A Survey on Service Quality Description

KYRIAKOS KRITIKOS
Politecnico di Milano
and
SALIMA BENBERNOU
Paris Descartes University
and
IVONA BRANDIC
Vienna University of Technology
and
CINZIA CAPPIELLO
Politecnico di Milano
and
MANUEL CARRO
U. Politécnica de Madrid
and
MARCO COMUZZI
Eindhoven University of Technology
and
ATTILA KERTÉSZ
MTA SZTAKI
and
MICHAEL PARKIN
Tilburg University
and
BARBARA PERNICI and PIERLUIGI PLEBANI
Politecnico di Milano

Quality of service (QoS) can be a critical element for achieving the business goals of a service provider, for the acceptance of a service by the user, or for guaranteeing service characteristics in a composition of services, where a service is defined as software and software-support (i.e., infrastructural) services which are available on any type of network or electronic channel.

The goal of this paper is to compare the approaches to QoS description nowadays presented in the literature, where several models and meta-models are included. Our survey is done by inspecting the characteristics of the available approaches, to reveal which are the consolidated ones and to discuss which are the ones specific to given aspects, and to analyze where the need for further research and investigation is. The approaches here illustrated have been selected based on a systematic review of conference proceedings and journals spanning various research areas in Computer Science and Engineering including: Distributed, Information, and Telecommunication Systems, Networks and Security, and Service-Oriented and Grid Computing.

Categories and Subject Descriptors: []:

General Terms: Documentation, Languages, Management, Performance

Additional Key Words and Phrases: service, description, provisioning, life-cycle, quality, QoS, model, meta-model, service level agreement, SLA

---

1. INTRODUCTION

Quality of Service (QoS) can be used for selecting among functionally-equivalent services and for controlling the various activities of service provisioning. A prerequisite of the usage of service quality is its proper, precise, and rigorous description, covering all possible phases of a service life cycle. Many service quality description approaches have been proposed in the literature. The goal of this paper is to systematically and comparatively review these approaches. The approaches are evaluated according to various criteria, which include their scope, formality, expressiveness, and applicability. Many important findings are uncovered from the analysis. In addition, areas for further research and investigation are spotted.

Before entering in the detail of the paper, it is worth precisely defining the area of interest. As stated in [O'Sullivan et al. 2002], a service is an action that is performed by an entity (the provider) on behalf of another one (the requester). Through the interaction between these two entities that involves various phases under the name of *service provisioning*, there is a transfer of value from the provider to the requester or recipient. Another interesting characteristic of services is that they can be composed of other services (e.g. a transportation service may be composed of land and air transport services). There can be various types of services depending on their nature and the means or channels in which they are available. For instance, the drawing of a bank cheque is a physical service which is available only on the bank counter so it requires the physical presence of the requester on a specific place and time in order to be invoked and delivered. As another example, a web service is a software program that is available over the Internet. The focus of this paper is on software and software-support (i.e., infrastructural) services which are available on any type of network or electronic channel. Thus, in the remainder of this survey the word "service" will have this designated meaning and any other type of services will be excluded from the analysis and discussion.

Service orientation has emerged in the last years as a paradigm that facilitates interaction between interoperating systems, but also as a general framework to

enable access to IT-based applications, since the benefits of adopting it include interoperability, just-in-time integration, easy and fast deployment, efficient application development, and strong encapsulation [Allen 2006; Georgakopoulos and Papazoglou 2008]. While in the past such interactions were stable and consolidated, several new application environments are now based on access to services with a much shorter time frame and cost, thus responding effectively to ever-changing market conditions, rapid technology improvements, and increased competition and customer needs. For instance, in the e-business area, services can be selected dynamically and composed in added-value new services, where the components of the composite services are selected from a number of candidate services offering the appropriate functionality. In many pervasive applications, access to services is based on context characteristics, such as location, environmental parameters, and the like. Utility services, such as the ones of telecom and energy providers, also require interoperability of very complex systems to guarantee that the service is delivered. Multimedia and multichannel applications need the composition and synchronization of several different services to provide a good user experience. In general, to provide such services, several phases are needed, from the selection of the adequate services to controlling the characteristics of service provisioning, in particular when the providers are not under the direct control of the service user. This composition of autonomously running services requires that the service usage rules specified in agreements are clearly stated and their compliance is verified.

Services can be offered and used across many functional levels following various IT architectures. The functional architecture model under our consideration, which we name as Service Functional Architecture Model (SFAM), is a simplified version of the one proposed by Skene, Lamanna, and Emmerich [Lamanna et al. 2003]. SFAM follows a three-layer architecture, as can be seen from Figure 1, where each layer may contain various functional levels. However, in order to be generic and capture any type of distributed system, a coarse-grained approach has been adopted, such that there is only one level at each layer. In the first layer, named *Application Layer*, business or user-oriented applications exist. These applications may use one or more services in order to fulfill a part or their whole functionality. In the second layer, named *Service Layer*, services with an electronic interface, such as Web Services or J2EE or .NET components, exist and are used to build or populate the applications. These services are hosted by various infrastructures that belong to the *Infrastructure Layer*. These infrastructures are responsible for managing the services underlying resources for communication, transactions, security and so forth, e.g. through a platform virtualization environment. It should be noted that every component of this architecture can be offered as a service to the same or other types of components. For instance, Infrastructure as a Service (IaaS) is the delivery of computing infrastructures as a service which fulfils the needs of hosted applications or services [Dikaiakos et al. 2009].

In addition to functional characteristics, services are characterized by a set of properties that are encompassed under the general term of QoS. While there exist various definitions of service QoS, we chose to modify the one proposed in [Kritikos and Plexousakis 2009]. According to this definition, "QoS is a set of non-functional attributes of the entities used in the path from the Web Service (WS) to the client

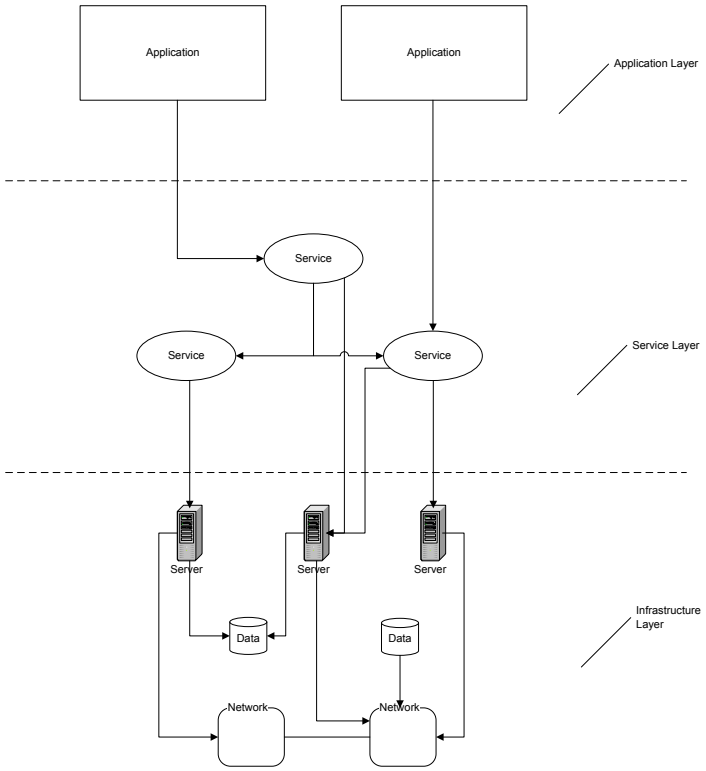

Fig. 1. Service Functional Architecture Model.

that bear on the WSs ability to satisfy stated or implied needs in an end-to-end fashion". This definition takes into account the usually neglected aspect that QoS must be seen in an end-to-end sense as it affects the end-to-end quality received by a WS client when requesting or invoking a particular WS. In this way, QoS characterizes any entity used in the path from the user to the WS, including the service infrastructure and its components. Thus, the QoS characteristics of all these entities constitute the QoS of a WS.

Obviously, the above definition is specific for software services and it does not apply to infrastructural ones. The reason is that the QoS of an infrastructural service depends on the QoS of its components and of the network that may connect them but it does not depend on the QoS of the network that connects it with its client. The problematic part of the definition is the requirement to include the QoS of the entities in the end-to-end path between the service and the client. In order to cater for both types of services, we should consider the QoS of all those contextual entities that are considered relevant to the interaction between the service and its client. In this way, we consider the QoS of the network that connects the service with its client for software services and we do not consider it for infrastructural ones. Thus, the above definition is modified as follows: "QoS is a set of non-functional attributes of those contextual entities that are considered relevant to the interaction

between the service and its client, including the service and the client, that bear on the service's ability to satisfy stated or implied needs".

The distinguishing feature of QoS with respect to functionality is its dynamicity. In particular, the values of some of the QoS attributes can vary without impacting the core function of the service which remains constant most of the time during the service's lifetime. Based on this reason, QoS can be used during service discovery in order to distinguish between many functionally-equivalent services. Moreover, QoS can be monitored and controlled during service provisioning in order to cater for the increasingly high user expectations with respect to the service's performance and other types of QoS attributes. In fact, QoS can play a significant role during several phases of the service life-cycle [Kritikos and Plexousakis 2009].

In general, QoS can be a critical element for achieving the business goals of a service provider, for the acceptance of a service by the user, or for guaranteeing service characteristics in a composition of services. First of all, the QoS description is used to select the best service among a set of functionally equivalent ones. Besides, quality is used to define a contract, i.e., a Service Level Agreement (SLA), between a service provider and a service user in order to guarantee that their expectations are met. In addition, such a contract feeds the service management system that is in charge of assessing the proper quality level during the service execution and enforcing it by taking appropriate recovery/adaptation actions, such as increasing the underlying service resources, substituting the faulty service, or recomposing it.

In all the previous cases, even if their area of application is very diverse, the need of being able to describe QoS in a precise, rigorous way has emerged, covering all possible phases of a service life cycle. In this paper, we will refer to the description of the QoS of a given service with the term *quality document*. We will use this term in a generic way, as a description of quality, and we will not address all issues related to managing such a document in a specific system architecture. Even not referring to specific architectural solutions, we will see that the problem of being able to write such quality documents is far from solved. In the literature, several approaches have been proposed, and there is no commonly agreed way of specifying QoS.

The goal of this paper is to compare the approaches to QoS description, by inspecting their characteristics in order to reveal which are the consolidated ones and to discuss which are the ones specific to given aspects, and to analyze where the need for further research and investigation is. The presented approaches have been selected based on a systematic review of conference proceedings and journals spanning various research areas in Computer Science and Engineering including: Distributed, Information, and Telecommunication Systems, Networks and Security, and Service-Oriented and Grid Computing.

This paper is structured as follows. First, in Section 2, we discuss in detail the service life cycle and we present a general classification of quality models which is then used in the remaining sections of the paper. According to this classification, we analyze the state of the art on quality models in Section 3, on quality meta-models in Section 4, and on Service Level Agreements in Section 5. In each section, we define a set of characteristics used for the comparisons and then we summarize and compare them according to level of satisfaction of these characteristics. An analysis

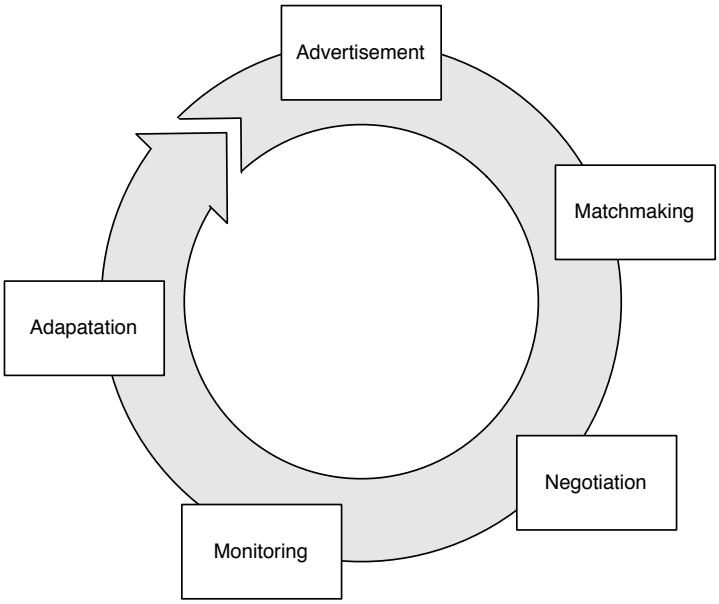

Fig. 2. Service life-cycle.

of interesting topics for further investigation is derived from these comparisons and it is discussed in the last section concerning future work.

2. SERVICE LIFE CYCLE AND QUALITY

Within an agreed/standard service life-cycle, like the one shown in Figure 2, QoS plays a significant role in almost every activity. Below we explain why quality documents are necessary for each activity of our reference life-cycle:

—Advertisement: requesters and providers publish or exchange quality requests and quality offers, respectively. Both of these quality documents are named as Quality-Based Service Descriptions (QSDs).

—Matchmaking: QSDs are matched in order to state if offers are able to support the user requirements. The result is that the advertised functionally-equivalent services are filtered and then selected based on their ability to satisfy the user quality requirements.

—Negotiation: QSDs or SLA templates are exchanged between service providers and requesters. The possible agreement between the parties involved leads to the definition of an SLA.

—Monitoring/assessment: the SLA is monitored, in order to discover customers and/or providers' violations of its functional and quality terms. Monitoring may also signal potential dangerous situations, that may lead to a violation of the SLA if corrective actions are not timely undertaken.

—Adaptation: in case of SLA unfulfillment, recovery/adaptation reactive and proactive actions may be taken. A possible recovery action might require a re-negotiation of the SLA or the execution of the matchmaking activity to find an

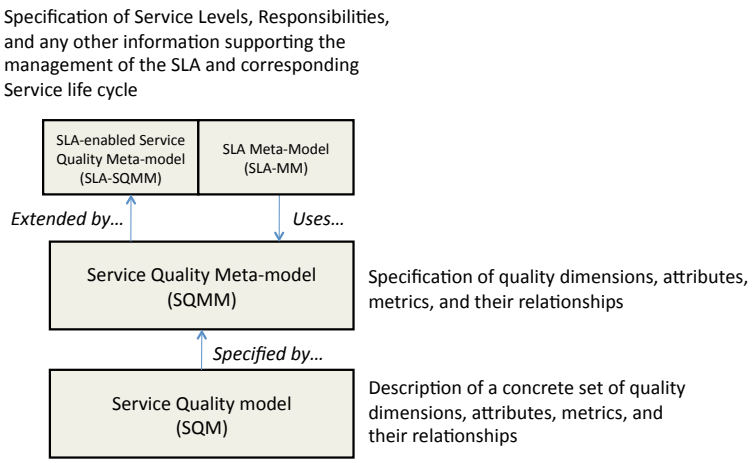

Fig. 3. Proposed Classification.

alternative service. It might also happen that an alert is sent to the assessment component of the monitoring activity that continues to execute.

Based on the above description, it can be easily noted that QSDs are used in the first two or three activities of the service life-cycle, while SLAs are used in the last three activities of this life-cycle. Thus, there is not any uniform and common quality document to be used across all the activities. This is a major drawback that requires time, as document transformations should take place from one format to the other, and reduces the automation degree of the activities.

In order to automate as much as possible the above activities, a clear and formal description of QoS is required. Moreover, service providers (SPs) and service requesters (SRs) should agree on the same language for expressing their quality documents. In this way, all the mechanisms used for supporting the service life-cycle can be properly enacted. Nowadays, in the literature many QoS description approaches can be found. In this survey, we firstly group them in three main categories as shown in Figure 3.

At the bottom level, we consider a specific type of quality documents that are called *Service Quality Models* (SQMs). SQMs are descriptions of a taxonomy or concrete list of QoS categories, attributes, metrics, and relationships that connect all of these quality entities. For example, a typical SQM may contain the *Performance* QoS category which includes the QoS attributes of *response time* and *throughput*. As it will be shown in Section 3, some proposed SQMs classify quality attributes in terms of relevant scenarios, other SQMs classify them in terms of their dependencies, while other SQMs classify them in terms of compliance to existing standards. Relying on these models means that SPs and SRs have to preliminary select which is the exact set of relevant quality attributes, where this selection is usually performed in an ad hoc way. SQMs are referenced in QSDs and SLAs in order to specify which quality attributes and metrics are going to populate the quality constraints that are used to describe quality offers, requirements, and service

levels. In other words, SQMs provide the concrete semantics of the quality terms that may be used in QSDs and SLAs, that is in other types of quality documents. In this way, all the service life-cycle activities as, for instance, matchmaking and monitoring, are designed around this set of quality attributes. Although the above procedure assists in producing suitable mechanisms for supporting the service life-cycle activities, the suitability of these mechanisms is specific for the considered scenario.

At the second level, we consider the *Service Quality Meta-Models* (SQMMs) proposed to provide the means for describing QoS in a more general and extensible way than SQMs. Actually, an SQMM is a conceptualization of the appropriate quality concepts and their relationships that can be used to capture and describe a SQM. For example, a typical SQMM will contain the concepts of *QoS category*, *QoS attribute*, and *QoS metric* and the relationships *contains* (from QoS categories to attributes) and *measuredBy* (from QoS attributes to metrics). So, an SQMM can describe many different SQMs, where the number of those SQMs and their actual difference mainly depends on the richness of the SQMM. In addition, SQMMs are used to specify service quality offers and requirements, i.e., *QSDs*, which are usually described by a set of constraints on some QoS attributes and metrics. Thus, existing SQMMs can be compared according to their expressiveness, as it will be shown in Section 4. On one hand, by adopting an SQMM, the mechanisms that use quality documents become more generic than those which adopt a specific SQM. Indeed, in this case, these mechanisms can be designed regardless of a specific set of quality attributes. In case SRs and SPs change the set of relevant quality attributes, the mechanisms remain the same. On the other hand, due to the intrinsic subjectiveness and complexity of quality, the existing SQMMs are not able to capture all the possible features of quality attributes and rely on a common understanding of the interacting parties about the concepts defined in the SQMM.

Finally, at the higher level, we consider the *SLA Meta-Models* (SLA-MMs). In this case, as it will be discussed in Section 5, the approaches involved allow the definition of *SLA*s and *SLA Templates* between the interacting parties. Since the terms of the agreement include Service Level Objectives (SLOs), which denote constraints on quality attributes or metrics listed in an SQM, and both SQMs and constraints may be defined by an SQMM, we highlight the existence of the three following cases: a) there are SQMMs, which we call *SLA-enabled SQMMs* (SLA-SQMMs), that are also able to define SLA specifications apart from QSDs; b) SLA-MMs may use one or more SQMMs in order to define and reference quality attributes and even specify SLOs; c) SLA-MMs may reference the contents of one or more SQMs. Various SLA description capabilities are taken into account when comparing existing SLA languages that concern the definition of the terms of the contract and various other information that may be used in supporting the service life-cycle activities.

In general, meta-models are used to define the abstract syntax of a language. Then, different concrete syntaxes may exist for the same language that are based on its abstract syntax. So, a meta-model drives the design of a language. Thus, one language has one and only underlying meta-model, while one meta-model may be used for the design of many languages. However, in practice, there is usually a

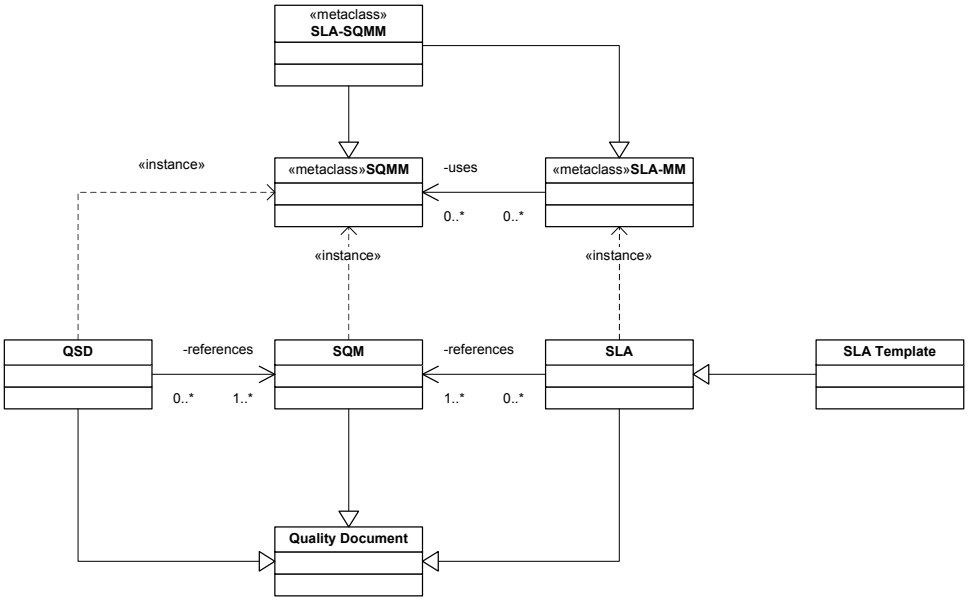

Fig. 4. UML diagram representing the main types of quality documents, their meta-models, and the inter-relationships between them.

one-to-one correspondence between a language and its underlying meta-model, as different languages of the same domain are designed by different modelers which have a different conceptualization of that domain. Indeed, to the best of our knowledge, *Service Quality Specification Languages* (SQSLs) and *SLA Languages* have an one-to-one correspondence with their underlying SQMMs and SLA-MMs, respectively. Thus, these corresponding terms will be used interchangeably in the remaining sections of this survey.

The UML diagram of Figure 4 represents the types of quality documents, their meta-models, and the various relationships involved between all of these entities.

3. SERVICE QUALITY MODELS

3.1 Background

As described in the previous section, an SQM focuses on the analysis of the set of quality attributes that are considered relevant in service applications. If we look at services as standalone software modules, then we can say that their quality is determined by the attributes that traditionally characterize software quality and, thus, by the attributes defined in the ISO 9126 model [ISO/IEC 2001]. A selection and/or specialization of such attributes is however needed to capture the peculiarities deriving from the service intended use, i.e., their composition to build service-based applications. Services in the publication and utilization phase are considered as black boxes that expose their external attributes to the audience of service-based developers. From this perspective, all of the external quality attributes proposed for traditional software are applicable, e.g. *privacy*, *security*, *performance*, and *reliabil-*

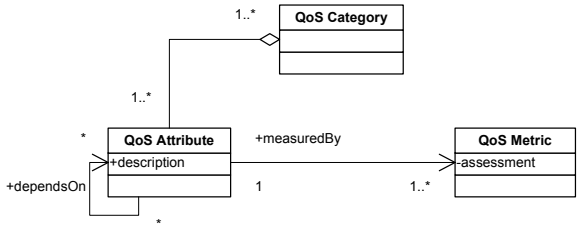

Fig. 5. SQM Structure.

ity. However, some internal quality attributes are not applicable (e.g., *analysability*, *changeability*) since they require the analysis of the software code that is hidden in the service-oriented programming or consider *portability* aspects which are de-facto covered by a service. On the other hand, some internal characteristics of software products can be useful at least during service design as they influence the way a composite service is built or executed. For instance, in case of the *stability* quality attribute, a composite service developer would be in favor of selecting service components that are more stable than others in order to build at design-time or run-time a stable composite service which will be used many times as it is without the need to update it at run-time. Thus, the ISO 9126 quality model could be used for defining software service quality but it has to be adapted accordingly by removing the non-applicable attributes and their categories (i.e., the majority of internal attributes) and refining or specializing other attributes (i.e., the rest of internal and all the external attributes).

