Svamp — An Integrated Approach to Modeling Functional and Quality Variability

Mikko Raatikainen*, Eila Niemelä†, Varvana Myllärniemi*, Tomi Männistö*
*Helsinki University of Technology (TKK), †VTT Technical Research Centre of Finland
*{Mikko.Raatikainen,Varvana.Myllarniemi,Tomi.Mannisto}@tkk.fi, †Eila.Niemela@vtt.fi

Abstract

Software variability modeling is a complex task. To manage this complexity, we introduce an approach called Svamp. The main contribution of Svamp is to model concepts through ontologies and offer tool support for capturing functional and quality variability in software product family architectures. Variability description languages are defined by different ontologies that provide meta-models. For structural and functional descriptions, the concepts, properties, and rules are defined by Kumbang ontology. Quality Attribute ontology defines the domain knowledge of a specific quality attribute, while Quality Variability ontology provides the concepts and rules related to quality variation. The approach is exemplified by our integrated tool suite, provided as a plug-in for the Eclipse platform.

1. Introduction

Variability is the ability of software to be efficiently extended, changed, customized, or configured for use in a particular context [23]. Typically, variability is defined in software when software is developed for reuse. For example, variability is defined in the assets and the architecture of a software product family during the domain engineering phase. The developed variability is then taken advantage for differentiation of software. For example, defined variability in the assets of a software product family is used to derive the different products of a software product family. Software, which differs from other software by taking advantage of variability, is referred to as a variant.

Variability can become complex, since the number of potential variants grows exponentially when new variability is introduced. In addition, variability concerns and affects not only functionality but also quality attributes of software. Consequently, in order to ensure correctness of functionality of a variant and predict its quality properties even in the most complex circumstances, variability needs to be explicated such that it can be efficiently managed and even automated with tool support. Toward this end, an essential characteristic is clarity of underlying concepts for different software artifacts. Conceptual clarity is especially important when variability spans different software artifacts such as requirements, architectural elements, functionality, and quality. Such variability can even affect diverse concerns of stakeholders in an organization.

In this paper, we discuss an approach to capturing variability of a software product family called Software variability modeling practices (Svamp). For given functional and quality requirements, we outline concepts for modeling the structure of features and components that contribute to the functionality and quality attributes of the components. The modeling concepts have been defined rigorously as ontologies. The feasibility of the concepts is shown with an integrated tool suite. With the resulting model, derivation of system variants seems feasible such that the variants fulfill functional and quality requirements.

The rest of the paper is organized as follows. Section 2 provides background of the method. Section 3 describes the approach. In Section 4, the developed tool suite is introduced. In Section 5, we discuss experiences and future research. Section 6 draws conclusions.

2. Background

Software variability management has emerged recently, especially in the area of software product families that focus on enhancing development of a set of different variants within an organization [3]. A key issue in and a lesson learned about the success of software product families is that the products of a software product family follow the same fundamental structure, referred to as a software product family architecture [2]. Consequently, software product family architecture seems to be especially relevant from the point of view of variability, although other development artifacts are affected as well.

A software architecture describes the high-level structure

of a software system. Software architecture is an important means, for example, for performing different types of analysis and for achieving different quality attributes, and communication. Variability is added to software family architecture while still retaining other key aspects of software architecture. The state of the art and practice for managing software architecture is based on views and viewpoints. A view is ”a representation of a whole system from the perspective of a related set of concerns” [8]. The guidelines for constructing and using a view are described in a viewpoint. The rationale for using different viewpoints is to take into account different stakeholder concerns, which are, in fact, a major intention of view-based approaches.

Several modeling approaches have been proposed to express variability. Feature models [11] are one of the first widely known approaches that take into account variability. A feature refers to user visible characteristics of a system. Recently, other modeling approaches peculiar to variability have emerged, such as ConIPF [7] and decision-oriented modeling [4]. These approaches introduce a modeling method with constructs for modeling software assets and variability within the assets.

