A Survey on Service Quality Description

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

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 either a software or a software-support (i.e., infrastructural) service which is 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 performed 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.

1. INTRODUCTION

A service is an action performed by an entity (the provider) on behalf of another one (the requester) [O'Sullivan et al. 2002]. Through the interaction between these two entities, which is called *service provisioning* and involves various phases, there is a transfer of value from the provider to the requester or recipient. Depending on the service nature and the means or channels it is available, different service types can be identified. For instance, the drawing of a bank cheque is a physical service which is available only on the bank counter so it requires the requester's physical presence on a specific place and time to be invoked and delivered. Web services are instead autonomous software systems available over the Internet. This paper focuses on software and software-support (i.e., infrastructural) services which are available on any network or electronic channel type. Thus, in this survey the word "service" will have this designated meaning and any other service type will be excluded from the analysis and discussion.

Service orientation has emerged lately as a paradigm facilitating 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.

An 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). For instance, in the e-business area, services can be selected dynamically and composed in added-value new services, where the composite service components 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 telecom and energy provider ones, also require interoperability of very complex systems to guarantee service delivery. Multimedia and multichannel applications require composing and synchronizing 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 considered is a simplified version of the one proposed in [Lamanna et al. 2003]. It follows a three-layer architecture. The first layer, the *Application Layer*, contains business or user-oriented applications which may use services to fulfill a part or their whole functionality. The *Service Layer* is the next layer containing services with an electronic interface which are used to build or populate the applications. Various infrastructures, which belong to the *Infrastructure Layer*, host these services and are responsible of managing the services underlying resources for communication, transactions, security and so forth, e.g. through a platform virtualization environment. Every architecture component can be offered as a service to the same or other component types. For instance, Infrastructure as a Service (IaaS) is the delivery of computing infrastructures as a service which fulfils hosted application or service needs [Dikaiakos et al. 2009].

Since they are intended to be discovered and used by other applications across the Web, services need to be described and understood both in terms of functional capabilities and service quality properties. *Service Quality* is a combination of several qualities or properties (e.g., availability, security, response time) of a service, and can be generally seen as a important factor in distinguishing the success of service providers (REF). The service quality description is the main driver in selecting the best service among a set of functionally equivalent ones. Besides, quality is used to define a contract, i.e., a 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, enforcing it by taking appropriate adaptation actions, such as increasing the underlying service resources, substituting or recomposing the faulty service, and determining which settlement actions apply based on the way the service was executed, such as the final cost or penalties to be paid by the service requester or provider, respectively, and negotiations for SLA termination.

In all previous cases, a prerequisite for using service quality is its proper, precise, and rigorous description, covering all possible service life cycle phases. In this paper, the term *quality document* is used to denote the QoS description of a given service. This term will be used in a generic way as a description of quality, and all issues related to managing such a document in a specific system architecture will not be addressed. Even not referring to specific architectural solutions, it will be shown that the problem of being able to write quality documents is far from solved. In the literature, several approaches have been proposed, and there is no commonly agreed way to specify QoS. This paper aims to systematically and comparatively review these approaches 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.

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, a definition of service quality is provided along with a small analysis of the service life cycle, while also

a general classification of quality models is presented, which is then used in the remaining paper sections. Section 3 presents a case study of the usage of service quality in Cloud Computing. According to the quality model classification, the state of the art on quality models is analyzed in Section 4, on quality meta-models in Section 5, and on Service Level Agreements in Section 6. In each section, a set of comparison characteristics are defined and then exploited to compare the approaches according to level of satisfaction of these characteristics. The last section analyzes interesting topics for further investigation.

2. SERVICE LIFE CYCLE AND QUALITY

By relying on the definition provided in [Kritikos and Plexousakis 2009], which mainly applies to software services, service quality is defined as a set of non-functional attributes of those contextual entities that are considered relevant to the service-client interaction, including the service and the client, that bear on the service ability to satisfy stated or implied needs. Moreover, service quality can be classified as Quality of Service (QoS) and Quality of Experience (QoE). QoS includes quality attributes that can be objectively measured (like *execution time*) and are typical constituents of Service Level Agreements (SLAs). QoE includes quality attributes that can be subjectively measured (e.g. *reputation*, *usability*) and reflects the perception of individuals or groups of service users.

The distinguishing feature of service quality with respect to functionality is its dynamicity. In particular, the values of some service quality attributes can vary without impacting the core service function which remains constant most of the time during the service's lifetime. Based on this reason, service quality can be used during service discovery to distinguish between many functionally-equivalent services. Moreover, it can be monitored and controlled during service provisioning to cater for the increasingly high user expectations with respect to the service's performance and other types of quality attributes. In fact, service quality can play a significant role during several phases of the service life-cycle [Kritikos and Plexousakis 2009]. This is evidenced in the reference service life-cycle of Fig. 1. Below each activity of this life-cycle is shortly analyzed along with the type of quality model it exploits:

—Advertisement: requesters and providers publish or exchange quality requests and quality offers, respectively. Such quality documents are called *Quality-Based Service Descriptions* (QSDs).

—Matchmaking: The QSDs are matched to examine 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 are exchanged between service providers and requesters. The possible agreement between the parties involved leads to the definition of another quality document, the 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.

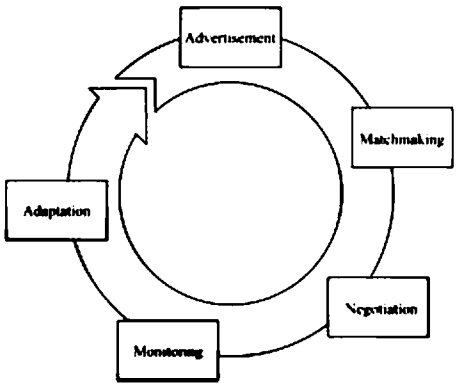

Fig. 1. Service life-cycle.

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

Apart from QSDs and SLAs, another document type is a Service Quality Model (SQM). All document types have been introduced for particular reasons, while some document types are used as building blocks of other types. Moreover, each type is exploited in specific activities of the service life-cycle. In particular, Service Quality Models (SQMs) have been used to describe concrete quality properties which can be exploited by other quality document types to express service quality capabilities/requirements or service levels. QSDs have been used to express service quality capabilities and requirements as a set of constraints on quality attributes and metrics. Such descriptions contain all the appropriate information for supporting the service matchmaking and negotiation activities. Finally, SLAs have been used not only to express the service levels in which a service can execute but also other information that is suitable for supporting the service provisioning activities. As a service level description of an SLA is actually a QSD (i.e. a set of service quality constraints), SLAs are actually built on top of the other document types.

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 4, 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 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.

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 Requests 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 Service Descriptions (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. Thus, QSDs describe all the appropriate information that is required for matchmaking and negotiating service quality. In this way, they are used in the respective service life-cycle activities. However, they are not used further in the service life-cycle, as they do not contain all the appropriate information that is required for supporting the rest of the service life-cycle activities, which mainly concern the service provisioning.

To this end, SLAs have been introduced to close this gap. An SLA is an important aspect of a contract for IT services that includes the set of QoS guarantees and the obligations of the various parties (see definition in [Keller and Ludwig 2003]). QoS guarantees are widely known as Service Level Objectives (SLOs) and are expressed as conditions on one or more QoS metrics, thus indicating 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, or degradation/upgrade levels if the agreed SL is violated/surpassed. The party obligations are usually expressed as action guarantees (e.g. compensation, recovery, or management actions) to be performed when a given precondition is met (e.g. a violation occurs). Other important SLA components are the organizational ones which correspond to information concerning service monitoring and reporting.

As can be seen from the above analysis, SLAs contain more information than QSDs in terms of supporting the service provisioning activity. Moreover, there is not any uniform and common quality document to be used across all the activities. This can be also observed in the reference service life-cycle. 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 meta-models and languages for describing service quality exist. In this survey, we firstly group them in two main categories. Figure 2 represents the types of quality documents, their meta-models, and the various relationships involved between all of these entities.

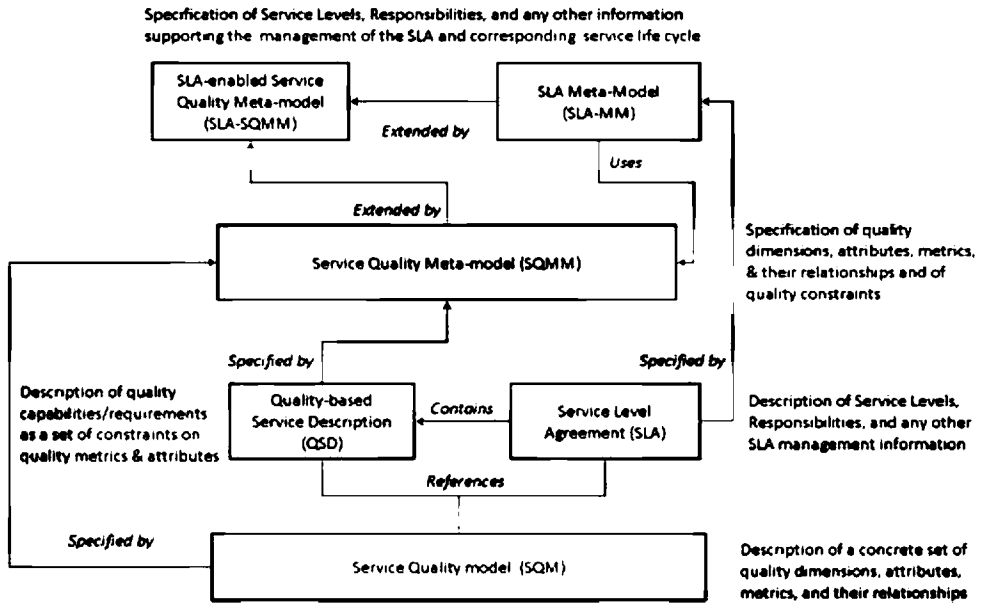

Fig. 2. The main types of quality documents, their meta-models, and the inter-relationships.

At the first level, the *Service Quality Meta-Models* (SQMMs) 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 *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 5. 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 quality attribute set. In case SRs and SPs change the relevant quality attribute set, 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 second level, the *SLA Meta-Models* (SLA-MMs) are considered. In this case, as it will be discussed in Section 6, the approaches involved allow the definition of *SLA*s and *SLA Templates* between the interacting parties. Since the agreement terms 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, the existence of the three following cases is high-

lighted: a) there are SQMMs, called *SLA-enabled SQMMs* (SLA-SQMMs), that can define SLA specifications apart from QSDs; b) SLA-MMs may use one or more SQMMs 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 considered, when comparing existing SLA languages, which concern the definition of the contract terms and various other information that may be used to support the service life-cycle activities.

In general, meta-models are used to define a language's abstract syntax. 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 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.

3. CASE-STUDY

Cloud computing[1] is an important step on the road to make distributed computing resources a *utility*, accessible at any time from any place. The advantage of cloud computing is that, like accessing electricity, gas or water utilities, it requires little or no cost to start accessing the utility provider's service. However, as with all utilities, cloud computing users expect to be given guarantees about how and when the service will be provided and an indication of the charges applied to its access and use. In the presence of these guarantees potential users can decide whether the service meets their requirements, compare the service's properties to other services, and determine if the value of the service is worth the advertised cost.

Cloud computing is a form of service-oriented computing (SOC). In SOC guarantees on the properties of computing services are known as QoS information. Such information can be used to describe a number of aspects of the service properties, such as the type, performance, and reliability of the hardware used.

For example, Figure 3 shows how a cloud provider supplies its cloud resources, e.g., as Infrastructure as a Service (IaaS), Platform as a Service (Paas) or Software as a Service (SaaS), together with a set of Service Level Properties.

