Software Qual J (2009) 17:65–98
DOI 10.1007/s11219-008-9059-2

A comprehensive quality model for service-oriented systems

Ivan J. Jureta · Caroline Herssens · Stéphane Faulkner

Published online: 10 September 2008

Abstract In a service-oriented system, a quality (or Quality of Service) model is used (i) by service requesters to specify the expected quality levels of service delivery; (ii) by service providers to advertise 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. Expressive quality models are needed to let requesters specify quality expectations, providers advertise service qualities, and composers finely compare alternative services. Having observed many similarities between various quality models proposed in the literature, we review these and integrate them into a single quality model, called QVDP. We highlight the need for integration of priority and dependency information within any quality model for services and propose precise submodels for doing so. Our intention is for the proposed model to serve as a reference point for further developments in quality models for service-oriented systems. To this aim, we extend the part of the UML metamodel specialized for Quality of Service with QVDP concepts unavailable in UML.

Keywords Service-oriented computing · Quality model · Preferences · Priorities · UML

1 Introduction

Quality has been variously defined as value, conformance to specifications, conformance to requirements, fitness for use, loss avoidance, or achieving and/or exceeding customer

I. J. Jureta (✉) · S. Faulkner
Information Management Research Unit (IMRU), University of Namur,
8, rempart de la vierge, 5000 Namur, Belgium
e-mail: ivan.jureta@fundp.ac.be

S. Faulkner
e-mail: stephane.faulkner@fundp.ac.be

C. Herssens
Information Systems Research Unit, Université de Louvain, 1, Place des Doyens,
1348 Louvain-La-Neuve, Belgium
e-mail: herssens@isys.ucl.ac.be

expectations (see, e.g., Reeves and Bednar 1994 for a business-oriented overview). Reasoned, structured, and systematic action taken to achieve desired quality—i.e., *quality management*—has been an active area of enquiry in various fields, most notably business (e.g., Feigenbaum 1951; Juran 1951; Deming 1982; Ishikawa 1985; Gravin 1988). In relation to software engineering, the International Organization for Standardization (ISO 1986) and the IEEE (IEEE 1989) define 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 has become a major issue in research and practice since the 1970s (Boehm et al. 1978). As increasingly complex software plays a critical role in business and other areas of life, it is clear that comprehensive and precise methods and tools are needed to create and run software that is, among other, safe, dependable, and efficient (Osterweil 1996). Service-oriented systems (e.g., Papazoglou and Georgakopoulos 2003; McIlraith and Martin 2003) intended for enabling the Semantic Web (Berners-Lee et al. 2001; Shadbolt et al. 2006) are no exception.

Service-oriented systems are one relevant current response to the increasing complexity of computing systems and the variability of their operating environments (e.g., Tennenhouse 2000; Kephart and Chess 2003; IBM 2005). 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 (McIlraith and Martin 2003; Papazoglou and Georgakopoulos 2003). Services are offered by *service providers*, i.e., organizations that ensure service implementations, supply service descriptions, and provide related technical and business support. A *service-oriented system* (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 fulfil a given service request. SOS fit well the Semantic Web, i.e., a next-generation of the World Wide Web on which services' properties, capabilities, interfaces, effects, and qualities, and data exchanged between services are described in an unambiguous, machine-understandable form. Within efforts towards the realization of the Semantic Web, ontologies prove a particularly relevant means for enabling the sharing, understanding, and automated processing of data about, and exchanged between, heterogenous services available on the Web. It is now well established that the aims of automated service discovery, access, composition, and management cannot be achieved without expressive ontologies (McIlraith et al. 2001; Horrocks 2002; Staab and Studer 2004; Battle et al. 2005).

An ontology is a specification of a conceptualization (Gruber 1993). Among the various ontologies relevant for a SoS—including those for, e.g., interfaces, capabilities, and behaviors of services—an ontology to describe quality attributes (such as, e.g., security, safety, performance) and reason thereon is necessary. A quality (or Quality of Service, i.e., QoS) ontology is used (i) in service requests to specify expectations on quality levels to achieve when fulfilling the request; (ii) by service providers to advertise quality levels that their services can achieve; and (iii) by service composers to select among alternative services, those that are to take part in fulfilling the service request. Limited expressivity of a quality ontology (a) unnecessarily restricts the requester when defining expectations on service delivery; (b) does not allow a service 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.

Various SOS quality ontologies proposed in the literature integrate concepts and constructs intended for the representation of similar aspects of a system's quality. For instance, all admit the need for metrics, so that all include constructs for the definition of a metric's identifier, unit, observed and desired values, and so on. Such apparent similarities

(reviewed in Sect. 4) led us to conclude that an effort is needed for their comparison and integration within a comprehensive framework. Knowing that work on SOS quality ontologies will further evolve, such a framework is intended to (i) integrate the previous results within a comprehensive and consistent quality model from which the various prior ontologies can be derived; (ii) assist in future work on SOS quality conceptualization by facilitating the positioning of new modeling primitives w.r.t. available ones; and (iii) to make it clear how various proposed ontologies overlap, and thus, what constructs are agreed upon as necessary when modeling quality. The framework is therefore not an ontology per se, but a model which integrates various relevant concepts and constructs for conceptualizing quality in SOS, and which is instantiated when defining ontologies.

The proposed model has two salient characteristics. First, it does not itself define particular qualities (such as, e.g., availability, reliability, safety, security, adaptability, maintainability)—instead, it lets the modeler decide what sets of measures that can be collected over the given SOS are aggregated (and how they are aggregated) to define such qualities. In this respect, the model is not an attempt at settling the debates on what a universal definition of, e.g., usability would be. Second, it integrates two submodels that are novel w.r.t. available quality ontologies—namely, a submodel for specifying priority between qualities, and a submodel for specifying the dependencies between qualities (i.e., how one quality varies when another one varies). Both submodels are particularly important for managing tradeoffs at runtime: without the former, it is unclear what quality to optimize when the optimization of some quality harms the degree of satisfaction of another; without the latter, we do not have an explicit representation of how qualities interact (i.e., we do not know how some variation of a quality affects the degree of satisfaction of others).

The proposed quality model has been validated in two realistic application settings. We have used part of the quality model elsewhere (Jureta et al. 2007a, b) to specify the quality information in service requests, to characterize the quality of individual services, and to define quality criteria used when comparing alternative services within a reinforcement learning algorithm for automated service composition. Herein we present a case study performed in cooperation with the European Space Agency (ESA). The case study illustrates how the proposed model has been used to represent quality information about a service that ESA makes available to researchers. We describe the case study in detail below (Sect. 2). The quality model is then presented and exemplified through the case study (Sect. 3). Source quality models used to define the proposed model are then reviewed (Sect. 4). Among the related efforts we discuss, we place particular attention on the part of the Unified Modeling Language specialized for QoS (Object Management Group 2005). We compare it in detail with our quality model, extend the UML QoS metamodel with novel primitives, and show how the extended model is used within the case study (Sect. 5). The paper closes with a discussion of the proposed model (Sect. 6), along with conclusions and pointers for future work (Sect. 7).

2 Motivation and case study

2.1 Motivation

We have highlighted earlier that expressive quality models are necessary to engineer high-quality systems. Our principal motive for the present work stems from our observation (discussed in detail later on Sect. 4) that currently available quality models cannot be used

to describe considerations clearly relevant when representing and reasoning about quality. Namely, they are of limited use when preferences and priorities are to be written down in a quality model. In other words, available models have difficulties expressing evidently relevant quality information. For example, we need to express in a quality model that it is preferred for the value of a quality metric to be in one given range than in another, whereby satisfying this preference is less important than maintaining some values for overall response time. In the said requirement, both preferences and priorities intervene. Moreover, we may consider this requirement conditionally on the satisfaction of another requirement—we therefore require not only a quality model which allows preference and priority orders to be expressed, but also conditional preferences and priorities. Such considerations are beyond the expressivity of most available quality models, as we show later on (Sect. 4). We cannot hope to engineer high quality systems if we cannot accommodate quality-related requirements that come naturally to engineers and future users.

2.2 Case study

ESA program on Earth observation allows researchers to access and use infrastructure operated and data collected by the agency. The infrastructure comprises a high performance computing cluster, significant storage capacity, and services accessible through the Web.[1] The platform, called "ESA Grid Processing on-Demand" can be used to access data collected by the Envisat ESA satellite, deployed to measure the atmosphere, ocean, land, and ice over a five year period. It is also possible for researchers to execute their own algorithms on the data.[2]

The case study focuses on the information provided by the MERIS instrument on the Envisat ESA satellite. MERIS is a programmable, medium-spectral resolution, imaging spectrometer operating in the solar reflective spectral range. MERIS is used in observing ocean color and biology, vegetation and atmosphere and in particular clouds and precipitation. In relation to MERIS, web services are made available by the ESA for access to the data the instrument sends and access and use of the associated computing resources.

We are interested in the remainder in the service called "MERIS MVGI Regional", which provides vegetation indexes for a given region of the globe. A vegetation index measures the amount of vegetation on the Earth's surface. MERIS is a particularly relevant for the acquisiton of such data for it increases precision over comparable instruments. The service which accesses the MERIS data generates regional maps for specific time periods. A visual example of what the service provides to the user is given in Fig. 2. Below, we briefly review the functional requirements that the service satisfies. We then consider quality requirements.

2.2.1 Functional requirements

The service processes MERIS data and extracts the vegetation index. The processing can be selected for any time range (with the start of the satellite mission as the earliest time point); an option is available to delimit the region of the world of interest. The graphical user interface used to access the service is shown in Fig. 1. Figure 2 illustrates the visualization of the output obtained for the Senegal region. The following are the required inputs of the service: *Time range*, *Bouding box* (to select a region of the globe), *Dataset*, *Publish site*, and *Projection type*.

[1] http://gpod.eo.esa.int.

[2] http://eopi.esa.int/esa/esa?type=file&ts=1186489893497&table=aotarget&cmd=image&id=1260.

