Semantic SLA for Clouds: Combining SLAC and OWL-Q

Kyriakos Kritikos[1] and Rafael Brundo Uriarte[2]
[1]*ISL, ICS-FORTH, Heraklion, Greece*
[2]*IMT School for Advanced Studies Lucca, Italy*

Keywords: SLA, Cloud Computing.

Abstract: Several SLA languages have been proposed, some specifically for the cloud domain. However, after extensively analysing the domain's requirements considering the SLA lifecycle, we conclude that none of them covers the necessary aspects for application in diverse real-world scenarios. In this paper, we propose SSLAC, where we combine the capabilities of two prominent service specification and SLA languages: OWL-Q and SLAC. These languages have different scopes but complementary features. SLAC is domain specific with validation and verification capabilities. OWL-Q is a higher level language based on ontologies and well defined semantics. Their combination advances the state of the art in many perspectives. It enables the SLA's semantic verification and inference and, at the same time, its constraint-based modelling and enforcement. It also provides a complete formal approach for defining non-functional terms and an enforcement framework covering real-world scenarios. The advantages of SSLAC, in terms of expressiveness and features, are then shown in a use case modelled by it.

1 INTRODUCTION

The services provided in the cloud require formal guarantees in the form of an SLA for delivering a certain service level. However, the large majority of providers offer only a textual description of the SLA terms and conditions, without any formal guarantee. This textual approach has many drawbacks; for instance, it leads to ambiguity and difficulties in automating the service discovery and negotiation activities.

A Service Level Agreement (SLAs) is a formalised contractual document representing the obligations and rights for the signatory parties in the delivery of the (cloud) service concerned. Several SLA languages have been proposed, even tailored exclusively for the cloud domain. However, after a comprehensive analysis of domain requirements (see Section 2), we can conclude that none of them captures all possible aspects required to support the SLA management lifecycle.

In light of this gap, this paper proposes SSLAC (Semantic SLA for Clouds), which combines two of the most prominent service specification and SLA languages: OWL-Q (Kritikos and Plexousakis, 2006) and SLAC (Uriarte et al., 2014). These languages have complementary features and their combination advances the state of the art from many perspectives. It enables, in the same combined language, semantic verification and inferencing as well as constraint-based SLA modelling and validation. Moreover, it provides a complete formal approach for defining non-functional terms and modifying them on demand as well as an enforcement framework for real-world scenarios.

To enable a practical implementation and ease of use, we took as a base the definition of SLAC. Based on its complementarity with OWL-Q, we first analysed SLAC's main gaps and extended it to cover some of them. The rest of the gaps were filled in by cross-referencing OWL-Q (term) specifications to cover the needed expressiveness as well as possess the suitable analysis power. The advantages of the result, a powerful SLA definition language and framework in terms of expressiveness, features, and SLA lifecycle coverage, are finally shown via its application on a real use case.

The rest of the paper is organised as follows. Section 2 systematically analyses the related works and unveils the shortcomings of the existing languages. Section 3 describes the extensions on SLAC and its combination with OWL-Q. Section 4 presents an use case on which SSLAC and its framework have been applied. Finally, Section 5 concludes the paper.

2 RELATED WORK AND BACKGROUND

The various SLA languages proposed differ in level of expressiveness, formality and compactness. To review

Kritikos, K. and Uriarte, R.
Semantic SLA for Clouds: Combining SLAC and OWL-Q.
DOI: 10.5220/0006299804320440
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 404-412
ISBN: 978-989-758-243-1

them, a synthesis of two frameworks was constructed: (a) the framework in (Kritikos et al., 2013), which was defined according to the service lifecycle; and (b) the one in (Uriarte et al., 2014), which defines a complementary set of criteria and focus on description, model validation capabilities and tool support. The latter is discussed, while the details of the former can be found in (Kritikos et al., 2013).

In the sequel, we describe the evaluation criteria of our comparison, summarise the results in Table 1 and discuss these results. We analyse the following SLA languages: WSLA (Keller and Ludwig, 2003), WS-Agreement (Andrieux et al., 2007), WSOL (Tosic et al., 2003), RBSLA (Paschke, 2005), Linked USDL (LUA for short) (Pedrinaci et al., 2014), SLALOM (Correia et al., 2011) and SLAC (Uriarte et al., 2014).

