■ 2006년 정보과학 논문경진대회 수상작

프로덕트라인 공학에서의 체계적인 핵심 자산 설계 프로세스

(A Systematic Process for Designing Core Asset in Product Line Engineering)

라 현 정 † 김 수 동 ††
(Hyun Jung La) (Soo Dong Kim)

요 약 프로덕트라인 공학은 한 프로덕트라인에 속하는 여러 어플리케이션들이 공유할 수 있는 핵심 자산을 재사용하는 새로운 패러다임으로, 대표적인 소프트웨어 재사용 방법으로 넓게 수용되고 있다. 핵심 자산은 프로덕트라인의 여러 멤버에서 재사용될 수 있기 때문에, 공통성과 가변성을 잘 정의하여 높은 재사용성을 가진 핵심 자산을 개발하는 것은 생산성을 향상시켜 고품질의 어플리케이션을 빠른 시간 내에 개발하는데 필수 요소이다. 프로덕트라인 공학을 적용한 기존 방법론에서도 핵심 자산의 중요성을 강조하였지만, 대개 공통성과 가변성을 분석하는데 초점이 맞추어져 있었다. 그리고, 일부 방법론에서는 핵심 자산을 개발하는 프로세스를 제안하고 있지만, 핵심 자산의 모든 구성 요소를 개발하는 체계적인 프로세스, 지침, 산출물 양식이 다소 부족하며, 이는 핵심 자산을 설계하는데 많은 어려움을 초래한다. 본 논문에서는 핵심 자산 설계를 위한 체계적인 프로세스와 기법, 산출물의 템플릿을 제안한다. 그리고, 제안된 프로세스가 실제로 어떻게 적용되는지 검증하기 위한 사례연구를 수행한다. 제안된 프로세스, 지침, 산출물 템플릿을 사용함으로써 보다 재사용성의 이점을 최대한 활용할 수 있는 동시에 고품질 핵심 자산을 체계적이며 효율적으로 개발할 수 있을 것으로 기대된다.

키워드 : 프로덕트라인 공학, 핵심 자산, 설계 프로세스

Abstract Product line engineering (PLE) is one of the most recent and emerging reuse approaches in software engineering. Core asset, which is a reusable unit of PLE, is shared by several members in a product line (PL). So, developing a well-defined core asset is a prerequisite to increase productivity and time-to-market. Existing PLE methodologies emphasize the importance of core asset but mainly focus on analyzing core asset. And, several processes for designing core asset do not fully cover all elements of core asset which is from product line architecture (PLA) to decision model and need to augment systematic process, detailed instructions, and templates of artifacts. These problems result in difficulty with designing core asset and applying PLE. In this paper, we present an overall process and templates of artifacts to design core assets. And, we apply proposed process to a case study in order to show its applicability. With the proposed process, detailed instructions, and templates of artifacts, we believe that we can more systematically and more easily design high-quality core assets and we fully cover product line architecture, component, and decision model when designing a core asset.

Key words : Product Line Engineering, Core Asset, Design Process

· 이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음. (KRF-2004-005-D00172)
† 정 회 원 : 숭실대학교 컴퓨터과학과
hjla@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
sdkim@comp.ssu.ac.kr
논문접수 : 2006년 5월 18일
심사완료 : 2006년 8월 23일

1. Introduction

PLE is an emerging reuse approach by using core asset which is larger-grained reuse unit than component in component-based development, and it has two processes; core asset engineering and application engineering [1]. Core asset engineering,

which is also called domain engineering or framework engineering, is to analyze a domain in order to derive common asset, to establish a PL that can be shared by various members in the domain, and finally to produce a core asset. Application engineering is to efficiently develop a specific application by instantiating the common assets and integrating application-specific functionalities with an instantiated core asset. Core asset is a reusable unit of PLE which can be shared by several members in a PL, and it consists of PLA, components, and decision model. Because a core asset is reused in several applications of a PL, a high-quality core asset helps an application engineer to develop a specific application more easily and effectively, and finally leads to increase productivity and time-to-market. Existing PLE methodologies emphasize the importance of core asset but mainly focus on analyzing core assets. And, several literatures propose process for designing core assets. But they do not fully cover all elements of core assets, which is from PLA to decision model, and need to strengthen systematic process, detailed instructions, and templates of artifacts. These problems result in difficulty with designing core assets and applying PLE.

In this paper, we present an overall process and templates of artifacts to design core assets. An overall process for designing core asset consists of five phases, and each phase is composed of several activities which propose detailed instructions. And, we apply a proposed process to a case study in order to show its applicability. The domain of case study is rental domain which has currently two members and, the case study is performed by following the proposed process step by step. With the proposed process, detailed instructions, and templates of artifacts, we believe that we can design high-quality core assets more systematically and more easily and we fully design PLA, component, and decision model when designing a core asset.

2. Related Works

FAST, which stands for Family-oriented Abstraction, Specification, and Translation, is organized into three sub-processes [2]. *Qualify Domain* is to identify families with business case analysis. *Engineer Domain* is to develop an environment and processes for application engineering and is more or less related to core asset development. *Engineer Application* is to produce family members in response to customer requirements. Engineer Domain consists of two activities; *Analyze Domain* and *Implement Domain*. In the Analyze Domain, commonality and variability are analyzed, decision model is defined, family design and application engineering environment is developed. And in the Implement Domain, application engineering environment is implemented and documented in order to be effectively used in Engineer Application. Application engineering environment contains library, code template, and various types of the tools that can help to build application. That is, an application engineering environment is similar to core asset. However, this methodology does not completely cover PLA that represents an overall structure of a core asset.

FOPLE, which stands for Feature Oriented Product Line Software Engineering, focuses on engineering principles and guidelines used for feature modeling, architecture design, and component development [3]. FOPLE consists of six activities; *Feature Modeling* capturing commonalities and variabilities in terms of product features, *Architecture Design* allocating features to architectural components and specifying data and control dependencies between architectural components, *Architecture Refinement* refining functional architecture and allocating components to concurrent processes and network nodes, *Candidate Object Identification* identifying candidate objects based on feature model, *Design Object Modeling* developing object model based on functional architecture, candidate object, and *Component Design* refining the process and deployment architectures into concrete component using design object model. This method fully covers from PLA to component, but there is a room to improve a way for dealing with variability such as decision model.

PuLSE, which stands for Product Line Software Engineering, has three main elements; Deployment Components, Technical Components, and Support Components [4]. Among theses, technical compo-

nents provide technical know-how needed to develop product line, and they consist of six activities; *PuLSE-BC* in which a tailored process to an enterprise context is produced, *PuLSE-Eco* which is used to identify a product line scope and to produce a product map, *PuLSE-CDA* in which domain concept and their interrelationship are elicited, structured, and documented to domain model and decision model, *PuLSE-DSSA* which is a stage to define reference architecture of product line, *PuLSE-I* specifying, instantiating, and validating a product and *PuLSE-EM* which controls the evolution of the product line infrastructure. Generally, PuLSE-CDA and PuLSE-DSSA can be included in core asset engineering. This method focuses on dealing with commonality and variability and has separate phases for developing reference architecture. However, this method needs to augment a way of designing components and their relationships which are included in reference architecture.

KobrA is a component-based PLE based on UML [5]. *Framework Engineering,* where a generic, reusable infrastructure is defined consists of five activities; *Context Realization* which describes the prominent properties of the environment and deciding framework scope, *Komponent Specification* which produces models describing what the component does and externally visible properties of komponents, *Komponent Realization* which produces models describing how Komponent does in detail, *Komponent Implementation* which describes how the Komponent Specification is implemented using language-level constructs and physical components, and *Component Reuse* which integrates an external component into a Komponent tree. After realizing execution environment in context realization, framework is gradually refined by iterating specification and realization. This method provides more or less detailed instructions and several templates including decision model. However, this method highly depends on components and does not exactly cover a phase for designing PLA.

