Analysis of Software Product Line Architecture Representation Mechanisms

Hwi Ahn
College of Information Science and Technology
KAIST
Daejeon, Republic of Korea
ahnhwi@kaist.ac.kr

Sungwon Kang
Department of Computer Science
KAIST
Daejeon, Republic of Korea
sungwon.kang@kaist.ac.kr

***Abstract*—Representing commonality and variability in the software product line architectures requires extension of the existing representation mechanisms (RMs). Various RMs including Orthogonal Variability Model (OVM) or extensions of the UML notation have been proposed by the well-known software product line (SPL) engineering methods. However, there is no research that analyzes to what extent they can express commonality and variability. This paper conducts an analysis of two representative RMs for the SPL architecture. To that end, first, it surveys various SPL engineering methods and identifies the two common types of RMs: the orthogonal RM type and the integrated RM type. Then it selects a well-known specific RM for each type: the mechanism of Pohl et al. and the KobrA mechanism. Then for analysis of the RMs various perspectives are derived from the concepts in *Orthogonal Variability Description Mechanism* [4]. Finally, the RM of Pohl et al. and the KobrA method are analyzed from these analysis perspectives. We believe that these results of the analysis provide essential guidance for improving existing RMs for SPL architecture or for defining new RMs for SPL architecture.**

Keywords-Software Product Line; Orthogonal Variability Description Mechanism; Orthgonal Variability Model; KobrA method; UML notation; Analysis

I. INTRODUCTION

The software product line (SPL) engineering is an approach to developing a family of software products based on the notion of common platform [1]. The SPL process consists of two phases: the domain engineering phase and the application engineering phase. The domain engineering phase produces a platform that contains commonality of the products as well as various assets which differ from product to product. As the domain engineering is a process that includes all stages of the conventional single product development, in the phase platform requirements are derived, platform architecture is designed, and platform implementation is realized. Then by applying bindings for products to the variability of the platform, requirements, architecture, and implementations for individual software products are produced. This approach often increases reuse dramatically and reduces time for developing software products significantly [2].

The SPL architecture reflects requirements of SPL and represents high level system design on which implementations of SPL are to be based. Unlike software architecture for single products, requirements of an SPL from which the SPL architecture should be design are the requirements of various products that constitute the SPL and define the scope of the SPL, which compels the SPL architecture to represent commonality as well as variability. Therefore, to represent commonality and variability syntactically and semantically, either new representation mechanisms (RMs) for the SPL architectures or extended version of the existing RMs for single product architectures are needed.

Among the various RMs suggested by the many SPL mechanisms for representing commonality and variability, the RMs that extend the UML notation or the feature model includes the works in [5, 6, 7]. They assign new meanings to the existing notations to support variability or use the existing extendible concepts of the traditional RMs. For example, the KobrA mechanism defines a new stereotype `<<variant>>`. Others suggested new RMs [1, 4]. Such new RMs are typically independent from the traditional RMs for single product architectures but are integrated with them orthogonally to represent variability. For example, Pohl et al. suggest a variability diagram that represents variability only and integrates it with the traditional RMs to represent SPL architecture.

Studying RMs is important because RMs reflect their developers' views on the notions of commonality and variability. If various RMs are analyzed with appropriate structured perspectives, we can find strong points and weak points of those RMs and can come up with a guidance to improve on them.

In this paper, based on various concepts for representing variability from *Orthogonal Variability Description Mechanism* (OVDM) [4], we derive perspectives for analysis. Then the two major RMs, i.e. that of Pohl et al.[1] and that of KobrA [5], are analyzed based on the perspectives. These two RMs represent the two ways of representing commonality and variability as the former uses a new and independent RM and the latter uses an extension of the UML notation.

The remainder of the paper is organized as follows: In Section II, related works are discussed. It contains the survey of RMs for the architecture of SPL and other related studies. In Section III, analysis perspectives are proposed. In Section IV, mechanisms to be analyzed are introduced. In Section V, analysis results are presented. Finally, Section VI is the conclusion.

II. RELATED WORKS

A. Scope of the Research

This research focuses on RMs for SPL architecture. The general SPL process we envisage is shown in Figure 1

together with the associated artifacts. The starting point of the SPL process is requirements (Reqs) for the platform. Platform requirements and platform design are linked by traceability (TR1). With TR1 and design decisions (D), platform design is derived. Similarly, platform implementation is derived from TR2, which links platform design and platform implementation, and implementation decisions. From platform artifacts, product artifacts are derived by applications of bindings (Bs), where bindings define which variants are associated with which variation points of the platform artifacts. Among the artifacts in Figure 1, $\text{Design}_{\text{platform}}$ denotes SPL platform architecture.