Description refers to (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; (e) the coverage of the cloud domain (wrt. cloud service types); (f) price model coverage (schemes & computation model); (g) dynamicity (i.e., capability to move between service levels (SLs) or modify SLOs based on certain conditions or per request); (h) (model) validation capabilities (i.e., capability to perform syntactic, semantic and constraint validation of SLAs); and (i) editor support. For the cloud *Coverage*, the evaluation considers whether the SLA language covers all cloud service types and if it is generic enough (assessed as 'a' denoting this ability), and whether it can cover all or some service types by providing respective cloud domain terms ('y' means all service types, 'p' means some). Non-coverage is denoted by 'n'.

Price model defines if a language: 'n': does not support price models; 'p': covers only pricing schemes; 'y': covers also the price computation model. Dynamicity denotes: (a) 'n': if the language does not cover this aspect; (b) 'SLO': if it covers it at the SLO level; (c) 'SL': if it covers it at the SL level enabling to transition from one SL to another or to modify a SL.

The language evaluation over its validation capabilities can map to multiple values: (a) 'n': no validation capabilities are offered; (b) 'sy': syntactic validation is enabled; (c) 'se': semantic validation is enabled; (d) 'c': constraint-based validation is enabled.

A language can provide: 's': a domain-specific *Editor*; 'g': a generic one; 'n': no editor. The *Discovery* criterion includes: (a) *metric definition*, which refers and also define quality metrics; (b) *alternatives*, which specifies alternative SLs; (c) *soft constraints*, which uses soft-constraints to address over-constrained requirements; (d) *matchmaking metric*, which supports metrics explicating the specification matching.

Negotiation. Meta-negotiation refers to the supply of information to support negotiation establishment; and *negotiability* to the ability to indicate in which way quality terms are negotiable. For *Monitoring*, an SLA language should define: (a) the *metric provider* responsible for the monitoring and (b) the *metric schedule* indicating how often the SLO metrics is measured.

Assessment defines: (a) the *condition evaluator*, i.e., the party responsible for SLO assessment; (b) *qualifying condition*s for SLOs; (c) the *obliged* party to deliver an SLO; (d) the SLO*assessment schedule*; (e) the *validity period* in which an SLO is guaranteed; (g) *recovery actions* to remedy for SLO violations.

The *Settlement* with respect to particular situations enables the definition of: (a) *penalties*; (b) *rewards*; (c) settlement actions. *Archive*, instead, is concerned with the ability to specify the SLA's *validity period*.

Management. *Framework* assesses whether an open- or closed-source management framework has been built on top of a SLA language. The respective assessment values are: (a) 'o': open-source framework exists; (b) 'y': framework exists but is not open-source; (c) 'n': no framework is available.

Based on the evaluation results in Table 1, no SLA language scores well in all criteria across all life-cycle activities. By considering all criteria, we could nominate WSLA and LUA as the most prominent languages but they still need to be substantially improved.

Concerning the description activity, SLAC (Uriarte et al., 2014) is the best, since it exhibits a good composability and dynamicity levels. Pricing models are also partially covered, a domain-specific editor is also offered and it is the sole language supporting constraint-based SLA model validation.

As far as matchmaking and negotiation activities are concerned, WS-A (Andrieux et al., 2007) seems to be slightly better than the rest, especially with respect to the second activity. However, matchmaking is not actually well covered by any language. For monitoring and assessment, LUA and WSLA seem to be the best languages, with WSLA being slightly better on the assessment part and the recovery action coverage. More than half of the languages provide a SLA management framework. Most of these are open-source, enabling possible adopters to extend it according to their needs.

Based on the above analysis, no SLA language prevails; so there is a need to either introduce yet another SLA language or improve an existing one. In this paper, we take the second direction and combine the capabilities of the OWL-Q (Kritikos and Plexousakis, 2006) and SLAC (Uriarte et al., 2014) languages to offer a language agglomeration that advances the state-of-the-art. The last column in Table 1 depicts the

Table 1: Evaluation results of SLA languages.

