Semantic SLAs for Services with Q-SLA

Kyriakos Kritikos and Dimitris Plexousakis

ICS-FORTH
Heraklion GR-70013, Greece
{kritikos, dp}@ics.forth.gr

Abstract. This paper reports the re-engineering efforts for OWL-Q, a prominent semantic quality-based service description language. These efforts have focused on making OWL-Q more compact without reducing its level of expressiveness as well as enriching it with semantic rules towards semantic validation of quality specifications and new knowledge derivation. It also presents a new OWL-Q extension called Q-SLA advancing the state-of-the-art by covering all possible information aspects needed which along with the semantic rules enable proper and automatic support to all service management activities. A particular use-case is also provided to highlight the main benefits of Q-SLA.

Keywords: service, management, quality of service, SLA, agreement, semantics, ontology, rules, description, validation

1 Introduction

The main advantages that service-orientation delivers lead to the proliferation of available services such that the task of identifying tasks completing an application functionality is simplified. Such a proliferation heads towards the effect of having equivalent functionality offered via different quality of service (QoS) capabilities. As such, the role of QoS is quite important in discovering only those services which can satisfy the respective application's QoS requirements. In fact, as also advocated in [6], QoS can play an crucial role in all activities related to the service-based application (SBA) management.

Before QoS can be exploited in these activities, it has to be described. As such, various service description languages have been proposed either focusing on supporting service discovery or going beyond that. The latter languages can actually define Service Level Agreement (SLA) templates and actual SLAs. SLA templates can be used for service discovery and negotiation as they describe both the QoS requirements and capabilities of the service requester and provider, respectively. The SLA is then the successful outcome of a service negotiation representing the agreement between the two aforementioned parties towards the responsibilities involved during the delivery of the respective service concerned. This SLA is used as a guide for the realisation of the subsequent management activities. To this end, an SLA spans the whole lifecycle of services and SBAs.

Unfortunately, the proposed SLA languages cannot capture all appropriate information required for supporting the SBA management activities [6]. In fact, it has been proven that some SLA languages can play complementary roles such that they can be combined into a more complete SLA language. In addition, very few of them are semantic-based so as to enable both the syntactic and semantic SLA validation. In fact, semantics can be used to derive extra added-value knowledge and provide automated support to the SBA management activities.

OWL-Q is a semantic, quality-based service description language [7] able to specify: (a) service quality models involving quality terms, such as QoS attributes and metrics, and their relationships and (b) quality service profiles used for non-functional service discovery. Moreover, OWL-Q is coupled with both semantic alignment [8] and service discovery algorithms [9]. The former can be used to address the heterogeneity in service quality term description as it enables quality term matching and alignment. To cover the aforementioned gap and enable OWL-Q to be exploited beyond service discovery, this paper proposes a novel OWL-Q extension called Q-SLA towards expressing SLAs. This extension covers all required service management aspects. It is also coupled with semantic rules enabling the semantic SLA validation and derivation of added-value knowledge.

By being an extension to OWL-Q, Q-SLA covers also the quality term description, something not featured by other SLA languages. SLA alignment is also supported leading to a better and more accurate matching of SLA templates for service discovery and negotiation, something invaluable in addressing the current real-world situation where equivalent quality terms like quality attributes or metrics are described differently by different service actors. This characterises well the cloud computing domain and the respective specifications of the availability metrics involved in the SLA templates offered.

Another contribution of this paper lies on the re-engineering of OWL-Q to reduce its complexity as indicated in the evaluation conducted in [6] such that the modelling effort is alleviated. Such a re-engineering resulted in a great reduction in the number of ontology concepts and relationships. It also involved the specification of an extensive set of rules covering additional cases in the semantic validation of the quality-based specifications.

This paper also includes a proof-of-concept application of Q-SLA in a real use case which highlights its main benefits. It also conducts an extensive comparison of Q-SLA with the state-of-the-art to highlight the Q-SLA's main advancement.

The rest of the paper is structured as follows. Section 2 explicates OWL-Q re-engineering efforts. Section 3 analyzes the Q-SLA extension. Section 4 explicates Q-SLA's application on a certain use case. Section 5 reviews the related work. Finally, Section 6 concludes the paper and draws future work directions.

2 OWL-Q

OWL-Q is a semantic quality-based service description language built on top of OWL. It comprises various facets covering different service quality aspects, including quality attributes, metrics, and units. OWL-Q has now been re-engineered

to become more compact without losing its expressiveness by reducing the number of facets, concepts and properties. In fact, the rule of design thumb was that the modeller should supply the least possible amount of information and then inferencing can be exploited to derive the extra knowledge needed. As such, this has lead to the enrichment of the axioms involved in quality terms classes.

The re-engineering efforts also concentrated on enriching the SWRL [4] semantic rules used for semantic validation and new knowledge derivation. The rules were also categorised based on the concerned aspect such that the validation/derivation can focus only on that aspect, thus speeding up the respective tasks' execution. Concerning validation, the consistency rules specified focus on different aspects and attempt to detect semantic errors in the quality specification, such as recursiveness in metric and attribute composition, correctness of range limits in value types and metric scheduling correctness (e.g., schedule start is before its end). On the other hand, the new knowledge derivation rules concentrate on capturing various quality term matching caces covering quality metrics, attributes, units and value types. Due to paper size restrictions, the analysis will now focus mainly on the core OWL-Q content.