3. Process, Instructions, and Templates of artifacts

The overall process to design a core asset is presented in Figure 1. The process consists of five phases; *Product Line Architecture Modeling, Preliminary Component Modeling, Preliminary Interface*

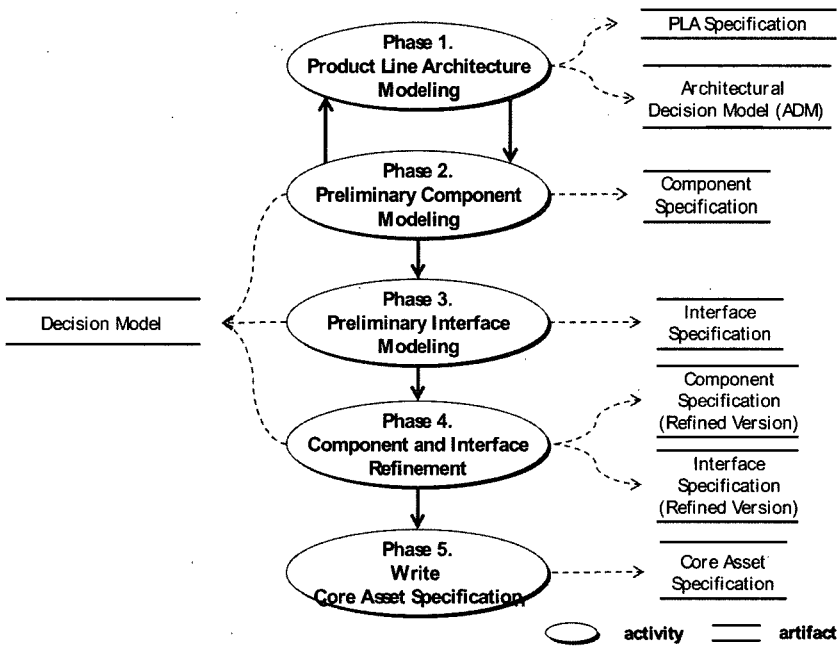

Figure 1 The Overall Process

Modeling, *Component and Interface Refinement*, and *Write Core Asset Specification*. Because a systematic process for designing a core asset should fully deal with all elements of the core assets, each phase is derived in terms of all elements.

3.1 Phase 1. Product Line Architecture Modeling

This phase is to design PLA that includes commonality and variability of all products in a PL. PLA is a generic architecture which can be used in all product line members and it represents the overall structure of all applications in a PL. So, this phase is important to design a well-defined and high-quality core asset. But it is a difficulty work to perform this phase because modeling architecture is very conceptual work and we also should consider architectural variabilities.

Input and Output : This phase begins with product line requirement specification and commonality and variability (C&V) model. Product line requirement is a requirement which is gathered from product line members and captures common and variable information of a PL. From this requirement, architectural-relevant requirements can be acquired and are mainly related to non-functional requirements. PLA realizes common and variable architectural-relevant requirements. C&V model is an output of analyzing core asset which is carried out before this phase and contains all information on C&V in the analyzed core asset.

Output artifacts of this phase are PLA specification and architectural decision model (ADM). PLA specification addresses architectural drivers, styles, components, and rationales in terms of each viewtype and ADM specifies architectural variations on drivers, styles and components-itself.

Instruction : This phase consists of five activities as shown in Figure 2; *Define Architectural Driver*, *Define High-level PLA*, *Assign Components*, *Write PLA Specification* and *Validate PLA*. Each activity is carried out in several steps.

• Activity 1a. Define Architectural Driver

This activity is to derive a set of architectural drivers for a PL. Acquiring right architectural drivers is an essential prerequisite to developing well-designed architecture [6]. This activity begins with product line requirement specification and C&V model, and ends with producing architectural

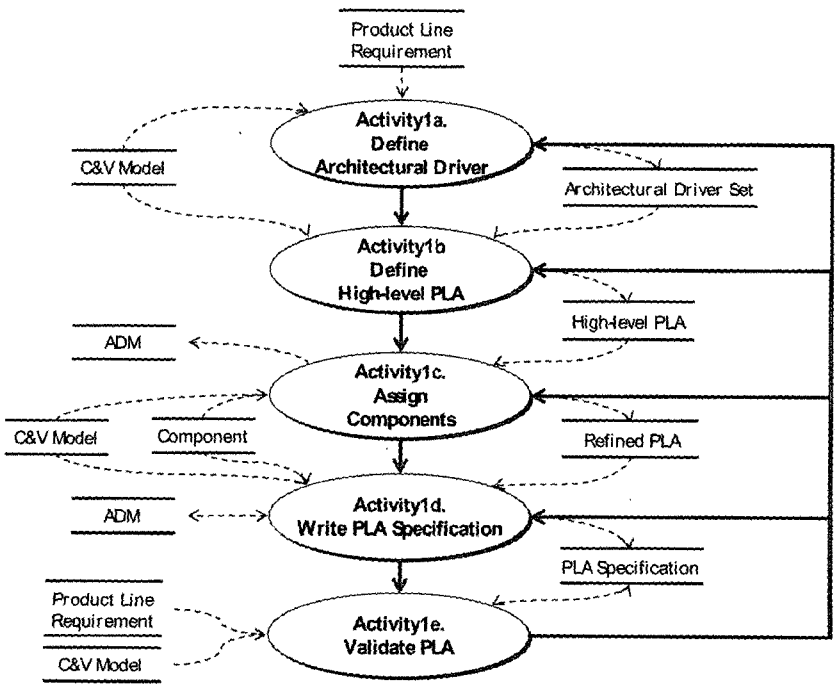

Figure 2 Activities of PLA Modeling

driver set table which is included in PLA specification. An architectural driver set table addresses an id of an architectural driver which is used as reference for following steps, a name of an architectural driver, description, and variability type as in Table 1. Because architectural drivers are derived from a set of requirements, identified architectural drivers may be applied to all the product line members or not. So, architectural drivers can be categorized to common or variable drivers.

First, architectural-relevant requirements are extracted from a product line requirement specification. Although all requirements may affect designing an architecture, only non-functional requirements are considered because all requirements do not have impact on the designing PLA and non-functional requirements have a tendency to have impact on shaping architecture heavily. Second, architectural drivers are derived from the architectural-relevant requirements by analyzing and itemizing as architectural drivers. An architectural driver is the combination of influential functional, quality, and business requirements shaping the software architecture for the base set of application [1]. For each extracted architectural-relevant requirement, a name is given for further references in following activities after architectural-relevant requirements are analyzed in terms of an architectural driver. Third, architectural drivers should be classified into the common and variable drivers (i.e. alternative and optional drivers) based on C&V model [7][8] since all the derived architectural drivers represent various needs of product line members and several members do not need some drivers among the derived drivers. Finally, the architectural drivers are prioritized according to the commonality and significance. In general, priority of an architectural driver is proportional to the degree of commonality and significance.

• Activity 1b. Define High-level PLA

This activity is to select architectural styles derived from architectural drivers and integrate selected architectural styles into high-level PLA. High-level PLA is a preliminary architecture which is configured with architectural styles, and its elements are not concrete and need to be refined further activity. This activity begins with a set of architectural drivers which is described in architectural driver set table and C&V model, and ends with producing high-level PLA, architectural resolution table, and ADM. Templates of architectural resolution table and ADM are presented in Table 2 and Table 3 respectively. High-level PLA and architectural resolution table are included in PLA specification.

