Louvain School
of Management

Working paper 08/06
Extending the UML QoS framework metamodel to express quality preferences and priorities

Caroline Herssens, Ivan J. Jureta, Stéphane Faulkner

Academic Year 2007- 2008

Louvain School of Management

EXTENDING THE UML QOS FRAMEWORK METAMODEL TO EXPRESS QUALITY PREFERENCES AND PRIORITIES

Caroline Herssens
ISYS, LSM, Université catholique de Louvain, Place des Doyens 1, B-1348 Louvain la Neuve, Belgium
caroline.herssens@uclouvain.be

Ivan J. Jureta, Stéphane Faulkner
PReCISE, University of Namur, Rempart de la Vierge 8, B-5000 Namur, Belgium
iju@info.fundp.ac.be,stephane.faulkner@fundp.ac.be

Keywords: UML, QoS, service

Abstract: Optimal levels of quality attributes usually cannot be achieved simultaneously within a system. Information about system stakeholders' preferences and priorities over quality attributes is needed in order to manage tradeoffs between quality attributes both at development time and at runtime. Such information has to be integrated in non-functional requirements specification. We extend the UML QoS Framework metamodel to allow UML QoS models built from the metamodel to incorporate specification about preferences and priorities over quality attributes.

1 INTRODUCTION

Ensuring the quality of software has become a major issue in software engineering research and practice since the 1970s (B.W. Boehm and Merrit, 1978). As increasingly complex software plays a critical role in business, comprehensive and precise methods are needed to create software products and services that are safe, dependable, and efficient (Osterweil, 1996).

The International Organization for Standardization (ISO, 1986) defines software quality as the totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs. Ensuring the quality of software therefore amounts to making sure that software behavior is in line with stated and implied needs.

It is widely acknowledged that quality needs to be taken into account across the various stages of the software development process (Mylopoulos et al., 1992; van Lamsweerde, 2001). Approaches focusing on ensuring quality during the development process by guiding functional requirements modeling decisions by quality considerations, so that the latter justify the former, are termed process-oriented. In contrast, product-oriented approaches (e.g., (Donzelli, 2004; V. Issarny and Sarikadis, 1998)) evaluate the quality of already developed software products, and are particularly relevant for, e.g., component selection (C. Alves, 2005).

It is common in both approaches to use Quality of Service (QoS) models[1] to specify quality that is expected of, or is observed in the system. Apart from being used to guide decisions during development (as in requirements engineering—see, e.g., (Mylopoulos et al., 1992)), such models can be subsequently used once the system is operational. This is the case, for instance in service-oriented systems (SoS).

A *service* is a self-describing and self-contained modular application designed to execute a well delimited task, and that can be described, published, located, and invoked over a network (Papazoglou and Georgakopoulos, 2003). Services are offered by *service providers*, i.e., organizations that ensure service implementations, advertise service descriptions, and provide related technical and business support. An SoS incorporates *service composers*. A service composer receives *service requests* from human users or other systems, then discovers, selects, and coordinates the execution of services so as to fulfill given service requests.

A quality or QoS model is used in a SoS (i) by service requesters to specify the expected quality levels of service delivery; (ii) by service providers to adver-

[1] According to the relevant ISO standard (ISO, 1998), QoS refers to characteristics that contribute to the overall quality of a service as perceived by the consumer of the service.

tise quality levels that their services achieve; and (iii) by service composers when selecting among alternative services those that are to participate in a service composition. Lack of an expressive quality model (a) unnecessarily restricts the requester when defining expectations on service delivery; (b) does not allow a provider to give a rich description of how its services perform; and (c) limit the set of criteria over which a service composer compares alternative services when performing service composition.

We have studied various quality models in our research on quality-driven reinforcement learning algorithms that guide composers at service composition (see, (DBR1, ; DBR2,)). We survey them elsewhere (DBR3,). One particular limitation shared by the various models is the difficulty in specifying preferences and priorities over values defined for various quality attributes. The models therefore cannot express information necessary when managing trade offs. Indeed, it is unlikely that various quality attributes can all be satisfied to the desired extent simultaneously in a given system and at all times, so that indications are required on what quality attributes to optimize within a set of interdependent and conflicting quality attributes. Preferences and priorities defined over quality attributes allow us to specify such information. A quality model is an apparent candidate for including modeling primitives for providing these indications. Note that in SoS, trade offs need to be managed at runtime, so that a lack of such information leads to inappropriate service compositions and subsequently dissatisfied users.

