Grant Agreement N° 215483

| | |
|---|---|
| *Title:* | *Initial Concepts for Specifying End-to-End Quality Characteristics and Negotiating SLAs* |
| *Authors:* | *CITY, FBK, INRIA, Polimi, SZTAKI, Tilburg, UCBL, UniDue, UPM, USTUTT* |
| *Editors:* | *Kyriakos Kritikos (Polimi) and Barbara Pernici (Polimi)* |
| *Reviewers:* | *Raman Kazhamiakin (FBK)*<br>*Manuel Carro (UPM)* |
| *Identifier:* | *CD-JRA-1.3.3* |
| *Type:* | *Deliverable* |
| *Version:* | *1.3* |
| *Date:* | *June 16, 2009* |
| *Status:* | *Final* |
| *Class:* | *External* |


**Management summary**

The aim of this deliverable is two-fold. Firstly, it aims at defining the initial concepts for specifying and negotiating end-to-end quality, i.e., a service quality meta-model suitable for the definition and negotiation of service quality specifications and SLAs. The research method for creating this quality meta-model follows a design approach. Initially, requirements are collected dictating the information, structure, and constraints that this meta-model should capture. Then, based on these requirements, the meta-model is designed and finally created. Secondly, this deliverable aims at proposing a methodology for decomposing end-to-end quality into quality specifications for individual SLAs. The research method for achieving this goal follows a hybrid approach: a proof-of-concept and a paper-based approach. In particular, the meta-model's effectiveness and sufficiency is highlighted by modeling a composite service negotiation scenario and its result, which is a decomposition of end-to-end quality into quality specifications of individual SLAs. Then, initial attempts (materialized in papers of WP members) are provided that address (composite) service negotiation.


---




*The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n 215483 (S-Cube).*

**Members of the S-Cube consortium:**

| | |
|---|---|
| University of Duisburg-Essen (Coordinator) – UniDue | Germany |
| Tilburg University – Tilburg | Netherlands |
| City University London – CITY | U.K. |
| Consiglio Nazionale delle Ricerche – CNR | Italy |
| Center for Scientific and Technological Research – FBK | Italy |
| French Natl Institute for Research in Computer Science and Control – INRIA | France |
| The Irish Software Engineering Research Centre – Lero | Ireland |
| Politecnico di Milano – Polimi | Italy |
| MTA SZTAKI – Computer and Automation Research Institute – SZTAKI | Hungary |
| Vienna University of Technology – TUW | Austria |
| Universit Claude Bernard Lyon – UCBL | France |
| University of Crete – UOC | Greece |
| Technical University of Madrid – UPM | Spain |
| University of Stuttgart – USTUTT | Germany |
| University of Amsterdam – VUA | Netherlands |
| University of Hamburg – UniHH | Germany |

**Published S-Cube documents**

These documents are all available from the S-Cube Web Portal at `http://www.s-cube-network.eu/`

# The S-Cube Deliverable Series

## Vision and Objectives of S-Cube

The Software Services and Systems Network (S-Cube) will establish a unified, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution, helping shape the software-service based Internet which is the backbone of our future interactive society.

By integrating diverse research communities, S-Cube intends to achieve world-wide scientific excellence in a field that is critical for European competitiveness. S-Cube will accomplish its aims by meeting the following objectives:

- Re-aligning, re-shaping and integrating research agendas of key European players from diverse research areas and by synthesizing and integrating diversified knowledge, thereby establishing a long-lasting foundation for steering research and for achieving innovation at the highest level.
- Inaugurating a Europe-wide common program of education and training for researchers and industry thereby creating a common culture that will have a profound impact on the future of the field.
- Establishing a pro-active mobility plan to enable cross-fertilisation and thereby fostering the integration of research communities and the establishment of a common software services research culture.
- Establishing trust relationships with industry via European Technology Platforms (specifically NESSI) to achieve a catalytic effect in shaping European research, strengthening industrial competitiveness and addressing main societal challenges.
- Defining a broader research vision and perspective that will shape the software-service based Internet of the future and will accelerate economic growth and improve the living conditions of European citizens.

S-Cube will produce an integrated research community of international reputation and acclaim that will help define the future shape of the field of software services which is of critical for European competitiveness. S-Cube will provide service engineering methodologies which facilitate the development, deployment and adjustment of sophisticated hybrid service-based systems that cannot be addressed with today's limited software engineering approaches. S-Cube will further introduce an advanced training program for researchers and practitioners. Finally, S-Cube intends to bring strategic added value to European industry by using industry best-practice models and by implementing research results into pilot business cases and prototype systems.

S-Cube materials are available from URL: `http://www.s-cube-network.eu/`

# Foreword

The aim of this deliverable is twofold.

- Firstly, it aims at providing the initial concepts for specifying and negotiating end-to-end quality, i.e., a service quality meta-model suitable for the definition and negotiation of service quality specifications and SLAs. The research method for creating this quality meta-model follows a design approach. Initially, requirements are collected dictating the information and constraints that this meta-model should capture. Then, based on these requirements, the meta-model is designed and finally created.
- Secondly, it aims at proposing a methodology for decomposing end-to-end quality into quality specifications for individual SLAs. The research method for achieving this goal follows a hybrid approach: a proof-of-concept and a paper-based approach. In particular, the meta-model's effectiveness and sufficiency is highlighted by modeling a composite service negotiation scenario and its result, which is a decomposition of end-to-end quality into quality specifications of individual SLAs. Then, initial attempts (materialized in papers of WP members) are provided that address (composite) service negotiation.

**Acknowledgments:** The editors would like to thank Andreas Metzger for his valuable help in planning this deliverable. A special thanks goes to the CITY team and especially to Khaled Mahbub for his helpful and detailed comments on the first two main sections of the deliverable. Finally, we thank all S-Cube members who have contributed to this deliverable and especially Raman Kazhamiakin and Manuel Carro for their valuable and detailed comments on earlier versions of the document.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

As Service-Based Applications (SBAs) operate in a highly dynamic environment, techniques are needed to aggregate individual quality levels of the services involved in the SBA composition in order to determine and thus check the end-to-end quality during run-time. This aggregation will typically span different layers of a service-based application and thus a common understanding of what the different quality attributes mean within and across these layers is needed. As of today there are only a few techniques and methods that address the quality characteristics in a comprehensive and cross-cutting fashion across all layers of a service-based application and integrating all phases of SLA conception, negotiation, agreement, monitoring and refinement. In addition, very few approaches address observing the context of a service-based application and its impact on quality.

In the previous deliverable of this WP, namely CD-JRA-1.3.2, the S-Cube Quality Reference Model (QRM) was devised. In this QRM, a set of relevant quality attributes was defined and structured accordingly into different quality groups. Quality attributes were collected from different research disciplines and consolidated into a coherent and extensive quality model. Different kinds of quality attributes that are important to SBAs were included like Quality of Service (QoS), Quality of Experience (QoE), and Quality of Information (QoI). While QoS is a cross-cutting kind of quality attribute relevant to services at each functional layer, other kinds of quality are only relevant to specific functional layers. For instance, QoI attributes are important for the infrastructure and service layer. So, this QRM provides an understanding of which quality attributes are relevant to each functional layer, how they are defined, and how they relate to each other. However, this understanding is just the first step.

In the current literature, quality attributes are assumed to be independent of each other. As a consequence, quality offerings and SLAs are defined based on this assumption containing in this way only unary constraints on some quality metrics measuring quality attributes. However, in order to support end-to-end quality provision, the dependencies between different quality attributes at the same or different functional layers and between the same quality attributes across the layers have to be modeled and defined. In this way, the effect that one quality attribute has on another one will be explicitly defined and considered during end-to-end quality aggregation.

Apart from the lack of modeling quality attribute dependencies, existing languages for quality definition offer limited capabilities for automated negotiation and quality-aware service composition. Moreover, they only provide isolated and fixed sets of quality attributes for expressing quality capabilities or requirements. Actually, the survey carried out in the first deliverable of this WP, namely PO-JRA-1.3.1, has uncovered the lack of a well established, rich, extensible, and semantically enriched quality definition language. This survey also revealed that service quality capabilities and requirements, as well as SLAs are described by many different formalisms and languages. So, the differences between these formalisms limit the fulfillment of the vision of automated and precise quality-based service matchmaking, selection and negotiation, as well as quality-aware service composition.

To this end, this deliverable designs and develops a quality meta-model encompassing the concepts for a rich, extensible, and semantically enriched quality definition language. This meta-model allows the description of every relevant aspect of quality for services and SBAs, including attributes, metrics, units, measurement functions and directives, constraints, value types, etc. It can also be used to describe the S-Cube's QRM and is extensible so as to allow the addition of new quality attributes when it is needed. In addition, it is semantically-enriched in order to be machine-processable or machine-interpretable. Moreover, it is capable of expressing quality capabilities and SLAs by using functions, operators and comparison predicates on quality metrics. Finally, this meta-model expresses quality attribute dependencies and allows the description of composition rules for possible combinations of composition constructs and quality metrics.

In addition to aggregating end-to-end quality, this deliverable also deals with the opposite problem: supposing that there are end-to-end quality requirements requested by a service requester, how end-to-end quality can be decomposed into individual service quality of the component services embodied into service quality specifications and SLAs. The tackling of this problem is crucial in cases where a composite service provider has to assure that the end-to-end quality offered by his service and agreed with a service requester is guaranteed. This deliverable approaches this instance of the end-to-end quality assurance problem by relying on the assumption that the composite service provider is engaged in a complex negotiation with many service providers, which have services that match the component services of the composite service, during or after the negotiation with the service requester. The complexity of this kind of negotiation is that all individual service negotiations depend on each other and have specific quality goals to achieve. So when one negotiation fails, then end-to-end quality has to be re-decomposed, while already agreed individual SLAs have to be re-negotiated and the other non-finished negotiations have to be reset.

As this problem is novel and very complex, there are currently very limited research approaches solving it. One goal of this WP is to provide solutions to this problem. To this end, this deliverable proposes initial solutions, which are based on three research approaches [1, 2, 3] of the WP members. These approaches along with another complementary one [4] can be used to solve this problem in a joint way. An explanation is given on how these research approaches can be used to solve the problem and what is still missing.

## 1.2 Research Method

There are two different objectives for this deliverable. For this reason, the developed methodology to address each objective was different:

- In order to provide the initial concepts for specifying and negotiating end-to-end quality, i.e. a quality meta-model, the work carried out in this deliverable followed a design approach. Firstly, requirements were collected dictating what information, structure, and constraints should be captured by the meta-model for service quality definition and negotiation. Then, these requirements were prioritized, filtered or extended, and structured into a coherent ordered and structured set. This set of requirements is materialized in section 2. Next, based on these requirements, an initial version of the meta-model was designed and developed. This meta-model was revised based on the requirements and the comments of the WP partners and took the final form that is highlighted and analyzed in section 3.

- In order to propose a methodology for decomposing end-to-end quality into quality specifications for individual SLAs, the work carried out in this deliverable followed both a proof-of-concept and a paper-based approach. To explain, the meta-model's effectiveness and sufficiency is highlighted by modeling a composite service negotiation scenario and its result, which is a decomposition of end-to-end quality into quality specifications of individual SLAs. Then, initial attempts (materialized in partner papers) are provided that can address composite service negotiation jointly and

thus give the solution to this WP's objective. Both of these approaches of our end-to-end quality decomposition methodology are analyzed in section 4.

## 1.3 Relation to the WP Challenges

In the following, we give a short summary of the research challenges in WP JRA-1.3 that concern the three main activities in the life-cycle of electronic contracts and then explain how this deliverable is related to them.

1. For the *Quality definition* (*contract definition*) activity, the overall research challenge is *modelling of end-to-end quality for service-based applications*, which is decomposed in the following research challenges:

   (a) *End-to-End Quality Reference Model*: In order to support end-to-end quality provision, we will aim at making the dependencies between different kinds of quality attributes explicit. In addition, we aim at understanding the dependencies between quality attributes at the same and different functional levels. One key means to achieve the above objective is to achieve a shared understanding of quality attributes between the S-Cube layers and disciplines by defining the S-Cube Quality Reference Model.

   (b) *Rich and Extensible Quality Definition Language*: We plan to develop a quality definition language, which allows describing every relevant aspect of quality for services and SBAs, including metrics, units, measurement functions and directives, constraints, value types, etc. In addition, this quality definition language will encompass a rich set of domain-dependent and global quality attributes (i.e, the ones referenced in the S-Cube Quality Reference Model) and will be extensible so as to allow the addition of new quality dimensions when needed. Further, this quality definition language will be semantically enriched – where feasible – to be machine-processable or machine-interpretable. Finally, this language must be applicable in complex SBAs, in which services can be invoked and composed with variable quality profiles. So, it must be capable of expressing quality capabilities and SLAs by using functions, operators and comparison predicates on quality metrics. It should also allow the description of composition rules for possible combinations of composition constructs and quality metrics. These rules can be used for aggregating the individual quality levels of the services involved in a service composition in order to determine and thus ultimately check end-to-end quality.

2. For the *Quality negotiation and agreement* (*contract establishment*) activity, the overall research challenge is *Techniques for Automated and Proactive Quality Contract Negotiation and Agreement*, which is decomposed in the following research challenges:

   (a) *Exploiting user and task models*: One key research objective regarding quality contract establishment is to exploit user and task models (please see deliverable PO-JRA-1.1.3), which codify user preferences and characteristics, in order to devise advanced automated negotiation techniques and protocols. Those advanced techniques could lead to service negotiators (e.g., autonomous components provided as core services) that perform the negotiation process on behalf of the service consumers (requestors) and providers.

