Variability Viewpoint for Introducing Variability in Software Architecture Viewpoints

Bedir Tekinerdogan
Department of Computer Engineering
Bilkent University
Ankara, Turkey
bedir@cs.bilkent.edu.tr

Hasan Sözer
Department of Computer Engineering,
Özyegin University
İstanbul, Turkey
hasan.sozer@ozyegin.edu.tr

ABSTRACT
Variability is the ability of a software system to be changed for a specific context, in a preplanned manner. As such, to facilitate the instantiation of a software architecture the variability concern needs to be explicitly addressed. Usually, architectural concerns are represented using architecture views that are derived from the corresponding architecture viewpoints. Different software architecture viewpoints have been introduced to support the modeling, understanding, communication and analysis of the software architecture for different stakeholders. Regarding variability we can observe that this has been mainly addressed in separate variability modeling approaches. In this paper we first provide a short overview of the approaches for dealing with variability at the architecture design level and then introduce the *variability viewpoint*. The variability viewpoint addresses the concerns for variability and can be used to introduce variability in software architecture viewpoints

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architecture –*Domain-Specific Architecture, Languages*

General Terms
Management, Documentation, Design

Keywords
Software Architecture Modeling, Architectural Views, Quality Concern, Variability

1. INTRODUCTION
Current systems are rarely designed for a single fixed system but need to embrace variability to cope with the different context and application requirements. Variability is the ability of a software artifact to be configured, customized, extended, or changed for a specific context, in a preplanned manner [1]. As such, to facilitate the instantiation of a software architecture, variability concerns need to be explicitly addressed [4][5]. A common practice is to model different architectural views for describing the architecture according to the stakeholders' concerns [2]. An architectural view is a representation of a set of system elements and relations associated with them to support a particular concern [2]. Different software architecture viewpoints have been introduced to support the modeling, understanding, communication and analysis of the software architecture for different stakeholders. Unfortunately, an analysis of the existing multi-view approaches reveals that they still appear to be incomplete when considering variability concerns. The ISO/IEC 42010 [7] standard intentionally does not define particular viewpoints to address the different concerns. In the architecture frameworks such as the V&B approach, variability concerns appear to be implicit in the different views. One could argue that for addressing variability several variability modeling approaches such as feature modeling have been introduced [8]. The difficulty here is that these approaches usually apply a separate model and do not depict the decomposition of the architecture and an additional translation from the variability model to the architecture design needs to be performed. On the other hand, existing integrated variability approaches [6] are usually fixed to a given design language or viewpoint. To represent variability concerns more explicitly, preferably an architectural view is required to model the architecture based on the required variability properties [5]. In this paper we first provide an overview of the approaches for dealing with variability at the architecture design level and then introduce the *variability viewpoint*. The variability viewpoint addresses the concerns for variability and can be applied to different viewpoints to introduce variability to the architecture components and connectors.

The remainder of this paper is organized as follows. Section 2 introduces the architecture variability metamodel that integrates the architecture elements with the variability concepts. Section 3 presents the variability viewpoint that is based on the metamodel in section 2. Section 4 presents examples of viewpoints to which the variability viewpoint is applied. Section 5 presents the related work and finally section 6 presents the conclusions.

2. ARCHITECTURE VARIABILITY METAMODEL
In Figure 1 we show the conceptual model for architectural view modeling. In fact, the conceptual model is based on the ISO/IEEE/IEC recommended standard for architectural description [7] but it enhances the standard to explicitly depict quality concerns and defines the relation to architectural views. In the figure the gray part represents the part of the standard whereas the lower part represents the proposed extensions.

The left part of the figure shows basically the definition of the architectural drivers. A system has one or more stakeholders who have interest in the system with respect to one or more concerns. Concerns can be functional or quality. Variability in software architecture is often treated as a quality attribute [4] and we consider variability as a quality concern. The right part of the

WICSA/ECSA 2012, August 20–24, 2012, Helsinki, Finland.

figure focuses on the architectural views for the different concerns. Each system has an architecture, which is described by an architectural description. The architectural description consists of a set of views that correspond to their viewpoints. Viewpoints aim to address the stakeholder's concerns. Functional concerns will define the dominant decomposition along architectural units that are mainly functional in nature. On the other hand, variability will affect the architecture views.