The UML QoS Framework (Group, 2006) is used for the definition of quality models. Namely, the framework provides a metamodel for expressing quality considerations and an accompanying QoS profile. In this paper, we show how the UML QoS Framework metamodel can be extended to allow the creation of QoS models in which preferences and priorities information can be made explicit and subsequently used in managing at runtime. Below (§2), we overview the current UML QoS Framework metamodel. We subsequently introduce and discuss the extensions to the metamodel (§3). Finally, we close with a discussion of related work (§4), conclusions and directions for future effort (§5).

2 UML QOS FRAMEWORK METAMODEL – AN OVERVIEW

The UML QoS Framework metamodel includes different submetamodels describing the QoS extension for UML. The QoSCharacteristics package contains the elements required to define QoSCharacteristics and QoSDimensions. The QoSConstraints package comprises the modeling elements used to describe QoSContracts and QoSConstraints. The last package, QoSLevels covers components specifying QoSModes and QoSTransitions. The current UML QoS Framework metamodel is given in Figure 1. We review the metamodel below:

- **QoSCharacteristics package**
  QoSCharacteristic. A QoSCharacteristic is a description for some quality consideration, such as, e.g., latency, availability, reliability, capability. A characteristic is quantified by means of specific parameters and methods. These concepts are provided by the metaclass QoSParameter. Extensions and specializations of such elements are available with the sub-parent self-relation. A characteristic has the ability to be derived into various other characteristics as suggested by the templates-derivations self-relation. The attribute isInvariant indicates if the value of the characteristic can be dynamically updated.

  QoSDimension. A QoSDimension specifies a measure that quantifies a QoSCharacteristic. The attribute direction defines the direction (increasing, decreasing) in which it is desired that the value of the QoSDimension moves. Unit and statistiqualQualifier attributes specify, respectively, the unit for the value dimension and the type of the statistical qualifier; e.g.: maximum value, minimum value, range, mean, frequency, distribution, etc.

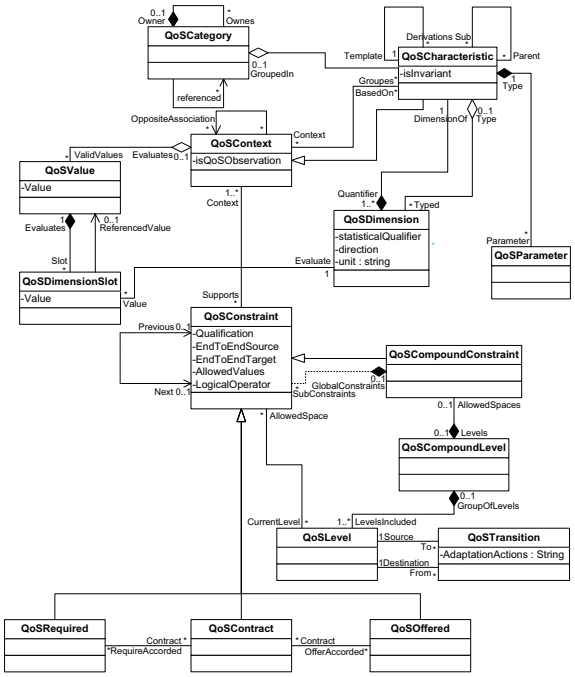
  QoSCategory. QoSCategories are used to group QoSCharacteristics related to the same abstract quality consideration or topic, such as, e.g., performance or security. While performance may group, e.g., latency and trouhgput, security might bring together, e.g., reliabiality and availability. QoSCategories are therefore not quantifiable themselves, but rely on the quantification of their components.

  QoSValue. QoSValues are instantiations of QoSDimensions that define specific values for dimensions depending on the value definitions given in QoSDimensionSlots.

  QoSDimensionSlot. A QoSDimensionSlot represents the value of QoSValue. It can be either a primitive QoSDmension or a referenced value of another QoSValue.