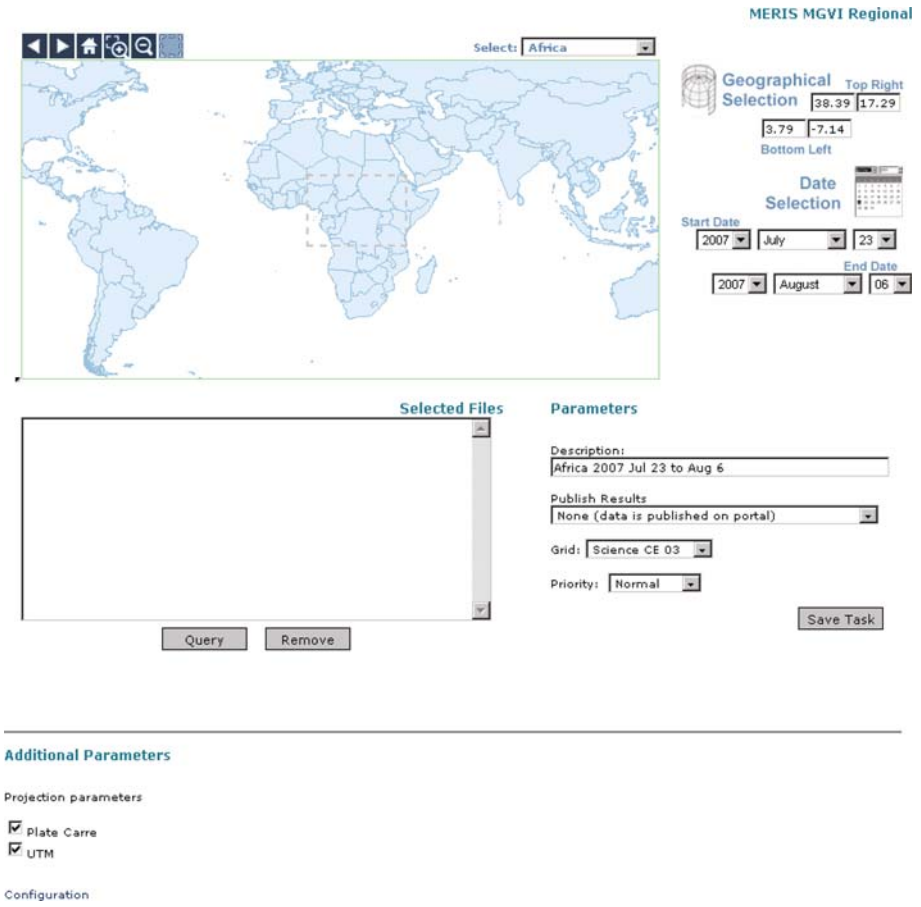


Fig. 1 Graphical user interface of the ENVISAT/MERIS MGVI web service

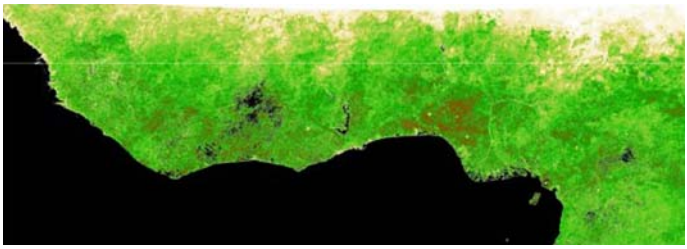


Fig. 2 An illustration of the result provided by the ENVISAT/MERIS MGVI web service

2.2.2 Quality requirements

Due to the calculations executed by the service and its parallel use, expected delays and availability are relevant quality considerations from the user's perspective. To make this information available to the users, quality considerations need to be expressed and measured during the execution of the service. We focus on three such considerations,

namely availability, reliability, and latency. For now we define them as follows. We then return to each throughout the paper and illustrate how they are defined using the model we propose.

- Availability indicates the duration when a component is available for quaries. Its value in percent is obtained as follows (Ran 2003):

$$A = \frac{upTime}{upTime + downTime}$$

  Historical data on the service's operation indicate that the values are higher than 94%.
- Reliability is a measure of confidence that the service is free from errors. Its value is given in percent and calculated as follows:

$$R = \frac{succeededAttempts}{succeededAttempts + failedAttempts}$$

  R historically remained above 82%.
- Latency measures the mean time taken by the platform to return the expected result. The value is given in minutes.

$$L = \frac{\sum_1^n networkTime + selection/compositionTime + executionTime}{n}$$

  where n is the total number of past executions. Latency should not exceed 6 hours by day of the selected period but must be superior to 4 hours by day of the selected period. (This range is certified for a service requestor having network bandwidth of a least 15 mbits/s.)

This basic specification is incomplete. Further explanations are needed. For example, how different values are obtained is not described in the above informal specification. A quality model will provide a checklist of relevant information and in this respect assist the requestor and the provider in evaluating and managing the quality of the service.

3 Quality model for service-oriented systems

We introduce the quality model which integrates all components of the ontologies compared and reviewed in the subsequent part of the paper (Sect. 4). We call it the *Quality-Value-Dependency-Priority (QVDP)* model. The instantiation of its modeling primitives is illustrated using simple examples mainly for clarity of presentation.

Definition 1 The *Quality-Value-Dependency-Priority (QVDP)* model $\mathbf{QVDP} \equiv \langle \mathbf{Q}, \mathbf{V}, \mathbf{D}, \mathbf{P} \rangle$ consists of: the Quality characteristics submodel **Q**, the Quality Value submodel **V**, the Quality Dependency submodel **D**, and the Quality Priority submodel **P**.

Overall, the QVDP intergates submodels for different purposes related by shared modeling primitives. **Q** integrates the concepts of *quality dimension* **q** which is instantiated to represent measurable properties of a given SOS and *quality characteristic* $\bar{\mathbf{q}}$. A quality dimension can also be an aggregate of other quality dimensions obtained by applying some *aggregation function*. By grouping measurable **q** into $\bar{\mathbf{q}}$, the modeler can define how more abstract quality aspects of a system (e.g., safety, security, usability, maintainability, etc.)

are conceptualized in a given SOS. A **q** is characterized using a set of attributes which contains predefined, commonly used attributes and optional attributes to be defined as needed by the modeler for a particular SOS or class of SOSs. **V** is instantiated when specifying desired values (in service requests or when advertising services) for various quality dimensions and/or quality dimension aggregates.

While it is established that a quality model should represent the information on quality characteristics and their measurable quality dimensions, as in **Q** and **V**, this information alone is of limited use in dealing with tradeoffs at runtime. Since it is unlikely that various quality dimensions can all be satisfied to the desired extent simultaneously in a given SOS, indications are required on: (i) how various quality dimensions are interdependent, and (ii) what quality dimensions to optimize within a set of interdependent and conflicting quality dimensions (this is of relevance when defining service requests). A quality model is an apparent candidate for including modeling primitives for providing these indications. To respond to (i), we complement **Q** and **V** with the quality dependency submodel **D**, which is instantiated to represent that some value of a quality dimension is accompanied by some value of another **q**. To address (ii), we include the quality priority submodel **P** which is instantiated to define a (partial or complete) priority ordering over the various quality dimensions. By extending the basic quality modeling information considered in **Q** and **V** with that of **D** and **P**, we enrich the quality information with information relevant for managing tradeoffs at runtime in a SOS. This is a novelty of the QVDP.

Below, each submodel of QVDP is defined for a particular SOS. We thus speak of, e.g., a set of quality dimenions in a submodel instance since the instantiation of the submodel for a given SOS will necessarily involve the definition of a non-empty set of quality dimensions. We believe that this facilitates the understanding of what is obtained when using the QVDP model.

3.1 Quality characteristics submodel (**Q**)

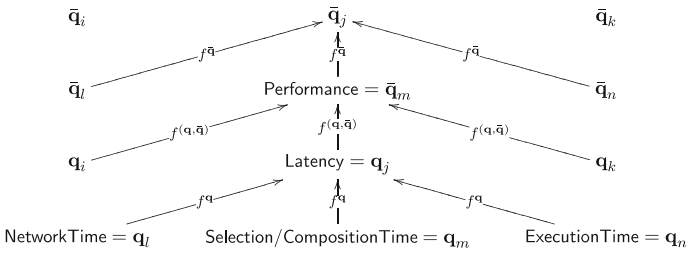
Definition 2 The **Quality Characteristics** submodel for a given SOS

$$\mathbf{Q} \equiv \left\langle \langle \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}, \{\alpha_1, \ldots, \alpha_m\}, f^{\mathbf{q}} \rangle, \langle \{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}, f^{\bar{\mathbf{q}}} \rangle, f^{(\mathbf{q},\bar{\mathbf{q}})} \right\rangle$$

consists of:

1. $\langle \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}, \{\alpha_1, \ldots, \alpha_m\}, f^{\mathbf{q}} \rangle$ which contains a set $\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ of quality dimensions defined for the given SOS, a set $\{\alpha_1,\ldots,\alpha_m\}$ of aggregation functions, and a function $f^{\mathbf{q}} : \mathbb{P}(\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}) \times \{\alpha_1, \ldots, \alpha_m\} \longmapsto \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ which maps a set of quality dimensions onto a quality dimension obtained by applying the aggregation function α on the set of quality dimensions. An aggregation procedure α specifies how to aggregate a set of quality dimensions. ($\mathbb{P}$ denotes powerset.)
2. $\langle \{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}, f^{\bar{\mathbf{q}}} \rangle$ which contains a set $\{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}$ of quality characteristics for the given SOS, and a function $f^{\bar{\mathbf{q}}} : \mathbb{P}(\{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}) \longmapsto \{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}$ which maps a sets of quality characteristics onto a quality characteristic which is defined as a group of the former.
3. A function $f^{(\mathbf{q},\bar{\mathbf{q}})} : \mathbb{P}(\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}) \longmapsto \{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_p\}$ which maps a set of quality dimensions onto a quality characteristic. The quality characteristic is defined as the given set of quality dimensions.

Example 1 Among the quality characteristics of the MERIS MGVI Regional service, we look into latency. Let Latency be an aggregate quality dimension 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 select and compose the services needed to fulfil the request (Selection/CompositionTime), and the time needed to execute the composition (ExecutionTime). Latency is part of the Performance quality characteristic. The following diagram illustrates how these measures fit into the terminology of the above Definition 2, and thus in the instance of **Q** for a given SOS.



Definition 3 A **Quality Dimension q** is a collection of the following attributes and attribute sets:

1. Name gives **q** a unique identifier. This identifier is the name of the quality dimension;
2. Description is a human-readable description of the quality dimension;
3. Purpose indicates why a given quality dimension is defined;
4. Type identifies the type of the quality dimension according to a taxonomy of variable types;
5. Unit defines the unit of measurement for **q**;
6. Aggregate is given for a **q** that is an aggregate of other quality dimensions. This attribute indicates the aggregation procedure and the quality dimensions aggregated to obtain **q**;
7. MeasurementSource indicates what is measured and how on/in the SOS in order to associate a value to the given quality dimension;
8. MeasurementTransformation defines the algorithm or formula used to transform the data obtained by measurement into the value reported for **q** (e.g., average, moving average, etc.);
9. $\{\mathsf{AdditionalAttribute}_1, \ldots, \mathsf{AdditionalAttribute}_n\}$ is the set of n additional attributes defined by the modeler.

A quality dimension can be seen as a metric used to quantify a quality characteristic using a certain (transformed) measurement on a property or behavior of a given SOS. Attributes are used to describe various characteristics of the metric.