Life-cycle	Criteria	WSLA	WS-A	WSOL	RBSLA	LUA	SLALOM	SLAC	SSLAC
Description	Formalism	Informal	Informal	Informal	RuleML Ontologies	Ontology	UML	CSP	CSP Ontology
	Coverage	[p,y]	[y,p]	[p,p]	[p,y]	[y,y]	[p,y]	[p,p]	[p,p]
	Reusability	yes	yes	yes	yes	yes	yes	yes	yes
	Composability	no	fair	no	no	no	no	good	good
	Cloud Domain	a	a	a	a	a	y	p	p
	Price Model	n	n	n	n	y	n	p	y
	Dynamicity	n	n	n	SL	n	n	SLO	SLO
	Validation	sy	sy	sy	sy,se	sy,se	sy	sy,c	sy,se,c
	Editor	g	g	g	g	s	s	s	s
Discovery	Metric Definition	yes	no	no	yes	no	no	no	yes
	Alternatives	impl	impl	impl	impl	no	no	no	yes*
	Soft Constraints	no	yes	no	no	no	no	no	yes*
	Matchmaking Metric	no	no	no	no	no	no	no	yes*
Negotiation	Meta-Negotiation	poor	fair	poor	poor	no	no	no	good*
	Negotiability	no	part	no	no	no	no	no	yes*
Monitoring	Metric Provider	yes	no	yes	no	yes	no	no	yes
	Metric Schedule	yes	no	no	yes	yes	no	no	yes
Assessment	Condition Evaluator	yes	no	yes	no	yes	no	no	yes
	Qualifying Condition	impl	yes	no	no	yes	no	yes	yes
	Obliged	yes	yes	yes	yes	yes	yes	yes	yes
	Assessment Schedule	yes	no	no	no	yes	no	no	yes
	Validity Period	yes	no	no	yes	yes	no	yes	yes
	Recovery Actions	yes	no	yes	yes	no	no	yes	yes
Settlement	Penalties	no	SLO	SL	SL	SLO	SLO	SLO	SLO
	Rewards	no	SLO	no	SL	SLO	no	yes	yes
	Settlement Actions	yes	no	no	yes	no	no	yes	yes
Archive	Validity Period	yes	yes	no	no	yes	yes	yes	yes
Enforcement	Framework	y	o	o	n	n	n	o	o

respective agglomeration result. As it can be seen, this agglomeration maps now to the best language covering nearly all aspects. Only the description and negotiation activities are yet not fully covered; full coverage is considered as future work direction.

The SSLAC evaluation values with asterisk denote individual OWL-Q features. Since the combination of SLAC and OWL-Q covers all lifecycle activities, in Section 3 we describe their actual combination.

2.1 Background

We now analyse the OWL-Q and SLAC languages to set up the background necessary to understand how their combination is realised in the next sections.

2.1.1 OWL-Q

OWL-Q is a prominent (Kritikos et al., 2013) semantic non-functional service specification language, carefully designed into various facets to cover all appropriate measurability aspects. It can specify 2 types of specifications: (a) quality models defining hierarchies of non-functional terms, such as metrics and attributes, along with their relationships; (b) non-functional service profiles specifying service non-functional capabilities as constraints over non-functional terms. This paper focuses on the first specification type that enables enriching the SLA language capabilities.

OWL-Q supports 2 model validation types: (a) syntactic; (b) semantic based on OWL semantics and semantic rules incorporated in OWL-Q. It is assorted by algorithms supporting: (a) the semantic alignment of non-functional specifications based on their terms; (b) their matchmaking (Kritikos and Plexousakis, 2014); (c) service negotiation (Comuzzi et al., 2009).

OWL-Q comprises 6 facets: *general*, *attribute*, *metric*, *unit*, *value type*, and *specification*, each mapping to a different measurability aspect. The *general* facet includes generic concepts, properties and relations, the *attribute* facet specifies non-functional properties, while the *value type* facet describes domains of value for terms like metrics. The focus now is on the remaining facets as they are related to this paper contribution.

The *metric* facet explicates how a non-functional property can be measured via the conceptualisation of a *Metric*. A metric can be mapped to a unit of measurement and value type; it can be raw or composite. Raw metrics (e.g., raw CPU utilisation) can be measured by a sensor or a measurement directive over a service instrumentation system. Composite metrics (e.g., average CPU utilisation) can be measured via formulas that apply a mathematical or statistical function over a set of arguments. An argument can be a metric, an attribute, a service property or another formula. Metrics can be mapped to one or more metric contexts explicating their measurement frequency and window.

The *unit* facet focuses on specifying units. Any unit is associated to a quantity kind (e.g., speed) and quantity (e.g., light speed). A unit can be single, dimensionless or derived. Derived units are produced by dividing sets of other units (e.g., *bytes per second*). Single units (e.g., *second*) can no longer be decomposed. A dimensionless unit (e.g., *percentage*) is a single unit that does not map to a quantity kind.

