Guidelines for Architecture Design of Software Product Line

Jeong Ah Kim[1], DongGi Kim[1(✉)], and JinSeok Yang[2]

[1] Department of Computer Education, Catholic Kwandong University, BeomIl Ro 579 24, Gangneung, Kangwon, Korea
clara@cku.ac.kr, remaindk0@gmail.com
[2] 18F, 145, Gasan digital 1-ro, Geumcheon-gu, Seoul, Korea

Abstract. Product Line Architecture design is a key activity for developing successful Software Product Line projects. But it is difficult and complex task since architecture of software product line should be considered with variability. In this research, we addressed detail guidelines for identifying the component of architecture from the feature model and defining the variability of component in concerns of feature.

Keywords: Software development · Product-line architecture design · Logical component modeling · Guidelines

1 Introduction

The objectives of traditional software development are to define, design, and implement the requirements for a single software application to be developed, and to produce defect-free software. Although reuse framework was considered to efficiently conduct this process, only common technical modules or general-purpose algorithm modules, which are not related to the application field, were reusable. In other words, reuse for improving productivity in the developing process for a single application was considered. This type of reuse cannot secure reusable organizational assets in similar application fields for the future. The development that considers only a single software application will not be able to secure general-purpose assets that can be used in a specialized application field, because the diversity of customer, environment, and infrastructure technology cannot be considered in the development phase of a single software application. The construction of a product line is a process that creates the foundation on which one or more applications with diversity can be developed rather than a single application. Product-line engineering aims to create a customized product based on a product line in which the concept of systematic asset management at the organizational level is introduced [1]. From the developer's perspective, the existing development approach appears reproduced by reusing libraries or developing frameworks [2]. Furthermore, managers and developers many not know how to introduce this concept since they thing this is new development methodology. There are few existing researches that are organized into an independent and complete methodology in which the product-line engineering process can be implemented. Most research

S. Misra et al. (Eds.): ICCSA 2019, LNCS 11623, pp. 415–422, 2019.
https://doi.org/10.1007/978-3-030-24308-1_34

works focus on the existing single-system development methodology and the techniques that can be applied to other areas [3–8].

In this study, guidelines for modeling activities of a product-line architecture model included in the product-line architectural design stage are provided. A product-line architecture design model, a design guideline that considers the variabilities and a modeling method that uses a real unified modeling language (UML)-based modeling tool are defined herein to develop a product-line architecture model. Furthermore, a case model to which these models and methods are applied is presented.

2 Proposed Product-Line Architecture Design Model

A design model for the product-line architecture proposed is defined in this section. The model defines three different views, each of which contains information according to the design perspective and share a relationship with one another.

- Conceptual View: This view provides the logical components constituting the product-line architecture, the information of the structure, and the interaction in which the relationship between the logical components are defined.
- Execution View: This view provides the physical components (executable files or libraries) where the identified logical components and their dependency information are executed.
- Deployment View: This view provides information on the relationship between physical computing devices and nodes to which physical components can be deployed. It also provides information on the physical component instances that are deployed and run on the nodes.

2.1 Conceptual View

The model elements of a conceptual view include at least one logical component and an interface of that logical component. In addition, the conceptual view consists of a structural relationship between the logical components and an interaction relationship diagram between the interfaces of the logical components.

This view represents the logical components, which are logical elements necessary to meet the product line requirements including features, and the interfaces that are provided by those components. “Logical” in this context means that the implementation environment for the product line or technology is not considered. A logical component (hereinafter referred to as a component) indicates a software module capable of performing independent functions. It classifies complex software properly so that the relevant software is manageable, and hides the complexity and diversity of methods for the implemented interface. Through this feature, outdated components can be replaced by new components that have different implementation methods.

A component control interface is a set of operations that does not have an implementation body due to the agreement by the developers to allow it to interact with other components or applications. The operations defined in the component control interface are implemented by the component. Therefore, a single component can be a unit that implements multiple interfaces.