  QoSContext. While constraints usually combine functional and non-functional considerations about the system, QoSContext is used to describe the context in which quality expression are involved. A context includes several QoSCharacteristics and model elements. A single QoSChar-

Figure 1: The original UML QoS Framework Metamodel



acteristic defines the QoSContext for expressions whose references are only to this QoSCharacteristic. The attribute isQoSObservation defines that a QoSContext represents an environment of quality observation. The quality observation records the values of characteristics included in the relation BasedOn. This way, constraints including more than one quality characteristic can be represented. The main aim of constraints is to limit the set of allowed values of characteristics.

- **QoSConstraints Package**
  QoSConstraint. The aim of QoSConstraints is to restrict values of QoSCharacteristics. Constraints describe limitations on characteristics of modeling elements identified by application requirements and architectural decisions. Constraints rely on contexts which establishes the QoS characteristics and functional element that can be involved in the constraints. To limit allowed values, constraints put maximum and minimum values to characteristics as well as dependencies between characteristics. These quality constraints can be seen from provider's and client's point of view leading to approaches named "constraints provided" and "constraints offered". The attribute qualification refers to the nature of the constraint, with the following possible values: guaranteed, best-effort, treshold-best-effort, compulsory-best-effort, and none. Each constraint is associated to at least one QoSContext which references values related to the constraint.

  QoSRequired. Required QoSConstraints can be defined either by the provider either by the client. When the requirements are defined by the client, the provider must support the required quality that fulfill the client's required constraints. This constraint limits the set of values tha client accepts for the given characteristic. The required constraints can also be defined by the provider, in this case, the client must achieve some required level of quality to obtain the quality that the provider offers.

  QoSOffered. The set of QoSOffered by a client or a provider defines its interface—that is, it advertises the qualities for which the offered component is designed. Evidently then, quality is not guaranteed for characteristics that do not appear in QoSOffered.

  QoSContract. QoSContract is assembles client and provider constraints. In general, client required QoS need to be subsets of provider offered QoS and similarly, provider required QoS need to be subsets of client provided QoS. If no matching is possible between offered and provided constraints, the contract needs to be negotiated between parties involved.

  QoSCompoundConstraint. A QoSCompoundConstraint is a set of constraints that together rep-

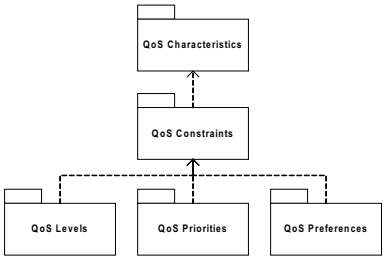


Figure 2: Extended packages

resent a constraint for one model element. Another purpose of compound constraints is to allow the representation of a global constraint composed of a set of subconstraints. This way, we can define a precedence relation between subconstraints, to represent, e.g., how to decompose a latency constraint in a set of subconstraints.

- **QoSLevelsPackage**
  QoSLevel. Depending of available infrastructure and particular algorithms, a service can be executed to several working modes; each working mode provides different qualities for the same services. A QoSLevel is intended to represent a mode of QoS that a service can support, so that a QoSLevel is associated to each of these working modes.

  QoSTransition. A QoSTransition specifies an allowed transition between QoSLevels.

  QoSCompoundLevel. A QoSCompoundLevel includes all QoSLevels involved in the quality behavior of a service.

3 EXTENDING THE METAMODEL WITH PREFERENCES AND PRIORITIES

Figure 2 presents the submetamodels of the original UML QoS Framework metamodel with our proposed extensions of the metamodel. Constraints are defined over characteristics and levels and priorities are specifications of constraints. QoSCharacteristics, QoSConstraints and QoSLevels were defined in the original metamodel while QoSPriorities and QoSPreferences are added submodels allowing to introduce, repectively, the concepts of priorities and preferences. Besides these submodels, various extensions have been added to the model in order to express all elements available in QVDP. Figures 2 and 3 summarize the extended UML QoS Framework Metamodel. Changes and extensions to the UML QoS Framework metamodel are introduced and discussed in the remainder of this section.

3.1 Priorities

When optimal values of different quality dimensions (or characteristics) cannot be simultaneously achieved, priorities over the given dimensions (characteristics) determine which dimensions (characteristics) are to be favored in optimization over others. Such priorities are defined over pairs of dimensions (respectively, characteristics) and pairs are combined to generate the QoSPriority submodel. Conditions can be binded to these priorities in order to delimit when such priorities hold. As the UML QoS Framework metamodel cannot express priorities over quality elements, the submodel presented here is an extension proposing extra classes to specify priorities over characteristics and over dimensions.