As shown in Table 2, an architectural resolution table addresses all architectural decisions solving selected architectural drivers and contains an id of an architectural decision, architectural decision type which can be architectural style or component, an architectural decision name of applied styles or components, rationale which is main reasons for selecting the architectural decisions, and variability type.

Table 1 Architectural Driver Set Table

ID	Driver Name	Description	Variability Type
...	...	...	Mandatory | Alternative | Optional

Table 2 Architectural Resolution Table

ID	Archi. Decision Type	Archi. Decision Name	Archi. Driver	Rationale	Variability Type
...	Style | Component	...	...	...	Mandatory| Optionall Alternative

Table 3 Architectural Decision Model

Variation Point		Variant	Var. Type	Case	Effect	Instantiation Task
ID	Name					
...	...	...	Opt | Alt	V1: Used	...	...
				V2: Not Used	...	...

In ADM as shown in Table 3, there are much information about variability; an id and a name of a variation point, a set of variants, variability type which is optional or alternative variability, an effect in which other variation points are filled in because of variability dependency relationship [9], and instantiation tasks which are described in detail in order to be effectively utilized in instantiating core asset.

First, appropriate architectural views are selected according to a set of architectural drivers. Architectural view is a representation of a set of system elements and the relationship associated with them [10]. There are several types of view type such as module view, component-and-connector view, and allocation view. Architectural view helps to mitigate more or less complex of PLA because view allows us to separate concerns while building or analyzing an architecture with different perspectives. Second, architectural decisions are selected in terms of architectural drivers. Architectural decision is a method to resolve and design architectural drivers [8], and it may include architectural styles and components [11]. Based on the selected views, an architect firstly explores and lists candidate architectural styles for each architectural driver. And, an architect decides appropriate architectural styles by using strategies, project policies, or constraints. Then, components are derived from architectural drivers which are not designed in PLA by using architectural styles or patterns. For example, member wants to maintain security by using firewall. This requirement is satisfied by using not a certain architectural style but a component which is called 'Firewall'. Third, selected architectural styles or components are classified into mandatory, optional, and alternative ones and these variable styles and components are specified in ADM. Finally, a high-level architecture is generated by integrating selected architectural styles. Mandatory styles are integrated into high-level PLA firstly, and variable styles are added into high-level PLA one by one.

• Activity 1c. Assign Components

This activity is to assign pre-identified components onto high-level PLA. That is, this activity is carried out by mapping components defined from functional and non-functional requirement analysis to coarse-grained elements in the high-level PLA. Before conducting this activity, identifying components should be completely performed. This activity begins with high-level PLA and pre-identified component list, and ends with producing a concrete PLA and ADM which is refined by adding component-itself variability.

First, pre-identified components are assigned to high-level PLA. The components can be derived from functional and non-functional requirements. These components can be acquired by clustering related use cases and this method is specified in phase 2. And, components derived from non-functional requirements can be additionally acquired while architectural decisions are selected. Second, components are assigned to high-level PLA. Mandatory components are firstly assigned and variable components are assigned to high-level PLA. Components acquired by clustering relative use cases are smaller grained and more or less concrete than coarse-grained elements in high-level PLA. So, the coarse-grained elements should be refined with the concrete components by decomposing or mapping. Finally, variable components are specified in ADM.

• Activity 1d. Write PLA Specification

This activity is to write a PLA specification including ADM. PLA, consists of mandatory and variable components and inter-component relationships. Mandatory components and inter-component relationships can be documented by using a conventional architectural description language or a graphical representation as in [10]. But variable components and inter-component relationships should be described by using additional representation because variability information should be fully documented and represented. Information on variability is effectively used in instantiating core asset, so architectural variabilities are separately managed in ADM. Although architectural variabilities are identified and specified before this activity, more detailed information on architectural variability are represented in ADM.

• Activity 1e. Validate PLA

This activity is to evaluate PLA whether PLA

satisfies product line requirement specification, especially architectural-relevant requirements. There are several evaluation methods [12]. In this paper, a check list is used. To validate PLA, PLA specification, ADM, and a pre-defined check list are required, and output of this activity is evaluated PLA and result. First of all, we define a check list to evaluate PLA for this activity as shown in Table 4.

Check points in the check list are classified with an artifact of each activity. Depending on activity goals, check points emphasize completeness, accuracy, efficiency, or conciseness. In architectural drivers, check point focuses on completeness, accuracy of architectural driver for product line requirements. Check points of architectural styles are efficiency of extracted styles for architectural drivers, the point of instantiated styles is completeness for quality attribute and functional requirements, and the point of PLA is its suitability for integrated style set. For ADM, completeness and efficiency are indicated. Based on the list, we may decide whether PLA design should be refined or finalized.

3.2 Phase 2. Preliminary Component Modeling

The goal of this phase is to identify preliminary components and to design and specify classes for each identified component. This phase and previous phase can be performed in parallel because components identified in this phase are assigned to high-level PLA.

Input and Output : This phase begins with product line specification, use case model and C&V model, and generates component specification and decision model. Component specification describes information on common and variable components and contains component list and family object model in which variability is depicted in a conventional object model. Like ADM, decision model specifies all information on inner-component variability and templates is same as ADM. Decision model should contain not only all types of architectural variabilities such as architectural driver, architectural style, and component-itself variablities but also all types of inner-component variablities such as attribute, logic, workflow, interface, and persistent variability [13]. Architectural variability is specified in "Phase 1. Product Line Modeling" and inner-component variability is specified in this and following phases.

Instruction : This phase consists of three activities as shown in Figure 3; *Cluster Use Cases into Component*, *Conduct Family Object Modeling*, and *Assign Classes*.

• Activity 2a. Cluster Use cases into Component

This activity is to identify candidate components by grouping related use cases. Generally, architecture consists of components and inter-component relationships. Through this activity, one element of architecture is identified. This activity begins with

Table 4 Check List for Validating PLA

Artifact	Check Point
Architectural Driver	Do the architectural drivers meet non-function requirement?
	Are derived architectural drivers used for designing PLA?
	Is the priority of driver right?
	Are variable architectural drivers defined based on adequate criteria?
Architectural Style	Are selected views satisfied with all architectural drivers?
	Are all of drivers applied into styles?
	Isn't a driver unnecessarily applied into several styles?
	Is the selected style efficient?
PLA	Is identified overlapped area right?
	Does the overlapped area resolve effectively?
	Do extracted components cover PL requirements?
Architectural Decision Model	Are all variations of drivers delegated to adequate variation point?
	Is all variants necessary?
	Are all variability propagations in overlapped area detected?
	Are all variation points and variants consistent through the artifacts?

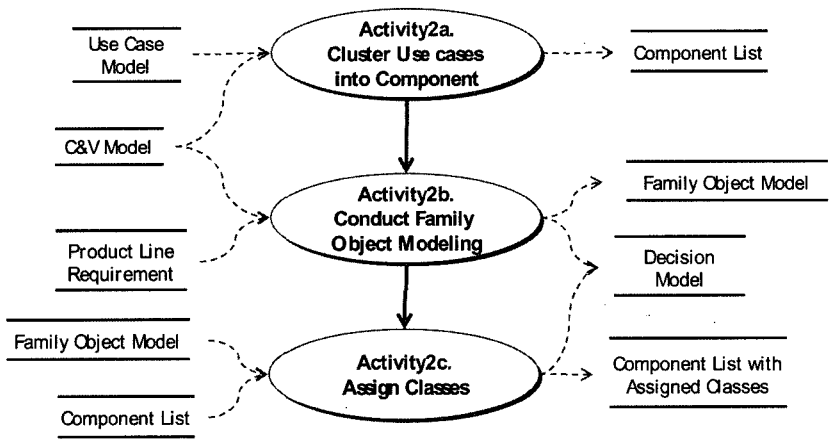

Figure 3 Activities of Preliminary Component Modeling

use case model of a PL and C&V model which is for classifying component into two groups; components not containing inner-component variability and components containing inner-component variability. And, this activity ends with producing component list which contains an id and a name of components and use cases assigned in a component.