SRs form service requests containing their QoS preferences and constraints and, if these are within the capabilities of the Cloud Provider, they can form SLAs, or contracts for service provision. The advantage of using and agreeing an SLA between the provider and consumer is that it allows the provider to organize its internal resources for maximum efficiency and for the consumer to be assured that the service will operate according to the details given in the SLA; through evaluation of the SLOs against the relevant metrics of the service's performance, they can

[1]A list of cloud computing success stories can be found here: http://cloudtp.com/cloud-computing/cloud-computing-success-stories

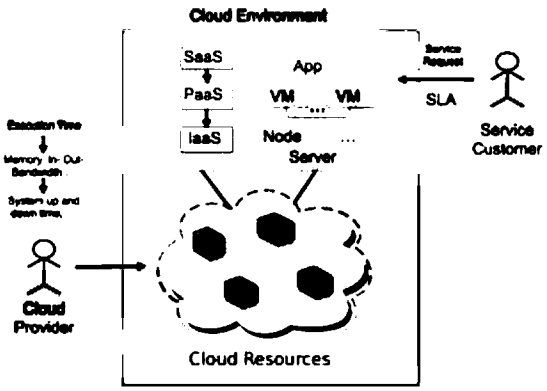

Fig. 3. The way cloud resources are supplied by a cloud provider.

determine if the SLA has been met. Such SLAs are usually described by the most widely used SLA languages such as WS-Agreement.

The presence of QoS information in this simple cloud computing scenario shows how the the cloud platform quality can be advertised and guaranteed for the SR.

4. SERVICE QUALITY MODELS

4.1 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 services are considered as standalone software modules, then their quality can be determined by the attributes that traditionally characterize software quality and, thus, by the attributes defined in the ISO 9126 model [ISO/IEC 2001]. However, this model has to be adapted accordingly to capture the peculiarities deriving from the service intended use, i.e., their composition to build service-based applications 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 external attributes). 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 external quality attributes proposed for traditional software are applicable, e.g. *privacy*, *security*, *performance*, and *reliability*. 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 software product characteristics 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 select service components that are more stable than others to build at design- or run-time a stable composite service which will be used many times as it is without being updated.

Thus, 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 service types like the infrastructural ones. For this reason, different contributions can be found in the literature which propose various SQMs. These SQMs' structure 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 needed only for measurable quality attributes [Kritikos and Plexousakis 2009].

The most referenced QoS categories among all SQMs are the *Performance*, *Security*, *Dependability*, and *Configuration*, which usually contain specific quality attributes as it will be shown in Section 3.3. As there is a differentiation on how SQMs categorize the rest of the service quality attributes, it was decided not to evaluate SQMs in terms of specific categories but generally on the extensiveness of their quality attribute categorization. Moreover, SQMs were evaluated also on the level of detail in their categorization in order to inspect the SQM richness. Section 3.3 evaluates SQMs in terms of the quality attributes they contain in order to distinguish which attributes are the most common ones and to provide a proof for the correct evaluation of the SQM richness.

4.2 Methodology and Analysis

The most significant SQMs proposed for services and their applications have been collected. Only generic SQMs were considered and not specialized models proposed for security, data, and network aspects. This is because 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-based ones.

A set of comparison criteria have been selected in order to analyze and compare the different SQM approaches according to the type and value of the information that they contain. A summary of these criteria is shown in Table I, while their thorough presentation is provided later on in this section. The evaluation results of the examined SQMs according to the comparison criteria are shown in Table II. In this table, the SQMs are sorted according to their chronological order (from the oldest to the newest). In this way, interesting conclusions concerning trends in SQM modeling can be drawn.

The UML diagram depicted in Figure 4 shows an ideal meta-model for expressing SQMs that satisfy all the comparison criteria set. The concepts and associations marked with red show the common conceptual elements among all SQMs. This meta-model could be that part of an SQMM related to the appropriate and rich quality attribute description.

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

4.2.1 *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

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(s) does a QoS attribute refer to?
Association with Assessment Guidelines	Are there any assessment guidelines associated to a QoS attribute?
Metric Identification	Is there any metric associated to a QoS attribute?

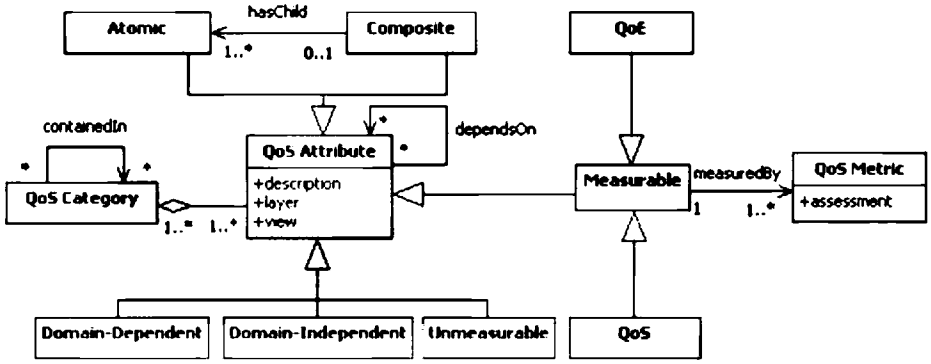

Fig. 4. Ideal meta-model for SQMs.

category number the higher the comprehensiveness of the approach. Categories also improve the model's readability. In fact, a classification enables users to better explore the model and optimize the search activity. Categories that are defined in many SQMs are: *Performance*, *Security*, *Dependability*, and *Configuration Management*. A flat model would contain only one quality category while an extensive one can contain up to 9 or 10 categories. Grades of extensiveness: *flat* (1 category), *fair* (2-4 categories), *good* (5-7) categories and *extensive* (8-10) categories.

The evaluation results are presented in the second column of Table II. The majority of SQMs uses categories to classify the attributes and improve the model's understandability. Moreover, there is a balance between all the criterion-specific

Research Approach	**Extensiveness**	**Detail Level**	**Domain Dep/ncy**	**SP/SR View**	**QoS & QoE**	**Composite & Atomic**	**Dep/ncy Types**	**Layer**	**Metric**	**Ass/ent Guid/es**
[Sabata et al. 1997]	fair	fair	domain ind.	SP	QoS	both	no	service	partial	fair
[Ran 2003]	good	fair	domain ind.	SP	QoS	both	no	all	partial	fair
[Colombo et al. 2005]	flat	fair	domain ind.	both	both	atomic	no	serv. & appl.	complete	fair
[The OASIS Group 2005]	fair	fair	domain ind.	both	QoS	both	quant.	serv. & appl.	complete	good
[Cappiello 2006]	extensive	high	domain ind.	both	both	both	no	all	partial	none
[Truong et al. 2006]	good	fair	domain ind.	SP	QoS	both	quant.	serv. & infr.	partial	fair
[Brandic et al. 2006]	good	fair	domain ind.	SP	QoS	both	quant.	serv. & infr.	partial	fair
[Sakellariou and Yarmolenko 2008]	good	good	both	both	QoS	both	no	serv. & infr.	partial	fair
[Cappiello et al. 2008]	extensive	high	both	both	both	both	quant.	all	none	none
[Frutos et al. 2009]	good	high	domain ind.	SP	QoS	both	quant.	all	partial	none
[Nessi Open Framework 2009]	fair	fair	domain ind.	both	both	both	quant.	serv. & appl.	none	none
[Kritikos and Plexousakis 2009]	extensive	high	both	both	both	both	no	all	partial	fair
[Mabrouk et al. 2009]	extensive	good	both	both	both	both	quant.	all	partial	fair

Table II. Comparison of Research Approaches Providing Service Quality Models

SQM partitions (apart from the *flat* partition which has only one approach) with the partition corresponding to good extensiveness to have a very small precedence. Finally, by observing the last nine values of the corresponding criterion column, the trend that SQMs are improving according to this aspect (with the exception of the approach in [Nessi Open Framework 2009]) can be revealed.

4.2.2 *Level of Detail in Quality Categories.* This criterion is used to inspect if each category contains a sufficient quality attribute number. In this way, the higher the number the higher 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 it is *high*. In this way, we can generalize to express the *level of detail* of the whole SQM. Thus, we have defined 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 evaluation results are presented in the third column of Table II. 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. Finally, approaches with higher level of detail than fair are starting to appear after 2005. This means that SQMs have slightly improved over the years according to this aspect.

4.2.3 *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 *response time* and *availability* (domain-independent) attributes. On the other hand, domain-dependent quality attributes 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 evaluation results are presented in the fourth column of Table II. 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 domain [Sakellariou and Yarmolenko 2008; Kritikos and Plexousakis 2009; Mabrouk et al. 2009] or some quality attributes used in some domains according to a particular context [Cappiello et al. 2008]. The latter four approaches have been proposed recently.

4.2.4 *Consideration of Service Provider and Requester Views.* This criterion considers the relevance degree of the different quality attributes with respect to the SP and SR. Some quality attributes are particularly relevant for SPs, such as *availability* and *response time*. Other quality attributes are important to SRs, such

as *response time* and *usability*. Thus, some quality attributes are both important to SPs and SRs, so they belong to their *common* view, while other attributes 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 quality attribute set that can be used to express both SP capabilities and SR requirements. Thus, this criterion's evaluation for a specific SQM would be: *SP* (if the SQM contains *SP-specific* and *common* quality attributes), *SR* (if it contains *SR-specific* and *common* attributes), and *both* (if it contains *common*, *SP-specific*, and *SR-specific* attributes).

The evaluation results are presented in the fifth column of Table II. More than half of the SQMs consider quality attributes that correspond to both provider and requesters view. So, researchers have understood the need of expressing both views in an SQM. Moreover, the evaluation results of this criterion are in accordance with those of the previous one. 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 providers are more concerned about having their services used across many application domains so they focus more on those quality attributes that are generic and tend to distinguish their services 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. Finally, it can be observed that while in the past most SQMs were capturing only the SP view, this situation has changed recently as the more recent SQMs (from 2006 and on) tend to cover both views.

4.2.5 *QoS and QoE Consideration.* QoE attributes certainly reflect the requester view. However, two main questions arise that may be addressed by the evaluation of this criterion and the observation of the evaluation results 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 attribute types?". Thus, these two questions actually concern which views do the QoS attributes reflect. Obviously, both sets of attributes are important and should be represented by an SQM. So, the evaluation of this criterion for a specific SQM would be: *QoS* (only QoS), *QoE* (only QoE), and *both*.

The evaluation results are presented in the sixth column of Table II. It is easily noticeable that there is a balance between the SQMs that contain only QoS attributes with those that contain also QoE attributes. Moreover, the more recent SQMs tend to cover both attribute types. This means that SQM modelers have understood the importance of modelling both QoS and QoE attributes.

By inspecting this criterion's results and those of the previous one, some other interesting facts can be inferred. First, when an SQM covers only the SP view, then it covers only QoS attributes. This means that an SP considers the QoS attributes

as more important. 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 an SQM additionally covers the SR view, then it covers either QoS or both QoS and QoE attributes. This signifies that the SR-view corresponds to both attribute types. On one hand, the SR-view domain-independent attributes will tend to be QoE attributes because these attributes can be assessed differently from users across the various application domains, as 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 because in a specific application domain the requirements and the service usage are specific or 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 be assessed by the same objective way by users that have similar expectations.

4.2.6 *Atomic and Composite Quality Attribute Inclusion.* Some attributes are *composite* and can be computed by evaluating other attribute values. For example, *response time* (the parent) is a composite attribute since it can be assessed by evaluating *latency* and *network delay* (its children). Other attributes, like *execution time*, are *atomic* as they do not rely on any attribute. The composability aspect is important as it can be used later by meta-models to capture this inherent relationship type 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 attributes. In addition, even if this relationship is captured by a simple connection (symbolic or phrasal), it can be useful in service monitoring to check for example if the increase in a child attribute value causes an increase in the parent attribute value. So it is important to inspect if both attribute types and parent-child relationships are captured. Thus, this criterion's evaluation for an SQM would be: *atomic* (only atomic attributes are included), *composite* (only composite attributes are included), and *both* (both attribute types are included along with a connecting relationship).

The evaluation results are presented in the seventh column of Table II. Most of the SQMs contain both atomic and composite quality attributes along with a connecting relationship. In addition, as all SQMs that have been proposed after 2005 contain both attribute types, there is a trend of SQM improvement with respect to this aspect. Finally, by inspecting the evaluation of the SQM of [Colombo et al. 2005] according to this and the first two criteria, it can be observed that this SQM contains only atomic quality attributes, while its category extensiveness is flat and its level of detail is fair. Indeed, 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 but only QoS attributes that can be immediately measured by specific metrics without introducing more higher measurement levels.

