Modeling Variability in the Component and Connector View of Architecture Using UML

Maryam Razavian, Ramtin Khosravi
Department of Electrical and Computer Engineering,
University of Tehran
m.razavian@ece.ut.ac.ir
rkhosravi@ece.ut.ac.ir

Abstract

Modeling variability is a key aspect of variability management in software product families. Product line architecture (PLA) is one of the major assets of a product line, from which individual product architectures are derived. Consequently, modeling variability in architecture becomes an issue worthy of consideration. In this paper, we propose a variability modeling method which is specifically devised for the component and connector (C&C) view of architecture. We use UML 2 as the architecture modeling language. Modeling solutions are proposed and classified based on the type of variable element and the techniques used to realize variability. We have also studied the ways to avoid cluttering the view when including variability. An example case is utilized to clarify different aspects of our proposed method.

1. Introduction

Software product line engineering deals with development of families of related products, while making use of commonalities and variabilities among them [8,19,11,2]. *Variability management* is about handling introduction, use, and evolution of variability [23]. *Variability modeling* methods represent the variabilities among products of a family in such a way that it provides better understanding of variabilities and also assists engineers in task of variability management.

Different variability modeling methods have been proposed by various product line approaches [1,6,9,15,16,19,21,22]. There are two aspects in variability modeling that are considered in existing methods: representing variability in software artifacts and modeling variabilities and their relationships in an *orthogonal variability model*. According to [7], a variability modeling method encompasses variability modeling within software artifacts while supporting an orthogonal variability model which includes information about decisions in the domain space and relationships among them (Figure 1). In this paper, we restrict our attention to modeling variability in software architecture (the shaded box in Figure 1). Note that an effective variability management is not achieved without considering the orthogonal variability model. Nevertheless, we leave the discussion on this issue to another work.

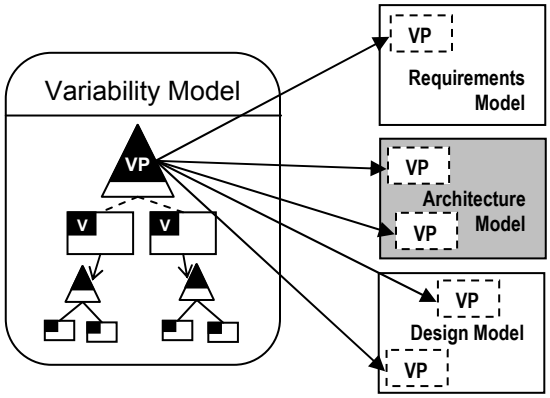

Figure 1: Dimensions of Variability Modeling

The variability should be captured in the architecture as it is reflected on other assets such as feature and use case models. As a result, the explicit representation of variability becomes an essential part of a product line architecture description and variability should be handled in every aspect of architecture design. To study the effect of variabilities on PLAs, the architect must be able to explicitly model variable elements in the architecture as well as how the variabilities are realized. To be more specific on the term "elements in the architecture", we must specify which view of architecture we are talking about, as architecture is known to be a complex entity represented in multiple views [10].