In addition, different approaches to modeling variability in existing models of software assets have emerged. For example, orthogonal variability modeling [19] augments existing models with variability specific information. Covamof [22] augments existing models with a variability specific model and another model that captures dependencies of a variability model. The methods can therefore be used in conjunction with any software artifact such as requirements or detailed design, or with any architectural viewpoint.

The modeling concepts, however, focus typically on functionality or structure of software. Quality attribute variability, especially at the architectural level, seems still to be a research challenge [13].

3. Svamp modeling concepts

The Svamp approach is to model functional and quality variability at the architectural level. The approach adheres to state-of-the-practice in architecture description by applying different viewpoints. More specifically, a feature and structural viewpoint specifies the structure and functionality, and also variability within these. The structural viewpoint is also referred to informally as a component viewpoint. The elements in a structural viewpoint, that is, its components, are then augmented with quality attributes and, further, quality variability information. The architectural level was selected in the present approach since it seems to be especially significant for variability, as argued above.

Consequently, the approach uses several integrated models to model a software product family (Figure 1): a Kumbang model, consisting of structural and feature viewpoints

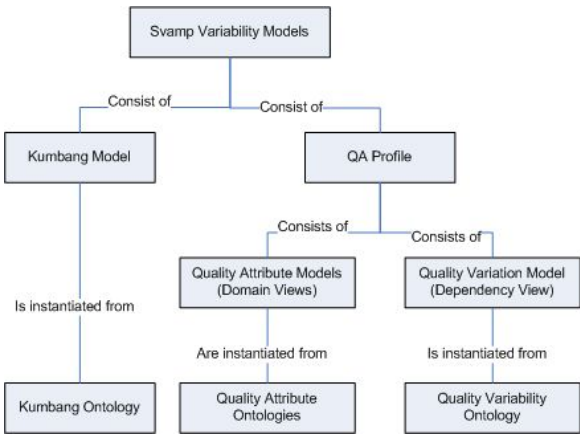

Figure 1. Svamp variability models and ontologies.

for functional and structural characteristics; a quality attribute profile, consisting of a quality attribute model for each quality attribute of the components in the structural viewpoint of the Kumbang model; and a quality variability model for expressing variability within these quality attributes. Each of these three models is defined in its own ontology; the corresponding ontology provides a meta-model for the modeling concepts.

Kumbang concepts form the basis for modeling since other models use the components defined in a Kumbang model; hence, the Kumbang model needs to be specified first. Roughly, Kumbang concepts synthesize existing feature modeling methods and structural modeling of architectural components, in particular Koala [25]. Kumbang adds explicit variability concepts into these methods and provides formal semantics for the concepts. In the following, we only briefly outline basic capabilities of Kumbang, whereas a comprehensive description can be found in [1].

The feature viewpoint is used for modeling feature types, which represent user visible functional characteristics of a system. Kumbang uses the term ”type” to refer to an element in the variability model, while elements referring to a specific variant are, e.g., feature instances or simply features. Features can be composed such that other features are their subfeatures. Such a composition structure is specified within a feature type using subfeature definitions, which specify the cardinality and possible types of composed features. Further, feature types can inherit each other. Feature types can be characterized with attribute definitions, which represent name/value pairs. Finally, constraints can be used to specify more elaborate rules for selection of different feature instances; in a very simple case, by specifying that a certain feature requires another feature. Hence, Kumbang

feature modeling concepts synthesize many existing feature modeling methods, and can be used to capture typical variability constructs found in other feature modeling methods.

Structural viewpoint specifies component types. A component type represents a distinguishable architectural element with explicitly defined interfaces. The approach is ignorant as to whether a component actually refers to, e.g., a run-time or design-time element or a specific component technology, as far as the component adheres to this definition. Again, the term "component type" is used in the variability model similarly as in features. An interface type represents a set of operation signatures; these are attached to component types using interface definitions. Interface direction is provided or required and the interface can be optional. Components can be composed with each other. The construct used for specifying component composition is a part definition that specifies the cardinality of possible types of composed components. Similarly to feature types, component types can inherit each other, be characterized by attribute definitions, and specify constraints that restrict how instances in the structural viewpoint can be selected. Hence, the variants of structural viewpoint can differ in terms of composition of components, connections between the interfaces, and attribute values defined.

