MODELING QUALITY ATTRIBUTE VARIABILITY

Eila Niemelä, Antti Evesti and Pekka Savolainen
VTT Technical Research Centre of Finland, Kaitoväylä 1, 90571 Oulu, Finland

Keywords: Modeling, software architecture, quality attribute, variability, ontology, software product family.

Abstract: Due to the emerging service orientation of software architectures, the importance of quality aspects and the ability to manage the changing quality requirements of a service have raised the question of how to explicitly define quality requirements and how to assure that quality requirements are defined and handled in the same way by all developers involved in the development of the service. The contribution of this paper is a novel approach, which allows to define metrics for quality attributes as quality ontologies, to specify execution qualities as quality profiles according to a quality variability model and quality ontologies, and to model quality properties as an integrated part of software architecture. The Unified Modeling Language (UML) and its extension mechanisms are used for defining quality profiles. The approach is applied to reliability and security modeling and supported by an integrated tool chain developed on top of the Eclipse platform.

1 INTRODUCTION

Since the 90s the development of software intensive systems has focused on component based software architectures and software family engineering. The key issue has been the software family architecture that provides a common architecture and software components, which are used for implementing the defined architecture for a set of products (Clements, Northrop & Northrop 2001). The developed variability management practices provide solutions for defining and managing functional variability in software development (America et al. 2000, Bachmann, Bass 2001, Bosch et al. 2001). Service orientation, which is emerging in software families as well, brings new challenges by requiring techniques and mechanisms for handling quality variability at run-time (Ping, Xiaoxing & Jian 2005).

This paper focuses on quality variability management of execution qualities, such as reliability and security. In order to handle quality variability at run-time, the following assumptions have to be true; First, the quality attributes shall be defined in an unambiguous way. Second, the quality attributes have to have quantitative metrics. Third, the quality characteristics have to be explicitly defined in the architecture models. Furthermore, quality characteristics have to be measured at run-time by the mechanisms that are suitable for the case at hand. Finally, the dynamic system has to be able to make decisions concerning validity and correctness of configurations in the contexts where the reconfigurations occur. This paper introduces a novel approach that tackles the three above mentioned challenges. After that, the last two challenges can be solved by utilizing the quality attribute (QA) knowledge of individual architectural elements. Quality attributes monitoring and decision making mechanisms required for QA adaptation are out of the scope of this paper.

The quality definitions exploit ontology design principles by providing concepts, properties and rules for a quality attribute. In this paper, we use the reliability ontology as an example of a quality ontology. The quality profiles are created based on the quality variability model and the quality attribute ontologies. Quality profiles are used as predefined quality characteristics mapped to architectural elements while designing the family architecture. The proposed technique is exemplified by a tool chain developed by the authors on top of the Eclipse platform. Evaluation tools are applied for predicting whether the architecture meets the defined quality requirements.

The structure of the paper is as follows. After introduction, the related research is examined. Sections 3 and 4 introduce our approach and the developed tools. Section 5 discusses the advantages and shortcomings of the approach as well as our future work. Conclusions close the paper.

Niemelä E., Evesti A. and Savolainen P. (2008).
MODELING QUALITY ATTRIBUTE VARIABILITY.
In *Proceedings of the Third International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 169-176
DOI: 10.5220/0001759801690176

2 RELATED WORK

2.1 Architecture Evolution and Quality

Architecture is the fundamental organization of a software system embodied in its components, their relationships to each other and to the environment (IEEE 2000). Software architecture also includes the principles guiding its design and evolution. Thus, the architecture is the key asset in software family engineering; it assembles the family requirements in the means of a common structure realized as software components. Recent software systems are typically networked systems that embody service architecture. The service architecture mostly refers to the software architecture of applications and middleware, although communication technologies also create requirements and challenges for the service architectures. Therefore, a modern software family architecture shall meet the requirements set by family members (i.e. business and application viewpoint) as well as the requirements and constraints set by the applied distribution topology and communication technologies (i.e. technical viewpoint).