978-1-4244-1968-5/08/$25.00 ©2008 IEEE

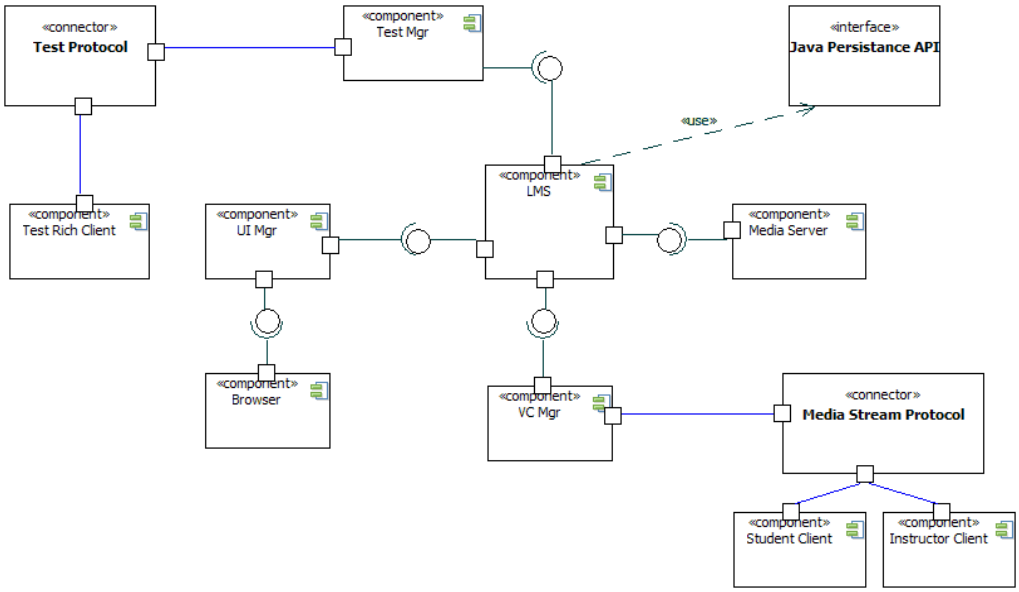

Figure 2: The Architecture of Virtual University System without Variabilities

In this work, we focus on the components and connector (C&C) view, as it is common to architecture description languages (ADLs) [1,25], as well as multiple view approaches [17,5,20].
Our approach to modeling variability in C&C has the following aspects:

- A *variability representation* method specified for this view based on UML 2 profile
- A *two-level architecture representation* method intended for abstracting away complexities
- Guidelines on how to represent individual product architectures

Our proposed method has two important characteristics. First, the method is based on UML 2 as the architecture modeling language. An obvious benefit is that the architect uses a standard language, widely used among practitioners. Also, this enables more precise modeling of variability concepts and the corresponding realization mechanisms. Moreover, it provides a basis for more detailed analysis of the effects of variabilities in the architecture.

The second characteristic of our method is that it focuses on architecture as a separate artifact. As a result the specific concerns associated to this artifact are addressed by the method and it allows a precise definition of variability in the architecture. Our method for modeling variability is categorized based on type of variable element (component, connector, etc.) and the type of variability (optional, alternative, etc.). In each case, we show how to model variability considering different realization mechanisms which can be used in that case.

Another problem to be addressed is how to avoid cluttering the architecture representation when including variable elements and their variant realizations. We propose a solution based on two-level representation of architecture, such that the main representation excludes variants, and the secondary diagrams model realizations of variabilities separately.

To serve as a basis for specific examples during the rest of the paper, we introduce an example system in section 2. Basic concepts about software architecture and variability modeling are defined in section 3. The proposed variability representation method is discussed in section 4, whereas section 5 presents the proposed method of abstracting complexities in architecture representation. Section 6 presents a solution for modeling product architecture. Our method is compared to the related work in section 7.

2. Example Case

We use an example to illustrate our method, which is taken from a case study we have performed. Several variabilities in this example are ignored for the sake of simplicity. The example involves a product family in the domain of e-learning systems for virtual universities. This system consists of different basic e-learning functions such as virtual classrooms and online assessments. Figure 2

illustrates a possible architecture for the system, without considering variabilities.

The *LMS* (Learning Management System) component contains the basic business logic of the system. There are various types of clients, such as the *web interface*, an *assessment client*, and a *virtual classroom* application, all connected to the *LMS*, which provides them with required business logic through a number of interfaces. The relationship between the *LMS* and the clients follows the common client/server style. In order to eliminate the dependency of *LMS* to the clients, a number of server plug-ins are devised to act as controllers of clients. *Test Mgr*, *UI Mgr* and *VC Mgr* are samples of the controller plug-ins. The *virtual class* plug-in in turn, communicates with *VC clients* through a client/server data streaming style. The data in the system are stored in a central database accessed by the *LMS* component.

The variabilities which occur at architectural level are depicted in Table 1.