   (b) *Proactive SLA negotiation and agreement*: Based on the envisioned advances in automated negotiation, we aim to address the current state-of-the-art limitations by starting negotiation when there is evidence that the need for deploying a new service and/or change the conditions of deploying a current service is likely to arise but has not arisen yet. Thus, our proactive negotiation approach is based on forecasting at run-time a number of factors related to the

deployment of services. The availability of accurate forecasts can lead to effective proactive run-time negotiation strategies for service clients.

3. For the *Quality assurance* (*contract enactment*) activity, the overall research challenge is *Novel run-time techniques for predicting quality attributes*, which is decomposed in the following research challenges:

   (a) *Run-time Quality Assurance Techniques*: S-Cube will investigate how standard and consolidated offline software quality assurance techniques can be extended to be applicable while the application operates. In addition to extending the quality assurance techniques to the operation phase, synergies between the different classes of analytical quality assurance techniques will be exploited.

   (b) *Quality Prediction Techniques to Support Proactive Adaptation*: To support the vision of proactive adaptation, novel quality prediction techniques need to be devised. Depending on the kind of quality attribute to be predicted, these can range from ones that built on traditional techniques to ones that exploit modern technologies of the Future Internet.

In this deliverable, we deal with challenge 1b by designing and implementing a rich and extensible quality definition language, i.e., a service quality meta-model. Moreover, we deal with challenge 2a and partially 2b by analyzing a research approach [1] that defines and exploits user and task models and two other research approaches [2, 3] that advance the state-of-the-art in service contract negotiation and agreement. In addition, challenge 2a is also addressed in the meta-model, where specific concepts like negotiation strategies, cost and service selection models, quality capabilities and requirements, negotiable quality attributes and metrics represent information which is explicitly modeled but can also be drawn from user and task models. This information will be the basis for devising advanced automated negotiation techniques and protocols.

# Chapter 2

# Requirements for the S-Cube Quality Modeling Language

Before starting this section, we must first explain that a service quality description model is a meta-model and not a model. The difference between a service quality meta-model and a service quality model is that the former contains those language concepts that are required for specifying quality attributes and other quality entities across all service layers (*BPM*, *Service* and *Infrastructure*), while the latter is a conceptual model or taxonomy that contains selected, concrete quality attributes and other concrete quality entities. For example, the S-Cube Quality Reference Model is a service quality model. In some cases, the service quality model can also contain specific dependencies between quality attributes. A service quality meta-model is implemented by a service quality specification language that is used to create and specify a service quality model.

The service quality meta-model that has to be developed should be reusable across the service life-cycle, especially for the service discovery and negotiation activities. For this reason, the meta-model's design was based on a set of requirements collected from these two activities. Concerning quality-based service discovery, after studying and processing several research efforts, Kritikos and Plexousakis [5] have come up with some general and specific requirements that this service quality meta-model must satisfy. These requirements have been refined and extended with requirements affecting the negotiation of SLAs in order to produce the final collection for the service quality meta-model. An analysis of these requirements is given below.

## 2.1 Extensible and Formal Service Quality Model

In the presence of multiple services with overlapping or identical functionality, service requesters need objective quality criteria to distinguish one service from another. It is not practical to come up with a standard service quality model that can be used for all services in all domains [6]. This is because quality is a broad concept that can encompass a number of context-dependent non-functional properties such as privacy, reputation and usability. Moreover, when evaluating quality of services, domain specific criteria must be taken into consideration. For example, in the domain of phone service provisioning, the penalty rate for early termination of a contract and compensation for non-service, offered in the service level agreement are important quality criteria in that domain. Therefore, an extensible service quality model must be proposed that includes both the generic and domain specific criteria. In addition, new domain specific criteria should be added and used to evaluate the quality of services without changing the underlying computation (i.e., matchmaking and ranking) model. Finally, the semantics of quality attributes/concepts must be described in order to ensure that both service provider and consumer talk about the same quality attribute/concept. Sometimes, generic quality attributes with the same name like "application availability" may have different meanings to the parties that describe them (network level of

the hosting system, application that implements the service) or they may be computed differently. Other times, domain-dependent quality attributes may have a different name but obviously the same meaning. So it is important to describe quality attributes/concepts not only syntactically but also semantically in order to have a better discovery (matchmaking) process with higher precision and recall.

## 2.2 Syntactical Separation

Service quality specifications should be syntactically separate from other parts of service specifications, such as interface definitions (WSDL). This separation allows the specification of different quality properties for different implementations of the same interface [7]. Moreover, while functional constraints rarely change during runtime, quality constraints do change quite often. So the separation of service quality offerings from WSDL descriptions enables that, if needed, service quality offerings can be deactivated, reactivated, created, or deleted dynamically according to the change of quality without any modification of the underlying WSDL file [8]. Last but not least, a service quality offer could be referenced from multiple WSDL files and thus be reused for different services [9].

## 2.3 Both Client and Service Quality Specifications

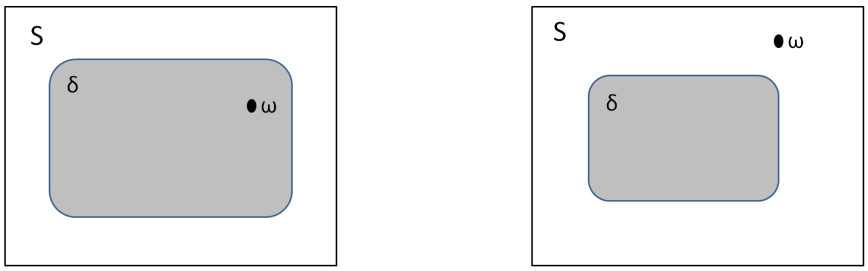
It should be possible to specify both the quality properties that clients require and the quality properties that services provide [7]. Moreover, these two aspects should be specified separately so that a client-service relationship has two quality specifications: a specification that captures the client's requirements and a specification that captures the service's offering. This separation allows us to specify the quality characteristics of a component, the quality properties that it provides and requires, without specifying the interconnection of components. The separation is essential if we want to specify the quality characteristics of components that are reused in many different contexts.

## 2.4 Refinement of Service Quality Specifications

As previously described, syntactical separation provides reusability. Apart from reusability, another form of extensibility is equally important; service quality specifications should not only be reused but also be refined. Refinement means creating a new service quality offering by referencing an older one and by adding constraints like refinement of an older quality restriction or creation of a new one [7, 10]. In addition, templates of service quality offerings can be created, usually for every application domain [7, 11].

## 2.5 Fine-Grained Service Quality Specifications

It should be possible to specify quality properties at a fine-grained level [7]. As an example, performance characteristics are commonly specified for individual operations. As another example, the *data policy* dimension is applicable to arguments and return values of operations. A service quality model must allow service quality specifications for the service, its interfaces, operations, attributes, operation parameters, and operation results. Generally speaking, any service object can have quality attributes (e.g., elements defined in WSFL [12]) [13]. Moreover, as a service can be composite and composite services are actually Service Based Applications (SBAs), we also capture the requirement that quality should be defined for both the SBA (i.e., service) and its constituent services (which are actually external operations for the SBA/service).




(a) Conformant (b) Non-conformant

Figure 2.1: Conformance in asymmetric models

## 2.6 Symmetric Model of Service Quality Specifications

In [14], it is endorsed that a symmetric model of QoS specification must be adopted. The reason for this lies beneath.

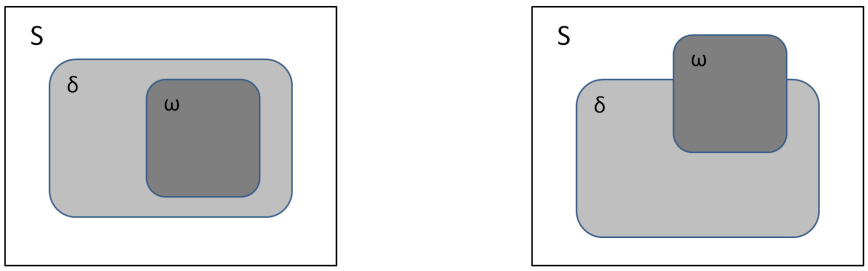
### 2.6.1 Asymmetric Models

Let $S$ be a multidimensional space whose dimensions are given by domains of quality parameters. Traditionally, a demand $(\delta)$ has been viewed as a subspace in $S$, whereas an offer $(\omega)$ has been viewed as a point in $S$. Thus, checking the conformance amounts to checking whether the point (the offer) belongs to the subspace (the demand) or not. See Figures 2.1(a) and 2.1(b), respectively. This checking can be computed easily by evaluating $\omega$ in $\delta$. As an example, if a service owns the offer $\omega = \{MTTF = 120\}$, then it is conformant to the demand $\delta_1 = \{MTTF \geq 100\}$ because $120 \geq 100$, but not to the demand $\delta_2 = \{MTTF > 120\}$ because $MTTF_{\delta_2} > MTTF_{\omega}$. It must be noted here that the variable $MTTF$ represents the Mean-Time-To-Failure metric measuring *Reliability* [15].

This interpretation of conformance results in a model which is asymmetric with regard to the expressiveness of service quality specifications. This semantics makes it very difficult to specify offers when it is needed something else than a point, as an example to specify some uncertainty or a space. However, checking space inclusion is harder and this is the reason most proposals have opted for the point inclusion method. As well, these approaches with an asymmetric model usually own a limited expressiveness because conditions are restricted to simple expressions involving single parameters, so complex expressions are not allowed.

### 2.6.2 Symmetric Models

Alternatively, an offer can be also considered as a sub-space, just as demands, so that it represents the ranges of quality-of-service values that the corresponding service guarantees to supply. In this way, an offer $(\omega)$ is conformant to a demand $(\delta)$ whenever the offer's sub-space is inside the demand's sub-space (see Figure 2.2(a)), otherwise the offer is not conformant (see Figure 2.2(b)). As an example, if a service owns the offer $\omega = \{MTTF \geq 120\}$, then it is conformant to the demand $\delta_1 = \{MTTF \geq 100\}$, but not to the demand $\delta_2 = \{MTTF > 120\}$ because the offer's instance value $\{MTTF = 120\}$ is out of the demand's space. It must be noted that in this example the offer does not satisfy the constraint of the second demand. However, the offer may partially match the second demand if it respects some of the demand's constraints. In this way, the violation of one constraint of the service quality demand does not condemn a service quality offer to be totally rejected from the useful results of the quality-based service matchmaking process.




(a) Conformant (b) Non-Conformant

Figure 2.2: Conformance in symmetric models

Continuing the previous example and analysis, sometimes the bounds of service quality specifications have to be defined softly and not crisply (e.g., when the bounds are based on measurements which entail a measurement error). So, this case, where the offer is $\omega = \{MTTF \geq 120\}$ and the demand is $\delta_2 = \{MTTF > 120\}$, should not result in a rejection of the offer, as the bound of the demand can entail a small but important measurement error. One way to solve this problem is to introduce weights to the "doubtful" bounds signifying their confidence and techniques like semiring-based constraint satisfaction can be used to perform the matchmaking.

The above interpretation of conformance results in a symmetric model because service quality in demands and offers can be specified in the same way. These semantics make the offer guarantee the complete range, not only a concrete value, i.e., we can not make any assumption on a concrete value, because it is equally possible any value in the subspace, and there is no control to get a concrete value. As well, symmetric approaches usually achieve a greater deal of expressiveness to specify quality-of-service, since there is usually no restriction on the number of involved parameters or type of operators, so that non-linear or more complex expressions are allowed. Finally, it must be noted that the asymmetric models are particular cases of the symmetric models, if the upper and lower bounds of all quality attributes or metrics coincide in the symmetric approach.

## 2.7 Relevant Aspects for Service Quality Attributes Models

A service quality attributes model is a part of a service quality model referencing a set of concrete quality attributes and their metrics. For each domain, the attributes in that domain are important inputs to the overall quality of a service. Some attributes are common across domains and some are specific to domains. Each quality attribute is measured with the help of one or more quality metrics. Each attribute and metric must have the following aspects [16, 17, 15].

### 2.7.1 Basic Aspects

#### Associated Attributes and Concepts

**Unique names** Quality attributes and metrics should have unique names in their domain. This does not prevent attributes from having different names in different domains but imposes the constraint that they should have a unique name only in their domain. Domain-independent attributes should also have unique names. When this naming constraint can be overridden, then equivalence rules should be defined that infer the equivalence of the same but differently named quality attributes or metrics.

**Weight** The weight of the attribute or metric relative to its domain and user preferences. This weight can also help in calculating the rank of a service quality offering.

**Domain** The domain that this attribute or attribute belongs to. For instance, is it a cross-domain attribute or an attribute specific for a domain?

**Functional level** The functional level that this attribute refers to. There can be three different levels: a) *business*, b) *application or service*, and c) *infrastructure/resource*, which actually represent the functional layers introduced in the S-Cube project. At each of these three levels, different quality attributes are usually present. For example, in the application level we have *availability* and *reliability*, while in the resource level we have *memory usage* and *available bandwidth*. However, there are also attributes that are common in some levels so service quality specifications should clearly distinguish between them by stating the level that they refer to. For example, the availability of the service is different from the availability of the resources it uses although it depends on it.

**Metric value type** A metric should have a value type indicating its allowed value range. For instance, a metric such as failure rate measuring availability can be expressed by the set [0.0,1.0]. The *MTBF* (Mean Time Between Failures) metric measuring reliability can be instead expressed with an integer in [0,365].