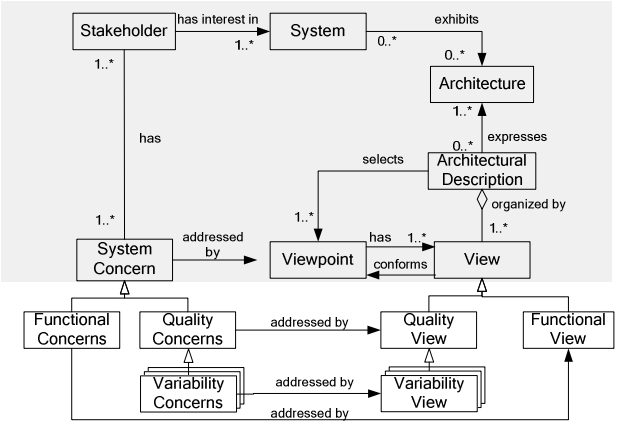


Figure 1. Conceptual model for Architectural Views based on ISO/IEEE/IEC Standard

Based on the conceptual of Figure 1 and the corresponding literature we have defined the metamodel in Figure 2 that integrates both architecture modeling and variability concerns. The left part shows the software architecture that includes architecture elements which are architecture components and architecture relations. The right part of the metamodel has been largely adapted from the metamodel as defined by Pohl et al. [8]. Hereby *Variability Element* is defined as an element that represents an architecture element which includes variability. Note that variability elements can be both components and connectors. Variability elements are either *Variation Point* (VP) or *Variant*. VP is defined as the location in the system in which the variation can occur. This could be thus either architecture component or architecture relation. At each VP different *Variants* can occur. *Variability Dependency* is an association relation between VP and variant and defines the different dependency relations including *Mandatory, Alternative, Or,* and *Xor.* Variability Constraint represent the configuration constraints including *Mutex* and *Requires.* The constraint can hold for both between VP and Variants. Requires constraint between two VP/Variants means that when we include a VP/Variant, then the other VP/Variant must be included too. The mutex constraint means that once we include a VP/Variant, then the other VP/Variant must be excluded.

The metamodel in Figure 2 aims to represent a general model for representing variability for architecture design. A close analysis of the literature shows that we can in essence distinguish between *integrated variability* approaches [6] and *separate variability* approaches. *Integrated variability approaches* represent variability within the adopted models (e.g. UML class diagrams, use case diagrams etc.). On the other hand *separate variability modeling* provides both a separate variability model and the model to which the variability applies. In fact both approaches seem to have their advantages and disadvantages. In the integrated approach, variability is directly represented and thus visible in the model abstractions. As such no translation from a separate variability model to the model abstractions are required. The counterpart of the approach is that the model abstractions are 'polluted' with the variability concerns. In the separate variability approach the model abstractions are nicely separated from variability concerns, but this requires the translation effort from variability models to model abstractions.

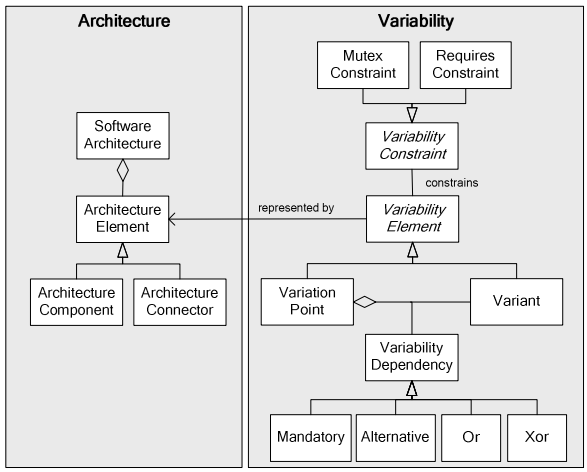


Figure 2. Metamodel that integrates architecture modeling with variability

3. VARIABILITY VIEWPOINT

In this section we present the variability viewpoint that is based on the metamodel as defined in Figure 2. The template for the viewpoint is adapted from [3]. The viewpoint is shown in Table 1. The relation from variability viewpoint to architecture viewpoint is shown in figure 3.

