Extending WSLA for Service and Contract Composition

Antonella Longo, Marco Zappatore and Mario A. Bochicchio
Dept. of Innovation Engineering, University of Salento, via Monteroni, Lecce, 73100, Italy

Keywords: SLA Management, SLA Measurement, SLA Composition, SLA Design.

Abstract: Cloud Services (CSs) nowadays experience constantly improving successes in IT scenarios. Dynamic allocation of network, storage and computational resources, the hiding of visibility of internal IT components, as well as the pay-per-use paradigm are becoming more and more widespread ways to provide and consume services. The complexity of CSs is often due to service chains into which third-party services are aggregated in order to satisfy user requests. This confirms the need of modeling both contracts and corresponding Service Level Agreements (SLAs) referring to services provided to customers. Similarly, time-related variability issues in CSs require run-time performance monitoring and reporting solutions capable of comparing SLAs and feeding requesters with effective resource reservation and allocation policies. A detailed analysis in contracts and SLAs management has revealed a lack of expressivity in SLA specification and a consequent inadequacy in tools for describing and managing SLAs and contract composition. Therefore, we propose an extension of WSLA, a widely known SLA description language. We aim at modeling contracts and SLAs with additional details to support contract owners during service composition and its monitoring. The proposed approach has been adopted to develop and validate a tree-graph-based tool, to simplify SLA and contract composition.

1 INTRODUCTION

In modern IT environments, a wide variety of tasks requires services to be created and provided to end-users. A non-trivial problem concerns the definition of service terms that have to be agreed between parties and rendered as viable contracts. Arguably, these contracts have to be at the same time strategically effective and operationally feasible. In this scenario, Service Level Agreements (SLAs) (Telemanagement (TM) Forum, 2008) play a key-role in describing service relevant features of a service. Indeed, SLAs specify the agreements between service producers and consumers in terms of Quality of Service (QoS), service level guarantees, Service Level Objectives (SLOs), compliance to users' requests, service time-sustainability. They also define measurements and criteria to be used when services fail or deviate beyond specific tolerability thresholds from the agreed-upon contract terms.

SLAs exhibit different types of complexity, ranging from single to multiple, co-operating providers and from single to bulks of diverse customers; each SLA may concern single or combined services, and the SLA specification may be very qualitative and difficult to be transformed in machine-readable elements. SLA specification complexity may be inherent with each of the SLA elements (i.e. contract duration, working window, multiple targets for a single SLA, etc.).

SLAs usually refer to scenarios where a provider and a customer agree upon the delivery of single services with SLAs. Current IT scenarios, instead, involve cloud paradigm or Business Process Outsourcing (BPO), thus requiring paying much more attention to service composition issues. However, available methodologies and frameworks lack in formalizing how IT managers can effectively compose services, contracts they are defined into and the corresponding SLAs.

The issues arising from both service composition and corresponding SLAs has a significant relevance in Cloud Computing (CC) and they affect service providers and customers as well. On the one hand, since CC must provide on-demand access to customizable computational resources, IT managers and service providers need to assess precisely Service Levels (SLs) when they design the overall contract for the delivered service. On the other hand,

Longo A., Zappatore M. and Bochicchio M..
Extending WSLA for Service and Contract Composition.
DOI: 10.5220/0005401503070315
In *Proceedings of the 17th International Conference on Enterprise Information Systems* (ICEIS-2015), pages 307-315
ISBN: 978-989-758-098-7

service consumers require fast service provisioning, reliable resource exploitation, reduced management efforts and frequent interaction with providers, thus generating a great variety of SLOs that makes more difficult to negotiate and finalize service selection. Moreover, consumers require visibility on SLAs monitoring data and demand auditing tools.

Since CC services impose aggregating many sub-services into more complex, higher-level services, IT managers must supervise the way such services are functionally composed and also to combine SLs into the delivered final output in order to effectively satisfy business needs. The selected composing contracts and specifications should allow users to obtain easily the terms and conditions of the overall contract without being domain-specific.