**Measurement scale** The meta-model should allow to specify the measurement scale of quality metrics. A measurement scale controls the value type and the type of operations allowed for a metric. It also specifies indirectly the way one value expression bound to one scale can be transformed to another value expression of another compatible scale (both scales belonging to the same metric). So specific scales can be compatible if they belong to the same scale type and there is a *scale transformation function* that transforms their expressions into each other. A scale can be categorized into five types: *nominal*, *ordinal*, *interval*, *ratio* and *absolute* [18]. *Nominal* scales concern metrics that have as value type a set of unrelated numbers or strings. The members of this set cannot be compared (no ordering). Specific nominal scales can be compatible if there is a *one-to-one mapping function* between their corresponding value types. *Ordinal* scales apply to metrics that have an ordered set as value type. Metrics belonging to different ordinal scales cannot be added, multiplied, divided or subtracted in QoS constraints. We can transform one ordinal scale expression into another one with the help of *monotonic functions*. *Interval* scales preserve not only ordering but also differences. However, they do not preserve ratios. The operations of addition and substraction are allowed between different interval metrics. We can transform one interval scale expression into another one with the help of *affine transformation functions* of the form: $M = a * M^{'} + b$. *Ratio* scales preserve ordering, size of intervals and ratios. In a ratio scale there is always a zero element representing the total lack of the measured attribute. All arithmetic operations are allowed between different ratio metrics. We can transform one ratio scale expression into another one with the help of *mapping functions* of the form: $M = a * M^{'}$. Finally, the following facts are true for an absolute scale: **a)** measurement is made simply by counting the number of elements in the measurement set; **b)** measured attribute takes the form: "num of occurrences of $x$ in the entity"; **c)** all arithmetic analysis is meaningful; **d)** the set of acceptable transformations between different absolute scale expressions is the *identity transformation function*.

**Metric's ordering function** The characteristic of the function from metric values to overall quality values. For instance, some metrics such as the one measuring *availability* are monotonic, at least in typical business scenarios. That is, the more the metric of availability increases the better. As far as negotiation is concerned, quality metrics should have at least a partial ordering relation among their values [1]. So monotonic metrics are already appropriate as monotonicity is harder than partial ordering. Thus, the

partial ordering requirement should be true for every non-monotonic metric. As an example of partial ordering, for a metric measuring *availability* we can have the situation that specific set of values are mapped to different quality levels, e.g. $ql([0.9, 0.95)) = 1 < ql([0.95, 0.99)) = 2 < ql([0.99, 1.0]) = 3$ where $ql$ is a function mapping a set of values to a finite set of quality levels.

**Metric's temporal characteristic** Each metric's value is measured at a specific time-point so it should be associated with a timestamp. With this timestamp it will be possible to reason about which metric values are more recent than the others, as newer values should influence the values of statistical metrics (e.g., for *average availability*) more than older ones. Moreover, the values of a metric that are measured at a specific time interval of the day (e.g., values gathered at 17:00-18:00 every weekday) should be associated with each other in order to reveal trends in service performance as the service load can be specific at this time interval.

### Properties

**Dynamicity of quality metrics** The meta-model should specify whether the value of a quality metric is constant or varies with respect to the environment of the service. In the former case, the metric is *static*. As an example, the *MTBF* metric measuring a service's *reliability* usually remains constant as it depends on the service's host machine and the service implementation. In the later case, the value of the metric should be computed with respect to the change in the environment. For instance, a service's *availability* changes very frequently and depends on many environmental factors like the service demand.

**Measurability of quality attributes** Quality attributes can be *measurable* or *unmeasurable*. Measurable attributes are measured by quality metrics. For example, response time can be measured by a metric averaging the individual response times of a service in a specific time space. Unmeasurable attributes can not be measured. For instance, the security model attribute (i.e., defining the exact security model supported by a service) has a fixed value before service execution and is not measurable. However, these quality attributes have a specific value type and can have an ordering function that provides a partial ordering between the values they take.

**Controllability of quality attributes or their metrics** The meta-model should allow to specify the attributes that are not under control of the service provider and that may affect the quality of the offered service. For example, in case of services accessed over the Internet, service quality must be affected by the network properties and the service provider may not have any control over these properties. In this case, the service provider makes a special agreement with a network provider in order to cater for the user requirements regarding *response time* or other network-dependent and uncontrollable quality properties. For the above reason, quality attributes should be distinguished between *controllable* and *uncontrollable* ones.

**Negotiability of quality attributes or their metrics [1]** Not all quality attributes, either technical or domain-independent, are negotiable. A quality attribute is negotiable if its value can be set by the service provider at runtime. Non-negotiable quality attributes are the ones for which the value cannot be set at runtime by the service provider. For example, when a service is invoked, its *reputation* is fixed, regardless of the technique adopted for assessing *reputation*. Moreover, the service provider or requester can always decide whether to allow or not the negotiation of a specific quality attribute. For example, although *response time* can be considered as negotiable, it might be the case that a provider's provisioning infrastructure does not allow the adaptation of the value of this attribute based on the user requirements.

#### Aggregation

**Inter-attribute dependencies** An increase in a metric measuring *availability* is not always desired if it comes in conjunction with reductions in the values for other positively-monotonic metrics of attributes, as the overall quality might not improve. For instance, for a trip service, if the price of trips were to decrease while the promptness (on arrival and departure times) metric were to become worse, then this decrease in price might not help the overall quality of the trip service. The characteristics of metrics can be quite rich and need to be further categorized. For this reason, dependencies between quality attributes should be captured. There can be two types of dependencies: quantitative and qualitative. Quantitative dependencies explicitly denote the way a quality attribute can be measured by one or more other quality attributes and are usually expressed by formulas. For example, *response time* is computed by the addition of *network* and *service latency*. Qualitative dependencies express empirical relationships between quality attributes and often reflect the trade-offs providers make in their service implementations [19]. For example, *response time* is negatively correlated to throughput, that is an increase in the former's value leads to a decrease in the latter's value. Of course, qualitative dependencies can be characterized, in some cases, by a stochastic quantitative dependency model (e.g. a function derived from empirical data using regression) but this model would be different for every service. This is the main difference with respect to quantitative dependencies, where the formulas apply to every service and are not specific for each service.

**Quality grouping** Quality attributes could be grouped into Quality Groups. Quality Groups can contain other Quality Attributes and Quality Groups. For example, quality attributes *response time*, *latency*, *throughput*, *execution* and *transaction time* can belong to the *performance* Quality Group. As another example, the *performance* Quality Group could be contained in the more general Quality Group of *Runtime Related Quality*.

**Aggregation formulas on different levels** There must be a description (mathematical or otherwise formal) of how a quality metric's value of a complex service can be derived from the corresponding quality metrics' values of the individual services that constitute the complex one. For example, the execution time metric $T_C$ of a complex service $C$, which is defined as a sequence of two services $A$ and $B$, can be computed as the sum $T_A + T_B$ of the execution time metrics of the two individual services. This description is essential for the automated estimation of the values of quality metrics for a complex service that is composed of other services and individual operations. So this description is needed for automating the service quality analysis process, a prerequisite for a successful quality-based service discovery. In addition, it helps automating the service composition process and delaying individual service selection as late as possible (i.e., at runtime).

### 2.7.2 Additional Aspects

The above aspects of quality metric description are not enough. Additional aspects must be developed. In [20], it is described that there is a need for several semantic meta-models that would be used in the formal representation of quality and other constraints. These meta-models include: quality metric meta-model, measurement unit meta-model, measured properties meta-model and measurement methods meta-model.

#### Metric Meta-model

We have already described what should be the content of a measured properties meta-model. In the quality metric meta-model, a metric has a name and a short human-readable textual description. In addition, the measured property (e.g. time, quantity of information, information transmission rate, etc.) that the metric quantifies is referenced.

**Metric complexity** Moreover, metrics can be composite or simple (resource). Examples of composite metrics are: *maximum response time* of a service, *average availability* of a service, or *minimum throughput* of a service. Examples of resource metrics are: *system uptime*, *service outage period*, or *number of service invocations*. Resource metrics are directly measured from the instrumentation of the service's system through the use of Measurement Directives [13]. On the other hand, composite metrics can be computed by other quality metrics with the help of a formula.

**Metric's measurement methods** Each composite metric formula can contain mean, median, sum, minimum, maximum, and various other arithmetic operators, or time series constructors. It should also be accompanied by appropriate unit conversion rules. Measurement directives, formulas and unit conversion rules are concepts that should be defined in the measurement methods meta-model.

**Metric's scheduling** Keller and Ludwig [13] endorse that each formula must reference a *schedule* or a *trigger*. A schedule defines the time intervals during which the formulas are executed to compute the metrics, while a trigger defines a point in time to which the execution of monitoring activity can be tied. The measurement directives and formulas should clearly specify the layers of the Service-Based System where monitoring can be performed [4, 21]. E.g. the response time of service S can be measured at the interface level of S, or it can be translated into properties of the infrastructure on which S is running (e.g. required database queries, server load, and other properties can be captured from the infrastructure of S).

**Invariant metric relationships** According to [20], the semantic description of a quality metric can also specify invariant relationships with other quality metrics, particularly those that measure the same property. For example, "average response time" should always be less than or equal to "maximum response time". While such information is probably redundant, it is very important for quick and easy discovery of conflicts. In general, when several quality metrics are specified in the same service offering, checking various dependencies and relationships given in the ontology helps to avoid various conflicts.

### Unit meta-model

The quality metric meta-model should be accompanied by an appropriate measurement units meta-model. Such a meta-model should define three types of units: base units like "second" and "byte", multiples of base units like "millisecond" and "megabyte", and derived units like "transactionsPerSecond" and "bytesPerSecond". Synonyms, including abbreviations, should be specified for all three types. Semantic descriptions of all measurement units should contain information about what kind of quality property is measured. When a measurement unit is used for a particular quality metric, one can check whether the definition of the measurement unit and the definition of the quality metric refer to the same measured property. The three types of measurement units differ in what additional information should be specified in their semantic description.

**Currencies** Special types of measurement units are monetary units representing currencies, and units derived from them. Examples are "CanadianDollars" and "CanadianDollarsPerHour". Usually countries give a special name to 1/100 (or sometimes 1/1000) part of their currency (e.g., "CanadianCents") and this information should also be represented in the measurement units meta-model. Also, some multilingual countries have several synonyms (from different languages) for their currency. As exchange rates between currencies vary, it is important to capture their dynamicity with appropriate constructs. Thus, instead of specifying fixed formulas for converting between different currencies, a meta-model could reference one (or maybe more) currency conversion services that can be consulted for up-to-date conversions.

**Meta-model translations**

According to Tosic et. al. [20], an important issue that must be raised is the development of independent, third-party services for meta-model translations between different quality metrics, measurement units, and currencies. For example, a service can provide quality guarantees using "average throughput [invocations/s]" for a particular "number of invocations [invocations]", while one of its consumers can reason about quality using "average response time [ms]". To understand each other, they have to perform a meta-model translation. However, in some cases services will not be able perform meta-model translations themselves. For example, they might not understand the languages used to represent different meta-models. In such cases, the service or its consumer should consult a specialized external meta-model translation service for help during the process of service offering negotiation.

## 2.8 Expressiveness and Correctness in Constraint Definitions

A service quality offer or demand (i.e. specification) must be comprised of quality constraints. The basic representation of a quality constraint consists of a name, an operator, and a value [7]. The name is typically the name of a quality metric, although it can also be the name of a metric *aspect*. The definition of a metric aspect is given in the next paragraph. The permissible operators and values depend on the quality metric type. A metric type specifies a domain of values. These values can be used in constraints for that dimension. The domain may be ordered. For example, a numeric domain comes with a built-in ordering ("$<$") that corresponds to the usual ordering on numbers. Set and enumeration domains do not come with a built-in ordering; for those types of domains we have to describe a user-ordering of the domain elements. The domain ordering determines which operators can be used in constraints for that domain. For example, we can not use inequality operators ("$<$", "$>$", "$\geq$", "$\leq$") in conjunction with an unordered domain.

Apart from the above three parts that constitute each quality constraint, two further attributes should be associated with quality constraints. The first should state if the constraint is *hard* or *soft*. The second attribute models the *weight* of the constraint and signifies its importance. These two attributes of constraints can be very useful in the service matchmaking and negotiation processes when there are over-constrained quality demands and some of their quality constraints should be relaxed. Techniques like Mixed-Integer Programming, Constraint Optimization and Semiring-Based Constraint Satisfaction can be used for solving this problem [15].

Aspects are statistical characterizations of quality constraints like: *percentile*, *mean*, *variance*, and *frequency*. They are used for characterization of measured values over some time period. For example, the percentile aspect could be used to define an upper or lower value for a percentage of the measurements or occurrences that have been observed. Aspects can be proved to be very useful in cases where we want to guarantee that the measurements or occurrences of a quality metric present some special characteristics and we do not want to produce a new complex metric from the basic quality metric for each of these characteristics. However, they must be used carefully especially in cases where many aspects are created for one metric.

Quality constraints are usually connected by the "and" logical operator, although they can also be connected by other logical operators, into expressions. A service quality offer or demand should contain one complete expression or just one constraint.

Quality constraints should be joined into *Constraint Groups* (CG) or *Constraint Group Templates* (parameterized CGs) in order to be reused by many service quality specifications [8]. Other reusability constructs can also be created even for expressions.

Based on the above analysis, a constraint definition model has a *basic* expressiveness if it uses at least the basic constraint representation and the logical operator "and" to connect constraints into expressions. If other constructs are used, like aspects, types of constraints, more logical operators and constraint groups, then the constraint definition model is *rich* concerning its expressivity.

A constraint definition model must enable the description of *correct* constraint specifications, where *correct* means syntactic and semantic correctness of constraint expressions. Tools and techniques should be used that must check this type of correctness (after the constraint model creation) or disallow the wrong use of the constraint modeling constructs (during the constraint model creation) in order to assist the service provider and requester in creating their quality specifications correctly.