4.2.7 *Types of Dependencies.* There can be two dependency types between quality attributes: *quantitative* and *qualitative*. The former are expressed by mathe-

matical formulas or constraints and a specific subclass of them are the composite attribute derivation formulas included in parent-child relationships, while the latter are expressed symbolically or descriptively. An qualitative dependency example 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 it must be inspected if concrete dependencies between quality attributes exist in an SQM without considering which modeling constructs are used, as SQMs and not SQMMs are evaluated. Thus, this criterion's evaluation for a specific SQM would be: *no* (no dependencies are expressed), *quant.* (only quantitative dependencies are expressed), *qual.* (only qualitative dependencies are expressed) and *both* (both dependency types are expressed).

The evaluation results are presented in the eighth column of Table II. Only qualitative dependencies are addressed by the SQMs. In addition, all of these quantitative dependencies concern composite attribute derivation formulas. Finally, by comparing the evaluation results with those of the previous criterion, it can be inferred that the majority of the SQMs that specify both composite and atomic quality attributes also specify the way the composite attributes are produced from the atomic ones.

4.2.8 *Layer Designation.* Our reference model consists of three layers: *application*, *service* and *infrastructure*. The *application* and *service* layers usually have an identical attribute set 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 quality attribute set with respect to the other two layers. So, as it is important to clarify to which service layers an SQM refers to, this criterion's evaluation 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 evaluation results are presented in the ninth column of Table II. As all SQMs are built by having specialized focus on the components of the service layer, i.e. the software services, they contain service-layer quality attributes. Almost half of the SQMs also contain either infrastructure-layer or application-layer attributes, while the same result applies for SQMs that have adopted a holistic approach. Thus, apart from the initial SQM that contained solely service-layer quality attributes, the researchers have quickly understood the need of covering quality attributes from other layers apart from the service one so as to be able to characterize the quality of all types of services.

4.2.9 *Metric Identification.* This criterion is used to inspect if SQMs contain quality metrics used to measure quality attributes. Metrics are entities that encapsulate all appropriate measurement details of an attribute such as measurement values, units, formulas, and schedules. However, a metric's measurement formula or assessment algorithm was decided to be used as a separate comparison criterion from the current one because 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 it must be assessed first which SQMs associate specific

metrics to attributes and then the existence or not of assessment guidelines for the incorporated metrics. Thus, this criterion's evaluation for each SQM will take the following discrete values: *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 attributes are associated with at least one corresponding metric.

The evaluation results are presented in the tenth column of Table II. More than half of the SQMs associate metrics to a subset of the contained quality attributes, while only two SQMs are complete in 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 widely used (in QSDs and SLAs) attributes. Finally, two SQMs do not associate metrics to attributes, which significantly limits their usage in specific service quality description and matchmaking scenarios.

4.2.10 *Association with Assessment Guidelines.* An SQM may provide algorithms to assess the quality attributes that it defines. This criterion's evaluation 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 evaluation results are presented in the eleventh column of Table II. More than half SQMs just specify some guidelines for each metric, while one third of the considered SQMs do not provide metric assessment guidelines. Thus, as the latter SQMs cannot be used in the monitoring and assessment service life-cycle activity, their use is limited mostly in the advertisement and matchmaking activities. Only one approach provides precise assessment algorithms for all defined metrics and associates metrics to all its contained attributes. So, this approach is suitable for annotating QSDs and SLAs which can be used across all service life-cycle activities. However, it can be used in specific scenarios as it contains a small set of domain-independent quality attributes and not domain-dependent ones. Finally, by correlating the evaluation results of this criterion with those of the previous one, it can be inferred that SQMs, which associate metrics to some or all of their quality attributes, do provide for them simple assessment rules which could be further used to create precise assessment algorithms.

4.3 Overall Analysis

Based on the above analysis, none 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 an SQM's extensiveness, structure, and generality, four approaches 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 an SQM's attribute assessment and applicability, the approach in [Colombo et al. 2005] can be distinguished. Moreover, all approaches have the worst behavior with respect to the "Types of Dependencies" criterion. Thus, a new SQM is needed combining the characteristics of the best approaches in the above two criteria partitions and describing all the possible but realistic inter-attribute dependencies.

Apart from the criteria-based analysis, the frequency of the service quality categories and attributes across all SQMs was assessed in order to distinguish the most

Service Quality Category	Service Quality Attribute	Approach Reference												
		Sabata et al. 1997	Ran 2003	Colombo et al. 2005	The OASIS Group 2005	Cappiello 2006	Truong et al. 2006	Brandic et al. 2006	Sakellariou and Yarmolenko 2008	Cappiello et al. 2008	Frutos et al. 2009	Nessi Open Framework 2009	Kritikos and Plexousakis 2009	Mabrouk et al. 2009
Performance	Response Time		X	X	X	X	X			X	X	X	X	X
	Processing Time					X	X	X	X	X			X	
	Latency		X	X		X	X		X	X	X		X	
	Timeliness	X		X		X			X					
	Precision	X												
	Throughput		X	X	X	X			X	X	X	X	X	X
-	Availability		X	X	X	X	X		X	X	X	X	X	X
-	Accessability				X	X	X			X	X		X	X
-	Accuracy	X		X		X	X			X	X		X	
-	Reliability		X	X		X	X			X	X	X	X	
-	Capacity		X				X		X	X	X		X	
-	Believability						X							
-	Maintainability			X			X					X	X	
-	Relative Importance	X												
-	Complexity			X										
-	Customer Service			X										
-	Dependability			X		X				X		X	X	
-	Stability		X	X	X					X			X	X
-	Trust			X		X								
-	Understandability			X						X				
-	Integrability											X		
-	Interoperability					X						X		
-	Resource Efficiency											X		
-	Reusability											X		
-	Scalability		X			X				X		X	X	
Security	Authentication		X		X	X	X			X		X	X	X
	Authorization		X		X		X			X	X	X	X	X
	Level	X					X						X	
	Integrity		X		X		X			X			X	X
	Confidentiality	X	X		X	X	X			X	X		X	X
	Accountability		X				X			X	X		X	
	Traceability		X		X					X	X		X	X
	Non-repudation		X		X	X				X	X	X	X	X
	Data Encryption									X	X	X	X	
	Isolation											X		
Configuration	Virtual Organization					X	X	X						
	Location						X	X						
	Level of Service	X					X		X	X				
	Service Version						X							
	Supported Standard		X	X	X		X			X			X	X
-	Cost	X	X			X	X	X	X	X	X		X	
Data	Maturity/Age			X		X				X				
	Timeliness					X				X			X	
	Reliability					X				X				
	Completeness					X			X	X			X	

Table III. Attributes Defined in Service Quality Approaches

frequent ones. Table III shows which quality attributes have been defined by which SQM(s).

Although several different QoS attributes and categories can be found in the various proposals, it is possible to single out the most frequent and consider them as

the "basic" and most important QoS attributes and categories, respectively. This is because all other attributes either capture secondary features or are more context-dependent (i.e. very specific), while they appear very scarcely in the proposed SQMs. As can be seen from Table III, *response time*, *latency*, and *throughput* are the attributes that mostly represent the *performance* category, which is present in most 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. This means that either these attributes are not so important for composite service developers or the SQM modelers have neglected them and focused only on their external counterparts.

Data quality is a multidimensional concept that defines the suitability of the used data for the application in which is involved. In the literature, there exist several contributions about data quality attributes and taxonomies/SQMs (e.g., [Redman 1997], [Strong et al. 1997]). The most important and representative data quality attributes (according to the data quality research) are *accuracy*, *completeness*, *consistency*, and *timeliness*. Since the service output is mostly composed of information, data quality can be considered as a part of the service QoS and it can drive thoroughly the analysis of the required input and provided output. Thus, data quality 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. However, data quality aspects are scarcely considered. The only data quality aspect that is mostly considered is *correctness*.

Some SQMs take into account particular network aspects. In 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*.

5. SERVICE QUALITY METAMODELS

5.1 Background

SQMMs have been mainly used to describe the service quality capabilities or requirements of an SP or SR, respectively. Thus, apart from their ability to describe SQMs, SQMMs can specify *QSDs*. 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 service functional and quality capabilities. The results of the QBSM process may be ranked, if it is needed, by executing the service selection process.

As it was analyzed in Section 2, some SQMMs can be also considered as SLA-MMs because they can describe SLAs. These SQMMs are called *SLA-enabled* SQMMs (SLA-SQMMs). The corresponding languages of this SQMM type include QML [Frølund and Koistinen 1998], WSLA [Keller and Ludwig 2003], WSOL [Tosic et al. 2003] and SWAPS [Oldham et al. 2006]). SLA meta-models that do not define quality attributes and the corresponding service quality capabilities are not con-

Table IV. 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

sidered in this section (e.g., WS-Agreement [WS-AGREEMENT 2003]). Section 6 will analyze all kinds of SLA languages with the appropriate SLA-based criteria.

Some security aspects like *trust* 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, another partition of SQMMs is considered in this section which maps to languages, such as Trust-Serv [Skogsrud et al. 2004], PeerTrust [Nejdl et al. 2004], WS-Trust [Nadalin et al. 2007], and P3P [Cranor et al. 2006], that describe a service quality part which is not described in the rest of the SQSLs. The SQMMs of this type are called *security-based* SQMMs.

5.2 Methodology and Analysis

In order to analyze all 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; De Paoli et al. 2008; Kritikos 2008; Kritikos and Plexousakis 2009]. These criteria mainly reflect the formality, expressiveness, complexity, and applicability of the examined SQMMs. The summary of our selected criteria (without their grouping) is shown in Table IV, while their complete presentation is provided later on.

Based on the SQMM categorization of the previous subsection, there are actually three SQMM partitions: *pure*, *SLA-enabled*, and *security-based*. The content of these partitions with respect to the corresponding SQMMs can be seen in Table V. The evaluation results of the examined SQMMs according to our comparison criteria are presented in one tree structure (see Fig. 6) and one table (see Table VI). The use of the tree structure is due to two main reasons: 1) some criteria were closely related to each other, and 2) a tree is a more user-intuitive means for presenting related evaluation results. Table VI is separated into three clusters, each one presenting

Partition	Approach Name	Approach Reference
Pure	WS-QoS	[Tian et al. 2003]
	WSAF-QoS	[Maximilien and Singh 2004]
	DAML-QoS	[Zhou et al. 2004]
	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]
	PCM	[De Paoli 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-Trust	[Nadalin et al. 2007]
	P3P	[Cranor et al. 2006]

Table V. SQMM partitions and their corresponding approaches

the evaluation results of one particular SQMM partition.

An ideal SQMM that satisfies most of the comparison criteria is visualized as a UML diagram in Figure 5. The parts that are highlighted with the red color correspond to those conceptual elements that are common among all SQMMs.

In the remainder of this section, each criterion or group of criteria is presented along with its evaluation results in separate subsections, where the analysis of the evaluation results 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.

5.2.1 *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 their representation formalism. 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 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. The evaluation of this criterion for each SQMM could get the following values: *informal*, *UML*, and *ontologies*.

