Capturing the Semantics of Quality Requirements into an Intermediate Predesign Model

Vladimir A. Shekhovtsov[1], Christian Kop[2], Heinrich C. Mayr[2]

[1]Department of Computer-Aided Management Systems,
National Technical University "KhPI", Kharkiv, Ukraine
shekvl@yahoo.com

[2] Institute for Applied Informatics,
Alpen-Adria-Universität Klagenfurt, Austria,
{chris|mayr}@ifit.uni-klu.ac.at

Abstract: We present an approach to capturing the semantics of quality requirements in an intermediate predesign step residing between quality requirements elicitation and conceptual design. We propose Quality-Aware Predesign Model (QAPM) to be used at this step. In this model, the problem domain is viewed as a set of concerns. Out of this set, concerns related to quality are separated from those related to the main functionality of the system. Quality concerns are represented by hierarchical quality models incorporating quality characteristics and indicators. The semantics of both functional and quality concerns is modeled using Klagenfurt Conceptual Predesign Model (KCPM) concepts with necessary modifications. On basis of this, QAPM offers the set of concepts to represent the semantics of cross-cutting relationships between the concerns.

Key words: software quality, requirements engineering, conceptual predesign

1 Introduction

Ensuring the quality of software is one of the major problems today. This starts already with requirements elicitation. The problems in this field related to a software quality influence all later phases of the software development process. Therefore, more attention should be paid on methods to collect quality requirements. Particularly, capturing the semantics of quality requirements before performing design-time activities in a quality-driven software process is important for the following reasons:

1. Since this semantics forms an important body of knowledge that can be lost in later stages of the software process it is preferable to capture it before transition to these stages [Ch07].
2. If the representation of these requirements refers to the abstract design-time notions (classes, attributes, etc.), it can be too difficult for the system users to understand and validate and can lead to early design decisions (e.g., decomposing the system into design-time artifacts unfamiliar to the end user) [KM98].
3. This semantics reflects design-independent view of system quality in a particular domain. In this view, the representation of the semantics of quality and the functionality affected via this quality does not depend on any design notation or methodology. This way, it can serve as a basis for the description of the problem space for a domain.

To solve the above problem we follow Klagenfurt Conceptual Predesign [KM98, KM02] and Aspectual Predesign [SK05, Sh06] approaches and propose to establish an intermediate semantic model (*predesign model*) residing between quality requirements elicitation and conceptual design. The purpose of this model is to describe the notion of the software quality that can be used at different stages of the software process, and capture the quality requirements semantics in a way that can easily be understood and verified by the system users and can be mapped into different design notations. We call this model *Quality-Aware Predesign Model* (QAPM).

The rest of the paper is organized as follows. Section 2 gives background information about software quality and existing predesign approaches. Section 3 describes the proposed predesign model; a prototype tool support for this model is outlined in Section 4. Section 5 discusses the related work. Section 6 concludes the paper and shows the directions for future research.

2 Background Information

2.1 Software Quality

Quality models. Quality model is defined as "the set of characteristics and relationships between them, which provides the basis for specifying quality requirements and evaluating quality" [IS01]. In most approaches, quality models are based on hierarchies of quality attributes [Ca05, FC03, Fi03]. Top-level attributes represent general quality characteristics (functionality, reliability, etc.); bottom-level attributes represent more concrete sub-characteristics (e.g., reliability can be decomposed into fault tolerance, recoverability, etc.)

Specific quality models for different domains are proposed in the literature [FC03, Ol99, TP03], but the most important effect on industry is achieved by their standardization. The most widely known quality model standard is ISO 9126 (1991), in recent years it was extended, for example in [IS01] (ISO 9126-1 quality model) the top-level characteristics are grouped into several categories, namely, *product quality characteristics* (external and internal) and *quality in use characteristics*. A hierarchy of ISO 9126-1 product quality characteristics will be used further in this paper (it is shown in Table 1):

Top-level characteristic	**Sub-characteristics**
Functionality	suitability, accuracy, interoperability, security, functionality compliance
Reliability	maturity, fault tolerance, recoverability, reliability compliance
Usability	understandability, learnability, operability, attractiveness, usability compliance
Efficiency	time behavior, resource utilization, efficiency compliance
Maintainability	analyzability, changeability, stability, testability, maintainability compliance
Portability	adaptability, installability, coexistence, replaceability, portability compliance

Table 1: Product quality characteristics in the ISO 9126-1 quality model [IS01]

Following general practice, we assume that quality sub-characteristics are supposed to be quantified via quality measures (indicators). For example, “time behavior” sub-characteristic can be quantified via turnaround time, response time, CPU elapsed time, I/O processing time and several other indicators. This quantification process is often called an *operationalization* of the quality model [TP03]. Such operationalization is evolutional: the number of quantifiable quality sub-characteristics increases over time as more knowledge about the domain and the expected system becomes available. In the meantime, quality model should be able to express the fact that some of the quality characteristics are formulated in imprecise, qualitative form.