The *specification* facet models non-functional service profiles. For this facet, we analyse 2 main concepts of interest: (a) condition contexts and (b) price models. Condition contexts are attached to simple constraints / conditions to denote their application context. They mainly map to the object being measured (e.g., IaaS service, SaaS input parameter) and the measurement context of the metric involved in the constraint.

*PriceModel*s represent the computation procedure to calculate a service's price. They also set hard constraints over the price's low and upper limits and its monetary unit (e.g., *euros*). Price models can be split into one or more *PriceComponent*s which explicate the way the service pricing can be computed based on certain aspect (e.g., IaaS or network utilisation). The overall service price equals the sum of the prices that can be computed from these components. A price component can also specify aspect-specific low and upper bounds over service price and is associated to a formula mapping to the price computation procedure.

2.1.2 SLAC

SLAC is a SLA language for cloud services, which focuses on: *(i)* formal aspects of SLAs; *(ii)* supporting multi-party agreements; *(iii)* ease-of-use; *(iv)* proactive management of cloud SLAs.

Its main differences with existing SLA languages are: it is domain specific; its semantics are formally defined to avoid ambiguity; it supports the main cloud deployment models; it enables specifying multi-party agreements. SLAC also comes with an open-source software framework[1] enabling cloud SLA specification, evaluation and enforcement. A novel mechanism over SLAC to cope with the cloud dynamism (Uriarte et al., 2016) has been proposed. Intuitively, the SLAC's formal semantics associates a set of constraints to each language term, which are evaluated, at design-time, to identify inconsistencies in the specification and, at run-time, to verify the compliance with monitoring data collected from the cloud system.

An SLA comprises a unique identifier and at least two involved parties. The agreement's terms express the features of the service together with their expected values. Each SLA requires defining at least one term, either a *Metric* or a *Group* of terms (which enables term re-use in different contexts). For each term, the party responsible to fulfil it is defined along with the contractors of the respective service. SLAC supports different metric types, e.g., numeric, constrained by open or closed value *Interval*s and a certain Unit. Finally, *Guarantees* specify the commitment to ensure the agreement terms and, in case of a term violation or other events, explicate the actions to be taken.

[1] `http://rafaeluriarte.com/slac/`

3 COMBINING OWL-Q AND SLAC

Section 2.1 highlighted OWL-Q and SLAC complementarity. OWL-Q can enable the semantic description of any kind of non-functional term, even those not covered by SLAC. As such we aim at combining these two languages together to produce a combination advancing the state-of-the-art (see Section 2).

This combination resolves the main SLAC drawbacks, shown in the 2nd last column of Table 1, to reach the combination's evaluation results, shown in the table's last column. These drawbacks are summarised as follows: (a) a higher formality degree is missing; (b) new metrics cannot be defined and only a fixed metric set, mainly on the IaaS level, is available; (c) similarly, the modeller can select only from a fixed unit set; (d) metric and condition context information is missing; (e) the metric measurement provider and condition evaluator are not specified; (f) pricing computation models cannot be expressed; (g) matchmaking metrics do not accompany the SLA specification to guide the SLA matchmaking & negotiation activities.

To realise this combination, two main approaches were followed: (a) for some information aspects not covered by SLAC, an extension was created to accommodate them; (b) missing gaps are covered by cross-referencing OWL-Q specifications which cover the description of metrics, units, metric & condition contexts, and pricing models.

We mainly extended SLAC syntax, see Table 2, to support specifying information related to matchmaking and negotiation activities. This modification led to differentiating between the syntax of a template for negotiation and of an agreement. This extension provides flexibility to negotiation and enables specifying preference relations and negotiable terms, while facilitates matchmaking via negotiable metric intervals.

The syntax is formally defined in the Extended Backus Naur Form (EBNF), in which italic denotes non-terminal symbols and teletype terminal ones. The symbol ::= is used when a new rule is added to the SLAC syntax, while $+=$ is used to extend an existing syntactic core language rule, i.e., the extensions are added into the existing language definition. The /= specifies a special case of core language rule extension which can be used only in template specification.

Templates are extended SLA specifications also including general aspects of *Negotiation* between the involved parties, described in a certain section. This section defines the negotiation's *Strategy* and *Protocol*. The supported strategies are: *Vertical* (user budget is used to enhance the quality for the term with highest priority), and *Horizontal* (user budget is exploited to in-

Table 2: Modification on the syntax of SLAC to make it compatible with OWL-Q.