Example 2 Latency for the MERIS service is a collection of the following quality dimensions: NetworkTime, Selection/CompositionTime and ExecutionTime. Let the dimension NetworkTime be an aggregate of SendTime and ReceiveTime, the former being the time for the service request to arrive from the requester to the composer, while the latter is the time for the composition execution output to be sent from the composer to the requestor. The following can describe NetworkTime:

Name	NetworkTime
Description	Average time (over 100 same requests) it takes to send and receive information between a service requestor and a service composer w.r.t. a given service request
Purpose	Used as an indicator when deciding what quantity of network bandwidth to demand from the bandwidth provider
Type	Continuous/Ratio
Unit	Millisecond (ms)
Aggregate	$\text{NetworkTime} = f^{\mathbf{q}}(\{\text{SendTime}, \text{ReceiveTime}\}, \text{AggregateSum})$
Measurement Source	Measurement sources for SendTime: `requestTime` attribute value for a `ServiceRequest` class instance and `requestReceptionTime` attribute value for a `Composer` class instance. Measurement sources for ReceiveTime: `requestCompletionTime` attribute value for a `Composer` class instance and `requestEndTime` attribute value for a `ServiceRequest` class instance
Measurement Transformation	$\text{NetworkTime} = \frac{1}{100}\sum_{i=1}^{100}(SendTime_i + ReceiveTime_i)$

Above, MeasurementSource assumes that a service request is described, among other, with a class called `ServiceRequest` which carries the attribute `requestTime`, and that its value can be collected by, e.g., a monitoring service (or some other SOS component). This value is then used, along with other values (as indicated in the MeasurementSource attribute) to calculate SendTime and ReceiveTime. In other words, MeasurementSource identifies where to find data to calculate quality dimension values.

We place no particular constraints on the structure of a quality characteristic. It is merely a means to organize quality dimensions that are considered as somehow related by the modeler.

Definition 4 A **Quality Characteristic** $\bar{\mathbf{q}}$ is a set of distinct (aggregate) quality dimensions.

3.2 Quality value submodel (**V**)

V is instantiated to define how quality is to be measured on a given SOS. At runtime, it is necessary to provide means for expressing desired values of the various quality dimensions. This information is subsequently used by composers to discriminate among alternative services when performing service composition—services that cannot achieve desired values for a set of quality dimensions will not be selected to participate in a composition. Simplistic models for expressing desired values over quality dimensions involve the definition of a single desired value. This is inappropriate because interdependencies between different behaviors of the given SOS are likely—it is thus necessary to relate the achievement of a particular quality dimension value with conditions on the system and/or its operating environment, and conditions that the achievement of some quality dimension value entails in the system and/or its operating environment. Hence the concepts of value precondition and value postcondition in the Quality Value submodel. Uncertainty in system operation leads to the probabilistic characterization of value and value pre/postcondition pairs.

Definition 5 The **Quality Value** submodel **V** for a given SOS is a set of p tuples, where each tuple $i = 1,\ldots,p$ is of the form

$$\left\langle \mathbf{q}_i.\mathsf{Name}, \left\{ \left\langle \mathbf{v}_{i,1}, \mathbf{v}_{\mathsf{Pre},i,1}, \mathbf{v}^P_{\mathsf{Pre},i,1}, \mathbf{v}_{\mathsf{Post},i,1}, \mathbf{v}^P_{\mathsf{Post},i,1} \right\rangle, \ldots, \left\langle \mathbf{v}_{i,n}, \mathbf{v}_{\mathsf{Pre},i,n}, \mathbf{v}^P_{\mathsf{Pre},i,n}, \mathbf{v}_{\mathsf{Post},i,n}, \mathbf{v}^P_{\mathsf{Post},i,n} \right\rangle \right\}, \left\{ \mathbf{v}^U_{i,1}, \ldots, \mathbf{v}^U_{i,m} \right\} \right\rangle$$

where

1. $\mathbf{q}_i.\mathsf{Name}$ is the value of the attribute Name of the quality dimension $\mathbf{q}_i$.
2. $\left\{ \left\langle \mathbf{v}_{i,1}, \mathbf{v}_{\mathsf{Pre},i,1}, \mathbf{v}^P_{\mathsf{Pre},i,1}, \mathbf{v}_{\mathsf{Post},i,1}, \mathbf{v}^P_{\mathsf{Post},i,1} \right\rangle, \ldots, \left\langle \mathbf{v}_{i,n}, \mathbf{v}_{\mathsf{Pre},i,n}, \mathbf{v}^P_{\mathsf{Pre},i,n}, \mathbf{v}_{\mathsf{Post},i,n}, \mathbf{v}^P_{\mathsf{Post},i,n} \right\rangle \right\}$ is the set of values for a given quality dimension $\mathbf{q}_i$, where each tuple associates a particular value $\mathbf{v}$ with a value precondition $\mathbf{v}_{\mathsf{Pre}}$ and the probability $\mathbf{v}^P_{\mathsf{Pre}}$ that the value $\mathbf{v}$ will be achieved if the precondition holds, and a value postcondition $\mathbf{v}_{\mathsf{Post}}$ and the probability $\mathbf{v}^P_{\mathsf{Post}}$ for the postcondition to hold if the given value of $\mathbf{q}$ is achieved.
3. $\mathbf{v}$ declares values for a given $\mathbf{q}$. The values are declared using syntax and semantics which differ depending on whether Type of $\mathbf{q}$ is continuous or discrete. If continuous, the semantics is defined in terms of an interpretation $(\mathbb{C}, \cdot^{c\mathcal{I}})$, which uses the continuous domain $\mathbb{C}$ defined by Type of $\mathbf{q}$, and an interpretation function $\cdot^{c\mathcal{I}}$ which associates with each v a $v^{c\mathcal{I}}$ in $\mathbb{C}$. If discrete, the domain of the interpretation is a set $\mathbb{D}$ which includes all allowed discrete values as defined by the value of Type of $\mathbf{q}$, and the interpretation function $\cdot^{d\mathcal{I}}$ associates with each e a $e^{d\mathcal{I}}$ in $\mathbb{D}$. Below, the syntax, semantics, and syntax rules are given for the continuous and discrete case.

Type of $\mathbf{q}$ *is continuous*		Type of $\mathbf{q}$ *is discrete*	
Syntax	Semantics	Syntax	Semantics
v	$v^{c\mathcal{I}} \in \mathbb{C}$	e	$e^{d\mathcal{I}} \in \mathbb{D}$
$\neg v$	$\mathbb{C} \setminus v^{c\mathcal{I}}$	$\neg e$	$\mathbb{D} \setminus e^{d\mathcal{I}}$
$(\geq v)$	$\{v'^{c\mathcal{I}} \mid v'^{c\mathcal{I}} \geq v^{c\mathcal{I}}\}$	E	$E^{d\mathcal{I}} \subseteq \mathbb{D}, E^{d\mathcal{I}} = \{e^{d\mathcal{I}} \mid \forall e \in E\}$
$\neg(\geq v)$	$\mathbb{C} \setminus (\geq v)^{c\mathcal{I}}$	$\neg E$	$\mathbb{C} \setminus E^{d\mathcal{I}}$
$(\leq v)$	$\{v'^{c\mathcal{I}} \mid v'^{c\mathcal{I}} \leq v^{c\mathcal{I}}\}$	$E_i \vee E_j$	$E_i^{d\mathcal{I}} \cup E_j^{d\mathcal{I}}$
$\neg(\leq v)$	$\mathbb{C} \setminus (\leq v)^{c\mathcal{I}}$	$E_i \wedge E_j$	$E_i^{d\mathcal{I}} \cap E_j^{d\mathcal{I}}$
$(\leq v) \vee (\geq v)$	$(\leq v)^{c\mathcal{I}} \cup (\geq v)^{c\mathcal{I}}$		
$(\leq v) \wedge (\geq v)$	$(\leq v)^{c\mathcal{I}} \cap (\geq v)^{c\mathcal{I}}$		
Syntax formation rules			
$\lambda^c ::= v \mid (\geq v) \mid (\leq v)$		$\lambda^d ::= e \mid E$	
$\Lambda^c ::= \lambda^c \mid \neg\Lambda^c \mid \Lambda^c_i \vee \Lambda^c_j \mid \Lambda^c_i \wedge \Lambda^c_j$		$\Lambda^d ::= \lambda^d \mid \neg\Lambda^d \mid \Lambda^d_i \vee \Lambda^d_j \mid \Lambda^d_i \wedge \Lambda^d_j$	
$\mathbf{v} ::= \Lambda^c$		$\mathbf{v} ::= \Lambda^d$	

4. $\mathbf{v}_{\mathsf{Pre}}$ gives an assertion which describes the precondition for the value(s) defined in $\mathbf{v}$ to be achieved.
5. $\mathbf{v}^P_{\mathsf{Pre}}$ indicates the probability that a value in $\mathbf{v}$ will be achieved if the precondition $\mathbf{v}_{\mathsf{Pre}}$ holds.
6. $\mathbf{v}_{\mathsf{Post}}$ gives an assertion which describes the postcondition that holds after achieving a value in $\mathbf{v}$.

7. $\mathbf{v}^P_{\mathsf{Post}}$ indicates the probability that the postcondition $\mathbf{v}_{\mathsf{Post}}$ holds if a value in $\mathbf{v}$ is achieved.
8. $\left\{\mathbf{v}^U_{i,1},\ldots,\mathbf{v}^U_{i,m}\right\}$ is the set of partial preference orderings $\mathbf{v}^U \equiv (\cdot) \overset{\mathsf{U}}{\succeq} (\cdot)$ on values for a given $\mathbf{q}_1$. $\overset{\mathsf{U}}{\succeq}$ defines a partial order (i.e., $\overset{\mathsf{U}}{\succeq}$ is a reflexive, transitive, and anti-symmetric relation). Both $(\cdot)$ follow the syntax and semantics of either Λ^c or Λ^d (depending on whether Type is continuous or discrete). Values specified on the left hand side of $\overset{\mathsf{U}}{\succeq}$ are preferred at least as much as the values given on the right hand side of the preference ordering relation. Strict ordering is defined in a strainghtforward manner (i.e., $x \overset{\mathsf{U}}{\succ} y \equiv x \overset{\mathsf{U}}{\succeq} y \wedge \neg(y \overset{\mathsf{U}}{\succeq} x)$).

Example 3 The following provides the instantiation of the quality value submodel for one value of NetworkTime, one of the dimension aggregated to define the Latency characteristic. Preconditions and postconditions are normally written in a formal language (e.g., the Semantic Web Rule Language (SWRL), Horrocks et al. 2003). Below, natural language and simple structured expressions are used to avoid introducing more formalism in this paper. The probabilities are usually estimated and continually updated at runtime.

$\mathbf{q}$.Name	NetworkTime
$\mathbf{v}$	$(\leq 3500\,\text{ms}) \wedge (\geq 1500\,\text{ms})$
$\mathbf{v}_{\mathsf{Pre}}$	ConnectionFailureProbability ≤ 0.05
$\mathbf{v}^P_{\mathsf{Pre}}$	70%
$\mathbf{v}_{\mathsf{Post}}$	Change of network bandwidth unnecessary
$\mathbf{v}^P_{\mathsf{Post}}$	90%
$\mathbf{v}^U$	$((\leq 3500\,\text{ms}) \wedge (\geq 1500\,\text{ms})) \overset{\mathsf{U}}{\succ} (> 3500\,\text{ms})$

3.3 Quality dependency submodel (**D**)

The quality dependency submodel is instantiated to express interdependencies between values of distinct quality dimensions. As noted earlier, the dependency submodel is of particular relevance for managing tradeoffs: it is the quality dependency model that makes explicit the possible tradeoffs, and summarizes how values of quality dimensions are related so that the outcome of tradeoffs can be anticipated.