Table 1: Variabilities of Virtual University PL

Variability	Description	Variants
Test Communication Type	Types of communication based on whether the Test Client and Test Mgr is Secure or not	Secure or non-secure protocol based on TCP
Presentation Style	UI may be JavaScript based or plain HTML	JavaScript enabled or basic HTML
Persistence Manager	The framework used for mapping objects to relations	Hibernate and TopLink
Virtual Class Video	The content of virtual class may contain video or not	Video based VC Mgr
Bandwidth	The compression degree used by Media Streaming Protocol	High and Low Bandwidth Media Stream protocol

3. Preliminaries

Although the notion of software architecture has been introduced for a while, there are different perspectives on the subject [5,18,17,20]. In one perspective, software architecture is considered as a single dimensional concept that is modeled precisely in an architecture description language (ADL). The second perspective defines architecture as a complex entity that is represented in multiple views. Various sets of views are defined in different approaches [17,10,20]. We base our work on the method proposed in [10]. Our focus in this paper is on the C&C view of architecture which could be considered as the intersection of the ADL and multiple view perspectives.

Component and connector views define models which consist of elements that have some runtime presence and also the interaction between the elements. In this view, components are system's units of execution and ports are places where the component interacts with its environment. The interactions among components are carried out with connectors. Components interact with each other in various ways such as invocation, event multicasts or more complex communication protocols. The way in which components use connectors to interact is defined by roles that can be considered as connectors interfaces.

We choose UML 2 as the notation for modeling C&C view of architecture in our method since it is a well-known and widely accepted notation for modeling software. Also, it is extensible by means of standard extension mechanisms. More importantly, UML 2 provides good support for architectural building blocks (components, connectors, ports, roles). The way to represent C&C views in UML 2 is discussed in [13]. Architectural components are modeled with UML 2 components and connectors are modeled as classes. We use a combination of UML ports and the corresponding provided and required interfaces to support C&C ports.

Before explaining details of the variability modeling method, we review the basic concepts involved in the method. Variability occurs at different levels of abstraction and has widespread impact on all software artifacts. Considering *variants* as product specific realizations of variabilities, *variation points* can be more precisely defined as places in the core assets where a choice between zero or more variants are made. Different types of variation points are as follows [22]:

- An optional variation point is the choice of selecting zero or one from the one or more variants.
- An alternative variation point is the choice between one of the one or more variants.
- An optional variant variation point is the selection (zero or more) from the one or more variants.
- An alternative variant variation point is the selection (one or more) from the one or more variants.

4. Representing Variability in C&C View

In this section we propose a solution for representing variability information in C&C view of the architecture. In our work, variability is realized

by having multiplicity in architectural elements. In other words, architectural variability is enabled by having optional or alternative architectural elements. In our view, variability is realized by the notion of variation point.

As variabilities may occur in assets of all levels of abstraction, each software artifact induces particular challenges for variability representation. Two issues should be considered in order to define a variability representation method for a specific asset:

- Types of variable elements in the asset
- Techniques used to realize the variabilities

Variability arises in three forms in C&C view: variable component, variable connector, and variable interface.

Realization techniques are means to be put into variability practice within different assets. The purpose of realization technique is to support the encapsulation of the way in which two or more variants differ from each other. These techniques have been studied and classified in [6,3,12]. Each realization technique is associated with particular stages of development lifecycle. At architectural level mostly inheritance and interface realization techniques are applicable. Parameterization is not taken into account as a realization technique at architectural level since it is directly related to the components implementation. Interface realization is a technique which is used when several versions of a component are built with the same interface. In this case, the variability is realized by variants which are different versions of a component and all adhere to the same interface. Inheritance technique is used when different variants share some features and are capable of being generalized to an abstract element.

To support variability within C&C view, we propose an extension to UML component diagrams by defining a UML profile. Our method proposes a solution for modeling variability in various situations which are formed based on the following three factors:

- Realization technique
- Variation point type (optional or alternative)
- Variable element type (component, connector or interface)

In other words, based on the variability mechanism used by the architect or type of variation point or based on the variable element (component, connector or interface), a specific solution for modeling variability is proposed. In the followings different possibilities are assessed and in each case a solution for modeling variability is offered. In each modeling solution, the alternative and optional variation points are marked with <<alt vp>> and << opt vp>> respectively.