Definition. The **QoS Priority** metamodel contains a set of (conditioned) priority orders on distinct QoSDimensions, and/or a set of (conditioned) priority orders on distinct QoSCharacteristics. A priority order for QoSDimensions $\mathbf{d}_i$ and $\mathbf{d}_j$ given as $\mathbf{d}_i \overset{P}{\succeq} \mathbf{d}_j$ indicates that improving the value of $\mathbf{d}_i$ is at least as important as improving the value of $\mathbf{d}_j$. If we define a priority order $\mathbf{c}_i \overset{P}{\succeq} \mathbf{c}_j$ between two characteristics $\mathbf{c}_i$ and $\mathbf{c}_j$, then we interpret it as follows: improving any of the quality dimensions defining $\mathbf{c}_i$ is at least as important as improving any of the quality dimensions defining $\mathbf{c}_j$. $\overset{P}{\succeq}$ is a partial order; strict priority is defined as usual (i.e., $x \overset{P}{\succ} y \equiv x \overset{P}{\succeq} y \wedge \neg(y \overset{P}{\succeq} x)$). A priority order can have a condition which must be true for the priority order to apply; for instance $(\mathbf{d}_i \overset{P}{\succeq} \mathbf{d}_j)@\phi$ indicates that the order applies only if ϕ holds. A condition is written as an assertion.

We introduce the following metaclasses to integrate the QoS Priority metamodel in the UML QoS Framework metamodel:

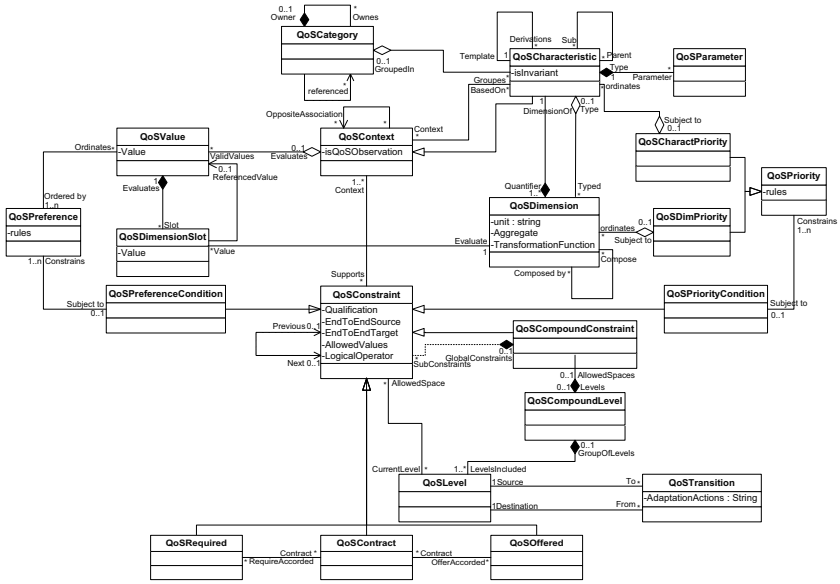
QoSPriority. The main class of this submodel is used to express rules that define priorities over characteristics or dimensions. These rules determine the order in which dimensions or characteristics are considered for improvement/optimization when tradeoffs arise. A rule defines an order relation between elements.

QoSDimPriority. and **QoSCharactPriority.** These classes are specializations of QoSPriority for priorities over, respectively, characteristics and dimensions.

QoSPriorityCondition. Conditions on priorities are constraints specifying when priorities hold. We can thus express, e.g., the priority that holds only if some value over a quality dimension is achieved.

