ArchFeature: A Modeling Environment Integrating Features into Product Line Architecture

Gharib Gharibi and Yongjie Zheng

School of Computing and Engineering, University of Missouri-Kansas City, Kansas City, USA
ggk89@mail.umkc.edu, yzheng@umkc.edu

Abstract. An important task in product line architecture (PLA) modeling is developing the involved variation points and maintaining their conformance to product line features. However, existing modeling tools and approaches still require manual management of variation points and manual maintenance of feature-PLA relations, which is expensive and error prone. In this paper, we introduce a new PLA modeling environment named ArchFeature. It can automatically manage variation points in the PLA model, create and maintain feature-PLA relations, and derive new architectural instances. The key idea of ArchFeature is to develop the product line features and PLA side-by-side in the same environment, and integrate their specifications in a single model. The goal is to reduce the modeling effort and increase the quality of the PLA models.

Keywords: variability modeling · software product line architecture · extensible architecture description languages

1 Introduction

A key success factor of a modeling tool for product line architectures (PLAs) [1] lies in its ability to easily express and manage the variability [2] of a product line [3]. The variability is modeled in the PLA as variation points governed by guard conditions that identify related product line features [4]. Each feature is an end-user visible characteristic distinguishing the products of a product line.

While many approaches and tools for modeling the PLA exist [5-7], two main challenges remain. First, variation points need to be manually managed in the PLA. Specifically, developers have to manually specify the guard condition of each variation point and enter the names of the related features. This can cause significant overhead in the PLA development and prevent developers from focusing on application-specific design. Often, a variation point is related to multiple features and has child elements that are related to other features. It is tedious, time consuming, and error prone to manually manage such variation points. Second, it is difficult to manually maintain the conformance between a product line feature and its design (i.e. variation points) in the PLA. Product line features and PLA are commonly developed and evolve as two independent models using different tools. As the product line evolves, the two models can drift away

and become inconsistent. For instance, a feature can translate to multiple scattered variation points in the PLA. If this feature is removed, the developer needs to manually inspect the entire PLA model to identify the related variation points and remove them. This process can introduce inconsistencies and create a conceptual discrepancy between the product line features and architecture in terms of the defined variability.

In this paper, we present a novel PLA modeling approach and a tool called ArchFeature. The goal is to reduce the modeling effort and increase the quality of the PLA models. ArchFeature supports side-by-side development and evolution of product line features and PLA in a modeling environment. It provides a number of automated functions, including (1) managing variation points in the PLA, (2) creating, maintaining, and visualizing the feature-PLA relations, and (3) deriving new architectural instances. The essence of ArchFeature lies in its modeling approach, which integrates the specifications of features, PLA, and their relations in a single model. It is based on an extension of an existing architecture description language (ADL) called xADL [8]. Overall, ArchFeature has the following benefits:

- *Reducing the time and effort required to develop and maintain the PLA models.* ArchFeature completely encapsulates the management of variation points in the PLA from the developer. Given an architectural change made for a product line feature, ArchFeature can automatically update the related variation points and their guard conditions. This also increases the quality of the PLA model in terms of variability definition.
- *Bridging the abstraction gap between the product line features and PLA.* ArchFeature integrates the development of features and PLA in the same environment and defines them in a single model. In addition, it can automatically maintain feature-architecture conformance in the evolution of the product line.

2 The Modeling Approach

The underlying modeling approach of ArchFeature is based on an extension that we made to an existing XML-based ADL called xADL. Specifically, we developed new language constructs (e.g., *feature*) to integrate the specifications of features and feature-PLA relations into the PLA model.

Fig. 1 illustrates an overview of ArchFeature interface (Fig. 1. B) and the PLA model (Fig. 1. A). The PLA model includes the specifications of a product line feature (i.e. *Game*) and one of its related variation points (i.e. Component *PlayGame*). Our definition of features is based on feature models [9]. However, we developed new elements to support the automated functions of ArchFeature. For example, the *featureColor* element specifies a color to be used in the visualization function, which highlights all the PLA elements related to a feature. The *archElements* element (Lines 5-8) includes links to all the elements related to the feature (e.g., Component *PlayGame*). *archElements* is important to automate the operations of creating and maintaining feature-PLA relations. Another important element is the *type* element, which depicts the way a feature may vary. It can be *Optional*, *Alternative*, or *OptionalAlternative*. In case of *Alternative* and

OptionalAlternative features, the *variants* element is used to identify the feature's variants. Note that ArchFeature does not address the feature relations (e.g., mutual dependency). This is partially because the focus of our work is modeling the PLA. In addition, many existing approaches have specifically addressed the feature relations [10].