Based on the above analysis, it is easy to understand that the ISO 9126 quality model is not adequate for representing service quality. In addition, it applies only to software services and not to other types of services like infrastructural ones. For this reason, different contributions can be found in the literature which propose various SQMs. The structure of the proposed models is based on the use of taxonomies in which categories, related to different analyzed aspects, are defined. Each category contains a set of attributes that are entities which can be verified or measured in the service. Most of the models associate each attribute with a definition and, in some cases, also the related metric and assessment formula are provided. The latter information is, of course, needed for measurable quality attributes [Kritikos and Plexousakis 2009]. The UML diagram depicted in Figure 5 presents the common structure of an SQM.

3.2 Methodology and Analysis

In order to analyze and compare the different SQM approaches according to the type and value of the information that they contain, a set of comparison criteria have been selected. A summary of these criteria is shown in Table I, while a thorough presentation of them is provided later on in this section.

We have collected the most significant SQMs proposed for services and their applications. These SQMs are depicted in Table II. It should be noted that we have taken into our consideration generic SQMs and not specialized models proposed for

Table I. Summary of the Comparison Criteria of Service Quality Models

Criterion	Summary
Quality Categorization Extensiveness	How many quality categories does the SQM have?
Level of Detail in Quality Categories	Does each quality category contain a sufficient number of quality attributes?
Containment of Domain-Independent and Domain-Dependent Quality Attributes	Does the SQM contain both domain-independent and domain-dependent quality attributes?
Consideration of Service Provider and Requester Views	Are the contained quality attributes relevant to the SP, SR, or both?
QoS and QoE Consideration	Does the SQM contain both quality attributes that are measured objectively and those that are measured subjectively?
Atomic and Composite Quality Attribute Inclusion	Are there any relationships expressed between atomic and composite QoS attributes?
Types of Dependencies	Is there any type of inter-attribute dependencies expressed?
Layer Designation	To which service layer or layers does a QoS attribute refer to?
Association with Assessment Guidelines	Are there any assessment guidelines associated to a QoS attribute?
Metric Identification	Are there one or more metrics associated to a QoS attribute?

security, data, and network aspects. The main reason of this omission is that in the analysis of each security and data quality model the service level and service type attributes are not considered, since the respective classifications have been often defined for generic applications and not for service applications. The results of the evaluation of the examined SQMs according to our comparison criteria are shown in Tables III and IV.

In the remainder of this section, we are going to present each criterion along with its evaluation results in separate sequential subsections. In the end, a global analysis of the SQMs across all criteria is given, while also the most frequent quality categories and attributes in the considered SQMs are distinguished.

3.2.1 *Quality Categorization Extensiveness.* Quality attributes are usually categorized into quality categories. This categorization is required as the set of quality attributes is usually large and it considers several aspects such as the higher the number of categories the higher the comprehensiveness of the approach. Categories also improve the readability of the model. In fact, a classification enables users to explore in a better way the model and to optimize the search activity. Categories that are defined in many SQMs are: *Performance*, *Security*, *Dependability*, *Configuration Management*, etc. A flat model would contain only one quality category while an extensive one can contain up to 9 or 10 quality categories. Grades of extensiveness: *flat* (1 category), *fair* (2-4 categories), *good* (5-7) categories and *extensive* (8-10) categories.

The results of this evaluation are presented in the second column of Table III. It

ID	Approach **Reference**
1	[Sabata et al. 1997]
2	[Anbazhagan and Nagarajan 2002]
3	[Czajkowski et al. 2002]
4	[Ran 2003]
5	[Lee et al. 2003]
6	[Liu et al. 2004]
7	[Colombo et al. 2005]
8	[The OASIS Group 2005]
9	[Cappiello 2006]
10	[Truong et al. 2006]
11	[Brandic et al. 2006]
12	[Sakellariou and Yarmolenko 2008]
13	[Cappiello et al. 2008]
14	[Frutos et al. 2009]
15	[Nessi Open Framework 2009]
16	[Kritikos and Plexousakis 2009]
17	[Mabrouk et al. 2009]

Table II. The service quality models examined

Research Approach ID	**Extensiveness**	**Detail Level**	**Domain Dependency**	**Provider Requester** View	**QoS** & **QoE**	**Composite** & **Atomic**
1	fair	fair	domain ind.	SP	QoS	both
2	fair	fair	domain ind.	SP	QoS	both
3	flat	fair	domain ind.	SP	QoS	atomic
4	good	fair	domain ind.	SP	QoS	both
5	flat	fair	domain ind.	SP	QoS	atomic
6	flat	low	domain ind.	both	both	atomic
7	flat	fair	domain ind.	both	both	atomic
8	fair	fair	domain ind.	both	QoS	both with rels
9	extensive	high	domain ind.	both	both	both
10	good	fair	domain ind.	SP	QoS	both with rels
11	good	fair	domain ind.	SP	QoS	both with rels
12	good	good	both	both	QoS	both
13	extensive	high	both	both	both	both with rels
14	good	high	domain ind.	SP	QoS	both with rels
15	fair	fair	domain ind.	both	both	both with rels
16	extensive	high	both	both	both	both
17	extensive	good	both	both	both	both with rels

Table III. Comparison of Research Approaches Providing Service Quality Models *(cont.)*

Research Approach ID	Dependency Types	Layer	Metric	Assessment Guidelines
1	no	service	partial	fair
2	no	service	none	none
3	no	serv. & appl.	none	none
4	no	all	partial	fair
5	no	serv. & infr.	none	none
6	no	serv. & appl.	complete	fair
7	no	serv. & appl.	complete	fair
8	no	serv. & appl.	complete	good
9	no	all	partial	none
10	no	serv. & infr.	partial	fair
11	no	serv. & infr.	partial	fair
12	no	serv. & infr.	partial	fair
13	no	all	none	none
14	no	all	partial	none
15	no	serv. & appl.	none	none
16	no	all	partial	fair
17	no	all	partial	fair

Table IV. *(cont.)* Comparison of Research Approaches Providing Service Quality Models

is possible to notice from these results that the majority of SQMs uses categories to classify the attributes and to improve the understandability of the model. Moreover, there is a balance between all the criterion-specific partitions of SQMS with the partition corresponding to good extensiveness to have a very small precedence. Finally, the trend that SQMs are improving according to this aspect (with the exception of the 15th approach) can be revealed from observing the last nine values of the corresponding column of the criterion.

3.2.2 *Level of Detail in Quality Categories.* The subject of research of this criterion is if each category contains a sufficient number of quality attributes. In this way, the higher is the number the higher is the probability that the most significant attributes are covered in this category. So when a quality category does not contain more than two attributes, its *level of detail* is: *low*. When it contains three to five attributes, it is *good*, while when it contains more than five attributes it is *high*. After characterizing in this way each quality category, we can generalize to express the *level of detail* of the whole SQM. In this case, we have the following levels: *very low* (every category has low *level of detail*), *low* (some categories have low and other have good *level of detail*), *fair* (some categories have low and other have good or high *level of detail*), *good* (all categories have good *level of detail*), *very good* (some categories have good and other have high *level of detail*), *high* (most categories have high *level of detail*).

The results of the evaluation of this criterion are presented in the third column of Table III. It is easily noticeable from these results that more than half of the SQMs have a fair level of detail which means that some of their categories contain less than three quality attributes. For the rest of the approaches, SQMs that present high level of detail are more than those having good, while there is only one approach with low level of detail. Finally, by observing the results, it can be noticed that

SQMs with low level of detail cease to appear before the half of the table and that approaches with higher level of detail than fair are starting to appear after this point. This means that SQMs have slightly improved over the years according to this aspect.

3.2.3 *Containment of Domain-Independent and Domain-Dependent Quality Attributes.* Domain-independent quality attributes are general/technical attributes that can characterize all services in any possible application domain. For example, it is difficult to find an SQM that does not contain the following (domain-independent) attributes: *response time* and *availability*. On the other hand, domain-dependent quality attributes are attributes that characterize services (or their parts) of one or more but not any application domain. For example, data-related attributes like *validity* and *timeliness* characterize the input or output data of services that manipulate data. So the evaluation of this criterion for a specific SQM would be: *domain ind.* (only domain-independent), *domain dep.* (only domain-dependent), and *both*.

The results of this evaluation are presented in the fourth column of Table III. It can be observed from these results that all the approaches tend to be general and not domain specific. In fact, all the SQMs propose domain-independent criteria that can be generally used in every context in which non functional properties are considered, while only four approaches enumerate also some domain-specific attributes of a particular application domain [Sakellariou and Yarmolenko 2008; Kritikos and Plexousakis 2009; Mabrouk et al. 2009] or some quality attributes that can be used in some domains according to a particular context [Cappiello et al. 2008]. The latter four approaches have been proposed very recently.

3.2.4 *Consideration of Service Provider and Requester Views.* this criterion considers the degree of relevance of the different quality attributes with respect to the service provider and requester. There are some quality attributes that are particularly relevant for the service provider such as *availability* and *response time*. And, in the same way, there are there are also quality attributes that are important to the requester such as *response time* and *usability*. As you can see, there can be some quality attributes that are both important to service providers and requesters so they belong to their *common* view, while there can be other attributes that are important only for one of these parties so they are either *SP-specific* or *SR-specific*. If the SQM captures both views, it should provide a set of quality attributes that is possible to use to express both provider capabilities and users' requirements. Thus, the evaluation of this criterion for a specific SQM would be: *SP* (if the SQM contains *SP-specific* and *common* quality attributes), *SR* (if the SQM contains *SR-specific* and *common* quality attributes), and *both* (if the SQM contains *common* and *SP-specific* and *SR-specific* quality attributes).

The results of this evaluation are presented in the fifth column of Table III. It is easily noticeable from these results that more than half of the SQMs consider quality attributes that correspond to both provider and requesters view. This means that researchers have understood the need of expressing both views in an SQM. Moreover, another interesting observation is that the results of the evaluation of this criterion are in accordance with the results of the evaluation of the previous

criterion. First, domain independence signifies that the selected quality attributes are more specific to the service and independent of its usage in specific applications, so at least the provider view is covered. This is because the provider is more concerned about having his service used across many application domains so he focuses more on those quality attributes that are generic and tend to distinguish his service from other services independently of the application domain. Second, domain dependence signifies that the selected attributes are specific to an application and its usage, so they have an impact on the user's expectations. Thus, domain-dependent attributes tend to cover the requester view. Indeed, by cross-checking the results of this and the previous criterion (domain-independence), we can observe that when only domain-independent attributes are contained in an SQM, then at least the SP's view is captured. In addition, when an SQM also captures domain-dependent attributes, then also the requester's view is captured. A final interesting characteristic that can be observed from the results is that while in the past most SQMs were capturing only the SP view, this situation is changed in the recent years as the more recent SQMs (from 2006 and on) tend to cover both views.

3.2.5 *QoS and QoE Consideration.* Quality of Service (QoS) includes quality attributes that can be objectively measured (like *response time*) and are typical constituents of SLAs. On the other hand, Quality of Experience (QoE) includes quality attributes that can be measured subjectively (e.g., *reputation*, *usability*) and reflects the perception of individuals or groups of service users. From the above definitions, it is easily understandable that QoE attributes reflect the requester view. However, two main questions arise that may be addressed by the evaluation of this criterion and the observation of the results of the evaluation of the previous criterion: a) "are all requester-view quality attributes QoE or not?", and b) "what is the situation with the provider-view quality attributes, in other words do the provider-view attributes contain QoS, QoE, or both types of attributes?". Obviously, both sets of attributes are important and should be represented by using an SQM. Thus, the evaluation of this criterion for a specific SQM would be: *QoS* (only QoS), *QoE* (only QoE), and *both*.

The results of this evaluation are presented in the sixth column of Table III. It is easily noticeable from these results that most SQMs contain only QoS attributes. This means that most researchers are interested in developing SQMs which contain attributes that can be measured objectively. Perhaps, they do not take into account QoE attributes because either they do not consider them so important or some of these attributes are difficult to be measured or they require user feedback which is not always easy to be collected and processed. Fortunately, in the recent years this situation is changing as the more recent SQMs tend to cover both types of attributes.

By closing looking the results of this criterion and those of the previous one, some other interesting facts can be inferred. First, when only the SP view is covered in an SQM, then only QoS attributes are also covered by the SQM. This means that the quality attributes that are more important to an SP tend to be QoS attributes. This is quite reasonable as the SP-view quality attributes are in their majority domain-independent attributes that should be measured objectively in order to be able to meaningfully compare services across all application domains. Second, when

also the SR view is covered by an SQM (apart from the SP view), then either QoS or both QoS and QoE attributes are covered by the SQM. This signifies that the SR-view corresponds to both QoS and QoE attributes. Obviously, on one hand, the SR-view domain-independent attributes will tend to be QoE attributes. This is because these attributes are going to be assessed differently from users across the various application domains because in each domain the usage and the requirements from a service are different with respect to the other domains. On the other hand, SR-view domain-dependent attributes will tend to be QoS attributes. This is because in a specific application domain the requirements and the usage of a service are specific or they vary in a specific way according to the user expectations, while the user's domain expertise is high. In this way, the SR-view domain-dependent attributes will tend to be measured with well-established domain metrics or will be assessed by the same objective way by users that have similar expectations.

3.2.6 *Atomic and Composite Quality Attribute Inclusion.* Some attributes are *composite* and can be computed by evaluating the values of other attributes. For example, *response time* (the parent) is a composite attribute since it can be assessed by evaluating *latency* and *network delay* (its children). Other quality attributes, like *execution time*, are *atomic* and do not rely on other attributes. The composability aspect is important as it can be used later by meta-models to capture this type of inherent relationship between the parent and child quality attribute. Moreover, if this relationship is associated with a specific mathematical formula, then it can also drive the way the parent quality attribute is measured from its child quality attributes. In addition, even if this relationship is captured by a simple connection (symbolic or phrasal), it can be useful in case of service monitoring for checking for example if the increase in the value of the child attribute causes an increase in the value of the parent attribute. So it is important to consider if both types of attributes are captured and if the relationship between parent and child attributes is captured. Thus, the evaluation of this criterion for a specific SQM would be: *atomic* (only atomic attributes are included), *composite* (only composite attributes are included), *both* (but types of attributes are included but with no parent-child relationships) and *both with rels* (both types of attributes are included along with their parent-child relationships).

The results of this evaluation are presented in the seventh column of Table III. As it can be easily seen, the majority of the SQMs contain both atomic and composite quality attributes. Moreover, most SQMs that contain both types of attributes also capture the connecting relationships between composite and atomic quality attributes. In addition, it should be noted that all SQMs that have been proposed after 2005 contain both types of attributes. So there is a trend of SQM improvement with respect to this aspect. Finally, by inspecting the results of the evaluation of the first and second criterion apart from the current results, it can be inferred that an SQM contains only atomic quality attributes when its category extensiveness is flat and its level of detail is low or fair. Indeed, by thinking reasonably, such an SQM is built and structured in this way for supporting fact computation (i.e. service matchmaking and selection) algorithms which require that no inter-attribute dependencies exist and that there are only QoS attributes that can be immediately measured by specific metrics without introducing other more higher

levels of measurement.

3.2.7 *Types of Dependencies.* Not all quality attributes are independent from each other. There can be two types of dependencies between quality attributes: *quantitative* and *qualitative*. The former are expressed by mathematical formulas or constraints and a specific subclass of them are the parent-child relationships (i.e attribute derivation formulas), while the latter are expressed symbolically or descriptively. An example of qualitative dependency is that *availability* and *reliability* have "a positive correlation" i.e an increase of the one's value causes an increase to the other's value. So we want to check if concrete dependencies between quality attributes exist in the SQM. However, we have to be careful here as this criterion is for the model and not for the meta-model so we do not care about which modeling constructs are used. Thus, the evaluation of this criterion for a specific SQM would be: *no* (no dependencies are expressed), *quantitative* (only quantitative dependencies are expressed), *qualitative* (only qualitative dependencies are expressed) and *both* (both types of dependencies are expressed).

The results of this evaluation are presented in the second column of Table IV. As it can be easily seen, qualitative and quantitative dependencies are scarcely addressed by the SQMs. The analyzed approaches present the attributes together with their definitions but do not analyze thoroughly the influences among the different attributes.

3.2.8 *Layer Designation.* Our reference model (i.e. SFAM) consists of three layers: *application*, *service* and *infrastructure*. The *application* and *service* layers usually have a set of identical attributes with the exception that the application attributes are produced from their service counterparts. Moreover, some business or user-oriented attributes are associated to the application layer. The infrastructure layer usually contains a completely different set of quality attributes with respect to the other two layers. So it is important to clarify to which service layers the SQM refers to. Thus, the evaluation of this criterion for a specific SQM would be: *service*, *serv. & appl.* (service and application), *infr.* (infrastructure), *serv. & infr.* (service and infrastructure) and *all* (all layers are referenced).

The results of this evaluation are presented in the third column of Table IV. As it can be easily seen, all SQMs contain service-layer quality attributes, something quite reasonable as all SQMs are built by having specialized focus on the components of the service layer, i.e. the software services. Moreover, more than half of the SQMs also contain either infrastructure-layer or application-layer attributes. This means that these SQMs have been built with the aim of addressing two out of three layers of our SFAM model, characterizing in this way the quality of different types of services. In addition, six SQMs have adopted a holistic approach that aims to characterize the quality of all types of service across the three layers of our model. Finally, it should be noted that only the two initial SQMs contain solely service-layer quality attributes. This means that the researchers have quickly understood the need of covering quality attributes from other layers apart from the service one.

3.2.9 *Metric Identification.* identification of the quality metric used to measure a quality attribute. Metrics are entities that encapsulate all appropriate measurement details of an attribute such as measurement values, units, formulas, and

schedules. However, we have decided to split the measurement formula or assessment algorithm of a metric from this evaluation and use it as a separate comparison criterion. The rationale for this choice is that some SQMs provide the name of a metric and some of its details, when they associate it to an attribute, but not an assessment guideline or algorithm for it. So we wanted first to evaluate which SQMs associate specific metrics to attributes and then to evaluate the existence or not of assessment guidelines for the incorporated metrics. So the evaluation of this criterion for each SQM will take particular discrete values, such as *none* if no metrics are provided in the SQM, *partial* if the SQM associates metrics to a subset of the quality attributes, and *complete* when all the attributes are associated with at least one corresponding metric.

The results of this evaluation are presented in the fourth column of Table IV. It can be noticed from these results that more than half of the SQMs associate metrics to a subset of the contained quality attributes, while only three SQMs are complete with respect to this aspect. The reason of having many *partial* SQMs can be twofold: a) some quality attributes are not measurable or are difficult to measure (e.g. QoE attributes), and b) there was a decision of associating metrics only to the most popular or referenced/used (in QSDs and SLAs) attributes. Finally, it must be noted that almost one third of the SQMs do not associate metrics to attributes, which significantly limits their usage in specific service quality description/advertisement and matchmaking scenarios.

3.2.10 *Association with Assessment Guidelines.* An SQM may provide algorithms to assess the quality attributes that it defines. The evaluation for this criterion would be: *none* if guidelines are not provided, *fair* if simple assessment rules are provided, and *good* if the authors specify precise assessment algorithms.

The results of this evaluation are presented in the fifth column of Table IV. As it can be easily seen, more than half of the considered SQMs do not support the users at all in the assessment procedures, while less than half SQMs just specify some guidelines for each metric. This means that the majority of the SQMs cannot be used in the monitoring and assessment service life-cycle activity, limiting in this way their use mostly in the advertisement and matchmaking activities. It must be noted that only one approach provides precise assessment algorithms for all defined metrics. This approach also happens to associate metrics to all its contained attributes. Thus, this approach is suitable for annotating QSDs and SLAs which can be used across all service life-cycle activities. However, this approach can be used in specific scenarios as it contains a small set of domain-independent quality attributes and not domain-dependent ones. Finally, by continuing to correlate the evaluation results of this criterion with those of the previous one, it can be inferred that most of the SQMs that associate metrics to some or all of their quality attributes do provide simple assessment rules for them which could be further used to create precise assessment algorithms.

3.3 Overall Analysis

Based on the above analysis, no SQM can be distinguished as the best according to its evaluation on all the considered criteria. On one hand, by considering the first six criteria plus the "Layer Designation" one which relate to the extensiveness,

structure, and generality of the SQM, four approaches with IDs 9,13,16, and 17 can be distinguished as the best [Cappiello 2006; Cappiello et al. 2008; Kritikos and Plexousakis 2009; Mabrouk et al. 2009]. On the other hand, by considering the last two criteria which relate to the attribute assessment and applicability of the SQM, then the approach with ID 8 [Colombo et al. 2005] can be distinguished. It must be also noted that all approaches have the worst behavior with respect to the "Types of Dependencies" criterion. Thus, a new SQM is needed that should both combine the characteristics of the best approaches in the above two criteria partitions and describe all the possible but realistic inter-attribute dependencies.

In addition to the criteria-based analysis, we have assessed the frequency of the service quality categories and attributes across all SQMs in order to distinguish the most frequent ones. Table V shows the service quality attributes considered by the approaches surveyed in this paper. Each considered attribute is associated with the approach, if the approach provides a corresponding definition.

In summary, although several different QoS attributes and categorizations can be found in the various proposals, it is possible to single out a few of them that appear most often and can be considered as "basic" QoS attributes and categories, respectively. All other noticed attributes either capture secondary features or are more context-dependent (i.e. very specific), while they appear very scarcely in the proposed SQMs. In this way, we can consider that the most frequent QoS attributes and categories are also the most important ones. As can be seen from Table V, *response time*, *latency*, and *throughput* are the attributes that mostly represent the *performance* category, which is present in the majority of SQMs. Another important category is *security* which has three attributes, namely *authentication*, *authorization*, and *non-repudation*, that are steadily present in the SQMs. *Availability*, *accuracy*, and *reliability* are the remaining three most important attributes. Another interesting observation is that internal software product quality attributes are not represented at all in any approach. This means that either these attributes are not so important for composite service developers or the SQM modelers have neglected them and have put their focus on their external counterparts.

It is worth to note that data quality aspects are scarcely considered. The only data quality aspect that is mostly considered is *correctness*. The *accuracy* attribute defined by different contributions refers to the correctness of the service and also of the data that it provides. Since the output of a service is mostly composed of information, data quality can be considered as a part of the QoS and it can drive thoroughly the analysis of the required input and provided output. In the literature, we can find several contributions about data quality attributes and taxonomies/SQMs (e.g., [Redman 1997], [Strong et al. 1997]). Data quality is a multidimensional concept that defines the suitability of the used data for the service in which is involved. The most important and representative data quality attributes (according to the data quality research) which should be part of a QoS model are *accuracy*, *completeness*, *consistency*, and *timeliness*. These attributes could be easily applied to the service world in order to check the correctness of data, the existence of missing or contradictory values, and the updateness of the information provided.