All these aspects may severely hinder specific features in CC, such as providers' accountability when services do not match with their defined levels or when delivered services exhibit failures beyond the allowed guarantees. Therefore, IT cloud managers need tools to assess how much they adhere to the promised SLOs for the service chains they manage, especially if resources are dynamically reconfigured at run-time and customers must be guaranteed through customized SLAs.

SLA technologies have been profitably applied so far in Service-Oriented Architectures (SOAs) (Baker and Dobson, 2005) thanks to their suitability to model domain information better than ad-hoc representations. Indeed, SLAs are: a) flexible and potentially language-independent; b) capable of defining both measurable quantities (e.g., accuracy, availability, latency, cost) and non-measurable aspects (e.g., reputation). Therefore, many formal specifications have been developed to model SLAs and their subject of application: WSLA (Keller and Ludwig, 2002), SLANG (Lamanna et al., 2003), BPEL (Xiang et al., 2004) are amongst the most widely known. We focus on WSLA because it is based on XML and its template is general enough to cover different domain scenarios.

From such premises, this paper proposes a descriptive template for contract and SLA composition in service chains within the CC environments. We started from WSLA and we have extended it in order to overcome WSLA expressivity limitations when dealing with SLA specification and composition. The new model will allow Cloud service providers to automatically manage contracts. As a side positive effect of having such a more suitable template for describing service composition, SLAs and delivered services allow identifying accountability issues more efficiently in CC scenarios. The proposed template introduces significant new aspects such as composition rules and topologies and a more detailed SLA specification description.

We also propose a tool (Contract-Aware Service Composer, CASCO) easing contract composition design and visualization as well as the collaboration amongst different IT professionals. In order to validate the proposed model, five different Key Performance Indicators (KPIs) from CC (i.e. availability, response time and Mean Time To Response (MTTR), Mean Time Between Failure (MTBF), Mean Time To Failure (MTTF)) have been implemented in CASCO.

The paper is organized as follows. Section 2 analyses existing works on contract representation models and SLA data management. Section 3 presents our extension of WSLA. The design and implementation of the supporting tool are described in Section 4. Section 5 applies the model to a real use case scenario. Section 6 hosts conclusions.

2 RELATED WORKS

A SLA referring to a specific service should consider descriptive aspects (agreeing parties, QoS, obligations); matching between delivered services and QoS; collected metrics (when and by whom); penalties for service failures and unmatched SLs; actions for undelivered services; providers' responsibilities; technology re-alignments affecting SLAs and delivered services. SLAs are enforced by service management policies, thus ad-hoc IT frameworks are needed, such as the Information Technology Infrastructure Library (ITIL). These solutions ensure that deployed applications and services meet the required SLs at deployment, operation, support and maintenance stages (Allen, 2006). SLAs need to be machine-readable, since they are essential in SOAs, in order to support service discovery, selection and processing w.r.t. QoS. This originates numerous language specifications to define SLAs. Some of them combine high-level modeling languages and knowledge representation techniques. The TAPAS project (Lodi et al., 2007) proposed the SLAng language, where SLAs are defined by mixing Natural Language (NL), Object Constraint Language (OCL) and Meta-Object Facility (MOF). In (Paschke , 2008) a Rule-Based SLA (RBSLA) approach is described: SLA management policies are provided as RuleML constructs (Boley et al., 2001) and then processed by a specific tool. PANDA (Anon., 2007),

is a Multi-Agent System focused on contract negotiation, monitoring and Virtual Organization evaluation. Although it provides some appealing features (e.g., SLA template storage, partner profile catalogue), it is primarily limited to Enterprise Resource Planning Systems (ERP). Despite the high-level of expressivity, those approaches do not offer the best suitability to machine processing. Another drawback is their difficult integration with documents that traditionally describe Web Services, as WSDL (Christensen et al., 2001).

