a kind of formal method and evolutionary requirements. This formal method can automatically determine the range of evolution and automatically change feature models. It can avoid the error of artificial modification of the feature model. In AutoEvoSPL, we formalize a feature model and the evolution process of a feature model by Communication Membrane Calculus (CMC) [31]. We will use the key information extracted from the evolutionary requirements to change the formalized model. And then it will generate a new feature model automatically which meets the evolutionary requirements.

The method generates a new feature model automatically by evolutionary requirements and CMC. In this article, we will use CMC to model feature model of SPL, and give the mutual mapping between CMC and a feature model of SPL. At the same time, we will present a method to describe the evolutionary requirements of SPL by CMC. And then, we will use the evolutionary requirements described by CMC to change the feature model described by CMC and generate a new feature model described by CMC. Finally, the new feature model described by CMC will be mapped into a feature model of SPL. During the process, getting feature models and requirements are semiautomatic, the other steps are got automatically. We have designed a tool named AutoEvoFM to implement the method of AutoEvoSPL. To verify the effectiveness of the proposed method, several experiments are designed. The data are collected from SPLOT and website of Mobile Media.

The remainder of the paper is structured as follows: We introduce the basic definition of a feature model of SPL and CMC in section II. In section III, we introduce the framework of automatic evolution of feature model. We give the mapping between feature models and CMC in section IV. Subsequently, the mapping process from evolutionary requirements to CMC is given, and the requirements mapping methods are shown in section V. We show the process of splitting an evolutionary need, the method of evaluating the reasonability of need, and the process of evolving a feature model in section VI. And the implementation of AutoEvoFM is shown in section VII. And then, we verify the effectiveness of AutoEvoFM by the experiments based on the several public feature models in section VIII. Finally, we give the related works in section IX and conclude the paper with final thoughts (Section X). And we also give an example to explain the way of using our method at the end of section IV, V and VI.

II. BACKGROUND

A. FEATURE MODEL

In recent years, the SPL is mainly used to reduce the amount of work, reduce costs, improve quality and speed up the development of software products by a series of reusable components. A product line describes the relationships among modules such as exclusion, dependencies and so on through feature modeling. Each module is a feature, and each feature represents a corresponding functionality of SPL [19]. A product is a series of features with code. And a feature model is a combination of a series of effective products with restrictions and relations. Feature models are used in many large companies such as Boeing, Siemens, and Toshiba Co [19], [23].

Feature models can use a tree structure to describe the relationship of features. In the tree structure, each node represents a feature, and different connections of nodes represent different relationships of features.

Fig. 1 describes a feature model of the global positioning system (GPS) by a tree diagram [4]. In the tree diagram, root node is the name of GPS and the sub-nodes represent the functionality contained in GPS.

The relationship between the parent node and the sub-node can be divided into the following four types:

1) Mandatory. The node is the necessary node for its parent node. It indicates that in all products of the SPL, this feature must be contained if its parent feature is selected. In Fig. 1, Routing and Interface are functionality that GPS must provide.
2) Optional. The node is an optional node for its parent node. This means that in products of the SPL, if you select its parent feature, you can select this feature or not. In Fig. 1, Traffic avoiding is an optional feature of GPS.
3) Alternative. The parent node has only one sub-node. This means that if the parent feature is selected in products, only one of the features among these alternative features can appear in the product. In Fig. 1, Screen can select only one sub-feature between Touch and LCD.
4) Or. The parent node has at least one sub-node. It indicates that in products of SPL, once its parent feature is selected, one or more of or features can be in the products. In Fig. 1, at least one functionality between 3D map and Auto-rerouting must be selected, when Routing appears in products.

Besides, the symbols at the bottom of the tree represent the constraints between two features, and the constraints contain two kinds of relationships:

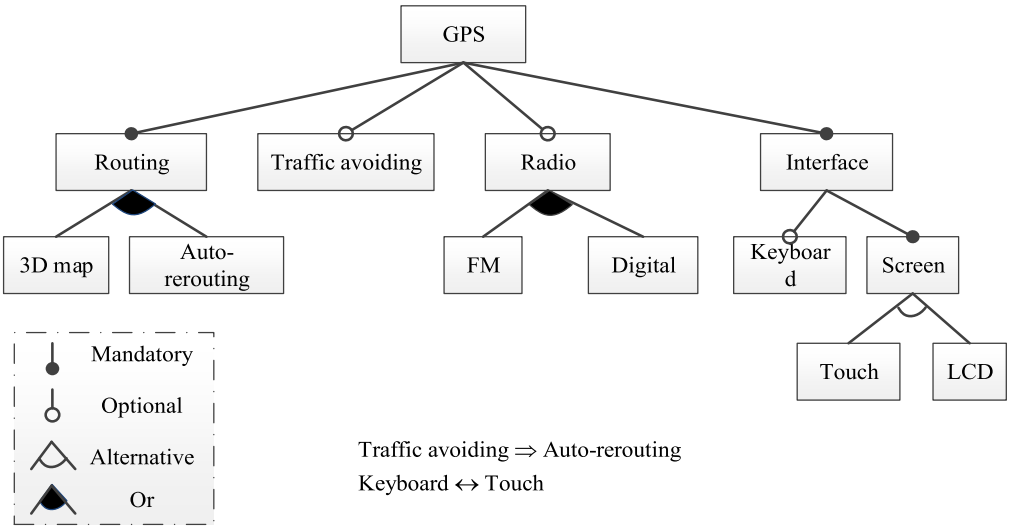

FIGURE 1. A feature model of the GPS.

1) Requires. In products of SPL, if we select feature A, feature B must be select. $A \Rightarrow B$ is used to represent the requires relationship between feature A and feature B, and arrows point to feature B. $\neg A \vee B$ is the mathematical description of requires, where F is the feature set of SPL, and $A \in F$, $B \in F$. In Fig. 1, Traffic avoiding and Auto-rerouting have the relationship of requires.
2) Excludes. One of feature A and feature B exists in products of SPL. $A \leftrightarrow B$ is used to represent the excludes relationship. The two arrows point to feature A and feature B respectively. $\neg A \vee \neg B$ is the mathematical description of the excludes, where F is the feature set of SPL, and $A \in F, B \in F$. In Fig. 1, Touch and Keyboard are a pair of excludes.

This article focuses on how to use the evolutionary requirements to evolve a feature model of SPL. We will give a formal definition in section IV to describe feature model.

B. COMMUNICATION MEMBRANE CALCULUS

CMC is a formal method suitable for modeling evolutionary problems [31]. It comes from membrane computing. It can describe the structure and relationship of modules. And it also can describe the communication process among modules. So it is suitable for modeling the structure of feature models and the evolution of feature models. CMC can automatically determine the range of evolution and automatically change feature models to avoid the error of artificial modification of feature models.

Definition 1 (Communication Membrane Calculus): CMC is a multivariate set

$\prod = (V, \mu, O_1, ..., O_n, (R_1, \rho_1), ..., (R_n, \rho_n))$, where

1) V is a collection of all objects that the system contains,
2) μ is the structure of membranes,
3) $O_i, 1 \leq i \leq n$ is a multiset of objects within membrane i,
4) $(R_i, \rho_i), 1 \leq i \leq n$ are sets of reaction rules and priority of the reaction rules in membrane i. R_i is the rule set in the membrane. ρ_i is a priority set of rules.

In CMC, objects are represented as $a : C$, a is the name of objects, and C is the class of objects. The rules are represented as $A \rightarrow B$, it means change A to B, $\rightarrow$ is an identifier of rules. And the priority between two rules is represented by $>$. Such as $\rho_i = \{a : O \rightarrow b : O > a : O \rightarrow a : R\}$, it indicates that the reaction rule $a : O \rightarrow b : O$ take precedence over reaction rule $a : O \rightarrow a : R$.

And CMC has 8 kinds of reaction rules:

1) Object changing rules: $u : C \rightarrow v : C$, that indicates the change in the name or number of objects.
2) Object transferring rules: $u : C \rightarrow v : C_{to\_k}$, indicating that objects are passed to the membrane k.
3) Membrane dissolution rules: $u : O \rightarrow \sigma$, indicating that the objects' membrane is dissolved. At the same time, the objects, rules and sub-membrane in the membrane are deleted.
4) Membrane creation rules: $u : O \rightarrow [_m\mu, O_1, ..., O_n, (R_1, \rho_1), ..., (R_n, \rho_n)]_m$, indicating the creation of a new membrane m in object's membrane.
5) Membrane transferring rules: $u : O \rightarrow \lambda_k$, showing that the whole membrane of the object moves into membrane k and becomes the sub-membrane of membrane k.
6) Replication rules: $u : R \rightarrow (r_i, \rho_i)_{copy}$, indicating that rules (r_i, ρ_i) are added to the membrane where the object is located.
7) Transferring rules: $u : R \rightarrow (r_i, \rho_i)_{in\_k}$, showing that the rules (r_i, ρ_i) pass into membrane k and become rules of membrane k.
8) Deletion rules: $u : R \rightarrow (r_i, \rho_i)_{delete}$, indicating the deletion of rules (r_i, ρ_i) in the object's membrane.

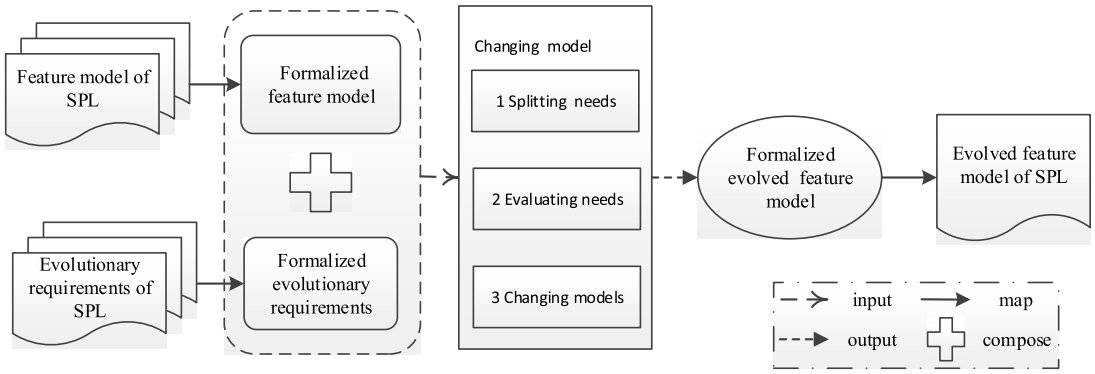

FIGURE 2. Framework of AutoEvoSPL.

The details of CMC can be found in our previous article: A calculus for modeling the process of evolution-communication membrane calculus [31]. Using Definition 1, the static structure of a feature model and the process of evolving feature models can be described completely.

III. THE FRAMEWORK OF AutoEvoSPL

An SPL's feature model can guide the evolution of SPL, so we should pay attention to the evolution of feature models when the SPL evolved. In this paper, CMC is used to model the feature models and the evolution process of feature models, to achieve automatic evolution of feature models and obtain new feature models. Fig. 2 presents the framework of the method of evolving SPL's feature model automatically by CMC and evolutionary requirements. This framework is the technical framework of our method.

Fig. 2 shows the framework of the method proposed in our article. In AutoEvoSPL, there are two input to start the process of evolving a feature model. They are a feature model and evolutionary requirements. Firstly, the feature model of SPL are mapped into formalized feature model, and the evolutionary requirements of SPL are changed to formalized evolutionary requirements, the formalized feature model and evolutionary requirements are composed as the input of next step. And then, using steps of splitting needs, evaluating needs and evolving models, formalized evolved feature model is got as the output of the multiple changes of model. Finally, the formalized evolved feature model is changed to evolved feature model of SPL. And the evolved feature model of SPL is a new SPL feature model that meets evolutionary requirements.

Firstly, using mapping rules, feature model of SPL can be changed to the formalized feature model named FEvoFM which is described by CMC, and evolved FEvoFM can be changed to evolved feature model of SPL, the mapping between feature models and FEvoFM is shown in Section IV. And evolutionary requirements can be changed to formalize evolutionary requirements named AutoEvoRe by our method, and AutoEvoRe is made up of AutoEvoNeeds, the process of getting AutoEvoNeed is shown in Section V. Then the formalized feature model can be changed by every need from the AutoEvoRe. The evolved FEvoFM is got after changing the original FEvoFM by every need in AutoEvoRe. The process of changing FEvoFM is shown in Section VI. Finally, the evolved FEvoFM changes to a feature model of SPL by the mapping rules in Section IV.

AutoEvoRe is the set of AutoEvoNeeds. An evolutionary requirement contains many small changes, and every change will evolve FEvoFM once. And then the evolved FEvoFM is the final result of changing the initial FEvoFM by every AutoEvoNeed in AutoEvoRe. And the definition of an AutoEvoNeed is in Section V.

IV. FEvoFM

With the development of SPL, in addition to the parent-child relationships between two features, a feature model can also contain constraints between two features. When formalize feature model by formal methods, we should not only consider the type of features but also discuss how to describe the relations among features. This section gives the mapping between feature models and CMC, it will change a feature model to FEvoFM, and after changing FEvoFM with the evolutionary requirements, the changed FEvoFM can be mapped to a feature model too.

A. ABSTRACTION OF SPL'S REATURE MODELS

In order to facilitate the formal modeling of SPL's feature models, we need give an abstract definition of the feature models.

Definition 2 (ThrTFM): ThrTFM is a three tuple (FN, FR, FC) to represent a SPL's feature model Fn, where FN represents a set of feature name that contains the feature type, FR represents a set of relationships between two features, and FC represents a set of the constraints in the feature model.