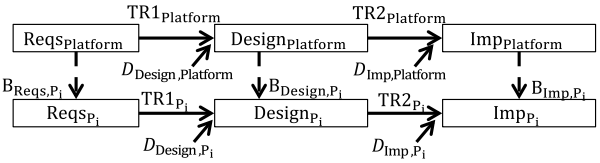


Figure 1. The General SPL Process and its Artifacts

Binding is a major concern of this paper based on the definition of the SPL architecture. Taylor et al. [3] said, "In general, product-line architectures capture the complete architectures of multiple related products simultaneously. Conversely, reference architectures can be incomplete or partial." (p. 594). Based on this definition, they distinguished the SPL architecture with a reference architecture, which defines a general solution for a specific domain of products [3] by two differences: scope and completeness of architecture. In terms of scope, the SPL architecture describes an explicit product family rather than describing a lot of diverse solutions within the specific domain. In terms of completeness, the SPL architecture should represent complete architectures for products in the family rather than remaining some parts undefined. Therefore, the SPL architecture should capture not only SPL platform architecture but also architectural binding for products.

B. SPL Methods

Until recently, various methods have been introduced for solving various issues related to SPL. RM is a part of these methods and addresses issues related to commonality and variability of SPL. Although RM is important as explained in Section I, mechanisms and surveys that explicitly focus on RMs are rare. Therefore, for the analysis in this paper we extract from the existing methods the RMs for SPL architecture among their various aspects for SPL engineering. To do this, first we survey the existing methods, following the steps below:

Step 1-a) Find methods whose reference count is greater than 200.

Step 1-b) Examine survey papers on SPL methods.

Step 2) Select methods that contain RMs.

Step 3) Identify major methods on which various other methods are based and group these methods as variations of the major methods.

We proceed with both Step 1-a and Step 1-b simultaneously because Step 1-b complements Step 1-a. If We survey only the methods that have high reference count, it may not reflect recent methods.

In Step 2, we found approaches including RMs among results of Step 1-a. Finally, in Step 3, we identified major methods in Table II.

TABLE I. SURVEY PAPERS

Survey Paper	Brief Description
Chen et al. [9]	Survey of variability management approaches
Sinnema et al. [12]	Survey of variability modeling techniques
Schobbens et al. [14]	Survey of various branches of [7] to derive generic semantics of the feature model

TABLE II. MAJOR METHODS INCLUDING RMS

RM for SPL	Brief Description
Feature Model [7,10]	Architectural models relevant to the feature model of a domain analysis
Pohl et al. [1]	Orthogonal Variability Model
KobrA [5,11]	An extension of UML.
Gomaa et al. [6]	An extension of UML.
Bachmann et al. [12]	An orthogonal variability modeling.

In Step 1-b, we examined surveys in Table 1 to get overall trends of methods for SPL. Chen et al. [9] surveyed 36 variability management approaches and classified them based on two criteria: key issues and hierarchical inheritance and 20 of these 36 approaches were classified as variability modeling. In Step 2, we found 13 approaches that considered variability representation among these 20 approaches. In Step 3, we identified Feature Model [7], Koalish [15] and Kumbang [16], KobrA [17], and COVAMOF [18] as major approaches.

In addition to Chen et al. [9], we examined two interesting surveys: Sinnema et al. [12] and Schobbens et al. [14] surveyed overall variability modeling techniques with focus that is broader and abstract than a RM. Also, it didn't cover representing commonality, which is the core of SPL artifacts.

Schobbens et al. investigated several variations of Feature Model [7]. It provided formal and generic semantics of a feature model, that can be applies to all variations of the feature model.

During our survey, we discovered that the feature model, which has many variations [9], was closer to the representation of a requirements engineering phase than the representation of an architecture phase. The feature model did not show architectural implication of representing variability and commonality.

RMs in methods, which are results of this survey, can be classified into two groups: integrated variability RM and orthogonal variability RM [9]. An integrated variability RM is an extension of existing RM for software architecture. As such RMs there are extension of UML [5,6] and formal ADLs (Architecture Description Languages) [16] to support SPL architecture.

On the other hand, an orthogonal variability RM explicitly divides a variability model from other models such as architectural models, requirements models, and so forth. This RM can then be used from a requirements engineering phase to an implementation phase because it is not bounded to a specific artifact. Pohl et al. [1] and Bachmann et al. [12] provide this kind of RM.