There are two important properties of quality models from the perspective of their implementation for the particular project.
1. Different categories of stakeholders have their own perspectives of quality, so the particular set of quality characteristics varies by different categories of stakeholders [Si97] (e.g., maintainability is perceived by the system administrators whereas time behavior is perceived by the end users etc).
2. There are interdependencies between the different elements of the quality model. Meeting requirements related to one characteristic can affect another one positively or negatively. For example, achieving the goal related to reliability can positively affect the goal related to usability but negatively – the goal related to time behavior (performance). Chung et al. [Ch00] elaborated special notation for representing these interdependencies, part of it is shown on Table 2:

Contribution	**Description**
makes (++)	affected goal is impossible to achieve without achieving an affecting goal
breaks (--)	affected goal is impossible to achieve with achieving an affecting goal
helps (+)	affected goal is easier to achieve with achieving an affecting goal
hurts (-)	affected goal is easier to achieve without achieving an affecting goal

Table 2: Interdependencies between quality characteristics [Ch00]

Quality Requirements. Different classifications of requirements are proposed in literature, some of them hide quality requirements category under the notion of non-functional requirements, but current trend (in particular, supported by the publications originated from the people involved in producing ISO 9126 standard documents, e.g., [Sy06]) is to separate these two categories. For example, the classification by Glinz [Gl07] explicitly addresses quality requirements. First, two kinds of concerns (matters of interest in a system [Fi06]) are introduced: (a) *functional concerns* related to expected system functionality and (b) *quality concerns* related to quality characteristics defined by some quality model (e.g., ISO 9126-1). Furthermore, the set of requirements is decomposed into two main categories: (1) *functional requirements* related to functional concerns; (2) *quality requirements* related to quality concerns. In this paper, we will follow this classification.

Firesmith [Fi03, Fi05] proposes the model of quality requirements directly based on hierarchical representation of the quality model. In this model, low-level elements of quality model hierarchy (indicators) form the foundation for the quality criteria and quality requirements. Every quality criterion reflects “a single aspect of quality of the system” [IS01]. Usually criteria can be seen as quality indicators connected to the particular system artifacts or its operations, e.g., for “response time” quality indicator the criterion can be "*response time for searching the customer by name*", "*response time for bank account withdrawal*" etc. Quality criteria together with threshold values form the quality requirements. For example, the requirement based on described criteria could look like this: "*response time of searching the customer by name must not exceed 1 second*". In this paper, we will use this model with necessary modifications to structure the quality requirements.

2.2 Klagenfurt Conceptual Predesign

In this section, we briefly introduce the process (Klagenfurt Conceptual Predesign, KCP) [KM98, KM02] using special semantic model (Klagenfurt Conceptual Predesign Model, KCPM) in an intermediate step of the software process residing between requirements engineering and conceptual design.

Initial Steps. The process of Conceptual Predesign starts with an identification of organizational units within the universe of discourse (UoD). These units are homogenous with respect to tasks and terminology used, e.g., they can be the departments of the given organization (sales, finances, etc.) Then, the identification of the tasks to be supported by the IS and organizational units that are occupied with that tasks is performed with the help of the special table "Organizational Unit / Task" shown on Fig.1a.

The next step is establishing a requirements elicitation/collection plan. For every task, the identification of stakeholder groups is performed (users, management, experts, etc.) who may provide relevant information concerning this task. This activity is supported with the table "Task / Information provider" (Fig.1b).

Org.unit / Task	Sales	Production	Management
Accounting	+		+
Payroll		+	

(a)

Task / Provider	Accounting	Payroll
Users	+	
Managers		+

(b)

Figure 1: Tables "Organization Unit / Task" (a) and "Task / Information Provider" (b)

Requirements Elicitation. After the plan is established, the initial requirements collection takes place. The requirements are collected in free-text form and indexed by task, stakeholder, organizational unit, date, etc. Then, this information is transferred into the semantic model (KCPM) either manually (after finding implicit information, eliminating redundancies, etc.) or via NLP text analysis [Nib02].