4.1. Variation in Components

The architecture of each product of a software product line could be represented by different set of components. This implies that a specific feature could be realized by different components in family of software systems. Variability within components is realized by inheritance when the variant components share some common features and are capable of being generalized to an abstract component. In this case, the element which models a variation point is the abstract component. In the context of virtual university example, two variant components *JavaScript UI Mgr* and *Basic HTML UI Mgr* are generalized to *UI Mgr* component. The fact that these two variant components can participate in the product specific architecture exclusively makes the *UI Mgr* be an alternative variation point. Figure 3, depicts proposed modeling method in the virtual university example. In case contribution of component variants in the product is optional, the abstract component is marked by <<opt vp>> stereotype. In the case where an optional component does not contribute in the product specific architecture, the links and connectors associated to it will be also excluded from the architecture. The architect should take this case into consideration to avoid any problem in the architecture.

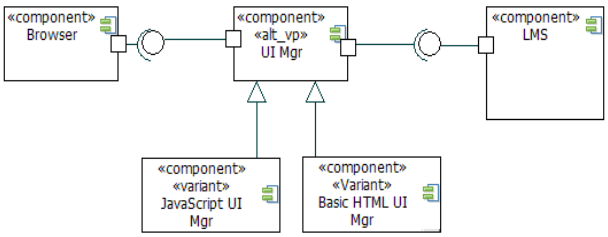

Figure 3: Variability in Presentation Style

When we are faced with versions of a component that all adhere to the same interface but have different implementation, variability is realized by interface realization. Since the interface encapsulates the place where the variability resides it could be recognized as the variation point. In the virtual university example both variant components *Hibernate* and *TopLink* realize the *Java Persistence API* while their implementations are different. Consequently, *Java Persistence API* is regarded as an alternative variation point (Figure 4).

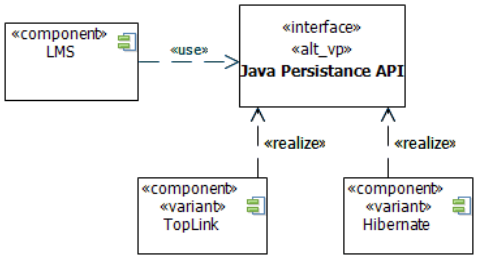

Figure 4: Variability in Persistent Manager

4.2. Variation in Connectors

The variation among two product architectures might take place because the connectors between the common components of both architectures are different. More precisely, the variation in connectors between the same components implies that they differ in the manner that they communicate, control, synchronize, use or invoke each other. This kind of variability is realized by inheritance when the connectors are capable of being generalized to an abstract connector. As mentioned in section 3, the architectural connectors are modeled with the UML class element [13]. In the virtual university example, two connector variants *HighBW Media Stream* and *LowBW Media Stream* participate in the product specific architectures exclusively. The connector variants are generalized to an abstract connector called *Media Stream* which encapsulates the variability and as a result is an alternative variation point (Figure 5).

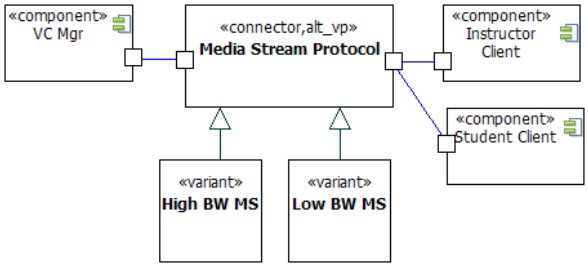

Figure 5: Variability in Bandwidth

Consider the case where two common components communicate via set of connectors which propose shared roles. In other words, the connectors adhere to a same interface. In this case the common interface encapsulates the variability within connectors. As a result, the variability could be realized by interface realization technique. The shared interface is the variation point and based on its type is marked with <<alt vp>> or <<opt vp>> stereotypes.

4.3. Variation in Interfaces

In some cases, different product architectures share a common component which offers different interfaces within the context of each individual product. The same situation could occur for the similar connectors in two product architecture. Although interfaces may be variable, but it is recommended to be avoided since mostly in these cases the variability will arise in the associated components or connectors as well. However if variability arises within interfaces the variation point element is modeled by UML notes and each variant is marked by <<variant>> stereotype.

4.4. Other Types of Variation Points

While we discussed different types of variation points and their realization in the previous sections, it should be noted that there is also a simpler and more basic case of variability within components, connectors or interfaces. This is the case where the variability arises due to the presence or absence of a single component, connector or interface in specific product architecture. Here that specific element (e.g. component) becomes an optional variation point and is marked with <<optional>> stereotype. In the context of virtual university, the optional feature of video in virtual class results in having an optional component *Video enabled VC Mgr* which is a specialization of *VC Mgr* (Figure 6).