The component data interface defines the dataset needed for establishing interaction between components. The data interface defines the dataset that will be used by the logical components and does not define an operation that undertakes the data processing.

A component group logically binds components that are functionally related. However, this is not an essential element in the model. With component groups, an abstract hierarchy between the components can be represented. By using the component group, several components can be logically grouped together to simplify the complexity that can arise from the relationship between the individual components.

The conceptual view provides a component structure diagram that can schematize the relationship between the logical components, a component interaction diagram, and a component interface diagram.

On the one hand, the component structure diagram represents the dependency among the logical components, i.e., the structural association between the components.

On the other hand, the component interaction diagram sequentially represents the interaction between the logical components to meet specific requirements, using the interface implemented by those components.

Moreover, the component interface diagram represents the relationship between the interface implemented by the logical component and the logical component that uses the interface.

2.2 Execution View

An execution view has at least one physical component as a model element. It has a dependency diagram between physical components. The physical components identified in the execution view must be correlated with at least one logical component. More specifically, the physical component is divided into an executable component and a library component. The executable component indicates a component that has the same starting point as the main() function and, thus, can be executed independently, while the library component indicates a component that is called by an executable component during the execution and can be deployed independently. The execution view provides a physical component structure diagram that can be used to schematize the relationship between the declared physical components.

2.3 Deployment View

A deployment view has at least one computing device node as a model element. It also has a phase information diagram between the nodes. The identified node in the deployment view must have at least one physical component instance.

3 Guidelines for Applying Product-Line Architecture Design Model

3.1 Guidelines for Logical Component Modeling

- GUIDELINE1: It is divided into two types of logical components: a logical component with high cohesion and a logical component with low coupling. In other

words, one logical component is identified using a unit that has logical cohesion, and it is then verified if the component has low coupling with other logical components. Such a component is identified as an independent unit. Hence, the requirements that have high mutual-functional relevance are grouped together, and the implementation of the function is assigned to one logical component.

- GUIDELINE2: The features having high mutual-functional relevance among the mandatory features are grouped together and the implementation of that function is assigned to one logical component.
- GUIDELINE3: The optional and alternative features are identified as independent logical components by separating them from the logical components that implement the mandatory features.
- GUIDELINE4: If there is a structural relationship between the features, and an optional feature in the child feature, then the logical component for the optional feature is identified independently. Furthermore, an association is established with the logical component that corresponds to the parent feature.
- GUIDELINE5: The logical components for the alternative features are independently identified when a structural relationship between the features and an alternative feature in the child feature is found. Moreover, the inheritance relationship is established with the logical component corresponding to the parent feature.
- GUIDELINE6: When a design is conducted using pattern styles, such as model-view-controller (MVC) and client-server (C/S), it is possible to identify for each element a corresponding logical component.
- GUIDELINE7: A two-level feature without any further sub-features is identified as a logical component. If all the features defined at third-level are mandatory features, then the features defined at two-level can be identified as logical components.
- GUIDELINE8: The logical components that correspond to the features defined at three-level are identified. The mandatory features defined in three-level can be grouped together and identified as one logical component, and the optional features can be grouped together and identified as another logical component.
- GUIDELINE9: The top-level feature of the functional feature is identified as a logical component, but the group feature is not. If there is a strong dependency between the top and bottom features among the functional features, then the bottom feature is not separated into an independent logical component. In this case, only the top functional feature is identified as a logical component.

3.2 Interface Structuring Guidelines

- GUIDELINE10: Among the operations that will be implemented by the logical component, those operations that will be provided to the outside of the product line, or to other components, are defined as control interfaces.
- GUIDELINE11: Among the operations provided by the logical component, those operations with different functions and characteristics are defined as independent interfaces.
- GUIDELINE12: Among data in the logical component, data that will be provided to other components is grouped into a dataset and is defined as a data interface.