Several attempts have been made for managing architecture evolution. Evolution has been taken into account by designing architectures that are, e.g., maintainable, portable and modifiable. Architectural styles and patterns provide diverse support for different quality attributes (Clements, Northrop & Northrop 2001). Styles and patterns provide an implicit way to achieve the desired quality; they are selected at design time and specific evaluation methods are used for estimating how well quality requirements are met (Dobrica, Niemelä 2002). Moreover, evaluation methods are qualitative or predictive. Thus, design and quality evaluation is more or less heuristic and the results depend heavily on the expertise of the architect. Still, qualitative methods have been found useful while analyzing evolution qualities but the predictive methods that are applied to the execution qualities, still lack industrial applications.

2.2 Quality Variability Management

In (Etxeberria, Sagardui & Belategi 2007), six existing modeling approaches for specifying variation in quality attributes are compared. The results show that only some of the approaches provide support for characterization and quantitative metrics of quality attributes, and none of them allows to defining the priority levels of qualities. Furthermore, all these approaches focus on the design-time quality variability management.

In the Family Evaluation Framework (van der Linden, F. et al. 2004), the highest maturity level includes automated selection, optimization and verification of variants. The quality options are realized as variation points. The use of variation points means that quality variability is static, i.e. binding is made at design time. However, in service oriented systems, quality shall be changed in time according to the context, i.e. taking into account the external state of a system (environment and user), and the internal state of the system (i.e. capabilities, resources, regulations etc.).

Context based adaptation is studied from different viewpoints; e.g. how to adapt a service according to user's preferences, or how to adapt resources according to available networking and computing capabilities. Application-aware adaptation is part of context-awareness; it means collaboration between the system infrastructure and individual applications (Noble et al. 1997). The system infrastructure manages the resources; it monitors resource levels, notifies applications of relevant changes, and enforces resource allocation decisions. Each application independently decides how to adapt when notified. Resource management is centralized but adaptation is controlled in a decentralized way. This is a mixed controlling architecture with centralized monitoring, and un-centralized decision-making. In service oriented systems, quality variability management requires a similar kind of approach.

2.3 Ontology Orientation

Ontology-orientation refers to design and modeling methods, techniques and practices used in the creation of software systems that inherently possess the ability to understand and utilize the ontology that describes their computational surroundings and binds software to its surroundings. Ontology is a shared knowledge standard or knowledge model defining concepts, relations, rules and their instances, which comprise the relevant knowledge of a topic (Zhou 2005).

Ontology is used for capturing, structuring and enlarging explicit and tacit knowledge on a topic across, people, organizations, and computer and software systems (Gruber 1995). A simplified ontology contains only a hierarchical classification (a taxonomy) showing relationships between

concepts. Appropriate understanding of software semantics can be achieved, if the semantic properties and relations of the software are captured to a form that can be further utilized computationally. Therefore, a notion for capturing the quality characteristics of each software entity has to be explicitly specified.

3 THE APPROACH

Figure 1 depicts the overview of the approach as an activity diagram. The main steps of quality variability modeling (i.e. the activities of quality engineers and a software family architect in Figure 1) are:

- To define the ontology of a specific quality attribute (QA) based on the domain knowledge of that quality attribute.
- To define the quality profiles for the QAs based on the quality requirements of a product family, the related quality attribute ontologies and the quality variability model (see section 3.2).
- To map the quality characteristics to architectural elements.

In Figure 1, stakeholders are named on the left side and each swim-lane represents the activities (rounded rectangles) of one stakeholder and the input and output (rectangles) of each activity.

Quality engineers are responsible for defining quality attribute ontologies. Each QA ontology is orthogonal and managed separately in order to get flexibility for its evolution. Some concepts are related to each other in different QA ontologies. These relationships are also defined in ontologies. The QA ontology, i.e. a set of quality attribute ontologies, is the first model component of the approach.

A software family architect is responsible for defining the family architecture and quality variability model. The quality variability model that is the same for all QAs forms the second model component of the approach. The quality variability model is used while defining quality profiles. A