To the right of Figure 1. A is an example of a component's specification (i.e. Component *PlayGame*). It is based on the original version of xADL. A variation point (Lines 2-11) is embedded to indicate that this is an optional component. It includes a *guard* condition (Lines 3-10) defined as a Boolean expression with an equality operator. The *symbol* element represents the name of the related product line feature (i.e. *Game*). The *value* element indicates when the component should be included. It can be *true*, *false*, or the name of a variant in case of alternative features. If a component is related to multiple features, the guard conditions of all the features are connected via Boolean operators. Guard conditions of all variation points in the PLA are automatically created and managed in ArchFeature as further explained in the following section.

3 ArchFeature

ArchFeature is a modeling environment that is based on the integrated modeling approach introduced in Section 2. It aims to address the challenges of manual development of variation points and maintenance of feature-architecture conformance in modeling PLAs. Fig. 1. B illustrates the interface of ArchFeature. It consists of a feature list and a PLA graphical editor. The feature list enables creating and managing product line features. The PLA editor supports creating, visualizing, and managing PLAs. The figure shows a PLA of a chat application and its features that were developed in ArchFeature. The PLA and features are both defined in the same model (shown in Fig. 1. A). Based on this model, ArchFeature enables the following operations.

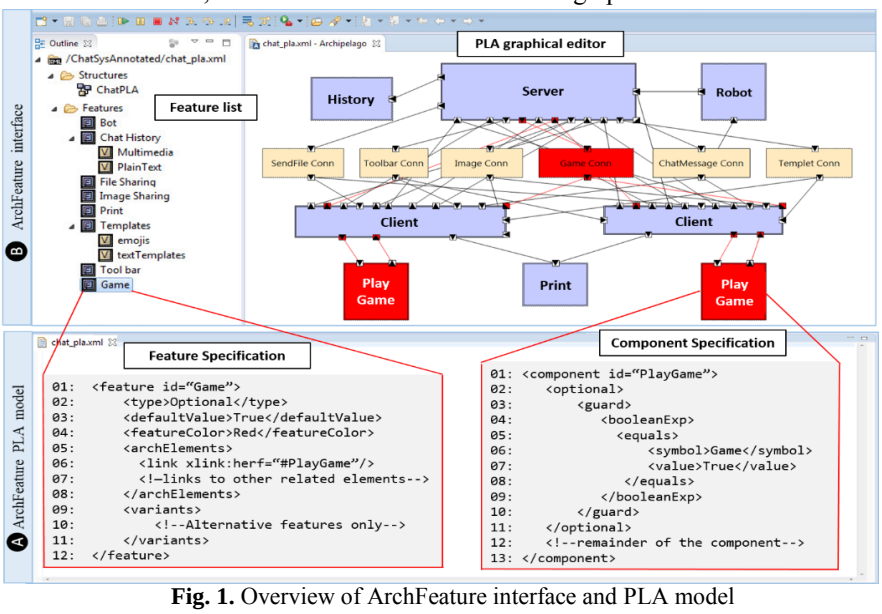

Fig. 1. Overview of ArchFeature interface and PLA model

Automatic Variability Management. Unlike other tools that require manual development of variation points, ArchFeature fully automates this process and encapsulates it from the user. In particular, when a feature is selected, a guard condition referring to that feature is set in the background. When a new architectural element is created, a variation point with the guard condition will be automatically integrated in the element's specification. To support variation points related to multiple features, Boolean operators (e.g., OR) are used to connect multiple guard conditions corresponding to different features. Moreover, ArchFeature can automatically manage variation points of hierarchical elements. It ensures that the guard condition of a parent element covers all the guard conditions of its child elements. Thus, the parent element will always be included when any of its child elements is included.