The element in FN is a symbol with superscript, such as F_n^M, F_n represents the name of a feature, the corner represents the type of a feature. There are four types of features: mandatory (M), optional (O), alternative (A), and or (R). In the

following, a feature with no superscript means it can be any features.

In a feature model, if feature F_{n0} is the root node of feature F_{n1}, then F_{n0} is called the parent feature of F_{n1} and F_{n1} is the sub-feature of F_{n0}. F_{n1} can be a feature or several features with a parent feature named combination of features.

An element of FR represents a relationship of parallel or inclusion between two arbitrary features, such as $F_{n0} \supset F_{n1}$, $F_{n1} \cup F_{n2}$, $F_{n0} \supset F_{n2}$. If F_{n1} is a sub-feature of another feature F_{n0}, we call that F_{n0} and F_{n1} have inclusion relationship, denoted as $F_{n0} \supset F_{n1}$. For three features F_{n0}, F_{n1} and F_{n2}, if F_{n1} and F_{n2} are sub-features of F_{n0}, then F_{n1} and F_{n2} have the parallel relationship, denoted as $F_{n1} \cup F_{n2}$.

If there is no inclusion or parallel relationship between any two features, the relationship between the two features will not appear in FR, such as $F_{n0} \supset F_{n1}$, $F_{n0} \supset F_{n2}$, $F_{n1} \cup F_{n2}$, $F_{n1} \supset F_{n3}$, the relationship between F_{n3} and F_{n2}, will not appear in FR.

An element of FC represents a constraint condition of excludes or requires between two features. If the feature F_{n1} and F_{n2} exclude each other, F_{n1} and F_{n2} cannot exist at the same time in any software generated by the SPL, denoted as $F_{n1} \leftrightarrow F_{n2}$. If feature F_{n1} depends on feature F_{n2}, when F_{n1} exists in a software generated by the SPL, F_{n2} must exist in the software too, denoted as $F_{n1} \Rightarrow F_{n2}$. If $F_{n1} \Rightarrow F_{n2}$, and $F_{n2} \Rightarrow F_{n1}$, then $F_{n1} \Leftrightarrow F_{n2}$.

In the feature model, the tree diagram contains all the features, types of features and relationship of features. And constraints need to be considered in selecting features to create software, and they ensure the availability and security of the generation of software products. Therefore in the describing of feature models by CMC, all the features and corresponding types should be described first, then, describing the relationship between two features, and last, considering how to describe the constraints between two features. Using the ThrTFM defined in Definition 2, we can abstract the tree diagram of the feature model in Fig. 1, which is convenient to model SPL's feature models by a formal method.

B. MAPPING BETWEEN FEATURE MODEL AND FEvoFM

FEvoFM is a feature model described by CMC, and it can be changed using the change mechanism of CMC. This section introduces the method of mapping between FEvoFM and feature model.

Definition 3 (FEvoFM): FEvoFM is a multivariate set $F = (V, \mu, O_{F_1}, ..., O_{F_n}, (R_{F_1}, \rho_{F_1}), ..., (R_{F_n}, \rho_{F_n}))$. Where V is a collection of all the names of objects which represents the constraint in the feature model, μ is a collection of the membrane structure which represents relationships between two features, O_{F_i} is a objects multiset to represent constraint in membrane F_i, R_{F_i} is a reaction rule set to represent detection process of constraint relation in membrane F_i, ρ_i is a priority set of rules.

Definition 3 is another way of representing a feature model. Definition 2 and Definition 3 are two different ways to describe the feature models. And there is a mapping method to change the description ways. The mapping method is as follows:

1) The element in FN is corresponding to the name of a membrane. Each feature corresponds to a membrane, the presentation of a membrane is same as the representation of a feature.
2) The parallel and inclusion relationships in FR are mapped to the membrane structures of sibling and parent-child. The inclusion relation $F_{n0} \supset F_{n1}$ corresponds to the parent-child structure $[_{F_{n0}}[_{F_{n1}}]_{F_{n1}}]_{F_{n0}}$, and the parallel relation $F_{n1} \cup F_{n2}$ corresponds to the sibling structure $[_{F_{n1}}]_{F_{n1}}[_{F_{n2}}]_{F_{n2}}$. In other words, the collection of mapped membrane structures is μ in FEvoFM.
3) The excludes and requires in FC are corresponding to objects and rules in CMC, where
   - The excludes $F_{n1} \leftrightarrow F_{n2}$ is corresponding to the specific objects and rules in the membrane F_{n1} and membrane F_{n2}. The objects and rules in the membrane F_{n1} are: $Mutex_{F_{n1}-F_{n2}} : O$, $Mutex_{F_{n1}-F_{n2}} : OO_i : O \rightarrow Mutex_{F_{n1}-F_{n2}} : OMutex_{F_{n1}-F_{n2}} : O_{to-F_{n1}}$, $Mutex_{F_{n1}-F_{n2}} : O \rightarrow Check_{F_{n1}-not} : O_{to-F_{n2}}$, $Mutex_{F_{n1}-F_{n2}} : OMutex_{F_{n1}-F_{n2}} : O \rightarrow Mutex_{F_{n1}-F_{n2}} : OCheck_{F_{n1}-in} : O_{to-F_{n2}}$, $Check_{F_{n2}-in} : O \rightarrow Feasible_{F_{n1}-F_{n2}} : O_{to-F_{n0}}$, $Check_{F_{n2}-in} : O \rightarrow unFeasible_{F_{n1}-F_{n2}} : O_{to-F_{n0}}$, The objects and rules in the membrane F_{n2} are: $Mutex_{F_{n1}-F_{n2}} : O$, $Mutex_{F_{n1}-F_{n2}} : OO_i : O \rightarrow Mutex_{F_{n1}-F_{n2}} : OMutex_{F_{n1}-F_{n2}} : O_{to-F_{n2}}$, $Mutex_{F_{n1}-F_{n2}} : O \rightarrow Check_{F_{n2}-not} : O_{to-F_{n1}}$, $Mutex_{F_{n1}-F_{n2}} : OMutex_{F_{n1}-F_{n2}} : O \rightarrow Mutex_{F_{n1}-F_{n2}} : OCheck_{F_{n2}-in} : O_{to-F_{n1}}$, $Check_{F_{n1}-in} : O \rightarrow Feasible_{F_{n1}-F_{n2}} : O_{to-F_{n0}}$, $Check_{F_{n1}-in} : O \rightarrow unFeasible_{F_{n1}-F_{n2}} : O_{to-F_{n0}}$.
   - The requires $F_{n1} \Rightarrow F_{n2}$ are associated with membranes F_{n1} and F_{n2} specific objects and rules in correspondence, objects and rules in the membrane F_{n1} are, $Bing_{F_{n1}-F_{n2}} : O$, $Bing_{F_{n1}-F_{n2}} : OO_i : O \rightarrow Bing_{F_{n1}-F_{n2}} : OBing_{F_{n1}-F_{n2}} : O_{to-F_{n2}}$, $Check_{F_{n2}-in} : O \rightarrow Feasible_{F_{n1}-F_{n2}} : O_{to-F_{n0}}$, $Check_{F_{n2}-not} : O \rightarrow unFeasible_{F_{n1}-F_{n2}} : O_{to-F_{n0}}$, and objects and reaction rules in the membrane F_{n2} are, $Binged_{F_{n1}-F_{n2}} : O$, $Bing_{F_{n1}-F_{n2}} : OBinged_{F_{n1}-F_{n2}} : O \rightarrow Check_{F_{n2}-in} : O_{to-F_{n1}}$, $Bing_{F_{n1}-F_{n2}} : O \rightarrow Check_{F_{n2}-not} : O_{to-F_{n1}}$.

The reaction rules in mapping 3 use the object transfer rules to transfer objects between the two membranes with constraints. These rules represent the process of checking the objects are in or not in the two membranes with constraints. If membranes have the objects, the new objects are generated or transferred to make the next rule react. Finally, if the receiving membrane includes corresponding objects, then it transfers the objects $Feasible_{F_{n1}-F_{n2}} : O$ to the skin membrane F_{n0}, otherwise, it transfers $unFeasible_{F_{n1}-F_{n2}} : O$ to the skin membrane F_{n0}. $Feasible_{F_{n1}-F_{n2}} : O$ and

$unFeasible_{F_{n1}-F_{n2}}$: O represent the detection results. This allows the skin membrane to know whether all of the features are satisfied with constraints. The representation of constraints is changed to the objects and rules in every membrane. So when the constraints change, the membranes must be changed.

The mapping process is reversible. It can describe a given feature model by CMC, and map FEvoFM to ThrTFM which is proposed in Def. 3 by the above three mapping rules.

Theorem 1: If ThrTFM can describe a feature model, FEvoFM can describe the feature model too.

Proof: FN is the feature names in a feature model, from the mapping rules of 1, it is easy to know that for every element in *FN*, there is a membrane to represent it. And *FR* describes the relationship between two features in a feature model. From the mapping rules of 2, we can use the way of $[_{F_{n0}}[_{F_{n1}}]_{F_{n1}}]_{F_{n0}}$ and $[_{F_{n1}}]_{F_{n1}}[_{F_{n2}}]_{F_{n2}}$ to describe the relationship in *FR*. At the same time, for each element in *FC*, we can find objects and rules in the corresponding membranes in FEvoFM. Therefore, when using (FN, FR, FC) to describe a feature model, the feature model can be specified to the form of $(V, \mu, O_{F_1}, ..., O_{F_n}, (R_{F_1}, \rho_{F_1}), ..., (R_{F_n}, \rho_{F_n}))$ too.

Theorem 2: If FEvoFM can describe a feature model, ThrTFM can describe the feature model too.

Proof: In FEvoFM, every membrane represents a feature. From the definition of ThrTFM, every element in *FN* is represented as a feature too. μ in FEvoFM can describe the relationship among features. From the mapping rules of 2, it is a bijection between μ and *FR* in ThrTFM. So when using μ to describe the relationship among features in a feature model, we can use *FR* to describe the relationship too. At the same time, the rules and objects in the membrane describe the constraint of the membrane. And in ThrTFM, *FC* can represent the constraint relation of a feature model. Accordingly, when using $(V, \mu, O_{F_1}, ..., O_{F_n}, (R_{F_1}, \rho_{F_1}), ..., (R_{F_n}, \rho_{F_n}))$ to describe a feature model, the feature model can be described to the form of (FN, FR, FC) too.

From the two theorems, we can know when describing a feature model, FEvoFM and ThrTFM have the same power. In our method, by using the mapping rules given in this section, we can change ThrTFM to FEvoFM firstly, and after changing FEvoFM with the evolutionary requirements, the evolved FEvoFM can be mapped to ThrTFM too.

In FEvoFM, μ is a set of feature pairs with a relationship of parallel or inclusion. In the description of mapping 2, μ will be very long. We can merge feature pairs to reduce the length of the μ. The results of mapping 2 can be combined and merged to get a weight in brackets "[]". The process of combing and merging is:

1) Find the root node in the tree diagram (the membrane F_{n0} representing the feature model's name) and use it as a skin membrane,
2) Remove all of the sibling structures in the results of mapping 2,
3) Find all membranes that have a parent-child relationship with F_{n0}, put these membranes into $[_{F_{n0}}]_{F_{n0}}$ in parallel, and get a new membrane structure μ_x, such as, $[_{F_{n0}}[_{F_{n1}}]_{F_{n1}}]_{F_{n0}}$ and $[_{F_{n0}}[_{F_{n2}}]_{F_{n2}}]_{F_{n0}}$ should be changed to the structure of $[_{F_{n0}}[_{F_{n1}}]_{F_{n1}}[_{F_{n2}}]_{F_{n2}}]_{F_{n0}}$,
4) the membranes of newly added to μ_x are used as new parent membranes by turns, find all of the membranes that have a parent-child relationship with the new parent membrane, and they are juxtaposed into the new parent membrane in turn, until all the membranes are obtained in the newest μ_x.

Finally, we can get μ_x, it is the string μ that we want. μ represents the relationships among all the features in the feature model. Then, a complete model of multi tuple $(V, \mu, O_{F_1}, ..., O_{F_n}, (R_{F_1}, \rho_{F_1}), ..., (R_{F_n}, \rho_{F_n})$ can represent a feature model. V represents all names of objects in every feature, μ describes the structure of all the features, O_i is the objects in each feature, if there is no constraint relation of this feature, O_i is an empty set, (R_i, ρ_i) is a set of reaction rules and the priority of the rule of each feature, if there is no constraint of this feature, (R_i, ρ_i) is empty.

C. EXAMPLE

To explain our method, we give an example to help to understand our method. In the following sections, we will use the example to show how to use our method to change FME with evolutionary requirements.

Example:

1) Feature model named FME, FME has two mandatory features A and B, A has two optional features C and D, B has two alternative features E and F and two or features G and H;
2) Evolutionary requirements, they are that add an or feature I to B, delete the feature G, change C to an optional feature J.

Fig. 3 shows the feature model of the example.

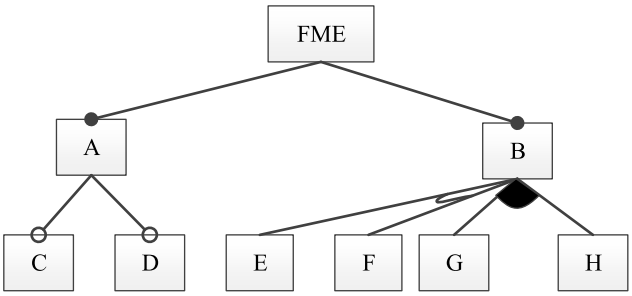

FIGURE 3. Feature model of FME.