In order to integrate feature and structural viewpoint, implementation constraints can be used to specify relationships between them. In a very simple case, a specific feature may require a specific component. In general, the constraints can be bi-directional and impose many-to-many relations between viewpoints. Consequently, the implementation constraints can be as complex as can be specified with Kumbang constraint language [12].

To address quality attributes, the variability model needs to be then augmented with information on its quality characteristics. This is done by specifying the quality properties using the quality attribute model (QA model) and the quality variability model (QV model), defined separately from Kumbang (cf. Figure 1). The components of the structural viewpoint are supplemented with relevant quality profiles. Similarly to Kumbang, both QA model and QV model have been defined as ontology, the former as quality attribute (QA) ontologies and the latter as quality variability (QV) ontology.

Each QA ontology defines the technical dimension of the quality attribute. For example, the main concepts of the security QA ontology are security assets, attributes, threats, solutions, and metrics (Figure 2) [20], whereas the reliability QA ontology defines processes, methods, models, and metrics [26]. That is, QA ontologies are quality attribute specific, and, hence, the concepts in each ontology are different. QA ontologies are orthogonal and managed separately because different expertise is required for defining different QA ontologies. Furthermore, the concepts defined

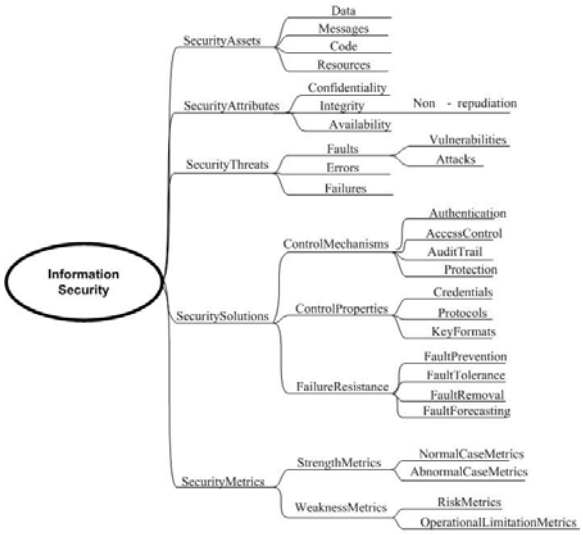

Figure 2. The security QA ontology [12].

in a QA ontology depend on the dissected entity: in defining the security QA ontology, the focus was on information security of service centric systems, while in the reliability QA ontology, the focus was on reliability-aware architecting. Thus, the scope of the reliability QA ontology is larger; therefore, more concepts have been defined.

The QA metrics concept (Figure 3) consists of metrics classes, e.g., strength metrics and weakness metrics. Concepts of QA metrics are common for all quality attributes, whereas only part of the metrics classes and actual metrics in the metrics classes can be shared by different QA ontologies and the others are quality attribute specific. Each metric has the following properties: description; purpose; target, i.e., where the metric can be used; applicability, i.e., when the metric can be used; a set of formulas; range value for the measurement; and the best value of the measurement unit. A rule set constrains the formulas and the used measurement unit by defining the set of targets of measurement, the set of value ranges for the measurement unit, and the time when the metric is valid.

The QV model is defined by four concepts: importance, scope, binding time, and dependency map. The importance of the QA is defined by three distinct property values, i.e., high, medium and low. The importance property is required for making decisions on QA variation. Rules related to the importance property define whether QA variation can take place, for example, QA of high importance cannot be changed at run-time or it can be lowered to the medium level only; in what circumstances QA variation is allowed, for example, QA of low importance can be removed while making

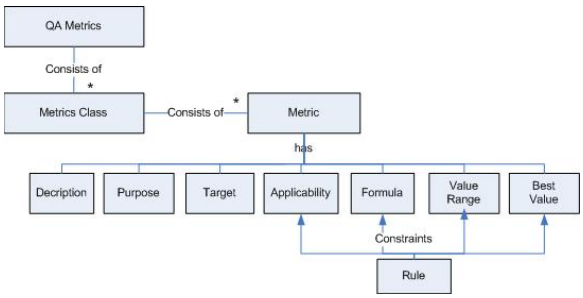

Figure 3. Concepts related to metrics of the QA ontology.

tradeoffs; and how quality attribute variation is to be carried out, for example, QA has to be fixed in product derivation. Some of the rules can be generic, but more often they are software product family specific.

