An Aspect-Oriented Framework for Software Product Line Engineering

Lei Tan, Yuqing Lin and Huilin Ye
School of Electrical Engineering and Computer Science
The University of Newcastle
Callaghan, NSW, Australia
Email: lei.tan@uon.edu.au,
{*yuqing.lin, huilin.ye*}*@newcastle.edu.au*

***Abstract*—Software Product Line Engineering (SPLE) is a relatively new software development paradigm to improve systematic software reuse. There are two key assets underpin the software product line (SPL) development: a feature model and a reference architecture. To deal with complex crosscutting behaviours in SPLs and also manage the impact of non-functional requirements (NFRs), we propose an aspect-oriented framework in this paper. The proposed framework is able to improve the modeling of interrelationships between design factors and representation of the variabilities in product family. We introduce a small case study to illustrate our approach at the end.**

***Keywords*-software quality; software product line; domain engineering; aspect-oriented; crosscutting concern**

I. Introduction

Software Product Line Engineering (SPLE) [1] is a newly emerging software engineering approach that focuses on providing systematic software reuse. Software reuse is to use and adapt existing software assets or knowledge from similar systems to achieve similar functionalities in the similar operating environments [4]. SPLE focus on developing a software product family and its members within a particular domain. A product family is a collection of software systems that share a great commonality in a particular application area.

SPLE consist of two processes, domain engineering and application engineering. The tasks of establishing a software product line (SPL) belong to domain engineering, and the processes that developing a particular member product from an SPL are included in application engineering. Tasks in domain engineering are designed as the basis for product derivation later on. When developing a member product of a software product family, the reusable artefacts from domain engineering will be refined, modified and configured. SPLE save the effort of developing every single software system from scratch. SPLE also provide systematic requirement mapping and traceability to ensure that requirements will be realised by final products. Once the reusable artefacts are defined and successfully implemented, the individual member products can be produced through a very organized customization.

The remainder of the paper is organized as follows. In section 2, we will introduce background about SPLE and also some open problems. In section 3, we propose a comprehensive framework that dealing with these problems from the requirement level down to the architectural level by using aspect-oriented techniques. In sections 4, we present a case study to demonstrate how we develop our framework and achieve aspect identifications and mappings. Section 5 concludes the paper and discusses the future work.

II. Background

In domain engineering, there are two key reusable artefacts: a feature model and a reference architecture. These two artefacts are significant in an SPL as they are implemented as the basis for an SPL establishment and products derivation.

A. Feature Model

A feature model [5] represents all the possible member products of an SPL in terms of features and the relationships among features. In a feature model, features are prominent and distinctive system requirements or characteristics in an SPL [7]. A feature model is used to configure member products, also used to refine software architectures. To produce member products in the application engineering, a set of desired features are selected from a feature model based on customers' requirements and feature relationships.

B. Reference Architecture

Another important artefact derived from a feature model is called a reference architecture. A reference architecture [3] is a software architecture that provides the common structures, components and their relationships to the existing systems in a particular domain of an SPL. In an SPL, a reference architecture describes the structures and respective elements and relations for the whole product family, i.e. for multiple products in the product line. So it stresses the commonality of the architecture and also describes the planned variability.

C. Open Problems

In current SPLE development and modeling, there are some open problems have been identified in both feature model and reference architecture.

- **Some complex feature relationships are hard to manage**: One-to-one relationships between features are modeled effectively by feature modeling. The more complex feature relationships, such as one-to-many or group features, are hard to handle and managed properly. As a result, feature selections in configuration become error-prone and the quality of configured product is barely preserved.
- **Quality attributes are not properly modeled in feature models**: A feature model is produced from requirement engineering based on requirements documents on products from a software family. However, quality attributes are not well modeled and the impact between quality attributes and features are not specified in current feature modeling.
- **Transformation from a feature model to a reference architecture is not straightforward**: The main purpose of a feature model is to guide the member software derivation in application engineering. But the mechanisms of systematic mapping from features to components are not clearly defined, and the process is not well organized. This information should be represented explicitly to enhance SPLE to improve the efficiency of product derivation.
- **Variability representation is not adequate**: Variability representation mechanisms are needed in both feature models and reference architectures, because variability management is critical to product derivation. All the possible features for products and the possible variations to the reference architecture should be described in an easy understandable way.