Definition 6 The **Quality Dependency** submodel $\mathbf{D} \equiv \{\mathbf{d}\}$ for a given SOS is a set of dependency relations for pairs of distinct quality dimensions. Each dependency relation is written using the following formation rule:

$$\mathbf{d} ::= \left(\mathbf{q}_i.\mathsf{Name} \xrightarrow{\Lambda_k|\Lambda_l} \mathbf{q}_j.\mathsf{Name}\right)@(\phi,P) \mid \left(\mathbf{q}_i.\mathsf{Name} \xleftrightarrow{\Lambda_k|\Lambda_l} \mathbf{q}_j.\mathsf{Name}\right)@(\phi,P)$$
$$\mid \left(\mathbf{q}_i.\mathsf{Name} \xleftrightarrow{f} \mathbf{q}_j.\mathsf{Name}\right)@(\phi,P)$$

where

- $\mathbf{q}_i$.Name is the value of the Name attribute of $\mathbf{q}_i$ and $\mathbf{q}_j$.Name is the value of the Name attribute of some other quality dimension $\mathbf{q}_i$.

- Syntax and semantics of Λ_k follow that of Λ^c if the quality dimension $\mathbf{q}_i$ is of continuous type; if not, then the syntax and semantics of Λ^d are followed. Same applies for Λ_l and $\mathbf{q}_j$.
- The dependency is directed if Λ_k and Λ_l are related with "$\longrightarrow$", so that it if the value Λ_k of $\mathbf{q}_i$ is achieved, the value specified in Λ_l will be achieved for $\mathbf{q}_j$ at the probability P. "$\longleftrightarrow$" indicates that the dependency is directed both ways: if either of the quality dimensions reaches its given value Λ, the other will also have its value in its given Λ, at the probability P.
- When the relationship between the values of two quality dimensions can be described with a function, f gives that functional relationship.
- As the dependency may only be relevant when particular conditions hold, a condition ϕ can be added to indicate when the interaction applies (otherwise, ϕ remains empty). The condition is written as an assertion in the language used to write value precondition and value postcondition.
- P is a value that designates the probability for the dependency to be actually observed. It is usually estimated at runtime.

Example 4 The motivating example makes appear that the availability of the service is calculated with the help of the dimension DownTime. Similarly, the reliability depends on FailedAttempts. It seems clear that the number of failed attempts will be influenced by the down time of the system. The probability to observe this particular dependency of values is 0.8:

$$\left(\mathsf{DownTime} \overset{(increases)|(increases)}{\longrightarrow} \mathsf{FailedAttempts}\right)@(0.8)$$

3.4 Quality priority submodel (**P**)

In addition to **D**, **P** is another novelty of the proposed QVDP model. When a pair of quality dimensions is such that the values of both cannot be simultaneously optimized when, e.g., seeking appropriate service compositions, priority must be known over the pair in order to know which of the two to optimize at the expense of the other. The quality priority submodel is instantiated in order to make explicit the priority order between pairs of quality dimensions. Clearly, and as highlighted earlier, the priority submodel combines with the dependency submodel when managing tradeoffs: the latter indicates what pairs of quality dimensions are involved in tradeoffs, while the former indicates what to decide when tradeoff is to be performed between quality dimensions.

Definition 7 The **Quality Priority** submodel

$$\mathbf{P} \equiv \left\langle \{\langle \mathbf{p}_1^{\mathbf{q}}, \phi_1\rangle, \ldots, \langle \mathbf{p}_n^{\mathbf{q}}, \phi_n\rangle\}, \{\langle \mathbf{p}_1^{\bar{\mathbf{q}}}, \phi_1\rangle, \ldots, \langle \mathbf{p}_m^{\bar{\mathbf{q}}}, \phi_m\rangle\}\right\rangle$$

for a given SOS contains a set of (conditioned) priority orders for pairs of distinct quality dimensions, and a set of (conditioned) priority orders for pairs of quality characteristics. For quality dimensions, each priority order is $\mathbf{p}^{\mathbf{q}} \equiv \mathbf{q}_i.\mathsf{Name} \overset{\mathsf{P}}{\succeq} \mathbf{q}_j.\mathsf{Name}$, where $\mathbf{q}_i.\mathsf{Name}$ is different from $\mathbf{q}_j.\mathsf{Name}$. The given priority order indicates that optimizing $\mathbf{q}_i$ is important at least as much as optimizing $\mathbf{q}_j$. For each quality characteristic, the priority is given as $\mathbf{p}^{\bar{\mathbf{q}}} \equiv \bar{\mathbf{q}}_i \overset{\mathsf{P}}{\succeq} \bar{\mathbf{q}}_j$. $\overset{\mathsf{P}}{\succeq}$ 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 is conditioned if the optional ϕ is specified. If ϕ holds, the given priority order applied. ϕ is written as an assertion in the language used to write value precondition and value postcondition.

Example 5 A requester of a service providing information such that those provided by the MERIS MGVI Regional web service can instantiate the quality priority submodel to indicate in a service request that it is strictly more important to him to optimize reliability than latency, i.e., $\mathbf{p}_i^{\mathbf{q}} = \mathsf{Reliability} \overset{P}{\succ} \mathsf{Latency}$. This priority reflecting the fact that due to the long execution time expected for extracting the result, the requestor prefers to insure himself that the service will properly achieve and wait longer than the opposite. At the level of quality dimensions, the same request can indicate, e.g., that it is strictly more important to optimize the network time than to optimize the selection and composition time (i.e., the requester is interested in having the request transmitted rapidly, even if this entails longer time to find the appropriate services and their composition): $\mathbf{p}_k^{\mathbf{q}} = \mathsf{NetworkTime} \overset{P}{\succ} \mathsf{Selection/CompositionTime}$.

4 Comparison with prior quality models

Table 1 summarizes the comparison. It indicates concepts and constructs of prior quality ontologies and models that correspond to those of **QVDP**. The table highlights the novelty of including submodels and constructs for representing preference, dependency, and priority among quality dimensions and characteristics.

Only some of the many approaches capable to describe service quality are considered here. Other approaches include: QuA (Staehli et al. 2003), QML (Frolund and Koistinen 1998), WSOL (Tosic et al. 2002), UniFrame (Brahnmath et al. 2002), CORBA Trading Object Service (Object Management Group 1997), QuO (Loyall et al. 1998), SLAng (Skene et al. 2004). They are not overviewed here for two reasons: first, previous comparisons (e.g., Skene et al. 2004) indicate that the fragments of their models specialized for service QoS description are subsets of the set of concepts and constructs present in the approaches considered herein; second, the aim here is only to identify key concepts and constructs manipulated when specifying a service's QoS, so that we do not discuss how each of the concepts adapts to a particular implementation framework—therefore, we do not consider in this section the results focused on adopting and adapting the reviewed approaches to particular implementation technologies and frameworks.

In Table 1, "$\surd$" indicates that the given concept or construct can be expressed in the relevant quality model, but the name of concept or construct for doing so cannot be identified in the cited work. "×" indicates that the given QVDP concept or construct has no corresponding concept or construct in the relevant quality model.

4.1 Q-WSDL

D'Ambrogio proposes Q-WSDL (D'Ambrogio 2006), a Quality of Service[3] (QoS) extension to WSDL (Christensen et al. 2001) which includes a set of QoS characteristics

[3] According to the relevant ISO standard (International Organization for Standardization 1998), QoS refers to characteristics that contribute to the overall quality of a service as perceived by the consumer of the service. A QoS characteristic is a quantifiable aspect of QoS which is defined independently of the means by which it is represented, managed, or controlled.

Table 1 Comparison of **QVDP** with a selection of prior quality models

QVDP	Q-WSDL	WSLA	DAML-QoS	Maximilien and Singh	Zeng and colleagues
$\mathbf{Q}$	QoS profile	Service Level Agreement	QoS Profile	√	Quality vector
$\mathbf{q}$	QoS dimension	Metric	Metric	Quality	Quality criterion
Name	name	name	metricName	√	√
Description	definition	√	√	√	√
Purpose	×	×	×	×	√
Type	type	type	√	√	√
Unit	unit	unit	√	√	√
Aggregate	×	√	√	√	√
Measurement Source	source	Function or Measurement Directive	√	√	√
Measurement Transformation	×	√	Function	AggregateQuality	√
Additional Attribute	×	×	×	QAttribute	×
α	×	Function or Measurement Directive	√	√	√
$f^{\mathbf{q}}$	×	√	√	√	√
$\bar{\mathbf{q}}$	QoS characteristic and QoS category	QoS Property	×	×	×
$f^{\bar{\mathbf{q}}}$	×	×	×	×	×
$f^{(\mathbf{q},\bar{\mathbf{q}})}$	√	√	×	×	×
$\mathbf{V}$	×	Service Level Objectives or Action Guarantees	QoS Precondition, QoS Effect, QoS constraints	QoS Policy	×
$\mathbf{v}$	value	√	√	QValue typical	√
$\mathbf{v}_{\mathsf{Pre}}$	×	√	√	×	×
$\mathbf{v}^{P}_{\mathsf{Pre}}$	×	×	×	×	×
$\mathbf{v}_{\mathsf{Post}}$	×	√	√	×	×
$\mathbf{v}^{P}_{\mathsf{Post}}$	×	×	×	×	×

Table 1 continued

QVDP	Q-WSDL	WSLA	DAML-QoS	Maximilien and Singh	Zeng and colleagues
$\mathbf{v}^U$	direction	√	×	×	√
D	×	×	×	QRelationship	√
d	×	×	×	ValueImpact, ValueDirection	√
P	×	×	×	×	√
$\mathbf{p^q}$	×	×	×	×	√
ϕ (for $\mathbf{p^q}$)	×	×	×	×	×
$\mathbf{p^{\bar{q}}}$	×	×	×	×	√
ϕ (for $\mathbf{p^{\bar{q}}}$)	×	×	×	×	×

accepted in a UML extension for specifying QoS (Object Management Group 2005). In Q-WSDL, a service's QoS is described through QoS characteristics (e.g., Availability), each quantified through QoS dimensions (e.g., UpTime). A QoS dimension is described with a value, unit of the value, source of measurement (e.g., measured, assumed, predicted, etc.), and optionally, a type of statistical value (e.g., mean, variance, etc.) and an order relation to compare values (i.e., to answer which values are preferred). QoS categories bring together QoS characteristics related to a common subject—e.g., reliability and availability are grouped in the dependability category.

Q-WSDL highlights the need to specify preferences over quality values, yet remains limited in doing so compared to QVDP. Namely, Q-WSDL allows specifying the desired direction for the value of the quality dimension, but cannot deal with value preconditions and value postconditions, or uncertainty thereof (i.e., through probabilities). Q-WSDL is not extensible, in that it has no apparent mechanism for introducing additional attributes to finely describe a quality dimension. In contrast to all other models noted in Table 1, Q-WSDL omits aggregation of quality characteristics. Note that lack of means to represent dependency and priority information in Q-WSDL entails that any directives on tradeoffs need to be specified outside the quality model. Additional representation formalism (unspecified in case of Q-WSDL) is thus necessary, and further effort is consequently required for ensuring consistency between the information in the quality model and the tradeoff formalism. Such problems are avoided in QVDP.