First, use cases are clustered into a component by using clustering method [14]. In the clustering method, functional dependencies between use cases are defined by measuring sub-systems, actors, shared data and use case relationship. And, related use cases are clustered into a component by using clustering algorithm, and conflicts whose type are use case belonging to multiple components and use case isolated from others must be resolved by considering coupling between conflicted use cases and other sets of use cases. Finally, components and use case list are specified in component list table.

• Activity 2b. Conduct Family Object Modeling

This activity is to model data and structure of the members (applications) in a PL in terms of commonality and variability. This activity is similar to traditional object modeling except for dealing with commonality and variability, so we must present variability representation in the family object model such as stereotype, tagged value, and etc. This activity begins with product line requirement specification and C&V model. C&V model is used to represent variability information in the family object model. And this activity ends with producing a family object model and a decision model.

Because this activity is same as a traditional method of object modeling except for dealing with variability, it is important to specify and represent variability for classes and relationships. Figure 4 shows a variability notation by representing stereotype and note (e.g.«VP» and {vType=L}) in the family object model [15].

• Activity 2c. Assign Classes

This activity is to allocate related classes to each identified component. This activity begins with pre-defied component list and classes which are represented in the family object model, ends with producing component list with assigned classes. This activity can be automatically performed by using clustering method [14].

First, we draw an interaction diagram such as sequence diagram for each use case in a component. Second, objects participating in each use case are identified because a sequence diagram represents messages and related classes which send and get the messages. Third, a set of classes which are included in a sequence diagram is assigned to a component. Fourth, conflicts in assigning classes onto components are resolved by using criteria proposed in [14]. These criteria consider number of message passing paths, weight for the role of an entity class in terms of Create,

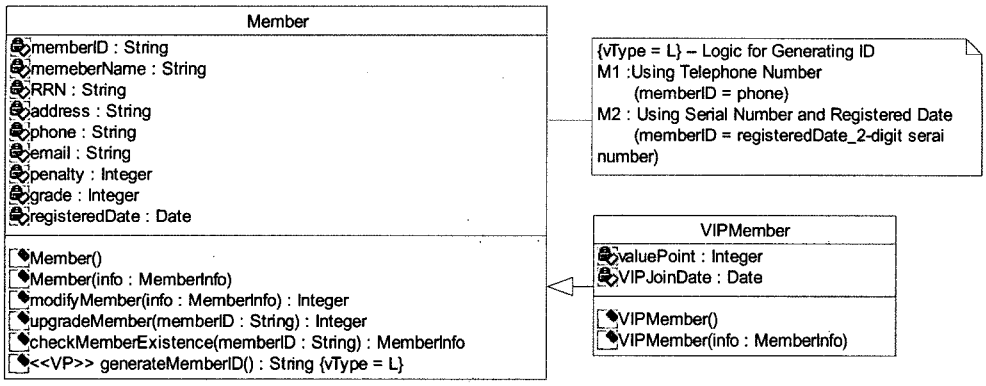

Figure 4 Sample for Variability Notation

Retrieve, Update, and Delete, and relationships between classes such as dependency, aggregation, association, inheritance, and composition. Finally, variability information can be added in decision model which is continuously refined for designing core asset. Components having variability are classified in this step.

3.3 Phase 3. Preliminary Interface Modeling

The objective of this phase is to define and model functionalities of components and identify inter-component relationship. This phase is similar to interface modeling of component-based modeling except for not dealing with customized interface. Customizing components is performed in the source-level, while instantiating a core asset is performed in the model-level. So, designing a customized interface is not considered in this phase.

Input and Output : This activity begins with use case model and component specification, and ends with interface specification and refined decision model. Interface specification describes information on provided and required interfaces and workflow designs which represent message-passing relationship between classes and components. Through this phase, decision model is refined by adding interface variabilities.

Instruction : This phase consists of three activities as shown in Figure 5; *Cluster Design Provided Interface*, *Design Workflows*, and *Design Required Interface*.

• Activity 3a. Design Provided Interface

This phase is to define interfaces for invoking services provided by components and to model functionalities of components. Input of this phase is C&V Model and component specification, and output of this phase specification of provided interfaces

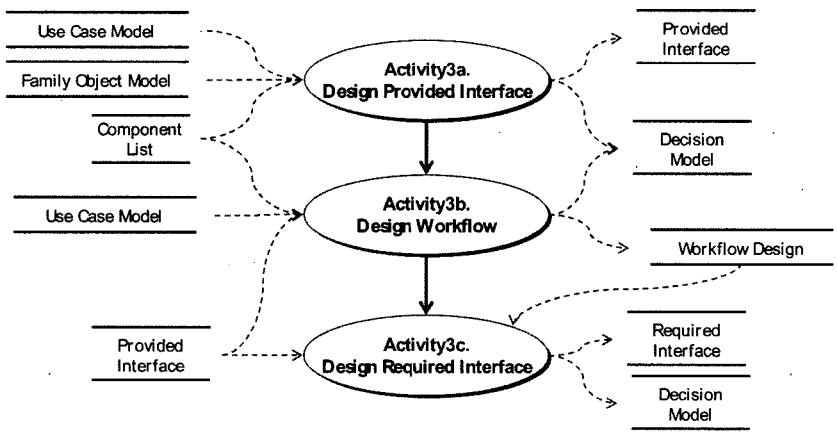

Figure 5 Activities of Preliminary Interface Modeling

and decision model to which interface variability is added. Provided interface specification may contain signature, return type, parameters, pre and post conditions, invariants, and variability information of an operation specified in provided interfaces.

First, once component is chosen, we extract several messages which are sent from an actor to system for each use case included in a chosen component by using sequence diagram. Interfaces should be defined from the view point of product line members, so we consider the set of use cases in each component. We can simultaneously extract operations to be included in provided interfaces from messages which are sent from an actor to system. Second, we collect the defined operations of all use cases in the chosen component and define the collections as provided interface of the chosen component. Step 1 and 2 must be iteratively performed until provided interfaces of all components are defined. Finally, provided interfaces are specified by using extracted operations, and interface variabilities are represented and specified in an interface specification and a decision model. Similar to family object model, several notations such as stereotype and tagged value are used in order to represent variabilities.

• Activity 3b. Design Workflows

This activity is to model and design the message flows for each function in a component. Input of this activity is C&V model, component specification, and provided interface specification. And output is workflow design model by using an interaction diagram such as sequence diagram and refined decision model.

First, for each method in an interface, one identifies actors and objects or external components participating in its computation by using sequence diagram. This step is similar to drawing sequence diagram except for dealing with both classes and components in order to identify inter-component relationships, and variability information. Like family object model and provided interface specification, workflow variability information should be represented in workflow design model and decision model by using stereotype, tagged value, etc.

• Activity 3c. Design Required Interface

This activity is to identify and specify external components required by current component and identify and model inter-component relationship as specifying required interfaces. Input of this activity is provided interface specification and workflow design model, and output is required interface specification and refined decision model. Template of required interface specification is same as that of provided interface specification.

First, we explore operations provided by other component interfaces by using workflow design. If external service may not already be provided by existing components, this should be defined in the existing components. Then, the external services are specified in required interface specification, and especially variability information is represented and specified.

3.4 Phase 4. Component and Interface Refinement

The objective of this phase is to refine functionalities and behaviors for each component and to refine decision model. Through this phase, components and interfaces which are defined in phase 3 and 4 contain platform-specific information. This phase consists of four activities as shown in Figure 6; *Refine Family Object Model*, *Refine Workflow Design*, *Refine Provided and Required Interfaces*, and *Refine Decision Model*.