III. Proposed Framework

We propose a framework by adapting the concept of Aspect-Oriented Software Development [2]. Our framework takes quality attributes as a driver, to model the relationships between quality attributes and software elements by emphasising the impact of quality attributes and NFRs on software features/components to preserve the quality of member system derivation. In a software product family, one quality attribute may crosscut multiple features/components, and this crosscutting behaviour may result in different levels of the quality attribute for various member products. In this case, an advanced modeling structure is needed to isolate quality attributes, and address the impact of quality attributes to corresponding features/components of member systems.

Aspect-Oriented Software Development (AOSD) provides a modularized structure to represent crosscutting concerns explicitly. To adapt the idea of AOSD, we propose to decompose quality attributes and NFRs of an SPL into aspectual concerns, which have crosscutting relationships with software features and components. These quality-related concerns affect the way how software factors interact with each other to achieve the corresponding quality requirements for software members. These concerns should be considered as "Aspects" to investigate their relationships with other software elements. Moreover, variability in an SPL could be managed better by these aspectual concerns as the crosscutting relationships with different quality levels reflect the existing of member systems in an SPL.

Our framework is proposed on the requirement and the architectural level of domain engineering, rather than the code level. Any crosscutting concern decomposed from quality attributes could be considered and modeled as "Aspect" to represent its relationships with functional features and components, and to address its impact on both levels of SPL development. The framework is composed by two layers: an AO feature modeling and an AO reference architecture development. The main expectation of the AO feature modeling is to convert conventional feature model into several components. We also expect to define impact of aspectual concerns on both concrete and aspectual features. The goal of the AO reference architecture is to specify the mappings from the AO feature model to the reference architecture. The options in the reference architecture for satisfying the NFRs and quality factors at different levels will also be presented.

A. Aspect-Oriented Feature Model

To develop the AO feature model, we extend conventional use case and decompose it into small business flows. We are interested in systems' internal processes and responsibilities in order to realize the behaviours within the system. If a particular system behaviour existing in multi-business processes and interacts with many other user visible functionalities of the system, this would be a candidate of aspectual feature. To fit use case models in SPLs, variabilities existing in use cases need to be addressed.

To better understand the AO feature model, we propose to include crosscutting concerns in feature models and divide features into several categories as follows:

- **Concrete Feature**: Concrete feature represents basic functionalities of a product family. These features represent the fundamental services the systems provide. In the context of aspect-oriented framework, these features are crosscut by concerns.
- **Aspectual Feature**: Aspectual features also represent the functionality of a product line, and these features crosscut other features.
- **Aspectual Concerns**: Aspectual concerns are the key requirements and system considerations crosscutting the concrete features and aspectual features. The concerns describe the impact of NFRs and quality requirements on the system composition and the way features

interact to each other.

According to these feature categories, the tasks of the AO feature modeling include developing related models to treat crosscutting features (functional) and concerns (non-functional) in a systematic way. Modeling concrete features is relatively easy as concrete features correspond to the user visible functionalities of the system. Current use case modeling and scenarios based approaches are able to address the functional requirements of the software systems. The main task is to develop a model which is able to clearly represent the concrete features and relationships between these concrete features.

Aspectual concerns could be identified to describe the impact of NFRs on the functional features. An aspectual concern corresponds to system requirements, it describes the impact of system requirement on concrete features and NFRs. To address these impacts, the contribution of each concrete feature to satisfy the NFRs will be examined. Since all combined concrete feature options are identified, we are able to specify the related quality levels that can be achieved by these options. Based on the identified relationships between concrete features and NFRs, aspectual concern candidates could be identified.