Some of the proposed SQMs take into account particular network aspects. In

Attributes	Approach ID												
	1	2	4	6	7	8	9	10	13	14	15	16	17
Performance (Response Time)		X	X		X	X	X	X	X	X	X	X	X
Performance (Processing Time)		X		X			X	X	X			X	
Performance (Latency)		X	X		X		X	X	X	X		X	
Performance (Timeliness)	X				X		X						
Performance (Precision)	X												
Performance (Throughput)		X	X		X	X	X		X	X	X	X	X
Availability		X	X		X	X	X	X	X	X	X	X	X
Accessibility		X				X	X	X	X	X		X	X
Accuracy	X	X			X		X	X	X	X		X	
Reliability		X	X		X		X	X	X	X	X	X	
Capacity		X	X					X	X	X		X	
Believability								X					
Maintainability					X			X			X	16	
Relative Importance	X												
Complexity					X								
CustomerService					X								
Dependability		X			X		X		X		X	X	
Stability		X	X		X	X			X			X	X
Trust				X	X		X						
Understandability					X				X				
Integrability											X		
Interoperability		X					X				X		
Resource Efficiency											X		
Reusability											X		
Scalability		X	X				X		X		X	X	
Security (Authentication)			X			X	X	X	X		X	X	X
Security (Authorization)			X			X		X	X	X	X	X	X
Security (SecurityLevel)	X							X				X	
Security (Integrity)		X	X			X		X	X			X	X
Security (Confidentiality)	X	X	X			X	X	X	X	X		X	X
Security (Accountability)			X					X	X	X		X	
Security (Traceability)			X			X			X	X		X	X
Security (Non repudiation)			X			X	X		X	X	X	X	X
Security (Data Encryption)									X	X	X	X	
Security (Isolation)											X		
Configuration (VirtualOrganization)							X	X					
Configuration (Location)								X					
Configuration (LevelOfService)	X							X	X				
Configuration (Service Version)								X					
Configuration (Supported Standard)		X	X		X	X		X	X			X	X
Cost	X		X	X			X	X	X	X		X	
Data (Maturity/Age)					X		X		X				
Data (Timeliness)							X		X			X	
Data (Reliability)							X		X				
Data (Completeness)							X		X			X	

Table V. Attributes defined in service quality approaches

these SQMs there is usually a *Network* quality category which mainly contains the four most frequent network attributes, namely *bandwidth*, *network delay*, *jitter*, and *packet loss*.

4. SERVICE QUALITY METAMODELS

4.1 Background

SQMMs have been mainly used for describing the service quality offered or requested by an SP or SR, respectively (WS-QoS [Tian et al. 2003], RAN-UDDI [Ran 2003], WSAF-QoS [Maximilien and Singh 2004], CRSL [Degwekar et al. 2004], DAML-QoS [Zhou et al.], QoSOnt [Dobson et al. 2005], QRL [Cortés et al. 2005], UML QoS [The OMG Group 2005], WSMO-QoS [Wang et al. 2006], OWL-Q [Kritikos and Plexousakis 2006], onQoS-QL [Giallonardo and Zimeo 2007], WS-QoSOnto [Tran 2008], QoS-MO [Tondello and Siqueira 2008], MWLXL [Ma et al. 2008]). As service description is a prerequisite for service discovery, the content of SQMMs has been used for quality-based service matchmaking (QBSM) and service selection in service registries. QBSM is a process executed after functional service discovery (FSD) to further filter out a registry's service descriptions (SDs) based on their service quality capabilities with respect to the service quality requirements of a SR. It must be noted that SDs specify both the functional and quality capabilities of services. The results of the QBSM process may be ranked, if it is needed, by executing the service selection process.

Apart from their ability to describe SQMs, SQMMs are also able to produce *QSDs*. QSDs are often associated with a *validity period* or *expiration time* which signifies when they become outdated. Depending on which party is producing them, QSDs can be separated into *Service Quality Offers* (produced by an SP) and *Service Quality Request* (produced by an SR). Service Quality Requirements are further separated into *Service Quality Requirements* and *Service Selection Models*. The latter denote the significance of each quality attribute or metric to the SR by associating it with a specific weight and are used for ranking SDs. Both Service Quality Offers and Requirements are expressed as a set of *quality constraints*. A quality constraint usually contains a comparison operator that is used to compare a quality metric or attribute with a value. Sometimes, a quality constraint may also contain the unit of the compared value. This conceptualization of the content of the QSDs is common among the majority of the SQMMs and is visualized as a UML diagram in Figure 6.

As was briefly analyzed in Section 2, there are some SQMMs that are able to describe SLAs such that they are also considered SLA-MMs. These SQMMs were called *SLA-enabled* SQMMs (SLA-SQMMs). The corresponding languages of this type of SQMMs include QML [Frølund and Koistinen 1998], WSLA [Keller and Ludwig 2003], WSOL [Tosic et al. 2003] and SWAPS [Oldham et al. 2006]). It should be stressed that in this section we are only interested in meta-models that define quality attributes and the corresponding service quality capabilities and levels, so we do not take into account SLA languages that do not encompass an SQMM (e.g., WS-Agreement [WS-AGREEMENT 2003]). Section 5 will analyze all kinds of SLA languages with the appropriate SLA-based criteria.

As far as security is concerned, we consider that some aspects of it like *trust*

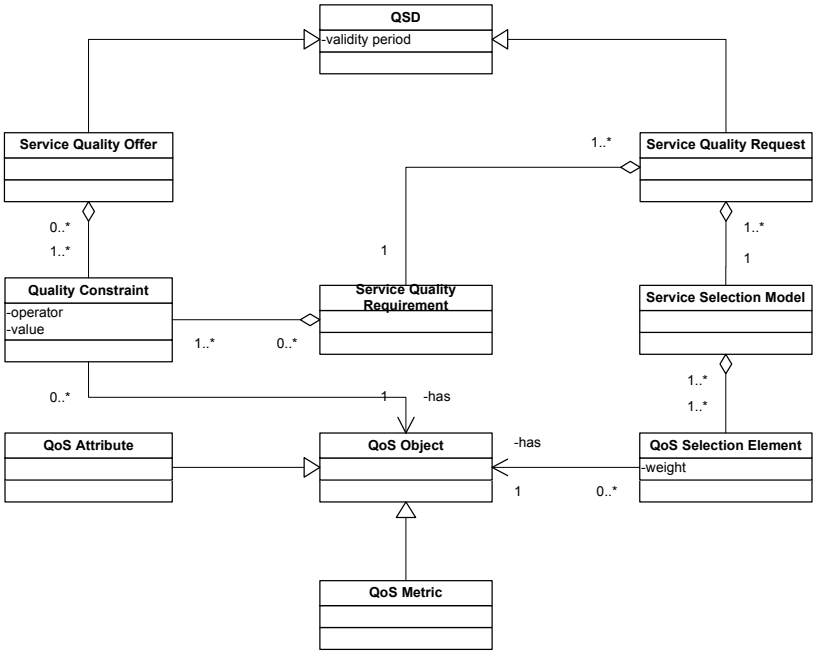

Fig. 6. QSD Conceptualization.

and *privacy* are orthogonal to service quality and are usually separated from the service quality description with respect to the other security aspects. In this way, we consider another partition of SQMMs which map to languages, such as TrustServ [Skogsrud et al. 2004], PeerTrust [Nejdl et al. 2004], WS-ABAC [Shen and F.Hong 2006], WS-Trust [Nadalin et al. 2007], and P3P [Cranor et al. 2006], that describe a part of service quality which is not described in the rest of the SQSLs. We call the SQMMs of this type *security-based* SQMMs.

4.2 Methodology and Analysis

In order to analyze all SQSLs and their underlying SQMMs and compare them on their ability to define quality, a set of comparison criteria have been chosen, which were either devised by the authors or collected from other research approaches [Frølund and Koistinen 1998; Tosic et al. 2002; Maximilien and Singh 2002; Cortés et al. 2005; Paoli et al. 2008; Kritikos 2008; Kritikos and Plexousakis 2009]. These criteria mainly reflect the expressiveness, complexity, and applicability of the examined SQMMs. The summary of our selected criteria (without their grouping) is shown in Table VI, while their complete presentation is provided later on in this section.

Based on the SQMM categorization of the previous subsection, there are actually three partitions of SQMMs: *pure*, *SLA-enabled*, and *security-based*. The content of these three partitions with respect to the corresponding languages of the SQMMs can be seen in Table VII. The results of the evaluation of the examined SQSLs ac-

Table VI. Summary of the Comparison Criteria of Service Quality Specification Languages

Criterion	Summary
Formalism	The language's expression formalism
Quality Model	Expressiveness in defining SQMs
Metric Model	Expressiveness in defining metric models
Constraint Model	Expressiveness in defining service quality constraints
Complexity	The complexity of producing SQMs and QSDs
Service Description Separation	Separation of functional and quality-based SDs
Service Description Refinement	Refinement/reuse of QSDs
Service Description Granularity	The ability to define quality constraints for the various service components
Symmetric but Separate QSDs	QSDs should be defined for both SPs and SRs in the same way but separately
Class of Service	The ability to produce different QSDs for the same service
Connection	Connection of a language's QSDs with functional SDs of a specific language
Quality Matching	The way QSDs of SPs and SRs should be matched
Framework Support	In which type of frameworks is the language used

cording to our comparison criteria are presented in three pieces: two tree structures (see Figs. 7 and 8) and one table (see Table VIII). The use of the tree structures apart from the table is due to three main reasons: 1) it was impossible to present the comparison results of all criteria in one table, 2) some criteria were closely related to each other, and 3) the results of one criterion were too complex to be presented in a table. Moreover, as can be easily seen, Table VIII is separated into three clusters, each one presenting the evaluation results of one particular SQSL partition.

In the remainder of this section, we are going to present each criterion or group of criteria along with its evaluation results in separate sequential subsections. The analysis of the evaluation results for each criterion or group of criteria takes place both globally for all SQMMs and locally for each partition. In the end, a global analysis of the SQMMs across all criteria is given.

4.2.1 *Formalism.* This criterion has been chosen in order to distinguish SQMMs (i.e., the meta-models that define the abstract syntax of quality languages) depending on the formalism that has been used to represent them. Various formalisms have been used to express an SQMM including *informal* (such as DTDs or XML Schemas), *UML* and *ontologies*. Each formalism has its own advantages and disadvantages. For example, ontologies provide a formal, syntactic and semantic description model of concepts, properties and relationships between concepts. They are extensible, human-understandable and machine-interpretable and they enable reasoning support by using Semantic Web technologies. However, sometimes their expressive power is more than needed while also the tool support is not so efficient with respect to the other formalisms. Moreover, current ontology tools from the research community require expertise in knowledge representation. Based on the above analysis, the evaluation of this criterion for each SQMM could get the following values: *informal*, *UML*, *ontologies*, and *other*.

Partition	Approach Name	Approach Reference
Pure	WS-QoS	[Tian et al. 2003]
	RAN-UDDI	[Ran 2003]
	WSAF-QoS	[Maximilien and Singh 2004]
	CRSL	[Degwekar et al. 2004]
	DAML-QoS	[Zhou et al.]
	QoSOnt	[Dobson et al. 2005]
	QRL	[Cortés et al. 2005]
	UML QoS	[The OMG Group 2005]
	WSMO-QoS	[Wang et al. 2006]
	OWL-Q	[Kritikos and Plexousakis 2006]
	onQoS-QL	[Giallonardo and Zimeo 2007]
	WS-QoSOnto	[Tran 2008]
	PCM	[Paoli et al. 2008]
	MWLXL	[Ma et al. 2008]
SLA-enabled	QML	[Frølund and Koistinen 1998]
	WSOL	[Tosic et al. 2003]
	WSLA	[Keller and Ludwig 2003]
	SWAPS	[Oldham et al. 2006]
Security-based	Trust-Serv	[Skogsrud et al. 2004]
	PeerTrust	[Nejdl et al. 2004]
	WS-ABAC	[Shen and F.Hong 2006]
	WS-Trust	[Nadalin et al. 2007]
	P3P	[Cranor et al. 2006]

Table VII. SQMM partitions and their corresponding approaches

The results of this evaluation are presented in the second column of Table VIII. As it can be seen, the majority of the approaches are using either ontologies or informal formalisms (mostly schema languages which focus on the concrete syntax of a language), while only two are using UML. We believe that this result is reasonable as ontologies are powerful modeling formalisms and very expressive, while informal formalisms are simple and very well supported. As far as separate partitions are concerned, pure SQMMs are modeled mostly through ontologies, while, in the other two partitions, an informal formalism is the best modeling choice. A final remark to be made is that for pure and SLA-enabled SQMMs there is a recent trend to use ontologies for their modeling. This is quite reasonable as ontologies provide unambiguous semantics to quality terms and, thus, enable machines to automatically process and reason on ontology-specified QSDs in order to support service life-cycle activities like discovery and negotiation.

4.2.2 *Richness and Extensibility of the Service Quality Model.* In the presence of multiple services with overlapping or identical functionality, service requesters need objective quality criteria to distinguish between these services. However, it is not practical to come up with a standard SQM that can be used for all services in every domain. This is because quality is a broad concept that encompasses a number of non-functional properties such as privacy, reputation and usability. Moreover, when evaluating service quality, domain specific criteria must be taken into consideration. For example, in the *Traffic Monitoring* application domain, the *routes set* QoS attribute is the set of route types that a WS supports, e.g. a

Research Approach	Formalism	Quality Model	Metric Model	Constraint Description	Complexity	Class Of Service	Connection	Quality Matching	Framework Support
WS-QoS	Informal	Fair	Fair	Poor	Low	Many	WSDL	No	No
RAN-UDDI	Informal	Poor	Poor	Poor	Low	One	WSDL	No	Discovery
WSAF-QoS	Ontologies	Good	Poor	Poor	Low	One	OWL-S	No	Discovery
CRSL	Informal	Poor	Poor	Rich	Low	One	WSDL	No	Discovery
DAML-QoS	Ontologies	Fair	Fair	Rich	Low	Many	OWL-S	No	Discovery
QoSOnt	Ontologies	Fair	Fair	Good	Low	Many	OWL-S	No	Discovery
QRL	Informal	Poor	Fair	Rich	Low	One	No	No	Discovery
UML QoS	UML	Good	Fair	Rich	Medium	Many	No	No	No
WSMO-QoS	Ontologies	Fair	Good	Good	Low	Many	WSMO	No	Discovery
OWL-Q	Ontologies	Good	Rich	Rich	High	Many	OWL-S	Yes	Discovery
onQoS-QL	Ontologies	Good	Good	Rich	Medium	Many	OWL-S	Yes	Discovery
WS-QoSOnto	Ontologies	Fair	Good	Good	Medium	Many	No	No	No
PCM	Ontologies	Fair	Good	Rich	Medium	Many	WSMO	Yes	Discovery
MWLXL	Ontologies	Fair	Good	Poor	Low	Many	OWL-S	Yes	Discovery
QML	Informal & UML	Fair	Good	Excellent	Low	One	No	No	No
WSOL	Informal	Poor	Good	Rich	Low	Many	WSDL	No	Discovery
WSLA	Informal	Poor	Good	Excellent	Medium	Many	WSDL	No	Negotiation
SWAPS	Ontologies	Fair	Good	Rich	Medium	Many	WSDL	Yes	Discovery
Trust-Serv	Informal	Fair	Fair	Good	Low	Many	WSDL	No	Discovery
PeerTrust	Ontologies	Good	Good	Rich	Low	Many	OWL-S	No	Discovery
WS-ABAC	Informal	Good	Good	Good	Low	Many	WSDL	No	Discovery
WS-Trust	Informal	Fair	Fair	Poor	Low	One	WSDL	No	No
P3P	Informal	Good	Fair	Good	Low	Many	WSDL	No	Discovery

Table VIII. Comparison of SQMM approaches on eight evaluation criteria

WS may provide information only on national highways while other WSs may also consider inter-state or local routes. Therefore, a rich and extensible SQM must be enabled by the SQMM that includes both generic and domain specific attributes and that can be extended appropriately with the addition of new quality attributes.

However, the richness of the SQM should not stop at enumerating all possible quality attributes but depends also on other additional modeling capabilities that concern the offering of constructs that enable the description of a quality attribute in every possible detail/aspect. The following list summarizes all the criteria that are needed in order to assess the richness of an SQM:

(1) Enumeration of all possible quality attributes
(2) Modeling the attribute's domain (e.g. phone service provisioning) (i.e. the entity and its relation to the "attribute" entity)
(3) Modeling of inter-attribute relationships/dependencies (either one or both of the two types)
(4) Modeling the attributes compositionality (i.e. if it is composite or not (and what are its child attributes))
(5) Modeling the different views which an attribute may concern, i.e., the SP's, SR's or both views
(6) Distinguishing by using appropriate constructs between QoS and QoE attributes
(7) Distinguishing by using appropriate constructs between domain-dependent and domain-independent attributes
(8) Modeling the service layer an attribute refers to
(9) Modeling the association/relationship between quality attributes and metrics

It must be noted that for the last sub-criterion we do not inspect the richness of the metric model, as this is the subject of the next criterion.

As all of the SQMMs we are aware of advertise that they are extensible, we focus only on the richness criterion. The evaluation of this criterion for a specific SQMM depends on the meta-model's satisfaction on the nine previously analyzed sub-criteria. It must be noted that if a SQMM does not comply with all the modeling requirements for a specific aspect/sub-criterion, then this SQMM does not satisfy the sub-criterion. So if the considered SQMM satisfies only 1-2 sub-criteria, then it is considered *poor*. If it satisfies 3-4 criteria is considered *fair*. Otherwise, if it satisfies 5-7 criteria, it is considered *good*. Finally, if the meta-model satisfies 8-9 criteria, it is considered *rich*.

The results of this evaluation are presented in the third column of Table VIII. As it can be seen, most of the SQMMs can describe a *fair* in richness SQM. Then, the rest of the SQMMs can either describe a good or poor SQM. So, there is not any SQMM that can describe a rich SQM, which means that there is not a perfect approach capturing this modeling aspect. As far as separate partitions are concerned, the above general result also applies to pure SQMMs, while there is also a trend revealing that the richness of the SQM is improved in recent pure SQMM approaches. Moreover, SLA-enabled SQMMs do not perform well in this modeling aspect, while security-based SQMMs perform moderately (there is a balance between fair and good approaches). For SLA-enabled SQMMs this fact was actually expected because these approaches give more importance on how a quality

attribute can be measured than how it is modeled. As far as security-based SQMMs are concerned, they only encompass a fair or good SQM and not a poor one. This result is expected as in these approaches the definition of quality attributes is of major concern.

4.2.3 *Richness and Extensibility of the Quality Metric Model.* An attribute is measured through the abstraction of a metric. While a metric model is encompassed in a quality model, we chose to inspect the capability of a SQMM to express them in a rich way separately for two main reasons. First, quality attributes, the main entities in a quality model, and metrics are two different concepts. Second, both of these concepts require a quite rich description. In fact, the latter fact can be proven by inspecting the number of sub-criteria that were used in order to compare the SQMMs richness in describing both of these concepts. Thus, the overall richness of an SQMM depends on the richness of both its quality and metric models.

Based on the above reasoning, the SQMM should enable the creation of a rich and extensible service quality metric model. Moreover, the following aspects must be considered for every quality metric:

(1) *The dynamicity of the metric.* Metrics should be distinguished between dynamic and static metrics. Dynamic metrics measure dynamic quality attributes (like *availability*) that change over time and their values are computed according to a schedule or trigger. Static metrics measure static quality attributes (like *security*) and have a specific value that does not change continuously over time. This means that these attributes are not only controllable but also fixable. In other words, the SP is able to guarantee a fixed value for them for his service even if the service's context changes.

(2) *The value type for the metric.* A metric value type specifies a domain of values. These values can be used in constraints involving this metric. The domain of values may be ordered. For example, a numeric domain comes with a built-in ordering that corresponds to the usual ordering on numbers. So for numeric domains, only the maximum and minimum value of the domain along with its numeric type (e.g., real or integer) have to be modeled, unless the domain is not continuous. In the latter case, it can be expressed as a union of continuous domains. In practice, numeric domains are usually only continuous for quality metrics. 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. So for this type of domains, not only all the values of the domain have to be explicitly modeled but also their ordering has to be specified. Of course, depending on the semantics of the quality metric (e.g. if we do not care about which value it takes but on the amount of these values), the natural partial order of sets defined by inclusion can be used. In addition, the order in which the elements of an enumeration are defined may also be their sorting order but the user, in this case, has to define if it is increasing or decreasing. Finally, for unordered domains only the values of the domain have to be explicitly defined.

(3) *The unit of the metric.* The values a metric can take are measured in specific units. For example, a metric measuring *execution time* is usually measured in *seconds*. Concerning the modeling of units, describing just the name of a

unit is not enough because additional specific information regarding how to convert a value expressed in a specific unit to a value expressed in another unit has to be modeled. This information is crucial in cases where the SP and SR express a metric constraint using different units or when combining metric measurements of different units that come from different sources. Units should be separated into *basic* units and *derived* units. Basic units should have a name and a short abbreviation. Derived units are produced by other units with the following two alternatives ways: a) by multiplying the basic unit with a specific (float) value (i.e. multiples of units); b) by dividing one unit with another one. For example, the unit of *minutes* is produced by dividing the unit of *seconds* with 60. As another example, one unit for the *throughput* quality attribute is "bytes/second" produced by dividing the unit of "bytes" with the unit of "seconds". The first sub-type of derived units should be directly associated to the basic unit with a specific property/relationship, while another property stating the multiplying factor should be modeled. The second sub-type should be modeled by two relationships stating which units are directly proportional to it and which units are inverse proportional to it, while also a multiplying factor should be associated with this sub-type.

(4) *The measurement directive or function of the metric.* Quality metrics should be classified into *resource* and *composite* metrics. *Resource metrics* are directly measured from the service's system instrumentation through *measurement directives*. For measurement directives, a URI specifying how the value of a managed resource is going to be retrieved and the value type of the return value should be described. In addition, the *access model* (i.e., *push* or *pull*) must be specified in order to clear out if the party responsible for the measurement will ask for the value or get it when it is ready. Moreover, specific measurement directives may require a possible extra attribute (timeout) specification concerning the time duration that the measurement party will wait for in order to get the measurement value. *Composite metrics* are computed by applying statistical or other mathematical functions to other metrics. So there must be a description of both the function used and of the metrics used to compute the composite metric. Moreover, a function model should be provided in order to enable users to select the appropriate function for each composite metric.

(5) *The Schedule of the metric.* The SQMM should enable the definition of at least one of the following three kinds of time windows for the periodic or instantaneous calculation of new values for a quality metric:
   (a) calendar time window like week, month and/or year;
   (b) sliding windows like the last ten days;
   (c) expanding window or running total like from the beginning of the year until now.

(6) *The weight of the metric* relative to its implicit domain and user preferences. This weight can be used in calculating the rank of a service quality offering and indicates the impact that this metric has to the overall quality offered by a service.

(7) *The characteristic of the function from metric values to overall quality values.* For instance, some quality attributes (and their corresponding metrics) such as

execution time are negatively monotonic, while others like *availability* are positively monotonic. This means that depending on the metric's monotonicity, a higher value will correspond to a better (for positive) or worse (for negative) quality value and this view holds for any type of user. Thus, these QoS attributes and metrics do not need a function that maps their values to overall quality values or levels. On the other hand, there can be metrics, that we call non-monotonic, which do not belong to the above cases and they need such a mapping function. For example, for a metric enumerating the encryption algorithms that can be supported by a service, there can be a user function that maps the value of "AES–192" to service level 3, the value of "AES–256" to service level 2, and the value of "AES–128" to service level 1. So this metric is certainly non-monotonic. Moreover, the highest security value gets a lower quality value than the second highest one indicating that the user may be satisfied with the second highest value and does not want to pay more in order to have a more secured service. Thus, for non-monotonic metrics user-defined functions should be provided that express the preferences of the user regarding the values that these metrics can take.

It should be noted that the above situation is tailored for QBSM scenarios where each quality metric is considered independently of the other. However, in case that there are dependencies between quality metrics and attributes, functions (or n-ary constraints) should be used in order to capture them in conjunction to the aforementioned mapping functions. As far as service selection is concerned, the Simple Additive Weighting technique [Hwang and Yoon 1981] is commonly used, which requires that the values of each attribute or metric are normalized according to a specific evaluation function. In this case, the above mapping functions of non-monotonic metrics can be used provided that they map the metric values to the same range (usually the [0.0,1.0] range). For monotonic metrics, particular evaluation functions are used in the majority of the research approaches that do not have to be captured in an SQMM.