Using the Def. 2, the feature model of FME can be described to ThrTFM such as (FN, FR, FC), where $FN = \{FME, A^M, B^M, C^O, D^O, E^A, F^A, G^R, H^R\}$, $FR = \{A^M \subset FME, B^M \subset FME, A^M \cup B^M, C^O \subset A^M, D^O \subset A^M, C^O \cup D^O, E^A \subset B^M, F^A \subset B^M, E^A \cup F^A, G^R \subset B^M, H^R \subset B^M, G^R \cup H^R, E^A \cup G^R, E^A \cup H^R, F^A \cup G^R, F^A \cup H^R\}$, $FC = \emptyset$. And then, using the mapping

rules mentioned in part B of section IV, the ThrTFM can be changed to FEvoFM. As $FC = \emptyset$, it means V, O, R and ρ in FEvoFM of FME is $\emptyset$. Elements in FR can be mapped to $[_{FME}[_{A^M}]_{A^M}]_{FME}$, $[_{FME}[_{B^M}]_{B^M}]_{FME}$, $[_{A^M}]_{A^M}[_{B^M}]_{B^M}$, $[_{A^M}[_{C^O}]_{C^O}]_{A^M}$, $[_{A^M}[_{D^O}]_{D^O}]_{A^M}$, $[_{A^M}[_{C^O}]_{C^O}[_{D^O}]_{D^O}$, $[_{B^M}[_{E^A}]_{E^A}]_{B^M}$, $[_{B^M}[_{F^A}]_{F^A}]_{B^M}$, $[_{B^M}[_{G^R}]_{G^R}]_{B^M}$, $[_{B^M}[_{H^R}]_{H^R}]_{B^M}$, $[_{E^A}]_{E^A}[_{F^A}]_{F^A}$, $[_{E^A}]_{E^A}[_{G^R}]_{G^R}$, $[_{E^A}]_{E^A}[_{H^R}]_{H^R}$, $[_{F^A}]_{F^A}[_{G^R}]_{G^R}$, $[_{F^A}]_{F^A}[_{H^R}]_{H^R}$, $[_{G^R}]_{G^R}[_{H^R}]_{H^R}$. Then the elements can be comb and merge to $[_{FME}[_{A^M}[_{C^O}]_{C^O}[_{D^O}]_{D^O}]_{A^M}[_{B^M}[_{E^A}]_{E^A}[_{F^A}]_{F^A}[_{G^R}]_{G^R}[_{H^R}]_{H^R}]_{B^M}]_{FME}$. Finally, the feature model in Example can be changed to $F = (V, \mu, O_{FME}, ..., O_{H^R}, (R_{FME}, \rho_{FME}), ..., (R_{H^R}, \rho_{H^R}))$, where $V = \emptyset$, $\mu = [_{FME}[_{A^M}[_{C^O}]_{C^O}[_{D^O}]_{D^O}]_{A^M}[_{B^M}[_{E^A}]_{E^A}[_{F^A}]_{F^A}[_{G^R}]_{G^R}[_{H^R}]_{H^R}]_{B^M}]_{FME}$, $O_{FME} = O_{A^M} = \cdots = O_{H^R} = \emptyset$, $R_{FME} = R_{A^M} = \cdots = R_{H^R} = \emptyset$, $\rho_{FME} = \rho_{A^M} = \cdots = \rho_{H^R} = \emptyset$.

In this section, the method of abstracting the feature model to ThrTFM is given, and how to map ThrTFM to FEvoFM is given. The process of formal modeling of SPL feature model using CMC and reduction of the formal model to SPL feature model are completed.

V. AutoEvoNeeds

There are many ways to define the evolution of requirements of SPL, which can be domain oriented or feature oriented. This paper focuses on the evolution of SPL's feature models, so the expected evolutionary requirements are needed to be defined in feature oriented.

A. ABSTRACTION OF SPL'S EVOLUTIONARY REQUIREMENTS

We can use the 9 mapping rules named Requirements of Domain Oriented to Architecture of SPL to map oriented domain evolution requirements to features or combination of features [23]. Then using the method of making SPL feature model proposed by Neves et al. to model the evolutionary requirements [26]. Domain experts can rely on the modeling tools mentioned in above articles to complete the extraction and construction of features and combinations of features from requirements. The AutoEvoRe is the abstraction of evolutionary requirements which are constructed by domain experts.

Definition 4 (Evolutionary Feature): Evolutionary features are features to be evolved. The evolutionary features can be a feature or an evolutionary combination of features with the same parent feature.

In SPL, the evolutionary combination of features is a unique way of evolution. It comes from the evolution of adding software with related functionality to the SPL. Or the changes of using software with the better performance to replace parts functionality of the SPL.

In this section, how to use CMC to describe evolutionary requirements of SPL to AutoEvoRe is given. An evolutionary requirement cannot be recognized by FEvoFM. We should formalize the evolutionary requirement by CMC.

In FEvoFM, the evolutionary requirements are split into three parts: the features to be evolved and their parent features, the ways in which they evolve, and what features they evolve into. Thus, the evolutionary requirements of an SPL feature model can be represented by a set.

When evolving a feature model of SPL, it may change many features in the feature model. It is easy to know that an evolutionary requirement is made up of several evolutionary needs. When using the method we proposed in this paper, it handles an evolutionary need once. A complete evolution requires several modifications of the feature model by the proposed approach.

Definition 5 (Evolutionary Need): An evolutionary need is a three-tuple (F-target, Changes, F-goal). Where F-target is called the target features, which is an abstract expression of evolutionary features with the same parent feature, Changes is one of the evolution model named add, delete and replace, F-goal is known as goal features, it is the abstract representation of the evolution result of F-target.

An evolutionary need is a part of evolutionary requirement. An evolutionary requirement is a set of evolutionary needs. An abstract representation of F-goal and F-target mentioned in definition 5 is the three-tuple defined by definition 2, that F-target is ThrTFM of the target features, F-goal is ThrTFM of the goal features.

In the SPL's feature model evolution, we should not only pay attention to the features that need to be evolved but also pay attention to their parent features. It is because that when a feature is evolved, the relationship between the feature and its parent may be changed. For example, when adding sub-features into a feature, a parent-child relationship is added, the parent feature is influenced by the evolution, the parent feature needs to be considered in the evolution of adding features. When deleting features, the parent feature is influenced by its nodes being deleted. In the three kinds of changes, replacing features is minimal effect on the parent feature, but to describe the location of the change of features more accurately, we need to use a constant parent feature as a reference. Thus, in definition 5, it is emphasized that the features to be evolved and their parent feature are abstracted together in F-target and F-goal.

The three-tuple of evolutionary need describes the evolution of the SPL's feature model, but the three-tuple is not identified by FEvoFM. It is necessary to formalize the evolutionary need by CMC. The formalized evolutionary need is called an AutoEvoNeed.

Definition 6 (AutoEvoNeed): An AutoEvoNeed is an abstraction of evolutionary need using CMC. An AutoEvoNeed is represented as an object $Re_{F_{target}-Changes-F_{goal}} : O$, where Re is the identifier of the AutoEvoNeed, ^{O} is the type of the object, F_{target} is the formal representation of target features F-target by CMC, $-$ is a connector to join all the features and operations together, $Changes \in \{add, delete, replace\}$, F_{goal} is the formal representation of goal features F-goal by CMC.

The subscript of Re is evolutionary need information. Using the corresponding relation, the evolutionary needs of SPL's feature model can be changed into AutoEvoNeeds

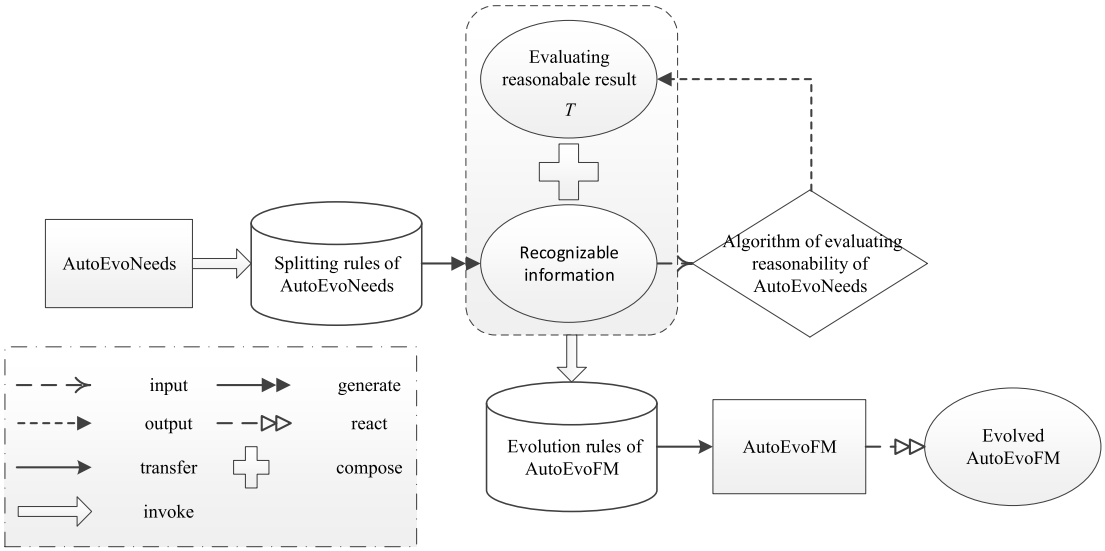

FIGURE 4. The Framework of AutoEvoChange.

which can be automatically distinguished by a FEvoFM. F_{farher} represents the parent feature in F-target. An AutoEvoNeed is an object that can change the system using the characteristic of CMC. The characteristic is object can trigger rules react to make the system change.

AutoEvoNeeds can evolve FEvoFM. But before using AutoEvoNeeds to evolve FEvoFM, we need to evaluate whether AutoEvoNeeds can trigger the evolution of FEvoFM and ensure the rationality of evolution. At the same time, if we want to evolve FEvoFM completely, every AutoEvoNeed in AutoEvoRe should trigger the evolution of FEvoFM once.

B. EXAMPLE

Consider the evolutionary requirements of the example in part C of section IV. The father features in the three needs are B, B, and A. Using the way in μ to describe feature structures in evolutionary requirements, the requirements can be represented as

$\{((\{B^M\}, \emptyset, \emptyset), add, (\{B^M, I^O\}, \emptyset, \emptyset)), ((\{B^M, G^R\}, \emptyset, \emptyset), delete, (\{B^M\}, \emptyset, \emptyset)), ((\{A^M, C^O\}, \emptyset, \emptyset), replace, (\{A^M, J^O\}, \emptyset, \emptyset))\}$.

Then use the three-tuple as the subscript of Re, the evolutionary requirements can be represented as three AutoEvoNeeds, $Re_{[_{B^M}\emptyset,\emptyset,(\emptyset,\emptyset)]_{B^M}-add-[_{B^M}\emptyset,[_{I^O}]_{I^O},(\emptyset,\emptyset)]_{B^M}}$, $Re_{[_{B^M}\emptyset,[_{G^R}]_{G^R},(\emptyset,\emptyset)]_{B^M}-delete-[_{B^M}\emptyset,\emptyset,(\emptyset,\emptyset)]_{B^M}}$, $Re_{[_{A^M}\emptyset,[_{C^O}]_{C^O},(\emptyset,\emptyset)]_{A^M}-replace-[_{A^M}\emptyset,[_{J^O}]_{J^O},(\emptyset,\emptyset)]_{A^M}}$.

VI. AutoEvoChange

The process of changing model is called AutoEvoChange. And FEvoFM and AutoEvoNeeds are two input of AutoEvoChange. The purpose of AutoEvoChange is evolving FEvoFM. AutoEvoChange is called changing model in Fig. 2. There are three parts in AutoEvoChange in this paper. Firstly, we need to split AutoEvoNeeds. Because all the information of evolution is contained in AutoEvoNeeds, we need to know where is changed, how to change it and what kind it will be changed. The splitting rules will complete the splitting of AutoEvoNeeds. Secondly, we need to determine whether an AutoEvoNeed is reasonable. The reasonability of AutoEvoNeeds means it can evolve FEvoFM correctly. Finally, we will use the results of the first two steps and the present evolution rules list to evolve FEvoFM. Fig. 4 shows the details of AutoEvoChange.

Fig. 4 is the framework of AutoEvoChange. It shows the details of ''Changing model'' in Fig. 2. When an AutoEvoNeed appears, it will be split by splitting rules, and the result of splitting is recognizable information. It is shown in part A of section VI. And then the recognizable information will be used as input of the algorithm shown in part B of section VI to evaluating reasonability of AutoEvoNeeds. Finally, the evaluating result T and the recognizable information will trigger the evolution rules, the corresponding evolution rules will be transferred into FEvoFM, then FEvoFM can evolve, and we can get its evolution result. The evolution rules and the evolution process of FEvoFM are shown in part C of section VI.

A. SPLITTING AutoEvoNeeds

An AutoEvoNeed is an input of changing AutoEvoChange. It contains all the information of evolution, and it needs to be judged whether it is the prerequisite for the evolution of SPL. To better extract information and prepare for the coming judgment, the AutoEvoNeeds needs to be split. We need to know the evolution model and the evolutionary features to evaluate reasonability of AutoEvoNeeds. And in the evolution rules list of feature models, all the evolution rules meet the three kinds of evolution model will be given. But the abstract objects are used in evolution rules, and we need to instantiate them firstly. The process of splitting AutoEvoNeeds contains three parts in Fig. 4. They are AutoEvoNeeds, splitting

TABLE 1. Splitting rules of AutoEvoNeeds.