- GUIDELINE13: One feature can be identified as a single control interface. However, if features among the mandatory features are grouped into one group, they can be identified as a single interface. In the case where there is an excess number of operations that need to be defined, it is desirable to divide the interface based on the feature. Although a reasonable number of operations included in the interface is academically defined from 7 to 9, it is recommended to define this number as being less than 15.
- GUIDELINE14: The interface corresponding to the variable feature must be identified independently from other interfaces. Inevitably, variable features are matched to the level of operation provided by the interface, but this is not recommended.

3.3 Physical Component Modeling Guidelines

- GUIDELINE15: The logical components with strong dependencies during the execution are grouped together and identified as one physical component, the independently executable logical components are identified as individual physical components, and are identified by considering the distribution units.
- GUIDELINE16: A physical component that executes mandatory features cannot be comprised of logical components that implement the optional feature alone.
- GUIDELINE17: In a real-time system, a task is modeled using executable components. In a general system, an application is modeled using an executable component.
- GUIDELINE18: All the components utilized by an executable component are modeled using library components.
- GUIDELINE19: The logical components that perform similar functions can be grouped together and identified as a single variable library component.
- GUIDELINE20: The name of a physical component can either be the same as the name of the logical component or controller, or can be added to the name.
- The dependency relationships among the identified physical components can be schematized into a structure diagram.

3.4 Node Modeling Guidelines

- GUIDELINE21: Product-line requirement-environment elements for hardware defined in the performance requirements specification are confirmed, and each hardware device is identified as a node.
- GUIDELINE22: An instance of the physical component defined by reviewing the quality requirements such as performance, stability, and reliability is assigned to a node.
- GUIDELINE23: At least one of the physical components to be deployed to the node must be a mandatory component.
- GUIDELINE24: If the cardinality value of the feature executed by the physical component is greater than 1, the physical component instances are assigned to different nodes according to the hardware specification and the number of identified nodes, to check whether the quality requirements such as performance can be satisfied.

4 Example of Application

4.1 An Example of Identifying Logical Components from Feature Models

Table 1 shows a list of identified logical components that have a level of variability similar to that of the first feature model and are functionally identifiable. Among the features with a level of variability similar to that of the first feature model, *TakeoffControl*, *LandingControl, WeaponControlOperation, FlightScenarioPlanning, Login*, and *SensorOperation* are the variable features. Among these features, as logical components whose boundaries can be matched with the corresponding feature, *LandingController, TakeoffController, WeaponOperator,* and *LoginManager* are identified as independent logical components. The variability and feature traceability of the logical component groups and the logical components are modeled using Rhapsody's model element tag attributes. The generation of the model for variability and feature traceability can be conducted simultaneously by adding a new tag with a VAR name and describing it in a feature expression for the features associated with that value.

Table 1. Identified logical component

Logical component group	Logical component	Feature	Variability
FlightDeviceControlGroup	**LandingController**	LandingControl	OPT
	TakeoffController	TakeoffControl	OPT
	FlightController	FlightControl	MAN
CommunicationGroup	CommunicationInterface	Communication	MAN
SensorGroup	**DisplayViewer**	SensorOperation ImageDisplay	OPT MAN
	ImageOperator	SensorOperation ImageOperation	OPT MAN
	LDRFOperator	SensorOperation LDRFOperation	OPT OPT
	ImageContrastAlgorLibrary	SensorOperation ImageContrastAlgor	OPT ALT
WeaponControlGroup	**WeaponOperator**	WeaponControlOperation	OPT
FlightScenarioGroup	ScenarioFileManager	ScenarioFileManagement	MAN
	MissionPlanEditor	FlightScenarioPlanning MissionPlan	OPT OPT
	FlightPlanEditor	FlightScenarioPlanning FlightPlan	OPT OPT

4.2 Modeling Structural Relationship Between Logical Components

One or more UML class diagrams are used to define the structural relationships among the logical components. In the example, the dependencies among the logical component groups are modeled before the direct relationship among the logical components is modeled. As shown in the figure below, the relationship between the identified logical component groups is marked as dependency, and the data dependency is described. The generation of model of the structural relationships at the logical component-group

level by abstracting the complexities of direct relationships among the logical components can enhance the understanding of the design.