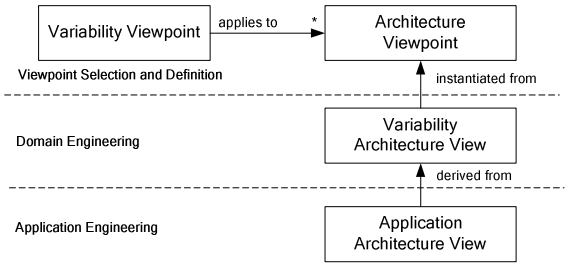
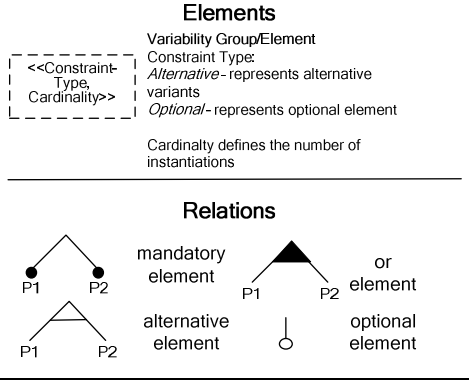


Figure 3. Metamodel that integrates architecture modeling with variability

As shown in Figure 3 the Variability Viewpoint can be applied to different viewpoints. Application of viewpoint means merging the elements of the variability viewpoint with the elements of the selected viewpoint. In this way variability mechanism are integrated to the selected viewpoint. Instantiating the variability viewpoint is therefore not directly possible but is performed indirectly over another viewpoint that defines the architecture components and connectors. If the variability viewpoint is applied to a particular viewpoint then in essence a customization of the corresponding viewpoint is defined. The resulting viewpoint is named *Variability – <Viewpoint Name>*. Simply, the term "Variability" is included as a prefix to the name of the viewpoint on which the Variability Viewpoint is applied. For example, if we apply the variability viewpoint to the Decomposition Viewpoint then the resulting viewpoint will be *Variability – Decomposition Viewpoint.*

Once the newly defined variability viewpoint has been defined then the corresponding architecture view with variability can be defined based on this viewpoint. Views that result from variability viewpoints represent in essence reference architecture from which different application architectures can be derived. The reference architectures are called *Variability Architecture Views*. In case of, for example, the merge with the *Variability Viewpoint* with the *Decomposition Viewpoint*, a variability reference architecture view called *Variability Decomposition View* can be defined. Based on the variability architecture views, application architecture views are derived to represent the architecture of a particular application. In fact, the above variability viewpoint approach can thus be considered as an integrated approach since the variability is directly visible in the architecture view. However, our approach is different from existing integrated viewpoint approaches since it cannot be directly used by itself but needs to be applied to another viewpoint before. From this perspective it also shares the benefit of both approaches. On the one hand it shows the variability directly in the corresponding view and there is no need for translation. On the other hand the variability and architecture model are loosely coupled.

Table 1. Variability Viewpoint Guide

Viewpoint Element	**Description**
Name	Variability Viewpoint
Overview	This viewpoint is used for defining the variability of architectural elements
Elements	Variability elements which are elements as defined in the viewpoint to which it applies
Relations	Variability elements which are relations as defined in the viewpoint to which it applies
Constraints	As defined by the viewpoint to which it is applied
Notation	

4. EXAMPLES

In this section we present example instantiations of the variability viewpoint. For this we will select the decomposition, uses, component & connector, and deployment viewpoints (styles) as defined in the Views and Beyond approach [2].