Name	Rule	Meaning
R_1	$Re_{F_{target}-Changes-F_{goal}} : O \rightarrow F_{target} Changes F_{goal} : O$	Extract evolution information from the AutoEvoNeed
R_2	$F_{target} Changes F_{goal} : O \rightarrow F_{target} : O F_{goal} : O$ $Changes : O_{to-Evolution} F_{farher} : Or : R_{to-Evolution}$	Split the information and transmit some information to related membranes to make these membrane change.
R_3	$F_{farher} : O \rightarrow (F_f = F_{farher}) : O_{to-Evolution}$	Replace F_f with F_{farher} and transmit it to evolution membrane.
R_4	$F_{target} : O \rightarrow (F_t = F_{target}) : O_{to-Evolution}$	Replace F_t with F_{target} and transmit it to evolution membrane.
R_5	$F_{goal} : O \rightarrow (F_g = F_{goal}) : O_{to-Evolution}$	Replace F_g with F_{goal} and transmit it to evolution membrane.

rules of AutoEvoNeeds and recognized information. AutoEvoNeeds are the input and recognized information is the output in the splitting process. And by using the splitting rules, the splitting process can be done. The splitting process of an AutoEvoNeed is to materialize abstract objects. We can split out target features, goal features and changes from an AutoEvoNeed $Re_{F_{target}-Changes-F_{goal}} : O$, and F_i, F_j, F_f used in evolution rules will become features in a real system with instantiated by target features, goal features and actives.

In CMC, every rule should be put into a membrane, so we put the splitting rules in a membrane named Splitting which are parallel to FEvoFM. The rules in Splitting are given in Table 1. These rules split AutoEvoNeeds.

Table 1 describes the rules of splitting AutoEvoNeeds. Firstly, it needs to split AutoEvoNeeds by rules R_1 and R_2. Then, by rules R_3, R_4 and R_5, the variables of F_t, F_g, and F_f can be instantiated. Finally, the result is passed to the membrane Evolution which is parallel to FEvoFM and has evolution rules of changing FEvoFM.

When an AutoEvoNeed comes, the model can be changed automatically. The rule R_1 can make the AutoEvoNeed be recognized without understanding the definition of CMC and knowing the information of AutoEvoNeed. Rules R_2 to R_5 are transmitting information to related membranes to notice the system where needs to change.

An AutoEvoNeed $Re_{F_{target}-Changes-F_{goal}} : O$ is placed into the membrane Splitting, and membrane Splitting contains all the reaction rules in Table 1. $Re_{F_{target}-Changes-F_{goal}} : O$ can trigger reaction rules in Splitting, then the initial object $Re_{F_{target}-Changes-F_{goal}} : O$ will be split. Finally, the object $Re_{F_{target}-Changes-F_{goal}} : O$ in membrane Splitting will be split into five different objects, which are passed to membrane Evolution.

B. EVALUATE AutoEvoNeeds

For every AutoEvoNeed in AutoEvoRe, we should evaluate reasonability of the SPL's AutoEvoNeeds. The reasonable of SPL's AutoEvoNeeds is the prerequisite for the evolution of SPL. The evaluating condition comes from evolving ways and evolutionary features. We will give the condition of evaluating reasonability of AutoEvoNeeds in this section.

In this article, features F_1 are sub-features of a feature model Fn, we can use $F_1 \subseteq Fn$ to donate the relationship between of F_1 and Fn. $F_1 \cap Fn$ represents the same features between F_1 and Fn, if the results of $F_1 \cap Fn$ are F_2, we can say that $F_1 \cap Fn = F_2$.

Adding features means to add new features to the original feature model. In adding features, features in the goal features F_{goal} cannot be same as the features in the feature model Fn except the parent feature F_{farher}, it means $F_{goal} \cap Fn = F_{farher}$. And the target feature F_{target} must be the sub-features of the feature model Fn, it means $F_{target} \subseteq Fn$.

Deleting features means to delete evolutionary features from the original feature model. The target feature to delete must be the sub-features of the feature model, it means $F_{target} \subseteq Fn$. The goal feature has the same feature F_{farher}, and the feature belongs to feature model, it means $F_{goal} \cap Fn = F_{farher}$.

Replacing features means to replace an evolutionary feature in feature model by a new evolutionary feature. When replacing features, the target feature must be the sub-features of the feature model, it means $F_{target} \subseteq Fn$, and the goal features have a same feature F_{farher} as the feature model, it means $F_{goal} \cap Fn = F_{farher}$.

Therefore, based on the above analysis of three changes of evolution, we can obtain the evaluating condition of AutoEvoNeeds:

1) If $Changes \in \{add\}$, then $(F_{target} \subseteq Fn) \wedge (F_{target} \cap Fn = F_{target}) \wedge (F_{goal} \cap Fn = F_{farher})$. It means if the evolution is adding features to the model, target feature F_{target} must be a sub-feature of the model Fn, F_{target} has a only sub-feature F_{farher} which is same to Fn, and F_{farher} is the only intersection between Fn and F_{goal}.
2) If $Changes \in \{delete, change\}$, then $(F_{target} \subseteq Fn) \wedge (F_{goal} \cap Fn = F_{farher})$. It means if the evolution is deleting or changing features from the model, target feature F_{target} must be a sub-feature of the model Fn, and F_{goal} has a only sub-feature F_{farher} which is same to.

To determine whether AutoEvoNeeds meet the evolving demands of the systems, it gives an algorithm to evaluate the reasonability of AutoEvoNeeds. If AutoEvoNeeds meet the

TABLE 2. Algorithm for evaluating reasonability of AutoEvoNeeds.

Input: structure of an SPL's feature model μ_{Fn}, evolution model Changes, structure of target features $\mu_{F_{target}}$, structure of goal features $\mu_{F_{goal}}$.
Output: results T or F.
Algorithm:
1. μ_{Fn}, Changes, $\mu_{F_{target}}$ and $\mu_{F_{goal}}$ are 4 different string arrays as the inputs. And the "[" and "] " are elements in a strings array, each subscript is an element of the string array too.
2. If $\mu_{F_{target}}$ is the substring array of μ_{Fn}, let $R_1 = T$ then proceed to step 3, and if $\mu_{F_{target}}$ is not the substring array of μ_{Fn}, output F and terminate algorithm.
3. $\mu_{F_{farher}}$ consists the first, second, and last two strings of $\mu_{F_{target}}$.
4. Use the word segmentation method to remove the"[" and "] " in $\mu_{F_{goal}}$, $\mu_{F_{farher}}$ and μ_{Fn}, and remove the repeated elements in the result to get three new string arrays $N_{F_{goal}}$, $N_{F_{farher}}$ and $N_{F_{goal}}$.
5. Compute the intersection N of $N_{Ftarget}$and N_{Fn}, and N is the same element between $N_{Ftarget}$and N_{Fn}.
6. Compute the intersection M of $N_{Fgoal}$and N_{Fn}, and M is the same element between $N_{Fgoal}$and N_{Fn}.
7. Evaluate whether $N_{Ftarget}$ and M is equal, if the result is equal, let R_2=T, then let R_2=F.
8. Evaluate whether $N_{Ffather}$ and N is equal, if the result is equal, let R_3=T, then let R_3=F.
9. Use "Changes" to evaluate R_1, R_2, $R_3$and N:
(1) When Changes[]={add} and R_1=T, if R_2=T and R_3=T, output T, and if R_2=F or R_3=F output F.
(2) When Changes[]={delete} and R_1=T, if R_3=T output T, and if R_3=F output F.
(3) When Changes[]={replace and R_1=T, if R_3=T output T, and if R_3=F output F.
(4) The rest of the output is F.

demand of system's evolution, the output of the algorithm is passed into membrane Evolution. In Evolution, the output can make the membrane Evolution send messages to FEvoFM. These messages can make FEvoFM evolve. The input of the algorithm is the structure of AutoEvoNeed, and the structure is deleted all the ",", "(", ")" and "Ø" in F_{target}, Fn and F_{goal}.

If the output of the algorithm is T, it means AutoEvoNeed satisfies the demand of the system. If the output of the algorithm is F, AutoEvoNeeds cannot trigger the evolution reaction, and it needs domain experts to reconstruct the evolutionary needs.

With evaluating reasonability of AutoEvoNeeds by the algorithm, the result T of the algorithm will be put into membrane Evolution. T is one of the initial objects to trigger a reaction of FEvoFM automatically.

C. EVOLVE FEvoFM

So far, all the preparations have been completed. In this section, we will give the method of evolving FEvoFM. Evolution rules in membrane Evolution are given. Using these rules and the results from part A and B of section VI, the FEvoFM can evolve automatically. In this section, we will provide the reaction rules and explain how to use these rules to evolve FEvoFM. To describe the evolution process of FEvoFM, a membrane named Evolution is built. Membrane Evolution has all the reaction rules required by the evolution of FEvoFM. And membrane Evolution is parallel to FEvoFM. Objects in membrane Evolution trigger rules to react. These reactions can transfer corresponding rules and objects to FEvoFM. Finally, the evolved FEvoFM can be got.

The evolution rules of FEvoFM mainly include replicating rules, transferring rules and deleting rules. In the rules, F_i, F_g, and F_f are used to represent the target, goal and parent features respectively, which makes the evolution rules more universal.

The evolution rules are designed with the definition of CMC. The adding features mean to create new membranes in the system. The replacing features mean to dissolve the previous membranes and create new membranes in the system. And the deleting features mean to dissolve the previous membranes in the system. Using the design principle and design process, we can get all the evolution rules of FEvoFM. These rules are shown in Table 3.

In Table 3, the first column lists the names of the reaction rules. Reaction rules R_6, R_9 and R_{10} describe the process of adding features. R_6 is describing the process of extracting information of the new feature from the object $O_{AddF_g} : O$, the information contains where and what to add a new feature. R_9 sends the adding object to the father feature and copies the related rules. R_{10} sends the coping rules to the father feature. Reaction rules R_7, R_{11} and R_{12} explain the process of deleting features. R_7 is describing the process of extracting information from the object $O_{DeleteF_i} : O$, the information contains

TABLE 3. The evolution rules of FEvoFM.

Name	Rule
R_6	$O_{AddF_g} : O \to [_{F_f} O_{F_g}, \mu_{F_g}, (R_{F_g}, \rho_{F_g})]_{F_f}\ Finish : R$
R_7	$O_{DeleteF_t} : O \to \sigma_{F_t/F_f}\ Finish : R$
R_8	$O_{F_t ChangeF_g} : O \to [_{F_f} O_{F_g}, \mu_{F_g}, (R_{F_g}, \rho_{F_g})]_{F_f}\ \sigma_{F_t/F_f}\ Finish : R$
R_9	$F_t : OAdd : OF_g : OF_f : OT : Or : R \to O_{AddF_g} : O_{\text{to}\_F_f}\ Translate : RF_f : R$ $Add : RF_g : R(R_6, highest)_{copy}(Finish : R \to (R_6)_{delete}, higher)_{copy},$
R_{10}	$Translate : RF_f : RAdd : RF_g : R \to (R_6, highest)_{in\_F_f}(Finish : R \to (R_6)_{delete}, higher)_{in\_F_f}$
R_{11}	$F_t : ODelete : OF_g : OF_f : OT : Or : R \to O_{DeleteF_t} : O_{to\_F_t} Translate : RF_f : R$ $Delete : RF_g : R(R_7, highest)_{copy}(Finish : R \to (R_7)_{delete}, higher)_{copy},$
R_{12}	$Translate : RF_f : RDelete : RF_g : R \to (R_7, highest)_{in\_F_f}(Finish : R \to (R_7)_{delete}, higher)_{in\_F_f}$
R_{13}	$F_t : OChange : OF_g : OF_f : OT : Or : R \to O_{F_t ChangeF_g} : O_{\text{to}\_F_f} Translate : RF_g : R$ $F_f : RChange : RF_g : R(R_8, highest)_{copy}(Finish : R \to (R_8)_{delete}, higher)_{copy},$
R_{14}	$Translate : RF_t : RF_f : RChange : RF_g : R \to (R_8, highest)_{in\_F_f}(Finish : R \to (R_8)_{delete}, higher)_{in\_F_f}$

where and what to delete the feature. R_{11} sends the deleting object to the father feature and copies the related rules. R_{12} sends the coping rules to the father feature. Reaction rules R_8, R_{13} and R_{14} describe the process of changing features. R_8 is describing the process of extracting information of the changing feature from the object $O_{F_t ChangeF_g} : O$, the information contains which and what is replaced. R_{13} sends the changing object to the father feature and copies the related rules. R_{14} sends the coping rules to the father feature.

$T : O$ in the reaction rules R_9, R_{11} and R_{13} comes from the result of the algorithm in part B of section VI. When the output of the algorithm is "T", the object $T : O$ is added into membrane Evolution. And $T : O$ is an object that can be recognized by rules.

Objects in membrane Evolution can trigger one of R_9, R_{11} and R_{13} reacting, when $T : O$ and the five objects from part A of section VI. are passed into membrane Evolution. Results of the reaction can create new objects, transfer rules and objects to the membrane F_{farher} in FEvoFM. The transferred rules and objects can make FEvoFM change. And the new objects in membrane Evolution trigger one of R_{10}, R_{12} and R_{14} reacting. The reacting will transfer deleting rules to F_{farher}. The deleting rules can make FEvoFM have the rules describing the constraint among features. The final FEvoFM is the evolved FEvoFM.