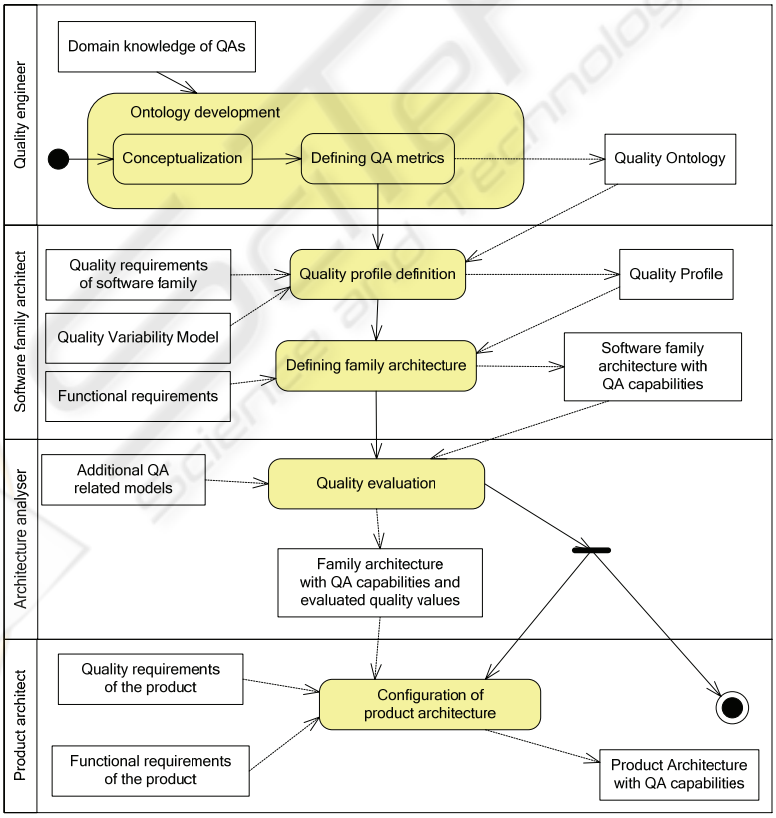

Figure 1: The overview of the approach.

quality profile includes all quality properties related to one QA. Thus, reliability and security have separate quality profiles. Each quality property defined in a profile has a standard set of characteristics given by the architect based on the quality variability model and the QA ontology. Quality profiles are used while mapping quality properties as stereotypes to architectural elements.

Architecture analyzers are responsible for the quality evaluation. Each QA requires different expertise, additional models and tools that utilize the provided design information. Finally, product architects derive the product architecture from the family architecture

3.1 Quality Attribute Ontology

Figure 2 presents a fragment of the reliability attribute ontology as a taxonomy including the concepts related to reliability metrics. The reliability metrics are mainly based on the IEEE 982.1 standard and aligned with the security metrics (Savolainen, Niemelä & Savola 2007) so that both metrics ontologies are more usable for software architects. Also combining the quality attribute ontologies is easier. Keeping quality ontologies separate made it possible to refine them concurrently by different quality engineers.

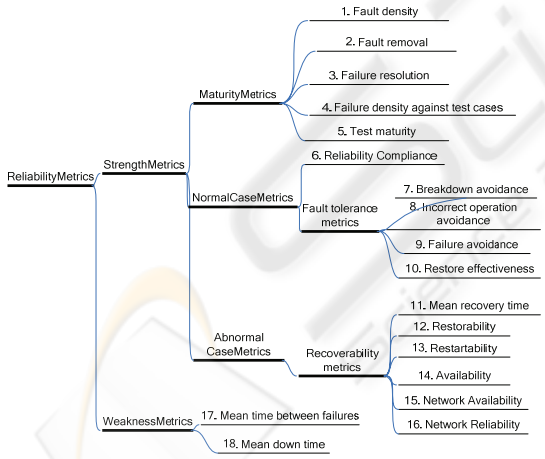

Figure 2: Reliability metrics Classes.

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. Each metrics has the following properties (Figure 3): description; purpose; target, i.e. where the metric can be used; applicability, i.e. when the metric can be used; one or more formulas; range value for the measurements; and the best value of the measurement. Rules constrain the formulas and used measurement units by defining the set of measurement targets and value ranges and the time when the metric is valid.

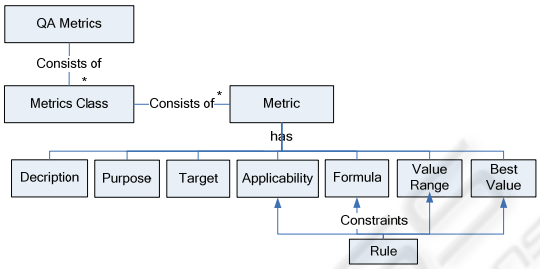

Figure 3: Concepts of QA metrics ontology.