Then aspectual features need to be identified, and to map aspectual concerns on concrete and aspectual features. An aspectual feature is a feature that crosscutting other concrete features. They are additional responsibilities that do not affect the main business flow. There are several situations in the aspectual concerns mapping. An aspectual concern could crosscut several concrete features (aspect-to-feature), this situation suggests several implementing options of concrete features to satisfy different requirements. Another situation is that several aspectual concerns crosscut a same concrete feature (aspects-to-feature), conflicts could exist in this situation to affect the achievements of some functional and non-functional requirements.

By the end, we will convert a feature model into three components, i.e. concrete features, aspectual features and aspectual concerns. The impact of aspectual concerns on both types of features will be identified as well. Moreover, the complex feature relationships will be decomposed into relationships among these three components and NFRs.

B. Aspect-Oriented Reference Architecture

Aspect-oriented reference architecture framework is to specify how features and aspectual concerns from the AO feature modeling are mapped onto the reference architecture. The key point is to identify crosscutting concerns caused at the architectural level and how the reference architecture is affected and composed accordingly.

Concrete features identified from the AO feature model need to be mapped onto the components at the architectural level, which means that the functionalities represented by features will be implemented by related components. Conventional SPLE approaches can be used to transform concrete features into components or subsystems to meet functional requirements. So the concrete features will be transformed as components and connectors to link these components.

To map the aspectual concerns to the architectural level, we will examine how the system reacts to user's input to satisfy both functional and non-functional requirements at different levels. We look at the impact of aspectual concerns on the components and detail the collaborations of components in terms of pointcuts and possible components to be plugged in. The pointcut represents where the control flow could be interrupted and all constraints on the crosscutting. Having the options that various of component been plugged in at the pointcut, the reference architecture describes all the possible architectures can be derived. As aspectual features are the additional tasks in the main workflow, thus, this kind of features becomes components to crosscut other components in most of the cases, e.g. they are the ones to be plugged in at the pointcut.

To complete the transformation, we also need to develop a set of the architectural scenarios [8] to describe systems. Since the requirements are from multiple products, it is common that we have multiple alternative flows for a use case corresponding to the different requirements and quality expectations. These alternative flows suggest alternative internal workflows of the system and will be mapped to parallel scenarios.

As the nature of SPLs, variability of components need to be addressed. Some of the variabilities can be located within a single component, and some of the variabilities suggest extra components and pointcuts in the system. Here we need to examine how the aspectual concerns are addressed in the architecture. These aspectual concerns suggest how the components interact and components crosscut each other for satisfying functional and non-functional requirements for various member systems.

IV. Case Study

In this section, we will present a small case study to illustrate our framework. The case study used is a product line of crisis management systems (CRS) [6]. To demonstrate our approach better, we modified the case by adding some features and expanding use case mapping diagrams.

First, we will identify all the features of the system and their responsibilities. By relating features to use case mappings, some of these features only exist in a single business flow, other features exist in multiple business flows. These features should be considered as candidates of aspectual features.

Then we move to the use case mapping. As mentioned, we decompose use case into small business flows and observe systems' internal processes and responsibilities. For example, by observing these flows, we identify that all

the use cases contain a same functional element called "rerankingCrisisPriority", which is an internal behaviour of the system. As this functional feature exists in several use case processes with interactions with other system functions, the corresponding feature could be looked as a potential candidate of aspectual feature. Note that only candidates that contribute to the achievement of aspectual concerns should be considered as aspectual features.

To identify aspectual concerns, we need to find out the related quality attributes and how they are decomposed into concerns. In this case, we know that several quality attributes, such as "Performance", "Security" and "Mobility", are affected by multiple features. These features are crosscut by quality attributes in terms of various aspectual concerns. Table I lists the crosscutting relationships between aspectual concerns and related quality attributes by "**X**".