It is worth mentioning that, when evolving features F_{target}, if there are constraint relationships between F_{target} and features F/F_{target}, it is necessary to generate another $Re_{F_{target}-Changes-F_{goal}} : O$ to make F/F_{target} change.

D. EXAMPLE

Consider the AutoEvoNeed $Re_{[_{B^M}\emptyset,\emptyset,(\emptyset,\emptyset)]_{B^M}-add-[_{B^M}\emptyset,[_{I^O}]_{I^O},(\emptyset,\emptyset)]_{B^M}}$ in Example, it is changed to a new object $[_{B^M}\emptyset, \emptyset, (\emptyset, \emptyset)]_{B^M} add [_{B^M}\emptyset, [_{I^O}]_{I^O}, (\emptyset, \emptyset)]_{B^M} : O$ by applying R_1. According to R_2, the object would be then changed to five objects $[_{B^M}\emptyset, \emptyset, (\emptyset, \emptyset)]_{B^M} : O$, $add : O$, $[_{B^M}]_{B^M} : O$, $[_{B^M}\emptyset, [_{I^O}]_{I^O}, (\emptyset, \emptyset)]_{B^M} : O$ and $r : R$, where $add : O$ and $r : R$ should be passed to the membrane Evolution. Finally by applying rules R_3, R_4 and R_5, the other three objects are changed to $(F_i = [_{B^M}\emptyset, \emptyset, (\emptyset, \emptyset)]_{B^M}) : O$, $(F_j = [_{B^M}\emptyset, [_{I^O}]_{I^O}, (\emptyset, \emptyset)]_{B^M}) : O$ and $(F_f = [_{B^M}]_{B^M}) : O$, and the new three objects are passed to membrane Evolution respectively.

The structure μ of F_{target}, F_{farher} and F_{goal} in the AutoEvoNeed are the input of algorithm in part B of section VI, where $Change = add$ is another input to judge whether the AutoEvoNeed satisfies the system demand. Such as the input of the first AutoEvoNeeds is $\mu_{F_{target}} = [_{B^M}]_{B^M}$, $\mu_{F_{goal}} = [_{B^M}[_{I^O}]_{I^O}]_{B^M}$, $\mu_{F_{farther}} = [_{B^M}]_{B^M}$ and $Change = add$. Finally, we get the results of the algorithm, and the result is stored in Evolution.

The changing mechanism of CMC and the rules in Table 3 are used to change FEvoFM of FME. In Evolution, R_9 is used firstly. Six objects in previous jobs are changed to five new objects and two copied rules, and $O_{Add[_{B^M}\emptyset,[_{I^O}]_{I^O},(\emptyset,\emptyset)]_{B^M}} : O$ is passed into feature B. Then, the copied rules are moved to feature B by applying R_{10}. Finally, using these objects and rules from Evolution, a new feature I can be added into B. FEvoFM of FME is changed to $F = (V, \mu, O_{FME}, ..., O_{H^R}, (R_{FME}, \rho_{FME}), ..., (R_{H^R}, \rho_{H^R}))$, where $V = \emptyset$, $\mu = [_{FME}[_{A^M}[_{C^O}]_{C^O}[_{D^O}]_{D^O}]_{A^M}[_{B^M}[_{E^A}]_{E^A}[_{F^A}]_{F^A}[_{G^R}]_{G^R}[_{H^R}]_{H^R}[_{I^O}]_{I^O}]_{B^M}]_{FME}$, $O_{FME} = O_{A^M} = \cdots = O_{H^R} = \emptyset$, $R_{FME} = R_{A^M} = \cdots = R_{H^R} = \emptyset$ and $\rho_{FME} = \rho_{A^M} = \cdots = \rho_{H^R} = \emptyset$.

Use another two AutoEvoNeeds in turn to evolve FEvoFM in the same way. Finally, the evolved FEvoFM is $F = (V, \mu, O_{FME}, ..., O_{I^O}, (R_{FME}, \rho_{FME}), ..., (R_{I^O}, \rho_{I^O}))$, where $V = \emptyset$, $\mu = [_{FME}[_{A^M}[_{J^O}]_{J^O}[_{D^O}]_{D^O}]_{A^M}[_{B^M}[_{E^A}]_{E^A}[_{F^A}]_{F^A}$

```
Input:
FN:FME_M, A_M, B_M, C_O, D_O, E_A, F_A, G_R, H_R
FR:(FME_M, A_M), (FME_M, B_M), (A_M+B_M), (A_M, C_O), (A_M, D_O), (C_O+D_O),
(B_M, E_A), (B_M, F_A), (B_M, G_R), (B_M, H_R), (E_A+F_A), (E_A+G_R), (E_A+H_R),
(F_A+G_R), (F_A+H_R), (G_R+H_R)
FC:""
Output1:
V:""
O:""
R:""
P:""
U:[FME_M[A_M[C_O]C_O[D_O]D_O]A_M[B_M[E_A]E_A[F_A]F_A[G_R]G_R
[H_R]H_R]B_M]FME_M
```

FIGURE 5. Change ThrTFM to FEvoFM.

$[_{H^R}]_{H^R}[_{I^O}]_{I^O}]_{B^M}]_{FME}$, $R_{FME} = R_{A^M} = \cdots = R_{I^O} = \emptyset$, $\rho_{FME} = \rho_{A^M} = \cdots = \rho_{I^O} = \emptyset$. And then, using the mapping rules in section V, evolved FEvoFM can be mapped to evolved feature model.

VII. THE TOOL OF AutoEvoFM

The tool of AutoEvoFM is designed and implemented in Java. In our program, there are three parts of computing the system. The first part is the method to change ThrTFM to FEvoFM. In the second part, we implement splitting of AutoEvoNeeds, evaluating of AutoEvoNeeds, and changing of FEvoFM with reaction rules. The output of this part is the evolved FEvoFM. The third part realizes the changing from evolved FEvoFM to new ThrTFM. And the new tuple is the result of the evolution feature model. The output of every part is exported to the TXT file. We use the tool to change the example. The design principle is as follows:

Firstly, ThrTFM (FN, FR, FC) of a feature model can be changed to FEvoFM Fn by the mapping rules between ThrTFM and FEvoFM. Fn is a necessary model for the whole work. The input and output of this part are shown in Fig. 5.

Secondly, a formalized evolutionary requirement is divided into several evolutionary needs named AutoEvoNeeds, and evolve FEvoFM with AutoEvoNeeds one by one. Every AutoEvoNeed must be operated by splitting and evaluating. An AutoEvoNeed is an object that can be recognized by the system and meet the evolutionary demand. In this part, it will change an evolutionary need of demand-oriented or feature-oriented to a three-tuple (F-target, Changes, F-goal) by the given tools and principles for a given need, and then (F-target, Changes, F-goal) can be transformed into an AutoEvoNeed. $Re_{F_{target}-Changes-F_{goal}} : O$ will be split into three parts in the membrane Splitting and evaluated by the algorithm of evaluating reasonability of AutoEvoNeeds, if the relationship between object $Re_{F_{target}-Changes-F_{goal}} : O$ and structure of Fn satisfies the conditions of evolution, it will proceed to the next step. Reaction rules in membrane Evolution are called by $Changes : O$. These invoked rules and objects are passed into Fn and make Fn change. Repeat the process until every AutoEvoNeed in AutoEvoRe acts on FEvoFM. The final result Fn is the evolved FEvoFM. The input and output of this part are shown in Fig. 6. Finally, Fn must be mapped to new ThrTFM (FN, FR, FC) by the mapping rules between FEvoFM and ThrTFM. (FN, FR, FC) is the evolved feature model based on the evolutionary requirement.

But the structure of FEvoFM is merged feature pairs, and we should split it first. The splitting principle of the membrane structure μ is designed:

1) Take the skin membrane as the first parent feature, and find all the sub-membranes of skin membrane contained in μ. All the sub-membranes have an inclusion relationship with the skin membrane and the sub-membranes have a parallel relation with each other,
2) Each sub-membrane from the previous step is used as new parent membrane, repeating step 1 until all the membranes do not have a sub-membrane.

The process of split the structure of μ, and map FEvoFM to ThrTFM are implemented in this part. The input and output of this part are shown in Fig. 7.

The final ThrTFM is the result we want. We can use it to guide the next evolution of SPL.

VIII. EXPERIMENTS

In this section, we use the existing feature models and evolutionary requirements to evaluate the effectiveness of the proposed method AutoEvoSPL. The required feature models in our experiments are from the acquisition of SPL tool (SPLOT) and other open literature.

A. EXPERIMENT SETTINGS

The purpose of this experiment is to evaluate the validity of proposed method. The experimental process is as follows: first, we collected evolutionary requirements and two versions of feature models of SPL. And then, we use the requirements and low version feature model as the input of our tool to change the low version feature model. Finally, the output is compared with high version feature model.

```
U:[FME_M[A_M[C_O]C_O[D_O]D_O]A_M[B_M[E_A]E_A[F_A]F_A[G_R]G_R[H_R]H_R]B_M]FME_M
Re:[B_M]B_M-add-[I_O]I_O
Output:
No Constrained Features Changed
U:[FME_M[A_M[C_O]C_O[D_O]D_O]A_M[B_M[E_A]E_A[F_A]F_A[G_R]G_R[H_R]H_R[I_O]I_O]B_M]FME_M
Input:
V:""
O:""
R:""
P:""
U:[FME_M[A_M[C_O]C_O[D_O]D_O]A_M[B_M[E_A]E_A[F_A]F_A[G_R]G_R[H_R]H_R[I_O]I_O]B_M]FME_M
Re:[B_M[G_R]G_R]B_M-delete-[B_M]B_M
Output:
No Constrained Features Changed
U:[FME_M[A_M[C_O]C_O[D_O]D_O]A_M[B_M[E_A]E_A[F_A]F_A[H_R]H_R[I_O]I_O]B_M]FME_M
Input:
V:""
O:""
R:""
P:""
U:[FME_M[A_M[C_O]C_O[D_O]D_O]A_M[B_M[E_A]E_A[F_A]F_A[H_R]H_R[I_O]I_O]B_M]FME_M
Re:[A_M[C_O]C_O]A_M-replace-[A_M[J_O]J_O]A_M
Output:
No Constrained Features Changed
U:[FME_M[A_M[J_O]J_O[D_O]D_O]A_M[B_M[E_A]E_A[F_A]F_A[H_R]H_R[I_O]I_O]B_M]FME_M
```

FIGURE 6. Change FEvoFM by AutoEvoRe.

```
Input:
V:""
O:""
R:""
P:""
U:[FME_M[A_M[J_O]J_O[D_O]D_O]A_M[B_M[E_A]E_A[F_A]F_A[H_R]H_R[I_O]I_O]B_M]FME_M
Output:
FN:FME_M, A_M, B_M, J_O, D_O, E_A, F_A, I_O, H_R
FR:(FME_M, A_M), (FME_M, B_M), (A_M+B_M), (A_M, J_O), (A_M, D_O), (J_O+D_O), (B_M, E_A), (B_M, F_A),
(B_M, I_O), (B_M, H_R), (E_A+F_A), (E_A+I_O), (E_A+H_R), (F_A+I_O), (F_A+H_R), (I_O+H_R)
FC:""
```

FIGURE 7. Change FEvoFM to ThrTFM.

We collected two data sets as experimental subjects from SPLOT http://www.splot-research.org/ and Mobile Media http://www.ic.unicamp.br/~tizzei/mobilemedia/. SPLOT is a website for SPL's developers to upload their feature models.

Among more than 200 feature models we chose 4 feature modes as our experimental data. They are Smart Home, Android SPL, Help System and SPL SimulES. These feature models have two consecutive versions and are designed by the same editor. In each model, there are more than 50 features. Mobile Media is an SPL for generating mobile media players. It is developed basing on Mobile Photo. Currently, there are 7 versions of Mobile Media and its website provides evolution requirements, feature models, etc.

In the experiment, comparing the differences between the two versions of a same feature model, we give the evolutionary requirements of SPLs from SPLOT. And the evolutionary requirements of Mobile Media are obtained from its website. Lack of space forbids the detailed description of each SPLs' feature model. The evolution requirements are shown in Table 4. In Table 4, the name of SPL is the name of low version feature model, and the evolutionary requirements will be used to evolve the low version feature model.

Next, we will change these requirements to AutoEvoNeed by the methods mentioned in section V. Domain experts can change these evolutionary requirements of Smart Home, Android SPL, Help System, SPL SimulES and Mobile Media to AutoEvoNeeds. The AutoEvoNeeds are shown in Table 5.

And then we can use the tool of AutoEvoFM to evolve Smart Home, Android SPL, Help System, SPL SimulES and Mobile Media.

B. RESULTS AND ANALYSIS

Using our tool and the data sets, we can get ten evolved feature models. We call evolved feature model as "E-SPL" for the sake of narrative convenience. Such as we can use "E-Smart Home" to represent evolved Smart Home. When using data sets from SPLOT, we will compare E-SPL with the higher version feature model. And when using data sets from Mobile Media, we will compare E-SPL with the next version

Table 4a. Evolutionary requirements of SPLs.