For this research, we select two mechanisms from this classification. First, we select the KobrA mechanism among the integrated variability RMs because KobrA is one of major methods to support SPL and provides precise definition of their RM and good practices. Second, we select Pohl et al. among orthogonal variability RMs as it includes a meta-model that precisely defines their RM and good examples of applying the RM.

III. The Analysis Perspectives

To analyze the two RMs with respect to SPL architecture, we use the concepts in OVDM for analysis perspectives. In this section, first, the concepts of OVDM are introduced in Sections III.A and III.B. Then, the analysis perspectives are presented in Sections III.C and III.D.

A. Orthogonal Variabiltiy Description Mechanism

OVDM was introduced in [4]. It defines an RM for representing commonality and variability. Although the name only mentions 'variability', it also covers commonality. OVDM is used to produce a *Product Family Modeling Language* (PFML) from a given *Product Modeling Language* (PML). Typically one would use for PMLs existing languages for modeling single software products. The other mechanism that goes with OVDM is *Orthogonal Commonality Description Mechanism* (OCDM), which is a mechanism for representing commonality in artifacts of SPL. OVDM and OCDM are concepts that can be applied to produce RMs for representing various artifacts such as requirements, architecture, implementation, and so forth at the product family level. However, in this paper we limit our discussion to SPL architecture. Figure 2 shows the OVDM's view on the relationship between the two kinds of RMs, i.e. one for commonality and the other for variability.

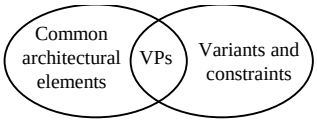


Figure 2. Relationship between an RM for Commonality and an RM for Variability

An RM for commonality consists of *variation point*s (VPs) and common architectural elements where common architectural elements are obtained by extending the semantics of traditional architectural elements from architectural elements of a single product to architectural elements of all products of the product family. Therefore, an RM for commonality can be defined as:

RM for Commonality = {VPs, common architectural elements}

An RM for variability consists of VPs, variants, and constraints. A VP is a place where variation occurs and a variant is an instance of the variation. A set of restrictions on VPs and variants (such as `requires` and `excludes`) constitutes constraints. Therefore, an RM for variability can be defined as:

RM for Variability = {VPs, variants, constraints}

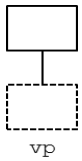
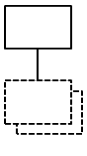
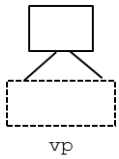
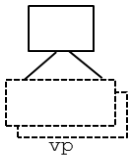
OVDM defines three variability types: `optional`, `inclusive_or`, and `exclusive_or`. `Optional` variability means that a variant can be selected or not for a VP. In the case of `inclusive_or` variability the number of selected variants for a VP can vary. In the case of `exclusive_or` variability only one variant can be selected for a VP among its variants.

B. Further Discussion on VP

VP belongs to the RM for commonality as well as the RM for variability. In contrast with common architectural elements, variants, and constraints that are inherited from the RMs for single products, VP is a newly introduced element of RMs for product families.

OVDM provides a classification of VPs depending on its variants and variability. The four different types of VPs are exemplified in Table III.

TABLE III. Classification of VPs

Variability \ Variants	Element VP	Collection VP
Simple VP		
Complex VP		

In Table III, the row is related to the number of places where variability occurs. If variation occurs at a single place, this place is a simple VP. On the other hand, if variability occurs at two or more places simultaneously, this is a complex VP. In both the examples in the Complex VP row, a VP contains two places of simultaneous variation.

The column of Table III is related to the number of variants that can be selected at once. If one variant can be selected at once, it is an element type VP. If a set of architectural elements can be selected at once as a single variant, it is a collection type VP. For example, if there is a car whose engine can be a gasoline engine, diesel engine, or yet another type of engine and only one type of engine can be selected, the engine is an element VP. However, if the car can select two different types of an engine at once to change them depending on situation, then the engine of this car makes a collection type VP.

This classification of VPs reflects a deep understanding of the notion of VP itself and gives a guidance when defining new RMs or improving existing RMs for modeling SPL architecture.

TABLE IV. THE ANALYSIS TABLE

RM for SPL Architecture / Analysis Perspectives			*Name of an RM*		
			Existence of Concept	Concept Modeling	Re-mark
Architectural Design	CM	**Inclusion of VP**			
		Construction of commonality expressing notation			
	VP	**Simple Element Type**			
		Simple Collection Type			
		Complex Element Type			
		Complex Collection Type			
	VM	**VM Type**			
		VM Style			
Architectural Binding to Product					

C. Analysis Perspectives

Based on the concepts of OVDM, analysis perspectives are derived. Analysis perspectives consist of the three categories: CM (Commonality Model), VP, and VM (Variability Model). Through these perspectives, we point out what the existing RMs are missing and make further discussion points. We think that this will help make sound and effective RMs. In the rest of this sub-section, explanation of each analysis perspective follows.