SLATemplate	::=	*SLA Negotiation*
Negotiation	::=	*Strategy Protocol*
Strategy	::=	`Vertical` | `Horizontal`
Protocol	::=	`Auction` | `SingleTextMediated`
	|	`Bilateral Negotiation`
TermTemplate	/=	*Term Weight*∗
Weight	::=	*Literal*
SLA	+=	… *ContractDates*
ContractDates	+=	(*EffectiveDate ExpirationDate*)$^?$
Term	+=	(`monitoring frequency` *Expr* *Unit* `window` *Expr Unit* `by` *Parties*)$^?$
Metric	+=	… | *OWLQ_URI*
Unit	+=	… | *OWLQ_URI*

crease overall quality proportionally to the user preferences given). The protocol can be: *Auction*, where consumers bid for resources; *SingleTextMediated* where a single document is used to mediate between conflicting participant preferences; and *Bilateral Negotiation*, where the SLA is defined by a bilateral negotiation.

The *Terms* definition in templates is also extended. The *TermTemplate* enables specifying *Terms* defined in the core language with a *Weight* representing their relative importance for the respective negotiation party. This weight is optional and used to build a preference model, exploited also for relaxed service matchmaking. It is defined in the scale of 10, i.e., the sum of all term weights must be 10. For non-weighted terms, the weight assumes the value necessary to reach 10. More formally, Equation 1 defines the weight of all non-weighted terms (W_{nw}), where T_w and $N_{t_{nw}}$ are the set of weighted and non-weighted terms, respectively.

$$W_{nw} = \frac{10 - \sum_{t_w \in T_w} t_w}{N_{t_{nw}}} \qquad (1)$$

The general language modifications (denoted with $+=$) cover missing SLAC information and make the languages composable. In particular, the agreement validity period (specified via the effective and expiration dates) can now be defined. Moreover, the monitoring frequency, window and agent that monitors a respective term can be specified. SLAC now cross-references *Metrics* and *Units* to their respective specification in OWL-Q via an *OWLQ_URI* to resolve the fixed term list drawback. This powerful mechanism enables to provide a formalism for these terms and exploit interesting OWL-Q features in SLAC, such as semantic verification and metric equivalence derivation.

3.1 Lifecycle Activity Coverage

Apart from the improved modelling features of SSLAC, we now describe its main benefits over the whole service lifecycle. These benefits come from employing a specific method which explicates where in the lifecycle each language is used individually and where in the combined form. Figure 1 reflects this method by depicting the lifecycle and the way it is covered by the combined framework of the language agglomeration.

The combined framework comprises 2 sub-frameworks. Initially, a SSLAC template is specified via the SLAC editor, then parsed and converted to OWL-Q to cover the service description activity. The latter OWL-Q specification is then exploited by the first sub-framework, the OWL-Q one, to support the service advertisement, matchmaking and negotiation activities. As such, OWL-Q is exploited until that lifecycle point, a well expected fact from the analysis in (Kritikos et al., 2013) regarding the service lifecycle coverage of service quality specification languages.

The negotiation result, i.e., the agreement, is then converted to its SSLAC form to preserve the semantics of the OWL-Q specification and incorporate the cross-references needed. From that point and on, the SLA enforcement sub-framework built around SLAC can be used as the produced SSLAC specification includes all the information needed to support the remaining activities. This framework is currently extended to support the monitoring and evaluation of dynamically defined metrics. It is also integrated with the OWL-Q one to support the semantic validation and processing of the SLA parts that cross-reference OWL-Q.

The bidirectional SSLAC-to-OWL-Q transformer is the main connection point between the 2 sub-frameworks exploited. It guarantees the consistency of models produced from one language to the other.

By combining both sub-frameworks and languages, the end result is a complete and powerful SLA management framework which apart from language completeness, supports the whole service lifecycle with

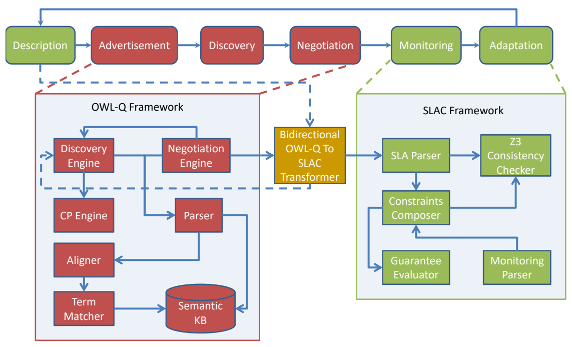

Figure 1: Architecture of integrated SLA frameworks.

specialised capabilities spanning: (a) all model validation types where semantic validation concerns validating elements cross-referenced in OWL-Q, while constraint-based validation caters for checking complex metric conditions; (b) semantic matchmaking and alignment over non-functional terms is supported by the OWL-Q framework (see Section 2.1); (c) SLA enforcement able to handle dynamically specified parts and the transitioning between service levels. Please note that support for negotiation is under-way.