The first, second, and third steps are to refine a pre-drawn family object diagram, workflow designs, and provided and required interfaces separately in order to append platform-specific information. And in the fourth step, we refine a decision model by using platform-specific information, and variability information is graphically represented by using variability matrix. Figure 7 shows a part of variability matrix. In the matrix, the 1st, 2nd, and 3rd row represent all the variation points in a core asset, and the 1st, 2nd, and 3rd column show component name, class name, and operation name specified in a core asset. Colored cells represent instantiation tasks in order to resolve variation points. These instantiation tasks are referred to a decision model. For example, a variation point named 'LV_1. For registering product' is related to a 'Product()' operation of 'Product' class in 'Com_01. C_InventoryMgr' component. To resolve this varia-

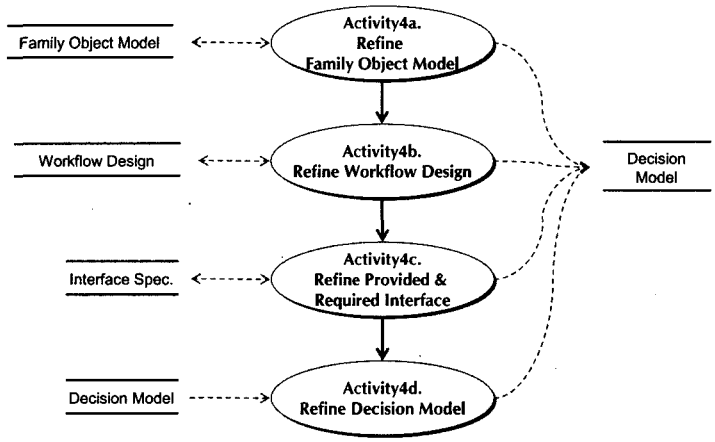

Figure 6 Activities of Component and Interface Refinement

			Architecture	Component				
			Component	Attribute			Logic	
			CV_1. C_WebUI Mgr	AV_1. ISBN	AV_2. author	AV_3. page	LV_1. For registering Product	LV_2. For Generating memberID
Com_01. C_InventoryMgr	InvemtoryCtrl	registerProduct()						
		ssearchItem()						
		calculateRentalFee()						
		searchProduct()						
		removeProduct()						
	Product	Attributes		D2. Add	D4. Add	D6. Add		
		Product()					D8. Modify algorithm	
		generateProductID()						D9.phone D10. serial no.
		searchProduct()						
		removeProduct()						
	Item	Attributes						
		searchItem()						
		changeRentalStatus()						
	ProductInfo	Attributes		D3. Add	D5. Add	D7. Add		
	ItemInfo	Attributes						

Figure 7 Variability Matrix

bility, application engineer modifies algorithms of 'Product()' operation as specified in 'D8. Modify algorithm'. Since variability matrix represents brief information on variabilities, we should refer to and use decision model in order to find out detailed instantiation tasks.

3.5 Phase 5. Write Core Asset Specification

The objective of this phase is to write a core asset specification. Core asset specification is a kind of core asset's user manual and aims to help the users to develop a target application by using the core asset. There can be two usages of these specifications; the former is used to determine whether to use the core assets, and the latter is used to instantiate the core assets. When application designers develop target applications by using core assets, first of all, they should determine whether the core assets is appropriate for the target application or not. To determine suitability of the core assets, the application designers only know the essential information on the core asset, such as the functionality of the core assets, variation points and variants provided by the core asset, etc. Therefore, in this case, detailed and complicated analysis or design information is not needed. After determining the suitability of the core asset, appli-

cation designers needs more detailed information on instantiating the core assets. So, there can be some information in Core asset specification; the depth of information can vary from non-detailed to detailed, such as functional and non-functional requirements provided by core asset, term dictionary, and instantiation manual which describes overall instantiation steps, etc. To specify this specification in an effective way, the specification can be described in a top-down manner. That is,. conceptual information of a core asset is first described and then detailed information is described. Example of the conceptual information is overall functional and non-functional requirements provided by the core asset. And example of the detailed information is design model of core asset.

4. Case Study

To show the applicability and practicality of the process, we present a case study applying the proposed process. The case study is for 'Rental Management' domain of which members are *Dream Library System* and *Best Video Rental System*. Before conducting a proposed process, commonalities and variabilities should be fully and completely identified. So, commonalties and variabilities in a rental PL are briefly presented in Table 5 and Table 6. Due to space constraints, we will focus on requirements about inventory management such as registering and deleting books or videos.

4.1 Phase 1. Product Line Architecture Modeling

In activity 1a, we can derive three mandatory and two optional architectural drivers from common and variable non-functional requirements as shown in Table 7. In column of 'Driver Name', 'A :: B' symbol means that B is a kind of sub-characteristics of A. For example, 'Reliability:: Data Consistency' means that 'Data Consistency' is a kind of sub-characteristics of 'Reliability'.

In activity 1b and 1c, architectural decisions are derived from upper architectural drivers. Selected architectural decisions and their description are shown in Table 8. To support three mandatory and two optional architectural drivers, 'layered style', 'shared-data style', 'C_LoggingService', 'C_TextUIMgr', and 'C_WebUIMgr' components are selected.

In activity 1b and 1c, we can represent these architectural decisions by using diagrams as shown in Figure 8. PLA of this figure is represented with module view and concrete components are already assigned to PLA; C_MemberMgr, C_InventoryMgr, C_RentalMgr, and C_ReservationMgr compoments.

In activity 1e, identified architectural variability can be specified and refied in ADM as Table 9. PLA of the rental core asset have two component-itself variabilities, 'C_LoggingService' supporting

Table 5 Common Functional and Non-functional Requirement

FR ID	Sub-Domain	Requirement	Common Functional and Non-functional Requirements
FR_01	Inventory Management	Register Product	This requirement is to register product information when a new item is acquired.
FR_02		Remove Product	This requirement is to delete product information.
NFR_01	Modifiability	Business Logic Modifiability	Business logic should be independent of changing user interface.
NFR_02	Reliability	Data Consistency	All function modules should share consistent data.
NFR_03	Accessibility	Intranet Based Accessibility	All functionalities can be used through intranet.

Table 6 Variable Functional and Non-functional Requirement

FVP ID	Sub-Domain	Requirement	Variable Functional and Non-functional Requirements
FVP_01	Inventory Management	FR_Inv01. Register Product	Information for registering a product
FVP_02			Logic for registering a product
FVP_03			Logic for generating ProductID
NFVP_01	Reliability	Save Transaction Log	Only Dream Library wants this NFR; This system wants all of the transactions to be logged before storing data to database.
NFVP_02	Accessibility	Internet Based Accessibility	Only Dream Library wants this NFR; Members can use functionality provided be Dream Library all over the world through internet

Table 7 Architectural Driver Set Table for Rental Core Asset

ID	Driver Name	Description	Variability Type
MD_01	Modifiability	Business logic should be independent of changing user interface. So, functionality can be more or less easily modified and maintained without changing other functionalities.	Mandatory
MD_02	Reliability:: Data Consistency	All PL members want systems in which all function modules should share consistent data.	Mandatory
MD_03	Intranet Based Accessibility	All functionalities can be used through Intranet.	Mandatory
OD_01	Reliability::Save Transaction Log	All members want all of the transactions to be logged before storing data to database.	Optional
OD_02	Internet Based Accessibility	Member can use some functionalities related to rental and reservation through the Internet.	Optional