KCPM consists of a small set of semantic concepts. Some of these concepts allow describing the static aspects of UoD. There are two main static concepts: *thing-type* (a generalization of the conceptual notions such as entity type, class and attribute or value type) and *connection-type* (representing relationships between things described from the point of view of all of the involved sides, these different aspects of a connection type are defined using *perspectives*). The information related to the static part of KCPM is usually collected into *glossaries* or *templates* (thing-type glossary and connection-type glossary). The metamodel for the static part of KCPM is shown on Fig.2.

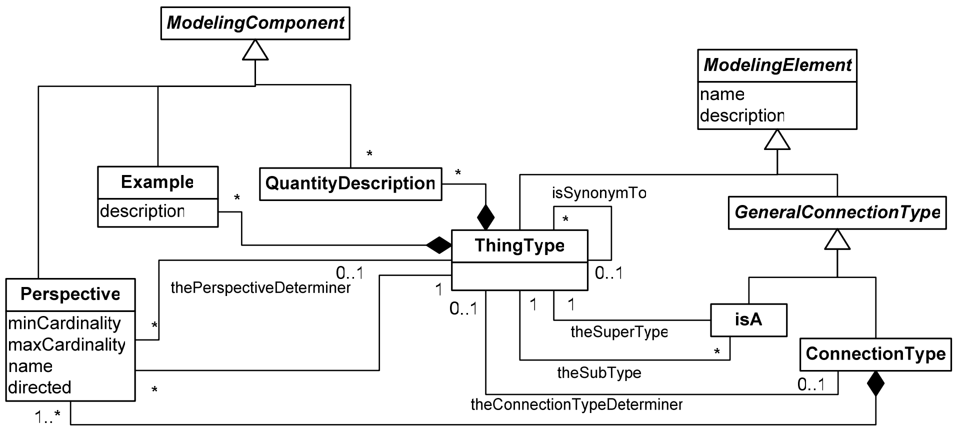

Figure 2: Static KCPM metamodel (from [Ba07a])

For behavior modeling, KCPM introduces the concepts of *operation-type* (modeling functional services called via messages), and *cooperation-type* (modeling actions performed under certain conditions). This information can also be collected into glossaries but users often find the graphical notation more convenient. An example of this notation is shown on Fig.3.

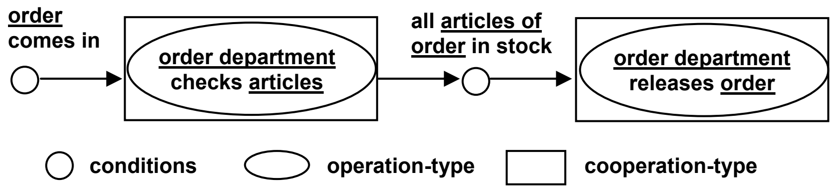
order comes in
order department checks articles
all articles of order in stock
order department releases order
conditions
operation-type
cooperation-type

Figure 3: Graphical notation for the dynamic KCPM concepts (from [KM02])

2.3 Handling Quality Requirements in Conceptual and Aspectual Predesign

KCPM Constraints. Though KCPM is built to capture the semantics of all kinds of requirements, more attention is paid to the functional ones. Non-functional requirements are mainly collected as constraints (see Fig.4 for the corresponding part of KCPM metamodel).

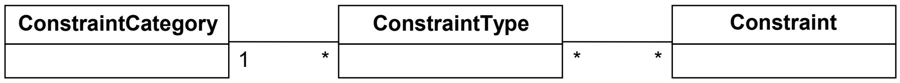

Figure 4: Part of KCPM meta-model describing the constraints [Ko02]

Each requirement represented by a constraint (e.g., *The System shall process a minimum of 8 transactions per second*) could be related to at least one *constraint type*. In [Ko02], a constraint type characterizes the place of requirement (e.g., "*time behavior*" or "*performance*") in one particular requirement classification. Every constraint type is connected to one *constraint category* corresponding to this classification (e.g., "*IEEE Std. 830-1993*"). This way of connecting constraint categories and constraint types to a constraint gives the designers more flexibility. They are allowed to define different constraint categories for different purposes ("*IEEE Std. 830-1993*", "*Web Portal Qualities*", "*My Characteristics*" etc). Within every category, it is possible to collect the types of constraints belonging to it. Once the types are defined, the designer is able to relate the collected constraints to one or more constraint types from different categories. In addition to this, if the particular performance requirement could be related to exactly one specific operation type (e.g., "*The operation "process an order" must return a result in less then 0.15 sec*") the model offers the shortcut for this case by shifting this requirement from the constraint glossary to the meta-attribute "duration" of the operation-type. This meta-attribute thus can be either the expected duration of the operation or a constraint describing the maximum acceptable duration of an operation.