4 USE CASE MODELLING WITH SSLAC

A hospital provides diagnostics based on Magnetic Resonance Imaging (MRI). Instead of having a datacenter, which could lead to high costs and loss of focus on core business, the hospital decides to outsource it to a cloud provider, which uses machine learning for MRI analysis and verifications of the connection between the brain regions and their functions.

This relatively common use case is not fully supported by the existing SLA languages. For example, the flexibility required to change the valid SLA terms is only partially supported via renegotiation, which does not guarantee the possibility to change the terms when required. Moreover, the capability to define new metrics and their context (e.g., measurement frequency) as well as the involved parties in each term is also only partially supported. In this section, we demonstrate the SSLAC advantages over the above issues via describing its application in this use case.

4.1 Modelling Consumer Requirements

The hospital poses requirements on three metrics: *response time*, i.e., the time needed to produce the MRI diagnostics, the *availability* of the service and *maximum number of requests* per hour. The provider must also offer the flexibility to change the SLA terms based on the consumer needs in a pre-defined manner to cater for different classes of consumers. The hospital managers define 3 main situations: *normal*, where at least 6 MRIs/requests can be sent for diagnosis/hour with response time from 6 to 10 minutes; *advanced*, where from 9 to 14 MRIs can be sent per hour with guaranteed response time of 4 to 7 minutes; and *emergency* with significantly reduced response time (at most 2 minutes) and at least 12 MRIs per hour. Availability should be greater or equal to 98% in all cases.

The service charging should be hourly based; upon consumer request, the service can change state (mapping to the above 3 situations) for the next 1 hour. For example, the service is in *normal* state and a doctor has an urgent case. The doctor will then request to change the service level to *emergency* such that all MRIs sent in the next hour will be diagnosed within 2 minutes. Based on the expected pricing scheme of hour-based charging, the hospital can only pay at most 1.2 euros per hour in the *normal* class; at most 2.4 euros in the *advanced*; and at most 4.2 euros in the *emergency*.

As the managers desire to negotiate the service terms with potential conforming service providers, they define requirements and priorities over all 3 terms, which are negotiable based on an horizontal negotiation strategy. The priorities are: 5 for availability and price; 2.5 for response time and request rate.

In SSLAC, each service class is modelled as a group, whose terms are valid only in that context. The service level changes are specified in the *Dynamic* section of the SLA and can be applied under consumer request. The priorities and negotiable terms are specified via the extension proposed in this paper, which is compatible with OWL-Q. This compatibility enables using several matchmaking algorithms proposed for this language (Kritikos and Plexousakis, 2014).

The hospital sends a template to a broker that ranks providers compatible with the request. Before matchmaking, the SLA consistency from different perspectives (syntactic, semantic and constraints), now available in SSLAC, is verified. The following examples of consistency error types can be detected: *constraint*: a definition of the same term in a *Group* and in the *Term* section would lead to a constraint error (2 constraints refer to the same term at the same level); *semantic*: metrics are subclassed via reasoning based on axioms. As such, it could be discovered that a metric is both raw and composite as it is associated to both a sensor and a metric formula. As these two subclasses are disjoint, a semantic error will be raised.

4.2 Offer, Negotiation and Final Agreement

3 functionally-equivalent services match the request (see Table 3). They include the same service levels and charging scheme and enable moving from one level to another when needed. From these 3 providers, only the first 2 non-functionally match the request, since the 3rd violates the 2 minutes response time constraint for the *critical* level. While almost all providers and hospital managers adopt the same metrics set from an OWL-Q ontology, the second provider uses her own metrics for availability and response time which are equivalent to the others. The latter is derived after aligning all non-functional specifications together.