3.2 Quality Variability Model

In a software family architecture, three types of quality variability can be identified:

- *Variability among quality attributes*; e.g. reliability is important for one family member but not relevant for the rest of family members. (optionality)
- *Diverse priority levels in quality attributes*; e.g. reliability is an extremely important property in a high-end product, while in other products only medium or low level reliability is needed. (degree)
- *Indirect variation,* i.e. functional or quality variation indirectly causes quality variation or/and vice versa. For example, improving the reliability of one component requires that all interrelated components are also at the same reliability level. (impact)

There can be one or more reasons (i.e. sources) for quality variability:

- *Subjective reasons.* The user of a software service prefers different qualities in different contexts.
- *Business reasons.* The type of application may set different quality criteria, e.g. differing measurement accuracy related to time, place and ratio for services intended for professional use as opposed to those for non-professional use.
- *Technological reasons*. Implementation technology or the amount of available resources may cause quality variation, especially when an

externally developed software or service is used without adaptation.

In order to manage quality variation, the related concepts, relations and rules have to be defined (Figure 4, Table 1). The quality variability model is used as a meta model for defining quality profiles, i.e. stereotypes of quality properties.

The *scope* of quality variation defines the size of impact the quality variation has in software architecture. There are four types of scopes; family, product, service and component. The software family type involves the widest scope; quality variation affects all family members, thus making this type of quality variation is strictly restricted. The software product type of quality variation may concern a composite of services or a service. The latter case is understood as an exception, and therefore, the scope of the product level type is defined wider than the service level.

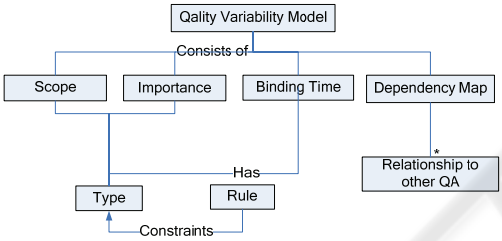

Figure 4: Concepts of the quality variability model.

The *importance* concept classifies the quality properties into three categories, which follow the change rules defined for them. The importance type 'high' indicates that the quality can't change in normal operation. However, the quality level can be lowered if the system fails to complete a service at the defined quality level, e.g. by changing resource allocation in a case in which the service can not be completed in a given time and it is essential for survival of the system. The QA management service is using the value of the importance concept while making decisions about adaptation strategies. An example of the adaptation rules for resource reservation (performance) is as follows; First, the services with the importance level 'high' have the option to select which resources, to which extent and at what time to use them. Second, the services with the importance level 'medium' have options for resources. Third, the services with the importance level 'low' get the remaining resources if still available. The level of importance can change only one level up or down, i.e. one level down in case of lacking resources, and one level up while returning to normal operation.

The *binding* time defines when variation takes place (i.e. design, assembly, start-up or run-time). This information is used while defining quality profiles, in quality evaluation and making decisions at run-time.

The *dependency* element maps one QA properties to the related QA ontology and other related QA properties, further defined according to the QA ontologies in question. Knowing the dependencies between quality variations is necessary in order to deal with the trade-offs, i.e. to make a decision regarding which variants may be in force at a certain time.

The decision-making rules are defined in a separate model because the rules depend on the system's quality goals. The information about allowed changes is defined by the concepts of the quality variability model. For example, if decreased performance means that the security level 'high' cannot be guaranteed, the decision rule finds which one, the required performance level or the required security level, should have the priority. The QA management service implemented as a middleware service makes the trade-off decision according to the information defined for both quality attributes. Thus, the decision model defines the rules for making trade-offs at run-time.

Table 1: Summary of the quality variability model.

Element	Description	Means
Source	why it is necessary to take quality variation into account	documented rationale
Scope	what quality variation can occur	constraints
Importance	how quality variation can take place	decision model, mechanisms
Binding time	when the variation can take place	constraints, decision model
Dependency	relations to other quality properties	decision model, mechanisms

4 TOOL SUPPORT

Figure 5 depicts the overview of the tool chain developed. In the first phase, QA ontologies are defined by the Protégé tool and stored in the repository in the Web Ontology Language (OWL) format. In the second phase, QA profiles are defined using the Quality Profile Editor (QPE). In the third

phase, the quality profiles are used while defining the software family architecture by mapping profiles to the architectural elements. Finally, architecture quality is evaluated by using evaluation tools and the evaluation results are stored in the architectural models.