XML-based implementations, more amenable to be machine-readable have emerged, such as the WSLA framework (Keller and Ludwig, 2002), firstly introduced into the not anymore available Web Services ToolKit (IBM Corp., 2006), for defining and monitoring SLAs for Web Services. It offers an extensible language and a runtime architecture comprising several SLA monitoring services, which may be outsourced to third parties to ensure maximum objectivity. WSLA language main elements are: subjects, services and obligations, SLOs, action guarantees. Although WSLA allows defining, negotiating, deploying, monitoring and enforcing SLAs, it has some limitations: 1) contracts signed by only two parties; 2) no ways to describe over-/under-achievement of contract obligations; 3) small support to corrective management actions.

Cremona architecture (Ludwig et al., 2004), by IBM, is a SLA middleware supporting contracts monitoring, with a three-level architecture but it does not offer decision-making capabilities based on agreement terms and does not support workflow handling for service interactions.

The Open Grid Forum (OGF) proposed the Web Service Agreement (WS-Agreement) (Andrieux et al., 2007) as a protocol to establish agreements between providers and consumers, built on top of Cremona. It is detached by WSLA in order to meet the need for a standard by the OASIS organization (OASIS, 2014). It allows describing agreement contexts and a set of attributes for a specific service (i.e., name; context; Service Description Terms, SDT; guarantee terms; constraints). It allows customers to ask for service requirements via XML files and providers to evaluate available resources. However, it does not allow to model third parties' contributions and no metrics are available for monitoring service choreographies.

A widely agreed, standardized model would enable to apply templates to every type of contracts and SLAs, and to categorize contract terms for different service domains. Therefore, based on WSLA and WS-Agreement, in (Stamou et al., 2013.10) a SLA digraph model (where nodes and edges represent SLA content) for automating SLA formulation and data handling was proposed, tailored to SLA data management over distributed web resources. The model was tested in (Stamou et al., 2013.6) in a SOA client-server scenario, where the server behaves as a cloud service marketplace using SLA templates for service offers. Customers submit their requirements over HTTP and the marketplace returns SLA templates matching the requests. The representation of SLA is broader, it includes e-business outsourcing contracts (Ward et al., 2002) and inter-organizational scenarios, where service and contract compositions occur, such as Cloud Computing. Modelling and monitoring aspects related to service chains and networks in many business ecosystems are usually based on graph models. BPMN (Dijkman et al., 2008) models represent another viable alternative, due to their suitability to be translated in BPEL (Xiang et al., 2004) but are effective for domain-skilled professionals, thus limiting their usage for non-technical stakeholders. At the moment there are no significant attempts to model service chains in Cloud Computing taking into account stakeholders perspectives. Moreover, researchers have focused their investigations on the Quality of Service (QoS) composition (Ben Mabrouk et al., 2009), (Cardoso et al., 2004) on modeling dependability among SLAs (Bodenstaff et al., 2008), or on aggregation patterns of SLAs (Ul Haq and Schikuta, 2010). These approaches consider metric composition or abstract aggregation patterns, but none so far has investigated on how to compose specific features of contract and SLA (e.g., working windows, start/end date, guaranteed calendar), together with QoS values. Indeed, SLA standardization policies are still in their infancy, thus providing IT professionals with just static or semi-structured information.

This relevant issue is related with the entire SLA data management process (Stamou et al., 2013.10) but also involves service elements (e.g., metrics, descriptions, provisioning guarantees). SLA content belongs to several domains, and various vocabularies of provisioning terms represent a primary cause of semantic and structural SLA heterogeneity, which complicates comparison and hinders SLA handling in distributed service markets (Stamou et al., 2013.10). Therefore, we believe that the digraph model employed in (Stamou et al., 2013.6), as well as the WSLA framework, represent a suitable foundational base for our approach.