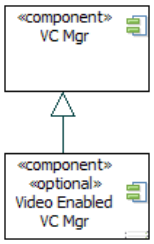

Figure 6: Variability in Virtual Class Video

Alternative and optional are just basic types of variation points. Other types of variation point including optional variant and alternative variant are not discussed yet. These types of variation points are modeled analogous to optional and alternative variation points respectively. Their semantic distinction is distinguishable by <<optv vp>> and <<altv vp>> stereotypes for optional variant and variant variation points respectively.

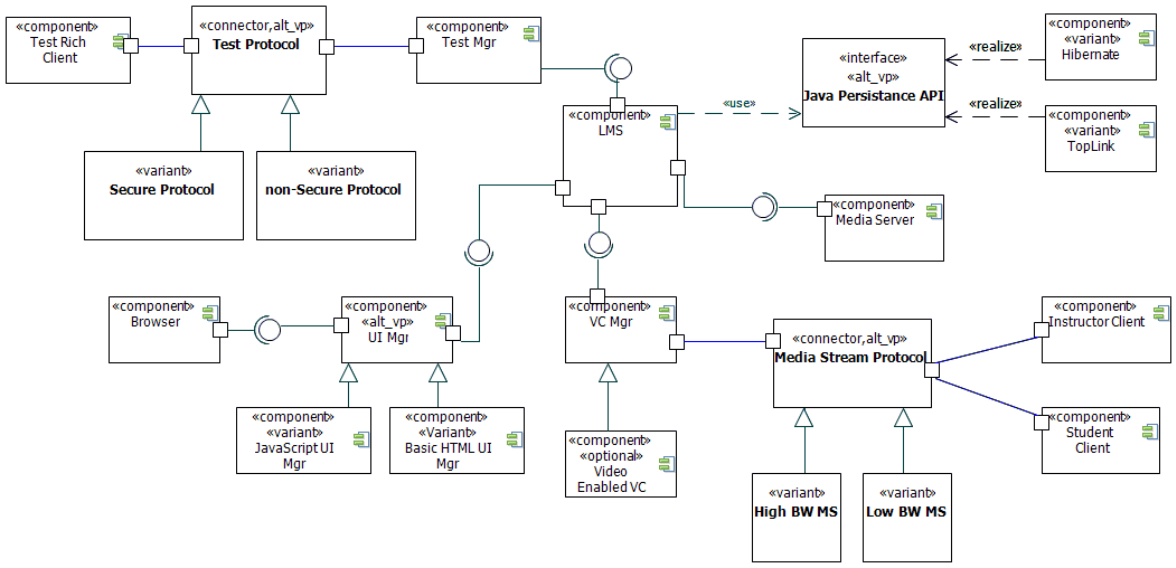

Figure 7: Detailed Representation of Virtual University PLA (Second Level)

5. Abstracting Complexities in Architecture Representation

One important goal of software architecture is to provide a big picture of the system by abstracting away the complexities and depicting the architecturally significant elements as well as the relationship between them. Variability defies this purpose by inducing complexities to the architecture representation and on the other hand, we believe, its representation at the architectural level is essential.

To solve this problem, we make use of the fact that architecture can be viewed at different levels of detail. We propose using a simplified model of the architecture at the first level that excludes variability details including all the variants, while on the other hand comprising variation points as they are represented by abstract components, abstract connectors or interfaces. In our example case, this will be a model similar to what is shown in Figure 2, except that the variable elements are marked with appropriate stereotypes such as <<alt vp>>.

At the next level of product line architecture, more detailed models of variability, including the variants will be presented. The issues discussed in the section 4 will be present in these levels.