2.1 OWL-Q Facets

OWL-Q comprises 6 core facets which are shortly analysed to set up the context for better understanding the SLA extension proposed. Figures 1-2 cover all facets, including all major concepts and respective relationships, where different colours have been used to highlight the different aspects involved.

Central. This facet covers the modelling of concepts, which are either cross-aspect or top with respect to a certain aspect, and their respective relationships. OWL-Q cross-aspect concepts span quality categories to used for creating quality term partitions in quality models, domains which can be related to quality terms, and object lists [] representing instance lists of OWL-Q classes to remedy for the non-decidability when adopting RDF lists. Aspect-related concepts are analysed in the rest of the facets. OWL-Q also enables specifying generic object and data properties to be attributed to instances of any or a sub-set of OWL-Q classes. The generic object properties denote positive or negative dependencies between quality terms and quality term compatibility. The generic properties involve representing names, descriptions and values (mapping to any xsd data type).

Attribute. This facet covers quality attribute specification. It distinguishes between measurable attributes that can be measured by quality metrics and unmeasurable ones usually mapping to a fixed value set and unit. Attributes can also be composite when comprising or computed from other quality attributes. A quality attribute is associated to the level that it concerns. We distinguish between 5 levels: IaaS, PaaS, SaaS, WFaaS (workflow as a service) and BPaaS (business process as a service). In this way, we not only cover all possible service types to which an attribute can be associated but also converge the cloud and service computing worlds.

Metric. This facet covers the necessary details for specifying the way quality attributes can be measured. Such details are encapsulated through the concept

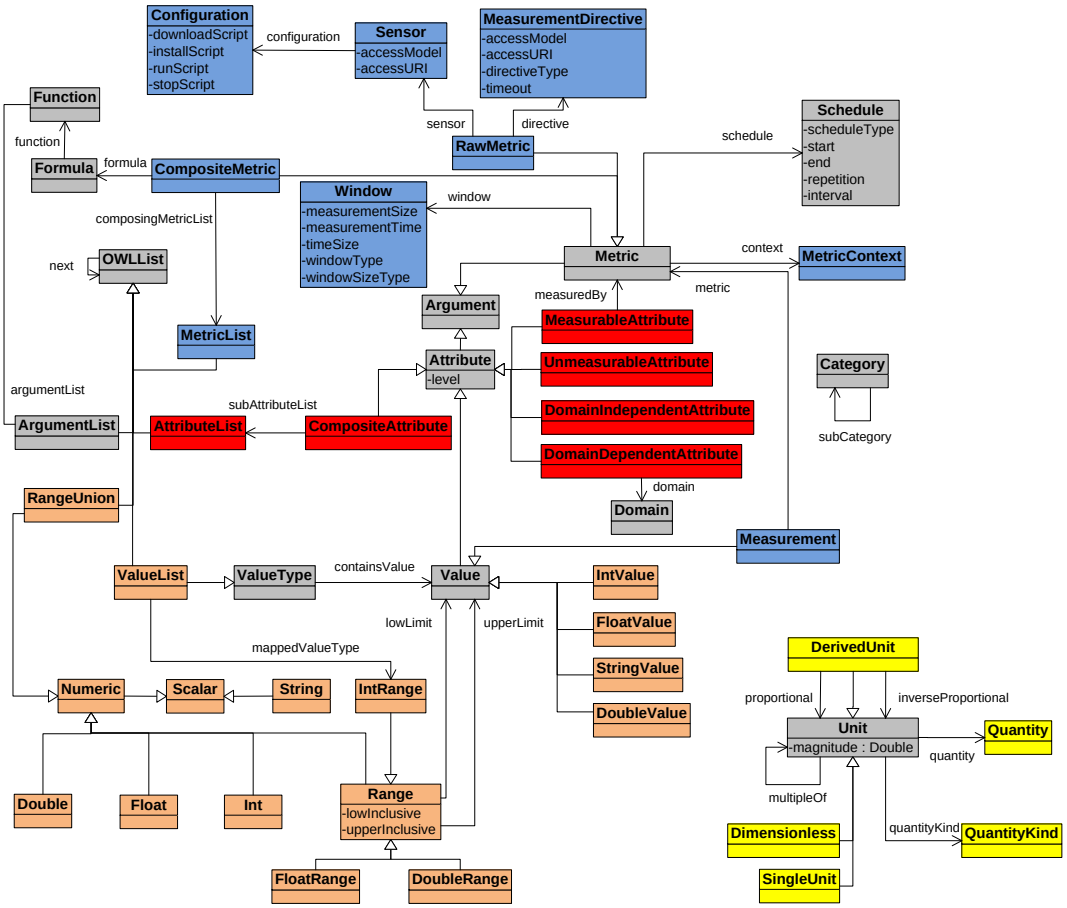

Fig. 1: OWL-Q's 5 out of 6 core facets

of a metric. Metrics can be classified as single or composite. The values of single metrics (e.g., *raw response time*) can be directly derived from the measurement system's instrumentation or from sensors. Composite metrics (e.g., *average response time*) are derived by applying a formula on a list of arguments, which can be constants or other formulas or metrics. Any kind of metric is associated to a value type (e.g., a range of integers in $(0, \infty]$ for execution time metrics) and a unit (e.g., *seconds*). Metrics are related to a certain *MetricContext* explicating scheduling and value aspects with respect to the measurement. *S*chedule class cover scheduling aspects in terms of when to start and end and how frequently to conduct metric measurement. Value aspects are covered by specifying a window of measurement indicating the amount of measurements to be considered for the (composite) metric computation. Through decoupling metrics from their context, the association of metrics to different schedules is enabled thus cover-