## 2.9 Support for Multiple Classes of Service Specifications

*Class of Service*, which is also commonly known as service level (e.g, see [22]), means the discrete variation of the complete service and quality provided by one service [10]. Classes of service make sense to be discussed at the level of services and not at the level of constraints or guarantees (e.g. response time) that are part of the overall service and quality. Classes of service can differ in usage privileges, service priorities, response times guaranteed to consumers, verbosity of response information, etc. The concept of classes of service also supports different capabilities, rights, and needs of potential consumers of the service, including power and type of devices they execute on. Further, different classes of service may imply different utilization of the underlying hardware and software resources and, consequently, have different prices. Additionally, different classes of service can be used for different payment models, like pay-per-use or subscription-based.

The issues of quality and balancing of limited underlying resources are particularly motivating for having multiple classes of service for services. If the underlying resources were unlimited, all consumers would always get the highest possible quality. Unfortunately, this is not the case, so it is reasonable to provide different quality to different classes of consumer. Providers of services want to achieve maximal monetary gain with optimal utilization of resources. Providing different classes of service and their balancing helps in achieving this goal because of the flexibility to accommodate several classes of consumer. On the other hand, consumers of such services can better select service and quality they need and are willing to pay for, while minimizing their price/performance ratio.

Providing classes of service is not the only possible way to customize constraints and management statements that a service offers to its consumers. There are various alternatives, including custom-made Service Level Agreements (SLAs), user profiles, parameterization, and separate ports. However, the practice of telecommunication service provisioning shows that classes of service have relatively low overhead and complexity of management.

From the above analysis, it is claimed that a service must provide multiple classes of service (i.e. constraints and management statements). So it is important to have a description of class of service included in the overall service description model. However, while the above discussion is interesting, we limit ourselves to quality constraints only and we do not take into consideration other non-functional constraints and management statements. Moreover, we consider that these other non-functional constraints and management statements are not very important to users as quality is, so they do not play an important role in the discovery process. Therefore, we sustain that it is of great interest to service providers that they provide multiple service quality offerings. In this way, they serve a wider range of consumers and they broaden their market segment. In addition, service consumers can have a wider solution space and (with the help of efficient quality-based discovery mechanisms) they can find the best solution (combination of the functional capability of service and its service quality offering) that minimizes their price/performance ratio. Thus, the service quality specification must support multiple service quality offerings of the same service of a service provider, an idea also adopted by other research efforts [9, 11].

## 2.10 Support for Alternative Service Quality Demand Specifications

Similarly to the previous requirement and in accordance with the requirement posed in section 2.6, service consumers should be allowed to specify alternative service quality demands. In this way, the probability of matching at least one of the alternative demands of the service consumer with one of the existing service quality offerings is increased. Moreover, service consumers can express a relaxation of one of their service quality demands to cope with cases where the initial demand (not its relaxation) is over-constrained (i.e, it cannot be matched with any of the existing service quality offerings). Finally, the alternative demand specification supports the definition of quality aspects that a client requests with respect to certain services (e.g., when using a service from a provider P, the client may accept to pay more given that the client trusts this service; or when using a service to find airplane tickets the client may tolerate a response time of X that it will not be tolerated if performing a banking transaction).

## 2.11 Other Useful Service Quality Information

A service quality offering should provide references to protocols needed for service management as well as entries of third parties that one side would be willing to trust (see [9, 1] and WS-Policy [23]). Moreover, a service quality offering must be related to the *price* element in order to relate the specified quality level to the cost of service usage per invocation. Finally, a service quality offering must have an *expires* attribute denoting the point in time until which the offer will be valid [9].

## 2.12 Negotiation-Specific Information

Apart from specifying which terms can be negotiated, how metric values are mapped to quality levels and which protocols are supported by the service provider or requester for performing negotiation, there is other equally important information that should be captured by the service quality meta-model. We separate this information into three parts and we analyze each part in the different subsections of this section.

### 2.12.1 Negotiation Actors and Roles

There are two different types of entities involved in a negotiation and a meta-model should make a clear distinction between them [1]. The first type represents the physical entities involved in the negotiation and these are the *negotiation actors*. These entities can be *users* or *agents*. Users delegate agents to take a specific role in a negotiation and act on behalf of them. When all the participants of a negotiation are agents, then the negotiation process can be fully automatic. Otherwise, it can be semi-automatic or manual.

The second type represents the logical entities involved in the negotiation which are the *negotiation roles*. These entities can be *service providers* or *service requesters* or *brokers*. Besides expressing capabilities for participating in a negotiation protocol, service providers and requesters have the capability of generating and evaluating offers within a negotiation. Conversely, brokers only expose capabilities that do not involve the generation or evaluation of negotiation offers.

By introducing a distinction between actors and roles, a meta-model enables a negotiation actor to take different negotiation roles in different negotiations. Let us consider the case of a user that requires the functionality of a certain service to complete an application package that he or she would like to sell to other potential consumers. In a first phase, the user takes the role of service requester and negotiates with other service providers in order to obtain the missing functionality for his or her application package. In a second phase, the same user takes the role of a service provider, looking for potential service requesters for its application package.

### 2.12.2 Negotiation Protocol

The negotiation protocol is the allowable sequence of messages exchanged used to negotiate and conclude (i.e., agree) a contract. In addition to defining the messaging behaviour the protocol should also unambiguously define the semantics and format, or schema, of the messages. This schema will indicate how both information relevant to the negotiation (e.g., the allocation of roles or the time allowed for negotiation, see below) and the description of the quality metrics of the resources being negotiated for should be rendered. To avoid confusion, the semantics of each message should also be formally defined; an example of how this can be achieved with the WS-Agreement protocol is given in [24].

Note that prior to the negotiation taking place, it is essential that the potential participants in a negotiation understand which/what negotiation protocol is supported. In order to determine this, techniques such as 'meta-negotiation' [25] or use of the WS-Policy framework [23], can be used to establish the negotiation protocol to be used in the negotiation proper.

### 2.12.3 Negotiation Strategy

The *strategy* of the participants (i.e service providers, requesters and brokers) involved in the negotiation should be defined. This strategy could be defined in the form of a set of *rules* that specify when an *action* like a generation of a new offer (i.e. service quality specification) or the acceptance or refusal of an offer should be taken.

### 2.12.4 Cost model

From the provider's point of view, the negotiation often relies on a *cost model* [1, 2], i.e., parameterized functions that are used to evaluate the cost sustained by the service provider for giving a certain level of quality in his or her service offers. Usually, cost models are additive [2], that is, the cost of a service (quality) offer is given by the costs associated by the cost model to each single quality value (of a metric) that appears in the offer. However, there will be cases that a service requester would also like to express his cost model (not in such an expressive way as the provider's one) instead of just a specific value representing his *budget*. In this way, he will be able to accurately express which quality level should be mapped to which price he wants to pay (less than or equal to his budget).

### 2.12.5 Exceptional conditions

The meta model should allow to specify the impact of exceptional conditions (e.g. denial of service, or service hardware failure) on the quality of service offerings.

### 2.12.6 Penalties

The meta model should allow to specify the consequence of not meeting a quality of service offering.

### 2.12.7 Time Limit

The meta model should allow the specification of a limit of time that the negotiators are prepared to negotiate, as well as maximum values or range of acceptable values for certain quality aspects.

# Chapter 3

# The S-Cube Quality Meta-Model

This chapter defines the required concepts for specifying end-to-end quality characteristics, service quality specifications, and SLAs by providing and analyzing the S-Cube's initial quality meta-model. This meta-model is carefully designed based on the requirements set in the previous section. It is separated into two main parts. The first part, named "Basic Quality Concepts", is dedicated to analyzing all the concepts required for defining service quality characteristics and metrics. This part stands as it is because it can be used for creating taxonomies of quality attributes and metrics, providing in this way a standard vocabulary of service quality terms that can be used in service quality contracts. Moreover, in this part of the meta-model, dependencies between quality attributes and aggregation rules for inferring end-to-end quality are modeled.

The second part of the meta-model, named "Quality Specification Constructs", builds on the first part by reusing some of its concepts in order to introduce the relationships that these concepts have with those concepts needed for creating and also representing service quality specifications and SLAs. So, the second part signifies not only how quality attributes and metrics can be used in order to populate service quality specifications and SLAs but also those (possibly non-quality) concepts that are needed in order to construct these specifications (e.g., contractors or third-parties, functional obligations, price, and penalties). In the remainder of this chapter, these two parts will be analyzed in detail, in the next two sections, by highlighting the main concepts introduced and their relationships.

## 3.1 Basic Quality Concepts

This section will be dedicated to analyzing the first part of the S-Cube's quality meta-model that concerns the definition of quality attributes and their related concepts. This part of the quality meta-model is displayed in Fig. 3.1. The main concept of the meta-model is the one defining a *Quality Attribute*. This concept has three main attributes, namely *name*, *URI*, and *weight*. The former two are used to uniquely identify the quality attribute, while the third one signifies the importance of this attribute with respect to the other quality attributes (either from a service provider or requester perspective).

Quality attributes can be distinguished between *Domain Independent* and *Domain Dependent* ones. The former represent quality attributes that are common among all application/business domains (e.g. *response time*), while the latter represent those quality attributes that are specific to an application domain (e.g. *covered area* in the *Traffic Monitoring* application domain).

A quality attribute can be either *Measurable* or *Unmeasurable*. Unmeasurable quality attributes cannot be measured at all. They represent static information about an entity which is qualitative in nature. For instance, the selection of a security model among the security models supported by a service is fixed before service execution and is not measurable. These attributes have a specific value type, they are unit-less and concern a specific service object. However, they can have an ordering function that provides a partial ordering between the values they take. Measurable quality attributes are measured through the abstraction of *Quality Metrics*. The latter concepts will be analyzed later on in this section.

Quality of Service (*QoS*) is a quality attribute sub-concept that represents those quality attributes that can be measured objectively or they are unmeasurable but can take objective values. So, for example, security attributes are QoS attributes but cannot be measured. However, the values that they take are objective. QoS attributes are typical constituents of SLAs (e.g *response time* and *availability*). On the other hand, Quality of Experience (*QoE*) represents those quality attributes that can be measured subjectively or they are unmeasurable but can take subjective values. These quality attributes reflect the perception of individuals or groups of service users (e.g *reputation* and *usability*).

Apart from QoS and QoE, other sub-concepts of quality attributes are the QoBiz and QoI [26]. QoBiz represents those quality attributes that are relevant to the business process level and they usually affect customer satisfaction and revenues. On the other hand, QoI represents those quality attributes that affect the quality of the information used by the service (e.g. *accuracy* and *completeness*).

Quality attributes can also be distinguished between *Functional* and *Non-Functional*. Functional quality attributes specify if a composite service behaves in a correct way (e.g. *safety*) and is interoperable (e.g. *compatibility*). On the other hand, non-functional quality properties are related more on the performance (like performance attributes) and other non-functional aspects than the functionality of the service.

*Composite* quality attributes (e.g. *response time*) are aggregated from other quality attributes, while *Atomic* ones (e.g. *capacity*) cannot be decomposed at all. The aggregation can be based on a formula for measurable quality attributes only. Other distinctive categorizations between quality attributes are the following. *Controllable* attributes (e.g. *execution time*) are always under the control of the service provider, while *Uncontrollable* ones (e.g. *network latency*) cannot be controlled by the service provider and may affect the quality of the offered service. A quality attribute is *Negotiable* if its value can be set by the service provider at runtime, while the values of *Non-Negotiable* quality attributes are specific and cannot be set at runtime by the service provider. So the negotiability of a quality attribute usually depends on the provider's capability to change its value during runtime to satisfy user needs.

Quality attributes can be logically organized into *Quality Group*s. For instance, *response time* and *throughput* are two quality attributes that belong to the *Performance* quality group. We have to note that we did not introduce another term here called *context*, which was used in the CD-JRA-1.3.2 deliverable, as it has the same usage and actual meaning as the *Quality Group* concept. Moreover, a quality attribute refers to a specific *Functional Level*. There can be three different types of functional levels: *business*, *application* and *resource*, and these levels are highlighted in the *Levels* enumeration. In this way, quality attributes that have the same name but refer to different functional levels are distinguished. For instance, a service's availability is different from the availabilities of the resources it uses although it depends on them.

One quality attribute cannot always be independent of all the other attributes. For example, *throughput* and *response time* are negatively correlated. So there can be dependencies between them. This knowledge is captured with the *Dependency* concept. There are two different relationships between quality attributes and dependencies. The *inDep* relation is used to connect all of the dependencies of a quality attribute with this attribute, while the *member* relation is used to connect those quality attributes that have a specific dependency between them with this dependency. Dependencies can be *Quantitative* or *Qualitative*. Quantitative dependencies contain a mathematical formula that expresses the way one attribute can be computed from other attributes. For instance, *response time* is always computed by adding *network latency* and *service latency*. Qualitative dependencies express empirical relationships between two or more quality attributes. They express the *impact* of change of a value of one attribute with respect to the new value that will be produced for the other attribute along with the *direction* (i.e., increase or decrease) of this new value. Considering the previous example of the negative correlation between *throughput* and *response time*, a small change in *response time* can produce a big (i.e., impact) negative (i.e., direction) change in *throughput*. However, the impact usually differs from service to service.