CM category is considered because RM for commonality is, together with RM for variability, one of the two axes to build the RM for SPL architecture as mentioned in Sections III.A and III.B. In CM category, there are two perspectives: *Inclusion of VP* and *Construction of commonality expressing notation*. *Inclusion of VP* shows if the SPL architecture can be clearly represented without causing confusion on commonality and variability by including VPs in a commonality model explicitly and regarding them as connection points to the variability model. *Construction of commonality expressing notation* checks if there are explicit semantics for common architectural elements.[1]

VP category is considered as a major part of the RM for SPL architecture because it differentiates the SPL architecture from single software product architecture and connects a commonality model with a variability model. VP category consists of *Simple Element Type*, *Simple Collection Type*, *Complex Element Type*, and *Complex Collection Type* reflecting the four types that are explained in Section III.B. These four analysis perspectives together can show how an RM can adequately represent VPs.

Like the importance of CM, VM category is important because VM shows how an RM can represent variability in SPL architecture. *VM Type* shows whether the VM of an RM for SPL architecture is orthogonal or integrated to the traditional RM. On the other hand, *VM Style* shows whether the model is based on a graphical RM or a tabular RM for representing variability. Depending on VM Type and VM Style, representation way of VPs and variants in VM is determined.

[1] As we explained above, we emphasize the commonality model since most RMs didn't provide this model explicitly in spite of its importance. Explicitly defining the commonality model is the first step to embrace a variability model into SPL architecture.

D. Answer Space for Each Column of the Analysis Table

For each perspective in Table VI, there are two aspects to analyze: *Existence of Concept* and *Concept Modeling*.

When analyzing each perspective in Table IV, *Existence of Concept* aspect shows existence of each perspective and how well an RM supports this perspective, if the RM supports it.

Concept Modeling aspect shows a concrete model where the RM supports a corresponding analysis perspective. Without this aspect, we cannot know if the analysis perspective of an RM is defined just conceptually or represented concretely in a specific model.

Finally, *Remarks*, which is the third column of Table IV, shows additional explanations that cannot be shown by Existence of Concept and Concept Modeling columns.

TABLE V. GENERAL ANSWER SPACE FOR EACH ASPECT

Aspect	Answer space	Description
Existence of Concept	Explicit	Defined or mentioned clearly
	Implicit	Not defined, but possible to represent
	No	Not defined, and not possible to represent
Concept Modeling	Names of a model	A model in which analysis perspective is represented
Remarks	Explanations or examples	Detail explanations

Table V shows the possible answers for each aspect to analyze each perspective in Table IV. Existence of Concept aspect has three possible answers: Explicit, Implicit, and No.

Explicit means that the RM defines the corresponding analysis perspective and has a notation for the perspective.

Implicit means that the corresponding analysis perspective is not defined, but it is implicitly supported by the RM. For example, although a RM does not mention *Complex Element Type*, this type can be expressed by notations provided by this RM. In this case, this RM supports the analysis perspective implicitly.

No means that the RM does not define the relevant analysis perspective and it is not possible to represent it.

The answer of *Concept Modeling* aspect is the name of a model in which relevant analysis perspective is represented. For example, the mechanism of Pohl et al. represents commonality with VPs in a component framework. Then an answer for the *Model* column is '*component framework*'. In the *Description* column, detailed information and explanation about the corresponding analysis perspective is written.

In the case of VM type, VM style, and architectural binding to Product, answer spaces for Existence of Concept are different from those in Table V. Table VI shows them. For *Existence of Concept* for *VM type*, there are two answers: *orthogonal* and *integrated*.

TABLE VI. ANSWER SPACES FOR EXISTENCE OF CONCEPT FOR VM TYPE, VM STYLE, AND ARCHITECTURAL BINDING TO PRODUCT

Analysis Perspective	Answer space	Description
VM type	Orthogonal	Used through all architectural artifacts.
	Integrated	Included in traditional architectural artifacts.
VM style	Graphical	Be a graphical diagram
	Tabular	Consist of tables and textual notations.
Architectural Binding to Product	Precisely defined	Define binding for each planned product.
	Loosely defined	Mention only concept of binding.
	No	Nothing about binding.