Example. In an SoS A requester can instantiate the QoS Priority metamodel to indicate in a service re-

Figure 3: UML QoS Framework Metamodel with proposed extensions in bold



quest that it is strictly more important to optimize security characteristic than performance characteristic. The QoSPriority rules will be such that Security $\overset{P}{\succ}$ Performance, this priority is an instance of QoSCharactPriority. At the level of quality dimensions, the same request can indicate, e.g., that it is strictly more important to optimize the security level of the composition than to optimize composition time. Following the above definition, and knowing that the QoSDimension CompositionSecurityLevel is one of the dimensions defining Security, and that CompositionTime is among the dimensions defining Performance, then we have CompositionSecurityLevel $\overset{P}{\succ}$ CompositionTime. This priority over dimensions is an instantiation of the QoSDimPriority class. The priority relation can be subject to some conditions; e.g., CompositionSecurityLevel $>$ Medium. In that case, this instance of the QoSPriorityCondition is associated to the QoSPriority instance.

3.2 Preferences

In the current UML QoS Framework metamodel, preferred values for QoS dimensions are defined by indicating the direction in which the values ought to vary. This is inappropriate when complex interdependencies are present between QoS dimensions—in such cases, direction is not enough, since only some subsets of possible values are desirable (e.g., even if some higher value might be desirable alone, reaching it might lead the system to perform, e.g., insecurely). The extension proposed below allows much more freedom in defining preferred values. Moreover, we allow preferences over values to be associated to conditions under which they apply.

Definition. The **QoS Preference** metamodel contains a set of preference orders on values of QoSDimensions. A preference order is defined for a particular QoSDimension—for a dimension $\mathbf{d}_i$, the following $((\cdot) \overset{U}{\succeq} (\cdot))_{\mathbf{d}_i}$ is a preference order which indicates that the value on the right of $\overset{U}{\succeq}$ is at least as preferred/desirable than the value on the right of $\overset{U}{\succeq}$. Both $(\cdot)$ follow the syntax and semantics of either Λ^c or Λ^d (see, Definition 3.2 below) depending on whether the QoSDimension is continuous or discrete. $\overset{U}{\succeq}$ is a partial order; strict preference is defined as usual (i.e., $x \overset{U}{\succ} y \equiv x \overset{U}{\succeq} y \wedge \neg(y \overset{U}{\succeq} x)$). A preference order can have a condition which must be true for the preference order to apply; for instance $((\cdot) \overset{U}{\succeq} (\cdot))_{\mathbf{d}_i} @ \phi$ indicates that the order applies only if ϕ holds. A condition is written as an assertion.

Definition. In a preference order $((\cdot) \overset{U}{\succeq} (\cdot))_{\mathbf{d}_i}$, values (i.e., $(\cdot)$) are defined using syntax and semantics which differ depending on whether the type of the QoSDimension is continuous or discrete. If continuous, the semantics is defined in terms of an interpretation $(\mathbb{C}, \cdot^{cI})$, which uses the continuous domain $\mathbb{C}$, and an interpretation function $\cdot^{cI}$ which associates with each v a v^{cI} in $\mathbb{C}$. If discrete, the domain of the interpretation is a set $\mathbb{D}$ which includes all allowed discrete values, and the interpretation function $\cdot^{dI}$ associates with each e a e^{dI} in $\mathbb{D}$. Below, the syntax,

semantics, and syntax rules are given for the continuous and discrete case.

dimension *is continuous.*	
Syntax	Semantics
v	$v^{cI} \in \mathbb{C}$
$\neg v$	$\mathbb{C} \setminus v^{cI}$
$(\geq v)$	$\{v'^{cI} \mid v'^{cI} \geq v^{cI}\}$
$\neg(\geq v)$	$\mathbb{C} \setminus (\geq v)^{cI}$
$(\leq v)$	$\{v'^{cI} \mid v'^{cI} \leq v^{cI}\}$
$\neg(\leq v)$	$\mathbb{C} \setminus (\leq v)^{cI}$
$(\leq v) \vee (\geq v)$	$(\leq v)^{cI} \cup (\geq v)^{cI}$
$(\leq v) \wedge (\geq v)$	$(\leq v)^{cI} \cap (\geq v)^{cI}$
Syntax formation rules	
$\lambda^c ::= v \mid (\geq v) \mid (\leq v)$	
$\Lambda^c ::= \lambda^c \mid \neg\Lambda^c \mid \Lambda^c_i \vee \Lambda^c_j \mid \Lambda^c_i \wedge \Lambda^c_j$	
$\mathbf{v} ::= \Lambda^c$	