3 WSLA MODEL EXTENSION

IT and CC service contracts often results from the composition of underpinning service contracts (and, in turn, of their specifications). Starting from them, the final provider's goal is to automatically obtain overall contract terms, after defining composition rules. A service-based approach enables to derive the final contract from composition rules, and contract specifications by applying these rules on the parameters of each low-level contract and service. Consequently, services from different providers can be used and composed. Since service composition is *"the ability to take existing services"* (or building blocks) *"and combine them to form new services"* (Piccinelli, 1999), if obligations must be guaranteed in service composition, service contracts (and SLAs) must be defined accordingly.

From such premises, we can define contract composition as the derivation of a contract, obtained by integrating other contracts, signed with different providers (Bochicchio et al., 2013). Terms and conditions of the composite contracts come from those of each component, after applying composition rules. Similarly, the definition of contract composition implies that a provider of a composite contract is able to extract and feed clients with terms related to the component contracts according a specific applied rule. Moreover, the final contract provider need evidence about service accountability of underpinning providers if necessary. It is evident that contract composition is the result of a service-oriented approach applied on Contract Management, since it is directly based upon service composition.

The contract model can be considered as built by three levels of abstraction, depending on whether we are considering the signed contracts, the provided/requested services or the resources needed to provide services. Inter-level links represent dependencies. Intra-level links represent resource or service compositions. In order to test the feasibility of our approach in SLA composition design, we prefer to strictly focus on service and SLA aggregation. In this way, we just consider contracts in terms of basic composing elements, thus forwarding the service-to-resource mapping to further research developments.

These aspects cannot be fully described by current SLA models and frameworks, therefore we define a WSLA extension that considers service level management aspects, according to (Stamou et al., 2013.6). Our template presents contracts as composed by three sections.

The *Agreement Info* handles general contract information such as name, description, start date and expiry date. The *Agreement Parties* models information related to parties involved in the contract. It is composed by at least two parties (i.e., a service provider, or contractor, and a service consumer, or customer), but additional participants can be added as well. The *Agreement Services* manages services information and it can be referred to one or more services. Each service exposes specific properties: a description, a guarantee calendar and its SLA, which is the core element of this component as well as of the entire template. Each SLA exhibits specific features: reference period, description, report date, specification.

SLAs specifications refer to multiple parameters: cost, start/end date, sample period, SLO (i.e., threshold value for SL), working window (i.e., time range during which SL must be generated), penalty, Service Level Indicators (SLIs) (i.e., metrics and algorithms to calculate SLA for service monitoring).

Directed tree graphs (digraph), (Bang-Jensen and Gutin, 2008) can be used to model this contract template. Figure 1 proposes the overall digraph, (nodes: template components; oriented edges: available properties; edge labels: node cardinalities).

In order to extend WSLA for describing SLA and contract composition, composition topologies and rules have been introduced.

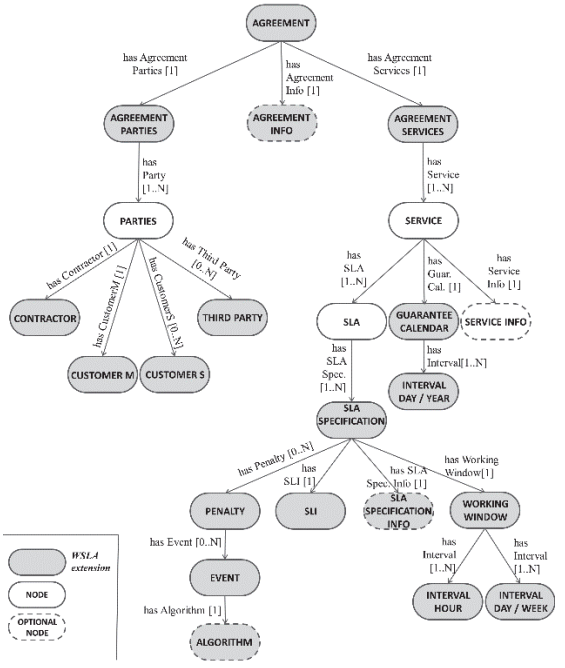

Figure 1: Contract template as a digraph.