4.2 WSLA

While focusing on the contracting between the service provider and requester, the Web Service Level Agreement language (Keller and Ludwig 2003) integrates similar kind of information as the Q-WSDL when describing quality characteristics. The Web Service Level Agreement (WSLA) language (Keller and Ludwig 2003) focuses on the contracting between the service provider and requester. It is used to specify quality of service within a Service Level Agreement (SLA). A quality dimension is expressed by a SLA parameter and described with a name, type, unit, definition and purpose. The value of a parameter is quantified by a metric, whereby a metric can be an aggregate of other metrics. If aggregate, it is given by a function or a measurement directive; a function uses other metrics as operands while a measurement directive specifies how an individual metric is retrieved from the source. For example, a function can be such that the value of the metric used to calculate the Availability characteristic can be obtained by dividing the value of UpTime by the value of the sum of UpTime and DownTime. Typical examples of measurement directives are the uniform resource identifier of a hosted computer program, a protocol message, or the command for invoking scripts of compiled programs. The notion of value specification is similar to the concept of obligation which define guarantees and constraints on SLA parameters. These obligations cover the service level objectives that represent promises with respect to the state of SLA parameters and the action guarantees that are promises of a signatory party to perform an action. Value preconditions and postconditions in QVDP enable can be used to express such constraints-related information. A service level objective expresses a commitment to maintain a particular state of the service in a given period, e.g., the SLA parameter TimeNeededToTransferData must be lower than 100 ms if the SLA parameter NetworkTime is less than 30 ms. An action guarantee expresses a commitment to perform a particular activity if a given precondition is met; e.g., sending an event to one or more signatory party and supporting parties,

opening a problem report, performing the payment of a penalty, or of a premium, and so on.

While close to QVDP in terms of specifying quality dimensions and quality characteristics, and describing values thereof, WSLA is similar to Q-WSDL in not providing indications on how to deal with tradeoffs. This is an important limitation, since a contract between a provider and a requester can involve situations in which tradeoffs need to be managed. For instance, adding dependency information would allow the provider to check whether some requested levels of quality dimensions can be realized: knowing their interdependency, the provider might highlight that the requested levels are unrealistic, or that more desirable levels can be achieved. Adding priority indications can lead to more extensive contracts, in which contingencies can be managed in a finer way: e.g., if the provider indicates that in some situation particular levels of some metrics cannot be achieved together, the requester might indicate which of the metrics is to be optimized in place of others when such a situation occurs. Both dependency and priority information thus appear relevant when contracting between the provider and the requester, but are absent from the WSLA.

4.3 DAML-QoS

Zhou et al. (2004) extend DAML-S with a means for describing QoS in a generic manner. Apart from the usual definition of concepts for metrics, metric types, and metric aggregation, their DAML-QoS ontology introduces the possibility for defining QoS characteristics whose value at the service's input is different from its value at output (e.g., in a converter, input and output bit rates differ). Since QVDP is not directly associated to a particular model of the service-oriented system, relating service inputs and outputs to different values can be accomplished by instantiating the quality value submodel, and defining a function to map specific values to inputs or outputs of a given service. Also, the service may only be capable of achieving some specific quality level if some external quality level is satisfied (e.g., for two interacting services, a minimal throughput rate at the first service might be needed if the second service is to ensure some desired throughput rate), so that a condition on QoS characteristics can be defined (as a QoSPrecondition). Similarly, the level of quality achieved over the service's various QoS characteristics may change the effects of the service—the effect of QoS can be defined as QoSEffect in DAML-QoS. Both of these are supported with value preconditions and postconditions in QVDP.

While DAML-Qos is innovative in terms of value preconditions and value postconditons when compared to other prior quality models, DAML-QoS cannot deal with preferences over values, dependencies, and priorities. It thus suffers from the same limitations as Q-WSDL and WSLA when contrasted to QVDP: we see no explicit means in DAML-QoS to address tradeoffs at runtime.

4.4 Maximilien and Singh

Maximilien and Singh (2004) describe service QoS through an ontology which abstracts from particular QoS characteristics: each QoS characteristic is called a quality, is associated to a typed variable, indications on how it is measured, and its relationships to other qualities (in terms of strength and direction). Namely, QMeasurement quantifies a quality while AggregateQuality combines several Qmeasurements into aggregate metrics. QRelationships indicate related qualities in a manner that shows through their values. A quality related to another has a valueimpact (weak, mild and strong) and a valuedirection

(opposite or parallel). A QRelationship is similar to a dependency specification, though the latter allows more precise information to be given on relationahips between metrics. The QoSPolicy specification indicates a level of commitment of the provider to the advertised policy (bestEffort, guaranteed, notSpecified or noGuarantee). As QoSPolicy aims at add constraints on the value, so that it is equivalent to the value specification.

An important characteristic of Maximilien and Singh's approach is the integration of information on relationships between qualities using the notions of value impact and value direction. Their approach qualifies value impact as either weak, mild, or strong, while value direction is described as either opposite or parallel. QVDP thus covers their dependency representation, adds the possibility to indicate conditions for interdependencies and uncertainty thereon, and allows more detail in describing interactions. Again, as Q-WSDL, WSLA, and the DAML-QoS, Maximilien and Singh's model has no means to deal with priority.

4.5 Zeng and colleagues

Zeng et al. (2003) use an ad-hoc informal quality model in which they represent price, execution duration, reputation, reliability and availability to guide service composition at runtime. A quality of service is expressed by means of a quality vector. The vector is composed of quality criteria that we define with quality dimensions. Each of the criteria is described similarly to our quality dimension, that is, by a name, a unit, a type, a short definition and an (optional) expression defining how to perform the measurement. The desired values are given with modalities which depend on the quality's purpose; e.g., execution price is minimised whereas reliability is maximized. The idea is the same in the preference fragment of our quality value submodel where an order relation is given between possible values. Zeng and colleagues represent priority by associating weights to each of the criteria used in the aggregation function which is optimized when performing service composition.

Zeng and colleagues' approach is closest to QVDP in terms of dependency and priority representation, although it is comparatively limited when dealing with value descriptions. Both interdependency and priority representations in Zeng and colleagues' proposal remain implicit from the specific set of quality criteria they use during composition. In this respect, theirs is not a quality model per se, but a particular case of an implicit quality model. In this sense, they do not discuss a quality model, but only specific quality dimensions and their application in service composition. In addition, QVDP differs in the detail it allows—e.g., we can specify conditions for priority orderings to apply while this is not considered by Zeng and colleagues.

5 QVDP and QoS in UML

Among the related efforts, the UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (Object Management Group 2005) stands out in its coverage of various concepts and constructs for conceptualizing quality and its status of standard. Therein, a metamodel is proposed as an extension to the UML metamodel to support the definition of QoS properties for systems of which other aspects are modeled with UML. In this section, we review that metamodel, compare it to QVDP, and extend the UML QoS metamodel with concepts and constructs available in QVDP and unavailable there. Finally, we propose a case study using the UML QoS Framework and our extensions derived form the service presented in Sect. 2.

5.1 Elements of the metamodel

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. We review these packages 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 QoSCharacteristic 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 the 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 represent 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.

- **QoSLevels package**
  - *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.

5.2 Comparison of the QVDP and the UML QoS Framework metamodel

Table 2 summarizes the comparison. In both models, QoS characteristics are quantified by dimensions which own unit, type and name. In the UML QoS Framework metamodel, Description and Purpose do not appear in the class describing dimensions. UML provides a way to define a parent characteristic from sub-characteristics but does not allow the

Table 2 Comparison of the QVDP and the UML QoS Framework metamodel

QVDP	UML QoS
$\mathbf{Q}$	QoSContext
$\mathbf{q}$	QoSDimension
Name	$\surd$
Description	$\times$
Purpose	$\times$
Type	$\surd$
Unit	unit
Aggregate	$\times$
Measurement Source	QoSDimensionSlot
Measurement Transformation	statisticalQualifier
α	$\times$
$f^{\mathbf{q}}$	$\times$
$\bar{\mathbf{q}}$	QoSCharacteristic and QoSCategory
$f^{\bar{\mathbf{q}}}$	Sub-parent relationship
$f^{(\mathbf{q},\bar{\mathbf{q}})}$	QoSParameter
$\mathbf{V}$	$\times$
$\mathbf{v}$	QoSConstraint
$\mathbf{v}_{\mathsf{Pre}}$	Provider QoSRequired
$\mathbf{v}^{P}_{\mathsf{Pre}}$	$\surd$ (QoSLevel)
$\mathbf{v}_{\mathsf{Post}}$	QoSOffered
$\mathbf{v}^{P}_{\mathsf{Post}}$	$\surd$ (QoSLevel)
$\mathbf{v}^{U}$	direction
$\mathbf{D}$	$\surd$
$\mathbf{d}$	QoSConstraint
$\mathbf{P}$	$\times$
$\mathbf{p}^{\mathbf{q}}$	$\times$
ϕ (for $\mathbf{p}^{\mathbf{q}}$)	$\times$
$\mathbf{p}^{\bar{\mathbf{q}}}$	$\times$
ϕ (for $\mathbf{p}^{\bar{\mathbf{q}}}$)	$\times$

composition of a dimension from other dimensions. This way, no aggregate function is provided by the QoSDimension metaclass. The QoSParameter metaclass defines how a characteristic is composed from dimensions. The measurement transformation concept appears in UML but possibilities are limited; the statistical qualifier only compute values with a modality (maxvalue, minvalue, mean, and similar). The source of measurement is represented by the QoSDimensionSlot metaclass in the UML QoS Framework metamodel. Valid values are limited by means of QoSConstraints elements; provider's required constraints allow to highlight preconditions while provider's offered constraints can define postconditions. The probability associated to these conditions can be given using QoSLevel. Even if levels do not define directly probabilities, they take into account available resources to determine if constraints can be respected. The attribute direction indicates preference relations over values for a same dimension. We can write dependencies as constraints over values of different dimensions. UML QoS does not provide any elements enabling to point up priorities among dimensions or characteristics.

We see that the UML QoS Framework metamodel can be usefully extended with concepts and constructs from the QVDP. The extension is outlined below.

5.3 Extending the UML QoS Framework metamodel

Figure 3 shows the submodels of the extended UML QoS Framework 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, respectively, 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 3 and 4 summarize the extended UML QoS Framework metamodel. Changes and extensions are given below.

- **Priorities submodel** As the UML QoS Framework metamodel does not account for priorities over quality elements, the submodel presented here is an extension introducing extra classes to specify priorities over characteristics and over dimensions.
  - *QoSPriority* The main class of this submodel is used to express rules that define priorities over characteristics or dimensions. These rules determine the order at which characteristics or dimensions are considered for optimization when services are being selected. A rule defines an order relation between elements. Different methods can be used in order to rank characteristics or dimensions; simple precedence relation can be established or weighted functions over elements can be used to account for some criteria.
  - *QoSDimPriority* and *QoSCharactPriority* These classes are specializations of QoSPriority defining specific elements for priorities over, respectively, characteristics and dimensions.