Aspectual Predesign. The main goal of an Aspectual Predesign technique [SK05, Sh06] is extending KCP to deal with crosscutting concerns in the problem space. It aims at capturing the semantics of "aspectual" (crosscutting) requirements ([Ch07, Fi06]) into a semantic model (Aspectual Predesign Model, APM) similar in its purpose to KCPM. In this model, crosscutting behavior units implementing quality requirements (advices [Fi06]) are represented via operation-types; pointcuts (rules that connect advices to the places in the model where they can be called) are represented via modified connection-types. Aspectual predesign can be seen both as an extension to KCP that allows mapping the aspectual requirements and as an intermediate step of the AOSD residing between aspect-oriented requirements engineering and aspect-oriented modeling.

KCPM and APM Merging Issues. We start from discussing the weak points of the above approaches with respect to representing the software quality.

Although the treatment of functional and non-functional requirements in KCPM gives the user significant flexibility, there are still some problems. The notion of quality is not explicitly addressed in the model, there is no quality model used. As a result, no quality attributes and measures are considered in the original model (the constraints can be supplemented only with plain-text description). In addition, it is not possible to take into account different perception of quality for different stakeholders. The notion of concern and the need of separating concerns are not considered as well.

Aspectual Predesign was also not aimed at collecting the quality requirements in general: it did not employ the notion of quality. Actually, it was only possible to integrate aspectual (crosscutting) requirements with operational representation (having some action associated with them). Additionally, its support for separating concerns was simplistic: it was only possible to represent artifacts statically crosscutting whole thing-types (i.e. all their operations at once in all situations) or particular operations called in all contexts.

It is clear that these two approaches are complimentary. Whereas KCPM represents quality requirements as constraints and allows user-supplied classification of these requirements, APM allows treating the requirements as belonging to crosscutting concerns and offers some guidance in separation of these concerns and specifying the composition rules representing crosscutting relationships. It seems feasible to merge these approaches in a way that makes the resulting technique benefit from their advantages. The results of this merge are presented in the following section.

3 Quality-Aware Predesign Model

Several problems need to be solved during QAPM development: (1) allowing flexible integration of the quality model; (2) extending the KCPM metamodel to integrate complete representation of quality requirements; (3) implementing support for quality requirements evolution; (4) implementing the semantic support for separation of quality-related and functional concerns; (5) implementing support for relationships between these concerns.

In this section, we describe our approach to resolving these problems.

3.1 Integrating Quality Models into QAPM

For allowing a flexible integration of the quality–related information, we introduce two new semantic concepts for our predesign model: a *quality characteristic* and a *quality model*. The metamodel for these concepts is shown on Fig.5. It is clear that we cannot use existing concepts (such as thing types) for this purpose because they represent types of things whereas concrete quality characteristics (performance, reliability etc.) are the instances of the particular high-level concept "quality characteristic". Actually, we need to express *hierarchy of instances*, which is not possible in KCPM. A quality characteristic is a semantic concept for elements from all levels of a quality model hierarchy; a quality model represents the particular instance of this hierarchy. For quality indicators, their units of measurement are values for "value domain" meta-attribute of the quality characteristic.

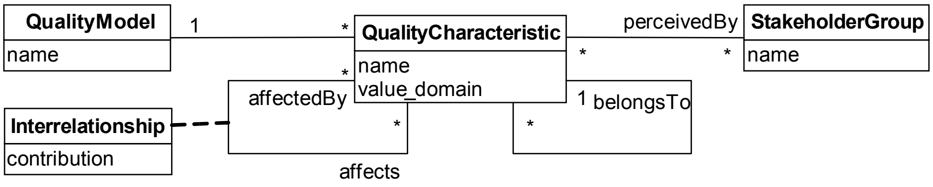

Figure 5: Part of the QAPM metamodel describing the quality model

Fig.6 contains the fragment of a quality model glossary corresponding to the ISO 9126-1 quality model. Unique identifiers (id#) are assigned to all characteristics on all levels of hierarchy to make it able to connect every quality characteristic to KCPM functional schema elements. It reflects the support for *quality requirements evolution*, making possible to express assessments of the base functionality via quality characteristics on different levels of refinement [Me07]. Different quality models correspond to different instances of glossaries; it is also possible to establish the separate meta-glossary containing the definitions of quality models. In this paper, we presuppose that only one quality model for the domain is defined.