dimension *is discrete.*	
Syntax	Semantics
e	$e^{dI} \in \mathbb{D}$
$\neg e$	$\mathbb{D} \setminus e^{dI}$
E	$E^{dI} \subseteq \mathbb{D}, E^{dI} = \{e^{dI} \mid \forall e \in E\}$
$\neg E$	$\mathbb{C} \setminus E^{dI}$
$E_i \vee E_j$	$E^{dI}_i \cup E^{dI}_j$
$E_i \wedge E_j$	$E^{dI}_i \cap E^{dI}_j$
Syntax formation rules	
$\lambda^d ::= e \mid E$	
$\Lambda^d ::= \lambda^d \mid \neg\Lambda^d \mid \Lambda^d_i \vee \Lambda^d_j \mid \Lambda^d_i \wedge \Lambda^d_j$	
$\mathbf{v} ::= \Lambda^d$	

We use the following metaclasses to introduce the QoS Preference metamodel in the UML QoS Framework metamodel:

QoSPreference. The purpose of QoSPreference metaclass is to sort values of dimensions. The sorting is established by rules determining a precedence order between values. Rules can delimit precedence over disjoint sets of value and not only following a modality as previously proposed with the direction attribute.

QoSPreferenceCondition. The QoSPreferenceCondition class is a specialization of the QoSConstraint class. It indicates when the preference on values holds.

Example. A requester will instantiate the QoSPreference submodel in order to specify the prefered values of the NetworkTime QoSDimension. The QoSPreference rules will be such that $((\leq 350ms) \wedge (\geq 150ms)) \overset{\cup}{\succ} (> 350ms)$. This preference is an instance od the QoSPreference class. This preference can be subject to a constraint as: ConnectionFailureProbability $\leq$ 0.02. This constraint is represented in the UML QoS Profile by means of an instance of the QoSPreferenceCondition class.

3.3 Additional Adjustments

3.3.1 Compose-composed relationship.

In the original metamodel, it is possible to compose a characteristic from other characteristics with the relation sub-parent. If we intend to allow QoSDimensions to be aggregated, we need to allow a QoSDimension to be composer of QoSDimensions. We need aggregation in order to provide, e.g., QoSDimensions whose values summarize the values of other QoSDimensions. We thus introduce the compose-composedby self-relationship to the QoSDimension metaclass. We base that extension on the following definition.

Definition. $aggregateQoSD : \mathbb{P}(\{\mathbf{d}\}) \times \{\alpha\} \longmapsto \{\mathbf{d}\}$ is the function that maps a set of QoSDimension instances onto a QoSDimension whose values are obtained by applying an aggregation procedure α on values of the originating set of QoSDimensions. α specifies how values are aggregated (α can be written as, e.g., an algorithm). ($\mathbb{P}$ denotes powerset.)

Example. ServiceLatency can be defined a QoSDimension whose values are obtained by summing the time needed to communicate the service request over the network and to receive the desired output (measured using NetworkTime), the time required for composing the services needed to fulfil the request (CompositionTime), and the time needed to execute the composition (ExecutionTime).

3.3.2 Aggregate and TransformationFunction

In the original UML QoS Framework metamodel, the value calculation is specified in the attribute statisticalQualifier of the QoSDimension class. That approach only allows us to define the modality under which the value of a dimension can be calculated on a given value. In order to define a dimension from other dimensions, the value calculation has to be more expressive. We augment expressiveness by replacing statisticalQualifier with two attributes: Aggregate which defines from which other dimensions the value is calculated and references the procedure used in the aggregation, and TransformationFunction which gives the detail of the procedure (i.e., α) used to compute the value of the dimension.

Definition. Aggregate is given for a QoSDimension that is an aggregate of other QoSDimensions. This attribute indicates the aggregation procedure and the QoSDimensions aggregated to obtain the value of the QoSDimension.

Definition. TransformationFunction defines the algorithm or formula used to transform the data obtained by measurement into the value reported for the dimension (e.g., average, moving average, etc.).

Example.The Aggregate attribute of the NetworkTime QoSDimension can take the following: ({ SendTime, ReceiveTime }, AggregateSum).

Example. The TransformationFunction defining how to combine these elements to obtain the value of the dimension can be instantiated by: $\frac{1}{100}\sum_{i=1}^{100}(SendTime_i + ReceiveTime_i)$.