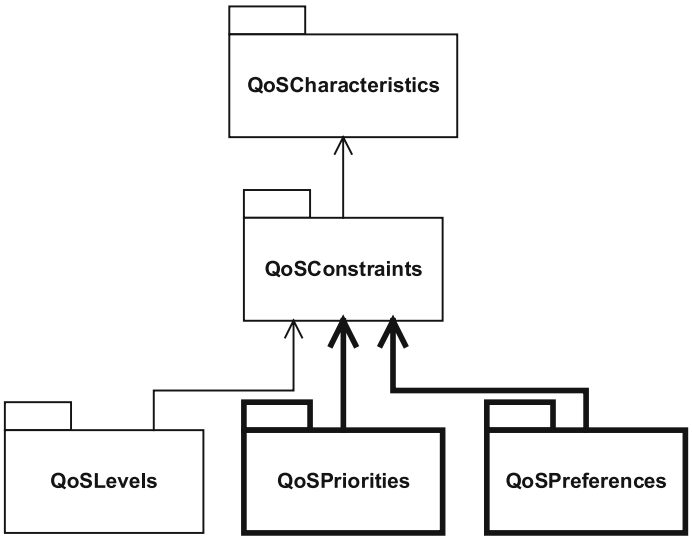


Fig. 3 Submodels of extended UML QoS Framework metamodel. Extensions are in bold

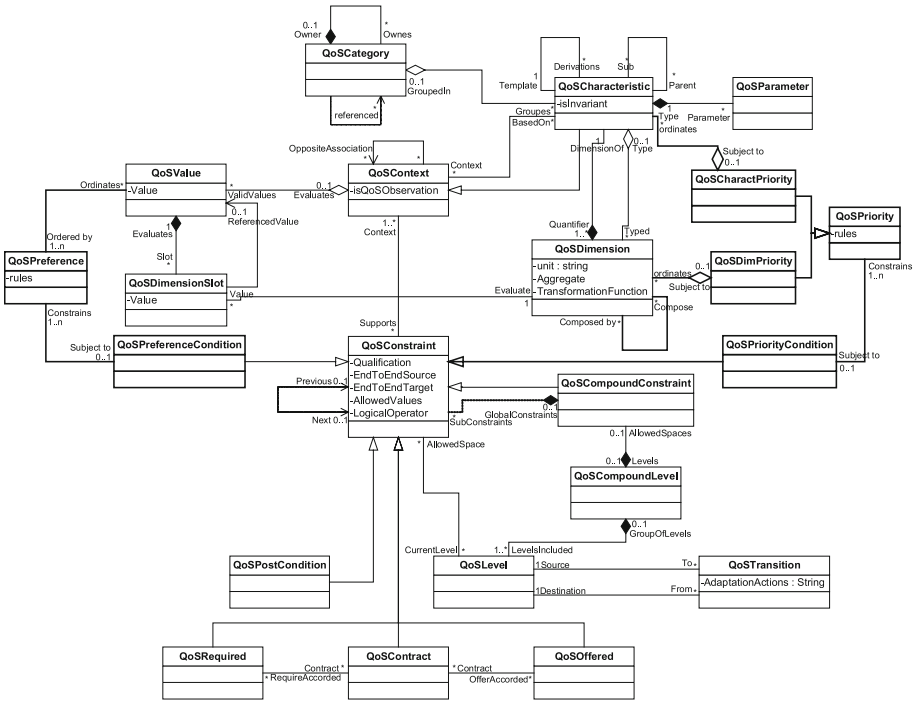


Fig. 4 UML QoS Metamodel with proposed extensions

- *QoSPriorityCondition* Conditions on priorities are constraints specifying when priorities hold. We use this element to specify, e.g., the priority that holds only if some value over a quality dimension is achieved.

- **QoSPreferenceSubmodel** We use the QoSPreferenceSubmodel to write preferences over values. In the original UML QoS Framework metamodel, preferences are defined by means of an attribute that indicates the preferred value direction (i.e., increase or decrease). The introduction of the QoSPreference submodel makes the framework considerably more expressive; e.g., set of values can be preferred over others under some specific conditions.
  - *QoSPreference* The purpose of QoSPreference class 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 conditions for the preference on values to hold.

- **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 allows us only to define the modality under which the value of a dimension can be calculated on a set value. In QVDP, it is possible to define a dimension from other dimensions, so that the value calculation has to be more expressive. We augment expressivity by replacing statisticalQualifier with two attributes: Aggregate which defines from which other dimensions the value is

calculated and TransformationFunction which provides the formula to compute the value of the dimension.

- **Compose–composed relationship** In QVDP it is possible to compose a dimension from other dimensions as for characteristics. This possibility is expressed in the metamodel thanks to the compose–composed by self-relationship of QoSDimension class.
- **QoSPostCondition** As no element in the original UML QoS Framework metamodel allows us to specify the postcondition expressed in QVDP, this possibility is added under the form of a class which is a specialization of the QoSConstraint class.

5.4 Case study

As suggested by the OMG, the UML metamodel can be used by service requestors and providers to define their respective requests and capabilities about QoS. We propose in this case study some examples of utilization of the UML QoS Metamodel to express such information. We refer to the service ENVISAT/MERIS MGVI Regional introduced in the Sect. 2 to illustrate how to use the metamodel and its submodels to organize QoS advertising specification.

The first submodel, the QoS Characteristic submodel, is illustrated in Fig. 5. It illustrates the characteristic latency and the dimensions used to compute its value. The calculation of one of these dimensions, Network Time is more developed with association of dimensions used to compose its own value and its transformation function. The Performance Category groups all characteristics related to the service performance. The modeling constructs initially introduced in the UML QoS Framework Metamodel do not allow the quantification of a QoS Dimension by another QoS Dimensions. However, in the context of the ENVISAT/MERIS MGVI Regional service, the latency is a critical point. Specifying how its different parts are computed permits to the user or the provider to specify its preferences at different levels and so give an upper limit to the ReceiveTime, which is used to compose the NetworkTime.

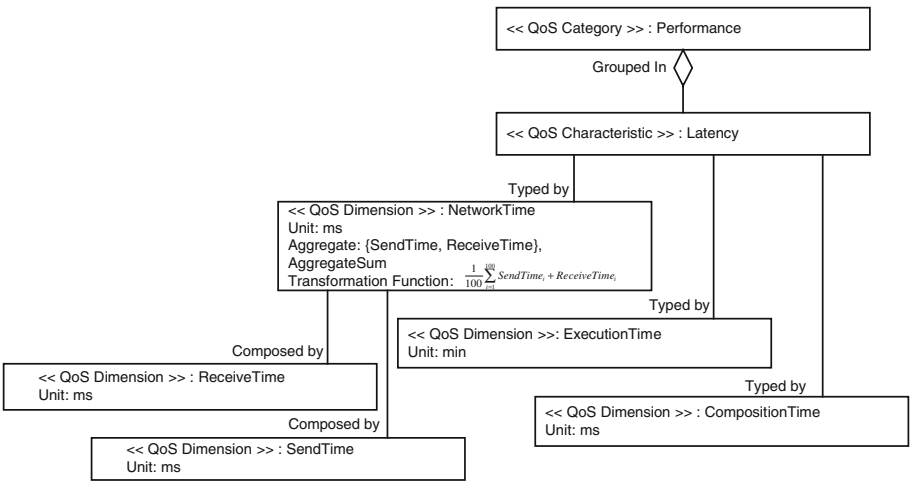


Fig. 5 UML QoS Characteristics submodel

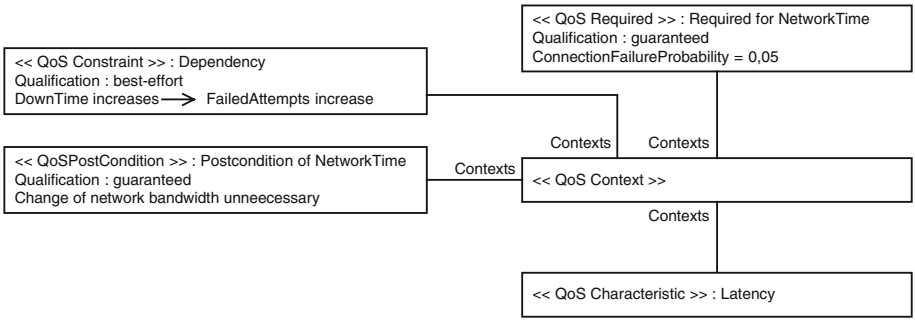


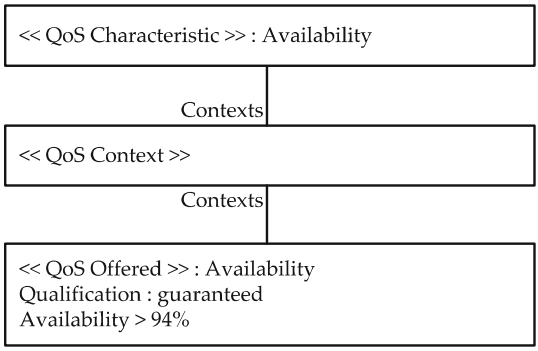
Fig. 6 UML QoS Constraints submodel

The second submodel, the QoS Constraint submodel, is presented in Fig. 6. It exposes the context and its related constraints. Among those, one constraint illustrates the dependency existing between the down time and the total number of failed attempts. One is used to specify the precondition necessary to an acceptable value of network time and another the induced postcondition. The specification of such constraints, made by the service provider, in the context of our MERIS service, are used to inform the service user of particular requirements and observations about service behavior. The specification of the QoS Dependency related to the DownTime and the number of FailedAttempts is useful for the user of the service because it enables the anticipation of quality degradations.

The constraint submodel is also used to indicate the service provider capabilities of QoS. These capabilities are described with the help of QoS Offered, a specialization of QoS Constraint. The Fig. 7 highlights the capabilities of the MERIS MGVI Regional Service for the availability characteristic. To advertise its capabilities about quality properties, the service provider has to specify them. In the context or our case study, these specifications are simply the observations made on the system that are expressed with the help of the QoS Offered metaclass.

The QoS Priorities submodel is expressed in Fig. 8 and concerns a priority fixed between the latency and the availability. Services as the MERIS MGVI Regional have an important latency, mainly due to the huge execution time of their requests. This way, as the latency is important, it is not a relevant characteristic to choose among alternatives and other characteristics as availability are more relevant to discriminate among available services. Such a specification is written by the user of the ENVISAT/MERIS MGVI Regional service that wishes express which quality properties to favor to others.