The scope defines four granularity levels for QA variation. That is, scope determines where quality attribute variation can take place by defining a set of boundary types, possible values being family, product, service, or component. One of the values has to be selected. Scope selection restricts the types of appropriate metrics and measuring techniques. For example, at the family and product levels, only those metrics intended for system-level use (cf. Target in Figure 3) can be used. Thus, there are relations between the QA ontologies and the QV model that are considered while defining QA profiles by the QPE tool (see section 4.2).

Binding time defines when quality attribute variation can take place; quality attribute can be changed at design-time, in assembly, in start-up, or at run-time. The binding time is needed for making design decisions and required adaptations and tradeoffs, i.e., QA variation, between quality attributes. Design time tradeoffs are made by determining the optimal architecture with the help of quality evaluation methods and supporting tools, e.g., estimating reliability by the RAP method [9]. Run-time adaptation is made by specific algorithms implemented as part of middleware services. In [18], an example of run-time performance adaptation is given.

The dependency map describes relations between variable quality attributes. This information is required for making tradeoffs between quality attributes. So far, methods exist for making tradeoffs at design-time but no generic solution for making run-time tradeoffs. The QV model is defined in more detail in [16].

4. Tool support

The Svamp approach is supported with a tool suite developed as plug-ins on the Eclipse Platform [5]. Kumbang Modeler is used to model the structural and feature viewpoint whereas Quality Profile Editor (QPE) is used to model quality properties.

4.1. Kumbang Modeler

Kumbang Modeler [14] is a tool that can be used for creating the Kumbang model, that is, to model functional and structural variability in a software product family architecture from feature and structural points of view. The user can specify product family features, architectural elements, and relations between them using constraints. Kumbang Modeler hides the complexity of concrete syntax behind a graphical user interface (Figure 4) and guides the user in the modeling task.

Kumbang Modeler checks the model for syntactic correctness. Further, it checks that at least one valid product configuration can be derived from the model. That is, it checks that all required interfaces can be connected to corresponding interfaces, all constraints can be satisfied, and no cyclic loops exist in inheritance or part structures. This checking is implemented using an efficient smodels inference engine [21], a general-purpose inference tool based on the stable model semantics of logic programs.

After the structural and functional modeling is completed, the user of the tool can augment the model with quality profiles. For this purpose, the tool suite transforms the relevant information of the Kumbang model into the UML2 model specified by Eclipse UML2 meta-model, which is the format understood by the QPE plug-in. The process of using the QPE tool is described in the following.

4.2. Quality Profile Editor

The Quality Profile Editor (QPE) tool [6] takes QA ontologies as input. These ontologies are defined by the quality engineers by using an ontology definition tool, e.g. Protégé. In addition, the software family architect responsible for modeling also needs a list of quality requirements. The user interface of the QPE tool helps in instantiation of QA ontologies. QA ontologies are imported in OWL (Web Ontology Language) files [17] for the QPE tool.

The QPE tool produces a QA profile that instantiates the related QA and QV ontologies and, hence, contains the defined quality properties with metrics, quality variation rules, and dependencies on other quality properties in the same QA profile or in other QA profiles. In QA profiles, the QA properties are defined as UML stereotypes. UML defines profiles as a lightweight mechanism to extend the UML meta-model for adapting the language with domain specific constructs. These extensions are defined by stereotypes that can also contain properties and tag definitions used to set values to property attributes.