The Quality oriented Architecting Environment (QoAE) is dependent on the exchanged data, which is transmitted via a file system. Using the existing file system ensures that data is transmitted via standard and commonly used techniques, like OWL files and UML project files of Eclipse. Since the majority of available ontology tools support OWL files, in the future it will be possible to replace Protégé by another ontology tool, without the need to change any other QoAE parts. The same applies to the used UML tool (TOPCASED).

In Figure 5 the arrows show how the data can be moved and modified. A notable thing is that while the evaluation tools cannot modify the architecture design, they can add the evaluation results to the architecture model. Detailed information about the tools is presented in (Evesti 2007; Immonen, Niskanen 2005).

From the quality variability management point of view, the QPE is the most interesting part of the environment, and therefore explained here more thoroughly. The procedure is as follows:

1. The architect opens a QA ontology in the QPE.
2. The QPE creates a list of available quality metrics based on the ontology. Each list item has a formula, a value range and the best value of measurement indicated.
3. The architect enters the quality properties defined for a system family and selects a metric for each quality property. The metrics are derived from the ontology.
4. The architect defines the dependencies between the quality properties in the same or other QA profiles.
5. Profile editor checks that properties' value lies within the valid range of the selected metric(s) and that the property name is unique.

The architect stores the valid profile.

Figure 6 depicts a snapshot of the QPE user interface. As can be seen, the QPE follows the quality variability model introduced in section 3. Except for binding time, all the elements of the quality variability model are fixed while designing a profile. Binding is made while mapping quality properties (from a profile) to architectural elements. Each quality property defined is represented as a stereotype in a model. Each stereotype is identified by a name or a quality property identification.

The operation of QoAE was evaluated by applying the developed ontology and profiles to a case example of Personal Information Repository (PIR) system, which is a reliable business-to-consumer (hospitals and patients) document delivery system. As a summary, the evaluation proved that although the tools worked correctly, several improvements are required. To be feasible the tool integration has to be seamless; the design information required for quality evaluation has to be extracted from the architectural models and automatically transformed to the form of the used evaluation tool. Therefore, a mechanism for extracting and transforming information for reliability simulation (within the RAP tool) is now under development.

5 DISCUSSIONS

The goal of our work was to create an approach and supporting tools for defining quality attribute variability and to import this design information into architectural models. Although application of the

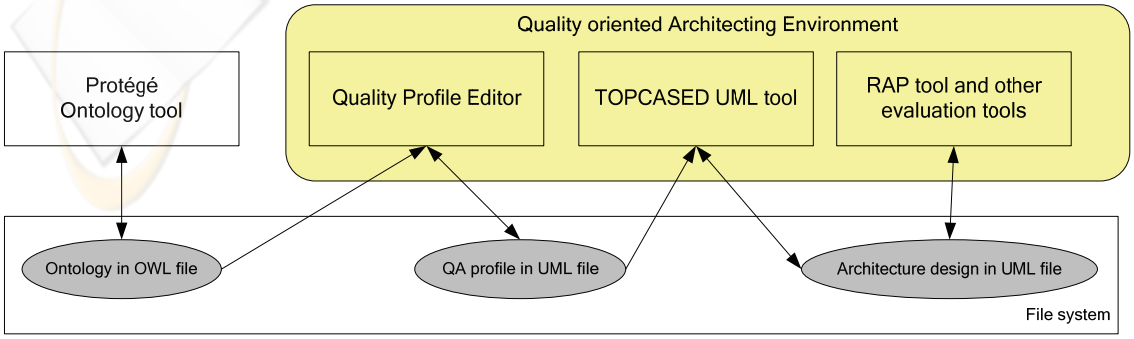

Figure 5: Overview of the QoAE.