The *Quality Metric* concept is used to capture all the appropriate knowledge needed for measuring a quality attribute. The distinction between this concept and the *Quality Attribute* one is that if a quality

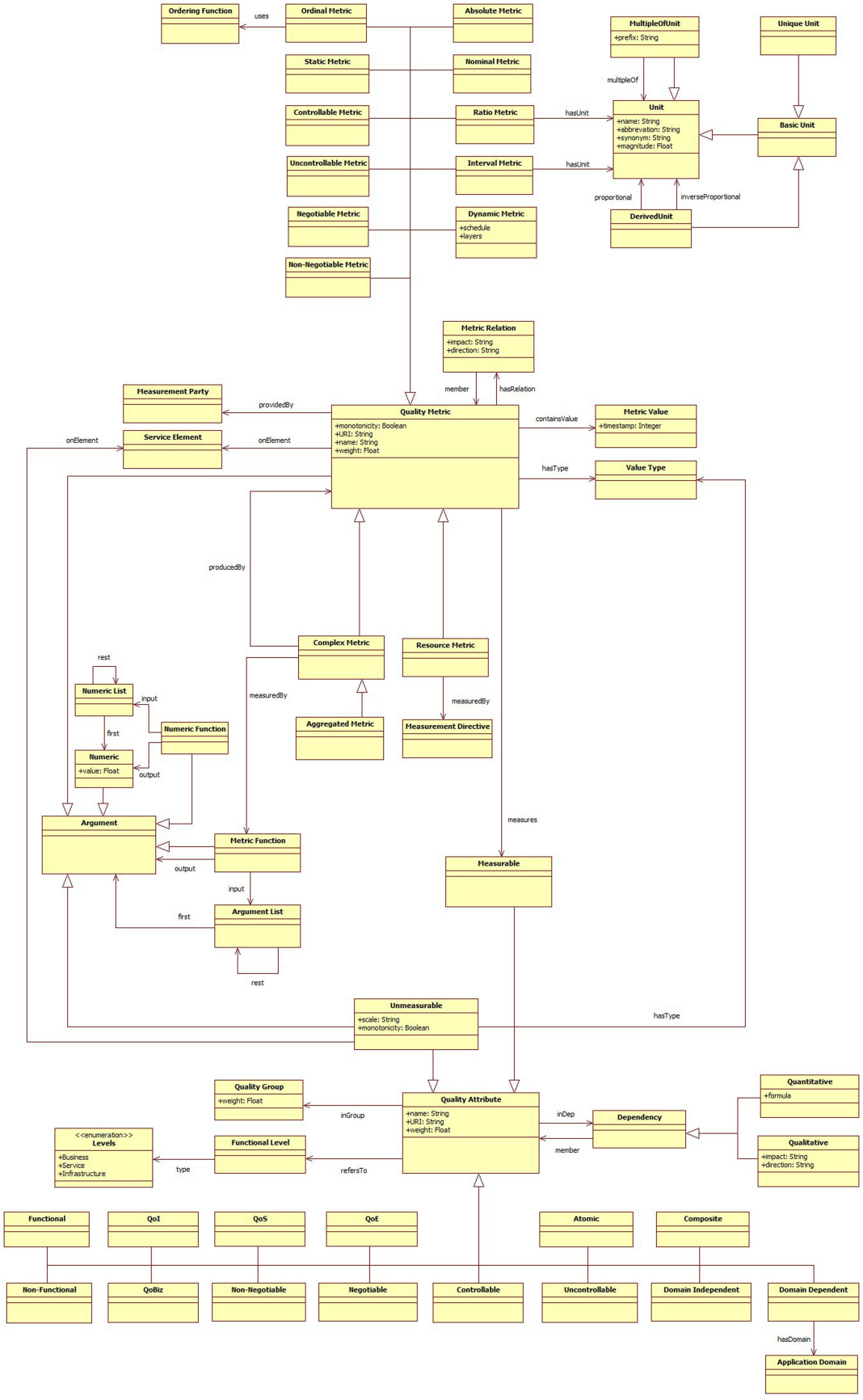



Figure 3.1: Basic Quality Concepts

attribute is measurable, then it can be measured by one or more quality metrics. For instance, *reliability* can be measured by the metrics of *Mean Time Between Failure* (*MTBF*), *Mean Time To Failure* (*MTTF*), and *Mean Time To Transition* (*MTTT*). If a quality attribute is not measurable, then it cannot be measured by any metric. This concept consists of four main attributes, namely *URI*, *name*, *monotonicity*, and *weight*. The first two attributes are used to uniquely identify the metric. The third one signifies the monotonicity (positive or negative) of the quality metric, i.e. if the greater is its value the greater is the quality offered. The fourth attribute represents the significance of the metric with respect to the other metrics used to measure its quality attribute.

Each quality metric has (see *hasType* relation) a specific *Value Type* (i.e., domain of values) and is applied on (see *onElement* relation) a specific *Service Object* (i.e., the service itself, its operations, its input or output data, etc.). For instance, service *response time* can have as value type the integer set $[1, 10]$ and is applied on/concerns the service itself. The measurement values (see *Metric Value* concept) of a metric are provided by a *Measurement Party* and have a specific timestamp attached to them to enable reasoning about them (e.g. statistical derivation).

Quality metrics can participate in *Metric Relations*. These relations are qualitative in nature and are derived from the qualitative dependencies of the corresponding quality attributes (that are measured by the quality metrics of the relations). For example, consider the qualitative dependency between *response time* and *data accuracy*. This dependency can have a medium impact (actually this impact depends on the capabilities of the service's hosting system) and a positive direction. Further, consider that the former attribute is measured by an average metric and the latter with the Mean Absolute Percent Error (MAPE) metric [15]. These two metrics can participate in a metric relation that has a medium impact and a positive direction. Please note that the derived metric relation has a different direction than the dependency of the corresponding attributes. This is due to the use of the MAPE metric which is negatively monotonic and has opposite monotonicity from the one expected for the *data accuracy* attribute.

A quality metric can be *Resource* or *Complex*. Resource metrics (e.g. *service status*) are measured by (see *measuredBy* relation) *Measurement Directives* directly from the instrumentation of the service's measurement system. Complex metrics (e.g. *average availability*) are computed from other metrics (see *producedBy* relation) through the use of *Metric Function*s (see *measuredBy* relation). Metric Functions take as input a list of arguments and produce as output an argument. An *Argument* can be a numeric value, a metric, a measurable quality attribute, a metric function, or a numeric function. A special sub-concept of complex metrics is *Aggregated Metric* used to represent those metrics that compute the aggregated quality of SBAs (i.e., of service compositions).

Similarly to quality attributes, quality metrics can be controllable or uncontrollable and negotiable or non-negotiable. Usually, if a quality attribute is uncontrollable or non-negotiable, then its metrics are also uncontrollable or non-negotiable, respectively. However, the opposite is not true, so there can be cases like the one where a quality attribute is controllable but one of its metrics is uncontrollable because it is computed by a different party than the one that controls the metric's attribute. These cases should be captured by the meta-model.

*Static Metrics* are metrics whose values are computed once and remain almost stable during the service's lifetime. On the other hand, the value of *Dynamic Metrics* is regularly changing based on a specific schedule. Moreover, the latter metrics should be related to the layers of the Service-Based System where monitoring can be performed (see attribute *layers*).

According to the scale of measure they use, quality metrics can be distinguished into nominal, ordinal, interval, ratio, and absolute metrics. *Nominal Metrics* have a value type which contains values that have no specific ordering. *Ordinal Metrics* have value types that have a specific ordering in their values. However, this ordering is not always machine-understandable and processable, especially in the case of string-based value types, and ordering functions should be used to create a specific order between the metric values (e.g. by mapping them to numbers). *Interval Metrics* have specific ordered value types where the interval between any two consecutive values is always constant. *Ratio Metrics* contain ordered value types that preserve not only the intervals but also the ratios between any two consecutive values.

Both interval and ratio metrics are usually associated (through the *hasUnit* relation) with a *Unit* of measure. *Absolute Metrics* represent a special type of metric that takes the form: "num of occurrences of $x$ in the measurement set".

Finally, *Units* have a specific name and abbreviation and a specific set of synonyms as attributes. They are distinguished between *Basic Units* and multiples of units (*MultipleOfUnit*). The latter units are produced from the former by multiplying a constant (*magnitude*). For instance, a value in 'milliseconds' can be produced from a value in 'seconds' by multiplying the latter value with 1000. Basic units are further categorized into unique and derived. *Unique Units* (e.g. 'seconds') do not depend on other units, while *Derived Units* do and can be produced from other units.

## 3.2 Quality Specification Constructs

This section will be dedicated to analyzing the second part of the S-Cube's quality meta-model that concerns the concepts needed for creating and representing service quality specifications and SLAs. This second part builds on the previous part, as quality characteristics and metrics are used to define constraints and service levels in service quality specifications and SLAs, respectively. The second part of the quality meta-model is displayed in Fig. 3.2.

This section is separated into two subsections, where the first one is dedicated to analyzing the concepts related to matchmaking, negotiation, and contracting (e.g., SLAs, penalties, quality offerings and demands), while the second one is dedicated to analyzing the concepts related to quality specification (e.g., constraints, KPIs, and predicates).

### 3.2.1 Matchmaking, Negotiation, and Contracting Information

The entities that actually create the service quality specifications and SLA documents are called *Actors*. These actors can be *Users*, *Companies*, or *Agents* acting on behalf of other actors. Users have a name, title and contact information as attributes, while other information for them (e.g. preferences) can be found on *User Profiles*. All the information about companies can be found in the corresponding *Company Profiles*. All of these actors can participate in service negotiations so they should have (see *strategy* relation) a specific *Negotiation Strategy*. In the special case that an actor is an agent, this actor can enforce part of or the whole negotiation strategy of its enabling actor.

Actors can take different *Roles* (see *takesRole* relation) in different cases. The roles that an actor can take are those of the service *Provider*, *Requester* and *Third Party*. An actor can take many roles even at the same time provided that these roles are not conflicting with each other. For example, a user cannot be concurrently both a provider and requester for the same service. A *Third Party* can be a *Broker* (providing service discovery facilities or mediating negotiations), a *Measurement Provider* or a *Condition Evaluator* (evaluating SLA conditions).

Providers provide a *Service* and have a specific *CostModel* associated to this service (see relations *costModel* and *forService*). A cost model consists of a set of CostFunctions that map a specific quality attribute to a specific cost. The actual cost of the service is computed by adding these cost functions when they are applied on the value of the quality attribute that is promised or delivered by the service. A service can deliver a *Service Object* (e.g. a video or a data file) and both the service and its object have specific *Quality Capabilities*. Quality capabilities are contained in service Quality Offerings which are service *Quality Specifications* with specific time *validity* (this is crucial for highly dynamic environments where quality is constantly changing). Moreover, quality capabilities are mapped into *Service Levels* which are established in service *Contracts* after a successful service negotiation between service providers and requesters.

Apart from service levels, contracts contain other important information like the time *validity*, the *penalties* that will be enforced when the contractors do not correctly fulfill their obligations (e.g. the service of the provider deviates from the required service level), and the exceptional conditions that can

happen during service execution and the corrective actions for these conditions. Especially, the penalties of a contract are both associated to a quality constraint and to the cost of the constraint's violation. A contract's contractors (see *contractor* relation) are the service provider and requester, while there are also other *Third-Party* entities that play an important role like the measurement providers (that provide instantiations of quality metrics), the condition evaluators (that check if a service level condition that includes a quality metric is violated), or the brokers (that perform service discovery or mediate in service negotiations).

A special type of contract is that of a *Composite Contract*. Composite contracts are used to represent contracts between a composite service provider and a requester. The fact that makes them special is that they depend on and consist of other contracts which are agreed between the composite service provider and the service providers of the component services of the composite service.

Requesters seeking for a specific service usually have a specific *budget* which is actually a *Cost* associated to a specific *currency*. These actors also have a specific *Quality Selection Model* and specific *Quality Requirements* expressed in service *Quality Demands*. Both the quality selection model and the quality demand are expressed inside a service *Quality Request* which is issued by the requester to a Broker or matchmaking engine in order to find the best service that fulfils his/her needs. A quality selection model contains a set of <quality attribute, weight> pairs that specify the importance of each quality attribute for the requester and how the best service should be selected. A service quality demand is actually a service quality specification.

### 3.2.2 Quality Specification Information

Both service quality capabilities and requirements are expressed in the same way through the use of *Rules* or *Constraints* (see *expressedBy* relation). The constraints can be simple, complex or *KPI*s. *Simple Constraints* take two *Arguments* that are compared based on a specific *Comparison Predicate*. Moreover, each simple constraint is either hard or soft (see *hard* attribute) and has a specific *weight* that signifies its importance. The latter two attributes can be very useful in the matchmaking and negotiation processes when there is no service quality offering that completely satisfies the service quality demand so some of the constraints of the service quality demand should be relaxed. *Complex Constraints* contain a list of constraints (*Constraints List*) and they apply a specific *Constraint Predicate* on this list. *KPIs* are a special kind of constraints that apply a specific formula to one or more *Process Performance Metrics* (which are quality metrics) in a specific analysis period. The outcome of the KPI's formula application is compared with an upper or lower bound *value* in order to infer the satisfaction of this KPI.

Finally, the last analyzed concept is the one of *Predicates*. Predicates can be n-ary, binary or unary according to the number of arguments (i.e., arity) on which they apply. Constraint predicates can be of any arity, while comparison predicates can be only *Binary Predicates*.