The evaluation results are presented in the second column of Table VI. Most of the approaches use either ontologies or informal formalisms (mostly schema languages focusing on a language's concrete syntax), while only two use UML. This result is reasonable as ontologies are powerful modeling formalisms and very expressive, while informal formalisms are simple and very well supported. Concerning

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

Table VI. Comparison of SQMM approaches on eight evaluation criteria

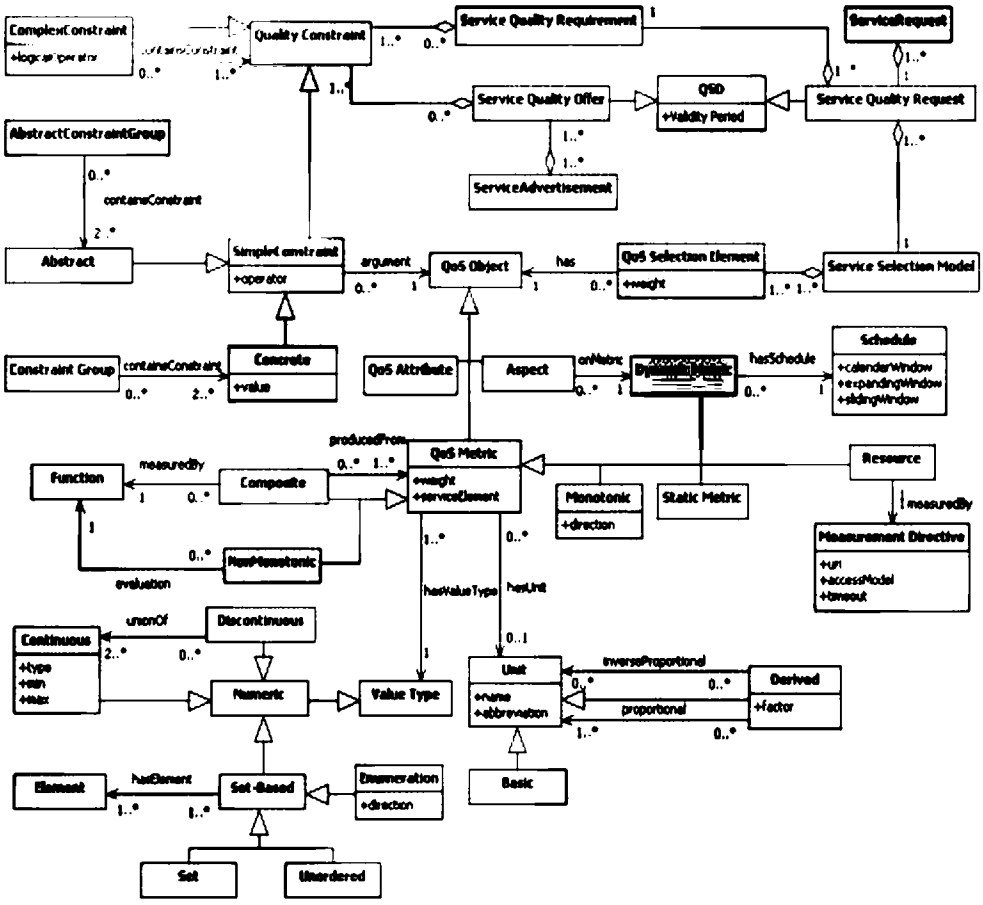

Fig. 5. An ideal SQMM.

the local partitions, pure SQMMs are modeled mostly through ontologies, while an informal formalism is the best modeling choice in the other partitions. For pure and SLA-enabled SQMMs there is a recent trend to use ontologies for their modeling. This is because ontologies provide unambiguous semantics to quality terms and, thus, enable machines to automatically process and reason on ontology-specified QSDs to support service life-cycle activities like discovery and negotiation.

5.2.2 *Richness in Defining Service Quality Models.* In the presence of multiple services with overlapping or identical functionality, SRs 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 considered. Therefore, a rich and extensible SQM must be enabled by the SQMM that includes both generic and

domain specific attributes and can be extended appropriately with the addition of new attributes. All the considered SQMMs advertise that they are extensible according to this aspect.

However, an SQMM richness depends also on other additional modeling capabilities that concern the offering of constructs that enable a quality attribute description in every possible detail/aspect. The following list summarizes all the modeling criteria that are needed to assess an SQMM's richness in defining SQMs:

(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 types)
(4) Modeling the attribute's 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.

This criterion's evaluation for a specific SQMM depends on the SQMM's satisfaction of the nine previously analyzed sub-criteria. 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*. 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 evaluation results are presented in the third column of Table VI. Most of the SQMMs can describe either a *fair* or *good* in richness SQM, while the rest can describe a poor SQM. As there is not any SQMM that can describe a rich SQM, there is not a perfect approach capturing this modeling aspect. Concerning the local partitions, the above general result also applies to pure SQMMs, while there is a trend revealing that the SQM richness is improved in recent pure SQMM approaches. Moreover, SLA-enabled SQMMs do not perform well in this modeling aspect because they give more importance on how a quality attribute can be measured than how it is modeled. On the other hand, security-based SQMMs perform moderately as the definition of quality attributes is one of their major concerns but not all attribute aspects need to be described.

5.2.3 *Richness in Defining Quality Metric Models.* An attribute is measured through the abstraction of a metric. While a metric model is encompassed in a quality model, the capability of a SQMM to express such models in a rich way was inspected separately for two main reasons. First, quality attributes and metrics are two different concepts. Second, both of these concepts require a quite rich description as the number of sub-criteria that were used to compare the SQMMs

richness in describing both of these concepts can reveal. So, an SQMM's overall richness depends on the richness of both its quality and metric models.

Thus, SQMMs should enable the creation of rich service quality metric models. The richness of the SQMM metric models was evaluated according to the following metric aspects:

(1) *The metric dynamicity.* Metrics should be distinguished between dynamic and static ones. Dynamic metrics measure dynamic quality attributes (like *availability*) that change over time and 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 over time. This means that these attributes are not only controllable but also fixable, so SPs are able to guarantee a fixed value for them for their services even if the services' context changes.

(2) *The metric value type.* 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, only the maximum and minimum value along with its numeric type (e.g., real or integer) have to be modeled for numeric domains. If the domain is not continuous, then it can be expressed as a union of continuous domains. In practice, numeric domains are used for most quality metrics. Set and enumeration domains do not have a built-in ordering, so a user-ordering of the domain elements must be described apart from the explicit modeling of these elements. Depending on the quality metric semantics (e.g. if the amount of values the metric can take only matters), the natural partial order of sets defined by inclusion can be used. The order in which an enumeration's elements are defined may also be their sorting order but the user has to define if it is increasing or decreasing. Finally, for unordered domains only the domain values have to be explicitly defined.

(3) *The metric unit.* The values a metric can take are measured in specific units, e.g. *seconds* for a metric measuring *execution time.* Concerning the modeling of units, describing just the unit name is not enough because additional 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 case the SP and SR express a metric constraint using different units or when combining metric measurements of different units originating from different sources. Units should be separated into *basic* and *derived* units. Basic units should have a name and a short abbreviation. Derived units are produced from other units by multiplying the component unit with a specific (float) value (i.e. multiples of units) and possibly dividing it with another one. For example, the unit of *minutes* is produced by multiplying the unit of *seconds* with 1/60. 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". Thus, two relationships stating which unit(s) is directly proportional and which is inverse proportional to it and an additional multiplying factor should be modeled for a derived unit.

(4) *The metric measurement directive or function.* Quality metrics should be classified into *resource* and *composite* metrics. *Resource metrics* are directly mea-

sured 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 to clear out if the party responsible for the measurement will ask for the value or receive 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 obtaining 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 and 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 metric schedule.* The SQMM should enable the definition of at least one of the following types 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, e.g. from this year's start until now.

(6) *The metric weight* relative to its implicit domain and user preferences. This weight can be used to calculate the rank of a service quality offer 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.* An SQMM should explicitly specify the exact monotonicity of monotonic metrics (e.g. negative for an *execution time* metric) and mapping functions for non-monotonic metrics. This information modeling is sufficient in most of the quality-based service discovery scenarios.
   The non-monotonic metrics need user-defined mapping functions to express the user preferences regarding the values that these metrics can take. For example, for the non-monotonic 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, 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.
   The above situation is tailored for QBSM scenarios where each quality metric is considered independently of the other. In case that there are dependencies between quality metrics and attributes, functions (or n-ary constraints) should be used to capture them in conjunction to the aforementioned mapping functions. The Simple Additive Weighting technique [Hwang and Yoon 1981] is commonly used in service selection and 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 most research

approaches that do not have to be captured in an SQMM.

(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 quality metric values for a composite service. So this description is needed for automating the quality analysis process, a prerequisite for a successful quality-based service discovery.

This criterion's evaluation for a specific SQMM depends on the SQMM's satisfaction of the eight previously analyzed sub-criteria. If a SQMM does not comply with all the modeling requirements for a specific 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 evaluation results are presented in the fourth column of Table VI. Most SQMMs encompass either a good or a fair metric model, while only one SQMM encompasses a poor model. In addition, only one approach (OWL-Q) captures a rich metric model, while it does not offer a rich quality model. Thus, a metric model is better captured in the current state-of-the-art SQMMs than a quality model. Considering the local results in each partition, SLA-enabled SQMMs capture a good metric model with respect to its richness. This observation was actually justified in the previous criterion's evaluation, where it was stated that SLA-enabled approaches pay more attention to metric than to attribute modeling. The majority of pure and security-based SQMMs present a fair metric model. Finally, there is a trend that pure SQMMs improve their metric model as the SQMMs proposed after 2004 do not encompass a fair or poor metric model. By combining this result with the respective result of the previous criterion, it can be deduced that the most recent pure SQMMs have increased their expressiveness as they cater for a better service quality and metric model.

5.2.4 *Expressiveness in Constraint Description.* A service quality specification comprises quality constraints. Each quality constraint consists of a name, an operator, and a value [Frølund and Koistinen 1998]. The name is typically a quality metric's name, 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. Only the values that a metric can take should 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, only the equal "=" and unequal operators "≠" can be used in unordered domains and not inequality operators ("<", ">", "≥", "≤").

Aspects [Frølund and Koistinen 1998] are statistical characterizations of quality constraints like: *percentile*, *mean*, *variance*, and *frequency*. They are used for the

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 very useful when the measurements or occurrences of a quality metric present some special characteristics and a new complex metric should not be produced from the basic quality metric for each of these characteristics.

Quality constraints are usually connected by logical operators into expressions. A service quality specification should contain one complete constraint expression 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, while *Constraint Group Templates* contain *abstract* quality constraints (i.e. the second constraint operand is not specified). Other reusability constructs can also be created even for expressions.

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

This criterion's evaluation results are presented in the fifth column of Table VI. Most of SQMMs use a rich constraint model. As there are no fair constraint models, SQMMs either perform above or below the average in this aspect. Moreover, in the SLA-enabled partition, there is a balance between those SQMMs that capture excellent constraint models and those that capture rich. This is because rich constraint representation, especially for service quality levels, is one of the cornerstone features of SLA languages, which is equally important for all SLA contracting parties. The security-based SQMMs mostly use a good constraint model because they do not require advanced constraint modeling features. In pure SQMMs rich constraint models are the majority. Moreover, pure SQMM modelers have quickly understood this feature's importance as their SQMMs moved from a poor to a better constraint model. By combining this result with the respective results of the two previous criteria, it can be deduced that pure SQMMs have increased their overall expressiveness over time.

5.2.5 *Complexity.* On one hand, a meta-model's *formalism* characterizes the explicitness in which the semantics of the meta-model's terms are expressed, while its *expressiveness or richness* concerns how well the domain of discourse (i.e. service quality) is modeled. On the other hand, a meta-model's complexity mainly concerns its size and structure and significantly impacts user understandability, the modeling effort involved, and the size and amount of information included in the produced descriptions. Obviously, the better the domain is modeled and more details are captured, the higher is the meta-model complexity. So, usually there is a trade-off between complexity and richness that is mainly regulated by the meta-model's quality. The latter depends on how accurately and extensively the meta-model

expresses its domain, the existence of formal and semantic inconsistencies in it, and the relevance and appropriateness of the modeled information. Thus, rich and qualitative meta-models must be developed such that their complexity is not very big.

It is very difficult to assess a meta-model's quality and very few approaches have been proposed focusing on some quality aspect [Jiang et al. 2004; Welty et al. 2003]. On the contrary, many metrics have been proposed for measuring meta-model or ontology complexity [Mens and Lanza 2002; Yi et al. 2004; Yoa et al. 2005; Yang et al. 2006]. Thus, the SQMM quality can be speculated by assessing the SQMM complexity through one of these metrics in combination with the SQMM richness.

The complexity metric that measures a meta-model's concept number was chosen for the following reasons: a) many SQMMs are not publicly available, so they cannot be evaluated with sophisticated complexity metrics; b) many complexity metrics are not applicable to informal SQMMs; c) the structure particularities of the informal SQMMs require using a simple and fair metric that does not depend on the SQMM structure and is easy to compute. However, such a simplistic metric prevents performing reasonable speculations about the SQMM quality, as the threshold on the number of sufficient concepts for modeling a domain depends on the subjective view of domain modelers.