Thus, in order to summarize, an SQMM should explicitly specify the exact monotonicity of monotonic metrics and uniform (according to their range) mapping functions for non-monotonic metrics. This information modeling is sufficient in the majority of the QoS-based service matchmaking and selection scenarios.

(8) *Aggregation of the values of a composite service's metric.* 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 Tc of a complex service C, which is defined as a sequence of two services A and B, can be computed as the sum Ta + Tb of the execution times 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 quality analysis process, a prerequisite for a successful quality-based service discovery.

Similarly to the analysis of the previous criterion, we consider that all SQMMs are

extensible regarding their metric modeling so we evaluate only the richness of their metric models. The evaluation of this criterion for a specific SQMM depends on the meta-model's satisfaction on the eight previously analyzed sub-criteria. It must be noted that if a SQMM does not comply with all the modeling requirements for a specific aspect/sub-criterion, then this SQMM does not satisfy the sub-criterion. So if the considered SQMM satisfies only 1-2 sub-criteria, then it is considered *poor*. If it satisfies 3-4 criteria, it is considered *fair*. If it satisfies 5-6 criteria, it is considered *good*. Otherwise, if it satisfies 7-8 criteria, it is considered *rich*.

The results of this evaluation are presented in the fourth column of Table VIII. As it can be seen, most SQMMs encompass a good metric model, less SQMMs encompass a fair one, and only 3 SQMMs encompass a poor model. In addition, only one approach (OWL-Q) captures a rich metric model, while it does not offer a rich quality model. In this way, we can definitely state that a metric model is better captured in the current state-of-the-art SQMMs than a quality model. Considering, now, the local results in each partition, it is easy to see that SLA-enabled SQMMs capture a good metric model with respect to its richness. This observation was actually justified in the evaluation of the previous criterion, where it was stated that SLA-enabled approaches pay more attention to metric than to attribute modeling. Security-based approaches present a good or fair metric model, while the same result applies to pure SQMMs. Finally, as it can be easily seen, there is a trend of pure SQMMs in improving their metric model as no fair or poor metric model is encompassed in them after 2004. By combining this result with a similar result of the previous criterion we can definitely say that the most recent pure SQMMs have increased their expressiveness as they cater for a better service quality and metric model.

4.2.4 *Expressiveness in Constraint Description.* A service quality offer or requirement (i.e. specification) must be comprised of quality constraints. Each quality constraint consists of a name, an operator, and a value [Frølund and Koistinen 1998]. The name is typically the name of a quality metric, although it can also be the name of a metric *aspect* or *function*. The permissible operators and values depend on the quality metric's value type. The values of a metric value type can be used in constraints for that metric. The domain may be ordered. 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 unordered domains but only the equal "$=$" and not equal operators "$\neq$".

Aspects [Frølund and Koistinen 1998] are simple and complex 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 requirement should contain one complete expression of constraints or just one constraint. Moreover, 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 [Tosic et al. 2003]. *Constraint Groups* contain a set of *concrete* quality constraints (i.e. apart from the quality metric given as a first operand to the constraint operator, also the second operand has a specific value or metric), while *Constraint Group Templates* contain *abstract* quality constraints (i.e. the second operand of the constrain operator is not specified). Other reusability constructs can also be created even for expressions.

For the evaluation of this criterion for a specific SQMM, we have distinguished between the following cases: If the SQMM does not satisfy any of the above requirements, then its expressiveness will be *poor*. If the SQMM allows comparison operators in constraints but does not check if they are compatible with the metrics used, then it is considered *fair*. If the SQMM allows comparison operators and it checks their compatibility with constraints, then it is considered *good*. If the SQMM is *good* and it also allows not only "and" but also other constraint expressions and formations, it is considered *rich*. Finally, if the SQMM is *rich* and also allows the specification of aspects, then it is considered as *excellent*.

The results of the evaluation of this criterion are presented in the fifth column of Table VIII. As can be seen, most of SQMMs use a rich constraint model. Another general observation is that there are no fair constraint models, which indicates that SQMMs either perform above or below the average in this aspect. However, the most distinguishing fact is that there are some excellent constraint models captured only in half of the SLA-enabled SQMMs. In fact, in this SQMM partition, all constraint models are either rich or excellent. This is quite reasonable as rich constraint representation, especially for service quality levels, is one of the cornerstone features of SLA languages, which is equally important for all contracting parties of a SLA. Concerning security-based SQMMs, they mostly use a good constraint model. This is quite reasonable as, according to their domain, these approaches do not require advanced constraint modeling features. In pure SQMMs rich constraint models are the majority. Moreover, it is easy to see that pure SQMM modelers have quickly understood the importance of this feature as their SQMMs moved from a poor to a better constraint model (apart from the last approach which we consider as an outlier). By combining this result with the similar ones from the previous two criteria, it can be definitely stated that pure SQMMs have increased their overall expressiveness over time.

4.2.5 *Complexity.* On one hand, the *formalism* of a meta-model characterizes the explicitness in which the semantics of the meta-model's terms are expressed, while its *expressiveness or richness*, according to our consideration, concerns how well the domain of discourse (i.e. service quality) is modeled. On the other hand, the complexity of a metamodel mainly concerns its size and structure and has a significant impact on its understandability by the users and on the modeling effort involved, the size, and the amount of information included in the produced descriptions. Obviously, the better the domain is modeled and the more details are captured, the more complex the meta-model becomes. So, usually there is a

trade-off between complexity and our notion of richness that mainly depends on how accurately and extensively the meta-model expresses its domain, on the existence of formal and semantic inconsistencies in it, and on the relevance and appropriateness of the modeled information. In other words, it is the quality of the meta-model that regulates this trade-off. Thus, the object of research is to develop rich and qualitative meta-models such that their complexity does not become very big.

Unfortunately, it is very difficult to assess the quality of a meta-model and very few approaches have been proposed that focus on some aspect of it [Jiang et al. 2004; Welty et al. 2003]. On the contrary, many metrics have been proposed for measuring the complexity of meta-models in general or of ontologies in particular [Mens and Lanza 2002; Yi et al. 2004; Yoa et al. 2005; Yang et al. 2006]. In this way, we have deliberately chosen to assess the complexity of the SQMMs by selecting one or more appropriate meta-model complexity metrics such that in combination with their richness to be able to speculate about their quality.

Among the various meta-model complexity metrics, we have chosen the one that measures the number of concepts of the meta-model for the following reasons: a) many SQMMs are not publicly available, so we cannot process them and evaluate sophisticated complexity metrics on them and thus, we have to rely on their presentation in papers and technical reports; b) many complexity metrics are not applicable to informal SQMMs; c) the particularities in the structure of the informal SQMMs lead us to use a simple and fair metric that does not depend on the meta-model's structure and it is very easy to compute. However, the use of such a simplistic metric prevents us from performing reasonable speculations about the quality of the SQMMs, as the threshold on the number of sufficient concepts for modeling a domain depends on the subjective view of domain modelers.

Based on the above analysis, we have used particular thresholds on the number of concepts involved in each SQMM in order to categorize them according to their complexity. So, if an SQMM has less than twenty-five concepts (25), it is evaluated of having *low* complexity. Otherwise, if it has from twenty-six to fifty concepts (26-50), then the SQMM is evaluated of having *medium* complexity. Finally, if the number of concepts in an SQMM is more than fifty (50), then the SQMM is evaluated of having *big* complexity.

The results of the evaluation of this criterion are presented in the sixth column of Table VIII. From these results, it is easily identifiable that most SQMMs have low complexity. Moreover, only six SQMMs have medium complexity and only one high (OWL-Q) which happens to be the most expressive SQMM according to the evaluation of the previous three criteria. In this way, we may come to the conclusion that modelers have realized that their SQMMs should have lower complexity in order to be understandable and be adopted more easily. However, this conclusion is not safe as the expressiveness of the majority of the SQMMs is not adequate. Concerning the local partitions, the results remain almost the same. By carefully inspecting the results, the trend that pure and SLA-enabled SQMMs of higher complexity are proposed lately can be revealed. This actually means that modelers are trying to increase the expressiveness of their SQMMs and, in result, the complexity of their SQMMs increases. Indeed, based on the analysis of the previous criterion, pure SQMMs have increased their expressiveness over the

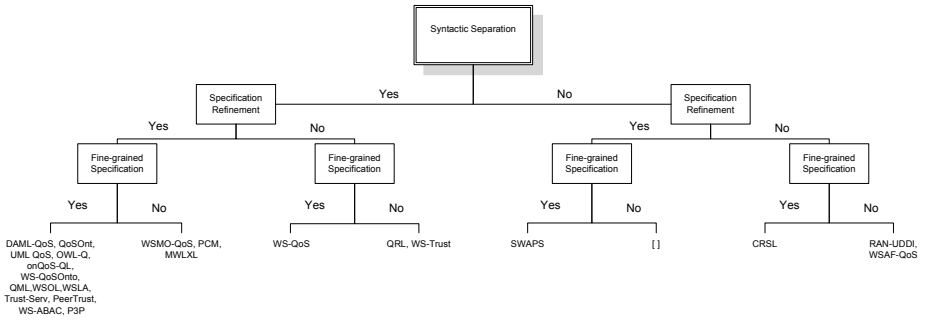

Fig. 7. Evaluation of SQMM approaches for the 4.6 group of criteria

years. As far as SLA-enabled SQMMs are concerned, they have also increased their expressiveness but not in quality-related aspects but SLA-based ones. This latter fact will be shown in Section 5. Security-based SQMMs have only low complexity as they are deliberately designed in this way, i.e. to produce short descriptions that are easily exchanged, processed, and matched by various entities over the Web.

4.2.6 *Service Description Separation, Refinement, and Granularity.* In this subsection, a group of three related criteria are presented and analyzed. The evaluation results are presented for the whole group in the tree-like structure of Fig. 7.

4.2.6.1 *Syntactical Separation of Quality-Based and Functional Parts of Service Description.* Service quality specifications should be syntactically separated from other parts of service specifications, such as interface definitions. This separation allows us to specify different quality properties for different implementations of the same interface. Moreover, while functional constraints rarely change during runtime, service quality constraints can change during runtime. So the separation of service quality offerings from WSDL descriptions permits service offerings to be deactivated, reactivated, created, or deleted dynamically without any modification of the underlying WSDL file. Finally, a service quality offer could be referenced from multiple WSDL files and thus be reused for different services. Thus, the evaluation of this criterion for a specific SQMM is: *yes* (for syntactical separation) and *no*.

4.2.6.2 *Support for the Refinement of Quality-Based Service Descriptions and their Constructs.* As previously described, syntactical separation provides reusability. But except from reusability, another form of extensibility is equally important. Service quality specifications should not only be reused but also refined. This means that we can create 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. In addition, templates of service quality offerings should be created and appropriately extended for every domain. So the evaluation of this criterion for a specific SQMM is: *yes* (supports refinement) and *no*.

4.2.6.3 *Support for Fine-Grained Quality-Based Service Description.* It should be possible to specify quality properties/metrics at a fine-grained level. As an exam-

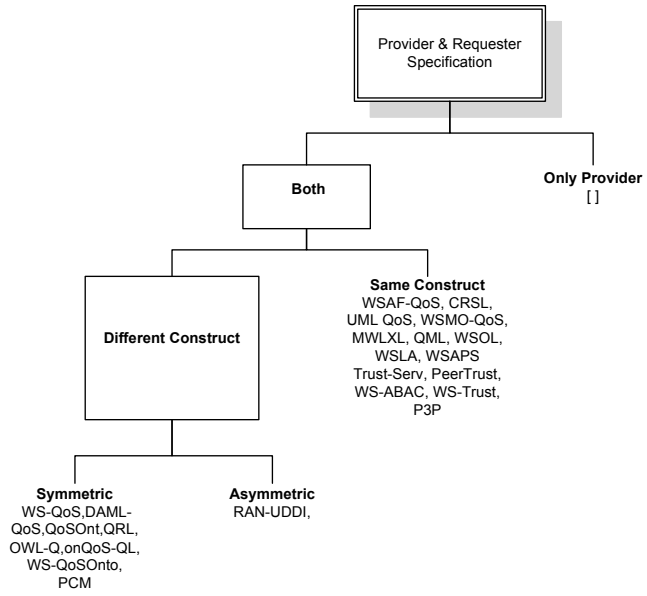

Fig. 8. Evaluation of SQMM approaches for the 4.7 criterion

ple, performance characteristics are commonly specified for individual operations. A SQMM must allow quality specifications for interfaces, operations, attributes, operation parameters, and operation results. So the evaluation of this criterion for a specific SQMM is: *yes* (it allows), and *no*.

As can be seen from Fig. 7, the most populated node is the one which corresponds to those SQMMs that enable all three features. So, most of the modelers have understood the importance of these three features for their SQMM. Moreover, the same result holds for all the partitions. Another observation is that the mostly supported feature among all SQMM approaches is syntactical separation, followed by specification refinement and then by fine-grained specification. The same order holds for the pure SQMM partition. For security-based SQMMs, the order remains almost the same, with the slight difference that fine-grained specification has the same importance as specification refinement. Finally, for SLA-enabled SQMMs the order is different. Specification refinement is the most important feature, while the other two are of equal importance. This result is quite reasonable as one of the highest design goals of SLA-enabled SQMMs is to support SLA templates and other re-usability constructs in order to enable their re-use and refinement by SPs and SRs.

4.2.7 *Support for Symmetric but Separate Quality-Based Service Description for Providers and Requesters.* Both the quality properties that clients require and the quality properties that services provide must be specified. Moreover, these two aspects should be specified separately so that a client-server relationship has two service quality specifications: a specification that captures the client's requirements (i.e service quality requirement) and a specification that captures service

provisioning (i.e service quality offer). 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.

Service quality requirements and offers should be specified in the same expressive way, i.e. symmetrically. Assuming that S is a multidimensional space whose dimensions are given by domains of quality parameters, then both a service quality offer and requirement should be expressed as a subspace in S. Traditionally, service quality offers have been described as points in S, i.e., asymmetrically. However, 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. Moreover, the probability that an offer is matched with a requirement is quite low. On the other hand, the probability of matching is higher when both types of QSDs are expressed as subspaces, while also more advanced protocols are enabled in service negotiation. In addition, symmetric approaches usually achieve a greater deal of expressiveness to specify QoS, 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.

To sum up, with this sub-criterion it is checked if both SPs and SRs can provide service quality specifications, if these specifications are defined separately with different constructs, and if these different specifications are allowed to have the same expressiveness. The evaluation of this criterion for a specific SQMM has the following values: a) *only provider*, b) *both with same construct*, c) *both with different constructs and asymmetrically*, and d) *both with different constructs and symmetrically*. The first and last values are the worst and best, respectively, while there is no definite order between the second and third value as each violates a different requirement.

As complex values are used in evaluating SQMMs for this criterion, a tree-based structure (see Fig. 8) was used to present the comparison results. The most populated node of the tree is the one representing the case where the SQMM allows both SPs and SRs to specify service quality specifications with the same construct, followed by the node representing the case where both SPs and SRs are allowed to specify service quality specifications with different constructs but symmetrically. This means that researchers have realized that both SPs and SRs should be allowed to specify their QSDs in exactly the same and expressive way. However, they still have not realized that Service Quality Offers and Requirements should be specified separately with different constructs. Concerning, now, the SQMM partitions, all SLA-enabled and security-based SQMMs use the same construct for expressing the two types of QSDs. For SLA-enabled SQMMs, this is a rational choice as the produced SLAs express both the views of the SP and SR. However, for security-based SQMMs, this is not a rational choice for privacy, as there must be a clear distinction between who is requesting privacy requirements and who is offering to satisfy these requirements, but it is rational for trust, as the specification of requirements and offerings is performed in a bilateral way. As far as pure QSMMs are concerned, the majority of the approaches express the two different types of QSDs in a separate

and symmetric way. Thus, the pure SQMM modelers have realized the significance of this design choice.

4.2.8 *Support for Classes of Service Description. Class of Service* means the discrete variation of the complete service and quality provided by one service. In our opinion a *Class of Service* has the same meaning as the *alternatives* have in WS-Agreement. In other words, a Class of Service is actually the set (of functional and) non-functional guarantees that are to be provided by a service in terms of a SLA. So, with this criterion we want to record if the language/meta-model can capture only one class of service specification or many. In this way, the evaluation of this sub-criterion for a specific meta-model is: *one* (class of service), and *many*.

The results of the evaluation of this criterion are presented in the seventh column of Table VIII. As can be seen, most of the SQMMs can represent many classes of service and this result concerns also each SQMM partition separately. So SQMM modelers have understood the advantage of allowing many classes of service specification for one service. Another significant observation is that for pure and SLA-enabled SQMMs this understanding is more anticipated in the most recent approaches, as all of them offer this important feature.

4.2.9 *Connection to Service Functional Specification Languages.* With this criterion, we want to check if the SQSL is connected to or references a Service Functional Specification Language (SFSL) like WSDL, WSMO, or OWL-S. On one hand, if the answer is positive for an SQSL, this means that there would be no effort required to extend this language (and probably all of its QSDs) in order to be used in a registry that contains functional SDs obeying to the connected SFSL. In result, there could be an increase in the SQSL's adoption. On the other hand, if the answer is negative, then the SQSL can be extended in order to be connected to any SFSL and not only to a specific one. In this way, it can be practically used with any registry that is bound to a specific SFSL. So the evaluation of this criterion for a specific SQMM contains either the name of a SFSL or the answer *no*.

The results of the evaluation of this criterion are presented in the eighth column of Table VIII. As can be seen, WSDL is the most referenced SFSL, followed by OWL-S (ontology-based SFSL), while another ontology-based language, namely WSMO, is referenced by only two SQSLs. Another observation is that four SQSLs do not reference any SFSL, being in this way SFSL-independent. Taking specific SQSL partitions into consideration, OWL-S (or ontology-based SFSLs more generally) is the most referenced SFSL in pure SQSLs for two main reasons: 1) ontological approaches are greater in population with respect to the rest of the approaches in this partition, and 2) the use of semantics has been proven to increase the accuracy in FSD, so pure SQMM modelers prefer a ontology-based SFSL for this reason. On the other hand, WSDL is the most referenced SFSL in the rest of the SQSL partitions. This result is expected because SLA-enabled and security-based SQSLs do not consider the FSD scenario when they are designed, as they regard it as an orthogonal issue, so they prefer to stay on the most popular and used SFSL (i.e., WSDL) so as to increase their adoption. Moreover, there is a trend, where the most recent SLA-enabled SQMMs are being connected to WSDL.

4.2.10 *Support for Quality Matching.* People may have different conceptualizations of the same domain, so it is possible that in different QSDs of the same SQMM the same concepts are expressed differently or different concepts are expressed in the same way. Concerning the QoS domain, this can be true for QoS metrics and QoS attributes but not for measurement units that are more or less standardized. This argument is also strengthened by the fact that the same QoS metric or attribute can be considered either composite or atomic in different SQMs depending on the level-of-detail required or the types of measurements supported. For instance, a service's *availability* is measured from high-level readings in one system's instrumentation, while it is measured from low-level readings (e.g from service's *uptime*) in another system's instrumentation. In the former case, the service's availability metric is resource/atomic, while in the latter case, the service's availability metric is composite.

As the basic part of QSDs is the one where SPs advertise their capabilities and SRs provide their requirements as a set of constraints on specific QoS metrics or attributes, these sets of QoS concepts must be matched in order to increase the accuracy of the QBSM process. For this reason, semantic QoS metric/attribute matching rules must be developed and used internally or externally to an SQMM in order to enrich and align the produced QSDs. Depending on the formalism used to expressed an SQMM, rules may be part of an SQMM's specification or may be developed externally in a possibly different format. In the latter case, it could be argued that these external rules are not actually part of the SQMM's specification and, thus, they should not be a criterion for comparing SQMMs. However, we believe that this information, either being internal or external, should be additionally modeled as it will be used to support service discovery and would increase an SQMM's adoption and significance. Thus, the evaluation of this criterion for each SQMM will be: *yes*, if the corresponding SQMM contains or is accompanied with quality matching rules and *no* otherwise.

The results of the evaluation of this criterion are presented in the ninth column of Table VIII. By inspecting these results, it is easily observable that most of the SQMM approaches have not considered this modeling aspect at all. Moreover, this is true not only globally but also locally in every partition. Another observation is that not all security-based SQMMs use or model these matching rules. This is an expected result as the quality attributes in this area are more or less standardized, while quality metrics are used less often. Thus, the modeling of matching rules is not actually required. A final observation is that the most recent pure and SLA-enabled approaches have understood the need of modeling this feature and have produced the required matching rules.

4.2.11 *Framework Support.* This criterion checks if any service discovery and negotiation framework has adopted the SQMM under inspection. If the answer is positive, then this means that the SQMM has been used in practice in one of the service life-cycle activities (apart from the first and obvious one). So this criterion evaluates the adoption and usage of every SQMM. In addition, it evaluates if SQMMs have been used in service negotiation apart from service discovery. The evaluation of a specific SQMM for this criterion is: *no* (support), *only discovery*, *only negotiation*, and *both* (discovery and negotiation). It must be noted that

all SLA-enabled SQMMs already have an underlying SLA enforcement framework implementation, so we chose not to explicitly represent this fact in this evaluation.

The results of the evaluation of this criterion are presented in the last column of Table VIII. As can be seen, the majority of the SQSLs have been used in service discovery implementations. In fact, this result holds not only globally but also in each partition. In addition, in conjunction with the results of the previous criterion, all the SQMM approaches that model matching rules have been used in service discovery implementations, thus realizing the need for increasing the accuracy of the service discovery activity. However, none of the implementing discovery frameworks could also perform service negotiation. Moreover, there is no service negotiation framework supporting any of the SQSLs. It must be noted here that there are negotiation frameworks supporting WS-Agreement but not SWAPS (which is a WS-Agreement extension).

Another observation that should be made is that most of the SLA-enabled SQSLs apart from WSLA are not used in any implemented negotiation framework. This actually reduces the dynamics and further spread of these languages. In addition, by considering the service life-cycle analyzed in Section 2, it should be discussed that only the SLA-enabled SQMMs have been used in service negotiation. This is rational as the QSDs of this type of SQMMs that are used in this activity are SLAs or SLA templates. Thus, it can be definitely said that the service negotiation activity is the stopping point of usage of the QSDs and the corresponding starting point of SLAs.

4.3 Overall Analysis

Based on the above analysis, there is not any perfect approach that has taken the best value in all criteria. From all the SQSLs, OWL-Q can be distinguished as the best among the good approaches which are found mainly in the pure SQMM partition. However, it must be extended in order to enable the specification of rich SQMs. In addition, a balance between expressiveness and complexity should be accomplished in this language by defining which are the obligatory and non-obligatory concepts so as to help modelers in defining their QSDs with less effort and more speed. Thus, the design of this language should be revised accordingly.

By considering the rest of the SQSL partitions, SWAPS is the best among all SLA-enabled SQSLs but it must significantly improve its attribute and metric model. This is also true for PeerTrust, the best among all security-based SQSLs. So, it can be easily seen that the SLA-enabled and security-based SQMMs need further improvement and extension in their attribute and metric models and far more changes with respect to those needed for the pure SQMM approaches.