The reason of storing all the quality model information in the predesign model reflects the “build-your-own model” paradigm of the quality model construction [FP97] and makes it possible to tailor already existing quality models for the particular problem domains. It is also possible to use existing quality model if the analysts feel it sufficient.

Quality Model: ISO 9126 for UoD: Banking			
id#	**name**	**belongs to**	**value domain**
Q01	Functionality		
Q01-1	Suitability	Q01, Functionality	
…			
Q01-4	Security	Q01, Functionality	
…			
Q04	Efficiency		
Q04-1	Time behavior	Q04, Efficiency	
Q04-2	Resource utilization	Q04, Efficiency	
…			
Q04-1-1	Response time	Q04-1, Time behavior	seconds

Figure 6: Part of the quality model glossary

To reflect the dependencies between quality characteristics and stakeholder categories, we introduce the cross-reference table shown on Fig.7. This table is filled immediately after the quality model information is introduced; it uses the stakeholder categories previously collected into the table “Task / Information Provider”.

Provider type **QC name**	Users	Managers
Q01, Functionality	+	
Q01-1, Suitability	+	
Q04-1, Resource utilization		+

Figure 7: Table “Information Provider / Quality Characteristic”

To reflect quality characteristics (QC) interrelationships we introduce another cross-reference table shown on Fig.8. The symbols used for the cells reflect the notation from [Ch00]: *makes* (++), *breaks* (--), *helps* (+), *hurts* (-). Empty entries correspond to non-existing interrelationships.

Quality Model: ISO 9126 for UoD: Banking			
Affected QC **Affecting QC**	Q01, Functionality	Q01-1, Suitability	Q04-1, Resource utilization
Q01, Functionality	▨	++	+
Q01-1, Suitability	+	▨	-
Q04-1, Resource utilization	-		▨

Figure 8: Table “Quality Characteristic Interrelationships”

3.2 Modeling Concerns and Requirements

Modeling Concerns. In our model, quality characteristics and sub-characteristics are treated (following [Gl07, Me06]) as concerns. We also follow [Me06] in distinguishing the dominant functional concern which controls the decomposition of the system and modeling all other concerns (in particular, all quality concerns) as crosscutting concerns. While the quality concerns form the primary interest of this paper, we assume that the dominant concern is the main functionality of the system. This concern defines the decomposition of the predesign model into the set of thing-types and other KCPM schema elements. We selected this asymmetric approach to concern modeling for QAPM as our first attempt to establish predesign-level separation of concerns. The multidimensional (symmetric) approach [MRA05] is also promising; we will investigate it in future. This approach treats all the concerns (including the functional ones) equally so we need to find a way to represent particular concerns and concern-independent composition rules.

In this paper, we also assume that quality concerns directly correspond to the quality characteristics and sub-characteristics in the underlying quality model (e.g., for ISO 9126 quality model the candidate concerns are "Efficiency", "Usability", "Time behavior" etc.) As a result, we do not need any special notation to represent these concerns; the quality model glossary depicted on Fig. 6 will serve the purpose of quality concern glossary as well.

It is also possible to have other functional concerns besides the dominant one. The treatment of these concerns is a target for future research.

Modeling Join Points. To be able to represent crosscutting relationships between functional and quality concerns, we need to specify a join point model [CJR06] based on captured requirements semantics. This model defines the set of all possible places where the functionality of the base concern can be extended or replaced with the functionality of the crosscutting concern. In our case, this model defines the set of all possible KCPM artifacts or their elements that can be affected with quality measures or imprecise goals. Note that some of join points refer to structural (static) KCPM artifacts (thing-types etc.) whereas some refer to behavioral (dynamic) ones (cooperation-types etc.) The elements of the joint point model are shown on Table 3.

Join point	**Description**	**Category**
thing-type	particular thing-type as a whole	structural
connection-type	particular connection-type as a whole	structural
operation-type	particular action or service call	behavioral
cooperation-type	particular sequence of actions with preconditions and post-conditions	behavioral

Table 3: Elements of the QAPM join point model

After the join point model is defined, the next step is to establish the semantics of quality requirements.