Concerning the alignment, there is a perfect match

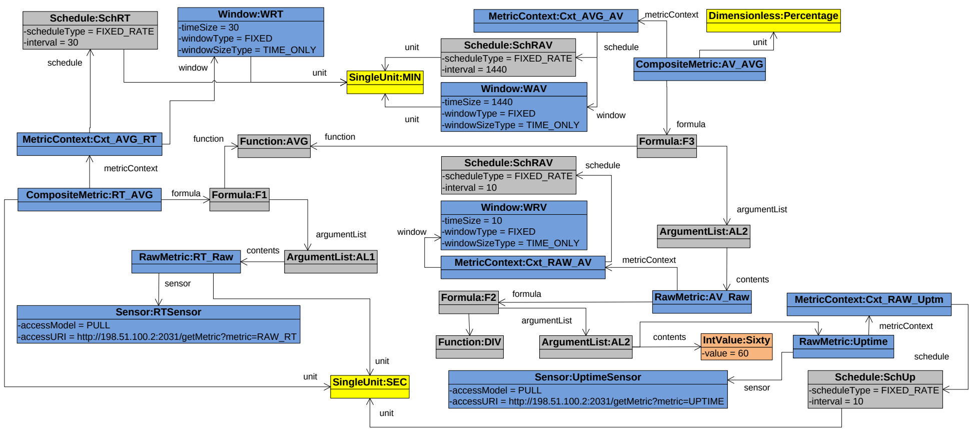

Figure 2: OWL-Q specification for the use case metrics.

Table 3: Service offerings of three cloud providers.

Provider	Level	RT	AV	RR	Price
M-Images	*Nor.*	[6,8]	[99,99.4]	[7,9]	[1.0,1.2]
M-Images	*Adv.*	[4-6]	[99,99.6]	[10,12]	[2.4,2.8]
M-Images	*Em.*	[1-3]	[99.5,99.9]	[13,15]	[3.9,4.2]
H-Analysis	*Nor.*	[7,9]	[98.5,99]	[6,8]	[0.8,1.0]
H-Analysis	*Adv.*	[4-7]	[99,99.4]	[9,11]	[2.0,2.4]
H-Analysis	*Em.*	[2-3]	[99.4,99.7]	[12,14]	[3.5,3.9]
MedImage	*Nor.*	[8,9]	[98,98.5]	[6,7]	[0.5,0.8]
MedImage	*Adv.*	[6-7]	[98,98.5]	[9,10]	[2.0,2.3]
MedImage	*Em.*	[3-5]	[98,98.5]	[12,14]	[3.1,3.5]

between the response time metrics by considering that for the 2nd provider, response time is computed via a formula which subtracts the times that the service response is received and request sent, while for the other providers, it is computed from a sensor. Based on the metric matching cases in the alignment algorithm (Kritikos and Plexousakis, 2014), when two metrics measure the same property and differ in one level, they can be considered equivalent (unless in our case a statistical function was used in the composite metric).

The availability metrics perfectly match on the higher level as they use the same statistical function and measure the same property. However, they are computed from different component metrics. The raw availability metric for the 2nd provider is computed by formula: $\frac{uptime}{downtime+uptime}$ while the same metric for the other providers is computed by: $1-\frac{downtime}{uptime+downtime}$. By taking the difference between these formulas and performing symbolic mathematical simplification, this difference is inferred as 0 such that we can conclude the raw availability metrics equivalence and the consequent composite availability metrics equivalence.

While explicating the matchmaking result, the main issue that the hospital managers face is which providers from those matched should be contacted to start negotiation, where 3 alternative cases hold in our scenario: (a) the 2 results are ranked and negotiation takes place only with the topmost result's provider; (b) both providers participate in the same negotiation; (c) broker filters the matched results based on the supported and preferred negotiation protocols. The hospital managers follow the third case and negotiate with the 2nd provider, as it matches the required negotiation protocol (single-text mediated protocol), by exploiting the respective broker service which exploits the already possessed knowledge about the hospital's negotiation preferences and strategy from its SLA template.

The 2nd provider also adopts a horizontal negotiation strategy. She also requires that all terms are negotiable. The preferences lead to price having the highest weight of 5, followed by availability with 3, and request rate and response time with a weight of 2.

By considering the preferences from both parties, a successful negotiation result is produced.The following facts can be derived: (a) for the *normal* and *emergency* levels, the maximum price provided, as it was lower than the maximum bound requested, was used to select the best possible values for each metric; (b) for the *advanced* level, the middle value for price was selected which also led to obtaining middle values for the rest of the metrics.

In the final agreement, it is specified that quality metrics are assessed and measured by an auditor. In particular, the response time should be reported by the customer and assessed by the auditor, while availability measurement and assessment bares only the auditor.