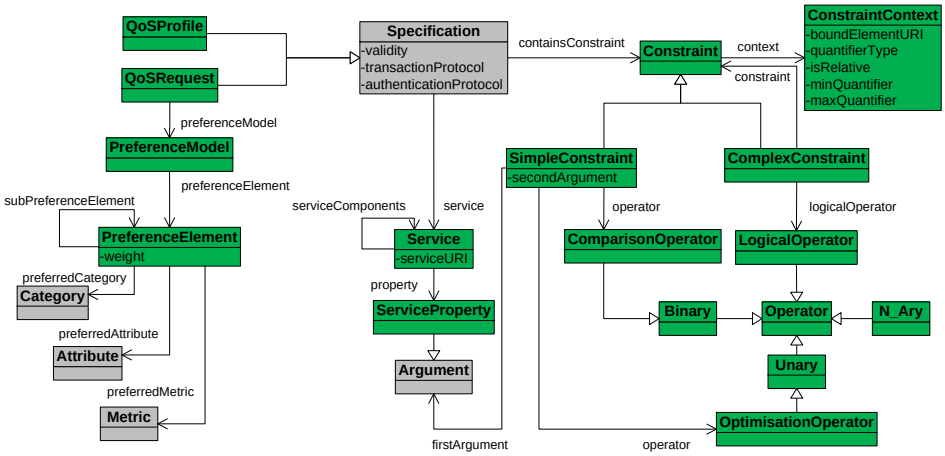

Fig. 2: OWL-Q's specification facet

ing the variability in metric scheduling and computation that can be exhibited in different monitoring systems. In addition, this facet covers defining actual measurements produced for a certain metric which are associated to a certain timestamp and value. As such, OWL-Q can support semantic databases operating over metric measurement data to infer interesting (event) patterns that, e.g., lead to Service Level Objectives (SLO) violations.

Unit. This facet concentrates on modelling units of measurements related to metrics. Units can be classified into single, derived or dimensionless. Derived units are computed from other units through dividing multications of different component units (e.g., *miles per second* is a division between the units of *miles* and *seconds*). Both single and derived units are associated to a dimension represented by the *QuantityKind* class and to a *Quantity*. For instance, a unit of *bytes per second* will be associated to both *speed* and *network speed* as quantity kind and quantity, respectively. This facet also covers unit compability through the *multipleOf* object property to denote the compatibility between units that are multiples of each other, such as *seconds* and *milliseconds*.

Value Type. This facet focuses on modelling of value types for metrics. Through such types we can check the validity of measurements, especially if they are produced through error-prone sources of information, like sensors or even humans. This facet involves the specification of two main classes, namely *Value* and *ValueType*. *Value* represents any kind of value which can be a component of a *ValueType*. Values are further classified into specialised sub-classes which map to widely-used XSD data types, such as integers and doubles. In addition, two specialised instances of Value have been developed to represent positive and negative infinity. *ValueType*s can be distinguished at the top-level into *Scalar* and *ValueList*. A *ValueList* is a list of values of the same type (e.g., integer). On the other hand, scalar value type which can be bounded or undounded. Unbounded

value types map to four main sub-classes, i.e., *Strings*, *Integers*, *Floats* and *Doubles*. Bounded value types are separated into ranges and unions of ranges. Ranges are characterised by two limits which might be or not included in the range and directly map to certain *Value*. Both limits should be of the same type (i.e., integers). Via the special value instances of infinity we can also represent semi-bounded ranges (e.g., $[1, \infty]$). The unions of ranges comprise non-overlapping ranges which contain the same kind of values (e.g., integers).

Specification. This facet focuses on specifying quality-based service descriptions which can be dinstinguished into QoS profiles and requests. A QoS request and profile represent the QoS requirements and capabilities for a particular service. Both specification types are linked to the respective *Service* concerned, which can be composite or single. Any kind of service is characterised by its URL. In this way, we actually abstract from the different ways the functional service description can be specified as a URL can map to the actual service endpoint (WSDL) or its actual web-based description in any kind of language, including semantic ones (but the URL and URI compatibility must be guaranteed in this case). Any specification has also a period on which it is valid.

A QoS request is associated to a PreferenceModel representing the requester's preferences on certain quality terms. Such a model takes a tree-like structure comprising nodes mapping to the respective quality term and its preference. The tree's top node represents the overall QoS while the rest of nodes map to different quality term types. The mappings from one node to its children denote the propagation of quality evaluations from lower to higher levels. For instance, consider the case of the *performance* category which could contain quality attributes nodes, like *response time* and *throughput*, with different preferences, like 0.6 and 0.4. The sum of all these preferences should equal to 1.0, while each preference denotes the relative importance of a child node towards determining its parent's quality value. Thus, if the normalised quality value of *response time* is 0.5 and 0.3 for *throughput*, then the normalised value for *performance* will be 0.42. Such a representation is in accordance to the Analytical Hierarchy Process [11]. It also provides significant support to the ranking of services after their matching and to the optimisation formula derivation for service composition problems.