Topologies allow chaining services to satisfy end users' requests. We adopted a four-pattern modeling

strategy (Ul Haq and Schikuta, 2010). *Series:* services execute jobs in sequence, so a service starts when the previous ends. *Parallel:* services execute jobs simultaneously. *Loop:* a service needs several cycles to complete its job. *Condition:* a parallel topology, with an initial choice for a specific path.

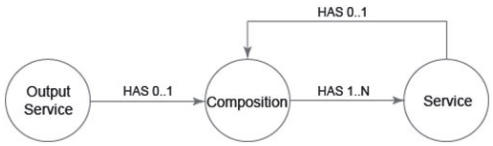

Figure 2: Composition topology model.

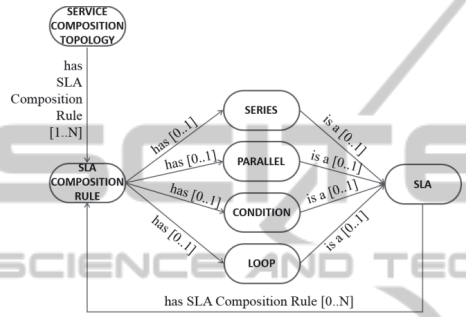

Figure 3: Composition rule model.

Topologies can be further composed (in a way similar to circuit elements are cascaded in circuit theory): by doing so, the final service can be obtained by recursively choosing the available topology patterns. Each composition requires one (loop pattern) or more services (other patterns) to be built and each final service may (or may not) derive from a topological composition. Figure 2 depicts the relation between services and composition.

After composition, SLAs must be modelled starting from input services SLAs. This is the main reason for the definition of a third model that specifies the formula for a given SLA and topology. This formulation has been called *Composition Rule*. Figure 3 illustrates this model as well as its relationship with SLAs and composition topologies.

4 TOOL DESCRIPTION

The main goal of CASCO is to test the model proposed in Section 3 and to help the users in composing SLAs for services contracts. Indeed, it can be useful for customer and service providers both in design and execution phases. During the design phase, composition can be simulated to verify the compliance with requested SLs, as well as to obtain SLA of composed services starting from elemental ones. During the execution phase the system offers an engine that is able to calculate the result of composite SLAs at run time.

The tool has been designed by starting from stakeholders and goals elicitation steps. We have identified multiple stakeholders for a generic service providing/consuming company, encompassing managers (i.e., general, technical, business, department, human resource, service, contract) and administration personnel, each having different tasks. For instance, a service manager can use the tool in order to 1) insert contracts signed with other companies to buy primary services; 2) access and visualize the service catalogue; 3) insert new primary services; 4) create a new service by composing services from the catalogue; 5) create new contracts to be signed with other firms or users.

From an architectural point of view (Figure 4), the tool is based on the widely-adopted Model-View-Controller (MVC) pattern for web applications, whose purpose is to achieve a clean separation between data management, user interface and business logic. According to this pattern, CASCO is based on a three-layer architecture, as represented in Figure 4.

In order to separate contract visualization issues from user-agent interaction during composition, the tool splits the front end (Presentation module) and the back end (Business Logic and Data Management module) by using three XML files, which provide information about agreements, topologies and rules.

The *Presentation module* is responsible for user operations management and input parsing, in order to generate the XML files that will be accessed by both Presentation and Manager modules

The *Manager module* is further divided in four subsections. The *Parser* allows extracting and manipulating XML data from agreement, topology and rule files in order to store them in the graph database (DB). The *Agreement Manager* handles contracts and operations that must be performed on them. The *SLA Compositor* is responsible for the creation of composed services, for the definition of corresponding metrics and parameters as well as for the storage of output services into the database. The *SLA Control* allows accessing services information and parameters for resources/services monitoring

The digraph model (Section 3) has been used as a basis for the adoption of a NoSQL graph DBMS (Neo4j v.1.9.6/v.2.0.0 (Neo Technology, Inc., 2014)), in order to store SLA and contracts information as well as composition topologies and rules.