4.3 Variability Modeling of Physical Components

The logical components assigned to the physical components in the example, and the corresponding variability of the physical components, are listed in Table 2. The variability of a physical component depends on the variability of a logical component assigned to that physical component. If one or more mandatory logical components are assigned to it, then the physical component cannot become a variable element. On the other hand, if all assigned logical components are optional design elements, then the physical component must become a variable element.

Table 2. Association between logical component and physical component

Physical component	Assigned logical component	Variability
GCSApplication	GCSApplication InterCoponentGateway ≪VP≫LoginManager	
ContrastAlgorLibrary	≪VP≫ImageContrastAlgorLibrary	≪VP≫
ImageDisplyer	≪VP≫DisplayViewer	≪VP≫
ScenarioEditor	≪VP≫FlightPlanEditor ≪VP≫MissionPlanEditor ScenarioFileManager	
SensorImageOperator	≪VP≫ImageOperator ≪VP≫LDRFOperator	≪VP≫
WeaponController	≪VP≫WeaponOperator	≪VP≫
Communicator	CommunicationInterface	
FlightDeviceController	FlightController ≪VP≫LandingController ≪VP≫TakeoffController	

According to the physical model's variability modeling rules, ContrastAlgorLibrary, ImageDisplayer, ScenarioEditor, SensorImageOperator, and WeaponController physical components could be specified as variable elements.

5 Conclusion

In this paper, three views and models for constructing a product-line architecture and specific construction guidelines are presented. The conceptual view allowed to define the logical units that make up the product-line architecture and the relationships among them. The guidelines for identifying logical units based on feature models are presented. The execution view allowed to define the execution information of the logical unit and suggested guidelines for distinguishing physical components from logical

components. In the deployment view, the relationship between the execution device and the physical component could be defined, and, part of the execution device, the criteria for identifying nodes was suggested. In this study, the modeling guidelines are suggested to establish feature-based traceability, which is essential for successful product-line engineering, and the traceability among models was suggested to be defined naturally and by stages. To confirm its effectiveness, the guidelines were applied to a GCS (Ground Control System) example. Although not applied to all examples, a novice engineer who built the product line for the first time proved that it was possible to apply the process more systematically according to the guidelines based on the model, which is a byproduct of the previous stage.

References

1. Bosch, J.: Software product lines and software architecture design. In: Proceedings - International Conference on Software Engineering, January 2001
2. Matinlassi, M.: Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA. In: Proceeding ICSE 2004, Proceedings of the 26th International Conference on Software Engineering, pp. 127–136 (2004)
3. Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley Professional, Boston (2004)
4. Capilla, R., Ali Babar, M.: On the role of architectural design decisions in software product line engineering. In: Morrison, R., Balasubramaniam, D., Falkner, K. (eds.) ECSA 2008. LNCS, vol. 5292, pp. 241–255. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88030-1_18
5. Tekinerdoganm, B., Cetin, S., Savcı, F.: Exploring architecture design alternatives for global software product line engineering. In: Proceedings of 6th International Conference on Software Engineering Advances, pp. 515–521 (2011)
6. Chaudhary, A., Verma, B.K., Raheja, J.L.: Product line development architectural model. In: Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology, pp. 749–753 (2010)
7. Gharibi, G., Zheng, Y.: ArchFeature: a modeling environment integrating features into product line architecture. In: Proceedings of 31st Annual ACM Symposium (2016)
8. Lima, C., Chavez, C.: A systematic review on metamodels to support product line architecture design. In: Proceedings of SBES 2016, Proceedings of the 30th Brazilian Symposium on Software Engineering, pp. 12–22 (2016)