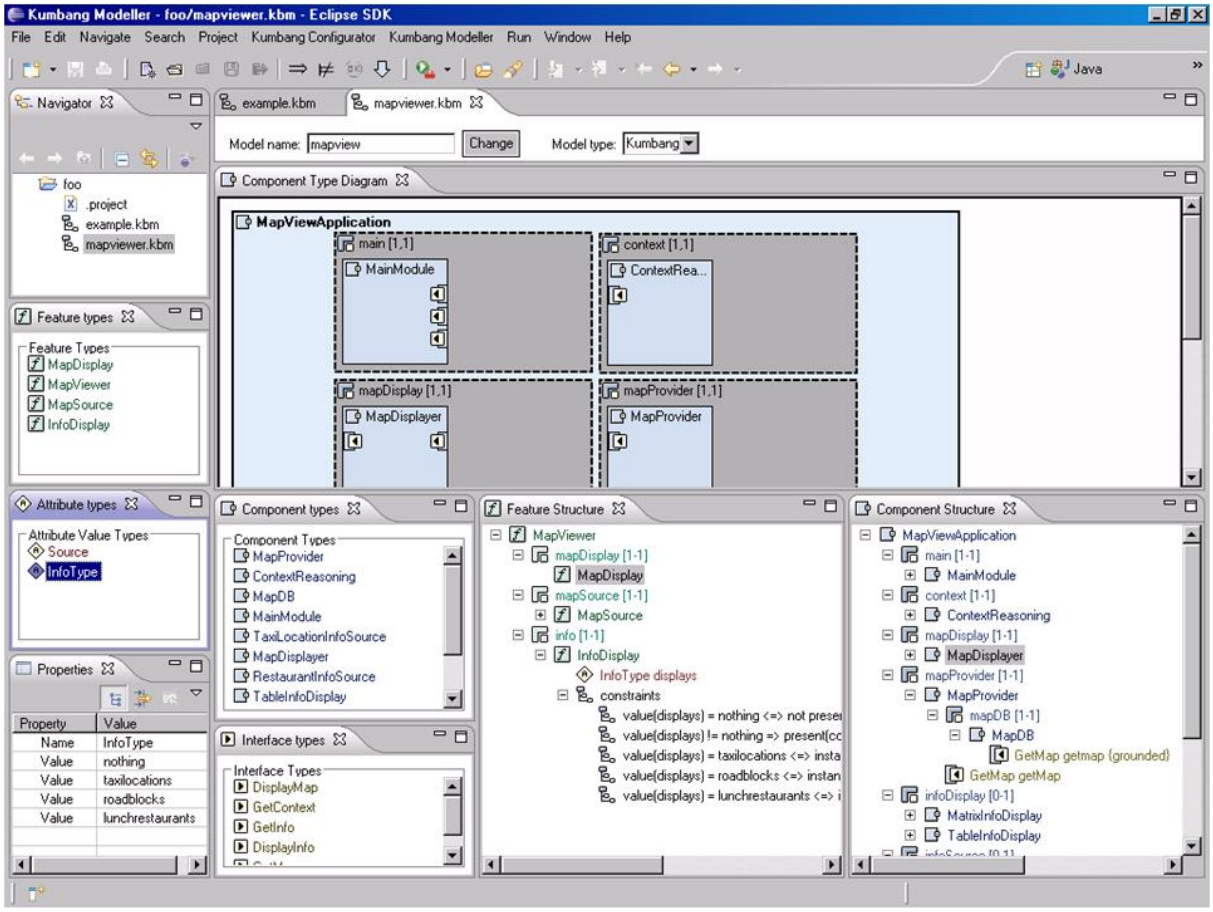

Figure 4. Kumbang Modeler graphical user interface[14].

Figure 5 depicts the user interface of the QPE tool. The left side is used to define quality properties that the family architecture has to meet. On the right side, the architect can select a quality property and bind an appropriate QA metric from one of the QA ontologies to it. Finally, dependencies on other quality properties are linked. For example in Figure 5, Req2 from Demo profile and R3 from the Reliability profile are linked to the Rel1 property.

