Information and Software Technology 80 (2016) 138–157

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Automating feature model refactoring: A Model transformation approach

Mohammad Tanhaei, Jafar Habibi*, Seyed-Hassan Mirian-Hosseinabadi

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

ARTICLE INFO

Article history:
Received 24 January 2016
Revised 25 July 2016
Accepted 30 August 2016
Available online 3 September 2016

Keywords:
Feature model refactoring
Model transformation & refactoring

ABSTRACT

Context: Feature model is an appropriate and indispensable tool for modeling similarities and differences among products of the Software Product Line (SPL). It not only exposes the validity of the products' configurations in an SPL but also changes in the course of time to support new requirements of the SPL. Modifications made on the feature model in the course of time raise a number of issues. Useless enlargements of the feature model, the existence of dead features, and violated constraints in the feature model are some of the key problems that make its maintenance difficult.

Objective: The initial approach to dealing with the above-mentioned problems and improving maintainability of the feature model is refactoring. Refactoring modifies software artifacts in a way that their externally visible behavior does not change.

Method: We introduce a method for defining refactoring rules and executing them on the feature model. We use the ATL model transformation language to define the refactoring rules. Moreover, we provide an Alloy model to check the feature model and the safety of the refactorings that are performed on it.

Results: In this research, we propose a safe framework for refactoring a feature model. This framework enables users to perform automatic and semi-automatic refactoring on the feature model.

Conclusions: Automated tool support for refactoring is a key issue for adopting approaches such as utilizing feature models and integrating them into the software development process of companies. In this work, we define some of the important refactoring rules on the feature model and provide tools that enable users to add new rules using the ATL M2M language. Our framework assesses the correctness of the refactorings using the Alloy language.

1. Introduction

Software product lines are known as an efficient way for developing a variety of related products in a specific domain [1]. One of the important properties of the SPL is managed reuse in a particular domain [2]. The feature model is one of the key elements in managing the core assets of the SPL. It can model similarities and variabilities in an SPL. The feature model is first introduced in the Feature-Oriented Domain Analysis (FODA) approach [3].

Feature models have been widely used in engineering as well as the development of software product lines since their introduction [4]. They are used in a variety of software product line development approaches [5] ranging from model-driven development [6] to software factories [7] to feature-oriented programming [8] to generative programming [9].

Feature model utilizes a tree-like structure to represent the variations in the SPL. A root feature can own some children, and the children can have their children too. Each feature of the feature model has some relations to other features and has some constraints in appearing in the configurations. To include a feature in a product configuration, one should meet all constraints related to that feature.

By the passage of time, external obligations and new requirements lead the feature model to change. In this situation, two different approaches can be used to deal with the changes at the feature model level.

– **Clean Approach:** Perform domain analysis and survey the feature model for the optimized location of changes and add support for new requirements to it.

* Corresponding author.
E-mail addresses: tanhaei@ce.sharif.edu (M. Tanhaei), jhabibi@sharif.edu (J. Habibi), hmirian@sharif.edu (S.-H. Mirian-Hosseinabadi).

http://dx.doi.org/10.1016/j.infsof.2016.08.011
0950-5849/

– **Dirty Approach:** Change the feature model in a way that the new requirement can be supported. This approach has higher speed and lower accuracy compared to the Clean Approach.

While the best way to deal with the changes in the SPL and especially the feature model is to use the clean approach, the changes on the feature model are usually performed on a restricted and local area without surveying the effect of the change on other relations and constraints of the feature model (dirty approach). That usually happens because of time pressure and cost constraints in SPL development approaches. After performing some modifications on the feature model and by gathering together the changes, several types of problems in the feature model might appear [10]. The extra effort that we have to do in future because of using the dirty development approach is known as the technical debt [11]. Features that cannot be present in any configuration, features that are misallocated, features with the wrong type, and useless integrity constraints are some examples of the problems in a feature model that are developed using the dirty approach.

In these situations, we need to perform some changes on the feature model, changes that do not alter the validity of the product configurations and improve the overall structure of the feature model and resolve the problems mentioned above. We refer to these modifications as "feature model refactoring", each being an alteration in the feature model that does not change the validity of its configurations but improves its non-functional properties. Refactoring consists of small steps, which when aggregated can bring about a significant change in the software artifacts [12]. Every organization that selects the dirty approach for developing a product line needs to perform some refactorings on the code, design, architecture as well as the feature model at a time. Refactoring is one of the ways to pay the technical debt [11]. The first step in performing refactoring is to find a precise way of defining and executing the refactoring rules. And the second step is to check the correctness of the modifications done on the feature model. Our framework helps with performing these two steps.