Particular thresholds on the number of concepts involved in SQMMs were used to categorize them according to their complexity. So, if an SQMM has less than 25 concepts, it is evaluated to have *low* complexity. If it has 26 to 50 concepts, then it has *medium* complexity. Finally, if it has more than 50 concepts, then it has *big* complexity.

This criterion's evaluation results are presented in the sixth column of Table VI. Most SQMMs have low complexity, while only five SQMMs have medium complexity. Only one SQMM (OWL-Q) has high complexity, which is the most expressive SQMM according to the evaluation of the previous three criteria. Concerning the SLA-enabled partition, there is a balance between the approaches with low and medium complexity. All security-based SQMMs exhibit low complexity. By inspecting the results, the trend that pure and SLA-enabled SQMMs of higher complexity are proposed lately can be revealed. This means that modelers 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 years. Section 6 will show that the SLA-enabled SQMMs have increased their expressiveness but in SLA-based aspects and not in quality-related ones. 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.

5.2.6 *Service Description Separation, Refinement, and Granularity.* This subsection presents and analyzes a group of three related criteria. Fig. 6 presents the group's evaluation results with a tree-like structure.

5.2.6.1 *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

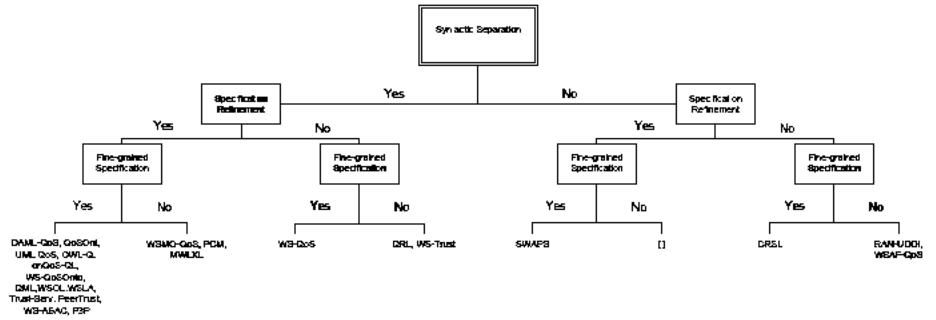

Fig. 6. Evaluation of SQMM approaches for the 4.2.6 group of criteria

allows to specify different quality properties for different implementations of the same interface. Moreover, while functional constraints rarely change during runtime, service quality constraints do change. So, the separation of service quality offers from WSDL descriptions permits these offers 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, this criterion's evaluation for a specific SQMM is: *yes* (for syntactical separation) and *no.*

5.2.6.2 *Support for the Refinement of Quality-Based Service Descriptions and their Constructs.* Apart from syntactical separation, another form of reusability is equally important. Service quality specifications should be also refined. This means that a new service quality offer can be created by referencing an older one and 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. Thus, this criterion's evaluation for a specific SQMM is: *yes* (supports refinement) and *no.*

5.2.6.3 *Support for Fine-Grained Quality-Based Service Description.* It should be possible to specify quality properties/metrics at a fine-grained level. As an example, performance characteristics are commonly specified for individual operations. A SQMM must allow quality specifications for interfaces, operations, attributes, operation parameters, and operation results. So, this criterion's evaluation for a specific SQMM is: *yes* (it allows), and *no.*

The most populated node of Fig. 6 is the one which corresponds to those SQMMs that enable all three features. So, most SQMM modelers have understood the importance of these three features. The same result holds for all the partitions. The mostly supported features among all SQMM approaches are syntactical separation and specification refinement, followed by fine-grained specification. The same order holds for the pure and security-based SQMMs. Finally, for SLA-enabled SQMMs the order is different. Fine-grained specification is the most important feature, while the other two are of equal importance.

5.2.7 *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 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 properties that a component provides or requires, without specifying the interconnection of components in order to enable the component reuse in 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 difficult to specify offers whose quality can vary. 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 achieve a better expressiveness to specify QoS, since there is usually no restriction on the number of involved parameters or operator types, so that non-linear or more complex expressions are allowed.

This sub-criterion inspects if both SPs and SRs can provide service quality specifications, if these specifications are defined separately with different constructs, and if these specifications are allowed to have the same expressiveness. This criterion's evaluation for a specific SQMM has the following values: a) *SP*, b) *same construct*, c) *asymmetric* (so also different constructs are used), and d) *symmetric* (where different constructs are used). 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.

The evaluation results are presented in the seventh column of Table VI. Most SQMMs allow both SPs and SRs to specify service quality specifications with the same construct, followed by those SQMMs that allow both SPs and SRs to specify service quality specifications with different constructs but symmetrically. Thus, researchers have realized that both SPs and SRs should be allowed to specify their QSDs with the same expressiveness. Concerning the SQMM partitions, all SLA-enabled and security-based SQMMs use the same construct for expressing the two QSD types. This is a rational choice for SLA-enabled SQMMs 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. The majority of pure SQMMs express the two different QSD types in a separate and symmetric way. Thus, the pure SQMM modelers have realized the significance of this design choice.

5.2.8 *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, this criterion assesses if a SQMM can capture only one class of service specification or many. In this way, this criterion's evaluation for a specific meta-model is: *one* (class of service), and *many*.

The evaluation results are presented in the eighth column of Table VI. Most SQMMs can represent many classes of service. This result also applies to each partition separately. So SQMM modelers have understood the advantage of allowing many classes of service specification for one service. For pure and SLA-enabled SQMMs this understanding is more anticipated in the most recent approaches.

5.2.9 *Connection to Service Functional Specification Languages.* This criterion inspects if the SQMM 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 SQMM, this means that there would be no effort required to extend this meta-model (and probably all of its QSDs) in order to be used in a registry containing functional SDs obeying to the connected SFSL. On the other hand, if the answer is negative, then the SQMM 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.

The evaluation results are presented in the ninth column of Table VI. 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 SQMMs. Three SQMMs do not reference any SFSL, being in this way SFSL-independent. Taking specific SQMM partitions into consideration, OWL-S (or ontology-based SFSLs more generally) is the most referenced SFSL in pure SQMMs 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 SQMM partitions. This is expected because SLA-enabled and security-based SQMMs 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) in order to increase their adoption. Moreover, there is a trend that the most recent SLA-enabled SQMMs are connected to WSDL.

5.2.10 *Support for Quality Matching.* As users may have different conceptualizations of the same domain, 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 measurement types 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, while in the latter case, the same metric is composite.

As the basic QSD part is the one where QoS capabilities or requirements are expressed 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. Thus, 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 SQMM's formalism, rules may be part of the 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, this information, either being internal or external, should be additionally modeled as it can be used to support service discovery and increase an SQMM's adoption and significance. Thus, this criterion's evaluation for each SQMM will be: *yes*, if the corresponding SQMM contains or is accompanied with quality matching rules and *no* otherwise.

The evaluation results are presented in the tenth column of Table VI. Most of the SQMM approaches have not considered this modeling aspect at all. This also holds locally in each partition. Moreover, not all security-based SQMMs use or model these matching rules as they are not required because the quality attributes in this area are more or less standardized, while quality metrics are used less often. Finally, the most recent pure and SLA-enabled approaches have understood the need of modeling this feature and have produced the required matching rules.

5.2.11 *Framework Support.* This criterion inspects 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 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). As all SLA-enabled SQMMs already have an underlying SLA enforcement framework implementation, we chose not to explicitly represent this fact in this evaluation.

The evaluation results are presented in the last column of Table VI. The majority of the SQMMs have been used in service discovery implementations. 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 can also perform service negotiation. Moreover, there is no service negotiation framework supporting any of the SQMMs. It must be noted that there are negotiation frameworks supporting WS-Agreement but not SWAPS (which is a WS-Agreement extension).

Most of the SLA-enabled SQMMs apart from WSLA are not used in any implemented negotiation framework. This actually reduces the dynamics and further spread of these SQMMs. In addition, by considering the service life-cycle analyzed in Section 2, it should be stated that only the SLA-enabled SQMMs have been used in service negotiation, as the QSDs of this SQMM type that are used in this

activity are SLAs or SLA templates. Thus, the service negotiation activity is the stopping usage point of the QSDs and the corresponding starting point of SLAs.

5.3 Overall Analysis

Based on the above analysis, there is not any complete approach that has taken the best value in all criteria. From all the SQMMs, 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, while a balance between expressiveness and complexity should be accomplished by defining which are the obligatory and non-obligatory concepts in order to help modelers in defining their QSDs with less effort and more speed. Thus, the design of this language should be revised accordingly.

Considering the remaining SQMM partitions, SWAPS is the best among all SLA-enabled SQMMs but it must significantly improve its attribute and metric model. This is also true for PeerTrust, the best among all security-based SQMMs. Thus, 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 last criterion results, 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.

6. SERVICE LEVEL AGREEMENTS

6.1 Background

6.1.1 *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 contractual relationships establishment and the enactment infrastructure set-up. Many types of contracts exist based on the type of value that is exchanged between two or more organizations [Grefen and Angelov 2002]. However, this paper considers only *service contracts*, as the focus 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.

Most of the service contract languages focused on describing the service functionality 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. Thus, they focused more on how the service behaves while they more or less neglected the QoS aspects of service behavior. However, as QoS plays an important role in the whole service life-cycle and is equally important to functionality, the focus is now 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 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.

6.1.2 *SLA Demystification.* According to [Paschke and Schnappinger-Gerull 2006], SLA documents contain *technical* (e.g. metrics, actions), *organizational* (monitoring and reporting), and *legal* components (legal responsibilities, invoicing and payment modes). The legal components are hard to be automated and enforced by a management system, so they are either omitted or neglected. The most common SLA components 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 SP and SR that sign the contract, and *supporting parties*, which have the role to support the SLA monitoring and assessment. These two role types are not mutually exclusive, as, for instance, a SP can provide measurements for the provided service's execution time. The *contract validity period* specifies for how long the SLA will be valid and enforceable. This field is important for *continuous* services as it determines the period that the SPs should provide their services to the requesters according to the directives of the agreed SLAs. The *service definitions* section specifies the service characteristics (i.e. functionality), components (i.e., operations, input, output, internal and external services for a composite service), and observable parameters (i.e. QoS metrics for the service and its components). *SLOs* are QoS guarantees that must 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 SLA parts should be de-

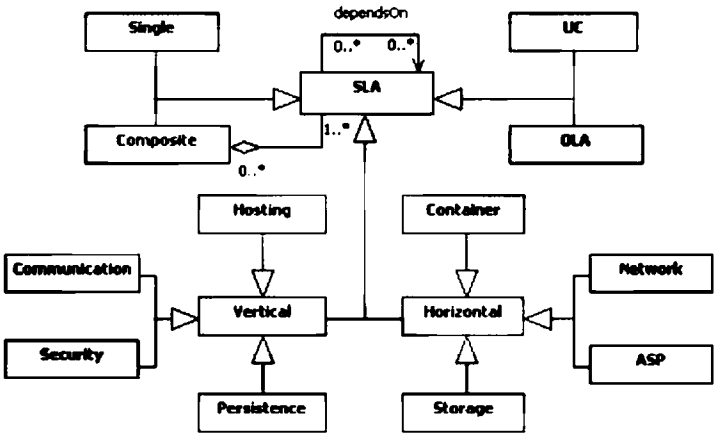

Fig. 7. SLA Categorization.

scribed [Muller 1999]: *limitations* and *renegotiation*. The former describes the limits of IT support during peak period demand conditions, resource contention by other applications, and general overall application workload intensities. These limitations should be agreed 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 to reflect changes in the (user, service or environmental) context.

Before SLAs are established, they are in a form which is called *SLA template*[2]. These SLA templates are used to describe, matchmake, and negotiate the SLs to 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. Complete SLA templates 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. Thus, they are offered in a "take it or leave it" basis [Hoffner et al. 2001]. Incomplete 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 a SLA instance. According to the *granularity of choice* and *specialization* [Hoffner et al. 2001], they may : a) be *monolithic* where values are inserted in predefined positions; b) have certain sections which can be included or removed; c) be clause-based [Fosbrook and Laing 1996].