By closely inspecting the results of the last criterion, pure and security-based SQMMs are used until the service discovery activity and they seem not to be exploited further in the service life-cycle. This can be justified by their inability to model SLA specific constructs which are considered more useful for supporting the rest of the service life-cycle activities. This should be the reason why SLA languages have been proposed, i.e., to fill the gap and play the significant role of mostly supporting those service activities that are beyond service discovery. The next section explains this role and provides an analysis of the capabilities of the SLA languages with respect to their support to the service life-cycle activities.

5. SERVICE LEVEL AGREEMENTS

5.1 Background

5.1.1 *Contracts and SLAs.* As the global economy is changing at a fast pace and the competition is rising due to technology advancements, organizations have to enter in dynamic business relationships with other organizations, whose result would be one or more cross-organizational processes. The basis for the establishment of such dynamic relationships are electronic *contracts*, which are legally binding digital agreements that safeguard the concerns of each participating organization [Hoffner et al. 2001], as they assist in speeding up and automating the various activities that support these relationships, which include the establishment of the contractual relationships and the set-up of the enactment infrastructure. While there can be various types of contracts depending on the type of value that is exchanged between two or more organizations [Grefen and Angelov 2002], from now on we only deal with *service contracts*, as the focus of this paper is on services. A service contract includes information, such as [Hoffner et al. 2001; Oren et al. 2005]: a) *parties* involved in the agreement, b) the *service*, including interface description and expected interactions, c) description of *norms* (*obligations*, *prohibitions*, *permissions*, etc.) imposed on each party concerning service provision and consumption, d) timing and conditions of contract termination. Apart from technical information, some service contracts may also contain legal procedures in case of breach of contract and arbitration.

In the past, most of the service contract languages were focusing on describing the functionality of a service and on automated contract execution monitoring [Oren et al. 2005], i.e., determining the state a contract is in, and which contract rules are in effect given this state. In other words, they were focusing more on how the service behaves while they were more or less neglecting the QoS aspects of service behavior. However, as the number of services providing the same functionality is increasing, QoS can be used in order to distinguish between them. As QoS can be equally important to functionality, there is now a focus on more specialized service contracts, which are called SLAs. SLAs describe how well a service performs its functions [Tosic and Pagurek 2005]. They contain quality guarantees that have to be respected during service execution and other important terms indicating the actions to be taken when these guarantees are violated. Thus, they increase the trust and consolidate the overall relationship between an SP and an SR, as the service will either meet the stated requirements or there will be consequences that will tend to compensate the client for the harm it suffers due to these requirements being missed [Skene 2007]. The last conclusion is very important as it signifies that both service contracts and SLAs are the basis for the establishment and support of business relationships.

5.1.2 *SLA Demystification.* Many definitions of the term SLA can be found in the literature. However, we rely on the definition given by Keller and Ludwig [Keller and Ludwig 2003], which states that an SLA is an important aspect of a contract for IT services that includes the set of Quality of Service (QoS) guarantees and the obligations of the various parties. QoS guarantees are widely known as Service Level Objectives (SLOs) and are usually expressed as conditions on one or more QoS metrics, indicating in this way the metrics allowed values. A set of SLOs

constitutes a specific Service Level (SL). There can be different SLs defined in an SLA, expressing the different modes a service may execute in different time periods of the day or of the week, or degradation/upgrade levels if the agreed SL is violated/surpassed.

According to [Paschke and Schnappinger-Gerull 2006], SLA documents contain *technical* (e.g. metrics, actions), *organizational* (monitoring and reporting), and *legal* components (legal responsibilities, modes of invoicing and payment). From these components, the legal ones are hard to be automated and enforced by a management system, so they are either omitted or neglected. The most common components of SLA languages are the following [Paschke and Schnappinger-Gerull 2006]: *involved parties*, *contract validity period*, *service definitions*, *SLOs*, and *action guarantees*. *Involved parties* are roles referenced inside a contract. They can be distinguished between [Keller and Ludwig 2003] *signatory parties*, which are usually played by the service provider (SP) and requester (SR) that sign the contract, and *supporting parties*, which have the role to support the monitoring and assessment of the SLA. These two types of roles are not mutually exclusive, as, for instance, an SP can provide measurements for the execution time of his provided service. The *contract validity period* specifies for how long the SLA will be valid and enforceable. This field is very important for *continuous* services as it determines for how long the SP should provide his service to the requester according to the directives of the agreed SLA. The *service definitions* section of the SLA specifies the characteristics of the service and its components and its observable parameters. In other words, this section is responsible of describing what is the functionality to be delivered by the service, what are its components (i.e., operations, input, output, internal and external services for a composite service), and what are the QoS metrics to be observed (e.g., *response time*, *availability*, etc.) for both its service and its components. As already explained above, *SLOs* are QoS guarantees. They should be met by a specific obliged party (e.g., SP) and have validity periods [Keller and Ludwig 2003], while they can also have qualifying conditions on external factors such as time of the day (i.e., when the SLO should be evaluated) as well as the conditions that a client must meet (e.g., a client's request rate is above a threshold). Finally, *action guarantees* [Keller and Ludwig 2003] express a commitment that a particular activity is performed by an obliged party if a given precondition is met (e.g. a violation occurs). The committing activities include compensation, reward, recovery, and management actions.

Besides the common SLA components, two additional parts should be contained in an SLA [Muller 1999]: *limitations* and *renegotiation*. The former describes the limits of IT support during conditions of peak period demand, resource contention by other applications, and general overall application workload intensities. These limitations should be agreed to by all parties in order to prevent finger pointing and user dissatisfaction. The latter describes how and under what circumstances the SLA must be changed through renegotiation so as to reflect changes in the (user, service or environmental) context.

Before SLAs are established, i.e., after successful service negotiation, they are in a form which is called *SLA template*[1]. These SLA templates are used for describing,

[1]The term *Contract Template* is used for service contracts

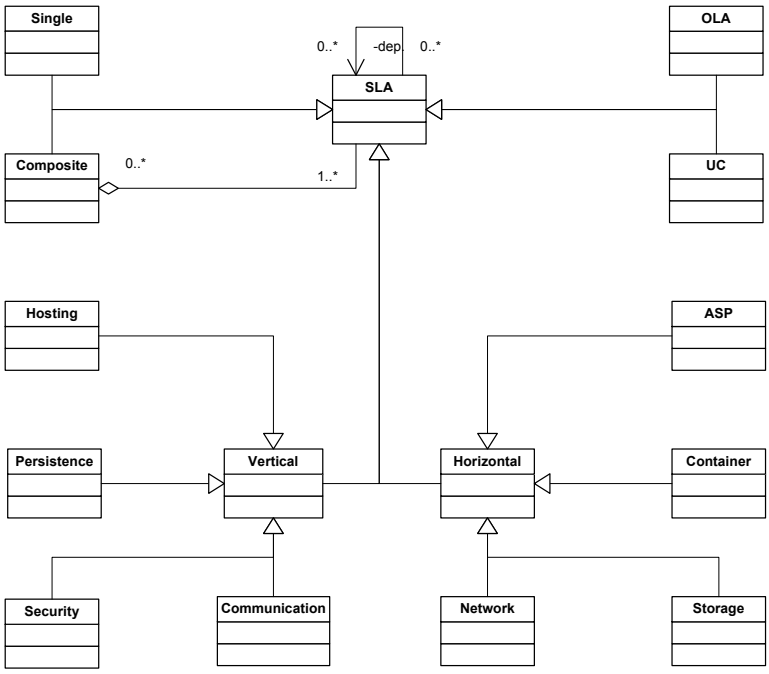

Fig. 9. SLA Categorization.

matchmaking, and negotiating the SLs that will be offered by a service of an SP to an SR. Thus, they are produced by both SPs and SRs. SLA templates can be complete or incomplete SLAs. In the first case, they represent SLAs which are commonly agreed among all participants in a restricted domain or are used as bilateral agreements between two organizations or as SLA offerings advertised by an SP to specific customer classes. In this case, the SLA template is offered in a "take it or leave" it basis [Hoffner et al. 2001]. In the second case, SLA templates can be seen as a skeleton with fields which must be completed according to the directives of the desired relationship between two organizations [Hoffner et al. 2001]. So, they are generic forms or templates that can be tailored to the specific circumstances of an SLA instance. In this case, according to the *granularity of choice* and *specialization* [Hoffner et al. 2001], SLA templates may: a) be *monolithic* where values are inserted in predefined positions; b) have certain sections which can be included or removed; c) are clause-based [Fosbrook and Laing 1996].

There can be different types of SLAs according to their composability, intended purpose [Paschke and Schnappinger-Gerull 2006], and the service usage based on the reference functional architecture model [Lamanna et al. 2003]. Figure 9 shows the three dimensions of SLA categorization along with their corresponding types. It should be stressed that these dimensions are orthogonal to each other and their types are mutually exclusive, so one SLA may belong to only one type of each dimension.

According to their composability, SLAs are distinguished between *Single* and

Composite. Single SLAs specify the SLs of one service, independently of the type of the service (i.e., single or composite), and are agreed between only two parties, the SP and SR. On the other hand, Composite SLAs specify the SLs of both the service, which should be provided to an SR by the SP, and some of its component or supporting services that are provided by third party SPs to the main SP. Thus, a Composite SLA may consist or depend on other SLAs. The latter fact does not exclude the possibility that a Single SLA may depend on one or more other SLAs. However, this dependency information is not included in the description of the Single SLA.

Based on their intended purpose, SLAs can be distinguished between *Operation Level Agreements (OLAs)*, and *Underpinning Contracts* (UCs). OLAs are usually informal SLAs with internal operational partners, while UCs are SLAs with external operational partners. Both of these SLA types specify the service that the operational partners are going to deliver to the SP. This service will be used in order to support the service specified in another SLA that the SP is promising to deliver to a prospective SR.

Based on the functional architecture model introduced in Section 1, there can be many different types of SLAs [Lamanna et al. 2003], corresponding to the different service usages present in the model. These types are divided into *Vertical* or *Infrastructural* SLAs, in which the service provides technical support to the client, and *Horizontal SLAs*, in which the client sub-contracts part of the functionality of his service to another service of the same type. Vertical SLAs, according to the granularity of the Infrastructure Layer, can be divided into *Hosting* (between an SP and a host), *Persistence* (between a host and a storage provider), *Communication* SLAs (between application, service, or host and an Internet service provider), and *Security* SLAs (between application, service, or host and a security provider). In other words, Vertical SLAs concern services that are either offered from the Infrastructure Layer to its layers above it or from a functional level of the Infrastructure Layer to a level above it in the same layer. Horizontal SLAs are divided into *ASP* (Application Service Provisioning) (between applications or services and other applications or services), *Container* (between container providers), *Network* SLAs (between network providers), and *Storage* SLAs (between storage providers). For this type of SLAs the granularity is very coarse-grained for the first two layers and then fine-grained for the Infrastructure Layer.

5.2 Methodology and Analysis

As an SLA is a document, it has a life-cycle that starts with its creation and ends with its disposal or archiving and it includes all the appropriate activities needed for the SLA's management. This life-cycle is tightly coupled with the service life-cycle introduced in Section 2. In fact, the same also holds for the contract life-cycle. The reason for this tight coupling is obvious. SLAs and service contracts in general, exist as long as the service they describe exists because this service is the reason for the establishment and very existence of the business relationship between the SP and SR. Indeed, all of the service contract and SLA management systems we are aware of actually support, directly or indirectly, the management of the service that is offered by the SP to the SR. Thus, through the support of the SLA/contract life-cycle, the service life-cycle is also supported and especially those activities that

correspond to service provisioning.

Various SLA life-cycles have been proposed in the literature, which differ on which activities they involve, on the level of granularity of these activities and on their terminology. However, none of them suits our needs as they are either coarse-grained or tend to neglect some activities. For this reason, we have devised an extended SLA life-cycle based on the research works of [Keller and Ludwig 2003; Parkin et al. 2008], which is depicted in Figure 10. In the following, using this life-cycle as a basis, we explain what are the life-cycle activities involved in the management of an SLA and how they relate to those of the service life-cycle:

—*Template Development*: A template of the SLA is developed by the SP or the SR according to the service quality capabilities and requirements, respectively. This activity is rationally executed after the service is implemented and tested but can happen before, during or after the service's deployment.

—*Advertisement*: After the SLA template is developed by the SP, it is advertised in intra- or inter-organizational repositories in order to be discovered by potential SRs. This activity happens after the service is deployed and maybe executed several times. It is a joint activity of the service and SLA life-cycles.

—*Matchmaking*: An SR makes a request represented by an SLA template to a broker or discovery service so as to find the SLA templates of those services that satisfy his quality requirements. This activity is after or in parallel with the functional service discovery activity. We can even say that this is a joint activity of the two considered life-cycles, as SLAs can represent all information needed for the functional and quality-based service discovery. As a result of this activity, the user's SLA template may change, if it is over-constrained, to reflect realistic performance situations in the respective application domain.

—*Negotiation*: This is certainly a joint activity of the two considered life-cycles. The SP of the best service (or all the SPs of the matched services) negotiates with the SR according to the content of their SLA templates. These SLA templates may change during the negotiation to reflect the compromises performed by the two parties during the negotiation.

—*Agreement & Deployment*: As the outcome of a service negotiation is not always successful, we separate this activity from the previous one. If the outcome is successful (agreement is reached), then a specific SLA is produced and signed by the two corresponding parties. This SLA has to be validated first and then deployed. SLA deployment is performed at two steps: a) the deployment system of a signatory party extracts from the SLA the appropriate configuration information and distributes it to the corresponding supporting parties so as to inform them about their roles and duties; b) deployment systems of supporting parties configure their own implementations in a suitable way. It should be noted here that all parties need to know the definitions of the interfaces they must expose, as well as the interfaces of the partners they interact with. Moreover, components of different parties cannot be assumed to be configurable in the same way, i.e., they may have heterogeneous configuration interfaces. This composite activity is usually performed before the service is executed. In case where no active instance of the service can execute in the corresponding SL of the SLA, then a new instance of the service must be deployed or more resources are given to a

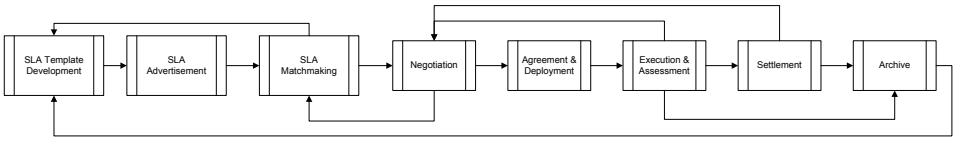

Fig. 10. The SLA life-cycle.

specific instance. In this case, we can have a service deployment activity running in parallel with the corresponding SLA activity.

—*Monitoring & Assessment*: While the service is executing, the SLA is also executed. The latter means that the service is periodically monitored and the agreed SLOs of the SLA are assessed. Monitoring is performed by the measurement components of the supporting parties run-time systems which maintain information on the current system configuration, as well as run-time information on the metrics that are part of the SLA. These components measure QoS metrics either from inside, by retrieving resource metrics directly from managed resources, or outside the SPs domain, e.g., by probing or intercepting client invocations. The condition evaluation components of supporting parties run-time systems compare the measurement information against the SLOs and notify the management systems (those of the signatory parties and an external one for the SLA (if it exists)) about violations. If the violation or number of violations is very critical, then the SLA is re-negotiated or is canceled (i.e, archived) according to the SLA's corresponding action guarantee. If not, appropriate corrective actions are taken by the management system of the SP (e.g. more resources are provided to the under-performing service). The choice of which corrective action to perform depends on many situational factors and on the business goals and policies of the SP. As it can be seen, this SLA management activity runs in parallel with the execution, monitoring, and recovery activities of the service life-cycle.

—*Settlement*: In this activity, it is determined if the SLA was met, what is the final cost to paid by the customer, and what penalties may apply to the provider for breaching the terms of the SLA. Negotiations for terminating an SLA may be carried out between the parties, in the same way as the SLA establishment is being done, or for re-executing it in a different SL and cost. This activity occurs after the termination of service execution.

—*Archive*: After the SLA settlement takes place, the SLA may be disposed or archived according to the signatory parties policies. However, even if the SLA is decided to be discarded, there is usually a statutory period (known as the "limitation period") where the SLA record must be kept because it is a legal document describing how services were provided. This activity is very important if it is accompanied with an audit trail mechanism because it can be used for identifying problems and patterns of wrong service behavior or trends in user requirements so as to improve the content of future developed SLA templates and even evolve the service's implementation towards correcting the identified problems and meeting the increased customer needs. Thus, this activity can provide significant input to the evolution activity of the service life-cycle.

The above SLA life-cycle is very similar to the ones that have been proposed for service contracts, e.g. in [Hoffner et al. 2001]. The only difference is that the contract life-cycle is more coarse-grained and more general as it also takes into account the functional aspects of service provisioning apart from the QoS ones. However, as our focus is on service quality and its description, we restrain ourselves to the SLA life-cycle, i.e., to the QoS aspects of service provisioning. Thus, for this reason, the analysis of service contract and SLA languages in this survey is based on a set of comparison criteria which are grouped along the SLA life-cycle activities. These criteria are used in order to compare these languages along the lines of the information they can describe which is necessary or appropriate for supporting the SLA life-cycle activities. In this way, as was explained above, by supporting the SLA life-cycle activities, the service life-cycle is also supported.

The summary of our selected criteria is shown in Table IX, while their complete presentation is provided later on in this section. In this table, we have used the term "Description" to cover the first two SLA life-cycle activities, i.e. the "SLA Template Development" and the "SLA Advertisement", as they are actually associated with the description of an SLA. Moreover, it should be explained that we have not analyzed the "Agreement & Deployment" activity by using any criterion, as this activity does not need any specific information which is not already covered by existing SLA languages. Finally, it must be stressed that, as it was explained in Section 4, very few SLA languages define quality and most of them regard that quality should be defined outside the SLA specification by different formalisms and languages and only referenced inside this specification. For this reason and in order not to repeat information, this section neglects from its analysis the quality description capabilities of the SLA languages.

Service contracts and SLAs are expressed by their respective languages which are shown in Table X. While there is no standard service contract language (SCL), there are two SLA languages which are widely used, namely WSLA [Keller and Ludwig 2003] and WS-Agreement [WS-AGREEMENT 2003], and can be considered as standards. It must be stressed that we have chosen the most representative SCLs because, as was explained above, the majority of these languages focuses more on the functional aspects of service behavior. The results of the evaluation of the examined languages according to our selected criteria are presented in Tables XI, XII, and XIII.

In the remaining part of this section, we are going to present each activity-based group of criteria along with their evaluation results in separate sequential subsections. In the end, there will be an overall analysis of the service contract and SLA languages across all criteria.

5.2.1 *Description.* Every SLA and service contract language must possess some properties that enable it to be a good candidate for representing SLAs. We have gathered a set of such four properties/criteria and we analyze them in the next four sub-subsections.

5.2.1.1 *Formalism.* Similarly to SQSLs, each SLA language adopts a specific formalism for expressing its meta-model. These formalisms include: *informal* (such as DTDs or XML Schemas), *UML*, *RuleML*, *Finite State Machines* (FSMs), vari-

Table IX. Summary of the Comparison Criteria of the SLA and Service Contract Languages

Life-cycle Activity	**Criteria**	**Summary**
Description	Formalism	The language's description formalism
	Coverage	The ability to express functional and quality terms
	Reusability	The ability to represent SLA templates and other reusability constructs
	Composability	The ability to represent composite SLAs
Matchmaking	Metric Definition	The ability to define quality metrics
	Alternatives	The ability to express alternative SLs
	Soft Constraints	The ability to express *soft* SLOs that may be violated
	Matchmaking Metric	Definition of the way SLAs produced by a specific language must be compared
Negotiation	Meta-Negotiation	The ability to represent any information to be used for negotiation establishment
	Negotiability	The ability to define which parts of its specifications are negotiable and in what way
Monitoring	Metric Provider	The ability to define the party responsible for producing a metric's measurements
	Metric Schedule	The ability to define the frequency of the production of a metric's measurements
Assessment	Condition Evaluator	The ability to define the party responsible for evaluating an SLO
	Qualifying Condition	The ability to define conditions that must hold in order to assess an SLO
	Obliged	The ability to express the party in charge of delivering what is promised in an SLO
	Assessment Schedule	The ability to express the frequency of the assessment of an SLO
	Validity Period	The ability to express the time period in which the SLO is guaranteed
	Recovery Actions	The ability to express corrective actions to be carried out when an SLO is violated
Settlement	Penalties	The ability to express penalties incurred when one party violates its promises
	Rewards	The ability to express rewards incurred when one party overwhelms its promise
	Settlement Actions	The ability to express actions concerning the final outcome of an SLA
Archive	Validity Period	The ability to express the period where an SLA is valid

ID	Approach Name	Approach Reference	Type
1	QML	[Frølund and Koistinen 1998]	SLA
2	WSML	[Sahai et al. 2002]	SLA
3	WSLA	[Keller and Ludwig 2003]	SLA
4	WS-A	[WS-AGREEMENT 2003]	SLA
5	SLAng	[Lamanna et al. 2003]	SLA
6	WSOL	[Tosic et al. 2003]	SLA
7	RBSLA	[Paschke 2005]	SLA
8	QoWL	[Brandic et al. 2006]	SLA
9	GXLA	[Tebbani and Aib 2006]	SLA
10	TrustCom	[TrustCoM Consortium 2007]	SLA
11	CC-PI	[Buscemi and Montanari 2007]	SLA
12	KISLA	[Toktar et al. 2007]	SLA
13	ULMS	[Unger et al. 2008]	SLA
14	DPL	[Milosevic and Dromey 2002]	Service Contract
15	X-Contract	[Molina-Jimenez et al. 2003]	Service Contract
16	BCL	[Linington et al. 2004]	Service Contract
17	SweetDeal	[Grosof and Poon 2004]	Service Contract
18	CTXML	[Farrell et al. 2004]	Service Contract
19	SWCL	[Oren et al. 2005]	Service Contract

Table X. The SLA and service contract languages examined

ants of *pi-calculus*, *ontologies* and *other*, and have been used for categorizing each language.

The results of this evaluation/categorization are presented in the first row of the *Description* composite SLA life-cycle activity of Tables XI, XII, and XIII. As it can be easily seen, the vast majority of the SLA-MMs is expressed with informal formalisms (mostly XML Schema but also other schema languages which focus on the concrete syntax of a language). The same result applies both locally and to each partition (i.e., SLA language and SCL partition). Obviously, the rationality of deciding to use a schema language is because it leads to a quick way of producing a language, surpassing in this way the expression of its abstract syntax. In addition, the majority of the approaches uses XML Schema for the concrete syntax and XML for the representation of the SLAs. XML is adopted due to its *platform-independence*, *simplicity*, and *ease of use*, the excellent range of tool support available enabling *automatability*, including editors, parsers, transformation engines, and document validity checkers, and the fact that is both *human and machine understandable and processable*. However, XML Schema and correspondingly XML lack proper and precise *language semantics* needed to perform semantic consistency of SLAs and the *formality* needed to perform *SLA analysis* (analysability) [Skene 2007] which can be used to reveal hidden obligations and other important nonvisible implications. For this reason, XML-based SLA descriptions are either transformed to another formalism [Linington et al. 2004] or other formalisms are adopted for expressing the SLA-MM like FSM (X-Contract), RuleML (RBSLA and SweetDeal) or Event Calculus (TCXML). While the adoption of stronger formalisms equips the SLA language with powerful tools to perform various forms of reasoning, there is usually a trade-off with simplicity, ease of use, and human-understandability and processing.