Figure 4 shows the application of variability viewpoint to the decomposition view. The view shows the decomposition of the system into different modules. In the given view we can see that the system can be decomposed using either the modules A or B. Module C is optional, while modules E and F are mandatory. Based on this view

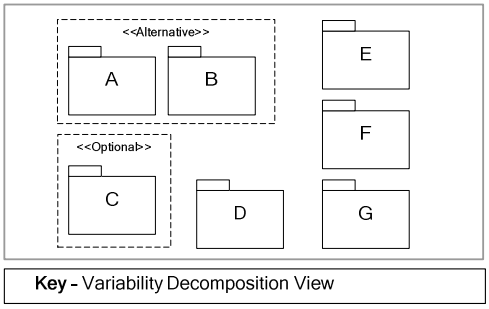


Figure 4. Variability Decomposition View

Figure 5 shows the application of variability viewpoint to the uses view. The view shows the dependency (uses) relations among the modules of the system. In the given view we can see that the alternatives A and B can optionally use (open bullet) the module C but have to (filled bullet) use module D. Module E can use either module F or module G.

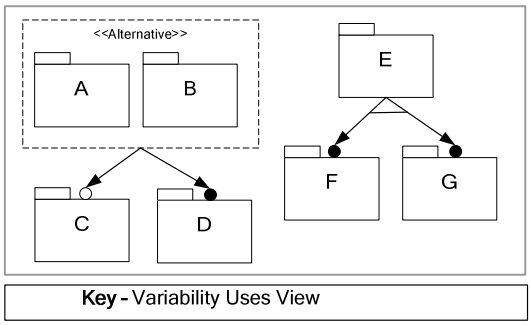


Figure 5. Variability Uses View

Figure 6 shows the application of the Variability viewpoint to the Component & Connector view. Hereby, Component D depends on an optional interface that is provided by Component C. Component D also depends on Component A, which is designed and implemented to be fault tolerant. There exist multiple implementations of the same functionality. At least one of them should exist in the system and they can be swapped at runtime as they constitute alternatives to each other.

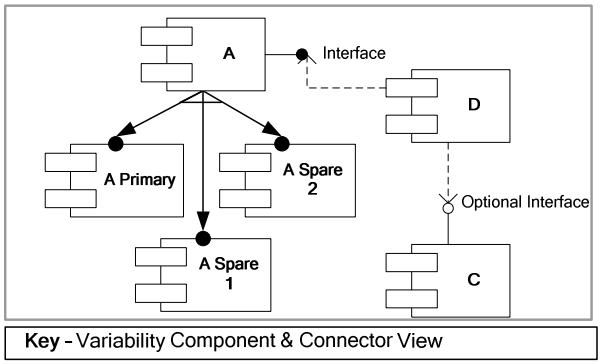


Figure 6. Variability C&C View

Figure 7 shows the application of variability viewpoint to the deployment view. In the given view we can see that the system is deployed on Node N1, optionally there will be a Node N2, and either Node N3 or Node N4. Node N1 has mandatory modules E, F and G, and the alternatives A and B. Node N2, if selected,

includes an optional module C and a mandatory module D. Node N3 and N4 both include module D.

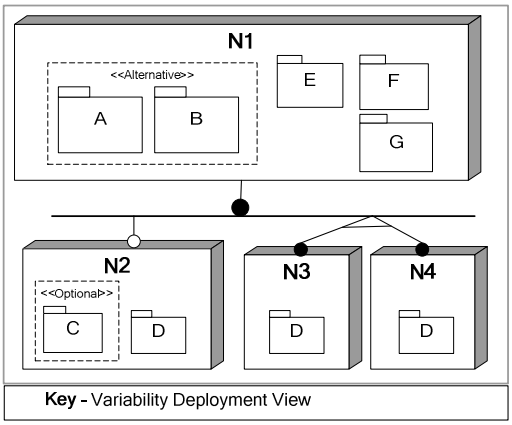


Figure 7. Family Deployment View

5. RELATED WORK

In [5] the authors describe the result of an exploratory study in which eleven major problems based on variability-related tasks were performed by participants. The paper discusses the implications of these problems on architecture description, methods and tools, and training of architects. In [4], based on a survey, the authors conclude that variability in the context of software architecture is poorly understood and for this they suggest to address variability as "first-class quality attribute". We believe that variability viewpoint is an important direction towards this goal.