There can be different SLA types 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 7 shows the three dimensions of SLA categorization along with their corresponding types. These dimensions are orthogonal to each other and their types are mutually exclusive, so

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

one SLA may belong to only one type of each dimension.

According to their composability, SLAs are distinguished between *Single* and *Composite*. *Single SLA*s specify the SLs of one service, independently of the service type (i.e., single or composite), and are agreed between two parties, the SP and SR. On the other hand, *Composite SLA*s specify the SLs of both the service, which should be provided to an SR by the SP, and some of its components 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. This 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 *Single SLA*'s description.

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 SLA types specify the service that the operational partners are going to deliver to the SP, which will be used to support the service specified in another SLA that the SP promises to deliver to a prospective SR.

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

6.2 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. This life-cycle includes all the appropriate activities needed for the SLA management and is tightly coupled with the service life-cycle introduced in Section 2. The same holds for the contract life-cycle. This tight coupling is justified as follows. 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 service contract and SLA management systems actually support, directly or indirectly, the management of the service offered by the SP to the SR. Thus, through the support of the SLA/contract life-cycle, the service life-cycle is supported and especially those activities that correspond to service provisioning.

Service contract life-cycles (e.g. in [Hoffner et al. 2001] are more coarse-grained

and general with respect to the SLA life-cycle as they consider both the service provisioning functional aspects and the QoS ones. However, as the focus is on service quality and its description, only the QoS aspects of service provisioning are considered. To this end, 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 to compare these languages along the lines of the information they can describe which is necessary for supporting the SLA life-cycle activities.

Various SLA life-cycles have been proposed in the literature, which differ on the activities they involve, the activities granularity level, and their terminology. However, none of them is suitable in this paper context as they are either coarse-grained or tend to neglect some activities. To this end, an extended SLA life-cycle has been devised based on the research works of [Keller and Ludwig 2003; Parkin et al. 2008], which is depicted in Figure 8.

In the following, using this life-cycle as a basis, the life-cycle activities involved in the SLA management are analyzed along with their relation to those of the service life-cycle:

—*Template Development*: A SLA template is developed by the SP or the SR according to the service quality capabilities and requirements, respectively. This activity is executed after the service is implemented and tested but can happen before, during or after the service 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. It is a joint activity of the service and SLA life-cycles.

—*Matchmaking*: SRs make 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 their quality requirements. This activity is after or in parallel with the functional service discovery activity. It may also be 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 a joint activity of the two considered life-cycles. The SP of the best service (or the SPs of the matched services) negotiates with the SR according to their SLA templates' content. These SLA templates may change during the negotiation to reflect the compromises performed by the two parties.

—*Agreement & Deployment*: As the outcome of service negotiation is not always successful, this activity is separated 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) a signatory party's deployment system 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) the supporting party deployment systems configure their own implementations in a suitable way. All parties need to know

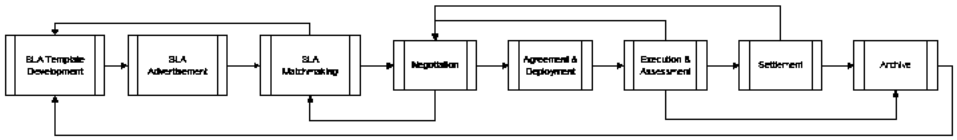

Fig. 8. The SLA life-cycle.

the definitions of the interfaces they must expose, as well as the interfaces of the partners they interact with. This composite activity is usually performed before service execution. When no active service instance can execute in the corresponding SL of the SLA, then a new service instance must be deployed or more resources are given to a specific instance. In this case, the service deployment activity runs 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 SLA's agreed SLOs are assessed. Monitoring is performed by the measurement components of the supporting parties run-time systems which maintain information on the current system configuration and the SLA metrics. 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 (of the signatory parties and the SLA's external one (if it exists)) about violations. If the violation or violation number is very critical, then the SLA is re-negotiated or canceled according to the SLA's corresponding action guarantee. If not, an appropriate corrective action is selected and performed by the SP's management system (e.g. more resources are provided to the under-performing service) according to the current context and the SP's business goals and policies. This SLA management activity runs in parallel with the service execution, monitoring, and recovery life-cycle activities.

—*Settlement*: This activity determines if the SLA was met, the final cost to paid by the SR, and which penalties may apply to the SP for breaching the SLA terms. Negotiations for SLA termination may be carried out between the parties, in the same way as the SLA establishment is performed, or for service re-execution in a different SL and cost. This activity occurs after the end 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 must be kept as it is a legal document describing how services were provided. If this activity is accompanied with an audit trail mechanism, it can be used for identifying problems and patterns of wrong service behavior or user requirement trends so as to improve the future developed SLA templates content and even evolve the service implementation towards correcting the identified problems and meeting the increased customer needs. Thus, this activity can provide significant input to the service evolution life-cycle activity.

The summary of the selected criteria is shown in Table VII, while their complete presentation is provided later on in this section. In this table, the term "Description" is used 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 an SLA's description. Moreover, the "Agreement & Deployment" activity was not analyzed by using any criterion, as this activity does not need any specific information which is not already covered by existing SLA languages. Finally, as it was explained in Section 5, most SLA languages regard that quality should be defined outside the SLA specification by different formalisms and languages and only referenced inside this specification. For this reason, 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 VIII. While there is no standard service contract language (SCL), there are two widely used SLA languages, namely WSLA [Keller and Ludwig 2003] and WS-Agreement [WS-AGREEMENT 2003], which can be considered as standards. The most representative SCLs have been chosen because, as was explained above, the majority of these languages focuses more on the functional aspects of service behavior. The evaluation results of the examined languages according to our selected criteria are presented in Tables IX and X.

In the remaining part of this section, each activity-based group of criteria along with their evaluation results is presented in separate subsections. In the end, there is an overall analysis of the service contract and SLA languages across all criteria.

6.2.1 *Description.* Every SLA and service contract language must possess some properties that enable it to be a good candidate for representing SLAs. A set of such four properties/criteria has been selected and is analyzed in the next four sub-subsections.

6.2.1.1 *Formalism.* Similarly to SQSLs, each SLA language adopts a specific formalism to express its meta-model. These formalisms include: *informal* (such as DTDs or XML Schemas), *UML*, *RuleML*, *Finite State Machines* (FSMs), and *ontologies*, and have been used for categorizing each language.

The evaluation results are presented in the first row of the *Description* composite SLA life-cycle activity of Tables IX and X. The vast majority of the SLA-MMs is expressed with informal formalisms (mostly XML Schema but also other schema languages which focus on a language's concrete syntax). The same result applies to the SLA language partition but not to the SCL one, where SCLs exploit mostly formal formalisms. Schema languages are selected as they lead to a quick way of producing a language, surpassing in this way its abstract syntax expression. In addition, the majority of the approaches uses XML Schema for the concrete syntax and XML for the SLA representation. 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 SLA consistency and the *formality* needed to perform *SLA analysis* (analysability) [Skene 2007] which can

Table VII. 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 SLA parts are negotiable and in what way
Monitoring	Metric Provider	The ability to define the party responsible of producing a metric's measurements
	Metric Schedule	The ability to define the production frequency of a metric's measurements
Assessment	Condition Evaluator	The ability to define the party responsible of SLO evaluation
	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 assessment frequency 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 SLA outcome
Archive	Validity Period	The ability to express the period where an SLA is valid

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). This justifies the choice of SCL modelers to adopt such formalisms, as the focus is on analyzing the functional service behavior. 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.

ID	Approach Name	Approach Reference	Type
1	QML	[Frølund and Koistinen 1998]	SLA
2	WSLA	[Keller and Ludwig 2003]	SLA
3	WS-A	[WS-AGREEMENT 2003]	SLA
4	SLAng	[Lamanna et al. 2003]	SLA
5	WSOL	[Tosic et al. 2003]	SLA
6	RBSLA	[Paschke 2005]	SLA
7	QoWL	[Brandic et al. 2006]	SLA
8	GXLA	[Tebbani and Aib 2006]	SLA
9	TrustCom	[TrustCoM Consortium 2007]	SLA
10	X-Contract	[Molina-Jimenez et al. 2003]	Service Contract
11	BCL	[Linington et al. 2004]	Service Contract
12	SweetDeal	[Grosof and Poon 2004]	Service Contract
13	CTXML	[Farrell et al. 2004]	Service Contract
14	SWCL	[Oren et al. 2005]	Service Contract

Table VIII. The SLA and service contract languages examined

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

Table IX. Evaluation results of SLA Languages

Most SLA and service contract languages can perform at most the structural and semantical *validity* forms, (i.e. discover syntactic and semantic inconsistencies with the help of DTDs, XML Schemas, and ontologies). However, another validity form is also required, which is called *SL validity*, in order to discover a specific *quality inconsistency* type in SLs, which concerns the *constraint consistency*. As SLs are composed from the logical combinations of SLOs, it must be checked if this combination is meaningful and correct. For instance, if two constraints of the form $X < a$ and $X > b$ are conjunctively combined, where X is a QoS metric and $b > a$, then metric X would not be allowed to take any value from its value type, so the

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

Table X. Evaluation results of Service Contract Languages

produced SL would not be valid. This problem is exacerbated when arbitrary functions are involved in the SLO constraint expressions. 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].

6.2.1.2 *Coverage.* SLA and service contract languages should be able to express in an efficient and complete way both functional and quality terms. The functional terms description should include the description of the service functionality, operations, 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 quality terms description should also include the description of the QoS metrics to be measured and various other quality concepts in order to be complete. If the SLA or service contract languages are not able to describe these additional quality terms, they reference these terms' external descriptions in respective SQSLs. Thus, this criterion's evaluation considers 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 evaluation results are presented in the second row of the *Description* activity of Tables IX and X. All the languages reference or explicitly define functional terms. Besides, the 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. Only four SLA languages enable the description of the func-

tional terms of the SLA, while no SCL enables this description. Moreover, the WS-Agreement and TrustCom SLA languages enable both the description and reference of the functional SLA terms. The latter language is an WSLA-based extension of WS-Agreement, so it is reasonable to inherit the majority of WS-Agreement's capabilities. As already explained in Section 5, the syntactical separation of functional and SLA descriptions enables the reuse of a specific contract among many services that exhibit the same quality capabilities. In addition, it facilitates the contract management and evolution (through the manipulation of changing SLs [Tosic et al. 2003]) and disables the repetition of a service's functional description in all the service's contracts. 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 service functionality and description externally and not visibly to an SLA is avoided.

Concerning the description of additional quality terms apart from SLOs, almost half of the languages only reference external quality descriptions usually found in SQMs. In the SLA language partition, there is no language that does not describe or reference additional quality terms and this result is quite reasonable. 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. 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.

6.2.1.3 *Reusability & Extensibility.* A SLA/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, SLA or service contract specification parts, like the functional and quality terms, should be reused across many SLA or service contract documents or extended appropriately. Thus, this criterion's evaluation for each language could get the following values: *no*, *part* (i.e. only the whole SLA/service contract is reusable and extensible), and *yes* (i.e. parts and whole SLA/service contract are extensible and reusable).

The evaluation results are presented in the third row of the *Description* activity of Tables IX and X. Most SLA languages are re-usable and extensible both in the SLA specification entirety and in its parts. This result applies also to the SLA language partition. Concerning the SCL partition, there is a balance between those SCLs that are re-usable and extensible only in their entirety and those that are also re-usable and extensible in their parts.

6.2.1.4 *Composability.* This property indicates the ability of an SLA or service contract language to represent composite SLAs or service contracts, respectively. It can be achieved when the language presents the following abilities: a) describe or reference composite service descriptions; b) define or reference metrics that are associated to composite services and are computed based on aggregation rules that depend on the composite service structure; c) cater for the different party types (i.e. third-party SPs) involved in composite contracts; d) define composite and

component-based SLs and their associations; e) define appropriate action guarantees that consider the contract's two-level hierarchy. 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 ability, 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 evaluation results are presented in the last row of the *Description* activity of Tables IX and X. Most languages has not even the basis of being composable. This result applies also to the SCL partition, where four out of five approaches are not composable at all, while one approach possesses only the first ability. This actually means that SCLs were not designed to represent composite service contracts or SLAs. Concerning the SLA languages partition, there is actually a balance between approaches that have fair/basic and no composability at all. Only one SLA (GXLA) language scores good in its composability. This means that the SLA modelers have not yet understood the need for representing composite SLAs.