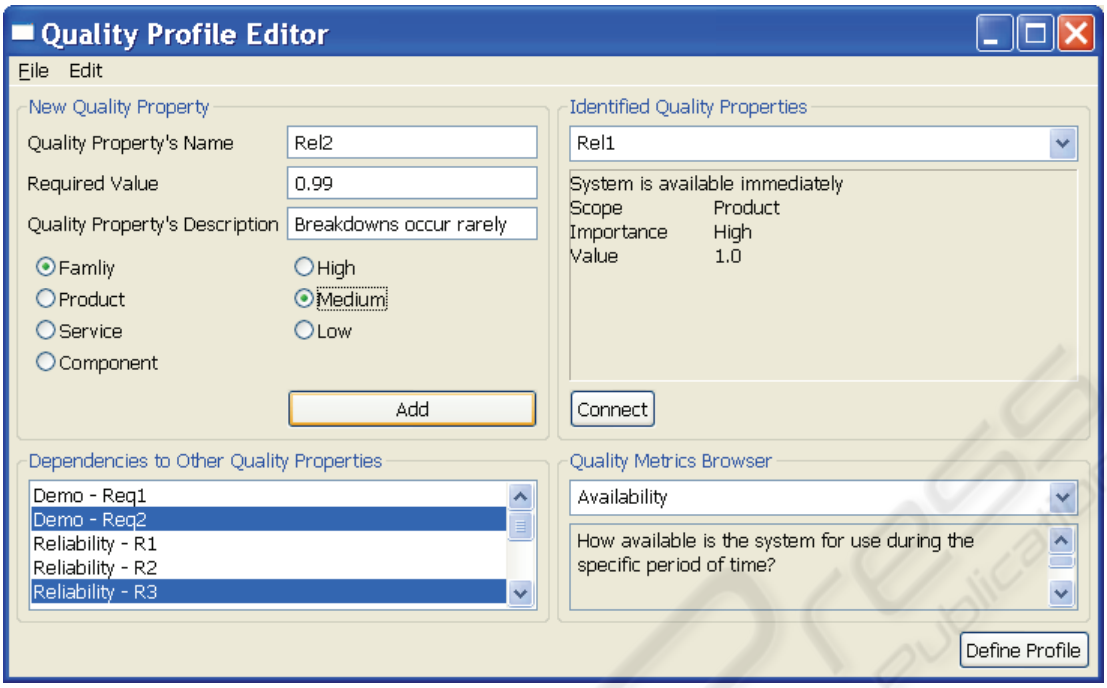

Figure 6: The GUI of the Profile Editor.

approach and supporting tool environment is still ongoing, the following observations have been made.

Defining a quality attribute ontology requires deep understanding of that quality attribute and its related methods, standards etc. The quality engineer, who is responsible for the work, also needs to know the main concepts related to software architecture design and variability management. In order to provide added value, QA ontologies shall be defined, applied, and refined in community-level activities. The presented reliability metrics ontology already includes eighteen metrics but might require refinement after their application in industrial cases. Concerning security metrics the situation is worse; only three metrics were found from standards, and other four were defined by the authors. Thus, community wide effort is especially required for developing quantitative security metrics, and techniques and tools for analyzing security at design-time and at run-time. However, we believe that the quality variation model is mature enough and can be applied in a more extensive way.

In order to make use of quality definitions in architecture design, quality ontologies have to be defined in a strict way. This means that each step has to be formalized and automated as far as possible. While there is already a range of quantitative metrics, no exact knowledge exists up to now concerning their coverage and applicability. Thus, empirical experiences are to be collected, e.g., for defining realistic estimates for prediction models.

We defined a quality variability model for creating the QPE by which quality profiles can be defined as UML profiles. This approach seems reasonable and working. However, as tool support is still evolving, attention shall be paid to ensuring an adequate maturity level of tools, in order to guarantee that models can be transformed from one to another without any need to make changes to them.

Our future work will focus on refining and combining QA ontologies (reliability and security), extending the architecting environment by domain specific ontologies, and applying the approach and developed tools to industrial cases. We assume that it will be a long journey to a point when quality attribute variability related to all execution qualities can be managed at run-time, i.e. all required monitoring mechanisms, measuring techniques, and decision models for making tradeoffs are defined, and validated. However, this paper presented the main concepts and justified why these concepts are required. The first attempt already made is the run-time performance adaptation based on service ontology and an adaptation mechanism as part of middleware (Pakkala, Perälä & Niemelä 2007).