There are two ways to represent variability at the second level:

- A single model which includes all variation points and their associated variants
- Separate models for realization of different variabilities

Apparently, the first way is about representing the architecture with all the related variability details (Figure 7). The second way is similar to what we do when organizing the design model as a collection of use case realization models in the Unified Approach [14]. Here, the architect organizes the architecture, along with details of variability realizations as a collection of models, each corresponding to one or more variabilities. For example, what we have presented in figures 3 to 6 are separate models related to different variabilities. In this case, a separate variability model can be used to capture all variabilities as well as their relationships. This plays a role analogous to the use case model. Since the variability model is not in the scope of this paper we refer it to the methods proposed in [7,19]. Figure 8 illustrates an example of representing the architecture with the orthogonal variability model (with notation proposed in [19]) and separate models. Of course, providing a variant-free view of the architecture is still needed to enable a cohesive view of the whole architecture.

6. Modeling Product Architectures

Product line engineering is composed of two phases: domain engineering and application engineering. Variability management plays an important role in both phases. Consequently, variability modeling should provide solutions for both phases.

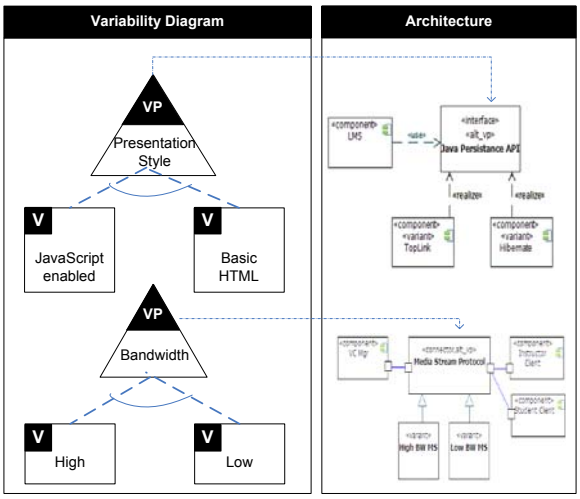

Figure 8: Linking Architecture Representation with an Orthogonal Variability Model

So far, we have proposed solutions for modeling architectural variability in the domain engineering phase. Deriving a product specific architecture from a product family architecture is an integral part of any variability modeling method [22], hence, we present some simple suggestions for that purpose in this section.

Regarding modeling the product specific architecture, in case a variant component or connector of a generalized element is selected, the generalized element is replaced by the variant element. If the variability in the product architecture is realized by interface realization the selected variant and the common interface will participate in the product specific architecture. Finally, when the number of selected variants of a specific variation point is more than one, the generalized element will contribute in the product architecture as well.

It should be noted that the product architecture has no link to variabilities of product family this complicates the maintenance and evolution of product architecture. Having stand-alone product architecture, it will not be trivial to infer what decisions were made during application engineering phase.

In our view, the product architecture should be linked to the orthogonal variability model in a way that the architecture could be defined as decisions which are made through architecture derivation. Since we did not cover the orthogonal variability model, at this paper we can not propose an exact solution for tracing the selected choices to associated variants. However, if the orthogonal variability model is represented in UML, the selected variants in architecture can be linked with the associated variant of variability model by means of tagged values.

7. Related Work and Discussion

Over the past few years several variability modeling methods have been developed. They have commonalities and differences regarding the modeling concepts that are emphasized. Some of these methods such as COVAMOF [22], FODA [15], FORM [16] and VSL [6] are rather general for all software artifacts and do not support the concerns associated to the specific artifacts such as architecture. Consequently, we do not discuss them in our comparison.

Clauss [9] and Ziadi et al. [26], exploit UML as their modeling language in their variability modeling method. However, the proposed method is specific to UML class diagrams which can be considered as a design level rather than architectural level artifact. They both propose UML profiles in which stereotypes and tagged values are defined in order to specify variation points and variants. The drawback of this approach is that extensive use of stereotypes and tagged values results in a cluttered model and will decrease scalability of the model. Ziadi proposes model transformations used to instantiate specific products.

Pohl et al. [19] uses orthogonal variability models to represent variability at architectural level. These models improve separation of concerns by separating variability and architecture representation. However, their representation does not indicate exactly which elements are variable and how the variabilities should be realized. Although they make use of architectural views, they ignore the specific characteristics of each view and artifact since they use a single modeling method based on their orthogonal variability model in all artifacts.