Orthogonal VM is a VM that is used through all architectural artifacts. It provides a separate diagram (or model) for representing variability and maps it to traditional architectural artifacts. On the other hand, *Integrated* VM extends existing RMs by assigning new or extended semantics.

Existence of Concept for *VM style* is *graphical* or *tabular*. *Graphical* means that a RM provides graphical notations. On the other hand, *Tabular* means a form of tables and textual notations.

Architectural Binding to Product has three possible answers: *precisely defined*, *loosely defined*, and *no*. *Precisely defined* means it provides explicit mechanisms to represent architectural bindings for each planned product. On the other hand, *loosely defined* means that RM does not provide explicit mechanisms although it has the concept of the architectural binding to products. Finally, *no* means that the notion of binding is not addressed.

IV. ANALYSIS OF SOFTWARE PRODUCT LINE ARCHITECTURE REPRESENTATION MECHANISMS

A. The Representation Mechanism of Pohl et al.

The mechanism of Pohl et al. introduced in [1] provides a guidance to build SPL with respect to the overall process and documentation. It covers domain engineering and application engineering from the requirements phase to the implementation phase. It emphasizes management of variability throughout the overall process. Therefore, OVM, which is a RM of Pohl et al. for documenting commonality and variability, supports domain engineering process orthogonally.

OVM provides a special diagram for documenting variability called a *variability diagram*. Elements of the variability diagram are VPs, variants, and constraints. They are mapped to elements of all kinds of artifacts produced during software development. For example, an OVM diagram can be mapped to a feature model, a list of requirements, architecture diagrams, and so forth. Figure 3 shows an example.

Pohl et al. also suggest a component framework to represent commonality. OVM defines a component framework that is a set of components and plug-in locations, which are same with VPs [1].

B. The KobrA Representation Mechanism

The KobrA was introduced in 2002 by Fraunhofer IESE. It is a follow-up method of Product Line Software Engineering (PuLSE™)., Introduced in 1999, PuLSE is a process-centric method like Family-Oriented Abstraction, Specification, and Translation (FAST), which was introduced by D. Weiss et al. in 1999 [8] for deploying SPL in various enterprise contexts [11]. Therefore, PuLSE™ did not provide concrete guidance from technical perspectives unlike FAST and FODA [7].

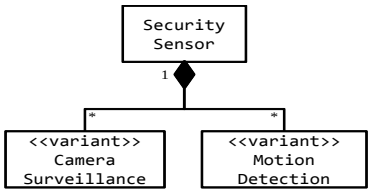


Figure 3. A Class Diagram for Security Sensor

KobrA was derived from this problem of PuLSE™. This is a ready-to-use customization of PuLSE™ [11]. Therefore, KobrA provides detailed techniques from initializing SPL to documenting variability and commonality.

For documenting variability and commonality, KobrA extends UML notation by assigning new semantics to stereotype `<<variant>>`, which is interpreted as a VP or a variant depending on the relevant decision model.

In KobrA, decision model is used for complementing insufficient graphical notation. Decision model defines VPs, constraints, and resolutions. Resolution denotes the same notion as binding.

ID	Question	Variation Point	Resolution	Effect
1	Class Camera-Surveillance needed?	class Camera-Surveillance	yes no	remove stereotype <<variant>> resolve decision 2: no remove class CameraSurveillance resolve decision 2: yes
2	Class Motion-Detection needed?	class Motion-Detection	yes no	remove stereotype <<variant>> resolve decision 1: no remove class MotionDetection resolve decision 1: yes

Figure 5. A Decision Model for Figure 4

Figure 4 shows that there are `SecuritySensor` consisting of two variants. However, this graphical representation does not indicate what variability exists between the two variants, for example, whether it is `optional` or `exclusive_or`. Therefore, a decision model, which is a table including variability, is used. Figure 5 is a decision model for Figure 4 and shows a relationship between `CameraSurveillance` class and `MotionDetection` class is `exclusive_or`.

V. ANALYSIS RESULTS

A. The Representation Mechanism of Pohl et al.

Table VII shows the results of analyzing the mechanism of Pohl et al. In terms of *Inclusion of VP* analysis perspective

included in CM category of table VII, Pohl et al. define the CM explicitly using a component framework, which includes VPs as plug-in locations, which are represented by dotted lines. In addition, Pohl et al. show *Construction of commonality expressing notation* analysis perspective by assigning common semantics to traditional architectural notations. Figure 6 shows an example of a component framework.

In Figure 6, `User Control Manager` component, `Authentication Manager` component and their interfaces are common. `User Control Plug-in` component, `Authentication Plug-in` component, and an interface of `Authentication Plug-in` are VPs.