For an availability violation, the provider must offer 10% discount, while a 5% discount must be given for each response time violation. The discounts sum up to 60% of the spending. Availability is averaged over 24 hours, where raw availability is calculated every 10 minutes based on uptime computed every 10 seconds. Response time is averaged over 30 minutes

Table 4: Final agreement of the use case in SSLAC (excerpt).

```
SLA, term groups:
  Normal:
   Provider → consumer:RT 7 minutes monitoring
  frequency 10 min window 24 hours by Auditor
   Provider → consumer:sslac/Throughput 8 #
   Provider → consumer:Availability 99 %
   Consumer → provider:cost 1 hour
  Advanced:  ...
 terms:
  [1,1] of Normal
guarantees:
   on violation of availability:
     if total_discount < 60
      discount +10 % of total_cost win month
dynamism:
     on consumer request:
       if week_violations > 3
          terminate contract  ...
```

from the raw response time values reported by the customer. If availability is violated more than 3 days or response time more than 12 times in a week, the consumer may end the contract without any penalty, by paying only the amount due to the number of requests issued within the current period till that time point.

Table 4 shows an excerpt of the final agreement. The service is defined in the term groups, including their monitoring frequency and window, and instantiated in the terms section (initially in *Normal* class). The guarantees section describes the violation and penalty for the defined terms, in form of discount over total cost within a month. Finally, the dynamism section specifies the aforementioned settlement cases, where the consumer has the right to terminate the agreement unilaterally, and the possible changes in the quality of service (omitted for the sake of brevity).

5 CONCLUSIONS

This paper has proposed SSLAC to fill the gaps in the current SLA languages in the cloud. SSLAC is a combination of OWL-Q, a prominent service nonfunctional specification language, and SLAC, a prominent domain-specific SLA language. The complementarity of these languages and the language integration mechanism have led to a combination that covers most SLA lifecycle aspects and enables the SLA syntactic, semantic and constraint verification and validation.

To realise this combination, we first extended SLAC to become compatible with OWL-Q. Then, we described the integration of the frameworks developed for these languages, including the bi-direction SLA transformation, to cover the whole SLA lifecycle. The benefits of SSLAC and its framework were demonstrated by applying it over a real use case.

Concerning future work, we plan to further enhance SSLAC so as to score optimally across all SLA lifecycle activities. Moreover, we will thorough evaluate SSLAC against a series of use cases. Finally, we will explore additional language and integrated framework extensions to further support service negotiation.

ACKNOWLEDGEMENTS

This research has received funding from the European Community's Framework Programme for Research and Innovation HORIZON 2020 (ICT-07-2014) under grant agreement number 644690 (CloudSocket).

REFERENCES

Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., and Xu, M. (2007). Web Services Agreement Specification (WS-Agreement). Technical report, Open Grid Forum.

Comuzzi, M., Kritikos, K., and Plebani, P. (2009). A semantic based framework for supporting negotiation in Service Oriented Architectures. In *CEC*. IEEE Computer Society.

Correia, A., e Abreu, F. B., and Amaral, V. (2011). SLALOM: a language for SLA specification and monitoring. *CoRR*, abs/1109.6740.

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., Pernici, B., Plebani, P., Cappiello, C., Comuzzi, M., Benbernou, S., Brandic, I., Kertész, A., Parkin, M., and Carro, M. (2013). A survey on service quality description. *ACM Comput. Surv.*, 46(1):1.

Kritikos, K. and Plexousakis, D. (2006). Semantic QoS Metric Matching. In *ECOWS*, pages 265–274. IEEE Computer Society.

Kritikos, K. and Plexousakis, D. (2014). Novel optimal and scalable nonfunctional service matchmaking techniques. *IEEE Trans. Services Computing*, 7(4):614–627.

Paschke, A. (2005). RBSLA: A declarative rule-based Service Level Agreement Language based on RuleML. In *CIMCA-IAWTIC*, pages 308–314. IEEE Computer Society.

Pedrinaci, C., Cardoso, J., and Leidig, T. (2014). Linked USDL: A vocabulary for web-scale service trading. In *ESWC*, pages 68–82.

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

Uriarte, R. B., Tiezzi, F., and De Nicola, R. (2014). SLAC: A Formal Service-Level-Agreement Language for Cloud Computing. In *UCC*, pages 419–426. IEEE.

Uriarte, R. B., Tiezzi, F., and De Nicola, R. (2016). *Dynamic SLAs for Clouds*, pages 34–49. Springer International Publishing, Cham.