Any kind of specification is associated to one (composite) constraint representing the set of quality capabilities or requirements that are offered or required, respectively, for the respective service at hand. Constraints can be distinguished into simple and composite. Simple constraints express conditions over a quality term's value. As such, a simple constraint is related to a quality term, a comparison operator, and a certain threshold value which should comply to the term's value type. On the other hand, composite constraints are logical combinations of constraints, expressed through well-known basic unary and n-ary logical operators, such as *NOT*, *AND* and *OR*.

A constraint also has a particular context associating the quality term condition to certain restrictions taking the form of: (a) the service or its parts (service object) for which the condition should hold denoted by associating the context with a URL pointing to the respective functional service specification part and

(b) a specification of how many relative service object instances must be accounted for and according to which way for the condition's evaluation (e.g., to express that a constraint violation occurs only when a certain subset of service object instances have measurements violating the respective condition).

3 Q-SLA

3.1 Extension Analysis

Q-SLA, depicted in Figure 3, has been developed as a sub-facet of the specification facet based on the original design rules for OWL-Q. This is a rational choice as an SLA is a kind of quality specification and many of the constructs in the original facet are actually re-used to specify the extension's respective constructs. In the following, we provide an analysis of the SLA sub-facet by focusing on certain important information aspects. The analysis is concluded with the supply of the respective rules that have been developed for this extension.

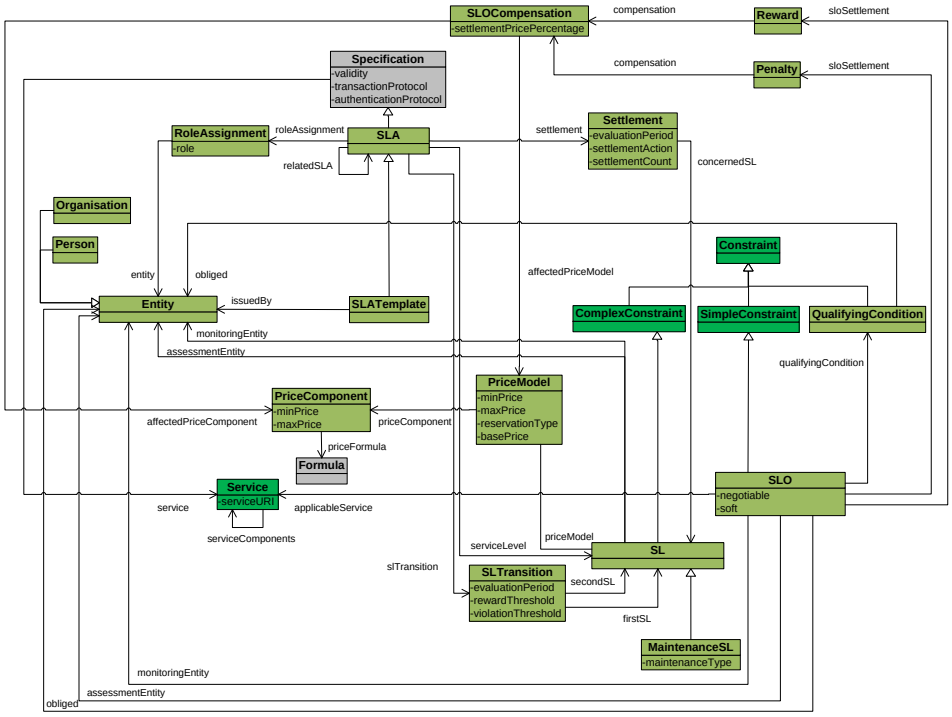

Fig. 3: The OWL-Q's SLA facet

SLA is considered as a sub-class of *Specification*. An SLA template in turn is a sub-class of *SLA* as it is a certain kind of SLA. An SLA comprises a set of

service levels (SLs) which explicate the different performance behaviors that a certain service can exhibit. Such a *SL* can denote normal performance behavior or behavior exhibited, e.g., during maintenance periods for the respective service (see *MaintenanceSL* class). A SL is considered as a kind of composite constraint as it explicates a set of particular quality capabilities to be delivered by the service concerned. Such capabilities are denoted via a specialised sub-class of single constraint called *SLO*, thus inheriting the respective condition and context information aspects. In addition, SLOs include the following information aspects:

1. a *qualifyingCondition* which is a condition that must hold in order for the SLO to be valid for assessment and possible compensation. Such a condition can refer to contextual restrictions at the customer side such as the number of concurrent incoming requests that can be served in a certain time period
2. the services on which the SLO applies – parts of a composite service or the whole composite service can be concerned
3. the *obliged* entity to guarantee the SLO
4. the entities responsible for the SLO's monitoring and assessment
5. a settlement in the form of a penalty or reward. It is highly important to include both settlement types in a SLA contract as this will motivate service providers to deliver even better SLs. As such, we consider that SLOs related to thresholds on worse values that the SLO quality term can take should be associated to penalties and SLOs related to thresholds on better values to be associated to rewards.
6. *negotiability* – we specify whether the SLO is soft or negotiable. This information aspect is relevant only for SLA templates as it can be exploited in the discovery and negotiation activities. It can be used in service discovery to better address over-constrained QoS requirements such that always a matchmaking solution can be derived. It can also enable more flexible service negotiations, by indicating whether the ranges of values promised or required for a certain quality term can be negotiable or not.