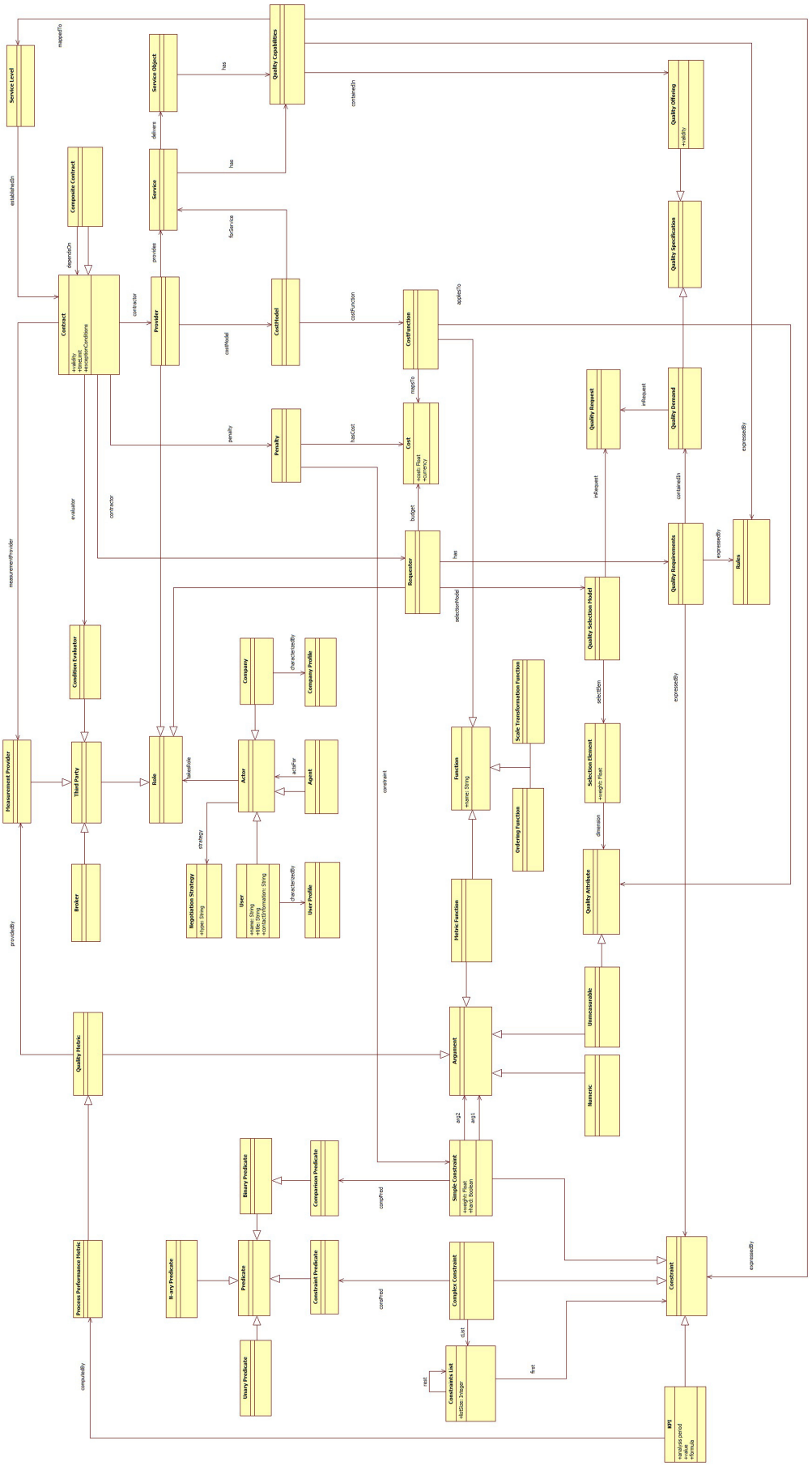



Figure 3.2: Quality Specification Constructs

# Chapter 4

# End-to-End Quality Decomposition through Service Negotiation

In this section, we attempt to solve the difficult problem of decomposing end-to-end quality, as it is defined in a composite service quality specification, into individual service quality of the component services (of the composite service) embodied into service quality specifications and SLAs. This is dictated by the second objective of this deliverable. The approach that is followed is that an instance of this problem is solved by mapping it into a composite service negotiation scenario. Through this scenario, the use of the previously introduced quality meta-model is highlighted, while an incomplete solution to the particular instance of the problem is given. The solution relies on combining the research work already performed by the WP partners on service quality negotiation.

This section is organized into two main sections. The first section explains the problem and which instance of it we attempt to solve, while also highlights the use of the meta-model. The second section provides an extended summary of the research works already performed by the WP members for service negotiation and explains how they can be combined in order to address the instance of the problem we want to solve.

## 4.1 Problem Instantiation and Meta-model Usage

### 4.1.1 Problem Instantiation

The second objective of the deliverable dictates that an approach for decomposing end-to-end quality into quality specifications for individual SLAs has to be developed. In order to better explain and cope with this quality decomposition problem, we take the following view: Supposing that there are end-to-end quality requirements requested by a service requester, a composite service provider has to assure that the end-to-end quality offered by his service and maybe agreed with the service requester is guaranteed. So, actually, the end-to-end quality decomposition problem is an instance of end-to-end quality assurance, as the composite service provider has firstly to decompose the end-to-end quality into individual service quality of the component services and then to ensure that the individual service quality does not deviate in order to assure the end-to-end quality of his composite service. Thus, in the end, the composite service provider has to negotiate with many service providers – which have services that match the component services of the composite service – the individual service levels of the component services in order to assure that the end-to-end quality of his composite service is guaranteed. The latter form of negotiation will be from now on called *Composite Service or Contract Negotiation*, while its successful result will be called *Composite Contract*. In this type of contract, both the agreed composite and individual service levels are defined.

While the problem of decomposing end-to-end quality into quality of component services has been dealt with successfully by many research attempts that solve the well-known Service Concretization or

QoS-aware Service Composition problem (see deliverable PO-JRA-2.2.1), the composite service negotiation problem sketched above has not been completely addressed by the research community [27]. We believe that the latter problem is very important and even more crucial, bigger, and complex than the Service Concretization problem which gives static and sometimes outdated or invalid solutions. In fact, the Service Concretization problem is a sub-problem of the composite service negotiation problem because the former has to be solved in order to find the solution of the latter. To better explain, a Service Concretization problem is solved in the very beginning of the composite service negotiation in order to decompose end-to-end quality and initiate the individual service negotiation. Then if an individual service negotiation fails, then it has to be solved again or re-negotiation at the higher level must take place before it has to be solved again. The overall goal is to produce a composite contract that will drive the execution of the composite service and its constituent services.

As it can be seen from the above analysis, composite service negotiation is a very novel and complex problem. The complexity lies on the fact that many different service negotiations take place at two different levels (for the composite service and its component services) and depend on each other. So the result of one negotiation may have a significant effect on the result of the other negotiations. In order to deal with this problem, new techniques have to be devised or other techniques from other research fields have to be explored (e.g. hierarchical transactions). While the goal of this WP is to propose a solution to this problem, it is difficult to reach a good and complete solution in a very short time. To this end, this deliverable does not propose a complete solution but sketches how four research approaches [1, 2, 3, 4] of the WP partners can be jointly used to solve this problem. The sketched solution is analyzed in the next section.

### 4.1.2 Meta-model Usage

In this subsection, we analyze how the developed quality meta-model can be used to model an example of a composite service negotiation. In this way, we follow a proof-of-concept approach in order to show the effectiveness and adequacy of the quality meta-model. Firstly, we start explaining this example, which is drawn from the *Wine Production* case study (see deliverable CD-IA-2.2.2), and then we show how the meta-model concepts can be used to model all the appropriate information.

#### Example explanation

Based on the *Wine Production* case study (see deliverable CD-IA-2.2.2), we consider that the *Quality Manager* is a user requester $R$ that has to detect critical conditions for one of the vineyards of the *Wine Producer* and to take appropriate actions. This user needs specific information like the temperature of a specific region of the vineyard in a fast and accurate way by a service that could be executed several times each day. Suppose, now, that there is a *Vineyard Forecast* composite service $S$ provided by a service provider (SP) $SP$, which takes as input the region to inspect and the critical temperature and produces as output the sub-regions that have a temperature below the critical temperature and the accuracy of the result. This service consists of three operations/services: *Weather Forecast* $S_1$, *Vineyard Sensor Network* $S_2$, and the *Accuracy Resolver* $S_3$ services. The first two services are executed in parallel, while the third one is executed after the end of the execution of the first two services. The first service takes as input the region and provides the temperature that it predicts with the appropriate accuracy, while the second service takes as input the region and the critical temperature and produces a list of sub-regions that have a temperature lower that the critical temperature and the accuracy of the result. These two services have to be provided externally and there are two SPs $SP_1$ and $SP_2$ that can provided them, respectively. The third service takes a input the output of the second service, filters the sub-region set based on the predicted temperature of the first service (if there are contradictions so a faulty sensor reports wrong information), and produces the accuracy of the final result. This service is implemented internally by the *Vineyard Forecast* SP.

Let us now suppose that all participants express service quality specifications in terms of three quality metrics: a) *average response time* $art$ (expressed in seconds and having as value type the integer set [1,5]), b) *minimum availability* $mav$ (having as value type the real set [0.0,1.0]), and c) *average accuracy* $acc$ (having as value type the real set [0.0,1.0]). Both the $SP$ and $R$ express that the requested service would provide the following capabilities: $art >= 1$ and $art <= 7$ and $mav >= 0.99$ and $acc >= 0.8$. Both $SP_1$ and $SP_2$ express the following capabilities: $art <= 1$ and $art <= 5$ and $mav >= 0.995$ for their services. Moreover, for service $S_1$ we have that $acc >= 0.8$, while for service $S_2$ we have that $acc >= 0.9$. In addition, the $SP$ knows that his internal service $S_3$ has the following capabilities: $art <= 1$ and $art <= 2$ and $mav >= 0.99999$ and $acc >= 0.85$. It is easy to see that if $SP$ chooses to implement his composite service's tasks with the services of $SP_1$ and $SP_2$, then the quality capabilities of his service would be guaranteed. So a composite service negotiation takes place between all the participants that results in a composite contract that depends on two other contracts which are also generated.

**Meta-model Conceptualization of the Example**

The *average response time* metric measures the quality property of *response time*. In our case, this property is an instance of the QoS, Measurable, Negotiable, and Controllable concepts and refers to the Service/SBA functional level. The same goes for the *availability* quality property which is measured by the *minimum availability*. However, *response time* belongs to the *Performance* Quality Group, while *availability* belongs to the *Reliability* quality group. Finally, the *average accuracy* metric measures the quality property of *accuracy*, which belongs to the *Data Quality* quality group and refers to the Service/SBA functional level.

Concerning now the metrics, *average response time* is an instance of Dynamic, Complex, Ratio, Controllable, and Negotiable classes and is applied on the *service* Service Element. The same goes for the *minimum availability* metric. However, *average response time* is measured in *seconds* (instance of Unit class), while *minimum availability* is measured as a *percentage*. Finally, the *average accuracy* metric is instance of Dynamic, Complex, Ratio, Uncontrollable, and Non Negotiable classes and is applied on the *operation result* Service Element.

Now, the *Composite Contract* instance $CC1$ that will be produced will have as contractors the $SP$ and $R$ participants (instances of the class *User*). The former takes the role of the *Provider*, while the latter takes the role of the *Requester*. The $SP$ provides service $S$ that is an instance of the $Service$ class and has a specific *Quality Capability* $QC$ that is mapped to a specific *Service Level* $SL$ inside the $CC1$ contract. $QC$ will be expressed through an instance of a *Complex Constraint*, which will be composed of a list of four*Simple Constraints* applied on the *and* Constraint Predicate. The first two simple constraints of the list will be referring to the *average response time* metric and will have specific comparison predicates and numeric values, while the third simple constraint will refer to the *minimum availability* and will have the $>=$ comparison predicate and the numeric value of $0.99$. The fourth simple constraint will refer to the *average accuracy* and will have the $>=$ comparison predicate and the numeric value of $0.80$.

The other two created contracts will be an instance of the *Contract* class and will have as contractors the $SP$ and the $SP1$ or $SP_2$ users, respectively. In these contracts, $SP$ takes the role of *Requester* while the other users take the role of the *Provider*. The contracts are populated in the similar way concerning the remaining terms as with the composite contract. Moreover, the composite contract will depend on these two contracts (i.e. the *depends on* relationship is used to connect the contract instances).

## 4.2 Initial approaches for negotiating composite SLAs

In this section, a solution to the composite service negotiation problem that combines three research approaches of the WP members is provided. First of all, in the first subsection, the solution is sketched,

while it is shown how one complementary – with respect to the three WP member research approaches – approach [4] can be used to support this solution. Then, in the remaining subsections, the three research approaches of the WP members are analyzed, while a justification of how they can be utilized in the proposed solution is given.

### 4.2.1 Solution

First of all, we suppose that a functional and quality-based discovery process has been executed by a broker and has inferred that there is a match between the service capabilities of the composite service and the service requirements of the service requester. Then, both the composite service provider and requester are engaged in a hierarchical and composite contract negotiation process which is carried out either by the broker or by the composite service provider's system. Our analysis supposes the existence of the broker that carries out the solution process.

The broker gets the quality capabilities and requirements of the two participants in the SBA level along with their negotiation capabilities and examines if there exists a broker-based negotiation protocol that can be enacted to start the negotiation between the two participants. If not, then the composite negotiation fails. Otherwise, negotiation between the participants initiates. Let us assume that this negotiation results in a composite SLA which has to be assured – i.e. there is a step between the final agreement where the end-to-end quality has to be guaranteed. Then, based on the end-to-end quality of this SLA, the structure of the composite service, and the service quality specifications and negotiation capabilities of candidate services stored in the broker's system or retrieved by this entity, a negotiation capability filtering, a Service Concretization (SC) problem solving and negotiation protocol discovery are executed sequentially in order to: a) decompose the end-to-end quality, b) map each component service of the composite service to a particular candidate service, and c) to choose a particular negotiation protocol so as to enact the negotiation between the composite service provider and each component service provider. Let us name this phase as *hierarchical engagement phase*. If there is no solution for the SC problem, then the control goes back to the root negotiation process so as to re-negotiate the SLA between the composite service provider and the requester (i.e., relax some quality constraints of the requester).