Modeling quality requirements. We propose to model quality requirements as constraints. To reflect the relationship between base and quality concerns, every such constraint will contain the references to particular quality concern (quality characteristic) and the element of dominant functional concern belonging to the joint point model (KCPM artifact). The QAPM metamodel of quality requirement is shown on Fig.9.

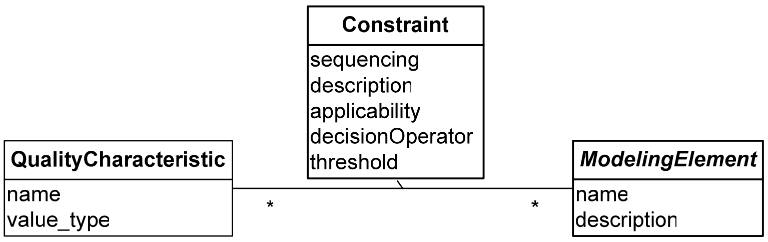

Figure 9: Part of QAPM metamodel describing the quality requirement as a constraint

As the *ModelingElement* is the root of the schema elements hierarchy in the KCPM metamodel, we decided to associate the *QualityCharacteristic* to this abstract meta-class and enhance this association using the *Constraint* associative meta-class. The meta-attributes characterizing the quality requirement constraint are as follows:

1. "*sequencing*" reflects the temporal and conditional dependencies between base and quality concern elements [Ch07]. The set of possible values reflecting these dependencies includes "before", "after", "wrap", "instead", "concurrently", "if", and "if not".
2. "*description*" contains the description of the requirement. For imprecise requirements, this meta-attribute is supposed to contain all the information available for the requirement, e.g., "the system must be secure". For refined requirements, the following two meta-attributes will be used as well.
3. "*applicability*" represents applicability condition for this requirement (e.g., "during peak hours", "during startup and shutdown", "if the system is in the safe mode" etc.)
4. "*decisionOperator*" contains the operator which needs to be applied to the threshold value to determine if the requirement is satisfied or not (e.g., "equals", "less" or more complicated operators)
5. "*threshold*" contains the threshold value.

Fig.10 shows the fragment of a constraint glossary representing imprecise (C01) and refined (C02) quality requirements. We suppose thing-type *Order* and cooperation-type *Order department checks articles* are already defined in a model.

id#	quality characteristic	functional element	sequencing	description	applicability	decision operator	threshold
C01	Q01-4, Security	D01, Order		the access must be secure			
C02	Q04-1-1, Response time	E01, Order department checks articles	wrap	the response time must be short	during peak hours	less	0.5

Figure 10: Quality requirements in the QAPM constraint glossary

Such representation is sufficient for both quantitative and qualitative requirements. The treatment of the requirements with operative representation (related to the specific action to be taken), is a target for future research, actually they can be implemented via introducing the third reference for the constraint pointing to the operation-type or cooperation-type describing the crosscutting behavior.

4 Tool Support

A prototype for tool support (*QAPMTool*) was implemented for the model. The design of the tool follows the APMTool architecture [Sh06], the difference is that the QAPMTool kernel does not use special XML-based format to exchange the information with other parts of the system, web service interface is implemented instead (Fig.11). The core of the system is based on a QAPM metamodel implemented using Eclipse Modeling Framework.

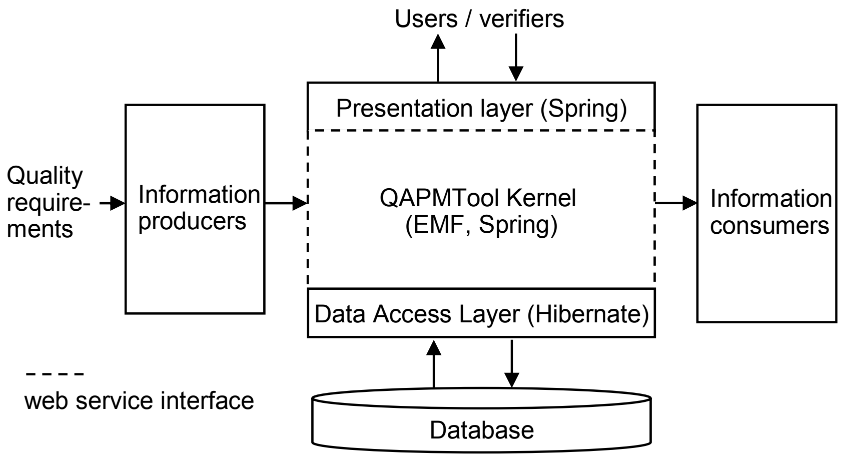

Figure 11: QAPMTool architecture

Currently, the prototype is limited to the support of the functionality of the QAPMTool kernel together with a database support. The database access is implemented using Hibernate, the domain layer uses EMF-generated code integrated with the Spring framework component support.