Table I
ASPECTUAL CONCERNS AND RELATED QUALITY ATTRIBUTES

Aspectual Concerns	Performance	Security	Mobility
Constant cache management	**X**	**X**	
Instant message exchange	**X**		**X**
Fast logging verification	**X**	**X**	

Down to the architectural level, identified aspectual features and concerns will be mapped to components. Variabilities at requirement level are also inherited by different architectural options to make sure that NFRs are achieved at the architectural level. Fig. 1 shows a set of components and their communications. Identified aspectual feature "rerankingCrisisPriority" is mapped as a component called "reranking" on the architecture and variable design options are achieved by aspect crosscutting in different ways. For example, "reranking" crosscuts components "Report" and "Task management" through port "data" to transmit data for systems' report generation and live task management. It also crosscuts components such as "Communication", "Execute mission" and "Task management" through another port to provide secure/instant communication and information update among all available resources.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a new comprehensive framework by using the concept of Aspect-Oriented software development to simplify complex features relationships and the impact of NFRs in SPLE. Our approach starts from feature modeling and provides mappings to SPLA development. The advantage of the framework is that it enhances the modularity of system, and it also preserves the traceability between features/concern models and the requirements. The transformation between the requirements and the reference architecture is more efficient. The SPLA development will be more accurate because NFRs and qualities are treated as part of the architectural elements.

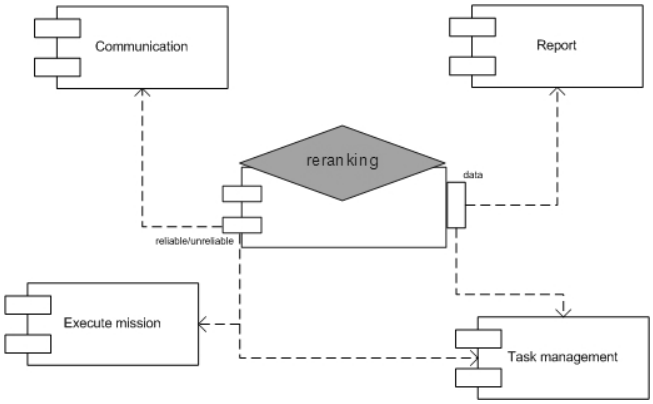

Figure 1. "reranking" crosscuts other components

For the future work, we need further investigate how to enhance the systematic mappings from the requirements to the architecture. Appropriate mechanisms are eager. The impact of NFRs and quality attributes should also be emphasized. We would like to conduct a real-world case study to evaluate our approach.

REFERENCES

[1] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*, Addison-Wesley, 2000.

[2] J. Brichau, R. Chitchyan, A. Rashid and T. D'Hondt, *Aspect-Oriented Software Development: An Introduction*, Wiley Encyclopedia of Computer Science and Engineering, 2008.

[3] W. Eixelsberger, M. Ogris, H. Gall and B. Bellay, *Software Architecture Recovery of a Program Family*, in the proceedings of the 20th International Conference on Software Engineering, pp.508-511, 1998.

[4] W. Frakes and C. Terry, *Software reuse: metrics and models*, ACM Computing Surveys, 28(2), pp.415-435, 1996.

[5] K. Kang, S. Cohen, J. Hess, W. Nowak and S. Peterson, *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report, Pittsburgh,PA, Software Engineering Institute, Carnegie Mellon University, 1990.

[6] J. Kienzle, N. Guelfi and S. Mustafiz, *Crisis Management Systems: A Case Study for Aspect-Oriented Modeling*, Transactions on Aspect-Oriented Software Development, 7, pp.1-22, 2010.

[7] K. Lee, K. Kang and J. Lee, *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering*, in the proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools, pp.62-77, 2002.

[8] E. Rommes and P. America, *A Scenario-Based Method for Software Product Line Architecting*, Software Product Lines-Research Issues in Engineering and Management, Berlin: Springer, pp.3-52, 2006.