ADLs are commonly used for describing the architecture of a single system, and as result generally lack concepts and constructs required for modeling the variability of product family architectures. However, there are also exceptions like Koalish [1] and xADL [25] which are devised for modeling PLAs. As previously mentioned, all ADLs represent architecture as a single view entity.

xADL is a XML based ADL which benefits from being highly extensible to new domains. The basic modeling elements in xADL are component type, connector type and interface type. There are some tools such as Ménage which are based on xADL and

facilitate visualizing the architectural representation. One drawback of xADL is that it is based on single view representation of architecture which weakens its practicality. Also issues like variability realization techniques are not taken into account in this variability modeling technique.

Koalish is a variability modeling language which extends Koala with variability. Koala is an ADL that exploits components and interfaces to specify the architecture without supporting variability. Koalish is further more text based and does not represent a graphical interface. Koala defines a connector as a binding between a required interface of one component to provided interface of the other one. Koala makes use of an abstraction mechanism which simplifies the architecture representation. One drawback of Koala is that the architectural connector is not treated as a first class entity and not modeled explicitly. The other drawback is that variabilities are specified in Koalish which is a text-based modeling language and hence provides a poor mechanism for variability representation. Koala also has the drawbacks mentioned for xADL.

Thiel et al. [24] and Bachman et al. [4] propose variability modeling method based on multiple view architecture. Thiel and Hein address variability as an architectural driver which is represented in multiple views including logical, process, physical and deployment. In addition, the effects of variability on each view are assessed. The product architecture is represented by decision model. The problem is that the modeling language used is not clearly defined. Bachman and Bass propose a variability modeling method limited to the static structure of architecture, which is similar to module view type, but the method lacks the same problem.

A framework for classifying variability modeling techniques is proposed in [23]. In order to give a comparison of our method with the other methods, we classify our method within the mentioned framework. A variability modeling method is characterized by the following features:

- Variability information representation (or choice representation)
- Representation of product model
- Abstractions used to manage complexities.

There are also some other features like formal constraints, quality attributes and support for incompleteness which are out of scope of this work.

Our method could be classified as a modeling technique which uses *multiplicity in structure* to model *choices*, since variability is realized by having multiplicity in architectural elements. Regarding the product model, our approach falls into the *stand-alone* architecture category since the product model is not traceable to variabilities of the product family. However, in the case where the product architecture is linked to the orthogonal variability model, our product model could be considered a *decision model*. The abstraction of complexities is supported by the *2-level* organization of architecture which is classified in the *multiple-layer* representation category. Since we make use of the component and connector model of UML 2, we can have composite structures, i.e. architectural elements (components or connectors) containing other elements. As a result we also support *hierarchy* in our PLA

8. Conclusion

In this paper, we presented a method for modeling variability in C&C view of the architecture. We devised a method based on UML 2 as the architecture modeling language, in order to avoid proposing an abstract general method which does not consider specific details of modeling languages. It also makes our method more suitable for practical use. Another advantage is that it is an extensible language and therefore available UML based tools can be used for modeling PLA, by means of our UML profile. Also, as mentioned in section 7, the composite structure of UML 2 makes it possible to have hierarchical abstraction in the architecture representation.

The method helps the architect recognize how the variability arises in the architecture in the following aspects:

- Identification of variable elements which could be the origins of variability in the architecture
- Choice of appropriate techniques for variability realization and reflection of these techniques on the architecture model
- Representation of the architecture and its associated variability in a single model

The precise definition of variability in the architecture enables the architect keep track of the effects of variabilities in different elements of the architecture. This also facilitates more precise evaluation of the architecture regarding non-functional requirements.

In order to achieve an effective variability management method, it is essential to employ an orthogonal variability model including information about variabilities derived from the problem space. We believe that an orthogonal variability model based on UML can benefit form ease of integration

with other models. Also this kind of variability model can facilitate developing tools intended for taking over some of variability management tasks. This is a complement to our method, which can be studied in future work.

Also, representing variability in other views of architecture, based on the variable element types of those and the associated realization techniques is another direction in which this work can be extended.

Acknowledgement

We would like to thank Kamran Fallah for his helpful comments on this work.

References

[1] T. Asikainen, T. Männistö and T. Soininen, "Kumbang: a domain ontology for modeling variability in software product families**",** Advanced Engineering Informatics, Vol. 21, 2007, pp. 23-40.