4 RELATED WORK

The aim of QoS models is to account for quality considerations both during the development process and once the system is deployed. In the former case, system stakeholders define quality requirements over quality dimensions of the QoS model; at runtime, quality is measured over the quality dimensions so that quality is assessed and, if needed, actions are taken to improve quality. Various QoS models have been proposed—we have cited those relevant to our efforts in this paper (G. Brahnmath and Burt, 2002; D'Ambrogio, 2006; Frolund and Koistinen, 1998; Keller and Ludwig, 2003; Loyall et al., 1998; Maximilien and Singh, 2004; Group, 1997; Group, 2006; Skene et al., 2004; Tosic et al., 2002; Zeng et al., 2003; Zhou et al., 2004).

The UML QoS Framework proposed in (Group, 2006) is used for the definition of quality models. It covers all usual and most specific constructs encountered in QoS models. In this section, we highlight similarities between the UML QoS Framework and other models for the three packages: characteristics, constraints and levels. Next, we summarize why all packages are not overviewed in all models and finally, we make references to QoS models about our extensions.

Quality models refer all to the QoSCharacteristic package, with constructs specifying characteristics, dimensions and values, e.g.:

- DAML-QoS, proposed in (Zhou et al., 2004), instanciates a QoSCharacteristic with a QoS Property while a QoSDimension is represented with a metric enabling to quantify a property;
- Maximilien and Singh propose in (Maximilien and Singh, 2004) a model representing the statistical qualifier attribute of the QoSDimension metaclass with an element named AggregateQuality.

Some models relate also to the QoSConstraints package, allowing to specify directions of preferences over values, e.g.:

- The informal quality model of Zeng (Zeng et al., 2003) considers that a value is either minimized, either maximized;
- Matchmaking in DAML-QoS (Zhou et al., 2004) instantiates QoSConstracts with a match between QoS required by a client and QoS offered by a provider;
- QML (Frolund and Koistinen, 1998) gives the possibility to fix constraints over dimensions and to bind these with a strength level.

Finally, some models refer to the QoSLevel package, e.g.:

- WSLA (Keller and Ludwig, 2003) introduces the QoSLevel concept with service level objectives reached under a given level of resources.

All modeling possibilities of the UML QoS Framework do not appear in each QoS model. Indeed, some models tempt to limit constructs expressiveness to propose simple QoS definition while utilization context of other models do not justify the availability of each concept, e.g.:

- The QoSContext metaclass is not encountered in most models, its representation is a challenge and not always necessary in the context of the model;
- The QoSLevel package is not overviewed in all models because it suggests a compulsory selection over a modality. Moreover, its definition can be simplified to an *available* attribute of the QoSDimension class.

Extensions that we propose to the UML QoS Framework in this paper are met in some models, e.g.:

- The priority submodel is introduced by Zeng in (Zeng et al., 2003). It is proposed with an aggregation function of dimensions, each of them associated to a weight. This weighting authorizes a precedence order between dimensions;
- The QoSPreference specification is encountered in WSLA (Keller and Ludwig, 2003) where wished values are defined by tresholds against which SLA parameters are compared;
- Composition of new dimensions based on other dimensions is met in Maximilien and Singh (Maximilien and Singh, 2004) whose define AggregateQuality as a dimension computed with other qualities;
- The DAML-QoS model (Zhou et al., 2004) proposes Aggregate and TransformationFunction concepts with a Function defining how to obtain the value of a dimension.

Most elements that we add to the UML QoS Framework are introduced in other models but with a limited expressiveness. In our extensions we permit to define these elements with a larger set of possibilities, e.g.; priorities and preferences can be associated to conditions restricting applicable cases.

5 CONCLUSIONS AND FUTURE WORK

Ensuring the quality of software lines software behavior with its requirements. To take into account quality in SOS, QoS must be fully specified. This specification concerns service providers, requesters and composers and allow them to publish their possibilities,

to express their needs or to select the most suitable alternative.