QA profiles are stored as separate files because of their evolution management; new QA properties can be added and existing ones removed without affecting other QA profiles. However, the software family architect is responsible for checking dependencies between QA properties (inside one QA profile or between the properties in different QA profiles), because the QPE does not check dependencies automatically while the QA profiles are updated.

The stereotypes in the QA profiles are used for mapping the QA properties to the structural elements of the family model (Figure 6). Thus, quality properties as UML2 stereotypes facilitate the viewpoint based approach by enabling a separate focus on components, features, and quality attributes. The only concept the family architect can select while mapping QA properties to the architecture models is Binding time. The reason is that the architecture design is the earliest possible phase when a decision about timing of QA variation can be made. Mapping of QA properties to structural elements can be made with any Eclipse UML2 compatible plug-in; in the case of Svamp it was TOPCASED [24].

The QA property information incorporated into the models is used while evaluating the software product family architecture. Evaluation is made using appropriate evaluation tools, i.e., the evaluation tools are QA specific. The RAP tool [10] supports reliability and availability prediction from the models of software product family. Thus, to evaluate the satisfaction of reliability aspects, UML2 mod-

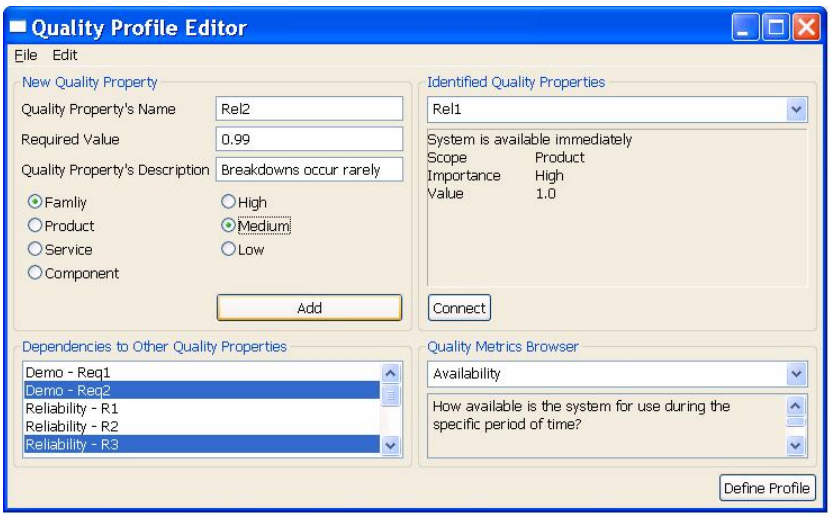

Figure 5. Defining quality properties with the QPE tool.

els produced by the TOPCASED tool are imported in the RAP tool and QA property information used in reliability and availability prediction.

5. Discussion and Further Work

The Svamp approach has been applied to example cases carried out in a laboratory. So far, the following observations have been made:

Functional variability modeling supports functional and structural views, but variability also occurs, e.g., in behavior and deployment views. Despite being feasible for modeling even dynamic concepts, Kumbang concepts per se are not convenient for modeling complex behavior; hence, extensions are needed.

Quality variability modeling supports security and reliability modeling in regard to metrics. More exploratory work is required for facilitating quality-aware architecting, such as performance ontology and quality-driven adaptation of software product families, i.e., tradeoffs made at run-time. In addition, other execution qualities need to be considered together with reliability and security. Further, feasibility of modeling different quality attributes and analyzing them design time needs to be studied in more depth.

As a result of the common tooling platform, tools are independent modules that can be integrated with other tools that conform to the Eclipse Platform and UML2. However, the tool suite needs further improvement, especially in regard to interoperability and automatic transformation of different models. Nevertheless, our experiences with Eclipse as a common platform are encouraging.

We have currently provided concepts and a tool suite for modeling variability. However, in order to take full advantage of variability modeling, a derivation tool and quality evaluation tools using the models are needed. Kumbang Configurator [15] can be used to automate product derivation by checking completeness, consequences, and consistency. Kumbang Configurator supports derivation based on Kumbang models, but currently does not take into account quality attributes. On the one hand, it seems feasible to extend Kumbang Configurator to support quality attributes during derivation; however, this requires further research. On the other hand, the RAP [10] tool supports reliability and availability prediction, but model transformation between Kumbang Configurator and the RAP tool has not been studied.