A *MaintenanceSL* is a kind of SL associated to an enumerated data type denoting the different maintenance types that can occur. We foresee three options for this: (i) on-demand, (ii) at particular time points, and (iii) both former options hold. Moreover, we enable moving from one SL to another via the notion of an *SLTransition*. This is obvious when we must transit from a normal to a maintenance SL. However, we also distinguish the two extra cases of downgrading or upgrading from one SL to another. The latter transition types can occur either on-demand (by clients when desiring to increase the SL to enable their applications to exhibit better performance levels or to decrease the SL to reduce costs) or when certain situations occur, such as the violation or surpassing of a certain number of SLOs in overall or for a certain time period. So as to specify all possible transition types, an SLA is associated to a SL transition via the *slTransition* object property. The extra flexibility enabled via modelling SLs and their transitions must be highlighted. As such, we support specifying flexible SLAs that do not have to be repeatedly re-negotiated when critical situations occur but gives the freedom to the signatory parties to explicate the most suitable

service performance behaviours and their allowed transitions. By agreeing on these SLs both parties are satisfied: (a) the client as all these SLs should match his/her requirements; (b) the service provider as it can decide on-demand when to transit from one SL to another, e.g., when increased customer load occurs such that it can still satisfy the performance requirements on lower SLs. By servicing more clients on lower SLs, the service provider can also maximise its gains.

To address service charging, a SL is related to a price model used to calculate the overall service cost. As such, as long as the SLA is in the respective SL, the charging is performed by this SL's price model. A price model comprises price components that must be added to produce the service cost. Each price component focuses on one cost aspect. It is computed via a formula over quality terms and service-specific attributes. For instance, a price component can focus on the resources provided by a IaaS and calculate the cost, based on the reservation type, as the amount of resource usage hours times the IaaS cost. Other components could then focus on network resources and data exchange costs.

Both a price model and its components can be associated to maximum and minimum price limits above and under which the service cost cannot move, respectively. Such cost constraints will hold irrespectively of whether the sum of a price model's components violate them. As such, a price model can be used as the means to guarantee the minimum possible gain even in SLO violations. Both the price model and its components are associated to a monetary unit (e.g., euros). The price model is also related to a reservation type stating whether the charging can be performed on-demand, via advanced reservation or spot prices.

A price model covers the normal service cost but the actual cost depends on whether SLO violations or surpasses have occurred. As such, the *Penalty* and *Reward* concepts were included to indicate the compensation kind that can be involved during an SLO assessment. Both concepts are associated to an *SLOCompensation* explicating the cost modifications to be performed. An SLO compensation is associated to the affected price model and price components in particular and indicates the percentage of cost to be discounted or rewarded.

The following must be highlighted. First, as there can be many SLO violations for a certain SL, the SL's price model guarantees that the service cost will not go beyond a certain threshold such that a minimum provider gain is guaranteed. Second, additional flexibility is allowed via SL transitioning. For instance, if the amount of SLO violations is big, it can be rational to move to a lower SL such that the provider still keeps up with its promises and receives a reduced amount of money again guaranteeing his/her gains. Without such lower SLs, the danger of SLA contract termination exists due to provider under-performance which could easily lead, if done frequently, in provider reputation reduction.

The SLA is also associated to a settlement when critical situations, not covered by it, occur. One such critical situation can map to the SLA being at the lowest SL and the number of SLO violations overpasses the limit posed. To this end, we have reached a point where a drastic action should be taken. Such an action can include the re-negotiation of the SLA or its cancelling.

While the previous analysis indicates that an SLA via Q-SLA can impose quality constraints on a composite service along with its components, there is another situation that can occur which Q-SLA still captures it. Suppose that a BPaaS is offered by a cloud provider. For this BPaaS, an SLA hierarchy can be posed where at the highest level there is an SLA between the provider and its customers, while at the lower levels there are SLAs involving SaaS/PaaS/IaaS services supporting the functionality and execution of the BPaaS. The latter SLAs involve the BPaaS cloud provider as a client and the providers of the exploited services. To capture such situations, a light integration approach is followed for the next two reasons: (a) the specification of composite SLAs raises the modelling complexity; (b) in typical and most common cases, the different SLAs are independently negotiated (e.g., in a top-down manner where first the top SLA is negotiated and then the rest). For instance, a BPaaS provider might form SLAs with SaaS/IaaS providers to properly support the provisioning of many BPaaS instances and then independently offer different SLA types (gold, silver and bronze) to different customer types. As such, the BPaaS provider should form these SLAs such that a violation of the top-level SLA due to a violation of a lower-level one will not lead to paying more than the penalty gained from the lower SLA. As such, the following SLA relationship types can be modelled: *dependsOn* indicating that one SLA depends on another one (e.g., a BPaaS SLA depends on SaaS/IaaS SLAs). This dependency type is considered as lightweight as it does not impose any tight integration. For legal issues, as an example, the BPaaS provider might indicate that some critical situations could be due to reasons out of its control such that the respective dependant SLAs can be referenced. In the future, we might consider modifying such modelling, in case the way SLAs are negotiated or specified is modified.