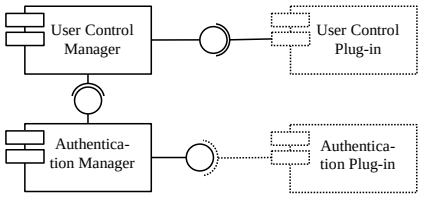


Figure 6. An Example of a Component Framework [1]

With respect to four analysis perspectives included in VP category of Table VII, Pohl et al. can explicitly represent a simple element type and a simple collection type. Pohl et al. explicitly define three kinds of variability. So it can support simple collection type and complex collection type by combinations of these three kinds of variability: `exclusive_or`, `inclusive_or`, and `optional`. Figure 7 shows simple collection type VP.

This representation shows that {`Camera Surveillance`, `Cullet Detection`}, {`Motion Sensors`, `Cullet Detection`}, {`Camera Surveillance`}, and {`Motion Sensors`} are variants for this VP in the mechanism of Pohl et al. There is no indirect way to represent this simple collection type and all notations are used as they are intended. Therefore, the mechanism of Pohl et al. can represent simple collection type explicitly.

However, in the case of complex element type and complex collection type, the mechanism of Pohl et al. can only represent them implicitly by using constraints among VPs and variants. It is because there is no notation for a set of VPs, which is necessary to represent complex element type and complex collection type.

Although there is no notation for a set of VPs, the mechanism of Pohl et al. can represent a collection type by using constraints among VPs and variants.

VM type analysis perspective included in VM category of Table VII is orthogonal. OVM, which is a variability model of Pohl et al., can be used through all artifacts of SPL architecture without changing any notation.

VM style analysis perspective included in VM category is graphical. Notations using in VM of Pohl et al. only include graphical notations.

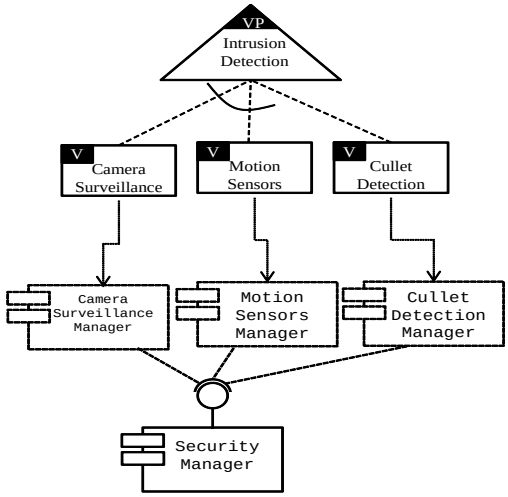


Figure 7. A Simple Collection Type VP

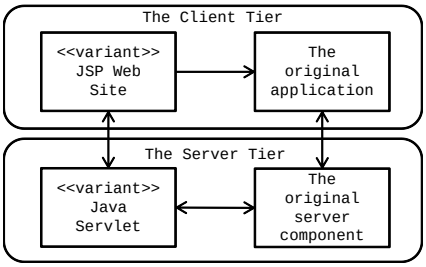
Concerning the *Architectural Binding to Product* analysis perspective, Pohl et al. loosely define it because there is no representation mechanism for representing products that will be produced by SPL although it explains what the architectural binding to products is and how it will be achieved.

B. The KobrA Representation Mechanism

Table VIII shows the results of analyzing the KobrA mechanism. In terms of CM category of table VIII, the KobrA mechanism defines the CM by using the UML user defined stereotype `<<variant>>`. As explained in Section IV.B, this stereotype can be interpreted as a VP or a variant depending on the decision table. Therefore, *Inclusion of VP* analysis perspective is implicitly mentioned in the KobrA mechanism like `<<variant>>` elements are implicitly regarded as VPs.

According to Muthig et al.[17], *Construction of commonality expressing notation* analysis perspective is explicit because semantics of all elements used in UML models of KobrA are explicitly extended to common meanings.

With respect to four analysis perspectives included in VP category of Table VIII, KobrA represents four types of VPs by using a decision model. In the decision model, each VP is resolved by two choices: `yes` or `no`. Effects of each resolution are written in informal languages. For example, in Figure 5, if the `CameraSurveillance` class is resolved as `yes`, then `<<variant>>` stereotype on it is deleted and a decision whose ID is 2 is resolved to `no`.