In our earlier work we have defined a general approach for modeling architecture viewpoints for quality concerns [12][11][9]. Similar to variability also other quality concerns cannot be easily represented in current architectural views and tend to crosscut elements within an architectural view. We have provided examples for two quality concerns, recoverability and adaptability. The variability viewpoint is defined using the approach as described in our earlier paper, but also has some interesting issues. Similar to the recoverability and adaptability viewpoints, variability is also crosscutting over different viewpoints. However, different from these two viewpoints variability is not just an 'overlay' mechanism but requires a more invasive impact on the architecture elements and relations. The result however is similar, better support for communication, guiding the design (decisions), and support for analyzing the architecture design alternatives.

In our approach we have applied the separation of concerns principle to separate the views for quality concerns. Similar to crosscutting concerns in Aspect-Oriented Software Development (AOSD) [2], quality concerns seem to crosscut the elements in the functional views. By separating these quality concerns and providing explicit abstractions in the viewpoints, we have supported an enhanced description of the architecture.

Architectural Perspectives [9] are a collection of activities, tactics and guidelines to modify a set of existing views to document and analyze quality properties. Architectural perspectives as such are basically guidelines that work on multiple views together. An analysis of the Architectural Perspectives and our approach shows that the crosscutting nature of quality concerns can be both observed *within* an architectural view and *across* architectural views. Both approaches focus on providing a solution to the crosscutting problem. We have chosen for providing separate architectural viewpoints for quality concerns.

6. CONCLUSION

We have provided the variability viewpoint for addressing variability concerns within the architecture views. The benefit of this is that variability is directly visible and there is no need for translation from a variability model to the architecture view. We have applied the viewpoint for four different viewpoints from the Views and Beyond approach. We believe that the variability viewpoint can also be applied to other viewpoints from both the Views and Beyond approach but also from other architecture frameworks. In our future work we will apply the variability viewpoint for defining variability to architecture views in a multiple product line engineering within an industrial case study.

7. REFERENCES

[1] F. Bachman and P.C. Clements. Variability in Software Product Lines. Technical Report CMU/SEI-2005-TR-012, Pittsburgh, PA, 2005.

[2] R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, J. Bakker, M. Pinto Alarcon, B. Tekinerdogan, S. Clarke, and A. Jackson, Survey of Aspect-Oriented Analysis and Design, AOSD Europe network of excellence, no. AOSD–Europe-ULANC-9, pp. 1-259, 2005.

[3] P. Clements et al. Documenting Software Architectures: Views and Beyond. Addison-Wesley, September 2011.

[4] M. Galster and P. Avgeriou. The Notion of Variability in Software Architecture – Results from a Preliminary Study. Fifth International Workshop on Variability Modeling of Software-Intensive Systems, pp. 59-67, 2011.

[5] M. Galster and P. Avgeriou. Handling Variability in Software Architecture: Problems and Implications, 9th IEEE/IFIP Working Conference on Software Architecture, pp. 171-180, 2011.

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

[7] [ISO/IEC 42010:2007] Recommended practice for architectural description of software-intensive systems (ISO/IEC 42010). (identical to ANSI/IEEE Std1471–2000), July 2007.

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

[9] N. Rozanski and E.Woods. Software Systems Architecture – Working with Stakeholders using Viewpoints and Perspectives. Addison-Wesley, 2005.

[10] H. Sözer, B. Tekinerdogan & M. Akşit. FLORA: A Framework for Decomposing Software Architecture to Introduce Local Recovery, Wiley Software Practice and Experience journal, Vol. 39, No. 10, pp. 869-889, July, 2009.

[11] H. Sozer and B. Tekinerdogan. Introducing Recovery Style for Modeling and Analyzing System Recovery. Seventh Working IEEE/IFIP Working Conference on Software Architecture, pp. 167 –176, 2008.

[12] B. Tekinerdogan, H. Sözer. Defining Architectural Viewpoints for Quality Concerns, in Proc. of the 5th European Conference on Software Architecture (ECSA 2011), LNCS 6903, pp. 26–34, 2