For a motivating example of refactoring a feature model, consider an SPL which has a feature named *simple access control* for authenticating users. By the passage of time, some set of new features is developed in the SPL that cannot be supported by the old way of checking access control. As a result, the development team adds a new feature to the feature model to support the requirements of the new feature named *advanced access control* which is an alternative to the *simple access control* feature. Every new product containing the newly developed features needs to use the *advanced access control* feature instead of the *simple access control* feature. In time, if one of the features which require the *advanced access control* feature to operate correctly becomes mandatory in the feature model, the *simple access control* becomes completely useless and cannot be present in any future product. Feature model refactoring can find such anomalies in the feature model and improve the overall quality of the feature model by performing some modification on it (e.g. removing useless features).

Using model transformation is a suitable approach for altering models. Model to Model (M2M) transformation languages allow one to transform a source model into a target model. In performing a transformation, some elements within the source model may not be transformed to the target model or some new elements based on the target model form may be added to the destination model [13]. To perform a transformation, one needs the semantics of the source and destination models. The semantics of the models specify model elements and their relations in that model. Metamodels can be used to define the semantics of the models [14]. In this paper, we use *Ecore* metamodel notation to define the feature model metamodel.

Performing changes on the feature model is not the end of the refactoring process. After altering a model, one should ensure that the set of the valid configuration of the SPL is not changed by refactoring a feature model [12]. We used the Alloy language to check the correctness of the performed refactoring on the feature model. Alloy is an analysis language that uses SAT-Solvers to analyze the models defined using its syntax [15]. It enables the user to define the models in first-order logic. Using Alloy allows the algorithms to be expressed with fewer lines of code as compared to other languages [15]. In this paper, we defined several predicates, rules, and a way to model the feature model in the Alloy language. Each feature model is transformed to an Alloy model, and then an assertion for assessing the equality of the feature model before and after performing refactoring is checked on the model. If Alloy Analyzer finds a contradiction in the models, modifications done on the feature model are not safe; otherwise, the modifications are deemed safe.

Automated tool support for refactoring is a key issue for adopting approaches such as utilizing feature model and integrating them into the software development process of companies. In this work, we define some of the important refactoring rules on the feature model and provide tools that enable users to add new rules using the ATL M2M language. Our approach not only proposes refactorings but makes sure that the refactorings are also performed correctly. The notion of the Mandatory From Lowest Common Ancestor (MFLCA) and Optional From Lowest Common Ancestor (OFLCA) in this paper helps in finding some inconsistencies and problems such as local dead features and false feature types in the feature model.

By refactoring a feature model, the related artifacts may be influenced. For example, by removing a feature, the artifacts such as code and design which are related to that feature should be revised. Feature mapping is one of the approaches to keep the relation between feature model and other artifacts of the SPL. By refactoring a feature model, all affected features and their related artifacts should be investigated. Seidl *et al.* [16] proposed a framework to co-evolve models and feature mapping in the SPL. One can use their framework to maintain consistency of the model, feature mapping, and other related artifacts after performing refactoring.

In the remainder of this paper, we first describe the background and history relating to the feature model, model transformation, and Alloy in Section 2. The overall methodology of safe refactoring on the feature model is presented in Section 3. Section 4 describes the idea of automating feature model refactoring using M2M tools. In that section, we first define the feature model metamodel in Section 4.1, and then we introduce the approaches used to specify the feature model refactoring using the M2M transformation languages in Section 4.2. The Alloy model needed to analyze the feature model is presented in Section 5. The aim of Section 6 is to evaluate the proposed framework using different approaches. The practical applicability of the framework is shown in Section 6.1 using an example feature model. Section 6.2 takes a look at the effect of refactoring on the structural, cognitive, and compound complexity of the feature model. The performance of the proposed approach is assessed in Section 6.3. Section 6.4 surveys the computational complexity of one of the patterns of the proposed framework. In Section 7 we survey the related works. The paper concludes in Section 8 with a summary and a discussion of possible future work.

2. Background

2.1. Feature model

Feature model was developed in 1990 and first used in Feature-Oriented Domain Analysis (FODA) development approach [3]. The