Apart from the modelling at the class level in terms of concepts, relationships and axioms, Q-SLA is also accompanied by a set of three semantic validation rules focusing on: (a) checking based on the respective metric's monotonicity and SLO's comparison direction whether the SLO should be associated to a penalty or reward; (b) checking whether the max price in a price model or component is always greater or equal to the min price; (c) checking whether there are no circles in SL transitions according to one direction (e.g., downgrading) without considering the maintenance SL. Such validation rules coupled with those generally applying for any specification kind equip modellers with an instrument that guides them towards specifying only semantically correct SLA descriptions.

4 Use-Case Application

The use-case concerns developing a traffic management application with the goal to monitor environment variables and sense critical situations and react via regulating the traffic accordingly such that accidents are rapidly addressed and pollution indicators do not exceed certain thresholds. Such an application includes three main components offered as a service: (a) a monitoring component sensing the environment conditions; (b) an analysis component (AC) obtaining

the monitored information and deriving a traffic management plan; (c) a traffic regulator component executing the plan produced by *AC*. The first and third components have been developed internally by the municipality as they regard sensitive data and own infrastructure manipulation. On the other hand, to reduce costs due to the heavy workload of *AC*, the municipality has decided to outsource *AC* to the *SP1* provider. To this end, it has to form an SLA with *SP1* involving the specification of the offered service's expected quality behaviour and the respective penalties to be enforced in deviations of this behaviour.

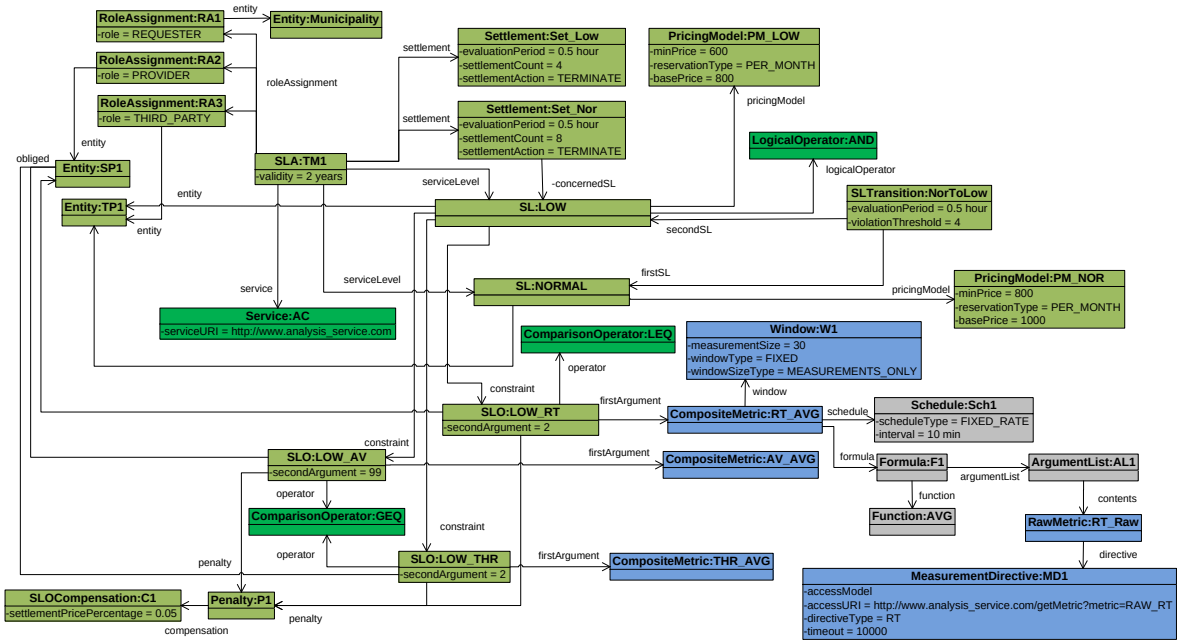

Fig. 4: Q-SLA's use-case application

The SLA to be signed (see its snapshot in Figure 4) will hold for two consecutive years. It includes, apart from the two signatory parties, a third trusted one TP_1 to take care of SLO monitoring & assessment and SLO violation reporting to the signatory parties. This SLA involves three SLs: (a) normal; (b) low; (c) maintenance. The normal SL maps to the following SLOs: $rt \leq 1$ min, $av \geq 99.99\%$, $thr \geq 6$ reqs/min, where rt maps to response time, av to availability and thr to throughput. The low SL comprises in turn the following SLOs: $rt \leq 2$ min, $av \geq 99\%$, $thr \geq 2$ reqs/min. The former two SLs include delivering quality capabilities that match the municipality's expectations. The normal SL is initially selected as the municipality is divided into six regions and all regions can be serviced concurrently in case of rush hours – this is actually satisfied through the constraints for *response time* and *throughput*. On the other hand, it is ac-

ceptable if one third of the regions can be concurrently serviced as this maps to an average situation characterised by still confronting the rush hours in a more limited manner but also (the more extended) non-rush hours in an excellent manner via the low SL which also leads to lower costs for the municipality.