Name of SPL	Evolutionary requirements
Smart Home 1.0	A mandatory feature of heating management and all of its sub-features are changed to a new optional feature named HVAC management which has an optional sub-feature named heating control. Heating control has two sub-features named manual heating and automate heating. Manual heating is a mandatory feature. Automate heating is an optional feature with an optional sub-feature remote heating control.
	Optional feature electronic windows changes to an optional feature automated windows.
	Mandatory feature manual light, optional feature smart light and optional feature predefine values are deleted.
	Mandatory feature manual illumination and optional feature automated illumination are added to light management, and the feature has two or features named automated perimeter illumination and automate in-house illumination.
	Optional feature privacy with a sub-option feature named encryption algorithm is added to internet, where the sub-feature has two alternative features named RSA and DSE.
	Or feature music simulator is changed to an or feature AV simulator.
	The fire department in fire aid group is changed to a mandatory feature, and other group in fire aid group is changed to an optional feature.
	Alarm is changed to an optional feature.
	An optional feature named blinds management is added to smart home, and the feature has a sub mandatory feature named manual blinds and a sun optional feature named automate blinds.
	An optional feature AV management with a sub optional feature automate AV is added to smart home.
	An optional feature moods is added to smart home.
Android SPL	All sub-features of captious are deleted.
	An optional feature speak passwords is added to system.
	A mandatory feature named gesture recognition with a mandatory feature and two optional features is added to system. The sub optional feature are named whole setting and double click the screen to wakeup and sub mandatory feature is named letter gesture setting. And the sub mandatory feature has twelve sub or features named ophone, slides right contacts, mgallery, vfile manager, slides up settings, ccamera, wmusic, zmessaging, slides left email, slides down sound recorder, ebrowser, scalendar.
	A sub alternative feature named use system language2 is added to language 2.
	An optional feature advanced2 is added to text correction, the optional feature has two mandatory feature named key popup dismiss delay and keypress vibration duration, and advanced2 has a constraint of interdependence with advanced. The key popup dismiss delay has two sub alternative features named no delay and default2. And the keypress vibration duration has three sub alternative features named _0ms, _50ms, _100ms.
	The mandatory feature advanced1 is added to advanced, and advanced1 has a constraint of interdependence with advanced.
	The optional feature photos is changed to an optional feature named mira vision.
	The optional feature day dream and its sub-features are deleted
	The mandatory feature font and its sub-features are deleted
	The optional feature large and its sub-features are deleted
	The optional features use system language and block offensive words are added to android.
Help system 1.0	Or features PH, PI and PL are changed to mandatory features.

feature model. Such as E-Smart Home 2.0 is compared with Smart Home 3.0, and E-Mobile Media 2 is compared with Mobile Media 3. The results of the experiments are shown in Table 6.

Table 4b. ***(Continued.)*** **Evolutionary requirements of SPLs.**

SPL simulES 1.0	Delete the mandatory feature orcamento and its sub-features.
	The name of engenheirosde software is changed to funcionarios. The sub-feature custo is restructured with two or features named padrao and custo especificado funcionario. The sub-feature habilidade is restructured to a feature with two or features named padrao fico para todos and especificapor funcionario.
	New sub-mandatory feature named tipodas cartas with two or features named cartas com descricoed realistas and cartas simples are added to projeto.
	Two or features named de complexidade fixa and de complexidade variadas are added to bugs.
	The mandatory feature tiposde jogo and its sub-features are deleted.
	Add a sub mandatory feature ordemde realizacao with two or features named waterfall and livre to jogabilidade.
	Add a sub mandatory feature turnos with two or features named sequenciais and paralelos to jogabilidade.
	Add a sub mandatory feature taredas with two mandatory features named ordederealizicao and funcionarios to jogabilidade. Ordederealizicao has two or features named waterfall and livre, and funcionarios has two or features named cumprem qualquer tarefa and cumprem apenas tarefasde codigo.
Mobile Media 1	A new mandatory feature named view photo is added to count the number of times a photo has been viewed into the feature of photo and Optional feature of sorting photos by highest viewing frequency is added into media management. New mandatory feature edit label added to edit the photo's label in media management.
Mobile Media 2	New optional feature favourite added to allow users to specify and view their favourite photos. And in favourite set favourite and view favourite are two mandatory features.
Mobile Media 3	New optional feature copy media added to allow users to keep multiple copies of photos.
Mobile Media 4	New features are added to send photo to other users by mandatory feature send photo in the optional feature SMS as sub-features of media management
Mobile Media 5	New or feature music and its sub mandatory feature paly music added to store, play, and organise music. The management of media (e.g. create, delete and label) was turned into or features. All extended functionalities (e.g. sorting, favourites and SMS transfer) were also provided
Mobile Media 6	New or feature view and its sub mandatory feature play view added to manage videos

In Table 6, the list of ''To be evolved SPL'' is the name of feature model to be evolved, and they are same as the list of ''Name of SPL'' in table 4 and 5. The list of ''Evolved SPL'' shows the name of evolved SPL. And the list of ''Actual SPL'' is the higher version or next version feature model. The list of comparing results shows the results of comparing ''Evolved SPL'' with ''Actual SPL''. If they are same, the result is ''$\surd$'', and if they are different, the result is ''$\times$''.

It is clearly that there are eight ''$\surd$'', and two ''$\times$'' in the list of results. The eight results of ''$\surd$'' show that the evolved versions got by AutoEvoSPL proposed in this paper are same as actual versions. It means the evolved versions are correct. These experimental results can illustrate the effectiveness of the method proposed in this paper. They show that the method presented in this paper is relatively effective.

From the last two results in Table 6, the two versions of Mobile Media are evolved incorrectly. It is because the evolutionary requirements given on the website are incomplete. There are several changes achieved in the new version are not mentioned in the requirements. Such as it adds a new optional feature named ''Capture Photo'' to be a sub-feature of Photo in version 6. The change is not mentioned in the requirements. The incorrect results are not caused by our tool. The false results can show that the method is effective, and the method can also be used to test whether the software product evolves according to the requirement as another application.

IX. RELATED WORKS

In this section, we will give an overview of related works on SPL evolution. The three major types of research relate to our method are evolutions of SPL, feature models of SPL and changes of SPL feature models.

Table 5a. AutoEvoNeed of SPLs.

Name of SPL	AutoEvoNeed
Smart Home 1.0	$Re_{[_{SmartHome^M}\varnothing,F[_{HeatingManagement^M}\varnothing,[_{ManualHeating^M}]_{ManualHeating^M}[_{SmartHeating^O}]_{SmartHeating^O},(\varnothing,\varnothing)]_{HeatingManagement^M},(\varnothing,\varnothing)]_{SmartHome^M}-replace-[_{SmartHome^M}\varnothing,F[_{HVACManagement^O}[_{HeatingControl^O}[_{ManualHeating^M}]_{ManualHeating^M}[_{AutomatedHeating^O}[_{RemoteHeatingControl^O}]_{RemoteHeatingControl^O}]_{AutomatedHeating^O}]_{HeatingControl^O}]_{HVACManagement^O},(\varnothing,\varnothing)]_{SmartHome^M}$ $Re_{[_{WindowsManagement^M}\varnothing,F[_{ElectronicIWindows^O}]_{ElectronicIWindows^O},(\varnothing,\varnothing)]_{WindowsManagement^M}-replace-[_{WindowsManagement^M}\varnothing,F[_{AutomatedWindows^O}]_{AutomatedWindows^O},(\varnothing,\varnothing)]_{WindowsManagement^M}$ $Re_{[_{LightManagement^M}\varnothing,[_{ManualLight^M}]_{ManualLight^M}[_{SmartLight^O}]_{SmartLight^O}[_{Pre-definedValues^O}]_{Pre-definedValues^O},(\varnothing,\varnothing)]_{LightManagement^M}-delete-[_{LightManagement^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{LightManagement^M}$ $Re_{[_{LightManagement^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{LightManagement^M}-add-[_{LightManagement^M}\varnothing,[_{ManualIllumination^M}]_{ManualIllumination^M}[_{AutomatedIllumination^O}[_{AutomatedPerimeterIllumination^R}]_{AutomatedPerimeterIllumination^R}[_{AutomatedIn-houseIllumination^R}]_{AutomatedIn-houseIllumination^R}]_{AutomatedIllumination^O},(\varnothing,\varnothing)]_{LightManagement^M}$ $Re_{[_{Internet^O}\varnothing,\varnothing,(\varnothing,\varnothing)]_{Internet^O}-add-[_{Internet^O}\varnothing,[_{Privacy^O}[_{EncryptionAlgorithm^O}[_{RSA^A}]_{RSA^A}[_{DSE^A}]_{DSE^A}]_{EncryptionAlgorithm^O}]_{Privacy^O},(\varnothing,\varnothing)]_{Internet^O}$ $Re_{[_{PresenceSimulator^O}\varnothing,F[_{MusicSimulator^R}]_{MusicSimulator^R},(\varnothing,\varnothing)]_{PresenceSimulator^O}-replace-[_{PresenceSimulator^O}\varnothing,F[_{AVSimulator^R}]_{AVSimulator^R},(\varnothing,\varnothing)]_{PresenceSimulator^O}$ $Re_{[_{FireAidGroup^M}\varnothing,[_{FireDepartment^R}]_{FireDepartment^R}[_{otherGroup^R}]_{otherGroup^R},(\varnothing,\varnothing)]_{FireAidGroup^M}-replace-[_{FireAidGroup^M}\varnothing,[_{FireDepartment^M}]_{FireDepartment^M}[_{otherGroup^O}]_{otherGroup^O},(\varnothing,\varnothing)]_{FireAidGroup^M}$ $Re_{[_{SmartHome^M}\varnothing,F[_{Alarm^M}\varnothing,F,(\varnothing,\varnothing)]_{Alarm^M},(\varnothing,\varnothing)]_{SmartHome^M}-replace-[_{SmartHome^M}\varnothing,F[_{Alarm^O}\varnothing,F,(\varnothing,\varnothing)]_{Alarm^O},(\varnothing,\varnothing)]_{SmartHome^M}$ $Re_{[_{SmartHome^M}\varnothing,F,(\varnothing,\varnothing)]_{SmartHome^M}-add-[_{SmartHome^M}\varnothing,F[_{AVManagement^O}[_{AutomatedAV^O}]_{AutomatedAV^O}]_{AVManagement^O},(\varnothing,\varnothing)]_{SmartHome^M}$ $Re_{[_{SmartHome^M}\varnothing,F,(\varnothing,\varnothing)]_{SmartHome^M}-add-[_{SmartHome^M}\varnothing,F[_{Moods^O}]_{Moods^O},(\varnothing,\varnothing)]_{SmartHome^M}$
Android SPL	$Re_{[_{Captions^O}\varnothing,F,(\varnothing,\varnothing)]_{Captions^O}-delete-[_{Captions^O}\varnothing,\varnothing,(\varnothing,\varnothing)]_{Captions^O}$ $Re_{[_{System^M}\varnothing,F,(\varnothing,\varnothing)]_{System^M}-add-[_{System^M}\varnothing,F[_{SpeakPasswords^O}]_{SpeakPasswords^O},(\varnothing,\varnothing)]_{System^M}$ $Re_{[_{System^M}\varnothing,F,(\varnothing,\varnothing)]_{System^M}-add-[_{System^M}\varnothing,F[_{GestureRecognition^M}[_{WholeSetting^O}]_{WholeSetting^O}[_{DoubleClickTheScreenToWakeup^O}]_{DoubleClickTheScreenToWakeup^O}[_{LetterGestureSettings^M}[_{oPhone^R}]_{oPhone^R}[_{SlidesRightContacts^R}]_{SlidesRightContacts^R}[_{mGallery^R}]_{mGallery^R}[_{vFileManager^R}]_{vFileManager^R}[_{SlidesUpSettings^R}]_{SlidesUpSettings^R}[_{cCamera^R}]_{cCamera^R}[_{wMusic^R}]_{wMusic^R}[_{zMessaging^R}]_{zMessaging^R}[_{SlidesLeftEmail^R}]_{SlidesLeftEmail^R}[_{SlidesDownSoundRecorder^R}]_{SlidesDownSoundRecorder^R}[_{eBrowser^R}]_{eBrowser^R}[_{sCalendar^R}]_{sCalendar^R}]_{LetterGestureSettings^M}]_{GestureRecognition^M},(\varnothing,\varnothing)]_{System^M}$ $Re_{[_{Language2^M}\varnothing,F,(\varnothing,\varnothing)]_{Language2^M}-add-[_{Language2^M}\varnothing,F[_{UseSystemLanguage2^A}]_{UseSystemLanguage2^A},(\varnothing,\varnothing)]_{Language2^M}$ $Re_{[_{TextCorrection^M}\varnothing,F,(\varnothing,\varnothing)]_{TextCorrection^M}-add-[_{TextCorrection^M}\varnothing,F[_{Advanced2^O}\mathrm{L},[_{KeyPopupDismissDelay^M}[_{NoDelay^A}]_{NoDelay^A}[_{Default2^A}]_{Default2^A}]_{KeyPopupDismissDelay^M}[_{KeypressVibrationDuration^M}[_{0ms^A}]_{0ms^A}[_{-50ms^A}]_{50ms^A}[_{-100ms^A}]_{100ms^A}]_{KeypressVibrationDuration^M}[_{KeypressSoundVolume^M}]_{KeypressSoundVolume^M},(\mathrm{L},\varnothing)]_{Advanced2^O},(\varnothing,\varnothing)[_{TextCorrection^M}$ $Re_{[_{Advanced^M}\varnothing,F,(\varnothing,\varnothing)]_{Advanced^M}-add-[_{Advanced^M}\varnothing,[_{Advanced1^M}\mathrm{L},\varnothing,(\mathrm{L},\varnothing)]_{Advanced1^M}F,(\varnothing,\varnothing)]_{Advanced^M}$ $Re_{[_{Display2^O}\varnothing,F[_{Photos^O}]_{Photos^O},(\varnothing,\varnothing)]_{Display2^O}-replace-[_{Display2^O}\varnothing,F[_{MiraVision^O}]_{MiraVision^O},(\varnothing,\varnothing)]_{Display2^O}$ $Re_{[_{Display2^O}\varnothing,F[_{Daydream^O}[_{Colors^A}]_{Colors^A}[_{PhotoTable^A}]_{PhotoTable^A}[_{Photos1^A}]_{Photos1^A}[_{Clock^A}]_{Clock^A}[_{PhotoFrame^A}]_{PhotoFrame^A}]_{Daydream^O},(\varnothing,\varnothing)]_{Display2^O}-delete-[_{Display2^O}\varnothing,F,(\varnothing,\varnothing)]_{Display2^O}$ $Re_{[_{Display2^O}\varnothing,F[_{Font^M}[_{Online^A}[_{Redflow^A}]_{Redflow^A}[_{Handwind^A}]_{Handwind^A}[_{Xiaoming^A}]_{Xiaoming^A}[_{Glamp^A}]_{Glamp^A}[_{Glint^A}]_{Glint^A}[_{GioFont^A}]_{GioFont^A}[_{Gardenia^A}]_{Gardenia^A}[_{Silicone^A}]_{Silicone^A}[_{Monkey^A}]_{Monkey^A}]_{Online^A}[_{Local^A}[_{Olive^A}]_{Olive^A}[_{Appaly^A}]_{Appaly^A}[_{Default1^A}]_{Default1^A}]_{Local^A}]_{Font^M},(\varnothing,\varnothing)]_{Display2^O}-delete-[_{Display2^O}\varnothing,F,(\varnothing,\varnothing)]_{Display2^O}$ $Re_{[_{Android^M}\varnothing,F[_{Large^O}]_{Large^O}[_{Default^O}]_{Default^O}[_{Small^O}]_{Small^O},(\varnothing,\varnothing)]_{Android^M}-delete-[_{Android^M}\varnothing,F,(\varnothing,\varnothing)]_{Android^M}$ $Re_{[_{Android^M}\varnothing,F,(\varnothing,\varnothing)]_{Android^M}-add-[_{Android^M}\varnothing,F[_{UseSystemLanguage^O}]_{UseSystemLanguage^O}[_{BlockOffensiveWords^O}]_{BlockOffensiveWords^O},(\varnothing,\varnothing)]_{Android^M}$
Help System 1.0	$Re_{[_{Priority^M}\varnothing,[_{PH^R}]_{PH^R}[_{PI^R}]_{PI^R}[_{PL^R}]_{PL^R},(\varnothing,\varnothing)]_{Priority^M}-replace-[_{Priority^M}\varnothing,[_{PH^M}]_{PH^M}[_{PI^M}]_{PI^M}[_{PL^M}]_{PL^M},(\varnothing,\varnothing)]_{Priority^M}$
SPL SimulES 1.0	$Re_{[_{Projeto^M}\varnothing,F[_{Orcamento^M}[_{OrcamentoFixoPadrao^R}]_{OrcamentoFixoPadrao^R}[_{OrcamentoEspecificaporProjeto^R}]_{OrcamentoEspecificaporProjeto^R}]_{Orcamento^M},(\varnothing,\varnothing)]_{Projeto^M}-delete-[_{Projeto^M}\varnothing,F,(\varnothing,\varnothing)]_{Projeto^M}$ $Re_{[_{Cartas^M}\varnothing,F[_{EngenheirosdeSoftware^M}[_{Custo^M}[_{CustoPadraoParaTodososEngenheiros^R}]_{CustoPadraoParaTodososEngenheiros^R}[_{CustoEspecificaporEngenheiro^R}]_{CustoEspecificaporEngenheiro^R}]_{Custo^M}[_{Habilidade^M}[_{HabilidadePadraoParaTodososEngenheiros^R}]_{HabilidadePadraoParaTodososEngenheiros^R}[_{HabilidadeEspecificaporEngenheiro^R}]_{HabilidadeEspecificaporEngenheiro^R}]_{Habilidade^M}[_{Maturidade^O}]_{Maturidade^O}]_{EngenheirosdeSoftware^M},(\varnothing,\varnothing)]_{Cartas^M}-replace-[_{Cartas^M}\varnothing,F[_{Funcionarios^M}[_{Custo^M}[_{Padrao^R}]_{Padrao^R}[_{CustoEspecificadoFuncionario^R}]_{CustoEspecificadoFuncionario^R}]_{Custo^M}[_{Habilidade^M}[_{PadraoFixoParaTodos^R}]_{PadraoFixoParaTodos^R}[_{EspecificaporFuncionario^R}]_{EspecificaporFuncionario^R}]_{Habilidade^M}[_{Maturidade^O}]_{Maturidade^O}]_{Funcionarios^M},(\varnothing,\varnothing)]_{Cartas^M}$