The majority of the SLA and service contract languages are able to perform at

Table XI. Evaluation results of SLA languages

Life-cycle Activity	Criteria	QML	WSML	WSLA	WS-A	SLAng	WSOL	RBSLA
Description	Formalism	UML	Informal	Informal	Informal	Informal	Informal	RuleML Ontologies
	Coverage	[p,y]	[p,p]	[p,y]	[y,p]	[y,y]	[p,p]	[p,y]
	Reusability	yes	yes	yes	yes	part	yes	yes
	Composability	no	fair	no	fair	fair	no	no
Matchmaking	Metric Definition	yes	no	yes	no	yes	no	yes
	Alternatives	impl	no	impl	impl	no	impl	impl
	Soft Constraints	no	no	no	yes	no	no	no
	Matchmaking Metric	yes	no	no	no	yes	no	no
Negotiation	Meta-Negotiation	poor	poor	poor	fair	poor	poor	poor
	Negotiability	no	no	no	part	no	no	no
Monitoring	Metric Provider	no	yes	yes	no	yes	yes	no
	Metric Schedule	no	yes	yes	no	yes	no	yes
Assessment	Condition Evaluator	no	no	yes	no	no	yes	no
	Qualifying Condition	no	no	impl	yes	no	no	no
	Obliged	no	no	yes	yes	yes	yes	yes
	Assessment Schedule	no	yes	yes	no	no	no	no
	Validity Period	no	yes	yes	no	no	no	yes
	Recovery Actions	no	yes	yes	no	no	yes	yes
Settlement	Penalties	no	no	no	SLO	no	SL	SL
	Rewards	no	no	no	SLO	no	no	SL
	Settlement Actions	no	no	yes	no	no	no	yes
Archive	Validity Period	no	yes	yes	yes	yes	no	no

Table XII. Evaluation results of SLA languages

Life-cycle Activity	Criteria	QoWL	GXLA	TrustCom	CC-PI	KISLA	ULMS
Description	Formalism	Informal	Informal	Informal	cc-pi calculus	Informal	Informal
	Coverage	[y,p]	[p,p]	[y,y]	[p,p]	[p,p]	[p,p]
	Reusability	part	yes	yes	part	yes	yes
	Composability	fair	good	fair	neutral	good	good
Matchmaking	Metric Definition	no	no	yes	no	no	no
	Alternatives	no	impl	impl	impl	impl	impl
	Soft Constraints	no	no	yes	yes	no	no
	Matchmaking Metric	no	no	no	yes	no	no
Negotiation	Meta-Negotiation	good	poor	fair	poor	poor	poor
	Negotiability	no	no	part	no	no	no
Monitoring	Metric Provider	no	no	yes	no	no	no
	Metric Schedule	no	no	yes	no	no	no
Assessment	Condition Evaluator	no	no	no	no	no	no
	Qualifying Condition	no	no	no	no	no	no
	Obliged	no	yes	yes	no	no	no
	Assessment Schedule	no	yes	yes	no	yes	no
	Validity Period	no	yes	yes	no	yes	no
	Recovery Actions	no	yes	no	no	no	no
Settlement	Penalties	SL	no	SLO	no	SLO	no
	Rewards	no	no	SLO	no	no	no
	Settlement Actions	no	no	no	no	no	no
Archive	Validity Period	no	yes	yes	no	yes	no

Table XIII. Evaluation results of service contract languages

Life-cycle Activity	Criteria	DPL	X-Contract	BCL	SweetDeal	CTXML	SWCL
Description	Formalism	Informal [p,n]	FSM [p,n]	Informal [p,n]	RuleML [p,n]	Informal [p,p]	Ontologies [p,p]
	Coverage	part	no	yes	yes	part	part
	Reusability						
	Composability	no	no	no	neutral	no	no
Matchmaking	Metric Definition	no	no	no	no	no	no
	Alternatives	no	impl	no	no	no	no
	Soft Constraints	no	no	no	yes	no	no
	Matchmaking Metric	no	no	no	no	no	no
Negotiation	Meta-Negotiation	poor	fair	poor	poor	poor	poor
	Negotiability	no	no	no	no	no	no
Monitoring	Metric Provider	no	no	no	no	no	yes
	Metric Schedule	no	no	no	no	no	no
Assessment	Condition Evaluator	no	yes	no	yes	no	no
	Qualifying Condition	yes	no	no	no	yes	yes
	Obliged	yes	yes	yes	yes	yes	yes
	Assessment Schedule	yes	yes	yes	yes	yes	yes
	Validity Period	no	no	yes	no	no	no
	Recovery Actions	yes	yes	yes	yes	yes	yes
Settlement	Penalties	SL	no	SLO	SL	SL	SL
	Rewards	SL	no	no	SL	SL	SL
	Settlement Actions	yes	yes	yes	yes	yes	yes
Archive	Validity Period	no	no	yes	no	no	no

most two forms of *validity*, the structural and semantical ones (i.e. discovering syntactic and semantic inconsistencies with the help of DTDs, XML Schemas, and ontologies). However, another form of validity is also required, which we name as *SL validity*, in order to discover a specific type of *quality inconsistencies* in SLs. This specific type concerns the *constraint consistency*. As SLs are composed from the logical combinations (usually conjunctions) of SLOs, it must be checked if this combination is meaningful and correct. For instance, if two constraints of the form $X < a$ and $X > b$, where X is a QoS metric and $b > a$, are conjunctively combined, then the QoS metric X would not be allowed to take any value from its value type and so the produced SL would not be valid. This problem is exacerbated when arbitrary functions are involved in the constraint expressions of SLOs. One good solution would be to transform the SL description into an appropriate constraint model and then use Constraint Logic Programming (CLP) (for arbitrary logical combinations of SLOs) or Constraint Programming (only for conjunctions of SLOs) techniques [Rossi et al. 2006] to check the constraint model's consistency [Müller et al. 2008].

5.2.1.2 *Coverage.* SLA and service contract languages should be able to express in an efficient and complete way both functional and quality terms. The description of functional terms should include the description of the functionality of the service, its operations, its input and output. Moreover, if the service is composite, then all of its tasks, both internal and external, should also be described. All languages are able to define SLOs. However, the description of quality terms should also include the description of the QoS metrics to be measured and various other quality concepts in order to be complete. Usually, if the SLA or service contract languages are not able to describe these additional quality terms, they reference external descriptions of them in respective SQSLs. Thus, the evaluation of this criterion takes into account these two description aspects and provides the following values for a language: [*n,n*] (no functional and no SLO description), [*n,p*] (no functional and only SLO description), [*n,y*] (only complete quality description), [*p,p*] (references to functional and only SLO description), [*p,y*] (reference to functional and complete quality description), [*y,y*] (complete functional and quality description).

The results of this evaluation are presented in the second row of the *Description* activity of Tables XI, XII, and XIII. It can be easily seen from these results that all the languages reference or explicitly define functional terms. Moreover, the vast majority of these languages references an external functional description of the offered service (e.g. WSDL or BPEL). This result also applies in each partition. Another interesting result is that only four SLA languages enable the description of the functional terms of the SLA, while no SCL enables this description. It must also be noted that the WS-Agreement and TrustCom SLA languages enable both the description and reference of the functional terms of the SLA. Of course, the latter language is an WSLA-based extension of WS-Agreement, so it is reasonable to inherit the majority of WS-Agreement's capabilities. This can be easily understood by comparing Tables XI and XII. As already explained in Section 4, the syntactical separation of functional and SLA descriptions enables the reuse of a specific contract among many services that exhibit the same quality capabilities. Moreover, it facilitates the management and evolution of a contract (through the

manipulation of changing SLs [Tosic et al. 2003]) and disables the repetition of the functional description of the service in all contracts of this service. On the other hand, the inclusion of the service functional description inside an SLA mitigates the risk that the client experiences a different functionality from the one requested without being able to claim for this violation in the agreement. In this way, abnormal behavior of SPs that change the functionality of the service and its description externally and not visibly to an SLA is avoided.

Concerning the description of additional quality terms apart from SLOs, the majority of the languages only references external descriptions of them usually found in SQMs. This result applies both globally and also in the SLA languages partition. In the latter partition, there is no SLA language that does not describe or reference additional quality terms. This result is, of course, more than expected. Moreover, it can be easily seen that the CSLs that have been constructed to accommodate for any possible electronic contract and not just SLAs are not very efficient in this matter as most of them do not reference any quality term but could be extended to do so. Finally, it must be noted that referencing external descriptions of the additional quality terms enables their reuse but may create problems in the *Matchmaking* activity of the SLA management life-cycle which is analyzed in the next subsection.

5.2.1.3 *Reusability & Extensibility.* An SLA or service contract language should enable the creation of templates and documents that can be re-used or extended for creating new SLA or service contract specifications, respectively. Moreover, parts of SLA or service contract specifications, like the functional and quality terms, should be able to be reused across many SLA or service contract documents or extended appropriately. Thus, the evaluation of this criterion for each language could get the following values: *no*, *part* (i.e. only the whole SLA or service contract is reusable and extensible), and *yes* (i.e. parts and whole SLA or service contract are extensible and reusable).

The results of this evaluation are presented in the third row of the *Description* activity of Tables XI, XII, and XIII. As it can be seen, the majority of the SLA languages are re-usable and extensible both in the SLA specification entirety and in its parts. This result applies both globally and in the SLA language partition. However, as far as the SCL partition is concerned, the result is reversed as there are three SCLs that are re-usable and extensible only in their entirety and two which are also re-usable and extensible in their parts.

5.2.1.4 *Composability.* This property indicates the ability of an SLA or service contract language to represent composite SLAs or service contracts, respectively. This property can be achieved if the language presents the following features: a) ability to describe or reference descriptions of composite services (i.e. the service and its components); b) ability to define or reference metrics that are associated to composite services and are computed based on aggregation rules that depend on the structure of the composite service; c) ability to cater for the different types of parties (i.e. third-party SPs) involved in composite contracts; d) ability to define composite and component-based SLs and their associations; e) ability to define appropriate action guarantees that take into account the two-level hierarchy of the

contract. Most of the languages possess at most both of the first two abilities. For this reason, this criterion has been evaluated for each language depending on the language's satisfaction of these five abilities according to their order. So, if the language does not possess any of the abilities, it is evaluated with *no* (i.e. it is not composable). If it possesses one of the first two abilities, it is *neutral*. If it possesses the first two abilities, it is considered *fair* (i.e. it has the basis for becoming composable). If it possesses the first three or four abilities, it is considered *good*. Finally, if it possesses all the abilities, it is evaluated with *yes* (i.e. it is composable).

The results of this evaluation are presented in the last row of the *Description* activity of Tables XI, XII, and XIII. As it can be seen, the majority of the languages have not even the basis of being composable. This result applies both globally and in the SCL partition. In fact, in the latter partition five out of six approaches are not composable at all, while one approach satisfies only the first of our criteria. This actually means that SCLs were not designed in order to represent composite service contracts or SLAs. As far as the SLA languages partition is concerned, there is actually a balance between approaches that have good, fair/basic and no composability at all. In fact, the SLA languages that score good in their composability (GXLA, KISLA, and ULMS) have been proposed recently. This actually means that the SLA modelers are starting to understand the need for representing composite SLAs but have not yet extended their languages appropriately or produced a new language that is capable of representing composite SLAs.

5.2.1.5 *Overall Analysis for the Description Activity.* By inspecting the performance of the examined languages on the four description criteria, it is easily comprehensible that there is no SLA or service contract language that meets our high standards. Only GXLA, KISLA, and ULMS can be distinguished based on their capabilities with respect to composability. However, these languages lack the appropriate formality or transformation to a respective formality which is needed for SLA (service contract) validity and analysis.

5.2.2 *Matchmaking.* In the current literature, there is only one research approach [Oldham et al. 2006] that performs proper matchmaking of SLA specifications so as to match the QoS requirements of the user with the QoS capabilities of the service. In this approach, the SLA specifications of WS-Agreement are enriched with semantic annotations from both domain-independent and domain-dependent ontologies, while rules are also used to infer the matchmaking. The usual procedure followed in the rest of the approaches is that matchmaking is performed during service negotiation, where one of the participants proposes a specific SLA (or service contract) template and the other one accepts it or changes it by implicitly checking every SLO and changing its limits (if they are not satisfactory) and by entering new SLOs or proposes a new one. This type of matchmaking is not at all efficient for the following reasons: a) QoS metrics are defined inadequately and syntactically in possibly different languages leading to low accuracy results; b) no matchmaking metric is defined so each party utilizes his own metric to infer if the proposed SLA template is the appropriate one c) the probability of matching is low because SLOs are usually expressed as hard constraints; d) it is time consuming as each time each

party receives, parses, and processes an SLA document and may send a modified version of it or a new one in the best case (i.e., when only one SP and SR negotiate).

Based on the content of most SLA life-cycles (including the one we propose) and the above reasons, SLA (or service contract) template matchmaking should be performed before service negotiation in order to discover those services of the corresponding SPs that suit the quality requirements of the user. This process can be effective and accurate if some prerequisites are met by the SLA and service contract languages, while there is a common, unique, and fair matchmaking metric that can be used to perform the matchmaking in such way that will always give the same results for the same input. In the following, we analyze three main description prerequisites and one specific requirement (i.e., the existence of a matchmaking metric) that must be met by an SLA language in order to enable the matching of its specifications. In the end, an overall analysis of the SLA languages ability to support this SLA management life-cycle activity will be given.

5.2.2.1 *Metric Definition.* Metric modeling capabilities have been already analyzed in Section 4, where four SLA languages were compared. The rest of the SLA (apart from TrustCom) and service contract languages under our consideration are not able to define QoS metrics but just reference external metric descriptions of SQSLs. In this way, two main problems may arise: a) language incompatibility – the involved SQSLs may encompass different metric meta-models so they can describe metrics in a different way and, thus, it will be difficult to transform one SQSL's metric description into the other's one when matchmaking SLA descriptions; b) even if the SQSLs are compatible, equivalent metrics described in these different languages may have a different name, so their descriptions have to be matched via metric matching rules [Kritikos and Plexousakis 2006] in order to infer their equivalence. These two problems reduce the accuracy of the matchmaking activity. To this end, languages that define metrics or enforce the use of a specific SQSL that defines metrics are preferred. For this reason, the evaluation of this criterion for each language would be *no*, if the language does not define metrics or reference metric descriptions from a specific SQSL, or *yes* otherwise.

The results of this evaluation are presented in the first row of the *Matchmaking* life-cycle activity of Tables XI, XII, and XIII. As it can be seen, only five SLA languages (four plus TrustCom that relies on WSLA) satisfy this criterion. In fact, these are the approaches that can define QoS metrics and other quality terms of SLOs on their own. So, by taking into account also the results of the *coverage* criterion of the *Description* composite activity, we come into the conclusion that all languages that can reference external metric descriptions do not determine which SQSL should be used to specify these descriptions. Thus, in this way, these languages may lead to low accuracy matchmaking results based on the above analysis.

5.2.2.2 *Alternatives.* In order to infer if two SLA or service contract specifications match, their part corresponding to the offered or required SL must be matched. Existing languages do not encompass the concept of SL but either they assume that it is the conjunction of all SLOs defined in the SLA/service contract or they emulate it either through the use of logical predicates that logically connect the defined SLOs or by offering different SLAs/service contracts for each service.

However, the ability to implicitly represent a SL is not enough as the probability that there is a match between the encompassing SLOs of the compared SLs of two SLA specifications is very low. Moreover, users have diverse needs that can be represented through trade-offs between the requested SL and its cost. For the above reasons, in order to increase the chances of matching SLA/service contract specifications, SLA and service contract languages should be able to represent alternative SLs. Alternative SLs represent the different modes in which a service can operate, in order to suit the diverse needs of different classes of users, and the variations of a requested SL by a SR that trade-off the SL with the price the SR is willing to pay. Thus, the evaluation of this criterion for each language would be *no* if the language is not able to represent SLs, *impl* if the language can define them implicitly, and *yes* if there is an explicit construct in the SLA language that is used to represent alternative SLs.

The results of this evaluation are presented in the second row of the *Matchmaking* life-cycle activity of Tables XI, XII, and XIII. As it can be seen, the majority of the languages are able to implicitly define alternative SLs which are needed in order to increase the chances of matchmaking with potential SRs. This result also applies to the SLA languages partition. Moreover, there is no language that explicitly defines SLs. Finally, among the service contract languages, only X-Contract is able to implicitly define SLs. Thus, the results clearly show that both SLA and service contract languages were not designed to support SLs at all but only SLA languages are able to implicitly define them.

5.2.2.3 *Soft Constraints.* Even if many alternative SLs are represented in an SLA/service contract template to be matched, there will always exist a problem [Kritikos 2008] where users express over-constrained SLs that cannot be matched by any alternative SL of any offered SLA/service contract template. An over-constrained SL means that its encompassing SLOs contain very restrictive constraints that are hard to be all satisfied. The root of this problem lies on the fact that SLOs are expressed as hard constraints that must be satisfied at all costs in order to infer the matchmaking. For this reason, the solution to this problem may come through the use of soft constraints. In particular, if SLOs are expressed as soft constraints, where the user expresses their significance through the use of a weight or level, then not all of them have to be satisfied when matching. In this way, there can be a match between an offered and requested SL, even if some not significant requested SLOs are violated. Thus, the evaluation of this criterion for each language would be *no* if the language cannot express soft constraints (i.e. SLOs), or *yes* otherwise.

The results of this evaluation are presented in the third row of the *Matchmaking* life-cycle activity of Tables XI, XII, and XIII. Based on these results, only three SLA and one service contract language are able to define soft constraints, while the rest of the languages define SLOs as hard constraints. Thus, only these four languages could be used for expressing SLAs/service contracts when over-constrained SRs SLAs/service contracts are issued and have to be matched with SPs SLAs/service contracts in a particular application domain.

5.2.2.4 *Matchmaking Metric.* As all languages differ in the way they define QoS metrics and SLOs, it would be very useful for implementing SLA/service contract matchmaking engines if a specific matchmaking metric was defined internally or externally in the SLA/service contract language. This metric would be used for matching SLAs/service contracts (defined by the respective language) in a fair and uniform manner according to the matchmaking requirements defined in [Kritikos 2008; Kritikos and Plexousakis 2009]. Thus, the evaluation of this criterion for each language would be *no* if no matchmaking metric is defined, or *yes* otherwise.

The results of this evaluation are presented in the last row of the *Matchmaking* life-cycle activity of Tables XI, XII, and XIII. As it can be easily seen, only three SLA languages (QML, SLang, and CC-PI) explicitly define a matchmaking metric with which their produced SLA specifications can be matched.

5.2.2.5 *Overall Analysis for the Matchmaking Activity.* Based on the presented results for the four criteria of the *Matchmaking* activity, there is no language that satisfies all of them. Only QML, TrustCom, and CC-PI satisfy three of the criteria. However, among these three SLA languages we consider QML as the best because we regard that the *soft constraints* criterion is the least significant with respect to the other ones which are required in order to properly support the SLA matchmaking activity. It must be noted here that QML is the oldest of all SLA languages and is not used any more. However, it was designed with the explicit goal of *contract conformance*, which is actually used for the matchmaking of the SLA specifications. Thus, based on our analysis, it can be deduced that the majority of the languages and especially the SCLs were not designed with the objective of matchmaking the quality terms of their specifications.

5.2.3 *Negotiation.* Service negotiation is one of the most important activities as it produces the final SLA/service contract document that will drive the service execution. For this reason, SLA/service contract languages must describe all the appropriate information that will be provided as input and assist the service negotiation process. This information can be categorized into two parts: *meta-negotiation*, and *negotiability*. In the following, we explain why these two types of information must be captured by SLA/service contract languages and what should be their content, while we explore if the languages capture this essential information. Finally, we conclude by inspecting the overall performance and support of the examined languages for this SLA management activity. It must be noted that we have excluded from the appropriate negotiation information the description of the *negotiation strategies* as they constitute private and sensitive information for the participants which must not be exposed in SLA/service contract templates.

5.2.3.1 *Meta-Negotiation. Meta-negotiation* is any information that can be used for negotiation establishment, i.e. for enabling and initiating the negotiation between the participants. We have identified the following information as meta-negotiation [Brandic et al. 2009; Comuzzi et al. 2009]: a) negotiation protocol support; b) description of negotiation capabilities; c) authentication method reference;

The *negotiation protocol* is the allowable sequence of messages exchanged used to negotiate and conclude (i.e., agree) an SLA/service contract. In addition to

defining the messaging behavior the protocol should also unambiguously define the semantics and format, or schema, of the messages. Each negotiation participant will be able to support a subset of all possible negotiation protocols. Thus, the supported negotiation protocols of all participants must be matched in order to find the appropriate one for enacting the negotiation. For this reason, the corresponding SLA/service contract template of each participant must reference all negotiation protocols that he can support. Moreover, this reference should include a pointer to the actual description of the supported negotiation protocol for two reasons: a) to enable reasoning on protocol compatibility and substitutability, i.e. if one protocol can be used in place of the other, and b) some supported negotiation protocols may not be widely known so they are not implemented in negotiation engines and brokers; therefore, they have to be defined, e.g. in BPEL, in order to be properly enacted by the negotiation broker.

In case that there will be no matching negotiation protocol, in [Comuzzi et al. 2009] it is advocated that the negotiation capabilities of the participants must be described in a more fine-grained way along with the possibilities of delegating part of a corresponding role in a negotiation protocol to trusted third-parties. Then, ontology-based reasoning can be used to infer if a specific negotiation protocol can be supported by the participants and their trusted third-parties. Thus, based on the above analysis, the fine-grained negotiation capabilities of the participants must be also advertised in their SLA templates apart from the coarse-grained ones.

While negotiating, the participants reveal and exchange important information which should not be exposed to third-parties listening on insecure channels established between the participants. For example, an SP does not want the offers he make to specific (e.g. privileged) clients to be viewed by all other clients. For this reason, the participants must describe in their SLA/service contract templates the authentication methods they prefer to be used for securing the channels exploited when exchanging negotiation information. Then, there will be a matching of the preferred authentication methods so as to select the most common one for the negotiation.

Based on the above analysis, each language is evaluated according to the number of meta-negotiation information it can describe. Thus, if it cannot describe any information, we evaluate it as *poor*. If it describes only one of the above three types of information, it is considered *fair*. If it can describe two types of meta-negotiation information, it is considered *good*. Finally, if the language describes all possible negotiation information, it is considered *rich*.

The results of this evaluation are presented in the first row of the *Negotiation* life-cycle activity of Tables XI, XII, and XIII. Based on these results, the vast majority of the languages is not capable of modeling any meta-negotiation information. This result holds also in every partition. Only WS-Agreement, TrustCom, and X-Contract determine or can represent negotiation protocols, while QoWL is able to both specify the negotiation protocol and the authentication method to be used for the SLA negotiation.

5.2.3.2 *Negotiability.* While *meta-negotiation* represents information which is used before service negotiation in order to enact it, *negotiability* represents information which is used during the negotiation. In particular, negotiability is the

ability of an SLA/service contract language to describe which parts of its specifications are negotiable and in what way. Focusing on quality, a language presents negotiability if it can characterize which quality terms are negotiable or not and which are the allowable values in the upper and lower limits of the negotiable quality terms. Thus, the evaluation of this criterion for each language would be *no* if the language does not define which terms are negotiable, *part* if the language characterizes only the negotiability of terms, and *yes* if the language also determines which are the allowed values or range of values for the upper and/or lower limits of each quality term.

The results of this evaluation are presented in the second row of the *Negotiation* life-cycle activity of Tables XI, XII, and XIII. Based on these results, only two SLA languages (WS-Agreement and TrustCom) specify in a special part of their produced SLA templates which terms are negotiable. However, they do not specify the way these terms are negotiable. On the contrary, the constraints used to define the negotiable terms are hard and do not specify if the limits of a quality term can take one or more values.