The maintence SL is transitioned at every midnight, it lasts one hour and maps to the lowest possible SL. The municipality is satisfied even with this level as during very late hours, the traffic is minimal. The respective SLOs mapping to this SL are as follows: $rt \leq 6$ min, $av \geq 80\%$, $thr \geq 0.5$ reqs/min.

The transit from normal to low SL is enabled in case of 4 SLO violations in a time period of half an hour. The municipality is also entitled to end the contract when 4 violations occur within half an hour in the low SL for rush hours and 8 violations in the same SL for non-rush hours. The pricing of AC service for each SL is constant: 1000 euros per month for the normal SL and 800 euros for the low one. Each SLO violation in a SL maps to a 5% discount. For the normal SL, the service price cannot go under 800 euros (accounting for 4 violations), while for the low SL it cannot go under 600 (accounting for 5 violations). A great number of violations in the high SL does not necessarily mean that the we must transit to the low SL. This depends on the time period in which a percentage of the total number of violations has occurred. On the contrary, a great number of violations leads to approaching or reaching the minimum price limits for a SL.

The three main properties are measured via code which intervenes in the traffic monitoring application and is provided by the third-party organisation, according to the following metrics:

- Average response time evaluated in a time period of 10 minutes with a time window of 30 measurements.
- Average availability evaluated every 10 minutes with a window of 10 measurements. It is computed from raw availability evaluated every 1 minute via dividing the number of times the service was up with this time period.
- Average throughput evaluated every 10 minutes with a window of 5 measurements. It is computed from raw throughput evaluated every 2 minutes by dividing the number of requests served with this time period.

Figure X visualises all aforementioned information in form of a Q-SLA/OWL-Q snapshot. Q-SLA can specify all this information, including the handling of rush hours encapsulated as qualifying conditions on the respective SL's SLOs.

5 Related Work

In this section, we focus on analysing only highly related work in SLA specification. The analysis relies on the extensive set of comparison criteria in [6] organised according to the activity in the service management lifecycle concerned. These criteria assess the completeness and suitable expressivity in covering all required information to properly support each management activity for any SLA language. As in [6] many SLA languages have already been reviewed, in this paper we concentrate on reproducing evaluation results for the most significant SLA languages as well as providing results for new ones, including Q-SLA.

In the sequel, we first shortly analyse the comparison criteria, then we present the comparison table derived and finally we discuss the results encapsulated by this table. The comparison criteria are analysed according to the management activity that they concern in the following paragraphs.

Description. This activity includes four main criteria deemed as important apart from an SLA language's ability to express quality terms: (a) the formalism in SLA description; (b) the coverage of both functional and non-functional aspects; (c) the re-usability in terms of SLA constructs to be used across different SLAs; (d) the ability to express composite SLAs.

Discovery. This activity includes four criteria: (a) *metric definition* mapping to the ability to refer but also define quality metrics; (b) *alternatives* – the ability to specify alternative SLs; (c) *soft constraints* – the ability to pose soft constraints to address over-constrained requirements; (d) *matchmaking metric* – existence of a metric explicating how specification matching can be performed.

Negotiation. This activity includes two criteria: (a) *meta-negotiation* – related to supplying information to support negotiation establishment; (b) *negotiability* – the ability to indicate which quality terms are negotiable and in which way.

Monitoring. For monitoring, it is imperative that an SLA language can define: (a) the *metric provider* responsible for performing the monitoring and (b) the *metric schedule* indicating how often the SLO metrics must be measured.

Assessment. The major criteria for the assessment activity include: (a) the *condition evaluator*, i.e., the party responsible for SLO assessment; (b) the ability to specify a *qualifying condition* for SLOs; (c) the ability to define the *obliged* party for delivering an SLO; (d) the ability to define the SLO*assessment schedule*; (e) the ability to define the *validity period* in which an SLO is guaranteed; (g) the ability to define *recovery actions* to remedy for SLO violations.

Settlement. The settlement with respect to particular situations comes with the ability of the SLA language to define: (a) *penalties*, (b) *rewards* and (c) settlement actions required for the final outcome of an SLA

Archive It is concerned with the ability to specify the SLA's *validity period.*

Table 1 summarises the evaluation results with the criteria as rows and the compared SLA languages as columns, while each cell indicates how well a respective language satisfies a certain criterion.

As it can be seen, Q-SLA is the best approach mapping to the best performance in all activities. Linked USDL Agreement (LUA in short) seems to come second but has the worst possible performance for two activities. The aforementioned languages are over-ruled only in the settlement activity by RBSLA.

These evaluation results also unveil places for Q-SLA improvement, mainly with respect to composability, meta-negotiation and recovery actions' criteria. Via such improvement, Q-SLA will reach its final goal towards becoming a complete semantic SLA language. This outcome could promote Q-SLA to become a standard or converge to a standard via joining similar standardisation efforts. Such a standard is currently missing in service and cloud computing and would definitely provide an added-value to many different aspects, including service management and better capturing of customer requirements.