[2] C. Atkinson, J. Bayer, and D. Muthig, "Component based product line development: The KobrA approach", Proceedings of the First Software Product Lines Conference (SPLC1), Kluwer Academic Publishers, 2000, pp. 289 – 309.

[3] F. Bachmann and L. Bass, "Managing variability in software architectures", Proceedings of the ACM Symposium on Software Reusability: Putting Software Reuse in Context, 2001, pp. 126-132.

[4] F. Bachman and P. Clements, "Managing variability in software product lines", Technical Report No. CMU/SEI-2005-TR-012, Software Engineering Institute, Carnegie Mellon, September 2005.

[5] L. Bass, P. Clements, and R. Kazman, "Software architecture in practice", Second Edition, Addison Wesley, 2003.

[6] M. Becker, "Towards a general model of variability in product families", Proceedings of the 1st Workshop on Software Variability Management, February 2003.

[7] K. Berg, J. Bishop and D. Muthig," tracing software product line variability – from problem to solution space", Proceedings of SAICSIT, pp. 182-191.

[8] J. Bosch, "Design and use of software architectures: adopting and evolving a product line approach", Addison Wesley, 2000.

[9] M. Clauss, "Generic modeling using UML extensions for variability", Proceedings of Workshop on Domain-specific Visual Languages, Jun 2001, pp. 11-18

[10] P. Clements, F. Bachman, L. Bass, J. Ivers, D. Garlan, R. Little, R. Nord and J. Stafford, "Documenting software architectures: views and beyond", Addison Wesley, 2003.

[11] H. Gomaa, "Designing software product lines with UML: from use cases to pattern-based software architectures", Addison Wesley, 2004.

[12] H. Gomaa and D. Webber," Modeling adaptive and evolvable software product lines using the variation point model", Proceedings of the 37th Hawaii International Conference on System Sciences, 2004.

[13] J. Ivers, P. Clements, D. Garlan, R. Nord, B. Schmerl and J.R.O. Silva, "Documenting component and connector views with UML 2.0", Technical Report No. CMU/SEI-2004-TR-008, Software Engineering Institute, 2004.

[14] I. Jacobson, G. Booch, and J. Rambaugh, "The unified software development process", Addison Wesley, 1999.

[15] K. Kang, S. Cohen, J. Hess, W. Novak and S. Peterson, "Feature oriented domain analysis (FODA) feasibility study", Technical Report No. CMU/SEI-90-TR-021, 1990

[16] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain Specific Reference Architectures", Annals of Software Engineering, vol. 5, 1998, pp. 143 – 168.

[17] P. Kruchten, "The 4+1 view model of architecture", IEEE Software Vol. 12, November 1995, pp. 42 – 50.

[18] N. Medvidovic, D. Rosenblum, D. Redmiles and J. Robbins, "Modeling software architectures in the unified modeling language", ACM Transactions on Software Engineering and Methodology Vol. 11, ACM Press, January 2002, pp. 2 – 57.

[19] K. Pohl, G. Bockle and F. van der Linden," Software product line engineering foundations, principles, and techniques", Springer-Verlag Berlin Heidelberg, 2005.

[20] N. Rozanski and E. Woods, "Software systems architecture: Working with stakeholders using viewpoints and perspectives", Addison-Wesley, 2005.

[21] K. Schmid and I. John, "A customizable approach to full lifecycle variability management", Science of Computer Programming, 53, 2..4, pp. 259-284.

[22] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "COVAMOF: A framework for modeling variability in software product families", Proceedings of the Third Software Product Line Conference (SPLC 2004), Lecture Notes on Computer Science Vol. 3154 (LNCS 3154), Springer Verlag, August 2004, pp. 197-213.

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

[24] S. Thiel and A. Hein, "Systematic integration of variability into product line architecture design", Proceedings of the Second International Conference on Software Product Lines, Lecture Notes on Computer Science Vol. 2379, Springer Verlag, 2002, pp. 130-153.

[25] A. van der Hoek, "Design-time product line architectures for any-time variability", Science of Computer Programming special issue on Software Variability Management 53, 2004, pp. 285–304.

[26] T. Ziadi, L. Helouet and M. Jezequel, "Towards a UML Profile for Software Product Lines", Springer Verlag Berlin Vol.3014, 2004, pp. 129-139.