The tool has been developed in PHP, for the front end part, respecting the HTML5 format, in order to

be natively compatible with mobile device, and Java for the back end. Moreover we used Spring-data-neo4j (v2.3.4) (Pivotal Software, Inc., 2014), a framework for mapping classes with the elements stored in Neo4j, and Neo4jPHP (v.0.1.0) (Adell, 2012), a PHP library for accessing Neo4j database.

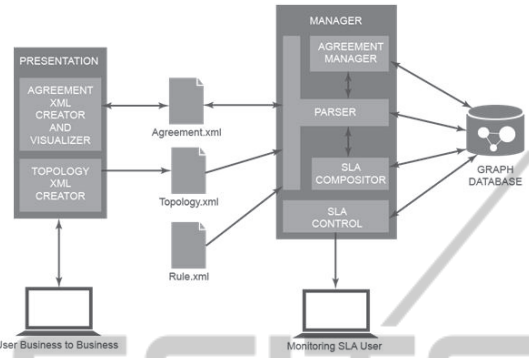

Figure 4: MVC-based architecture for CASCO tool.

5 APPLICATION SCENARIO

We refer to a set of five significant SLA metrics in CC, in order to check whether our template can capture all the relevant information for SLA composition. The metric computation phase demonstrates that all the elements needed to manage contracts have been correctly modeled and included within the proposed template. SLA values are calculated with respect to metrics capable of measuring whether a service is compliant with the contract terms and whether the service can achieve the desired business objectives. If we consider a service composition scenario, metrics must be the same and their measurement units must be commensurable and compatible (e.g., all time-related or all percentages, etc.).

The selected metrics are: Availability, Response Time, Mean Time To Failure (MTTF), Mean Time Between Failures (MTBF), Mean Time To Repair (MTTR). In order to better understand MTTF, MTTR and MTBF, some quantities must be defined.

Service Downtime (T_{SD}): it is the time span between the *i-th* occurrence of a service failure, $t_{d,i}$, and the *i-th* occurrence of its reactivation, $t_{u,i}$.

Service Uptime (T_{SU}): it is the time range between the *i-th* occurrence of the service start, $t_{u,i}$, and the *i+1-th* occurrence of its failure $t_{d,i+1}$).

Therefore, the time lapse between two downtime periods (Time Between Failures, TBF) is the sum of the time needed to have the service up again (Time To Repair, TTR) and the time elapsed before another failure occurs (Time To Failure, TTF).

We have defined the metrics as specified below.

1. Availability (Av): it describes the percentage of time a service S_A is available in a certain time lapse corresponding to the temporal window of analysis. Being T_{AW} the duration of the temporal window of analysis and T_{SD} the service downtime period, the availability *Av* is expressed as in (1):

$$Av(S_A) = (T_{TW} - T_{SD}) * 100/T_{TW} \quad (1)$$

2. Response Time (RT): it quantifies how long it takes to a service S_A to answer client's request. If we consider *N* client requests, each having its own response time $rt^{S_A}(t)_i$, the overall *RT* is the average of the distinct response times, as defined in (2):

$$RT(S_A) = \left(\sum_{i=i}^{N} rt^{S_A}(t)_i\right)/N \quad (2)$$

3. MTTF: it measures the mean time between two consecutive failures for the same service S_A. Let us *N* denotes the set of monitored events (i.e., service failures and restorations). Let us $t^{S_A}{}_{u,i}$ indicates the time from which the service is up for the *i-th* time and $t^{S_A}{}_{d,i+1}$ the time at which the same service fails again. Then, *MTTF* is expressed by (3):

$$MTTF(S_A) = \left(\sum_{i=1}^{N-1}\left(t^{S_A}{}_{d,i+1} - t^{S_A}{}_{u,i}\right)\right)/N \quad (3)$$