Then, an hierarchical negotiation process is executed, whose goal is to establish an SLA for each component service. If one of the component service negotiation fails, then the chosen candidate service is filtered and either negotiation takes place for the next candidate service for the same component service or the broker's execution goes back and re-executes the steps b) and c) of the hierarchical negotiation phase. The whole process stops either when all SLAs are established in an hierarchical manner or when this is not feasible any more.

#### Backbone System for the Proposed Solution

Engagement protocols are used in order to setup a negotiation. After closely looking at the engagement protocol described in the approach of Comuzzi and Spanoudakis [4], which was developed by the CITY WP member organization for another European research project, it is easy to see that this protocol is used to establish a composite monitoring framework for monitoring a Service-Based System (SBS) (i.e. a composite service). A similar engagement protocol could be used to establish a composite negotiation framework or broker that could mediate the negotiation both between the user and the composite service provider and between the composite service provider and the third-party providers. This composite broker would get the negotiation capabilities of all the participants (e.g. after discovering which candidate services can be used for executing a task of the composite service) and then would try to match them (by considering the research work of the first research paper [1]) in order to find out which are the possible negotiation protocols that can be used and which third-party service providers would be ruled out from negotiating with the composite service provider. Then this broker would run as an hierarchical process in order to execute the n+1 negotiations between the participants by executing simultaneously the

two different negotiation protocols analyzed in the research works of the second and third paper [2, 3]. So, actually, this research approach, if extended appropriately, could be the backbone of the composite negotiation system-solution.

### 4.2.2 Paper [1]: Semantic-Aware Service Quality Negotiation

The goal of Web service (WS) discovery is to select WSs that satisfy both the users functional and non functional requirements. Focusing on non functional requirements, a matchmaking algorithm usually takes place to verify if the quality offered by the WS provider overlaps the quality requested by the user. Since quality, in a provider perspective, is costly, a further step, a negotiation, should be performed to identify a mutually agreed quality level.

In this paper, a semantic description model for service discovery and negotiation is proposed. More specifically, OWL-Q, an extension of OWL-S for a rich, semantic, and extensible QoS-based service description is extended in order to describe the negotiation capabilities of the participants. OWL-Q is extended with a new set of concepts and properties, which are grouped as *Quality-related* and *Negotiation-specific* extensions. The former extend the ontology in order to accommodate the Negotiation Object, i.e., the description of what can be negotiated. The latter introduce new concepts, such as the negotiator actor or service consumers' negotiation strategies, which are then used to define the negotiators' decision model.

For what concerns the negotiation protocol, the extension of OWL-Q that describes various negotiation protocols, such as trading and tendering, is currently under development. In this paper, the research approach is limited to QoS configuration algorithms for which the the negotiators' strategies are parameterized. The WS QoS configuration protocols proposed in the paper can be implemented by a broker-based architecture for QoS negotiation.

Quality-related extensions are introduced to cope with two fundamental issues raised by the need to support QoS negotiation through an ontology:

- Not all QoS dimensions, either technical or domain dependent, are negotiable. A QoS dimension is *Negotiable* when its value can be set by the service provider at runtime, i.e., when the service is invoked. *Non-negotiable* QoS dimensions are the ones for which the value cannot be set at runtime by the service provider. For instance, when a service is invoked, its reputation is fixed, regardless of the technique adopted for assessing reputation. Moreover, the service provider can always decide whether to allow or not the negotiation of a specific quality dimension. In our SMS service running example, *response time* can be considered as negotiable, since the response time value can be altered by the provider at execution time according to the customer requirements. However, in case the provider's provisioning infrastructure does not allow such an adaptation, the developed ontology allows the provider to declare the response time as non negotiable;

- In order to automatically negotiate the QoS of a service, we also need to define a total ordering relation among admissible values identified for each QoS dimension. This ordering is established by the communities of domain experts that define the quality documents associated to a category of WSs.

Negotiation-specific extensions concern the concepts and properties, besides QoS definition, required to establish a negotiation framework. In particular, the following concepts are identified:

- **Negotiator Actor**. Usually, the participants involved in WS QoS negotiation are the service provider and the service consumer. However, a more flexible approach should consider that the execution of a negotiation may be delegated to a trusted third party, such as, for instance, an ad-hoc agent explicitly designed to negotiate on behalf of the service provider or the service customer. Introducing the negotiation actor also enables our framework to accommodate other negotiation

protocols, such as, for instance, single-text mediated negotiations, which require the existence of a third party trusted by all the negotiation participants;

- **Negotiation Strategy** As previously introduced, our semantic-aware negotiation framework relies on the assumption of having parameterized negotiation strategies, i.e., negotiation strategies that are fully determined when a negotiator actor specifies the values of a set of parameters, such as the initial offer and the degree of concession over time to the counterpart.
- **Cost model** From the provider's point of view, the negotiation often relies on a cost model, i.e., parameterized functions that are used to evaluate the cost sustained by the service provider for giving a certain level of quality in his or her service offers. Usually, WS QoS cost models are additive, that is, the cost of a service offer is given by the costs associated by the cost model to each single QoS value that appears in the offer.

Apart from proposing a specific ontological description model for service discovery and negotiation, the paper motivates and highlights the power of ontologies when they are used in conjunction with rules in order to infer new knowledge such as equivalence of quality attributes, matchmaking of a QoS offer with a demand, producing the price of a specific QoS level for a provider, etc. Generally, the whole discovery and negotiation algorithms may be written in the form of a modular set of rules so that only specific functions need to be actually implemented in places where mathematical tools are required.

**Research Approach Usage**

The work analyzed in this paper can be used for the semantic description and matchmaking of quality-based specifications, for the semantic description and matchmaking of the negotiation capabilities of the participants, and for the production of useful facts that can assist the actual negotiation process. So this work can be used in conjunction with an actual negotiation protocol, feeding it with valuable information before it is enacted. With the appropriate extensions, this work can be used for discovering which is the suitable negotiation protocol between a set of participants by automatically inspecting their negotiation capabilities and inferring the required knowledge.

### 4.2.3 Paper [2]: A Framework for QoS-Based Web service Contracting

This paper proposes a framework for the automation of the Web service contract specification and establishment. An extensible model for defining both domain dependent and domain independent Web service QoS dimensions and a method for the automation of the contract establishment phase are proposed. A matchmaking algorithm is proposed for the ranking of functionally equivalent services, which orders services on the basis of their ability to fulfill the service requestor requirements, while maintaining the price below a specified budget. An algorithm for the configuration of the negotiable part of the QoS Service Level Agreement (SLA) is illustrated, which is used to configure the agreement with the top-ranked service identified in the matchmaking phase.

A contract is a formal agreement between two or more parties to create mutual business relations or legal obligations. In electronic settings, contracts are composed of different parts, such as the definition of business partners, the specification of functional obligations, and quality, price, and penalties related to the object of the agreement. In a dynamic business scenario, the contract definition and establishment should be automated as much as possible, in order to allow organizations to dynamically modify business partners or reconfigure the SLAs on quality aspects negotiated with service providers. In the context of Web services, contracts on QoS aspects can also be used for process monitoring and for providing advanced capabilities, such as self-healing behavior with respect to functional or quality related failures during service provisioning.

Three activities define the lifecycle of an electronic contract, that is, (i) *contract definition*, (ii) *contract establishment*, and (iii) *contract enactment*. The contract definition activity concerns the establishment of a model for the definition of contract terms, which is understood and shared by the contracting parties. This is usually achieved through the definition of a contract template, which is then instantiated in an actual contract that reflects the domain dependent interests of providers and customers. The contract establishment activity concerns the instantiation of the contract template in an actual contract. It may involve the selection of the contract partner, among a set of potential partners, on the basis of functional requirements, the matchmaking of offered and required QoS, and the negotiation of the contract terms between the selected partner and the service customer. Finally, the contract enactment concerns the execution of the contract. In the specific context of contracts on QoS aspects, this activity coincides with the monitoring of the satisfaction of QoS guarantees negotiated by a service provider and the service requestor.

This paper contributes to the research on Web service QoS contracting in two ways. First, it proposes an extensible QoS model for Web services that is suitable to manage different QoS dimensions, either negotiable, non-negotiable, domain dependent, or domain independent, defined by heterogeneous metrics and measurement methods. The model entails the description of admissible values for a QoS dimension as a collection of totally ordered values or ranges, which can be translated into adimensional *levels* to reconcile heterogenous metrics and measurement methods. Second, it proposes mechanisms for service matchmaking, selection, and offer configuration able to exploit the aforementioned QoS description. In particular, the matchmaking algorithm extends current practice, which usually requires the provider offer to fully cover the service requestor's requirements, by considering also the service offers that only partially cover the requirements. Such a scenario becomes possible when matchmaking is coupled with a subsequent offer configuration phase, which guarantees that the final agreement between a service provider and the requestor belongs to the overlap of offered and required QoS. In between the matchmaking and SLA configuration phases, we also propose a service selection mechanism, which applies to the specific context of QoS-Based Web service contracting some basic results on multi-attribute auction theory developed in the field of microeconomics. The modeling concepts and the contract lifecycle approach described in the paper have been developed within the S-Cube activities towards developing concepts for specifying end-to-end quality characteristics and negotiating SLAs.

**Research Approach Usage**

While the first research approach can be used for modeling the negotiation capabilities of the negotiation parties and for assisting the service negotiation process by automatically providing useful facts like which services have quality capabilities that match the quality requirements of the requester, this approach is complementary as it proposes a framework for the automation of the Web service contract negotiation and establishment. So this approach can be utilized after the first approach execution (so some useful information is already derived for it) for performing the actual negotiation between the participants for a particular service, either this service is composite or not, and producing the corresponding SLA.

### 4.2.4 Paper [3]: A Dynamic Privacy Model for Web Services

Nowadays, Web services are being recognized as an emerging platform to quickly develop complex distributed applications. Many services (e.g., mortgage approval, travel agency) require service requestors to disclose some personal data (e.g., credit card number, home address). As the number of inappropriate usage and leakage of personal data is increasing, privacy concerns is becoming one of most important concerns of service requesters, service providers and legislators. In order to take into account the privacy concerns of the individuals, organizations (e.g Web services) provide privacy policies as promises describing how they will handle personal data of the individual. However, privacy policies do not convince potential individuals to disclose their personal data, do not guarantee the protection of personal information, and do not provide how to handle the dynamic environment of the policies.

In this paper, a framework is introduced based on an agreement as a solution to these problems. A privacy agreement model is proposed that spells out a set of requirements related to consumers privacy rights in terms of how service provider must handle privacy information. Two levels are defined in the agreement: (1) policy level (2) negotiation level. A formal privacy model is described in the policy level to provide upon it a reasoning mechanism for the evolution. The framework supports in the negotiation level of the agreement a lifecycle management which is an important deal of a dynamic environment that characterizes Web services. Hence, the privacy evolution is handled in this level. A negotiation protocol is proposed to enable ongoing privacy negotiation to be translated into a new privacy agreement.

**Research Approach Usage**

Privacy is considered an orthogonal quality attribute with respect to all other quality attributes and is dealt with separately in quality-based matchmaking and negotiation systems. To this end, the approach proposed by this paper can be used in order to enforce in parallel to the negotiation of the other qualities the negotiation of privacy information. So, actually, two different types of negotiation can take place in the system between the negotiation participants both at the level of a single service and at the level of the composite service.

# Chapter 5

# Discussion and Conclusions

In this deliverable, a rich and extensible quality meta-model has been proposed that can be used for precisely defining service quality models (like the S-Cube Quality Reference Model (QRM)), service quality specifications and SLAs. Moreover, this meta-model captures information that can be used in service negotiation like negotiation capabilities, strategies, and cost models. Its development has followed a design approach. Firstly, a set of requirements was gathered and then prioritized, filtered or extended, and structured into a coherent ordered and structured set. Secondly, based on these requirements set and the comments of the WP partners, the quality meta-model was designed and developed. The meta-model was expressed using both the OWL ontology language [1] and the UML modeling language [2]. In Section 5.1, we will elaborate on the usefulness of these two representations of the meta-model considering the role that quality plays in the service life-cycle.

The second objective of this deliverable was to propose a methodology for decomposing end-to-end quality into quality specifications for individual SLAs. To this end, it was first demonstrated that *composite service negotiation* can be used to achieve this objective and produce a composite contract that encompasses the quality specifications for individual SLAs. Then, a composite negotiation example, extending the Wine Production case study, was given, whose result, i.e. the composite contract, was specified based on the service quality meta-model. In this way, the meta-model's effectiveness and sufficiency was highlighted. Finally, a partial solution to the *composite service negotiation* problem was sketched based on the research work of the WP partners.

While composite service negotiation achieves to construct an SBA, it is not enough for ensuring that this SBA functions properly and assuring that its quality conforms to the service levels embodied in the composite contract. Section 5.2 shortly elaborates on this issue and explains how the next WP deliverable aims to address it.

## 5.1 Candidate Formalisms for the Meta-Model

The quality meta-model can be used in service quality definition, matchmaking, and negotiation as it captures appropriate and essential information for these three processes. However, the implementation formalism of this meta-model has a significant impact on how these three process can exploit the information captured by this meta-model to achieve their goals.