(a) A Web Application

ID	Question	Variation Point	Resolution	Effect
1	Does the system support a web application?	JSP Web Site, Java Servlet	yes no	resolve decision 2: yes resolve decision 3: yes resolve decision 2: no resolve decision 3: no
2	Is JSP Web Site needed?	JSP Web Site	yes no	remove stereotype <<variant>> remove JSP Web Site
3	Is Java Servlet needed?	Java Servlet	yes no	remove stereotype <<variant>> remove Java Servlet

(b) Decision Model for the Web Application in (a)

Figure 8. A Web Application and its Decision Table

ID	Question	Variation Point	Resolution	Effect
1	Does the intrusion system have camera surveillance and cullet detection?	CameraServeillance, CulletDetection	yes no	remove <<variant>> from CameraSurveillance and CulletDetection remove MotionDetection resolve decision 2: yes or resolve decision 3: yes or resolve decision 4: yes
2	Does the intrusion system have motion detection and cullet detection?	MotionDetection, CulletDetection	yes no	remove <<variant>> from MotionDetection and CulletDetection remove CameraServeillance resolve decision 1: yes or resolve decision 3: yes or resolve decision 4: yes
3	Does the intrusion system have only camera surveillance?	CameraServeillance	yes no	remove <<variant>> from CameraServeillance remove MotionDetection and CulletDetection resolve decision 1: yes or resolve decision 2: yes or resolve decision 4: yes
4	Does the intrusion system have only motion detection?	MotionDetection	yes no	remove <<variant>> from MotionDetection remove CameraServeillance and CulletDetection resolve decision 1: yes or resolve decision 2: yes or resolve decision 3: yes

Figure 9. The Decision Model for Figure 7

With KobrA, simple element type VP can explicitly be represented because the decision model includes a VP, which indicates a place where variability occurs. In addition, each variant of this VP is a single architectural element.

Complex element type VP can be implicitly represented as in Figure 8. In Figure 8, the decision 1 implies a complex element VP.

However, it is very hard to represent simple collection type and complex collection type VP with KobrA because there is only two ways to resolve a VP. Therefore, representing a set of architectural elements as a variant should be done in the Effect column of a decision table. It requires lots of complex representations written by natural languages. Figure 9 shows a representation of a collection VP. Although complex, representing a collection VP by KobrA is possible. Therefore, answers of *Existence of Concept* column for *Simple Collection Type* and *Complex Collection Type* of Table VIII are *Implicit*.

VM type analysis perspective included in VM category of Table VIII is *Orthogonal* because the decision model is used through all processes of SPL. In addition, *VM style* analysis perspective included in VM category of Table VIII is Tabular as Figures 8 and 9 show,

With respect to *Architectural Binding to Products* analysis perspective of Table VIII, KobrA is loosely defined. Although KobrA defines a set of products at the beginning of domain engineering, it does not provide an explicit RM for SPL architecture.

VI. Conclusion

This paper analyzed the two RMs for SPL architecture, that of Pohl et al. and that of KobrA. For this analysis, we first derived analysis perspectives from the various concepts in OVDM [4]. With these perspectives, we could arrive at several findings.

In the perspectives of CM category, Pohl et al. allow VPs and common architectural elements to be explicitly expressed in its CM. On the other hand, the definition of VP in the CM of KobrA is implicit and it is difficult to separate out VM from CM because VPs in the CM are regarded as VPs or variants depending on its context.

In terms of VP of Table IV, the complex element type and the complex collection type of VPs are not explicitly supported by the two analyzed RMs. It often forces designers to use notations in undefined new ways to express them.

With respect to VM of Table IV, both OVM and KobrA maintain it orthogonally. They use separate VMs, which can be used with all kinds of architectural artifacts without changing their notations. However, their VM styles are different. While the mechanism of Pohl et al. uses graphical notations, KobrA uses a tabular notation.

In terms of architectural binding to product analysis perspective, both define their architectural bindings loosely. Although both define what architectural binding is and how architectural binding can be done, there is no representation mechanism that can indicate which variant belongs to which product.

We believe that these results of the analysis provide essential guidance for improving existing RMs for SPL architecture or for defining new RMs for SPL architecture.

For future work, we will extend this analysis research to other well-known SPL mechanisms including Bachmann et al. [12] and Gomaa's mechanism[6]. By extending target mechanisms to analyze, we can gain a better insight into a good representation mechanism for SPL architecture.

TABLE VII. THE SUMMARY RESULTS OF ANALZING THE MECHANISM OF POHL ET AL.