Mapping Features to PLA. ArchFeature automatically manages the feature-PLA relations in the PLA model. This includes:

- *Creating the feature-PLA relations.* ArchFeature supports both automatic and manual ways to relate a feature to PLA elements. The automatic way is useful to map a feature to new architectural elements. The developer can select a feature from the feature list and then enter the editor to modify the PLA. All the new developed architectural elements will be automatically mapped to the selected feature. The manual way is useful for mapping a feature to existing PLA elements. The developer can select a feature and then right click an existing PLA element and select "*Add to Current Feature*". In both ways, the PLA model will be automatically updated.
- *Visualizing feature-PLA relations.* This function helps developers to understand how and where a certain feature is designed in the PLA. Selecting a feature will highlight, with a feature-specific color, all of its related elements in the PLA. The color can be changed by right clicking the feature and selecting "*Change Color*". In Fig. 1. B, the *Game* feature is selected. Correspondingly, all the related elements (e.g., Component *PlayGame*) in the PLA are highlighted in red. Without this function, developers need to manually inspect the entire PLA to identify feature related elements.
- *Maintaining feature-PLA conformance.* ArchFeature can automatically update features and architecture to maintain their conformance when evolutionary changes occur. For example, if a feature is removed, all of its related elements in the PLA will be automatically removed. Thus, the product line features and PLA are kept up-to-date and automatically synchronized.

Deriving New Architectural Instances. ArchFeature includes a tool, called *Selector*, which can automatically derive new architectural instances from the PLA. Based on the integrated modeling approach, it can automatically list all of the product line features and their values. Then, the developer can configure the PLA by simply changing the features' values and run the application. The result is a new architectural model describing a single product. The new correctly-derived models further prove the applicability and efficiency of ArchFeature in automatically managing and maintaining the PLA and its defined variability.

4 Implementation

We implemented and integrated ArchFeature in ArchStudio [11], an open-source Eclipse toolset used for developing architecture-based software systems. ArchStudio includes a number of tools such as *Archipelago*, *ArchEdit*, and *ArchLight* that can be used to visualize, model, and analyze software architectures. We reused and extended some of these tools in our implementation. First, we extended xADL and developed new language elements to define and integrate features and their relations in the PLA. Second, we extended Archipelago and focused on implementing the functions presented in Section 3. Third, we reused a prototype of a product derivation tool that existed in ArchStudio to build our Selector tool. The original prototype required manual preparation and configuration of the PLA's variation points to derive new architectural instances. We re-built this tool and fully automated the derivation process.

5 User Experience

To assess the applicability and effectiveness of ArchFeature, we released it to academic and industrial users. The academic users included the students of two classes that we taught at our school. They were given assignments and projects to develop the PLA of an open-source system of their choice. Some examples are Apache Giraph, Scaffold Hunter, and Sweet Home 3D. Students reported that using ArchFeature was easy and straightforward. With a basic knowledge of Eclipse, they were able to model a new PLA without any prior experience of ArchStudio. The automated functions of ArchFeature made it straightforward to model the PLA without worrying about modeling the variation points or maintaining their conformance with the product line features.

The industrial users included a software architect and a software engineer from Cerner Corporation [12]. They used ArchFeature to develop a full-featured architecture for Apache Solr 4.2.0 [13]. Solr is a Java-based open-source system that has approximately 181K SLOC and has been through more than ten years of development. A number of features have been added to it over time. However, an explicit architecture that distinguishes the elements related to different features did not exist before this case study. The architecture model of Solr developed in this case study includes one hundred and eighty three components, twenty eight features, and two hundred and twenty four feature-PLA relations in total. There were twenty seven core components representing the kernel functions of Solr, and the rest are variation points corresponding to twenty eight features. Out of the twenty eight features, fourteen were optional features, eleven were alternative features, and three were optional-alternative features. The total number of feature variants contained in the alternative features was one hundred and forty three. A main issue that was reported is the lack of additional feature types. In particular, an OR feature is needed to allow more than one variant to be selected. This can be addressed by modifying xADL's schema as discussed in Section 2, and it is made our future work. The PLA developed in this case study is available on the ArchFeature website listed in Section 7.

6 Related Work

Ménage [14] is one of the early tools focusing on PLA modeling. The variation point definition shown in Fig. 1 is based on the design of Ménage. A primary limitation of Ménage is that all the variation points and guard conditions in the PLA have to be manually created. Additionally, it offers limited support for relating features to the PLA and cannot visually distinguish elements related to different features.

Feature template [15] advocates superimposition of all variants in a single model called model template that refers to features through annotations. Similar to ArchFeature, Feature template defines Boolean formulas over the feature names and uses them during the derivation process. Unlike ArchFeature, Feature template does not support automatic creation of variation points or visualization of feature-PLA relations.

FeatureMapper [16] is a tool that supports mapping features from a feature model to solution artifacts expressed in EMF/Ecore-based languages (e.g. UML2). Both FeatureMapper and ArchFeature can automatically relate a feature to elements in the solution space. Both support visualization of the feature-PLA relations, and both can derive a model instance based on features configuration. A main difference between them is how variability is defined in the target model. FeatureMapper saves variability information and the feature-PLA relations in a separate mapping file, which can create a conceptual gap between the artifacts as they evolve over time.