Fig. 7 UML QoS Offered constraints



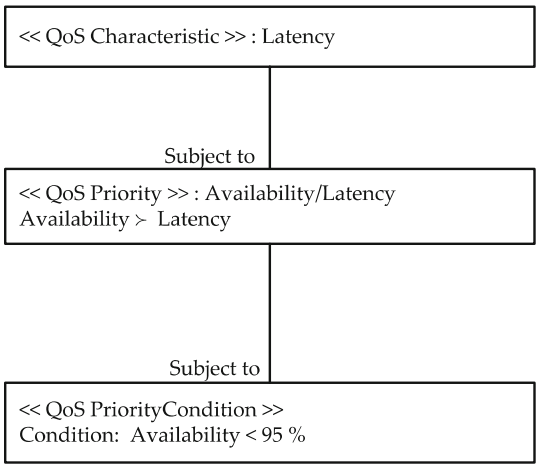


Fig. 8 UML QoS Priorities submodel

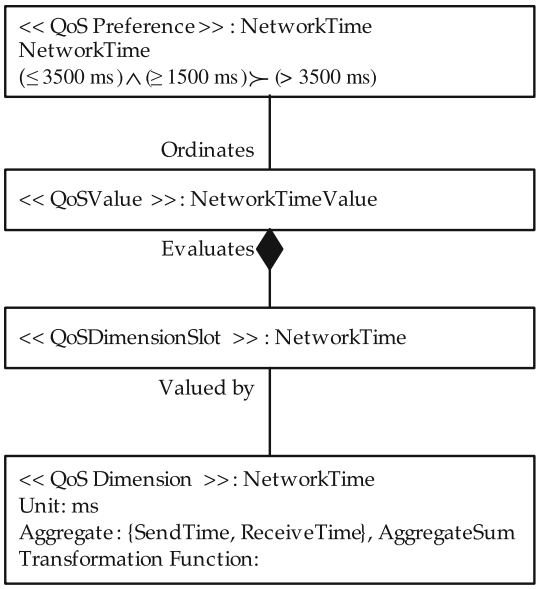


Fig. 9 UML QoS Preferences submodel

An example of expression abilities of the QoS Preferences submodel is proposed in Fig. 9. It illustrates how to specify favored values for the network time dimension. This specification of desired values for NetworkTime is written by the service user to make appear its expectations. The service user makes a similar specification for every QoS property.

The UML QoS Framework permits to service providers and users to express simply their advertisings and their requests. With our added extensions, it covers a large range of construct modeling and possibilities. Their complete description enables their powerful utilization in selection or composition approaches. Indeed, quality characteristics account for selection criteria in selection problems while they appear as local or global constraints in composition issues. UML specifications of quality requirements (or offers for providers)

need to be translated in a language such that Q-WSDL in order to be sent and used by services selectors or composers. The model of QoS needs or offers should be established with the help of an appropriate tool allowing to construct the UML model and to translate this into an exchangeable language between interested parts. Any tool that uses our QoS model will allow users to define their expectations, requirements and advertisements. A tool that incorporates all of our proposed modeling constructs can easily be used to state quality properties of the service. Moreover, a such tool can also involve coherence rules to check the consistency of a model, with transitivity of priorities or propagation of dependencies at different levels of specification.

6 Discussion

We have observed at the outset of the paper that quality has been variously defined. While quality is undoubtedly a polysemous concept, using it in software engineering requires agreement, even if local to a system or application domain, as to what is understood by quality management and quality assurance during the engineering, operation, and maintenance of software, and therefore service-oriented systems. Quality modeling is in this respect one of key activities. A quality model in this perspective has several purposes. Prominent among these are (i) to highlight the information to account for when representing and reasoning about quality, (ii) to indicate how quality is measured so as to assist in the assessment of a given software system, (iii) and to structure the information about users' and software engineers' quality requirements. The quality model proposed in this paper has been designed with the said three purposes in mind. It focuses on service-oriented systems whose salient characteristic is dynamicity during operation, whereby dynamicity herein amounts primarily to the uncertainty about the levels of quality that the system achieves during operation. The uncertainty arises from the fact that the pool of available services varies over time. This may be desirable w.r.t. quality when new, more appropriate services appear in the pool of available services, but can also be undesirable when some relevant services become unavailable resulting in inconsistent quality levels.

6.1 Experience

Service orientation is intended to enable large scale systems. Many competing services are therefore available to perform the same tasks. In such a setting, the service composer aims to *select* the set of services that optimally satisfies the quality considerations laid out in the request, and this relative to alternative sets of services that can perform the same tasks. Our quality model has been applied primarily to the problem of web service selection. In an SOS, web service requesters specify tasks that need to be executed and the quality levels to meet, whereas service providers advertise their services' capabilities and the quality levels they can reach. Service selectors then match to the relevant tasks, the candidate services that can perform these tasks to the most desirable quality levels. One of the key problems in QoS-aware service selection lies in managing tradeoffs among QoS expectations at runtime, that is, situations in which service requesters specify quality levels that cannot be simultaneously met. We have used the quality model presented in this paper within a wider service selection framework (Herssens et al. 2008a), in order to be able to deal with QoS tradeoffs. That framework consists of: (i) rich QoS models obtained by instantiating the model in the present paper, used by service requesters when expressing QoS expectations

and service providers when describing services' QoS; and (ii) a multi-criteria decision making technique that uses the models for service selection. The additional expressivity of the quality model presented in the present paper proved a significant advantage in dealing with QoS tradeoffs. Indeed, the multi-criteria method that was used requires the identification of preferences over QoS values and priorities over QoS criteria, whereby a criterion equates with a quality dimension. The conceptual bases laid out in the present paper proved immediately useful in constructing a web service selection framework that can deal with QoS tradeoffs.

The service selection framework (Herssens et al. 2008a) mentioned above uses a specific multi-criteria decision making method to rank alternative services. In our other work (Herssens et al. 2008b), we have also used our quality model to develop a generic addon to various service selection approaches so as to make these approaches aware of service requesters' priorities over quality dimensions and/or characteristics. Instead of developing a particular service selection procedure which accommodates priorities between QoS considerations, we provided an extension compatible with (i.e., that can be used with) available selection approaches. Our approach is based on the premise that SOS operate in a setting in which QoS levels vary and are observed at runtime. The approach is based on a specific class of multi-criteria decision making techniques, called the outranking methods. The approach enables the definition of a global priority constraint to be used as an ordinary constraint in a service selection algorithm. We do not ask for a specific class of service selection algorithms; any algorithm which proceeds to select the optimal services and accounts for QoS considerations can be used in conjunction with the present proposal. This is for instance the case with the reinforcement learning algorithm we suggested elsewhere (Jureta et al. 2007a, b). The global priority constraint is relevant because: (i) it allows priorities to be accounted for during service selection in algorithms that originally cannot accommodate priorities; (ii) it can be integrated with various available service selection algorithms, and regardless of their specific optimization functions; (iii) it enables automatic optimization of user preferences by the service composer. It is our quality model that provides the conceptual foundations for the definition of the global priority constraint, and thereby the extension of service selection algorithms to accommodate richer specification of quality-related requirements.

This quality model has primarily been applied to service-oriented systems, and we presented it in that context in the present paper. In this respect, we cannot advance claims on its applicability to other kinds of systems. We can however, identify arguments in support for its wider relevance. First, preferences and priorities expressed by the stakeholders (e.g., a system's users, owners, and so on) are not a kind of information specific to service-orientation. We have argued elsewhere (Jureta et al. 2008) that preferences and priorities are expressed by the system's stakeholders and should be accounted over the course of the requirements engineering phase in the software development process. Requirements engineering deals with the elicitation, analysis, and specification of stakeholders' functional and quality requirements. Decision on the computing paradigm to adopt (e.g., service-orientation or agent-orientation, or a combination, or otherwise) arises in part from the requirements that the stakeholders express. It follows that our quality model may be used, at least as a starting point in the definition of quality models for kinds of systems other than service-oriented ones. Second, we have used our model to extend the UML QoS Framework, which is not specific to service-oriented systems. Below, we consider the lessons learned from the use of our quality model in the context of service-oriented systems.

6.2 User evaluation

The quality model and the service selection approaches that arose from and use the model have been used in cooperation with the European Space Agency, and the MERIS project. While this does not provide enough empirical data on to reach definite conclusions on the ease of use of the present quality model, some preliminary observations are available.

It was to be expected that a more expressive quality model requires more effort in use. This is the case with our quality model. The additional effort equated to: (i) additional training of the modeling participants that is necessary to understand the new modeling primitives and their use; (ii) effort involved in acquiring and analysing the information to carry over to the instances of the modeling primitives; (iii) effort involved in using the instances of our quality model in decision-making during the engineering of a service-oriented system. The effort involved in (i) and (ii) are topics for research in requirements engineering, while (iii) is a concern for research in service selection. In practice, we have encountered difficulties in relation to both issues (i) and (ii). In a separate discussion (Jureta et al. 2008), we show that information about preferences and priorities is not properly accounted for in ontologies for requirements engineering. The direct consequence of this is that there is limited research on the elicitation and analysis of preferences and priorities in requirements engineering, and thereby little systematic guidance on these tasks. In addressing issue (iii), we have used multi-criteria decision making techniques developed mainly in management science. Our experiences with the use of these techniques are reported elsewhere (Herssens et al. 2008a, b). Overall, multi-criteria decision making techniques allowed us to automate some of the tasks involved in using the instances of our quality model to rank services, while accounting for preferences and priorities of the service requesters. In addition, these techniques incorporate features that allowed us to address issues (i) and (ii) to some extent. For example, and as explained elsewhere (Herssens et al. 2008a, b), the methods we chose can cope with partial specifications of preferences and priorities, thus limiting the amount of information to elicit before instantiating our quality model. This is critical, for it reduces the efforts of both kinds (i) and (ii) mentioned above.

A separate and difficult issue concerns dealing with change of quality dimensions or characteristics, and/or preferences and priorities at runtime. It still remains unclear how precisely such changes affect even the engineering of functional requirements, which appear easier to pin down than quality requirements. Some changes can be managed. We have performed and presented elsewhere (Jureta et al. 2007c) our investigations on the impact changes in functional and quality requirements on the requirements engineering process for SOS. We showed therein that keeping track of changes is an issue that can only be properly dealt via appropriate tool support, although we did conclude that no easy to use solution is available at present.

6.3 Strengths

Our model's principal strength is its additional expressivity in comparison to related quality models for SOS. This additional expressivity raised relevant questions in relation to the problems of web service selection. Namely, we have shown elsewhere (Herssens et al. 2008a, b) that the use of our quality model enables us to capture more detailed information from the service requesters—in particular, their preferences and priorities. This additional information proves invaluable when dealing with QoS tradeoffs during service selection.

Namely, when QoS levels requested by the users cannot be simultaneously reached, preferences and priorities obtained by instantiating our quality model allow us to perform tradeoffs in accordance with the requirements of the service requesters.

6.4 Weaknesses

We have observed in practice, within the context of the MERIS project, that additional effort was required for the acquisition, analysis, and specification of preferences and priorities. We have presented elsewhere (Herssens et al. 2008a, b) our first investigations on how this additional effort can be reduced within a broader framework for service selection in SOS. Our quality model does not feature primitives that are tailored to managing change in quality dimensions and characteristics, and/or preferences and priorities.