6 CONCLUSIONS

This paper introduced an approach which combines knowledge engineering with quality driven software architecture development. The metrics of one quality attribute were introduced and used together with the quality variability model for defining a quality profile and representing quality properties in architectural models. An integrated tool environment was built for supporting the approach.

It is commonly known that the role of the software architect is an extensive one; the architect should be able not only to understand business drivers and technical issues but also to be able to organize the work and to communicate the architecture to different stakeholders. Furthermore, quality engineering, even when focusing on only one quality attribute, requires a lot of domain knowledge. One of our contributions is that our approach separates knowledge management of quality attributes from technical software engineering. Ontologies help in developing and sharing architectural knowledge, while modeling assists in achieving high-quality software architectures. We believe that this kind of approach is required for future service oriented systems, which are co-developed and delivered globally, and locally adjusted to usage contexts.

REFERENCES

America, P., Obbink, H., van Ommering, R. & van der Linden, F. 2000. CoPAM: A Component-Oriented Platform Architecting Method Family for Product Family Engineering. *Software Product Lines, Experience and Research Directions*. 28-31 August. Boston: Kluwer Academic Publishers.

Bachmann, F. & Bass, L. 2001. Managing Variability in Software Architectures. *Symposium on Software Reusability*, Toronto, Ontario, Canada, 18-20, May. Toronto, Ontario, Canada: ACM Press.

Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, H. & Pohl, K. 2001. Variability Issues in Software Product Lines. *4th International Workshop on Product Family Engineering*, Bilbao, Spain: European Software Institute. Vol. LNCS 2290.

Clements, P., Northrop, L. & Northrop, L.M. 2001. Software Product Lines: Practices and Patterns. 3rd ed. Boston, MA, USA: Addison-Wesley.

Dobrica, L. & Niemelä, E. 2002. A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering, Vol. 28, No. 7, pp. 638-653.

Etxeberria, L., Sagardui, G. & Belategi, L. 2007. Modelling Variation in Quality Attributes. *1st Internatioonal Workshop on Variability Modeling of Software-Intensive Systems*. Jan 16-18, 2007. Lero The Irish Software Engineering Research Centre.

Evesti, A. 2007. Quality-Oriented Software Architecture Development. VTT Publications 636. Espoo: VTT.

Gruber, T.R. 1995. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human-Computer Studies, Vol. 43, pp. 907-928.

Std-1417-2000. IEEE 2000. IEEE Recommended Practice for Architectural Descriptions of Software-Intensive Systems. New York: IEEE.

Immonen, A. & Niskanen, A. 2005. A tool for reliability and availability prediction. *31st Euromicro Conference on Software Engineering and Advanced Applications*. 30 Aug. - 3 Sep. 2005. Porto, Portugal: IEEE.

Noble, B.D., Narayannan, D., Tilton, J.E., Flinn, J. & Walker, K.R. 1997. Agile Application-Aware Adaptation for Mobility. *16th ACM Symposium on Operating Systems Principles*. Saint Malo, France: IEEE.

Pakkala, D., Perälä, J. & Niemelä, E. 2007. A component model for adaptive middleware services and applications. *33rd Euromicro Conference on Software Engineering and Advanced Applications*. Lubeck, Germany, 28 - 31 Aug. 2007. IEEE.

Ping, Y., Xiaoxing, M. & Jian, L. 2005. Dynamic software architecture oriented service composition and evolution. *CIT'05: 5th international conference on computer and information technology*. Shanghai, China, 21-23 September 2005. IEEE.

Savolainen, P., Niemelä, E. & Savola, R. 2007. A Taxonomy of Information Security for Service Centric Systems. *33rd Euromicro Conference on Software Engineering and Advanced Applications*. Lubeck, Germany, 29-31 August. Germany: IEEE.

van der Linden, F., Bosch, J., Kamsties, E., Känsälä, K. & Obbink, H. 2004. Software Product Family Evaluation. *Software Product Lines*, LNCS 3154. Boston, MA, USA, Aug. 30- Sep. 2. Springer-Verlag.

Zhou, J. 2005. Knowledge Dichotomy and Semantic Knowledge Management. *1st IFIP WG 12.5 working conference on Industrial Applications of Semantic Web*. Jyväskylä, Finland, 25 - 27 Aug. 2005.