Table 8 Architectural Resolution Table for Rental Core Asset

ID	Archi. Decision Type	Archi. Decision	Archi. Driver	Rationale	Variability Type
MS_01	Style	Layered Style	Modifiability (MD_01)	To support 'MD_01', functionality related to business logic should be separated from functionality related to user interface. Therefore, these functionalities should exist in different layers.	Mandatory
MS_02	Style	Shared-data Style	Reliability:: Data Consistency (MD_02)	To maintain consistent data, data should be managed in a central place. So, data used in multiple components should share one database.	Mandatory
MCom_01	Component	C_TextUIMgr	Intranet Based Accessibility (MD_03)	To support 'MD_03', UI for the Intranet environment must be supported. 'TextUIMgr' performs functionality of managing UI for Intranet.	Mandatory
OCom_01	Component	C_Logging Service	Reliability:: Save Transaction Log (OD_01)	To support 'OD_01', a component for storing transaction log is needed.	Optional
OCom_02	Component	C_WebUIMgr	Internet Based Accessibility (OD_02)	To support 'OD_02', UI for the web environment must be supported. 'WebUIMgr' performs functionality of managing UI for the web. Member can use some services through the web.	Optional

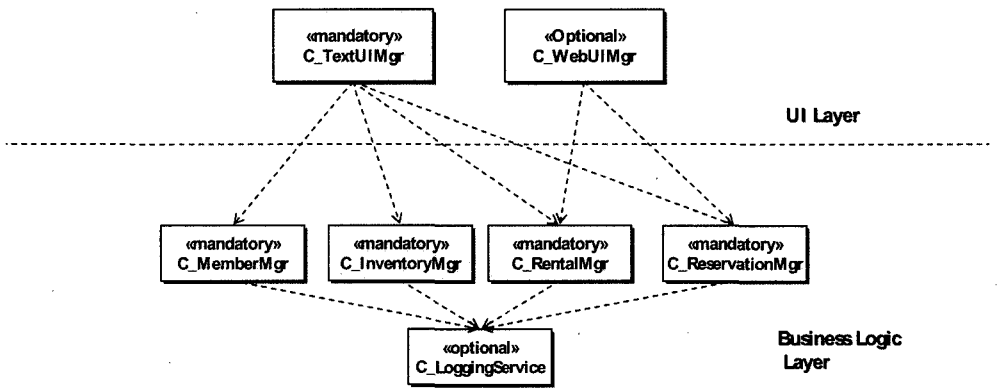

Figure 8 PLA of Rental Core Asset

'OD_01' and 'C_WebUIMgr' supporting 'OD_02', which can exist or not according to product line members.

4.2 Phase 2. Preliminary Component Modeling

Through this phase, a component specification and a decision model are produced. Component specification contains component list and a family

Table 9 ADM for Rental Core Asset

Variation Point		Variant	Var. Type	Case	Effect	Instantiation Task
ID	Name					
CV_01	C_LoggingService	OCom_01. C_LoggingService	Opt	V1:Yes	-	-
				V2:No	-	Remove this component.
CV_02	C_WebUIMgr	OCom_02. C_WebUIMgr	Opt	V1:Yes	-	-
				V2:No	-	Remove this component.

object model, and decision model contains variability information on attribute and logic. But, all these artifacts are refined at phase 7 by adding platform-specific information. So, we present these artifacts in 9.4.

4.3 Phase 3. Preliminary Interface Modeling

Through this phase, a interface specification and a decision model are produced. Interface specification contains provided interface specification, workflow design model, and required interface specification, and decision model contains variability information on workflow and interface. But like artifacts of preliminary component modeling, all these artifacts are refined at phase 7 by adding platform-specific information. So, we present these artifacts in 9.4.

4.4 Phase 4. Component and Interface Refinement

All the artifacts presented in this section refine existing artifacts of phase 3 and 4 by adding platform-specific information.

In activity 4a, we refine the family object model and component list. First, we can get component list by clustering relative use cases which are identified during C&V analysis. Table 10 shows identified components. Due to the space limitation, we only focus on inventory management.

Then, we conduct family object modeling by using product line requirement specification. Figure 9 shows overall classes and their relationships.

Based on Table 10 and Figure 9, we can get list of components to which classes of the family object model are assigned as in Table 11.

In activity 4b and 4c, we refine workflow designs, provided interfaces, and required interfaces. From identified components, we can refine provided interfaces, remodel workflow designs, and refine required interfaces by adding platform-specific information to preliminary artifacts of phase 3 and 4. Because required interface specification is same as provided interface specification, required interface specification is omitted.

Figure 10 shows the provided interface model of 'C_InventoryMgr' component and Table 12 describes one operation of the provided interface specification. In this specification, attribute and logic variability information are described in 'variability' row.

Figure 11 shows a workflow design of 'registerProduct()' operation which specified in provided interface, i.e. 'IpInventoryMgr'. This operation is affected by attribute and logic variability, so workflow represents variability information by using stereotype (i.e.«VP») and note.

Finally, we refine the decision model by using added information specified in the component specification and interface specification, and append 'variability matrix'. Attribute and logic variability, which occur in 'C_InventoryMgr', is described in the decision model such as Table 13. From this table, we know the fact that 'AV_01' affects 'LV_01', which is an instance of variability dependency. In 'Instantiation Task' column, detailed instructions and guidelines are specified in order to instantiate the provided core asset according to a target

Table 10 Component List Table for Rental Core Asset

ID	Component Name	Assigned Use Cases
Com_01	C_InventoryMgr	FR_01. Register Product
		FR_02. Remove Product
		FR_03. Search Product
Com_02	C_MemberMgr	...
Com_03	C_RentalMgr	...
Com_04	C_ReservationMgr	...

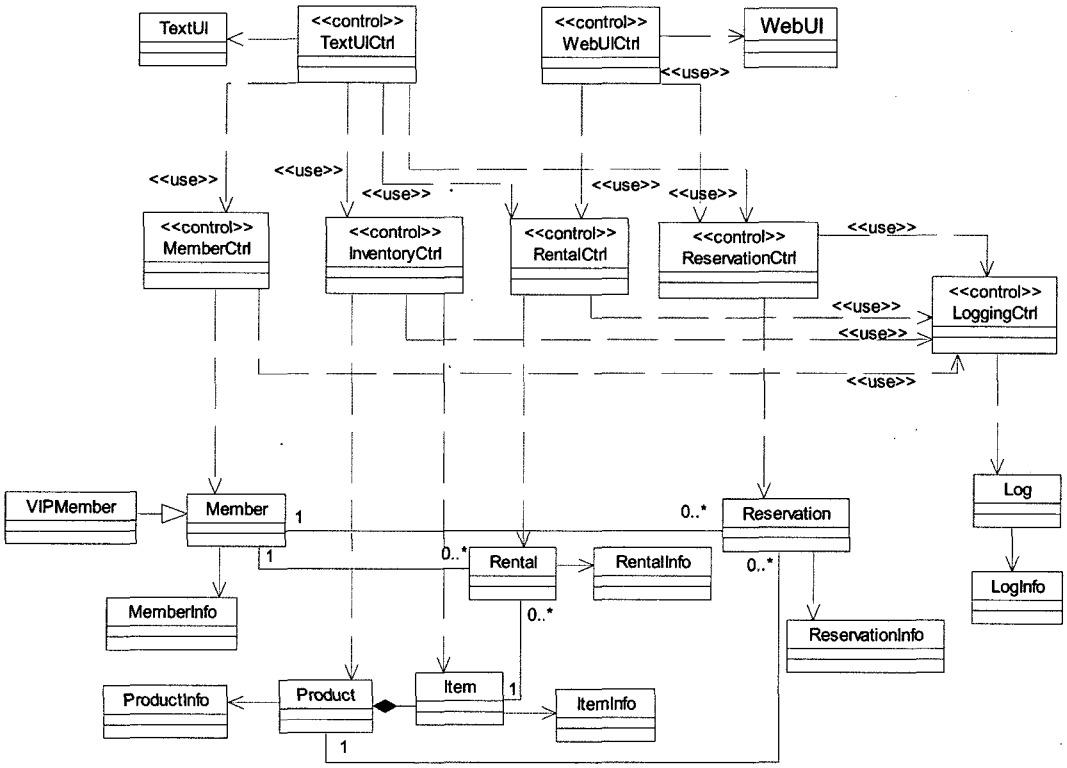

Figure 9 Family Object Model for Rental Core Asset

Table 11 Component List Table with Assigned Classes

ID	Comp. Name	Use Cases	Class List
Com_01	C_InventoryMgr	FR_01. Register Product	Product, ProductInfo, Item, ItemInfo, InventoryCtrl
		FR_02. Remove Product	
		FR_03. Search Product	
Com_02	C_MemberMgr	···	···
Com_03	C_RentalMgr	···	···
Com_04	C_ReservationMgr	···	···
MCom_01	C_TexUIUIMgr	-	···
OCom_01	C_LoggingService	-	···
OCom_02	C_WebUIMgr	-	···