Table 1: Evaluation results of SLA languages

Life-cycle Activity	Criteria	WSLA [5]	WS-A [1]	WSOL [12]	RBSLA [10]	LUA [3]	SLALOM [2]	Q-SLA
Description	Formalism	Informal	Informal	Informal	RuleML Ontologies	Ontology	UML	Ontology
	Coverage	[p,y]	[y,p]	[p,p]	[p,y]	[y,y]	[p,y]	[p,y]
	Reusability	yes	yes	yes	yes	yes	yes	yes
	Composability	no	fair	no	no	no	no	fair
Matchmaking	Metric Definition	yes	no	no	yes	no	yes	yes
	Alternatives	impl	impl	impl	impl	no	no	yes
	Soft Constraints	no	yes	no	no	no	no	yes
	Matchmaking Metric	no	no	no	no	no	no	yes
Negotiation	Meta-Negotiation	poor	fair	poor	poor	no	no	good
	Negotiability	no	part	no	no	no	no	yes
Monitoring	Metric Provider	yes	no	yes	no	yes	no	yes
	Metric Schedule	yes	no	no	yes	yes	no	yes
Assessment	Condition Evaluator	yes	no	yes	no	yes	no	yes
	Qualifying Condition	impl	yes	no	no	yes	no	yes
	Obliged	yes	yes	yes	yes	yes	yes	yes
	Assessment Schedule	yes	no	no	no	yes	no	yes
	Validity Period	yes	no	no	yes	yes	no	yes
	Recovery Actions	yes	no	yes	yes	no	no	no
Settlement	Penalties	no	SLO	SL	SL	SLO	SLO	SLO
	Rewards	no	SLO	no	SL	SLO	no	SLO
	Settlement Actions	yes	no	no	yes	no	no	yes
Archive	Validity Period	yes	yes	no	no	yes	yes	yes

6 Conclusions

This paper has proposed an extension to OWL-Q called Q-SLA to cater for specifying SLAs. This extension was designed to fill the gaps with respect to the capturing of information supporting all activities in the service-based application management. In addition, it is coupled with semantic rules enabling the semantic SLA validation such that the modellers are guided to specify only semantically correct SLA descriptions. This paper also reveals the great re-engineering effort on OWL-Q resulting in the reduction of its complexity and in producing of an axioms set and semantic rules which not only enable validating semantic quality models and quality-based service descriptions but also deriving new knowledge. This knowledge, for the time being, concerns matching quality terms from different OWL-Q quality specifications to assist in their alignment.

The paper has also provided a proof-of-concept application of Q-SLA on a specific use-case highlighting its main benefits, especially with respect to the increased flexibility via which SLAs can be expressed. It has also included a specific SLA language evaluation showcasing Q-SLA's superiority.

Particular research work directions have been planned. First, further validating Q-SLA/OWL-Q according to additional use-cases. Second, more complete SLA relationship handling. Third, coupling Q-SLA/OWL-Q with an editor enabling non-expert users in ontology modelling to specify SLAs and other types of quality-based service specifications. Fourth, developing a complete service-application management framework based on OWL-Q. Finally, developing adapters which transform non OWL-Q specifications to OWL-Q ones thus enabling the aforementioned framework to work with many other quality-based service languages. This will certainly promote using OWL-Q as well as reducing the modeller effort, when existing quality specifications are already in place.

Acknowledgements The research leading to these results has received funding from the European Commission's Framework Programme for Research and Innovation HORIZON 2020 (ICT-07-2014) under grant agreement number 644690 (CloudSocket).

References

1. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture. W3c working draft, W3C (Nov 2002), available at `http://www.w3.org/TR/ws-arch`
2. Correia, A., e Abreu, F.B., Amaral, V.: SLALOM: a language for SLA specification and monitoring. CoRR abs/1109.6740 (2011), `http://arxiv.org/abs/1109.6740`
3. García, J.M., Pedrinaci, C., Resinas, M., Cardoso, J., Fernández, P., Ruiz-Cortés, A.: Linked USDL Agreement: Effectively Sharing Semantic Service Level Agreements on the Web. In: ICWS. pp. 137–144. IEEE Computer Society (2015)
4. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C, http://www.w3.org/Submission/SWRL/ (2004)
5. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management 11(1), 57–81 (2003)
6. Kritikos, K., Pernici, B., Plebani, P., Cappiello, C., Comuzzi, M., Benbernou, S., Brandic, I., Kertesz, A., Parkin, M., Carro, M.: A Survey on Service Quality Description. ACM Computing Surveys 46(1) (2013)
7. Kritikos, K., Plexousakis, D.: Semantic QoS Metric Matching. In: ECOWS. pp. 265–274. IEEE Computer Society (2006)
8. Kritikos, K., Plexousakis, D.: Towards Aligning and Matchmaking QoS-based Web Service Specifications, chap. 1, pp. 216 – 257. IGI Global, USA (July 2012)
9. Kritikos, K., Plexousakis, D.: Novel optimal and scalable nonfunctional service matchmaking techniques. IEEE T. Services Computing 7(4), 614–627 (2014)
10. Paschke, A.: RBSLA: A declarative Rule-based Service Level Agreement Language based on RuleML. In: CIMCA-IAWTIC. pp. 308–314. IEEE Computer Society, Vienna, Austria (2005)
11. Saati, T.: The Analytic Hierarchy Process. McGraw-Hill (1980)
12. Tosic, V., Pagurek, B., Patel, K.: WSOL - A Language for the Formal Specification of Classes of Service for Web Services. In: Zhang, L.J. (ed.) ICWS. pp. 375–381. CSREA Press, Las Vegas, Nevada, USA (2003)