6.2.1.5 *Overall Analysis for the Description Activity.* Based on the above analysis, there is no SLA or SCL that meets the high standards posed. Only GXLA can be distinguished based on its capabilities with respect to composability. However, this language lacks the appropriate formality (or transformation to such a formality) which is needed for SLA validity and analysis.

6.2.2 *Matchmaking.* Only one research approach [Oldham et al. 2006] performs proper matchmaking of SLA specifications so as to match the user QoS requirements with the service QoS capabilities. In this approach, WS-Agreement specifications 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 remaining approaches is that matchmaking is performed during service negotiation, where one participant 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 and by entering new SLOs, or proposes a new one. This matchmaking type is not 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 its 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.

Based on the content of most SLA life-cycles (including the proposed one) and the above reasoning, SLA/service contract template matchmaking should be performed before service negotiation in order to discover those services of the corresponding SPs that suit the user quality requirements. This process can be effective and accurate if some prerequisites are met by the SLA and service contract languages

and 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, three main description prerequisites and one specific requirement (i.e., the existence of a matchmaking metric) are analyzed, which 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 is given.

6.2.2.1 *Metric Definition.* Metric modeling capabilities were analyzed in Section 5, where four SLA languages were compared. The rest of the SLA (apart from TrustCom) and service contract languages considered 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] to infer their equivalence. These two problems reduce the matchmaking activity's accuracy. Thus, languages that define metrics or enforce the use of a specific SQSL to define metrics are preferred. So, this criterion's evaluation 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 evaluation results are presented in the first row of the *Matchmaking* life-cycle activity of Tables IX and X. Five SLA languages (four plus TrustCom that relies on WSLA) satisfy this criterion, which are the approaches able to define QoS metrics and other SLO quality terms on their own. So, by also considering the results of the *coverage* criterion of the *Description* composite activity, it can be inferred that all languages that can reference external metric descriptions do not determine which SQSL should be used to specify these descriptions. Thus, the use of these languages may lead to low accuracy matchmaking results based on the above analysis.

6.2.2.2 *Alternatives.* Two SLA/service contract specifications match when their part corresponding to the offered or required SL is matched. Existing languages do not encompass the SL concept but either assume it is the conjunction of all SLOs defined in the SLA or emulate it either through the use of logical predicates that logically connect the defined SLOs or by offering different SLAs 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. Thus, SLA and service contract languages should be able to represent alternative SLs in order to increase the chances of matching their corresponding specifications. Alternative SLs represent the different modes in which a service can operate to suit the diverse needs of different user classes, 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, this criterion's evaluation for each language would be *no* if the language is not able to represent

SLs, *impl* if it can define them implicitly, and *yes* if there is an explicit language construct that is used to represent alternative SLs.

The evaluation results are presented in the second row of the *Matchmaking* life-cycle activity of Tables IX and X. Most languages are able to implicitly define alternative SLs which are needed 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 SCLs, only X-Contract is able to implicitly define SLs. Thus, both SLA and service contract languages were not designed to support SLs but only SLA languages are able to implicitly define them.

6.2.2.3 *Soft Constraints.* Even if many alternative SLs are represented in an SLA template to be matched, there will always exist a problem [Kritikos 2008] where users express over-constrained SLs which cannot be matched by any alternative SL of any offered SLA template. An over-constrained SL means that its encompassing SLOs contain very restrictive constraints that cannot be satisfied. As the root of this problem is that SLOs are expressed as hard constraints that must be satisfied at all costs to infer the matchmaking, its solution may come through the use of soft constraints. In particular, if SLOs are expressed as soft constraints, where the user expresses their significance through using weights or levels, 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 insignificant requested SLOs are violated. Thus, this criterion's evaluation for each language would be *no* if the language cannot express soft constraints (i.e. SLOs), or *yes* otherwise.

The evaluation results are presented in the third row of the *Matchmaking* life-cycle activity of Tables IX and X. Only two SLA and one service contract language can define soft constraints, while the rest of the languages define SLOs as hard constraints. Thus, only these three languages could be used to express SLAs/service contracts which could be exploited to solve the over-constrained user-requested SLs problem.

6.2.2.4 *Matchmaking Metric.* As languages may differ in the way they define QoS metrics and SLOs, it would be very useful when implementing SLA matchmaking engines if a specific matchmaking metric was defined internally or externally in the SLA language. This metric would be used for matching SLAs (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, this criterion's evaluation for each language would be *no* if no matchmaking metric is defined, or *yes* otherwise.

The evaluation results are presented in the last row of the *Matchmaking* life-cycle activity of Tables IX and X. Only two SLA languages (QML and SLang) explicitly define a matchmaking metric with which their produced SLA specifications can be matched.

6.2.2.5 *Overall Analysis for the Matchmaking Activity.* There is no language that satisfies all four criteria of the *Matchmaking* activity. Only QML and TrustCom satisfy three of the criteria. However, among these two languages QML is considered as the best because the *soft constraints* criterion is the least significant

one as it provides additional and not basic support the SLA matchmaking activity. 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 SLA specification matchmaking. Thus, 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.

6.2.3 *Negotiation.* Service negotiation is one of the most important activities as it produces the final SLA document that will drive the service execution. For this reason, SLA 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, it is explained why these two information types must be captured by SLA languages and what should be their content, while it is inspected if the languages capture this essential information. Finally, the overall performance and support of the examined languages for this SLA management activity is assessed. As *negotiation strategies* represent private and sensitive information for the participants which must not be exposed in SLA templates, it was considered that they do not constitute appropriate SLA information for the support of service negotiation.

6.2.3.1 *Meta-Negotiation. Meta-negotiation* is any information that can be used for negotiation establishment, i.e. to enable and initiate the negotiation between the participants. The following information has been identified 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 exchanged messages used to negotiate and conclude (i.e., agree) an SLA/service contract. The protocol should also unambiguously define the semantics and format, or schema, of the messages. Each negotiation participant may 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 for enacting the negotiation. For this reason, each participant's corresponding SLA/service contract template must reference all negotiation protocols that can be supported. Moreover, this reference should include a pointer to the supported negotiation protocol description 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 negotiation protocols may not be not implemented in negotiation engines and brokers as e.g. are not widely used; therefore, they have to be defined, e.g. in BPEL, in order to be properly enacted by the negotiation broker.

When no matching negotiation protocol is found, it is advocated in [Comuzzi et al. 2009] that the participant negotiation capabilities must be described in a more fine-grained way along with the possibilities of delegating capabilities 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 participants fine-grained negotiation capabilities 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 the insecure channels established between the participants. For example, SPs do not want the offers they make to specific (e.g. privileged) clients to be viewed by all other clients. For this reason, the participants must describe in their SLA templates the authentication methods they prefer to be used for securing the channels exploited when exchanging negotiation information. Then, preferred authentication methods will be matched so as to select the most common one for the negotiation.

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

The evaluation results are presented in the first row of the *Negotiation* life-cycle activity of Tables IX and X. 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.

6.2.3.2 *Negotiability.* While *meta-negotiation* represents information which is used to support the service negotiation enactment, *negotiability* represents information which is used during the negotiation. In particular, negotiability is the ability of a SLA 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 quality terms upper and lower limits. Thus, this criterion's evaluation for each language would be *no* if the language does not define which terms are negotiable, *part* if the language characterizes only the terms negotiability, and *yes* if the language also determines which are the allowed values or range of values for the quality terms upper and/or lower limits.

The evaluation results are presented in the second row of the *Negotiation* life-cycle activity of Tables IX and X. 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 a quality term's limits can take one or more values.

6.2.3.3 *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, they should be extended by modeling the participant negotiation capabilities, the authentication method used for the information exchange during the negotiation, and specific constraints that define the allowed values for the negotiable quality terms limits. Considering the latter extension, one good solution is proposed in [Andrieux et al. 2004] which is, however, not adopted in the language's formal specification. SCLs are not able to properly support the quality-based terms negotiation. This fact along with the inability of the SCLs to

support the SLA matchmaking activity prevents them from being widely adopted as the languages for expressing a service's quality-based behavior.

6.2.4 *Monitoring & Assessment.* During service execution, the service is monitored so as to assess if the SLOs defined in its SLA are violated. Service monitoring is performed by producing measurements according to the information that is encapsulated in the SLO metrics that are in the service scope. As both the metric definition and the association of metrics to service objects were evaluated in previous parts of this paper, the only additional information needed for *Monitoring* is who is in charge of performing the metric measurements, 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 SLAs is established after service negotiation.

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

The evaluation results are presented in the first row of the *Monitoring* life-cycle activity of Tables IX and X. Almost half SLA and almost all service contract languages are not able to specify the providers of SLO metrics as they assume that the measurements are only provided by the SP. This is a major limitation because measurements may be provided by other parties and the SLO evaluators, which may be different from SPs or even SRs, would not be able to assess the SLOs if they do not know the place from where SLO metric measurements can be obtained.

6.2.4.2 *Metric Schedule.* The schedule of a metric determines the production frequency of its measurements. This criterion's evaluation for each language would be *no* if the language cannot define metric schedules, or *yes* otherwise.

The evaluation results are presented in the second row of the *Monitoring* life-cycle activity of Tables IX and X. Only four SLA languages are able to specify metric schedules. Three (WSML, WSLA, and SLAng) of these four languages satisfy both the current and the previous criterion. The rest of the SLA languages and all SCLs do not model this feature. However, as this feature is used to specify the timing of the measurement productions of the SLO metrics, its lack can cause problems in the *Assessment* activity.

The SLA assessment is one of the most crucial SLA life-cycle activities. Thus, SLA languages 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 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, the purpose and content of this information is analyzed and it is evaluated if the examined languages have modeled it.

6.2.4.3 *Condition Evaluator.* Similarly to metric measurement, the SLO assessment should be made by a (supporting) party which is named *Condition Evaluator*. This party is 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. This criterion's evaluation for each language would be *no* if the language does not define this type of supporting party, or *yes* otherwise.

The evaluation results are presented in the first row of the *Assessment* life-cycle activity of Tables IX and X. 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 assessment activity is performed in the signatory parties management systems based on the information coming or pulled from the monitoring components. In this way, they limit the way an SLA management system can be implemented or distributed as they exclude the existence of third-party assessment components.

6.2.4.4 *Qualifying Condition.* Apart from the SLO to be evaluated, there can be a precondition, named *Qualifying condition*, which must hold to assess the SLO. This precondition may express assertions over service or other quality attributes or external factors such as the SR's *service request rate*. This criterion's evaluation 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 evaluation results are presented in the second row of the *Assessment* life-cycle activity of Tables IX and X. Only one SLA language (WS-Agreement) and two 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 an SLO's assessment, 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).

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

The evaluation results are presented in the third row of the *Assessment* life-cycle activity of Tables IX and X. Most SLA languages are able to define which is the obliged party in an SLO. Moreover, all SCLs model this information as their design is based on policies expressing obligations and various other implication types.

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

The evaluation results are presented in the fourth row of the *Assessment* life-cycle activity of Tables IX and X. Most of the languages specify an SLO's assessment schedule. This result applies also to the SCL partition, as SCLs have been designed to support this criterion. However, 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 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 SLA management system's monitoring components. The first approach is adopted by the SLA languages, while the second approach is adopted by all SCLs.

6.2.4.7 *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. An example of the value set this field can take is {*business days*, *regular working hours*, *maintenance periods*}. This criterion's evaluation for each language would be *no* if the language does not define the an SLO's validity period, or *yes* otherwise.

The evaluation results are presented in the fifth row of the *Assessment* life-cycle activity of Tables IX and X. 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, as there is a balance between the approaches that model this information and those that do not. Thus, SLA language designers have better understood the need of supporting this criterion with respect to those of the SCLs.