The quality meta-model has been represented using both the OWL and UML formalisms. Both of these formalisms cater for a formal quality description model. However, their expressiveness is different and partially complementary, while their tool and technique support is quite different. We are not going to describe and compare the expressivity of these formalisms. The prospective reader can refer to [28, 29] for a complete comparison. However, we are going to show how supporting tools and techniques can be

[1] Meta-model is available at the URL: `http://www.csd.uoc.gr/~kritikos/Quality_meta-model.owl`
[2] Meta-model is available at the URL: `http://www.csd.uoc.gr/~kritikos/Quality_meta-model.zip`

used to support or enable or implement the above three processes.

### 5.1.1 Quality Definition, Validation, and Alignment

Both OWL and UML can be used to define a quality meta-model. With this meta-model, quality specifications and models can be defined. However, after defining a quality model, the problem is if this model is valid and correct. OWL is accompanied by reasoning techniques that can infer if a specific ontological specification is consistent with its description model. However, this consistency partially solves our problem because it cannot guarantee that the constraint description is correct (i.e., that there is a solution allowed by the constraint set so that all quality metrics or unmeasurable attributes take a specific value from their value type). One solution could be that after ontology consistency, a translation of the constraint description into a constraint model is performed and then this model is checked for consistency with the help of a linear or constraint programming engine.

The ambiguity inherent in UML coupled with its support of multiple viewpoint modeling pose a great risk of inconsistency. Many classifications of UML inconsistencies exist in the literature today. Several proposals are also made for the mitigation of this risk, which do not provide a complete solution to this problem. One solution adopted by many approaches is to check the consistency of a UML model only after translating them into a more formal notation that naturally supports this kind of analysis. INRIA has already performed work towards this solution for service quality specifications. In the next paragraph, we sketch their solution, which will be implemented in the near future.

The transformation of the specification of a quality modeling language specification or a quality meta-model (QMM) to a formal methods language, Alloy, is proposed. The formal specification of the QMM in the form of an Alloy model is viable to different kinds of analysis. First, hundreds and thousands of models in the domain specified by the QMM can be easily generated. The generated models can be analyzed using an *oracle* which can be a program, a set of constraints or a human inspector. The analysis results in the identification of undesirable models that were allowed by the QMM. The QMM can be iteratively improved by adding expert-made Alloy facts that prevent the generation of undesirable models. The Alloy facts are generalizations to prevent the existence of some properties of the undesirable models. The generation of models itself can be guided by strategies that cover the entire QMM. The use of meta-model coverage criteria to guide the generation of hundreds of different models that instantiate different aspects of the QMM in different quality models is proposed. Second, the Alloy model of the QMM can be checked based on assertions. If a certain assertion must always be true for the QMM, a check is performed in a finite scope in an attempt to obtain *counter-examples*. A counter-example indicates that the QMM still does not conform to the desired assertion and the counter-example indicates that the language still allows the creation of erroneous models. We can therefore add new constraints as Alloy facts to the QMM specification to prevent the generation of these counter-examples.

Except from validating the models produced by a quality meta-model, another quality definition problem concerns the fact that two quality objects (i.e., instance of a concept) can be identical. For example, a service provider and requester may produce quality specifications that define the same quality metric differently. OWL and its supporting reasoning techniques can solve this problem by using alignment rules that dictate when two different objects are the same. For instance, QoS metric matchmaking rules are proposed by Kritikos and Plexousakis [30]. In this way, service quality specifications can be aligned with each other by exploiting alignment algorithms [15]. On the contrary, UML cannot solve this problem, which causes accuracy problems in the quality-based matchmaking algorithms and processes [15].

### 5.1.2 Quality-based Service Matchmaking

After the production of service quality offerings and demands (based on the quality meta-model), a matchmaking algorithm/process must be executed in order to find which offerings match with a demand requested by a user. Basically, there have been two different types of approaches used to solve this

quality-based service matchmaking problem with OWL. The first approach [11] complements the OWL-S functional description language with an OWL-based quality language used to describe a service quality profile and then proposes a matchmaking metric for these quality profiles. Unfortunately, this approach works only for integer-valued quality metrics and attributes as it uses in a erroneous way the OWL feature of cardinality constraints. The second approach [31] recognizes the fact that OWL cannot be used for reasoning about quality constraints so it proposes a hybrid solution: a) OWL is used to describe quality specifications and rules are used to matchmake quality concepts and align these specifications; b) the constraint part of these specifications is transformed to a specific constraint model and matchmaking a quality offer and demand is performed by creating a joined constraint model which is then solved. The latter approach is quite promising and could be used for exploiting the quality meta-model in quality-based service matchmaking.

On the contrary, UML cannot be used for solving this problem because first of all it cannot be used for matchmaking quality objects. Based on the UML-based solution of the consistency problem, a similar-principles solution is sketched in the following. In particular, the transformation of a service's $A$ quality model $Q_A$ to a model $M_A$ in the formal language Alloy is proposed. The client's query quality model $Q_q$ is transformed to a set of assertions $S_q$ in Alloy. Each assertion $a$ in $S_q$ is checked for a finite scope against the Alloy model $M_A$. If a counter example is discovered for a scope that is large enough or within certain time bounds, the quality model of service $A$ is declared as not suitable for the requirements of the client. This helps to filter out those services that do not match quality requirements of a client. If a counter example is not discovered in a large finite scope, it can be concluded that the service satisfies all requirements in the form of assertions of the query quality model $Q_q$. This approach is not limited to the use of Alloy but can be extended to other formal methods tools such as PROMELA, SPIN, and CLP. However, it is incomplete.

### 5.1.3 Service Negotiation

OWL along with rules can be used for assisting the service negotiation process [1]. The OWL querying mechanisms can be used for getting particular information from a quality model, while OWL reasoning mechanisms can be used for inferring new knowledge. On the contrary, UML does not have querying mechanisms, while it cannot also be used for inferring new knowledge automatically unless its specifications are transformed to different formalisms like Description Logics (DLs) or logic programs.

## 5.2 End-to-End Quality Assurance

While composite service negotiation, which is partially dealt with in this deliverable, achieves to construct an SBA, it is not enough for ensuring that this SBA functions properly and assuring that its quality conforms to the service levels embodied in the established composite contract. For ensuring that the SBA functions as expected, we must assure that the SBA is functionally *correct* and satisfies its quality requirements. This means that we must define a set of functional and non-functional properties of interest and to have some techniques able, in a possibly automatic way, to decide if the composite service as well as the services in the composition satisfy these properties. For instance, formal methods, most of them with a semantics based on transition systems (e.g. automata, Petri nets, process algebras), have been used to guarantee correct service compositions [32].

On the other hand, the contract enactment activity concerns executing the contract and assuring its satisfaction. In the specific context of contracts on quality aspects, this activity coincides with assuring the satisfaction of (end-to-end and individual) quality guarantees negotiated between a service provider and the service requestor (or third-party service providers). Novel techniques for the run-time and proactive assurance of quality, such as the (formal) analysis of service specifications during run-time, or the prediction of quality attributes can be used to support the contract enactment activity. This is considered

both from the service provider point of view (e.g., run-time quality assurance) as well as from the service requestor point of view (e.g., monitoring on whether the agreed quality is delivered).

The forthcoming deliverable CD-JRA-1.3.4 “Initial set of principles, techniques and methodologies for assuring end-to-end quality and monitoring SLAs” aims to address the above two problems by devising an initial set of principles, techniques, and methodologies for assuring end-to-end quality and for monitoring composite contracts and their constituent parts.

# Bibliography

[1] Marco Comuzzi, Kyriakos Kritikos, and Pierluigi Plebani. Semantic-aware service quality negotiation. In *Proc. ServiceWave. LNCS 5377*, pages 312–323, Madrid, Spain, December 2008.

[2] Marco Comuzzi and Barbara Pernici. A framework for qos-based web service contracting. *ACM Transactions on web*, 2009.

[3] Salima Benbernou and Hassina Meziane. A dynamic privacy model for web services. *Computer Standards and Interfaces*, 2009. submitted.

[4] Marco Comuzzi and George Spanoudakis. A framework for hierarchical and recursive monitoring of service based systems. In *4th International Conference on Internet and Web Applications and Services (ICIW 09)*, Venice, Italy, May 2009. IEEE Computer Society.

[5] Kyriakos Kritikos and Dimitris Plexousakis. Requirements for qos-based web service description and discovery. *Compsac 2007: 31st Annual International Conference on Computer Software and Applications*, 02:467–472, 2007.

[6] Yutu Liu, Anne H. H. Ngu, and Liangzhao Zeng. Qos computation and policing in dynamic web service selection. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *WWW (Alternate Track Papers & Posters)*, pages 66–73, New York, NY, USA, 2004. ACM.

[7] Svend Frølund and Jari Koistinen. Quality of services specification in distributed object systems design. *COOTS'98: Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, 5(4):179–202, 1998.

[8] V. Tosic, W. Ma, B. Pagurek, and B. Esfandiari. On the dynamic manipulation of classes of service for xml web services. Research Report SCE-03-15, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 2003.

[9] M. Tian, A. Gramm, M. Nabulsi, H. Ritter, J. Schiller, and T. Voigt. Qos integration in web services. Gesellschaft fur Informatik DWS 2003, Doktorandenworkshop Technologien und Anwendungen von XML, October 2003.

[10] Vladimir Tosic, Bernard Pagurek, and Kruti Patel. Wsol - a language for the formal specification of classes of service for web services. In Liang-Jie Zhang, editor, *ICWS*, pages 375–381, Las Vegas, Nevada, USA, 2003. CSREA Press.

[11] Chen Zhou, Liang-Tien Chia, and Bu-Sung Lee. Daml-qos ontology for web services. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 472–479, San Diego, CA, USA, 2004. IEEE Computer Society.

[12] Frank Leymann. Web services flow language (wsfl 1.0). Technical report, IBM Corporation, May 2001 2001.

[13] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[14] Antonio Ruiz Cortés, Octavio Martín-Díaz, Amador Durán Toro, and Miguel Toro. Improving the automatic procurement of web services using constraint programming. *Int. J. Cooperative Inf. Syst.*, 14(4):439–468, 2005.

[15] Kyriakos Kritikos. *QoS-based Web Service Description and Discovery*. PhD thesis, Heraklion, Greece, December 2008.

[16] E. Michael Maximilien and Munindar P. Singh. Conceptual model of web service reputation. *SIGMOD Rec.*, 31(4):36–41, 2002.

[17] Viktor Yarmolenko and Rizos Sakellariou. Towards increased expressiveness in service level agreements: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(14):1975–1990, 2007.

[18] Norman E. Fenton. *Software Metrics: A Rigorous and Practical Approach*. International Thomson Computer Press, Boston, MA, USA, 1996.

[19] E. Michael Maximilien and Munindar P. Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 8(5):84–93, 2004.

[20] Vladimir Tosic, Babak Esfandiari, Bernard Pagurek, and Kruti Patel. On requirements for ontologies in management of web services. In *CAiSE '02/ WES '02: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, pages 237–247, Toronto, Ontario, Canada, 2002. Springer-Verlag.

[21] Marco Comuzzi and George Spanoudakis. Describing and verifying monitoring capabilities for service based systems. In *Proceedings of the CAiSE 2009 Forum*, Amsterdam, Netherlands, June 2009.

[22] WS-AGREEMENT. WS-Agreement Framework. `https://forge.gridforum.org/projects/graap-wg`, September 2003.

[23] A.S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and U. Yalçinalp. Web services policy 1.5 - framework. W3C Recommendation, September 2007. `http://www.w3.org/TR/ws-policy/`.

[24] M. Aiello, G. Frankova, and D. Malfatti. What's in an Agreement? A Formal Analysis and an extension of WS-Agreement. Technical Report DIT-05-039, Informatica e Telecomunicazioni, University of Trento, 2005. `http://eprints.biblio.unitn.it/archive/00000776/`.

[25] I. Brandic, S. Pllana, and S. Benkner. Specification, Planning, and Execution of QoS-aware Grid Workflows within the Amadeus Environment. *Concurrency and Computation: Practice and Experience*, 20:331–345, March 2008.

[26] Aad Van Moorsel. Metrics for the internet age: Quality of experience and quality of business. Technical Report HPL-2001-179, HP Labs, August 2001.

[27] Shiyang Ye and Jun Wei. History heuristic based negotiation of service level agreements for composite service. In *QSIC '08: Proceedings of the Eighth International Conference on Quality Software*, pages 313–320, Oxford, UK, 2008. Computer Society.

[28] L. Hart, P. Emery, B. Colomb, K. Raymond, S. Taraporewalla, D. Chang, Y. Ye, E. Kendall, and M. Dutra. Owl Full and Uml 2.0 Compared. OMG TFC Report, March 2004.

[29] Colin Atkinson and Kilian Kiko. A Detailed Comparison of Uml and Owl. Technical Report TR-2008-004, Department for Mathematics and Computer Science, University of Mannheim, 2008.

[30] Kyriakos Kritikos and Dimitris Plexousakis. Semantic qos metric matching. In *ECOWS '06: Proceedings of the European Conference on Web Services*, pages 265–274, Zurich, Switzerland, 2006. IEEE Computer Society.

[31] Kyriakos Kritikos and Dimitris Plexousakis. Semantic qos-based web service discovery algorithms. In *ECOWS '07: Proceedings of the Fifth European Conference on Web Services*, pages 181–190, Halle, Germany, 2007. IEEE Computer Society.

[32] M.H. ter Beek, A. Bucchiarone, and S. Gnesi. Formal Methods for Service Composition. *Annals of Mathematics, Computing and Teleinformatics*, 1(5):1 – 10, 2007.