4. MTTR: it measures the mean time needed to repair a specific service S_A for *N* times. Let be $t^{S_A}{}_{d,i}$ the time from which the service is down and $t^{S_A}{}_{u,i}$ the time from which the same service is up again, *MTTR* is given by (4):

$$MTTR(S_A) = \left(\sum_{i=1}^{N-1}\left(t^{S_A}{}_{u,i} - t^{S_A}{}_{d,i}\right)\right)/N \quad (4)$$

5. MTBF: it measures the mean time occurred between two consecutive failures of the same service. It can be simply considered as the sum between (4) and (3), as depicted in (5):

$$MTBF(S_A) = MTTR(S_A) + MTTF(S_A) \quad (5)$$

New composed services (Cardoso et al., 2004) exhibit different metrics, depending on the composition rules and topologies selected during the design phase. Let us explain the five selected metrics w.r.t. series and parallel topologies, in the case we have two different services S_A and S_B that have to be composed sequentially and whose respective metrics are known before the composition is performed. As for the series topology, we assume separate services are independent (i.e., stochastically modelled as independent variables (Ben Mabrouk et al., 2009). This means that the *Av* of a composed service is the

product of the availability of each component, *RT* of the composed service is the sum of response time of each component service and *MTTF* is the reverse of the sum of single components' *MTTF*s.

Regarding the value associated with parallel compositions we consider the worst values of all the possible composition executions. For instance, to determine *RT* of parallel activities, we consider the activity with the longest *RT*. Table 1 resumes the rules mentioned above, for *Av*, *RT* and *MTTF*. These examples are independent from the specific domain.

Table 1: Examples of composition rules.

Metric	Rule (Series Topology)	Rule (Parallel Topology)
Av	$Av_S = Av(S_A) \times Av(S_B)$	$Av_{//} = Av(S_A) \times Av(S_B)$
RT	$RT_S = RT(S_A) + RT(S_B)$	$RT_{//} = max\{RT_A, RT_B\}$
MTTF	$MTTF_S = 1 / ((MTTF(S_A)) + (MTTF(S_B)))$	$MTTF_{//} = MTTF(S_A) + MTTF(S_B) - MTTF_S$

In service composition the time window (and the service calendar) in which the composite service is guaranteed depends on the composing services. It defines the time and calendar the composite service level is guaranteed. For example S_A is guaranteed from 8:00 to 20:00 in business days and S_B is guaranteed from 7:00 to 23:00 only in odd days (e.g., front office openings). When S_A and S_B are composed into S_C, the guaranteed windows changes according to composition topology: if $S_C = (S_A$ series $S_B)$, then S_C guaranteed service calendar is the intersection of S_A and S_B calendars. If $S_C = (S_A$ // $S_B)$ and S_A and S_B are homogeneous (the same kind of service), then guaranteed windows is the union of S_A and S_B calendars. If S_A and S_B are heterogeneous, S_C guaranteed calendar is the intersection of S_A and S_B service hours. If the S_C has an empty guaranteed window, it cannot be guaranteed.

Each component service within the SLA owns a specific measurement period: the composite measurement period is constrained by the corresponding component services. In general, it is the least common multiple (l.c.m.) of each component's measurement period, if component services start simultaneously. Otherwise, the two measurements periods need to be aligned. For example, let us suppose that S_A and S_B have a 2-month and a 3-month measurement period respectively. The measurement period of S_C should be the l.c.m. of S_A and S_B that is 6. If we suppose that S_A started a month later than S_B, then they will be out of phase and only after a 2-month transition S_C would be measured with a 6-month period. Similarly, the S_C contractual duration is the intersection of the duration of S_A and S_B.