6.2.4.8 *Corrective Actions.* When the signatory parties are informed about an SLO violation, corrective actions must be carried out at the obliged party's management system or at the global level by renegotiating or canceling the SLA/service contract. When corrective actions should be taken by the obliged party, the choice of which action to perform depends on many situational factors and the obliged party's business goals and policies. So, this party would not desire or be possible to advertise in an SLA/service contract what actions to perform in which SLO violation case. However, this party may advertise some corrective actions to be taken for the corresponding SLOs violations 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, this information should be certainly modeled in an SLA/service contract. So, this criterion's evaluation for each language would be *no* if the language does not model *corrective actions*, or *yes* otherwise.

The evaluation results are presented in the fifth row of the *Assessment* life-cycle activity of Tables IX and X. The majority of the language designers has recognized the significance of modeling this information. The same result also applies to the SCL partition. However, it does not apply to the other partition, as there is a balance between SLA languages that model corrective actions and those that do not. This means that SCL designers have better understood the importance of this criterion with respect to those of the SLA languages.

6.2.4.9 *Overall Analysis for the Monitoring & Assessment Activity.* Based on the overall performance of the examined languages on the monitoring and assessment criteria, WSLA is the best language that models all appropriate information that is required for supporting the *Monitoring* and *Assessment* life-cycle activities, as the support of these two activities was one of its design requirements which seems to be successfully implemented. 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.

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

6.2.5.1 *Penalties.* Penalties are paid by the SP if one or more SLOs are violated. Each SLO is usually associated with a specific penalty-amount. However, in some cases, the penalty to be paid could increase exponentially with the violation number [Paschke and Schnappinger-Gerull 2006]. The latter penalty type is not modeled 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 violation number in the SLOs that compose this SL. If a language is able to define penalties at the SL, then it can also define penalties at the individual SLO level. Thus, this criterion's evaluation 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 evaluation results are presented in the first row of the *Settlement* life-cycle activity of Tables IX and X. Most of the languages are able to specify penalties. The majority of these languages can define penalties for the whole SL, while only three languages define penalties for each SLO. Considering each partition separately, the majority of the SCLs is able to define penalties for each SL, while there is a balance between those SLA languages that are able to define penalties and those that are not. 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. The results show that SCL designers have better understood the need to model penalties than the SLA language designers.

6.2.5.2 *Rewards.* Rewards are paid by the SR if one or more SLOs are more than respected. Rewards can be defined at the SLO (via a specific value) or SL level (via a function). Rewards should be modeled as they would give extra motives to SPs to provide even better SLs than the ones offered to respective SRs in the past. This SL upgrade would lead to increase in profits, which is one main goal of SPs when they offer their services. Moreover, the SL's trust and reliability would also increase. This criterion's evaluation 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 rewards at the SL level.

The evaluation results are presented in the second row of the *Settlement* life-cycle activity of Tables IX and X. The results are different with respect to those of the previous criterion. Less than half of all languages are able to define rewards. Moreover, there is a balance between the languages able to define rewards at the SL level and those able to define rewards at the SLO level. In the SCL language partition, all SCLs are able to define rewards at the SL level. Concerning the other partition, the majority of the SLA languages is not able to define rewards as either

no real-world case of SLAs contains rewards or the SLA language designers have not recognized the need of modeling rewards. As SLA languages represent the majority of all languages, this explains the bad global result.

6.2.5.3 *Settlement Actions.* Settlement actions are mutually taken by both signatory parties to decide about the SLA/service contract final outcome. Thus, when there are no severe SLO violations or the violation number is not high or zero, the SLA 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 must 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 the conditions on which they are applied. So, this criterion's evaluation for each language would be *no* if no settlement actions can be defined, or *yes* otherwise.

The evaluation results are presented in the last row of the *Settlement* life-cycle activity of Tables IX and X. Half of the languages are not able to model settlement actions. Concerning the SLA partition, only two SLA languages are able to model such actions. This is a significant limitation that would discourage potential SPs or SRs from using them. Moreover, this explains the bad global result. This situation is reversed in the SCL partition, as all SCLs support the modeling of this information. This means that, indeed, the SCL design has been centered on the modeling of various compensation actions, including the settlement ones.

6.2.5.4 *Overall Analysis for the Settlement Activity.* Based on the evaluation results of the settlement criteria, only RBSLA among the SLA languages satisfies all criteria and can specify penalties and rewards on the SL. The same applies for three (SweetDeal, CTXML, and SWCL) out of five SCLs. Thus, these four languages can be used for appropriately supporting the *Settlement* life-cycle activity. However, these languages require significant effort and extensions from the SLA modeler to express different SLAs with different settlement actions and conditions. In addition, they force the SLA management systems adopting them either to support one by one the different settlement actions that may exist in the SLAs or to define appropriate extensions with which 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, to avoid such situations and further increase their adoption and universality, these languages should be extended appropriately to specify explicit constructs that model the various settlement actions that may exist in an SLA/service contract.

6.2.6 *Archive.* An SLA/service contract is archived in three distinct cases: a) the SLA 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 is canceled. In the second case, the SLA 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 SLA archiving, 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, each language is evaluated only

based on its capability to model the SLA validity period. If it does not model this period, the evaluation result is *no*, otherwise *yes*.

The evaluation results are presented in the last row of Tables IX and X. 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. So, the SLA language designers are starting to understand the need of modeling this information, while the corresponding SCL designers do not. From the SCL side, this conclusion can be explained by considering that these languages were designed with the focus on functionality and not on quality. Thus, as functionality does not change so much, there is no need to explicitly model the service contract validity period. The contract could be invalidated as soon as the business relationship between the contracting organizations ceased to exist for various reasons.

6.3 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, the analysis focuses 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 each activity's overall results, it can be inferred that there is no language supporting 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 SLA management activities, as they signify that these languages do not possess all appropriate modeling capabilities so they should be used for expressing only some SLA types. Based on the fact that the capabilities of the most widely used SLA languages, i.e. WSLA and WS-Agreement, are complementary with respect to the SLA management activities support, one solution that could be adopted is to design a new SLA language that unifies the capabilities of these languages by extending them and encompassing some modeling constructs of the one to the other. One such paradigm is TrustCom, which is the only SLA language that has a good evaluation score across all the activities. Another solution would be to design a new SLA language that could use the best modeling features of the two standardized ones and explicitly model the missing features.

Another interesting result that derives from the above analysis is that SCLs do not fully support most of the SLA management activities apart from those of SLA Monitoring & Assessment and Settlement. This can be explained by the focus of SCL design on service functionality, which was inevitable during SCL modeling time. 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 electronic contract type,

they cannot be used for specifying SLAs unless they are extended appropriately. SWCL is a SCL language that could be easily extended as it has the best score among all SCLs 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.

7. 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 approaches has been conducted in order to reveal their strengths and weaknesses and 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 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 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 which are the ones specific to given aspects. This comparison uncovered many interesting findings, while also spotted particular aspects of under-performance concerning each cluster's approaches. The next three subsections summarize the most important of these findings and draw directions for further research and improvement.

7.1 Discussion on Service Quality Models

Various SQMs have been proposed, from small or flat categories of service quality attributes to sophisticated taxonomies containing many categories and attribute types. In order to compare these approaches in a fair and consistent manner, a set of criteria were devised characterizing the extensiveness, information richness, structure, generality, and applicability of the considered SQMs. Concerning the first four aspects, the evaluation results have shown a trend that the approaches are improving over the years. In average, the SQMs have a satisfactory category number, where each category contains a small quality attribute number. 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 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 to validate or enrich 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 service

performance and usability under the perspective of the persons who really use the service to satisfy their needs, so they also constitute critical service selection factors.

An important evaluation result shows that apart from the very initial approaches that focused on quality attributes 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 within the same but also across the layers. Inter-attribute dependencies are extremely important as they reveal the influence one attribute has on the other. In this way, the service monitoring, assessment, and adaptation activities can exploit them to perform dependency analysis in order to detect wrong monitoring facts and discover the components of the same or different layer that caused an SLO violation.

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

Concerning the applicability comparison aspect, 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 to annotate QSDs and SLAs which can be used across all service life-cycle activities but in specific scenarios. This is because this SQM contains a rather small amount of domain-independent quality attributes and thus could not be used to capture any possible case in which also some domain-dependent quality attributes are needed. Most of the remaining approaches provide a metric description for some of the included attributes which does not contain a precise assessment formula but an assessment rules set. As the latter rules can be further used to create precise assessment formulas, these approaches could be exploited in all the service life-cycle activities, if extended appropriately.

Based on the above analysis, no SQM can be considered as optimal according to its evaluation on all the considered criteria. In fact, for particular partitions of the comparison aspects, different approaches are distinguished as the best. Thus, a new SQM is needed that should combine the characteristics of the best approaches in all the considered aspects, describe all the possible but realistic inter-attribute dependencies, and include also data quality attributes.

7.2 Discussion on Service Quality Meta-Models

Many SQMMs have been proposed, which 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 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 capturing the aspects of formality, expressiveness, complexity, and applicability.

Concerning formality, the results have shown that the majority of the approaches use either ontologies or informal formalisms. The former formalism is widely selected in pure SQMMs, while the latter is the best modeling choice in the other

two partitions, i.e., the SLA-enabled and security-based ones. Moreover, a recend trend has been revealed for the pure and SLA-enabled SQMMs in using ontologies as their formalism. The 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 of the approaches. The first criterion evaluated the SQM richness of the SQMM. The evaluation results 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 evaluated the richness of the quality metric model and its evaluation results 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. Finally, the third criterion evaluated 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 some pure and SLA-enabled SQMMs that encompass a very rich (excellent) constraint model. Moreover, pure SQMMs are improving on this matter over the years. Security-based SQMMs perform moderately in this aspect. 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 SQMM complexity 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 SQMMs 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 evaluated the connection of an SQMM with a Service Functional Specification Language (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 assessed if any service discovery and negotiation framework has adopted the SQMM under inspection. By inspecting the functionality of existing frameworks that use pure SQMMs and considering the fact that these SQMMs do not model some critical information for service monitoring and assessment, it is inferred that pure SQMMs are used until the service negotiation activity. The same result goes for security-based SQMMs. On the other hand, if the SLA-enabled SQMMs are improved on some modeling aspects, then they can be used across the whole service life-cycle.

Based on the above analysis, there is no SQMM that scores the best value in all criteria. This drawback prevents 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.

7.3 Discussion on Service Level Agreement Meta-Models

Two agreement language types 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 electronic contract type. Both language types were evaluated on a set of criteria that were grouped along the SLA life-cycle activities. These criteria assessed these languages along the lines of the information they can describe which is required 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 evaluation results, there is no language supporting in a satisfactory way all activities. On the contrary, for many activities a different language is awarded as the most appropriate one, while only few languages properly support a subset of all activities. In addition, there are some SLA management activities which are not properly supported by any language, including those of SLA description, matchmaking, and negotiation.

Concerning SLA languages, the analysis has shown that all SLA languages, including 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. Moreover, the capabilities of the WSLA and WS-Agreement languages are complementary with respect to the SLA management activities support. 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 as those features that are inadequate should be improved and those that are missing should be additionally modeled.

SCLs are not capable of fully supporting most of the SLA management activities apart from those of SLA Monitoring & Assessment and Settlement. This can be explained by the focus of SCL design on service functionality, which was inevitable during SCL modeling time. Thus, although these languages were designed to accommodate for any electronic contract type, they cannot be used to specify SLAs unless they are extended appropriately.

Based on the above analysis, there is a need for a new language able to express SLAs in a satisfactory way. Apart from satisfying all the 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 with the creation of several assisting tools or SLA management components for this new SLA language 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

The Acronyms Used Across the Paper and their Expansion

ACRONYM	ACRONYM EXPANSION
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
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.

Andrieux, A., Dan, A., Keahy, K., Ludwig, H., and Rofrano, J. 2004. Negotiability Con-

straints 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.

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.

De Paoli, F., 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.

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.

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.

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.

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.

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.

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

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.

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.

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.

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. 2004. DAML-QoS Ontology for Web Services. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*. IEEE Computer Society, San Diego, CA, USA, 472–479.