6.5 Future work

Work with more expressive models requires additional resources. Striking a balance between expressivity and efficiency is a difficult question, with answers undoubtedly confined to domain-specific experience. Experience in the use of the model points out that additional effort involved in specifying preferences and priorities is relevant for it provides guidance for service selection and composition in the face of uncertainty about which services are available at any moment. Automating the elicitation of preferences and priorities is a particularly relevant direction of future effort, especially as research and practice moves towards automated service selection and composition in which preferences and priorities can be revised and new elicited at runtime, then fed to automated service composers. Combining Brafman and colleagues' results (Brafman et al. 2006) with the present quality model is of current interest.

Definition of quality dimensions and characteristics also requires guidance. Requirements engineering frameworks, such as, e.g., Tropos (Castro et al. 2002), can be used to obtain initial functional and quality goals of the service requesters. Metric definition needed for the writing of quality dimensions and quality characteristics has been treated for instance by Basili and Rombach (1988) whose Goal-Question-Metric approach can be deployed to obtain quality dimensions and characteristics in a systematic manner. The use of our quality model within these frameworks is a topic of ongoing work.

We do not address in this paper the issue of model compatibility, that is, the situation in which users and provider define the same quality property with different names or measurement transformation. One appropriate approach towards ensuring the compatibility of models amounts to annotate then with semantic information, then proceed to semantic matching – the reader may be interested in the approach suggested by Zeng and colleagues (Zeng et al. 2007), where service requests are translated by accounting for semantic information that is available. Such a process could be applied to quality specifications produced in our approach in order to avoid compatibility issues.

Another relevant consideration for future effort is the provision of more precise semantics for the various submodels and the subsequent more rigorous definition of the relationships between the proposed submodels. Achieving this aim requires formal semantics which can be obtained in two ways. One is to locate the model within a specific application domain, and therefore have domain-specific semantics. Another way is to attempt to provide a general formal semantics for the notion of quality dimension and build

from there. In both cases the effort is currently only preliminary. Careful choices among alternative semantics and their potential extensions is needed, and must therefore be relegated to a separate future treatment.

Further use of the suggested quality model will allow us to collect additional data from its users. At present, such data is limited. Future users need to be surveyed, so as to obtain their evaluation and indications on their experience with the model. Obtaining such data is critical, for it will allow more appropriate guidance to the identification and prioritization of the various directions for future work.

7 Conclusions

Expressive quality models are needed to let requesters specify quality expectations, providers advertise service qualities, and composers finely compare alternative services. Having observed many similarities between various quality models proposed in the literature, we reviewed these and integrated them into a single quality model for service-oriented systems. We advance current research by integrating precise submodels for dependency and priority information closely with concepts and constructs already established as relevant when performing quality modeling.

The proposed quality model, the QVDP, integrates concepts and constructs for extensive representations of quality information. A simple, yet realistic example illustrated the relevance of instantiating the various proposed concepts and constructs when performing quality modeling. We highlighted the need for integration of dependency and priority information within any quality model for services. We pointed out how lack of dependency and priority information limits the value of various prior quality models. As a solution, we proposed precise submodels for representing dependencies and priorities between quality dimensions and between quality characteristics. The comparison of QVDP with prior quality models indicated that extending a particular quality model with dependency and priority submodels is possible. It also highlighted that an expressive and practical quality model ought to closely integrate concepts and constructs for all of the quality information already established as relevant in related research, with the concepts and constructs of the dependency and priority submodels. To achieve such close integration between the various relevant concepts and constructs, we proposed QVDP as an integrative model instead of extending an existing quality model with the dependency and priority submodels. Our intention is for the proposed QVDP model to serve as a reference point for further developments in quality models for service-oriented systems.

While we have used simple examples to illustrate the use of QVDP, parts of the model have been employed and tested elsewhere (Jureta et al. 2007a, b). There, we QVDP to specify quality expectations in service requests and defining criteria to guide the learning of optimal service compositions. Our proposed extension of the UML metamodel for QoS (Object Management Group 2005) further facilitates the use of novel concepts and constructs present within QVDP in actual applications. Our current efforts focus on giving formal semantics to parts of the model so that some automated reasoning can be performed and enable verification of quality expectations expressed in service requests, along with the assessment and prediction of system quality.

Acknowledgments We are grateful to Emmanuel Mathot of the European Space Agency, who provided precise information about the GPOD project and assisted our efforts in describing quality information of services related to the GPOD project. The first author acknowledges funding from the Belgian ICM/CIM Doctoral Fellowship Program.

References

Basili, V. R., & Rombach, H. D. (1988). The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering, 14*(6), 758–773.

Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., et al. (2005). Semantic web services framework (swsf).

Berners-Lee, T., Hendler, J., & Lassila, O. (2001, May). The semantic web. *Scientific American*.

Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., Macleod, G. J., & Merrit, M. J. (1978). *Characteristics of software quality*. North-Holland.

Brafman, R. I., Domshlak, C., & Shimony, S. E. (2006). On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research, 25*, 389–424.

Brahnmath, G., Raje, R. R., Olson, A., Auguston, M., Bryant, B. R., & Burt, C. C. (2002). A quality of service catalogue for software components. In *Proceedings of the Southeastern Software Engineering Conference*.

Castro, J., Kolp, M., & Mylopoulos, J. (2002). Towards requirements-driven information systems engineering: the Tropos project. *Information Systems, 27*(6).

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). Web services description language (wsdl 1.1).

International Business Machines (IBM) Corporation. (2005). Service-oriented architecture. *IBM Systems Journal, 44*(4).

D'Ambrogio, A. (2006). A model-driven wsdl extension for describing the qos of web services. In *Proceedings of the International Conference on Web Services (ICWS'06)*.

Deming, W. E. (1982). *Quality, productivity, and competitive position*. Massachusets Institute of Technology, Center for Advanced Engineering Study.

Feigenbaum, A. V. (1951). Quality control: Principles, practice, and administration. McGraw-Hill.

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

Gravin, D. A. (1988). Managing quality: The strategic and competitive edge. Free Press.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition, 5*(2), 199–220.

Herssens, C., Jureta, I. J., & Faulkner, S. (2008a). Capturing and using QoS relationships to improve service selection. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE'08)*.

Herssens, C., Jureta, I. J., & Faulkner, S. (2008b). Dealing with quality tradeoffs during service selection. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'08)*.

Horrocks, I. (2002). DAML+OIL: A description logic for the semantic web. *IEEE Data Engineering Bulletin, 25*(1), 4–9.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2003). Swrl: A semantic web rule language combining owl and ruleml.

IEEE. (1989). *Software engineering standards*. IEEE.

International Organization for Standardization. (1986). *ISO 8402 Quality management and quality assurance—Vocabulary*. International Organization for Standardization.

International Organization for Standardization. (1998). Cd15935 information technology: Open distributed processing—reference model—quality of service.

Ishikawa, K. (1985). *What is total quality control? The Japanese way*. Prentice Hall.

Juran, J. M. (1951). *Quality control handbook*. McGraw-Hill.

Jureta, I. J., Faulkner, S., Achbany, Y., & Saerens, M. (2007a). Dynamic task allocation wihin an open service-oriented mas architecture. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS'07)*.

Jureta, I. J., Faulkner, S., Achbany, Y., & Saerens, M. (2007b). Dynamic web service composition within a service-oriented architecture. In *Proceedings of the International Conference on Web Services (ICWS'07)*.

Jureta, I. J., Faulkner, S., & Thiran, P. (2007c). Dynamic requirements specification for adaptable and open service-oriented systems. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'07)*.

Jureta, I. J., Mylopoulos, J., & Faulkner, S. (2008). Revisiting the core ontology and problem in requirements engineering. In *Proceedings of the International Conference on Requirements Engineering (RE'08)*.

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

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *IEEE Computer, 36*(1), 41–50.
Loyall, J. P., Schantz, R. E., Zinky, J. A., & 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., & Singh, M. P. (2004). Toward autonomic services trust and selection. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'04)*.
McIlraith, S. A., & Martin, D. L. (2003). Bringing semantics to web services. *IEEE Intelligent Systems, 18*(1), 90–93.
McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems, 16*(2), 46–53.
Object Management Group. (1997). The corba trading services.
Object Management Group. (2005, May). Uml profile for modeling qos and fault tolerance characteristics and mechanisms specification, v1.0.
Osterweil, L. (1996). Strategic directions in software quality. *ACM Computing Surveys, 28*(4), 738–750.
Papazoglou, M. P., & Georgakopoulos, D. (2003). Service-oriented computing. *Communications of the ACM, 46*(10), 24–28.
Ran, S., (2003). A model for web services discovery with QoS. *ACM SIGecom Exchanges, 4*(1), 1–10.
Reeves, C. A., & Bednar, D. A. (1994). Defining quality: Alternatives and implications. *The Academy of Management Review, Special Issue: Total Quality, 19*(3), 419–445.
Shadbolt, N., Berners-Lee, T., & Wendy, H. (2006). The semantic web revisited. *IEEE Intelligent Systems, 21*(3), 96–101.
Skene, J., Lamanna, D. D., & Emmerich, W. (2004). Precise service level agreements. In *Proceedings of the International Conference on Software Engineering (ICSE'04)*.
Staab, S., & Studer, R. (Eds.). (2004). *Handbook on ontologies. international handbooks on information systems*. Springer.
Staehli, R., Eliassen, F., Aagedal, J. O., & Blair, G. (2003). Quality of service semantics for component-based systems. In *Proceedings of the International Conference on Reflective and Adaptive Middleware Systems*.
Tennenhouse, D. (2000). Proactive computing. *Communications of the ACM, 43*(5), 43–50.
Tosic, V., Esfandiari, B., Pagurek, B., & 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)*.
Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J. & Sheng, Q. Z. (2003). Quality driven web services composition. In *Proceedings of the International World Wide Web Conference (WWW'03)*.
Zeng, L., Lei, H., & Chang, H. (2007). Monitoring the QoS of web services. In *Proceedings of the International Conference on Service Oriented Computing (ICSOC'07)*.
Zhou, C., Chia, L.-T., & Lee, B.-S. (2004). Daml-qos ontology for web services. In *Proceedings of the International Conference on Web Services (ICWS'04)*.

Author Biographies

Ivan J. Jureta has, after graduating, summa cum laude, received the Master in Management and Master of International Management, respectively, at the Université de Louvain, Belgium, and the London School of Economics, both in 2005. He is currently completing his Ph.D. thesis at the University of Namur, Belgium, under Prof. Stéphane Faulkner's supervision. His thesis focuses on quality management of adaptable and open service-oriented systems enabling the Semantic Web.

Caroline Herssens received a Master Degree in Computer Science in 2005 at the Université de Louvain. In 2006, she graduated a Master in Business and Administration from the University of Louvain, with a supply chain management orientation. She is currently a teaching and research assistant and has started a Ph.D. thesis at the information systems research unit at Université de Louvain. Her research interests comprise service-oriented computing, conceptual modeling and information systems engineering.

Stéphane Faulkner is an Associate Professor in Technologies and Information Systems at the University of Namur (FUNDP) and an Invited Professor at the Louvain School of Management of the Université de Louvain (UCL). His current research interests revolve around requirements engineering and the development of modeling notations, systematic methods and tool support for the development of multi-agent systems, database and information systems.