6. Conclusion

This paper introduced a new approach to modeling variability of software product families by combining functional and quality attribute variability modeling. The concepts have been defined as multiple ontologies with different purposes: Kumbang ontology defines concepts for functional variability, Quality Attribute ontologies define concepts related to specific quality attributes, and Quality Variability ontology defines the meta-model for quality variation. The use of ontology orientation has enabled the development of automated tool support, constructed on the commonly used tooling platform Eclipse. The approach has been tested for feasibility with a simple example. However, more research is needed, especially for more complex systems and derivation support.

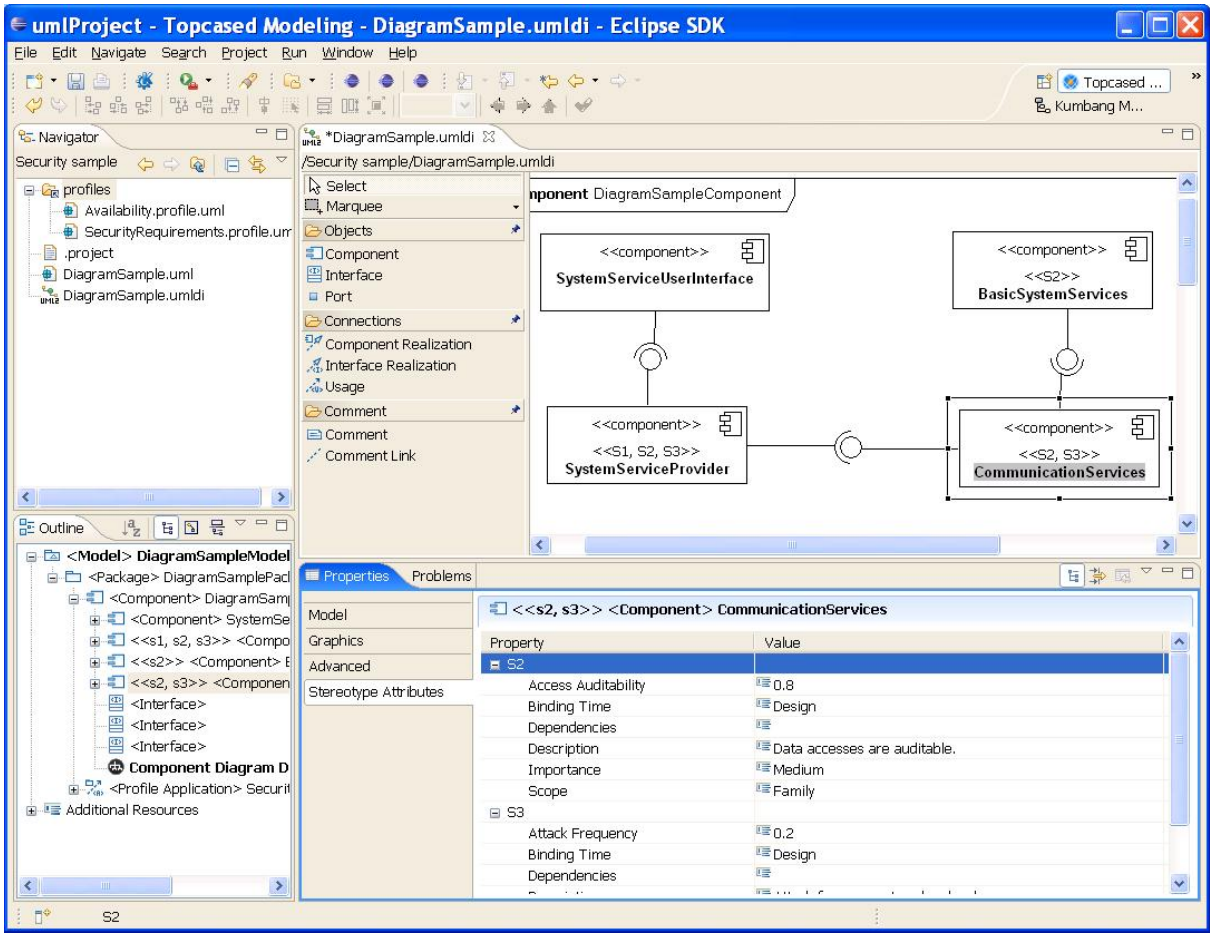

Figure 6. Mapping quality properties to the architectural elements of the structural view.

Acknowledgments

We thank our colleagues Timo Asikainen, Antti Evesti, Hanna Koivu, Pekka Savolainen, and Jiehan Zhou, who participated in the Svamp project by making valuable contributions to specific parts of the presented work. This work was funded by the Finnish Funding Agency for Technology and Innovation (Tekes) and by VTT.

References

[1] T. Asikainen, T. Männistö, and T. Soininen. Kumbang: A domain ontology for modelling variability in software product families. *Advanced Engineering Informatics*, 21(1), 2007.

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

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

[4] D. Dhungana, R. Rabiser, and P. Grunbacher. Decision-oriented modeling of product line architectures. In *WICSA '07: Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture*, 2007.

[5] Eclipse platform. http://www.eclipse.org/, 2008. Visited January 2008.

[6] A. Evesti. Quality-oriented software architecture development. Technical report, VTT Publications, 2007.

[7] L. Hotz, K. Wolter, T. Krebs, S. Deelstra, M. Sinnema, J. Nijhuis, and J. MacGregor. *Configuration in Industrial Product Families*. IOS Press, 2006.

[8] IEEE. *IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems -Description*, 2000.

[9] A. Immonen. A method for predicting reliability and availability at the architectural level. In T. Käkölä and J. Duenas, editors, *Software Product-Lines - Research Issues in Engineering and Management*. 2006.