5 Related Work

An approach to representing the quality requirements is described in [Ba07b]. This paper describes an ElicitO tool that uses domain and quality ontologies to aid in organizing elicitation interviews. Described quality ontology contains almost exact copy of the ISO 9126 quality model hierarchy and it is used as a quality model in QAPM. Quality requirements are represented using information from both ontologies; their representation is close to QAPM as it also follows the model of Firesmith. This approach, however, is different from our technique in several aspects:

1. It is limited to requirements elicitation via interviews supported by a specific tool; no other software process activities are addressed and no formal description of the model is given.
2. Though the paper [KMZ04] shows that KCPM glossaries could be used to represent domain ontologies, so building domain ontology and KCPM can bring close results, it is not the case for ElicitO. In particular, the ElicitO domain ontology does not address behavioral aspects of the system; actually, the process of defining this ontology is not

formalized and cannot be compared to KCP.
3. The quality ontology describes a hierarchy of concrete ISO 9126 quality characteristics; no upper ontology is discussed (similar in purpose to the OAPM metamodel shown on Fig.5), so it is less flexible.
4. The representation of quality characteristics' interrelationships and integration of different perceptions of quality for different stakeholder groups is not implemented.

There are other approaches close to QAPM in purpose. In particular, the papers [Ch06, Ch07] are devoted to establishing the semantic model for crosscutting requirements. The described approach does not specifically target quality requirements, modeling crosscutting requirements instead (such requirements can be related to quality or not). Elaborated quality-driven approach targeted to Web engineering is proposed in [Ca07], it also uses representation of quality requirements close to what is proposed in our approach, but it is limited by its domain.

6 Conclusions and Future Work

In this paper, we have proposed the quality-aware intermediate model based on the KCPM metamodel. It allows capturing quality requirements semantics into glossary entries that can be verified by the end users. We showed that this model flexibly integrates specific or standard quality models taking into account specific perception of quality for different stakeholder groups and interrelationships between quality characteristics. We also showed that this model could represent evolving quality requirements with different degree of refinement. This model separates base and quality-related concerns and flexibly describes the relationships between these concerns.

The main direction of the further development of this technique is related to its prospective integration into Ontology-Based Software Engineering (OBSE) framework [He05, Ba07a]. Actually, generic quality models (like ISO 9126-1 model) are very close to ontologies (e.g., by their organization, lifecycle and usage). The natural next step is to actually define quality ontology (or the set of ontologies) representing the external driving forces of a software process and use these ontologies along with domain and process ontologies to drive the software process as required by the OBSE approach.

Acknowledgements. The authors wish to thank Denis Knjazhev for his help in implementing the prototype for QAPMTool.

Bibliography

[Ba07a] Bachmann, A.; Hesse, W.; Ruß, A.; Kop, Ch.; Mayr, H.C.; Vöhringer, J.: A Practical Approach to Ontology-based Software Engineering. In: Proc. EMISA 2007, pp.129-142.

[Ba07b] Al Balushi, T.; Sampaio, P.; Dabhi, D.; Loucopoulos, P.: ElicitO: A Quality Ontology-Guided NFR Elicitation Tool. In: Proc. REFSQ'07. Springer, 2007.

[Ca05] Carvallo, J.P.: Systematic Construction of Quality Models for COTS-based Systems. PhD Thesis. Universitat Politècnica de Catalunya, Barcelona, 2005.

[Ca07] Cachero, C. et al.: Towards a Quality-Aware Engineering Process for the Development of Web Applications. Working Papers of Faculty of Economics and Business Administration, 07/462, Ghent University, Belgium, 2007.

[CJR06] Cazzola, W.; Jezequel, J.-M.; Rashid, A.: Semantic Join Point Models: Motivations, Notions and Requirements. In: Proc. SPLAT'06, Bonn, 2006.

[Ch00] Chung, L; Nixon, B.; Yu, E.; Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer, 2000.