Table 5b. ***(Continued.)*** **AutoEvoNeed of SPLs.**

	$Re_{[_{Projeto^M}\varnothing,F,(\varnothing,\varnothing)]_{Projeto^M}-add-[_{Projeto^M}\varnothing,F[_{TipodasCartas^M}[_{CartasComDescricoesRealistas^R}]_{CartasComDescricoesRealistas^R}[_{CartasSimples^R}]_{CartasSimples^R}]_{TipodasCartas^M},(\varnothing,\varnothing)]_{Projeto^M}}$ $Re_{[_{Bugs^O}\varnothing,\varnothing,(\varnothing,\varnothing)]_{Bugs^O}-add-[_{Bugs^O}\varnothing,[_{DeComplexidadeFixa^R}]_{DeComplexidadeFixa^R}[_{DeComplexidadeVariadas^R}]_{DeComplexidadeVariadas^R},(\varnothing,\varnothing)]_{Bugs^O}}$ $Re_{[_{Jogabilidade^M}\varnothing,F[_{TiposdeJogo^M}[_{SinglePlayer^R}]_{SinglePlayer^R}[_{MultiPlayer^R}[_{JogoporIP^R}]_{JogoporIP^R}[_{JogoviaConexaoComServidor^R}]_{JogoviaConexaoComServidor^R}]_{MultiPlayer^R}]_{TiposdeJogo^M},(\varnothing,\varnothing)]_{Jogabilidade^M}-delete-[_{Jogabilidade^M}\varnothing,F,(\varnothing,\varnothing)]_{Jogabilidade^M}}$ $Re_{[_{Jogabilidade^M}\varnothing,F,(\varnothing,\varnothing)]_{Jogabilidade^M}-add-[_{Jogabilidade^M}\varnothing,F[_{OrdemdeRealizacao^M}[_{Waterfall^R}]_{Waterfall^R}[_{Livre^R}]_{Livre^R}]_{OrdemdeRealizacao^M},(\varnothing,\varnothing)]_{Jogabilidade^M}}$ $Re_{[_{Jogabilidade^M}\varnothing,F,(\varnothing,\varnothing)]_{Jogabilidade^M}-add-[_{Jogabilidade^M}\varnothing,F[_{Turnos^M}[_{Sequenciais^R}]_{Sequenciais^R}[_{Paralelos^R}]_{Paralelos^R}]_{Turnos^M},(\varnothing,\varnothing)]_{Jogabilidade^M}}$ $Re_{[_{Jogabilidade^M}\varnothing,F,(\varnothing,\varnothing)]_{Jogabilidade^M}-add-[_{Jogabilidade^M}\varnothing,F[_{Tarefas^M}[_{OrdemdeRealizacao^M}[_{Waterfall^R}]_{Waterfall^R}[_{Livre^R}]_{Livre^R}]_{OrdemdeRealizacao^M}[_{Funcionarios^M}[_{CumpremQualquerTarefa^R}]_{CumpremQualquerTarefa^R}[_{CumpremApenasTarefasdeCodigo^R}]_{CumpremApenasTarefasdeCodigo^R}]_{Funcionarios^M}]_{Tarefas^M},(\varnothing,\varnothing)]_{Jogabilidade^M}}$
Mobile Media 1	$Re_{[_{Photo^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{Photo^M}-add-[_{Photo^M}\varnothing,[_{ViewPhoto^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{ViewPhoto^M}\varnothing,(\varnothing,\varnothing)]_{Photo^M}}$, $Re_{[_{MediaManagement^M}\varnothing,F,(\varnothing,\varnothing)]_{MediaManagement^M}-add-[_{MediaManagement^M}\varnothing,[_{Sorting^O}\varnothing,\varnothing,(\varnothing,\varnothing)]_{Sorting^O}F,(\varnothing,\varnothing)]_{MediaManagement^M}}$, $Re_{[_{MediaManagement^M}\varnothing,F,(\varnothing,\varnothing)]_{MediaManagement^M}-add-[_{MediaManagement^M}\varnothing,[_{EditLabel^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{EditLabel^M}F,(\varnothing,\varnothing)]_{MediaManagement^M}}$
Mobile Media 2	$Re_{[_{MediaManagement^M}\varnothing,F,(\varnothing,\varnothing)]_{MediaManagement^M}-add-[_{MediaManagement^M}\varnothing,[_{Favourite^O}\varnothing,[_{SetFavourite^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{SetFavourite^M}[_{ViewFavourite^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{ViewFavourite^M},(\varnothing,\varnothing)]_{Favourite^O}F,(\varnothing,\varnothing)]_{MediaManagement^M}}$
Mobile Media 3	$Re_{[_{MediaManagement^M}\varnothing,F,(\varnothing,\varnothing)]_{MediaManagement^M}-add-[_{MediaManagement^M}\varnothing,[_{CopyMedia^O}\varnothing,\varnothing,(\varnothing,\varnothing)]_{CopyMedia^O}F,(\varnothing,\varnothing)]_{MediaManagement^M}}$
Mobile Media 4	$Re_{[_{MediaManagement^M}\varnothing,F,(\varnothing,\varnothing)]_{MediaManagement^M}-add-[_{MediaManagement^M}\varnothing,[_{SMS^O}\varnothing,[_{SendPhoto^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{SendPhoto^M},(\varnothing,\varnothing)]_{SMS^O}F,(\varnothing,\varnothing)]_{MediaManagement^M}}$
Mobile Media 5	$Re_{[_{Media^M}\varnothing,F,(\varnothing,\varnothing)]_{Media^M}-add-[_{Media^M}\varnothing,[_{Music^R}\varnothing,[_{PlayMusic^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{PlayMusic^M},(\varnothing,\varnothing)]_{Music^R}F,(\varnothing,\varnothing)]_{Media^M}}$ $Re_{[_{MediaManagement^M}\varnothing,[_{CreateMedia^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{CreateMedia^M}F,(\varnothing,\varnothing)]_{MediaManagement^M}-replace-[_{MediaManagement^M}\varnothing,[_{CreateMedia^R}\varnothing,\varnothing,(\varnothing,\varnothing)]_{CreateMedia^R}F,(\varnothing,\varnothing)]_{MediaManagement^M}}$ $Re_{[_{MediaManagement^M}\varnothing,[_{DeleteMedia^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{DeleteMedia^M}F,(\varnothing,\varnothing)]_{MediaManagement^M}-replace-[_{MediaManagement^M}\varnothing,[_{DeleteMedia^R}\varnothing,\varnothing,(\varnothing,\varnothing)]_{DeleteMedia^R}F,(\varnothing,\varnothing)]_{MediaManagement^M}}$ $Re_{[_{MediaManagement^M}\varnothing,[_{EditLabel^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{EditLabel^M}F,(\varnothing,\varnothing)]_{MediaManagement^M}-replace-[_{MediaManagement^M}\varnothing,[_{EditLabel^R}\varnothing,\varnothing,(\varnothing,\varnothing)]_{EditLabel^R}F,(\varnothing,\varnothing)]_{MediaManagement^M}}$
Mobile Media 6	$Re_{[_{Media^M}\varnothing,F,(\varnothing,\varnothing)]_{Media^M}-add-[_{Media^M}\varnothing,[_{View^R}\varnothing,[_{PlayView^M}\varnothing,\varnothing,(\varnothing,\varnothing)]_{PlayView^M},(\varnothing,\varnothing)]_{View^R}F,(\varnothing,\varnothing)]_{Media^M}}$

In SPL evolution, the articles on SPL evolution fall into three categories: evolve code [13], [17], [20], [24], evolve architecture [11], [12], [16], [30] and verify change [2], [8], [32], [34]. In evolving code, it needs to determine the range location of code. Heider et al. used regression tests to determine the range of evolution and presented a tool-supported to add code among the range on existing SPL [17]. Mende et al. used grow-and-prune model to locate code by identifying similar functionality in SPL [24]. In evolving architecture, Deng *et al.* [11] proposed techniques for minimizing such impacts on software product line architectures based on model-driven development for distributed real-time and embedded systems. Garg *et al.* [16] presented Ménage, an environment specifically designed to manage evolving structure of SPL. Polzer *et al.* gave the method of integrating model-based product line techniques into a consistent automated framework. The framework was an abstraction of SPL architecture and supported for customizing representations [30]. In verifying change, Anquetil *et al.* [2] created a common traceability framework across the various activities of the SPL development, it can trace the changes in SPL. Teixeira *et al.* [34] proposed refinement notions and compositionality properties to formally define the foundations for the safe and modular evolution of SPL, enabling developers to perform changes in a systematic manner.

In feature models of SPL, Neves *et al.* [26] told us a feature model is a description of SPL architecture. Collecting recent articles about SPL evolution, Montalvillo and Díaz [25] found that feature models have been the most important topic and divided the articles on SPL evolution into four categories: identify change, analyze and plan change, implement change and verify change. Ferber *et al.* [14] gave a method of building a feature model from the existing SPL,

Table 6. Experimental results of SPLs.