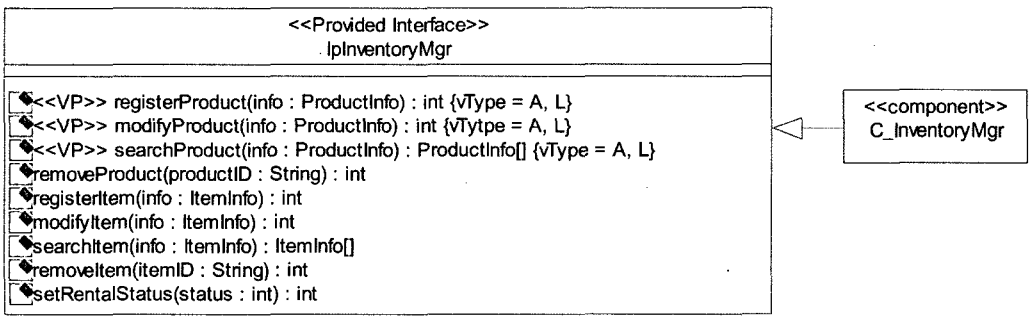

Figure 10 Provided Interface Model for C_InventoryMgr

Table 12 Provided Interface Spec. for C_InventoryMgr

Name	Property	Explanation
registerProduct	Signature	«VP» registerProduct(info : ProductInfo) : int {vType = A, L}
	Return Type	int : return '1' when registration is successfully completed. return '0' when registration is failed
	Parameter	info : a set of the information which are needed when registering a new product in an application
	Pre-condition	This product must not be registered in an application.
	Post-condition	New data of the product is registered in an application and stored in DB.
	Invariant	-
	Variability	To register a new product, required information and logic for processing variable input data vary in PL members. • VariabilityID : AV_01, AV_02, LV_01, LV_02 • VP Type : Attribute, Logic • Description : AV_01 and AV_02 are only used in M2(Dream Library). Members are different in generating productID(LV_02)
	Description	This operation provides functionality where productID is generated and new product is registered in an application. Required information for registering a product is as follows; • Common data : product name, registered date, manufacturer, rental fee, rental period, sales price • Variable data :author, the number of pages (These data are only needed in M2(Dream Library)) And members are different in generating productID. • M1 =registeredDate + 2-digit serial number • M2 = ISBN + 2-digit serial number

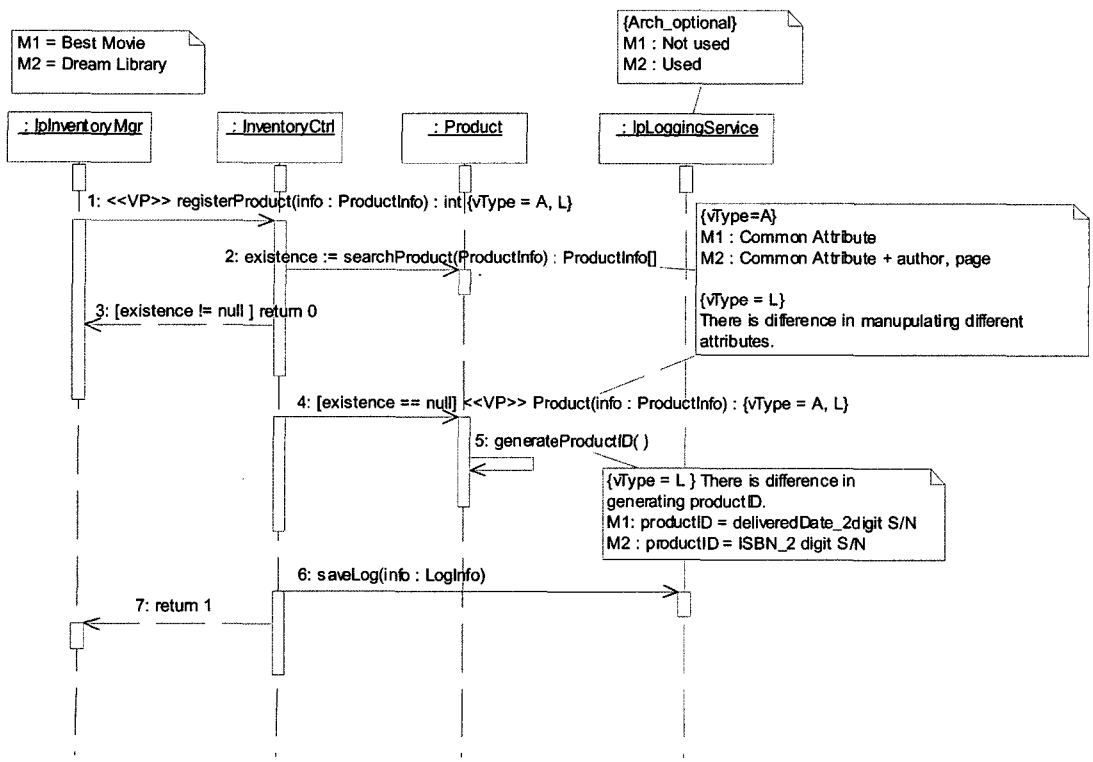

Figure 11 Workflow Design for RegisterProduct()

Table 13 Decision Model for C_InventoryMgr

Variation Point ID	Variation Point Name	Variant	Var. Type	Case	Effect	Instantiation Task
AV_01	Attribute for producer	attribute 'author'	Opt	V1: Used	LV_01	• Delete 'author' attributes in 'Product' class and 'ProductInfo' class (CE 700a. Detailed Comp. Spec.) • Perform attached task of V1 in LV_01
				V2: Not Used	LV_01	...
LV_01	Logic for managing product	V1: Do not use a variable attribute.	Opt	-	AV_01	• Remove «VP» and {vType = A, L} of 'registerProduct()' in 'Product' class and 'InventoryCtrl' class (CE 700a. Detailed Comp. Spec.) • Remove «VP» and {vType = A, L} in 'IpInventoryMgr' (CE 700c. Detailed Interface Spec.) • Remove «VP» and {vType = A, L} in 'registerProduct()' workflow design (CE 700c. Detailed Interface Spec.)
		V2: Use a variable attribute.		-	AV_01	...