RM for SPL Architecture / Analysis Perspectives			*The mechanism of Pohl et al. (OVM)*		
			Existence of Concept	**Concept Modeling**	**Remark**
Architectural Design	**CM**	**Inclusion of VP**	Explicit	Component framework	Plug-in locations
		Construction of commonality expressing notation	Explicit	Component framework	Defined as a structure of common components; using traditional notations for common semantics
	VP	**Simple Element Type**	Explicit	Orthogonal Variability Model	Notations for representing a Simple type ofVPs are explicitly defined.
		Simple Collection Type	Explicit	Orthogonal Variability Model	Three kinds of variabiltiy can explicitly be represented.
		Complex Element Type	Implicit	Orthogonal Variability Model	No notation for representing a set of VPs
		Complex Collection Type	Implicit	Orthogonal Variability Model	No notation for representing a set of VPs
	VM	**VM Type**	Orthogonal	Orthogonal Variability Model	Mapped to all kinds of software artifacts.
		VM Style	Graphical	Orthogonal Variability Model	Only providing graphical notations.
Architectural Binding to Product			Loosely defined	Orthogonal Variability Model	Defining the concept of the binding, but not for each planned products.

TABLE VIII. THE SUMMARY RESULTS OF ANALZING THE KOBRA MECHANISM

RM for SPL Architecture / Analysis Perspectives			*The KobrA Mechanism*		
			Existence of Concept	**Concept Modeling**	**Remark**
Architectural Design	**CM**	**Inclusion of VP**	Implict	UML models	Not mention VP in the CM explicitly; `<<variant>>` in UML models regarded as VP in the decision model.
		Construction of commonality expressing notation	Explicit	UML models	Using traditional notations for common semantics; clearly mentioned in [17].
	VP	**Simple Element Type**	Explicit	The decision model	Notations for resolution, variation point, and effect column of a decision model support a Simple Element type of VPs.
		Simple Collection Type	Implicit	The decision model	Not defined explicitly
		Complex Element Type	Implicit	The decision model	Not defined explicitly.
		Complex Collection Type	Implicit	The decision model	Not defined explicitly
	VM	**VM Type**	Orthogonal	The decision model	Used through all phases of the KobrA mechanism
		VM Style	Tabular	The decision model	Represented as a table form
Architectural Binding to Product			Loosely defined	The decision model	Defining the concept of the binding, but not for each planned products.

ACKNOWLEDGEMENT. This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (20100022431).

REFERENCES

[1] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*, Springer, 2005.

[2] SEI, A Framework for Software Product Line Practice: Version 5.0

[3] R. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*, Wiley, 2009.

[4] S. Kang, "A Method for Extending Software Modeling Languages to Languages for Modeling Families of Software," JSE, 2010.

[5] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wust, and J. Zettel, *Component-based Product Line Engineering with UML*, Addison-Wesley, 2002.

[6] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*, Addison-Wesley, 2004.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Tech Report CMU/SEI-90-TR-21, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1990.

[8] D. Weiss and C. Lai, *Software Product-Line Engineering: A Family-Based Software Development Process*, Addison-Wesley, 1999.

[9] L. Chen, M. A. Babar, and N. Ali, "Variability Management in Software Product Lines: A Systematic Review," SPLC 2009, 2009.

[10] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A feature-oriented reuse method with domainspecific reference architectures," Annals of Software Engineering, vol. 5, pp. 143-168, 1998.

[11] J Bayer, O Flege, P Knauber, R Laqua, D Muthig, and K Schmid, "PuLSEe: a methodology to develop software product lines," Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability, 1999.

[12] F. Bachmann, M. Goedicke, J. Leite, R. Nord, K. Pohl, B. Ramesh, and A. Vilbig, "A Meta-model for Representing Variability in Product Family Development," Software Product-Family Eng (PFE-5): Springer, pp. 66-80, 2004.

[13] M. Sinnema and S. Deelstra, "Classifying variability modeling techniques," Information and Software Technology, vol. 49, pp. 717-739, 2007.

[14] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps, "Generic semantics of feature diagrams," Computer Networks, vol. 51, pp. 456-479, 2007.

[15] T. Asikainen, T. Soininen, and T. Männistö, "A Koala-Based Approach for Modelling and Deploying Configurable Software Product Families," Software Product-Family Eng (PFE-5): Springer, pp. 225-249, 2004.

[16] T. Asikainen, T. Männistö, and T. Soininen, "Kumbang: A domain ontology for modelling variability in software product families," Advanced Engineering Informatics, vol. 21, pp. 23-40, 2007.

[17] D. Muthig and C. Atkinson, "Model-Driven Product Line Architectures," Software Product Lines (SPLC2): Springer, pp. 79-90, 2002.

[18] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "A customizable approach to full lifecycle variability management," Science Computer Program., vol. 53, pp. 259-284, 2004.