[10] A. Immonen and A. Niskanen. A tool for reliability and availability prediction. In *EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, 2005.

[11] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, SEI, 1990.

[12] KumbangTools. http://www.soberit.hut.fi/KumbangTools/, 2007.

[13] V. Myllärniemi, T. Männistö, and M. Raatikainen. Quality attribute variability within a software product family architecture. In *Short paper in Conference on the Quality of Software Architectures (QoSA)*, 2006.

[14] V. Myllärniemi, M. Raatikainen, and T. Männistö. Kumbang tools. In *Software Product Line Conference*, 2007.

[15] V. Myllärniemi, M. Raatikainen, and T. Männistö. KumbangSec: An approach for modelling functional and security variability in software architectures. In *1st Workshop on Variability Modelling of Software-intensive Systems*, 2007.

[16] E. Niemelä, A. Evesti, and P. Savolainen. Modeling quality attribute variability. In *ENASE 2008*, Submitted.

[17] OWL. http://www.w3.org/TR/owlfeatures/, 2008. Visited January 2008.

[18] D. Pakkala, J. Perälä, and E. Niemelä. A component model for adaptive middleware services and applications. In *EUROMICRO '07: Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)*, 2007.

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

[20] P. Savolainen, E. Niemelä, and R. Savola. A taxonomy of information security for service-centric systems. In *EUROMICRO '07: Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)*, 2007.

[21] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2), 2002.

[22] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. Covamof: A framework for modeling variability in software product families. In *Proceedings of Software Product Line Conference (SPLC)*, pages 197–213, 2004.

[23] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques. *Software — Practice and Experience*, 35, 2005.

[24] TOPCASED. http://topcasedmm.gforge.enseeiht.fr/website, 2008. Visited January 2008.

[25] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The Koala component model for consumer electronics software. *Computer*, 33(3):78–85, Mar. 2000.

[26] J. Zhou and E. Niemelä. Ontology-based software reliability modeling. In *Software and Services Variability Management - Concepts, Models and Tools*, 2007.