The UML QoS Framework (Group, 2006) is used for the definition of quality models, it covers most modeling elements introduced in existing service-oriented quality models (G. Brahnmath and Burt, 2002; D'Ambrogio, 2006; Frolund and Koistinen, 1998; Keller and Ludwig, 2003; Loyall et al., 1998; Maximilien and Singh, 2004; Group, 1997; Skene et al., 2004; Tosic et al., 2002; Zeng et al., 2003; Zhou et al., 2004). This metamodel enables numerous possibilities but we suggest some extensions to it. We propose to complete this metamodel with constructs expressing new elements. Our mains additions concern priorities and preferences information. We provide formal definitions and concrete examples to illustrate all new elements. These extensions cover these missing aspects while enable a simplified choice for service composition and selection. Indeed, priorities among qualities and preferences over values are necessary at the time of selection of services entering in a composition.

With the highlighting of these new constructs, they can be included in quality-driven reinforcement learning algorithms (DBR1, ; DBR2,). This addition leading to an optimal selection during services composition.

REFERENCES

B.W. Boehm, J.W. Brown, H. K. M. L. G. M. and Merrit, M. (1978). Characteristics of software quality. North-Holland, Amsterdam.

C. Alves, X. French, J. C. A. F. (2005). Using goals and quality models to support the matching analysis during cots selection. In *Proc. Int. Conf. on COTS-Based Software System, 146–156*.

D'Ambrogio, A. (2006). A model-driven wsdl extension for describing the qos of web services. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 789–796, Washington, DC, USA. IEEE Computer Society.

DBR1. To ensure double-blind reviewing, we omit this reference.

DBR2. To ensure double-blind reviewing, we omit this reference.

DBR3. To ensure double-blind reviewing, we omit this reference.

Donzelli, P. (2004). A goal-driven and agent-based requirements engineering framework. *Requirements Engineering*, 9:16–39.

Frolund, S. and Koistinen, J. (1998). Qml: A language for quality of service specification. Technical report, HP Laboratories, Palo Alto, California.

G. Brahnmath, R.R. Raje, A. O. M. A. B. B. B. R. and Burt, C. (2002). A quality of service catalogue for software components. In *Proc. Southeastern Softw. Eng. Conf.*

Group, O. M. (1997). The corba trading services.

Group, O. M. (2006). Uml profile for modeling qos and fault tolerance characteristics.

ISO (1986). Qml: nt. standard iso 8402. quality – vocabulary. Technical report, Int. Org. for Standardization.

ISO (1998)). Cd15935 information technology: Open distributed processing—reference model—quality of service. iso document iso/iec jtc1/sc7n1996. Technical report, Int. Org. for Standardization.

Keller, A. and Ludwig, H. (2003). The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network Systems Management*, 11(1).

Loyall, J. P., Schantz, R. E., Zinky, J. A., and Bakken, D. E. (1998). Specifying and measuring quality of service in distributed object systems. In *Proceedings of the International Symposium on Object-Oriented Real-Time Distributed Computing*.

Maximilien, E. M. and Singh, M. P. (2004). Toward autonomic services trust and selection. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'04)*.

Mylopoulos, J., Chung, L., and Nixon, B. (1992). Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng.*, 18(6):483–497.

Osterweil, L. (1996). Strategic directions in software quality. *ACM Comput. Surv.*, 28(4):738–750.

Papazoglou, M. P. and Georgakopoulos, D. (2003). Service-oriented computing. *Commun. ACM*, 46(10):24–28.

Skene, J., Lamanna, D. D., and Emmerich, W. (2004). Precise service level agreements. In *Proceedings of the International Conference on Software Engineering (ICSE'04)*.

Tosic, V., Esfandiari, B., Pagurek, B., and Patel, K. (2002). On requirements for ontologies in management of web services. In *Proceedings of the International Workshop on Web Services, e-Business, and the Semantic Web (WES'02)*.

V. Issarny, C. B. and Sarikadis, T. (1998). Achieving middleware customization in a configuration-based development environment: Experience with the aster prototype. In *Proc. Int. Conf. Config. Distrib. Syst., 207–214*.

van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. In *Proceedings of the International Symposium on Requirements Engineering (RE'01)*.

Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q. Z. (2003). Quality driven web services composition. In *Proceedings of the International World Wide Web Conference (WWW'03)*.

Zhou, C., Chia, L.-T., and Lee, B.-S. (2004). Daml-qos ontology for web services. In *Proceedings of the International Conference on Web Services (ICWS'04)*.