5.2.3.3 *Overall Analysis for the Negotiation Activity.* According to the evaluation results on the above two criteria of the *Negotiation* activity, only WS-Agreement (and TrustCom that extends it) can partially provide information that can assist this activity. However, it should be extended by modeling the negotiation capabilities of the participants, the authentication method used for the information exchange during the negotiation, and specific constraints that define the allowed values for the limits of the negotiable quality terms. Considering the latter extension, one good solution is proposed in [Andrieux et al. 2004] which is, however, not adopted in the formal specification of the language. As it can be easily understood, SCLs are not able to properly support the negotiation of the quality-based terms. This fact along with the inability of the SCLs to support the SLA matchmaking activity prevents their use as modeling languages of the quality-based clauses in service contract templates.

5.2.4 *Monitoring & Assessment.* After the SLA/service contract is established and corresponding service starts to execute, the service is monitored so as to assess if the SLOs defined in its SLA/service contract are violated. Service monitoring is performed by producing measurements according to the information that is encapsulated in the metrics defined in the SLOs that are in the scope of this service. As metric definition was evaluated in the previous section and association of metrics to service objects was evaluated previously in the composite activity named *Description*, the only additional information that is needed for *Monitoring* is who is in charge of performing the measurements for each metric, i.e. the *Metric Provider*, and how often the measurements are produced, i.e. the *Metric Schedule*. This information is encompassed in a metric meta-model, but is usually specified concretely only when the SLA/service contract is agreed after service negotiation.

5.2.4.1 *Metric Provider.* The Metric Provider is the responsible supporting party for producing the measurements of a specific metric. The evaluation of this criterion for each language would be *no* if the language does not define this responsible party, or *yes* otherwise.

The results of this evaluation are presented in the first row of the *Monitoring* life-cycle activity of Tables XI, XII, and XIII. As it can be easily seen, only five SLA and one service contract language are able to specify which is the party responsible for providing the measurements of metrics involved in the SLOs. This is a major limitation for the rest of the languages for two main reasons: a) most languages presuppose that the measurements are provided by the SP. This is not true as measurements of some metrics should be provided by third-parties or SRs and not SPs; b) The SLO evaluators, which may be different from SPs or even SRs, would be able to assess the SLOs only if they know from where to get the measurements of the SLO metrics.

5.2.4.2 *Metric Schedule.* The schedule of a metric determines the frequency of the production of its measurements. The evaluation of this criterion for each language would be *no* if the language does not define any metric schedule, or *yes* otherwise.

The results of this evaluation are presented in the second row of the *Monitoring* life-cycle activity of Tables XI, XII, and XIII. Based on these results, only four SLA languages are able to specify metric schedules. Compared to the results of the previous criterion, three (WSML, WSLA, and SLAng) of the four languages satisfying the previous criterion also satisfy this one. The rest of the SLA languages do not model this feature and the same fact applies for all SCLs. However, this feature is important as it is used in order to specify the timing of the measurement productions of the SLO metrics. Thus, the lack of this feature would cause problems in the *Assessment* activity.

The assessment of SLOs in one of the most crucial activities of the SLA life-cycle management. For this reason, the SLA/service contract language must model all appropriate information that could be used to support this activity. Apart from the main condition of the SLO (clause) that is modeled in all SLA (service contract) languages, other very important SLA assessment information that should be modeled is: a) *Condition Evaluator*, b) *Qualifying Condition*, c) *Obliged Party*, d) *Assessment Schedule*, e) *Validity Period*, and f) *Corrective Actions*. In the following, we analyze the purpose and content of this information and we evaluate if the examined languages have modeled it.

5.2.4.3 *Condition Evaluator.* Similarly to metric measurement, the assessment of SLOs should be made by a supporting party which is named *Condition Evaluator*. This party would be in charge of collecting the measured values of all metrics involved in an SLO, replacing the metrics with their values, and then checking if the SLO holds or not. The evaluation of this criterion for each language would be *no* if the language does not define this type of supporting party, or *yes* otherwise.

The results of this evaluation are presented in the first row of the *Assessment* life-cycle activity of Tables XI, XII, and XIII. As it can be easily seen, only two SLA languages (WSLA and WSOL) and two SCLs (X-Contract and SweetDeal) are able to model this information. All the other languages pre-suppose that the SLA/service contract assessment activity is performed in the management systems of the signatory parties based on the information coming or pulled from the monitoring components. In this way, they limit the way an SLA/service contract man-

agement system can be implemented or distributed as they exclude in this way the existence of third-party assessment components.

5.2.4.4 *Qualifying Condition.* Apart from the main condition that is evaluated, there can be a precondition, named *Qualifying condition*, which must hold in order to assess the SLO. This precondition may express assertions over service or other quality attributes or external factors such as the *service request rate* of the SR. The evaluation of this criterion for each language would be *no* if the language does not define qualifying conditions, *impl* if it defines them implicitly through other constructs, or *yes* otherwise.

The results of this evaluation are presented in the second row of the *Assessment* life-cycle activity of Tables XI, XII, and XIII. Based on these results, only one SLA language (WS-Agreement) and three SCLs are able to explicitly model the *qualifying condition* attribute, while another SLA language (WSLA) can implicitly define it through other constructs. All the other languages do not offer this capability. This lack leads to inability of expressing preconditions for the enactment of the assessment of an SLO, which would eventually lead to situations where SLOs are assessed wrongly with regards to timing or other excluding conditions (e.g. client-side or management-related restrictions).

5.2.4.5 *Obliged.* The responsible in charge of delivering what is promised in an SLO is usually called the *Obliged*. In many cases the obliged party of an SLO is the SP. However, in some cases it can be another party, e.g. a third-party provider of a service component. Thus, every language should associate an SLO with the party that promises it. The evaluation of this criterion for each language would be *no* if the language does not define the obliged party in the SLOs, or *yes* otherwise.

The results of this evaluation are presented in the third row of the *Assessment* life-cycle activity of Tables XI, XII, and XIII. As it can be seen, the majority of the SLA languages is able to define which is the obliged party in an SLO. Concerning the contract-based languages, all of them model this information as their design is based on policies expressing obligations and various other types of implications.

5.2.4.6 *Assessment Schedule.* An SLO is not assessed just one time but several times according to an *Assessment Schedule*. This schedule can be as quite simple as assessing when new values are measured for the metric of the SLO or complex representing a sequence of regularly occurring events. The evaluation of this criterion for each language would be *no* if the language does not define the assessment schedule, or *yes* otherwise.

The results of this evaluation are presented in the fourth row of the *Assessment* life-cycle activity of Tables XI, XII, and XIII. By inspecting these results, it can be easily seen that most of the languages specify the assessment schedule of an SLO. This result applies also to the SCL partition. As far as the SLA languages partition is concerned, most of the SLA languages do not model this information. This limits their application only in expressing SLAs that involve services whose performance should be checked at only one instant. The languages that are able to model this criterion follow two different approaches. In the first approach the schedule is defined concretely with timing constraints, while in the second approach the schedule is based on events originating from the monitoring components of the

SLA management system. The first approach is adopted by the SLA languages that model this information, while the second approach is adopted by all SCLs. One important observation that should be taken is that all SCLs have been designed in order to support this criterion, while most SLA languages have not.

5.2.4.7 *Validity Period.* While the assessment schedule determines when to assess an SLO, the validity period determines the time period in which the SLO is guaranteed and, thus, should be checked for validity. Example of the values of this field are *business days*, *regular working hours* or *maintenance periods*. The evaluation of this criterion for each language would be *no* if the language does not define the an SLO's validity period, or *yes* otherwise.

The results of this evaluation are presented in the fifth row of the *Assessment* life-cycle activity of Tables XI, XII, and XIII. As it can be seen, most of the languages do not model this information. This result also applies to the SCL partition. However, it does not apply to the SLA languages partition, where there is a balance between the approaches that model this information and those that do not. This means that the SLA language designers have better understood the need of supporting this criterion with respect to those of the SCLs.

5.2.4.8 *Corrective Actions.* When an SLO is violated and the signatory parties are informed about it, then corrective actions need to be carried out at the obliged party's management system or at the global level by renegotiating or canceling the SLA/service contract. In the case where corrective actions should be taken by the obliged party, as already discussed previously in this section, the choice of which action to perform would depend on many situational factors and on the business goals and policies of the obliged party. Thus, the obliged party would not desire or would not be possible to advertise in an SLA/service contract what actions to perform in which SLO violation case. However, there can be cases where this party advertises some corrective actions to be taken for the corresponding SLOs violations in order to increase its reputation and trust with respect to the SR or to guarantee the high gain that will get from assuring the agreed SL to a *golden class* customer. Thus, we believe that this information should be certainly modeled in an SLA/service contract. So, the evaluation of this criterion of each language would be *no* if the language does not model *corrective actions*, or *yes* otherwise.

The results of this evaluation are presented in the fifth row of the *Assessment* life-cycle activity of Tables XI, XII, and XIII. As expected, the majority of the language designers has recognized the significance of the modeling of this information. The same result also applies to the SCL partition. However, it does not apply to the other partition, as most of the SLA languages do not model corrective actions. This means that SLA designers have not actually understood the importance of this criterion.

5.2.4.9 *Overall Analysis for the Monitoring & Assessment Activity.* Based on the overall performance of the examined languages on the two monitoring criteria and the sixth assessment criteria, WSLA is the best language that models all appropriate information that is required for supporting the *Monitoring* and *Assessment* life-cycle activities. The support of these two activities was one design requirement of WSLA that seems to have been successfully implemented in the specification of

this language. By inspecting the two different partitions, SCLs offer well support, while apart from WSLA the rest of the SLA languages do not seem to support well the SLA monitoring and assessment activities.

5.2.5 *Settlement.* In this activity, it is assessed what has happened during the service's execution and what are the responsibilities of each signatory party according to the agreed SLA/service contract. So, for example, if a specific SLO was violated, then the SP has to pay a small penalty to the SR. In another example, if the service runs in a higher SL than requested, then the SR has to pay, apart from the actual cost of the service, a reward for getting a better SL. We have identified that the appropriate information to be modeled by an SLA/service contract language for supporting this activity is a) the incurred penalties, b) the incurred rewards, c) settlement actions.

5.2.5.1 *Penalties.* Penalties are paid by the SP if one or more SLOs are violated. In most cases, each SLO is associated with a specific penalty-amount. However, there are some cases where the penalty to be paid increases exponentially with the number of violations [Paschke and Schnappinger-Gerull 2006]. This second case is not captured by most languages as it would require the definition of the appropriate SL first and then its association to a specific policy or function that would increase exponentially or linearly according to the number of violations in the SLOs that compose this SL. It should be noted that if a language is able to define penalties at the SL, then it can also define penalties at the individual SLO level. Thus, the evaluation of this criterion for each language would be *no* if the language does not define penalties, *SLO* if it defines penalties at the SLO level, or *SL* if it defines total penalties at the SL level.

The results of this evaluation are presented in the first row of the *Settlement* life-cycle activity of Tables XI, XII, and XIII. Based on these results, more than half of all languages are able to specify penalties. Among these languages, most of them define penalties for the whole SL, while only four define penalties for each SLO. Considering each partition separately, the majority of the SCLs is able to define penalties for each SL, while the majority of the SLA languages is not able to define any kind of penalty. Moreover, there is a balance between the SLA languages that model penalties at the SL level and those that model penalties at the SLO level. Based on these results, we come to the conclusion that SCL designers have understood the need to model penalties, while, strangely, SLA language designers have not.

5.2.5.2 *Rewards.* Rewards are paid by the SR if one or more SLOs are more than respected. Similarly to the previous field, rewards can be defined at the SLO (via a specific value) or SL level (via a function). The evaluation of this criterion for each language would be *no* if the language does not define rewards, *SLO* if it defines rewards at the SLO level, or *SL* if it defines total rewards at the SL level.

The results of this evaluation are presented in the second row of the *Settlement* life-cycle activity of Tables XI, XII, and XIII. While someone would expect that the results would be the same in numbers with the previous criterion, this is not the case. Less than half of all languages are able to define rewards at any level. Moreover, again the languages able to define rewards at the SL level are more than

those able to define rewards at the SLO level. The same result holds for the SCL language partition with the exception that there is no SCL that is able to define rewards at the SLO level. Concerning the other partition, the majority of the SLA languages are not able to define rewards at all. So the bad global result is mostly due to the inability of the SLA languages to model rewards. This means that either no real-world case of SLAs contains rewards or that the designers of the SLA languages have not recognized the need of modeling rewards. In our opinion, rewards should be modeled as they would give extra motives to SPs in order to provide even better SLs than the ones that have offered to respective SRs in the past. This SL upgrade would lead to increase in profits, which is one of the main goals of SPs when they offer their services. Moreover, the trust and reliability of the SP would also increase.

5.2.5.3 *Settlement Actions.* Contrary to the cases where penalties or rewards have to be paid and corrective actions have to be taken if an SLO is violated at runtime, in which the responsible is one of the signatory parties, settlement actions are mutually taken by both signatory parties in order to decide about the final outcome of the SLA/service contract. Thus, in case where there are no severe SLO violations or the number of violations is not high or there are no violations at all, then the SLA's outcome is successful and maybe only penalties or rewards have to be paid. However, in the opposite case, it should be determined if the SLA would be canceled, re-negotiated or re-enforced (e.g. the service has to be re-executed). Thus, SLA/service contract languages should be able to model these settlement actions and in which conditions they should be executed. So, the evaluation of this criterion for each language would be *no* if no settlement actions can be defined, or *yes* otherwise.

The results of this evaluation are presented in the last row of the *Settlement* life-cycle activity of Tables XI, XII, and XIII. As it can be seen, most of the languages are not able to model settlement actions. The same result applies to the SLA language partition, where only two out of thirteen SLA languages are able to model settlement actions. This means that the bad global result is due to the inability of the majority of the SLA languages to express conditions for re-negotiation or other types of settlement, which is a significant limitation that would discourage potential SPs or SRs from using them. This situation is reversed in the SCL partition, as all SCLs support the modeling of this information. This means that, indeed, the design of SCLs has been centered on the modeling of various compensation actions, including the settlement ones.

5.2.5.4 *Overall Analysis for the Settlement Activity.* Based on the evaluation results of the above three settlement criteria, only RBSLA among the SLA languages is able to satisfy them all and also to specify penalties and rewards on the SL. The same applies for four (DPL, SweetDeal, CTXML, and SWCL) out of six SCLs. This means that these five languages can be used for appropriately supporting the *Settlement* life-cycle activity. However, they have to be extended appropriately in order to specify explicit constructs that could be used to model the various settlement actions that may exist in an SLA/service contract. Currently, these languages require significant effort and extensions from the SLA modeler in order to express

different SLAs with different settlement actions and conditions. In addition, they force the SLA management systems that adopt them either to support one by one the different settlement actions that may exist in the SLAs or to define appropriate extensions with which then the modelers may specify their SLAs. The latter may lead to a situation where various different versions of the same language are adopted by different SLA/service contract management systems. Thus, in order to avoid such situations and further increase their adoption and universality, these languages should be extended or even re-designed appropriately.

5.2.6 *Archive.* An SLA/service contract is archived in three distinct cases: a) the SLA/service contract is canceled, b) the maximum number of service invocations has been reached, or c) its validity period has expired. In the first case, settlement or corrective actions dictate when the SLA/service contract is canceled. In the second case, the SLA/service contract has to determine this maximum number of service invocations, and none of the existing languages is able to model this information. In the third case, the language has to model the SLA's validity period. Besides the timing of archiving the SLA, some parties desire to dispose the SLA. In this case, the SLA is first archived and then disposed when a specific statutory period is expired. Again, none of the existing languages models this information. Thus, for this activity, we evaluate each language only based on its capability of modeling of the SLA's (service contract's) validity period. If the language does not model this period, the result of its evaluation would be *no*, otherwise *yes*.

The results of this evaluation are presented in the last row of Tables XI, XII, and XIII. As it can be seen, while the modeling of an SLA's (service contract's) validity period is very important, less than half of the languages are able to offer it. Concerning each partition separately, there is a balance between SLA languages that support and do not support the modeling of this information. On the other hand, only one SCL models this information. The latter two results lead to the conclusion that the SLA language designers are starting to understand the need of modeling this information, while the corresponding SCL designers do not. From the side of SCLs, the latter conclusion could be explained by taking into account the fact that these languages were designed with the focus on functionality and not on quality. Thus, as functionality does not change so much, there was no need to explicitly model the validity period of a service contract. The contract could be invalidated as soon as the business relationship between the contracting organizations ceased to exist for various reasons.

5.3 Overall Analysis

The analysis performed has revealed some significant facts and limitations of existing SLA and service contract languages in their ability to appropriately support the SLA life-cycle management activities. In this subsection, our analysis will focus on the overall global level of SLA management activity support so as to reveal other interesting facts that are not obvious in a first sight.

First of all, by inspecting the overall results of each activity, it becomes easily comprehensible that there is no language that supports in a satisfactory way all activities. On the contrary, in each activity a different language is awarded as the most appropriate one. In addition, there are some SLA management activities

which are not satisfactorily supported by any language, including those of SLA Description, Matchmaking, and Negotiation, while others are properly supported by few languages, including those of SLA Monitoring & Assessment, Settlement, and Archive.

The above general results burden the SLA languages, which were explicitly designed to support all of the SLA management activities, and especially the two most widely used, i.e. WSLA and WS-Agreement, as they signify that these languages do not possess all appropriate modeling capabilities so they should be used for expressing only some types of SLAs. On the contrary, the capabilities of these languages are complementary with respect to the support of the SLA management activities. For this reason, one solution that could be adopted is to design a new SLA language that would try to unify the capabilities of WSLA and WS-Agreement by extending them and encompassing some modeling constructs of the one to the other. One such paradigm is TrustCom, which happens to be the only SLA language that has a good evaluation score across all the activities. Another solution would be, of course, to design a new SLA language that could use the best modeling features of those two standardized ones and would explicitly model those features that are missing.

Another interesting result that can be derived from the above analysis is that SCLs are not capable of fully supporting most of the SLA management activities apart from those of SLA Monitoring & Assessment and Settlement. The main reason of this inability of SCLs is the focus of their design on service functionality, which was inevitable in the time of their modeling. In this way, service quality, which has the main focus now because of its dynamicity, is either neglected or not appropriately modeled. Thus, although these languages were designed to accommodate for any type of electronic contract, they cannot be used for specifying SLAs unless they are extended appropriately. One such language that could be easily extended is SWCL, which has the best score among all SWCLs across all SLA management activities. This language along with TCXML are the most representative and recent SCLs which have included quality-related constructs in order to accommodate for the change of focus from service functionality to quality.

6. CONCLUDING REMARKS AND FUTURE WORK

This paper has focused on investigating the issue of service quality description. To this end, a systematic review of a large number of proposed approaches has been conducted in order to reveal their strengths and weaknesses and to highlight where the need for further research and investigation is. Initially, the approaches were separated into three clusters according to their scope: (1) *service quality models* (SQMs) which are taxonomies of service quality that can be used to annotate other types of quality documents like QSDs and SLAs, (2) *service quality meta-models* (SQMMs) which are meta-models capable of expressing SQMs as well as service quality offerings and requirements (i.e, QSDs), and (3) *service level agreement meta-models* (SLA-MMs) which are meta-models capable of describing SLAs. Then, there was a comparison of the approaches of each cluster according to a set of scope-specific criteria aiming at unveiling which approaches are the consolidated ones and discussing which are the ones specific to given aspects. This compari-

son uncovered many interesting findings, while also spotted particular aspects of under-performance concerning the approaches of each cluster. In the next three subsections, we are going to summarize the most important of these findings and then draw directions for further research and improvement.

6.1 Discussion on Service Quality Models

Various SQMs have been proposed in the literature, from small or flat categories of service quality attributes to sophisticated taxonomies containing many categories and types of attributes. In order to compare these approaches in a fair and consistent manner, we have devised a set of criteria characterizing the extensiveness, information richness, structure, generality, and applicability of the considered SQMs. Concerning the first four aspects, the results of the evaluation have shown a trend that the approaches are improving over the years. In average, the SQMs have a satisfactory number of categories, where each category contains a small number of quality attributes. Most SQMs mainly cover general (i.e. domain-independent) quality attributes, while a small number of them also covers specific (i.e. domain-dependent) ones. As the inclusion of general attributes tends to cover the SP view while the inclusion of the specific ones tends to cover the SR view, most SQMs mainly cover the SP view. Besides, most SQMs contain both composite and atomic quality attributes along with the connecting relation between them. The latter relation is very important during service monitoring as it may be used for validating or enriching the monitoring results of a service monitoring engine or component.

Another interesting finding is that the majority of the SQMs includes only QoS attributes, while only the most recent approaches also include QoE attributes. The latter result signifies that the researchers are starting to realize that apart from considering attributes that can be assessed objectively, attributes that are assessed subjectively based on user feedback are equally important as they reveal the performance and usability of a service under the perspective of the person who really uses the service in order to satisfy his needs, so they also constitute critical factors for service selection.

An important evaluation result shows that apart from the very initial approaches that were focusing on quality attributes that are associated to the service layer, the rest of the approaches are increasingly considering attributes that can be associated to the one or both of the two additional layers, namely the application and infrastructure layers. While this is a significant advance, it is outweighed by the lack of inter-attribute dependencies not only at the same layer but also across the layers. The existence of inter-attribute dependencies is extremely important as it reveals the influence one attribute has on the other and can assist the service monitoring, assessment, and adaptation activities in detecting wrong monitoring facts and in discovering the components of the same or different layer that caused an SLO violation (i.e. for dependency analysis purposes).

Another drawback of the current state-of-the-art approaches is that they scarcely consider data quality aspects. However, since the output of a service is mostly composed of information, data quality can be considered as a part of the QoS and it can drive thoroughly the analysis of the required input and provided output.

As far as the last aspect of comparison is concerned, that is the one of applicability, only one SQM [Colombo et al. 2005] associates metrics with concrete assessment

formulas to all the attributes it contains. However, it does not perform well with respect to the first four aspects. Thus, this SQM could be used for annotating QSDs and SLAs which can be used across all service life-cycle activities but in specific scenarios. The majority of the rest of the approaches provides a metric description for some of the included attributes which, however, does not contain a precise assessment formula but a set of assessment rules. The latter rules could be further used to create precise assessment algorithms, so these approaches could also be exploited in all the service life-cycle activities, if extended appropriately.

Based on the above analysis of our findings, no SQM can be distinguished as the best according to its evaluation on all the considered criteria. On one hand, by considering the first four comparison aspects of extensiveness, information richness, structure, and generality, four approaches [Cappiello 2006; Cappiello et al. 2008; Kritikos and Plexousakis 2009; Mabrouk et al. 2009] can be distinguished as the best. On the other hand, by considering the last comparison aspect, the applicability one, then the approach of [Colombo et al. 2005] can be distinguished. Thus, a new SQM is needed that should combine the characteristics of the best approaches in the above two aspect partitions, describe all the possible but realistic inter-attribute dependencies, and include also data quality attributes.

6.2 Discussion on Service Quality Meta-Models

Similarly to SQMs, a great number of SQMMs has been proposed. These SQMMs were separated into three partitions based on their scope such that the analysis can be conducted globally for all approaches and locally in each partition. *Pure SQMMs* are SQMMs able to express QSDs and QSMs. SLA-enabled SQMMs are additionally able to express SLAs. On the other hand, security-based SQMMs focus on particular aspects of service quality description. All SQMMs were evaluated based on a set of criteria that were capturing the aspects of formality, expressiveness, complexity, and applicability.