Gears [17] is a product line framework emphasizing automatic derivation of product-specific artifacts. It includes a product configurator, a feature model, and a set of reusable artifacts containing defined variation points. The product configurator automatically customizes each reusable artifact based on a feature portfolio, and derives artifacts from each stage of the development life-cycle that belong to a product instance. However, Gears does not support integrated modeling of features and PLA.

EASEL [7] is another tool that supports PLA modeling based on an extension of xADL. EASEL separates variable architecture elements of a PLA into a number of change sets. Each change set only contains the architecture elements that implement a specific feature. A main issue of EASEL is that it makes a PLA model less understandable as the definition of an architecture element is spread into multiple change sets.

There are tools that focus on mapping the product line features directly to source code, such as FeatureIDE [10] and CIDE [18]. In contrast, ArchFeature focuses on PLA modeling and feature-architecture mapping. The PLA addresses the product line's complexity and diversity at a higher abstraction level than source code. Therefore, it should play a central role in the development of software product lines.

7 Tool Availability

A video demo of ArchFeature is available at: https://youtu.be/FEFGmaDruWo. More information on ArchFeature are available at: http://info.umkc.edu/sail/archfeature/.

Acknowledgments. We thank Varun Narisetty for his help in implementing ArchFeature. We also thank Adam Carter and Jeffrey Lanning for their help with the case study.

References

1. Bosch, J., *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. 2000, Reading, Massachusetts: ACM Press, Addison-Wesley Professional.
2. Sinnema, M., et al. COVAMOF: A Framework for Modeling Variability in Software Product Families. in Third International Software Product Lines Conference (SPLC 2004). 2004. Boston, MA, USA: Springer Berlin / Heidelberg.
3. Pohl, K., G. Böckle, and F.J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. 1 ed. 2005, New York, New York: Springer. 468.
4. Czarnecki, K. and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. 2000, Reading, Massachusetts: Addison-Wesley Professional.
5. Groher, I. and R. Weinreich. Integrating Variability Management and Software Architecture. in Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on. 2012.
6. Ommering, R.v., et al., *The Koala Component Model for Consumer Electronics Software.* IEEE Computer, 2000. **33**(3): p. 78-85.
7. Hendrickson, S.A. and A. van der Hoek. Modeling Product Line Architectures through Change Sets and Relationships. in 29th International Conference on Software Engineering (ICSE 2007). 2007. Minneapolis, MN.
8. Dashofy, E., A. van der Hoek, and R. Taylor, *A Comprehensive Approach for the Development of Modular Software Architecture Description Languages.* In ACM Transactions on Software Engineering and Methodology, to appear. , 2005.
9. Kang, K.C., et al., *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. 1990, Software Engineering Institute.
10. Thüm, T., et al., *FeatureIDE: An extensible framework for feature-oriented software development.* Science of Computer Programming, 2014. **79**(0): p. 70-85.
11. University of California Irvine, I.f.S.R. *ArchStudio*. Available from: http://www.isr.uci.edu/projects/archstudio/.
12. Cerner Corporation. Available from: http://www.cerner.com/.
13. Apache Software Foundation. *Apache Solr*. Available from: http://lucene.apache.org/solr/.
14. Garg, A., et al. An Environment for Managing Evolving Product Line Architectures. in IEEE International Conference on Software Maintenance (ICSM 2003). 2003. Amsterdam, The Netherlands.
15. Czarnecki, K. and M. Antkiewicz, Mapping features to models: a template approach based on superimposed variants, in Proceedings of the 4th international conference on Generative Programming and Component Engineering. 2005, Springer-Verlag: Tallinn, Estonia. p. 422-437.
16. Heidenreich, F., J. Kopcsek, and C. Wende, FeatureMapper: mapping features to models, in Companion of the 30th international conference on Software engineering. 2008, ACM: Leipzig, Germany. p. 943-944.
17. Krueger, C.W., The BigLever Software Gears Unified Software Product Line Engineering Framework, in Proceedings of the 2008 12th International Software Product Line Conference. 2008, IEEE Computer Society. p. 353.
18. Kästner, C., S. Apel, and M. Kuhlemann, Granularity in software product lines, in Proceedings of the 30th international conference on Software engineering. 2008, ACM: Leipzig, Germany. p. 311-320.