To be evolved SPL	Evolved SPL	Actual SPL	Comparing Results
Smart Home 2.0	E-Smart Home 2.0	Smart Home 3.0	√
Android SPL 5.0	E-Android SPL 5.0	Android SPL 6.0	√
Help System 1.0	E-Help System 1.0	Help System 2.0	√
SPL SimulES 1.0	E-SPL SimulES 1.0	SPL SimulES 2.0	√
Mobile Media 1	E-Mobile Media 1	Mobile Media 2	√
Mobile Media 2	E-Mobile Media 2	Mobile Media 3	√
Mobile Media 3	E-Mobile Media 3	Mobile Media 4	√
Mobile Media 4	E-Mobile Media 4	Mobile Media 5	√
Mobile Media 5	E-Mobile Media 5	Mobile Media 6	×
Mobile Media 6	E-Mobile Media 6	Mobile Media 7	×

and they proposed that the feature dependencies and interactions should be added into a feature model to describe the complex relationship among features. Seidl *et al.* said SPL is often described in terms of a problem space and a solution space. The problem space is research on the method of using feature models to represent SPL, and the solution space is some research on sharing assets such as source code, design and test artifacts in SPL [33]. Neves *et al.* said SPL consists of feature models, configuration information and assets. They gave the evolution model of multiple evolutionary scenarios. They used the evolution model to determine the influence scape of evolution. And then they found the evolution assets among the scape by the configuration. This process is called security evolution method, it can ensure the correctness of the evolution [26], [27]. White *et al.* [36] gave the method of how to configure information evolution of SPL based on the feature models, and gave the method of evaluating the cost of SPL evolution with feature models.

And in changes of SPL's feature models, Nie *et al.* gave a method of model construction based on model difference and comparison of SPL using domain requirement. Extracting features from the domain requirement and comparing the features with the feature model, they achieved adding new features from domain requirements to feature models [22]. Pleuss *et al.* gave the method of managing the feature model of SPL using the model-driven knowledge. They used model fragments of cluster-related elements to determine the scope of changes. During the scope, it could add or remove feature [28]. Cordy *et al.* proposed a model-checking approach to support the evolution of models. And the approach supported a change of adding specific types of feature to an evolving SPL [10]. Botterweck *et al.* [5], [6] proposed using feature models to describe the evolution of product lines to integrate evolution into model-driven product line engineering.

In the articles of SPL evolution and feature models of SPL, we can find that the feature models are usually used to determine the influence scope of evolution to guide SPL evolution. But a feature model cannot guide evolution continually. It needs to evolve feature models as well. And among the articles of feature models changes, there is no method for evolving feature models in all evolving ways from evolution requirements. In Nie's article, the method they proposed can deal with adding features from requirements to feature models. It cannot delete features or change features. In Pleuss and Cordy's articles, they gave the method of changing feature models. And in Botterweck's articles, they also study the evolution of feature models, but the focus of this article is on how to determine the scope of the change, then change the feature model. They But both of the three articles did not tell us where the changes come. We do not know whether these changes meet the evolutionary requirements.

X. CONCLUSION

An SPL feature model can guide the evolution of SPL. To ensure the continuous availability of the feature model, the evolution of SPL requires the evolution of feature model according to evolutionary requirements. Most traditional SPL's feature models rely on the analysis of domain experts. At the same time, each SPL model needs domain experts to reconstruct the feature model when the SPL is evolved. This refactoring process entirely relies on domain experts' understanding of SPL and evolutionary requirements, and cannot guarantee the correctness of the feature model. This paper presents a method of automatically generating SPL's feature models with evolutionary requirements. We use a formal method named communication membrane calculus to describe the structure of feature models and evolution process of feature models. After describe feature models with the three tuple of feature model and evolutionary requirements with three-tuple evolutionary needs, the feature models can be changed automatically by evolutionary requirements, and the evolved feature model is got as the result of changing.

However, the tool of generating feature models automatically in this paper is too simple, it cannot support all change mechanisms of CMC, and it is proposed for the particular given environment or some target condition, it is not the general proposal. The tree structure of feature models cannot be directly used in the tool, and the results of the tool cannot change to the tree structure of feature models too. The method just find the mistakes when evolve a feature model, it cannot tell us the reason of mistakes and how to handle these conflicts. The visualization of the tool is simple. And Auto-EvoSPL requires evolutionary requirements to be integrity. Every change must be concluded in the requirements. We will continue to study the above issues, improve the experimental

tools, optimize the tool interface, and increase the demand dynamic replenishment mechanism.

REFERENCES

[1] V. Alves, P. M. Jr, L. Cole, A. Vasconcelos, P. Borba, and G. Ramalho, "Extracting and evolving code in product lines with aspect-oriented programming," in *Proc. Trans. Aspect-Oriented Softw. Develop. IV*, 2007, pp. 117–142.

[2] N. Anquetil *et al.*, "A model-driven traceability framework for software product lines," *Softw. Syst. Model.*, vol. 9, no. 4, pp. 427–451, 2010.

[3] S. Apel and D. Batory, "When to use features and aspects?: A case study," in *Proc. 5th Int. Conf. Generative Program. Compon. Eng.*, 2006, pp. 59–68.

[4] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated reasoning on feature models," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.*, 2005, pp. 491–503.

[5] G. Botterweck, A. Pleuss, A. Polzer, and S. Kowalewski, "Towards feature-driven planning of product-line evolution," in *Proc. 1st Int. Workshop Feature-Oriented Softw. Develop.*, Denver, Colorado, USA, Oct. 2009, pp. 109–116.

[6] G. Botterweck, A. Pleuss, D. Dhungana, A. Polzer, and S. Kowalewski, "EvoFM: Feature-driven planning of product-line evolution," in *Proc. ICSE Workshop Product Line Approaches Softw. Eng.*, 2010, pp. 24–31.

[7] N. Cacho, C. Sant'Anna, E. Figueiredo, A. Garcia, T. Batista, and C. Lucena, "Composing design patterns: A scalability study of aspect-oriented programming," in *Proc. 5th Int. Conf. Aspect-Oriented Softw. Develop.*, 2006, pp. 109–121.

[8] L. Chen and M. A. Babar, "A systematic review of evaluation of variability management approaches in software product lines," *Inf. Softw. Technol.*, vol. 53, no. 4, pp. 344–362, 2011.

[9] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. New York, NY, USA: ACM, 2001.

[10] M. Cordy, A. Classen, P.-Y. Schobbens, P. Heymans, and A. Legay, "Managing evolution in software product lines: A model-checking perspective," in *Proc. 6th Int. Workshop Variability Model. Softw.-Intensive Syst.*, 2012, pp. 183–191.

[11] G. Deng, G. Lenz, and D. C. Schmidt, "Addressing domain evolution challenges in software product lines," in *Proc. Int. Conf. Model Driven Eng. Lang. Syst.*, 2005, pp. 247–261.

[12] J. Díaz, J. Pérez, and J. Garbajosa, "Agile product-line architecting in practice: A case study in smart grids," *Inf. Softw. Technol.*, vol. 56, no. 7, pp. 727–748, 2014.

[13] R. Dyer, H. Rajan, and Y. Cai, "Language features for software evolution and aspect-oriented interfaces: An exploratory study," in *Transactions on Aspect-Oriented Software Development X*. Berlin, Germany: Springer, 2013, pp. 148–183.

[14] S. Ferber, J. Haag, and J. Savolainen, "Feature interaction and dependencies: Modeling features for reengineering a legacy product line," in *Proc. Int. Conf. Softw. Product Lines*, 2002, pp. 235–256.

[15] E. Figueiredo *et al.*, "Evolving software product lines with aspects: An empirical study on design stability," in *Proc. ACM/IEEE 30th Int. Conf. Softw. Eng.*, May 2008, pp. 261–270.

[16] A. Garg, M. Critchlow, P. Chen, C. Van Der Westhuizen, and A. Van Der Hoek, "An environment for managing evolving product line architectures," in *Proc. Int. Conf. Softw. Maintenance,*, Sep. 2003, pp. 358–367.

[17] W. Heider, R. Rabiser, P. Grünbacher, D. Lettner, "Using regression testing to analyze the impact of changes to variability models on products," in *Proc. 16th Int. Softw. Product Line Conf.*, 2012, pp. 196–205.

[18] R. Heradio, H. Perez-Morago, D. Fernandez-Amoros, F. J. Cabrerizo, and E. Herrera-Viedma, "A bibliometric analysis of 20 years of research on software product lines," *Inf. Softw. Technol.*, vol. 72, pp. 1–15, Apr. 2016.

[19] R. M. Hierons, M. Li, X. Liu, S. Segura, and W. Zheng, "SIP: Optimal product selection from feature models using many-objective evolutionary optimization," *Trans. Softw. Eng. Methodol.*, vol. 25, no. 2, p. 17, 2016.

[20] S. Jarzabek and H. D. Trung, "Flexible generators for software reuse and evolution: NIER track," in *Proc. 33rd Int. Conf. Softw. Eng. (ICSE)*, May 2011, pp. 920–923.

[21] C. Kastner, S. Apel, and D. Batory, "A case study implementing features using aspectJ," in *Proc. 11th Int. Softw. Product Line Conf.*, Sep. 2007, pp. 223–232.

[22] K. Nie and L. Zhang, "Software product line domain requirement model construction method based on model difference and model composition," *Chin. J. Comput.*, vol. 37, no. 3, pp. 539–550, 2014.

[23] Y. Li and W. Zhao, "A feature oriented approach to mapping from domain requirements to product line architecture," *J. Comput. Res. Develop.*, vol. 44, no. 7, p. 1236, 2007.

[24] T. Mende, F. Beckwermert, R. Koschke, and G. Meier, "Supporting the grow-and-prune model in software product lines evolution using clone detection," in *Proc. 12th Eur. Conf. Softw. Maintenance Reeng.*, Apr. 2008, pp. 163–172.

[25] L. Montalvillo and O. Díaz, "Requirement-driven evolution in software product lines: A systematic mapping study," *J. Syst. Softw.*, vol. 122, pp. 110–143, Dec. 2016.

[26] L. Neves *et al.*, "Safe evolution templates for software product lines," *J. Syst. Softw.*, vol. 106, pp. 42–58, Aug. 2015.

[27] L. Neves, L. Teixeira, D. Sena, V. Alves, U. Kulesza, and P. Borba, "Investigating the safe evolution of software product lines," *ACM SIGPLAN Notices*, vol. 47, no. 3, pp. 33–42, 2012.

[28] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski, "Model-driven support for product line evolution on feature level," *J. Syst. Softw.*, vol. 85, no. 10, pp. 2261–2274, 2012.

[29] K. Pohl, G. Böckle, and F. J. Van Der Linden, "Software Product Line Engineering: Foundations, Principles, and Techniques," in *Proc. 1st Int. Workshop Formal Methods Softw. Product Line Eng.*,2005, vol. 49, no. 12, pp. 29–32.

[30] A. Polzer *et al.*, "Managing complexity and variability of a model-based embedded software product line," *Innov. Syst. Softw. Eng.*, vol. 8, no. 1, pp. 35–49, 2012.

[31] J. Ren, L. Liu, and P. Zhang, "A calculus for modeling the process of evolution: Communication membrane calculus.," *J. Harbin Eng. Univ.*, vol. 39, no. 4, pp. 751–759, 2018.

[32] H. Sabouri and R. Khosravi, *Reducing the Verification Cost of Evolving Product Families Using Static Analysis Techniques*. Amsterdam, The Netherlands: Elsevier, 2014. pp. 35–55.

[33] C. Seidl, F. Heidenreich, and U. Aßmann, "Co-evolution of models and feature mapping in software product lines," in *Proc. 16th Int. Softw. Product Line Conf.*, 2012, vol. 1, pp. 76–85.

[34] L. Teixeira, P. Borba, and R. Gheyi, "Safe evolution of product populations and multi product lines," in *Proc. 19th Int. Conf. Softw. Product Line*, 2015, pp. 171–175.

[35] T. Vale, E. S. De Almeida, V. Alves, U. Kulesza, N. Niu, and R. De Lima, "Software product lines traceability: A systematic mapping study," *Inf. Softw. Technol.*, vol. 84, pp. 1–18, Apr. 2017.

[36] J. White, J. A. Galindo, T. Saxena, B. Dougherty, D. Benavides, and D. C. Schmidt, "Evolving feature model configurations in software product lines," *J. Syst. Softw.*, vol. 87, no. 1, pp. 119–136, 2014.

JUNQI REN received the B.S. degree from the Mathematics School, Jilin University, Changchun, China, in 2012, where she is pursuing the Ph.D. degree in computer software and theory. Her research interests include formal methods and software engineering.

LEI LIU received the B.S. and M.S. degrees in computer software and theory from Jilin University, China, in 1982 and 1985, respectively, where he is currently a Professor and a Doctoral Supervisor with the College of Computer Science and Technology. His research interests include program theory, semantic web, formal methods, and compiler theory.

PENG ZHANG received the B.S. and Ph.D. degrees in computer science from Jilin University, China, in 2009 and 2014, respectively, where he is currently a Lecturer with the College of Computer Science and Technology. His research interests include formal methods and cloud computing.

WENBO ZHOU received the M.S. degree in computer software and theory from Jilin University, China, in 2017, where he is currently pursuing the Ph.D. degree with the College of Computer Science and Technology. His research interests include formal methods and cloud computing.