[Ch06] Chitchyan, R., Sampaio, A. et al.: Initial Version of Aspect-Oriented Requirements Engineering Model, AOSD-Europe report (D36): AOSD-Europe-ULANC-17, 2006.

[Ch07] Chitchyan, R., Rashid, A., Rayson, P., Waters, R.: Semantics-based Composition for Aspect-Oriented Requirements Engineering. In: Proc. AOSD'07, ACM, 2007.

[FC03] Franch, X.; Carvallo, J.P.: Using Quality Models in Software Package Selection. IEEE Software, 20(1), 2003.

[Fi03] Firesmith, D.: Using Quality Models to Engineer Quality Requirements. J. of Obj. Technology, 2(5), 2003, pp. 67-75.

[Fi05] Firesmith, D.: Quality Requirements Checklist. J. of Obj. Technol., 4(9), 2005, pp. 31-38.

[Fi06] Filman, R.; Elrad, T.; Clarke, S.; Aksit, M.: Aspect-Oriented Software Development. Addison-Wesley, 2006.

[FP97] Fenton, N.; Pfleeger, S.: Software Metrics: A Rigorous and Practical Approach. 2nd ed. PWS Publishing, Boston, 1997.

[Gl07] Glinz, M.: On Non-Functional Requirements. In: Proc. RE'07, IEEE, 2007.

[He05] Hesse, W.: Ontologies in the Software Engineering Process. In: Proc. EAI Workshop, 2005.

[IS01] ISO/IEC 9126-1, Software Engineering – Product Quality – Part 1:Quality model, 2001.

[Ko02] Kop, Ch.: Rechnergestützte Katalogisierung von Anforderungspezifikationen und deren Transformation in ein konzeptuelles Modell. Doctoral thesis, Univ. Klagenfurt, 2002.

[KM98] Kop, Ch.; Mayr, H.C.: Conceptual Predesign – Bridging the Gap between Requirements and Conceptual Design. In: Proc.ICRE'98, 1998.

[KM02] Kop, Ch.; Mayr, H.C.: Mapping Functional Requirements: From Natural Language to Conceptual Schemata. Proc. SEA'02, 2002, p. 82-87.

[KMZ04] Kop, Ch.; Mayr, H.C.; Zavinska, T.: Using KCPM for Defining and Integrating Domain Ontologies. WISE 2004 Workshops, LNCS 3307, Springer, 2004, p. 190-200.

[Me06] Meier, S.; Reinhard, T.; Seybold, C.; Glinz, M.: Aspect-Oriented Modeling with Integrated Object Models. Proc. Modellierung 2006, GI-Edition, 2006, p. 129-144.

[Me07] Meier, S.; Reinhard, T.; Stoiber, R.; Glinz, M.: Modeling and Evolving Crosscutting Concerns in ADORA. In: Proc. Early Aspects at ICSE Workshop, 2007.

[MRA05] Moreira, A.; Rashid, A.; Araujo, J.: Multi-Dimensional Separation of Concerns in Requirements Engineering. In: Proc. RE'2005, IEEE, 2005.

[Ni02] Niba, L.C.: The NIBA workflow: From textual requirements specifications to UML schemata. In: Proc. ICSSEA'02, Paris, 2002.

[Ol99] Olsina, L.: Web-site Quality Evaluation Method: a Case Study on Museums. In: ICSE 99 - 2nd Workshop on Software Engineering over the Internet, 1999.

[Sh06] Shekhovtsov, V.; Kostanyan, A.; Gritskov, E.; Litvinenko, Y.: Tool Supported Aspectual Predesign. In: Proc. ISTA'2006, LNI P-84, GI-Edition, 2006, p. 153-164.

[Si97] Siakas, K. et al.: The Complete Alphabet of Quality Software Systems: Conflicts and Compromises. In: WCTQ & Qualex'97, 1997, p.. 603-618.

[SK05] Shekhovtsov, V.; Kostanyan, A.: Aspectual Predesign. In: Proc. ISTA'2005, LNI P-63, GI-Edition, 2005, p. 216-226.

[Sy06] Suryn W. et al.: Software Quality Engineering – where to find it in Software Engineering Body of Knowledge (SWEBOK). In: Proc. ISQM & INSPIRE, 2006.

[TP03] Trendowicz, A.; Punter, T.: Quality Modeling for Software Product Lines. In: QA-OOSE'03 Workshop, Darmstadt, 2003.