Core Asset Element (Comp – Class – Attr / Oper) \ Variation Point			Architecture		Component						
			Component		Attribute		Logic			Workflow	
			CV_01. C_Logging Service	CV_02. C_WebUI Mgr	AV_01. author	AV_02. page	LV_01. For Managing Product	LV_02. For Generating ProductID	LV_03. For Generating memberID	WV_01. Rental Workflow D12	WV_01. Rental Workflow D13
Com_01. C_Inventory Mgr	InventoryCtrl	registerProduct()									
		removeProduct()									
		modifyProduct()									
		searchProduct()									
		registerItem()									
		removeItem()									
		modifyItem()									
		searchItem()									
		setRentalStatus()								2	4
	Product	Attributes			D3. Delete	D5. Delete					
		Product()					D7. Modify algorithm	D8. Del_SII D9. ISBII_SII			
		generateProductID()							D10.phone D11. RRII_SII		
		searchProduct()					D7. Modify algorithm				
		modifyProduct()					D7. Modify algorithm				
		removeProduct()									
	Item	Attributes									
		Item()									
		modifyItem()									
		removeItem()									
		searchItem()									
		setRentalStatus()								6	7
	ProductInfo	Attributes			D4. Delete	D6. Delete					
	ItemInfo	Attributes									

Figure 12 A part of Variability Matrix

application.

Figure 12 shows 'Inventory Management' part of variability matrix. 'AV_01', 'AV_02', 'LV_01' and 'LV_02' are related to 'C_InventoryMgr' component. So, we resolve these variabilities by following D3~D9 decisions.

4.5 Phase 5. Write Core Asset Specification

Through phase 5, we write a core asset specification without specifying detailed and complicated information on instantiating a core

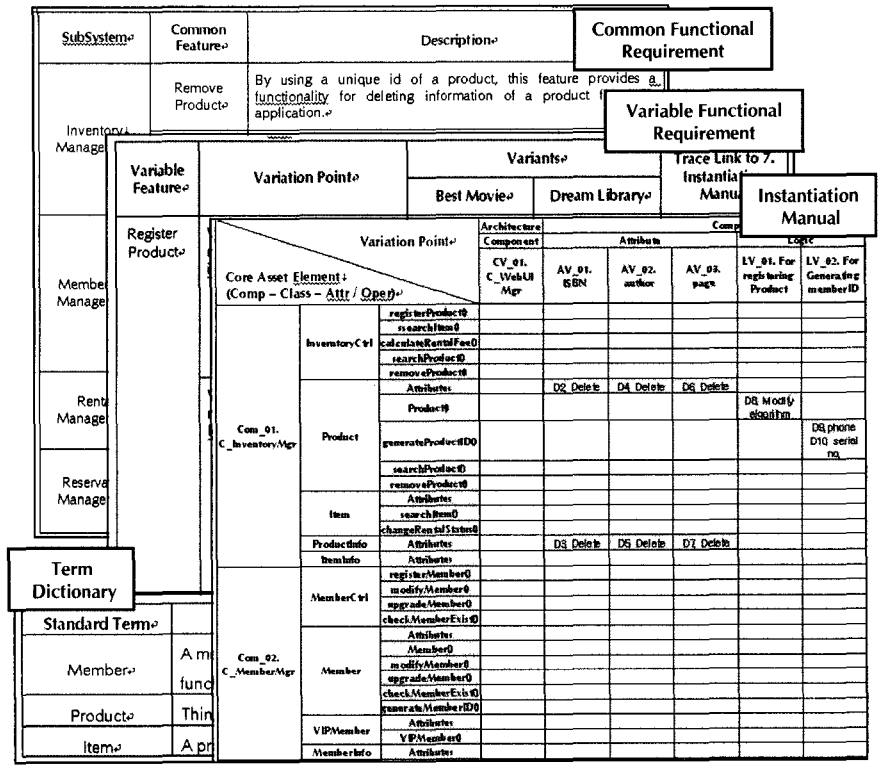

Figure 13 A Part of the Core Asset Specification

asset in order that an application engineer can use the provided core asset effectively. In core asset specification, there are enough information on a provided core asset; functional and non-functional requirements that can be used as is, functional and non-functional requirements that need to be instantiated, term dictionary, instantiation manual which describes overall instantiation instructions without detailed and complicated steps, etc. Figure 13 shows a part of the core asset specification, and this specification are used to determine whether this core asset is appropriate for the target application. Since each specified item refers to specifications generated through previous steps, application designers can know more detailed information.

5. Conclusion

PLE is one of the most recent and emerging reuse approaches in software engineering. Because a core asset is a larger-grained reusable unit of PLE, designing high-quality core asset leads to increase productivity and time-to-market.

In this paper, we presented an overall process, detailed instructions and templates of artifacts. The process consists of five phases; *Product Line Architecture Modeling*, *Preliminary Component Modeling*, *Preliminary Interface Modeling*, *Component and Interface Refinement*, and *Write Core Asset Specification*. Each phase is several activities which present detailed instructions. And we applied the proposed process to a case study which is for designing rental core asset of *Best Movie Renal Application* and *Dream Library Rental Application*. From the case study, we observed the needs of effective methods for managing a large amount of artifacts and offering these artifacts to an application engineer. And a method for managing variabilities may be reinforced.

The proposed process, detailed instructions, and templates of artifacts can help to more systematically and more easily design high-quality core asset by fully and importantly covering PLA,

component, and decision model. Especially, this proposed process may be appropriate for any domain which has a high commonality and low variability of features in a domain, such as rental and finance domain.

References

[1] Clements, P. and Northrop, L., *Software Product Lines: Practices and Patterns*, Addison Wesley, 2001.

[2] Weiss, D. and Lai, C., *Software Product-Line Engineering*, Addison-Wesley, 1999.

[3] Kyo C. Kang, Jaejoon Lee, and Donohoe, P., "Feature-Oriented Product Line Engineering," *IEEE Software*, Vol. 9, No. 4, pp. 58-65, Jul./Aug. 2002.

[4] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., and DeBaud, J., "PuLSE: A Methodology to Develop Software Product Lines," *Proceeding of symposium for Software Reusability '99*, ACM, pp. 122-131, 1999.

[5] Atkinson, C., et al., *Component-based Product Line Engineering with UML*, Addison Wesley, 2001.

[6] Bass, L., Klein, M., Bachmann, F., "Quality Attribute Design Primitives and the Attribute Driven Design Method," *Lecture Notes in Computer Science 2290, Proceedings of the PFE-4*, pp. 169-186, 2002.

[7] Bosch, J. *Design and Use of Software Architectures*, Addison-Wesley, 2000.

[8] Matinlassi, M., Niemela, E., and Dobrica, L., *Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture*, VTT Technical Research Center of Finland, 2002.

[9] Sinnema, M, Deelstra, S., Nijhuis, J., Bosch, J., "COVAMOF: A Framework for Modeling Variability in Software Product Families," *Lecture Notes in Computer Science 3154, Proceedings of the 3rd Software Product Line* Conference, pp. 197-213, 2004.

[10] Clements, P., et al., *Documenting Software Architectures Views and Beyond*, Addison-Wesley, 2003.

[11] Bass, L., Clements, P., Kazman, R, *Software Architecture in Practice*, Addison-Wesley, 2003.

[12] Clements, P., Kazman, R., and Klein, M., *Evaluating Software Architectures*, Addison Wesley, 2002.

[13] Kim, S., Her, J., and Chang, S., "A theoretical foundation of variability, in component-based development," *Information and Software Technology*, Vol. 47, p.663-673, July, 2005.

[14] Choi, S., Chang, S, and Kim, S., "A Systematic Methodology for Developing Component Frameworks," *Lecture Notes in Computer Science 2984, Proceedings of the 7th Fundamental Approaches to Software Engineering Conference*, pp. 359-373, 2004.

[15] Gomma, H., *Designing Software Product Lines with UML from Use Cass to Pattern-Based Software Architectures*, Addison-Wesley, 2004.

라 현 정

정보과학회논문지 : 소프트웨어 및 응용 제 33 권 제 5 호 참조

김 수 동

정보과학회논문지 : 소프트웨어 및 응용 제 33 권 제 1 호 참조