As far as formality is concerned, the results have shown that the majority of the approaches are using either ontologies or informal formalisms. The former formalism is widely selected in pure SQMMs, while the latter formalism is the best modeling choice in the other two partitions. Moreover, a recend trend has been revealed for the pure and SLA-enabled SQMMs in using ontologies as their formalism. This adoption of ontologies can be explained by their ability to provide unambiguous semantics to quality terms and, thus, to enable machines to automatically process and reason on ontology-specified QSDs in order to support service life-cycle activities like discovery and negotiation.

Three main criteria were used to evaluate the richness aspect apart from other secondary ones. The first criterion was evaluating the SQM richness of the SQMM. The evaluation results of this criterion have shown that no SQMM is able to provide a rich SQM. Fortunately, pure SQMMs are starting to improve on this aspect over the last years. However, SLA-enabled and security-based SQMMs do not perform very well on this matter. The second criterion was evaluating the richness of the quality metric model. The results of the evaluation of this criterion were better with respect to those of the previous one. Indeed, the majority of SQMMs is encompassing an adequately rich quality metric model. Moreover, pure SQMMs are again improving on this matter over the last years and this is also reinforced

by the existence of a recent SQMM (OWL-Q) [Kritikos and Plexousakis 2006] with a very rich quality metric model. Finally, the third criterion was used to evaluate the richness in constraint description. Here, the evaluation results are even more better as the majority of the approaches encompasses a rich constraint model. In fact, there are also some pure and SLA-enabled SQMMs that encompass a very rich (excellent) constraint model. However, this is not the case for security-based SLAs which encompass a good constraint model. Another interesting result for this evaluation is that again pure SQMMs are improving on this matter over the years. Thus, by closely inspecting the results of these three criteria, it can be inferred that pure SQMMs are continuously increasing their expressiveness, while the approaches of the other two partitions are more or less stable.

It was extremely difficult to assess the complexity of the SQMMs by using good measures based on various practical reasons. So it was decided to use a simple measure on the number of concepts/entities included in the SQMM and specific thresholds in order to evaluate the approaches in particular categories. The results have shown that most of the SQMMs have low complexity. Moreover, the trend that pure and SLA-enabled SQMMs of higher complexity are proposed lately is revealed. By considering also the fact that SLA-enabled SQMMs are increasing their expressiveness in pure SLA-based aspects, this actually means that modelers are trying to increase the expressiveness of their SQMMs and, in result, the complexity of their SQMMs increases with respect to the number of concepts/entities.

The aspect of applicability was assessed based on two criteria. The first criterion was evaluating the connection of an SQMM with an SFSL in order to assess if the SQMM can be used in registries that are bound to specific SFSLs. The results have shown that the majority of the SQMMs is connected to an SFSL. Moreover, the most referenced language was WSDL followed by OWL-S. The second criterion was assessing if any service discovery and negotiation framework has adopted the SQMM under inspection. Based on the evaluation results, it is shown that pure SQMMs are used until the service negotiation activity by inspecting the functionality of existing frameworks that have been using this type of SQMMs and on the fact that these SQMMs do not model some information that is critical during service monitoring and assessment. The same result goes for security-based SQMMs. On the other hand, SLA-enabled can be used across the whole service life-cycle.

Based on the above analysis of our findings, there is not any perfect SQMM that scores the best value in all criteria. We believe that the above drawback is preventing the wide usage of SQMMs in service management systems. Indeed, as it was already shown, there are no SQMMs that are used in service discovery, negotiation, and SLA description and enforcement. Thus, there is actually a gap that must be closed by introducing either a new SQMM or extending an appropriate existing one.

6.3 Discussion on Service Level Agreement Meta-Models

Two type of agreement languages have been proposed in the literature: SLAs and service contracts. The former mainly focus on quality aspects, while the latter have been designed to accommodate for any type of electronic contract. Both of these types of language were evaluated on a set of criteria that were grouped along the SLA life-cycle activities. These criteria were assessing these languages along

the lines of the information they can describe which is necessary or appropriate for supporting the SLA life-cycle activities. In this way, by supporting the SLA life-cycle activities, the service life-cycle is also supported.

By inspecting the overall results of the evaluation, there is no language supporting in a satisfactory way all activities. On the contrary, in many activities a different language is awarded as the most appropriate one, while only few of the languages properly support these activities. In addition, there are some SLA management activities which are not satisfactorily supported by any language, including those of SLA Description, Matchmaking, and Negotiation.

As far as SLA languages are concerned, the analysis has shown that the two most widely used languages, namely WSLA [Keller and Ludwig 2003] and WS-Agreement [WS-AGREEMENT 2003], do not possess all appropriate modeling capabilities and this result applies also to the other SLA languages. Moreover, the capabilities of these two languages are complementary with respect to the support of the SLA management activities. For this reason, there are some approaches that try to unify the best characteristics of these two languages, such as TrustCom [TrustCoM Consortium 2007]. However, this unification is not enough and those features that are missing should be additionally modeled.

Another interesting finding is that SCLs are not capable of fully supporting most of the SLA management activities apart from those of SLA Monitoring & Assessment and Settlement. The main reason of this inability of SCLs is the focus of their design on service functionality, which was inevitable in the time of their modeling. Thus, although these languages were designed to accommodate for any type of electronic contract, they cannot be used for specifying SLAs unless they are extended appropriately.

Concerning SLA languages, an overall conclusion is that there is a need for a new language able to express SLAs in a satisfactory way. Apart from satisfying all the above criteria of all the SLA life-cycle management activities, this language should be able to explicitly define SLs, their respective SLOs, and appropriate settlement actions when these SLs are violated or surpassed. The encoding used in this language should enable it to be platform-independent, simple, easy to use, and both machine and human understandable and processable. However, the formalism adopted should enable the analysis and the syntactic, semantic, and quality validation of the language's produced SLA specifications, either explicitly or through its transformation to another more powerful formalism. Finally, the high goal of automatability should be achieved for this new SLA language with the creation of various assisting tools or SLA management components that could be used by prospective SPs and SRs or incorporated in their management systems.

ACKNOWLEDGMENTS

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

APPENDIX

MAIN ACRONYMS

ASP: Application Service Provisioning
CG: Constraint Group
CLP: Constraint Logic Programming
FSD: Functional Service Discovery
FSM: Finite State Machine
IaaS: Infrastructure as a Service
OLA: Operation Level Agreement
QBSM: Quality-Based Service Matchmaking
QoE: Quality of Experience
QoS: Quality of Service
QSD: Quality-Based Service Description
SCL: Service Contract Language
SD: Service Description
SFAM: Service Functional Architecture Model
SFSL: Service Functional Specification Language
SL: Service Level
SLA: Service Level Agreement
SLA-MM: SLA Meta-Model
SLA-SQMM: SLA-Enabled Service Quality Meta-Model
SLO: Service Level Objective
SP: Service Provider
SQM: Service Quality Model
SQMM: Service Quality Meta-Model
SQSL: Service Quality Specification Language
SR: Service Requester
UC: Underpinning Contract
WS: Web Service

REFERENCES

Allen, P. 2006. *Service Orientation, winning strategies and best practices*. Cambridge University Press, Cambridge, UK.

Anbazhagan, M. and Nagarajan, A. 2002. Understanding quality of service for web services. IBM Developerworks website.

Andrieux, A., Dan, A., Keahy, K., Ludwig, H., and Rofrano, J. 2004. Negotiability Constraints in WS-Agreement. Technical report, GRAAP-WG. January. Submitted – Version 0.1.

Brandic, I., Buyya, R., Mattess, M., and Venugopal, S. 2009. Towards a Meta-Negotiation Architecture for SLA-Aware Grid Services. In *Proceedings of the 2nd International Workshop on Service-Oriented Engineering and Optimization (SENOPT 2008) in conjunction with HiPC 2008*. Bangalore, India, 1–9.

Brandic, I., Pllana, S., and Benkner, S. 2006. An Approach for the High-level Specification of QoS-aware Grid Workflows Considering Location Affinity. *Scientific Programming Journal 14,* 3-4, 231–250.

Buscemi, M. G. and Montanari, U. 2007. Cc-Pi: A Constraint-Based Language for Specifying Service Level Agreements. In *ESOP*. LNCS, vol. 4421. Springer, Braga, Portugal, 18–32.

Cappiello, C. 2006. *Mobile Information Systems – Infrastructure and Design for Adaptivity and Flexibility*. Springer-Verlag, Chapter The Quality Registry, 307–317.

Cappiello, C., Kritikos, K., Metzger, A., Parkin, M., Pernici, B., Plebani, P., and Treiber, M. 2008. A quality model for service monitoring and adaptation. In *Workshop on Monitoring, Adaptation and Beyond (MONA+) at the ServiceWave 2008 Conference*. Springer.

Colombo, M., Nitto, E. D., Penta, M. D., Distante, D., and Zuccalà, M. 2005. Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems. In *ICSOC*. 48–60.

Comuzzi, M., Kritikos, K., and Plebani, P. 2009. A semantic based framework for supporting negotiation in Service Oriented Architectures. In *Proceedings of 11th IEEE Conference on Commerce and Enterprise Computing (CEC09)*. IEEE Computer Society Press.

Cortés, A. R., Martín-Díaz, O., Toro, A. D., and Toro, M. 2005. Improving the Automatic Procurement of Web Services Using Constraint Programming. *Int. J. Cooperative Inf. Syst. 14,* 4, 439–468.

Cranor, L., Dobbs, B., Egelman, S., Hogben, G., Humphrey, J., Langheinrich, M., Marchiori, M., Presler-Marshall, M., Reagle, J., Schunter, M., Stampley, D. A., and Wenning, R. 2006. Platform for Privacy Preferences (P3P). Working group note, W3C. November.

Czajkowski, K., Foster, I., Kesselman, C., Sander, V., and Tuecke, S. 2002. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *In 8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2002), LNCS Vol. 2537*. 153–183.

Degwekar, S., Su, S. Y. W., and Lam, H. 2004. Constraint Specification and Processing in Web Services Publication and Discovery. In *ICWS*. IEEE Computer Society, 210–217.

Dikaiakos, M. D., Pallis, G., Katsaros, D., Mehra, P., and Vakali, A. 2009. Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing 13,* 5, 10–13. Guest Editorial.

Dobson, G., Lock, R., and Sommerville, I. 2005. QoSOnt: a QoS Ontology for Service-Centric Systems. In *EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE Computer Society, Porto, Portugal, 80–87.

Farrell, A. D. H., Sergot, M. J., Trastour, D., and Christodoulou, A. 2004. Performance Monitoring of Service-Level Agreements for Utility Computing Using the Event Calculus. In *WEC '04: Proceedings of the First IEEE International Workshop on Electronic Contracting*. IEEE Computer Society, San Diego, CA, USA, 17–24.

Fosbrook, D. and Laing, A. C. 1996. *The A-Z of Contract Clauses*. Sweet & Maxwell.

Frølund, S. and Koistinen, J. 1998. 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 5,* 4, 179–202.

Frutos, H. M., Kotsiopoulos, I., Gonzalez, L. M. V., and Merino, L. R. 2009. Enhancing Service Selection by Semantic QoS. In *ESWC*. 565–577.

Georgakopoulos, D. and Papazoglou, M. P. 2008. *Service-Oriented Computing*. Cooperative Information Systems. MIT Press.

Giallonardo, E. and Zimeo, E. 2007. More Semantics in QoS Matching. In *International Conference on Service-Oriented Computing and Applications*. IEEE Computer Society, Newport Beach, CA, USA, 163–171.

Grefen, P. and Angelov, S. 2002. On τ-, μ-, π-, and ϵ-contracting. In *WES*. Vol. 2512. Springer, Toronto,Canada, 68–77.

Grosof, B. N. and Poon, T. C. 2004. SweetDeal: Representing Agent Contracts with Exceptions Using Semantic Web Rules, Ontologies, and Process Descriptions. *Int. J. Electron. Commerce 8,* 4, 61–97.

Hoffner, Y., Field, S., Grefen, P., and Ludwig, H. 2001. Contract-driven creation and operation of virtual enterprises. *Computer Networks 37*, 111–136.

Hwang, C. and Yoon, K. 1981. Multiple Criteria Decision Making. *Lecture Notes in Economics and Mathematical Systems*.

ISO/IEC 2001. *ISO/IEC 9126-1 Software Engineering. Product Quality - Part 1: Quality model*. ISO/IEC.

Jiang, Y., Shao, W., Zhang, L., Ma, Z., Meng, X., and Ma, H. 2004. On the classification of umls meta model extension mechanism. In *UML*. 54–68.

Keller, A. and Ludwig, H. 2003. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management 11,* 1, 57–81.

Kritikos, K. 2008. QoS-based Web Service Description and Discovery. Phd thesis, Computer Science Department, University of Crete, Heraklion, Greece. December.

Kritikos, K. and Plexousakis, D. 2006. Semantic QoS Metric Matching. In *ECOWS '06: Proceedings of the European Conference on Web Services*. IEEE Computer Society, Zurich, Switzerland, 265–274.

Kritikos, K. and Plexousakis, D. 2009. Requirements for QoS-based Web Service Description and Discovery. *IEEE Transactions on Services Computing*. accepted.

Lamanna, D. D., Skene, J., and Emmerich, W. 2003. SLAng: A Language for Defining Service Level Agreements. In *FTDCS 2003: Proceedings of the 9th IEEE International Workshop on Future Trends of Distributed Computing Systems*. IEEE Computer Society, San Juan, Puerto Rico.

Lee, K., Jeon, J., Lee, W., Jeong, S.-H., and Park, S.-W. 2003. Qos for web services: Requirements and possible approaches. World Wide Web Consortium (W3C) note.

Linington, P. F., Milosevic, Z., Cole, J., Gibson, S., Kulkarni, S., and Neal, S. 2004. A unified behavioural model and a contract language for extended enterprise. *Data & Knowledge Engineering 51,* 1, 5–29.

Liu, Y., Ngu, A. H. H., and Zeng, L. 2004. QoS computation and policing in dynamic web service selection. In *WWW (Alternate Track Papers & Posters)*, S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, Eds. ACM, New York, NY, USA, 66–73.

Ma, Q., Wang, H., Li, Y., Xie, G., and Liu, F. 2008. A Semantic QoS-Aware Discovery Framework for Web Services. *ICWS: IEEE International Conference on Web Services*, 129–136.

Mabrouk, N. B., Georgantas, N., and Issarny, V. 2009. A Semantic End-to-End QoS Model for Dynamic Service Oriented Environments. In *PESOS Workshop at ICSE 2009*. IEEE.

Maximilien, E. M. and Singh, M. P. 2002. Conceptual model of web service reputation. *SIGMOD Rec. 31,* 4, 36–41.

Maximilien, E. M. and Singh, M. P. 2004. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing 8,* 5, 84–93.

Mens, T. and Lanza, M. 2002. A graph-based metamodel for object-oriented software metrics. *Electr. Notes. Theor. Comput. Sci. 72,* 2.

Milosevic, Z. and Dromey, R. G. 2002. On Expressing and Monitoring Behaviour in Contracts. In *EDOC 2002: 6th IEEE International Conference on Enterprise Distributed Object Computing*. IEEE Computer Society, Lausanne, Switzerland, 3–14.

Molina-Jimenez, C., Shrivastava, S., Solaiman, E., and Warne, J. 2003. Contract Representation for Run-time Monitoring and Enforcement. In *CEC 2003: IEEE International Conference on E-Commerce Technology*. IEEE Computer Society, Newcastle upon Tyne, UK, 103–110.

Müller, C., Cortés, A. R., and Resinas, M. 2008. An Initial Approach to Explaining SLA Inconsistencies. In *ICSOC 2008: Proceedings of the 6th International Conference on Service-Oriented Computing*. Lecture Notes in Computer Science, vol. 5364. Springer, 394–406.

Muller, N. J. 1999. Managing Service Level Agreements. *International Journal of Network Management 9,* 3, 155–166.

Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., and Granqvist, H. 2007. WS-Trust specification, http://www.ibm.com/developerworks/webservices/library/specification/ws-trust/. In *Technical report*. OASIS Working Draft.

Nejdl, W., Olmedilla, D., and Winslett, M. 2004. PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web. In *SDM 2004: Proceedings of the VLDB 2004 International Workshop on Secure Data Management in a Connected World*. LNCS, vol. 3178. Springer, Toronto, Canada, 118–132.

Nessi Open Framework. 2009. Quality Model for NEXOF-RA Pattern Designing. Tech. rep.

Oldham, N., Verma, K., Sheth, A., and Hakimpour, F. 2006. Semantic WS-Agreement Partner Selection. In *WWW '06: Proceedings of the 15th International conference on World Wide Web*. ACM Press, Edinburgh, Scotland, 697–706.

Oren, N., Preece, A., and Norman, T. 2005. Service level agreements for semantic web agents. In *AAAI Fall Symposium Series*. AAAI, Virginia,USA.

O'Sullivan, J., Edmond, D., and ter Hofstede, A. 2002. What's in a service? Towards Accurate Description of Non-Functional Service Properties. *Distributed and Parallel Databases 12,* 2-3, 117–133.

Paoli, F. D., Palmonari, M., Comerio, M., and Maurino, A. 2008. A Meta-model for Non-functional Property Descriptions of Web Services. In *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*. IEEE Computer Society, Beijing, China, 393–400.

Parkin, M., Badia, R. M., and Martrat, J. 2008. A Comparison of SLA Use in Six of the European Commissions FP6 Projects. Tech. rep., TR-0129, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence. April.

Paschke, A. 2005. RBSLA: A declarative Rule-based Service Level Agreement Language based on RuleML. In *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06)*. IEEE Computer Society, Vienna, Austria, 308–314.

Paschke, A. and Schnappinger-Gerull, E. 2006. A Categorization Scheme for SLA Metrics. In *Service Oriented Electronic Commerce: Proceedings zur Konferenz im Rahmen der Multikonferenz Wirtschaftsinformatik*. LNI, vol. 80. GI, Passau, Germany, 25–40.

Ran, S. 2003. A model for web services discovery with QoS. *SIGecom Exch. 4,* 1, 1–10.

Redman, T. C. 1997. *Data Quality for the Information Age*. Artech House, Inc., Norwood, MA, USA. Foreword By-A. Blanton Godfrey.

Rossi, F., van Beek, P., and Walsh, T. 2006. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA.

Sabata, B., Chatterjee, S., Davis, M., Sydir, J., and Lawrence, T. 1997. Taxonomy for QoS Specifications. In *Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on*. 100–107.

Sahai, A., Durante, A., and Machiraju, V. 2002. Towards Automated SLA Management for Web Services. Tech. Rep. HPL-2001-310, HP Laboratories, Palo Alto, CA. July.

Sakellariou, R. and Yarmolenko, V. 2008. *High Performance Computing and Grids in Action*. Chapter Job Scheduling on the Grid: Towards SLA-Based Scheduling.

Shen, H. and F.Hong. 2006. An Attribute-Based Access Control Model for Web Services. In *Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2006)*. 74–79.

Skene, J. 2007. Language Support for Service-Level Agreements for Application-Service Provision. Phd thesis, Department of Computer Science, University College London, London, UK. November.

Skogsrud, H., Benatallah, B., and Casati, F. 2004. Trust-Serv: Model-Driven Lifecycle Management of Trust Negotiation Policies for Web Services. In *Proc. 13th World Wide Web Conf.*

Strong, D. M., Lee, Y. W., and Wang, R. Y. 1997. 10 Pitholes in the Road to Information Quality. *Computer 30,* 8, 38–46.

Tebbani, B. and Aib, I. 2006. GXLA a Language for the Specification of Service Level Agreements. In *AN 2006: Proceedings of the First International IFIP TC6 Conference on Autonomic Networking*. Lecture Notes in Computer Science, vol. 4195. Springer, Paris, France, 201–214.

The OASIS Group. 2005. Quality Model for Web Services. Tech. rep., The Oasis Group. September.

The OMG Group. 2005. UML™ Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. Tech. Rep. ptc/2005-05-02, The OMG Group. May.

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

Toktar, E., Pujolle, G., Jamhour, E., Penna, M. C., and Fonseca, M. 2007. An XML Model for SLA Definition with Key Indicators. In *IPOM 2007: 7nth IEEE International Workshop on IP Operations and Management*. LNCS, vol. 4786. Springer, San Jose, USA, 196–199.

Tondello, G. F. and Siqueira, F. 2008. The QoS-MO ontology for semantic QoS modeling. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*. ACM, Fortaleza, Ceara, Brazil, 2336–2340.

Tosic, V., Esfandiari, B., Pagurek, B., and Patel, K. 2002. 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*. Springer-Verlag, Toronto, Ontario, Canada, 237–247.

Tosic, V., Ma, W., Pagurek, B., and Esfandiari, B. 2003. 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.

Tosic, V. and Pagurek, B. 2005. On comprehensive contractual descriptions of web services. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*. IEEE Computer Society, Hong Kong, 444–449.

Tosic, V., Pagurek, B., and Patel, K. 2003. WSOL – A Language for the Formal Specification of Classes of Service for Web Services. In *ICWS*, L.-J. Zhang, Ed. CSREA Press, Las Vegas, Nevada, USA, 375–381.

Tran, V. X. 2008. WS-QoSOnto: A QoS Ontology for Web Services. In *SOSE 2008: Proceedings of the 4th IEEE International Workshop on Service-Oriented System Engineering*. IEEE Computer Society, Jhongli, Taiwan, 233–238.

Truong, H.-L., Samborski, R., and Fahringer, T. 2006. Towards a Framework for Monitoring and Analyzing QoS Metrics of Grid Services. In *International Conference on e-Science and Grid Computing*. IEEE Computer Society Press, Amsterdam, The Netherlands.

TrustCoM Consortium. 2007. TrustCom Framework V4 – Appendix A: Profiles. Report Deliverable D63, European Union. January.

Unger, T., Leymann, F., Mauchart, S., and Scheibler, T. 2008. Aggregation of Service Level Agreements in the Context of Business Processes. In *EDOC '08: Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference*. IEEE Computer Society, Munich, Germany, 43–52.

Wang, X., Vitvar, T., Kerrigan, M., and Toma, I. 2006. A QoS-Aware Selection Model for Semantic Web Services. In *ICSOC*, A. Dan and W. Lamersdorf, Eds. Lecture Notes in Computer Science, vol. 4294. Springer, 390–401.

Welty, C., Kalra, R., and Chu-Carroll, J. 2003. Evaluating ontological analysis. In *Proceedings of the ISWC-03 Workshop on Semantic Integration*.

WS-AGREEMENT. 2003. WS-Agreement Framework. `https://forge.gridforum.org/projects/graap-wg`.

Yang, Z., Zhang, D., and Ye, C. 2006. Ontology Analysis on Complexity and Evolution Based on Conceptual Model. In *DILS*, Springer, Ed. Vol. 4075. 216–223.

Yi, T., Wu, F., and Gan, C. 2004. A comparison of metrics for uml class diagrams. *SIGSOFT Softw. Eng. Notes 29,* 5, 1–6.

Yoa, H., Orem, A. M., and Etzkorn, L. 2005. Cohesion metrics for ontology design and application. *Journal of Computer Science 1,* 1, 107–113.

Zhou, C., Chia, L.-T., and Lee, B.-S. DAML-QoS Ontology for Web Services. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services, year = 2004, isbn = 0-7695-2167-3, pages = 472–479, doi = http://dx.doi.org/10.1109/ICWS.2004.44, publisher = IEEE Computer Society, address = San Diego, CA, USA*.