The tool has been verified and validated. A set of agreements has been simulated, proving that all the SLAs aspects necessary to calculate and composed SLAs are represented in the model. Moreover, system results conform to the rules described before.

Our approach starts from the assumption that the user uses a Graphical User Interface (GUI) to submit services and contract terms and conditions. The GUI helps the user to express the request in terms of services and SLAs requirements, translated into machine-understandable specifications by the tool. This is the reason why specific tests have been executed to evaluate the tool usability for IT managers and contract owners.

We selected 20 Italian users as testers, aging from 30 to 50 years old and belonging to technical and juridical areas. The candidates have been assigned with a series of tasks representative of a concrete deployment scenario. Task#1: to retrieve a service purchased by a company. Task#2: to register an agreement and to compose a new output service. Task#3: to evaluate the application completeness and user-friendliness. The users were required to fill a questionnaire to assess their mood against the proposed system and the features it exposes.

A 10-point psychometric Likert scale (Allen and Seaman, 2007) has been used to measure users' responses in evaluating each task, according to the easiness in retrieving information, navigational clarity and message clarity. In this way, users were allowed to express their opinion within a numerical scale ranging from 1 to 10, where 1 stands for the worst possible evaluation and 10 represents the best one. Table 2 enlists average scores for the features evaluated when performing a specific task and it also shows task mean execution times across all tested users). At the end of the questionnaire, users' comments were collected, thus observing their concerns was only about explanatory labels and the localization of the application (which actually is in English to better match references about service contracts and composition) and do not refer to possible issues in contract and SLA design or missing modelling functions.

Table 2: Overall average scores (AS) from voters' pool and mean execution times (MTE) for each task.

	Evaluated Feature	AS	MTE
Task1	Easiness in retrieving information	6.45 / 10	42.3 [s]
	Clarity of navigation and messages	6.8 / 10	
Task2	Clarity of agreement registration reqs.	8.5 / 10	183 [s]
	Easiness of composite service creation	6 / 10	
Task3	Clarity of content presentation	9.25 / 10	352 [s]
	Clarity of graphical representation	9,35 / 10	

6 CONCLUSIONS

A model for managing SLA information when dealing with contracts and SLA composition in service-based IT environments has been presented in this paper. Starting from the widely known, WSLA framework, we have proposed an extension capable of overcoming two main issues in managing SLAs.

At first, we tackle the lack in standard models representing service contracts and their SLAs in SOA and service network environments. We mainly address contract and SLA composition aspects, by proposing a flexible template for IT service contracts that can be applied to several domains. The original WSLA model has been complemented by considering two new aspects: composition topologies and rules to cope with Cloud Computing scenarios, where IT resources are usually reserved and allocated dynamically according to users' requests. Moreover, cloud services are provided to customers as the composition of multiple, third-party services, thus requiring specific attention to the corresponding contracts that have to be signed and their terms and guarantees.

Secondly, we target the possibility to make SLAs effectively machine-readable, thus allowing easier design and monitoring policies for IT professionals managing service networks. The choice of WSLA, due to its XML-based structure has been done right in that direction. In order to ease SLA machine readability, we modeled the contract template as a digraph. The implementation has been done by revolving to a NoSQL graph-based DBMS.

A specific tool, named CASCO (Contract-Aware Service COmposer), has been designed, developed and implemented in order to provide IT professionals with a solution allowing contract and SLA evaluation and assessment during the design phase. A set of five SLA metrics has been selected to evaluate template feasibility via to our tool. We have ascertained that WSLA extension can handle all the information for describing contracts of composed services and their SLAs.

In the near future, this research will be widened and by considering three additional aspects: 1) template extension to describe SLAs related to BPO contracts; 2) ontological formalization of the template; 3) adoption of rule-based policies to perform real-time monitoring of the composed services as in Complex Event Processing.

REFERENCES

Adell, J., 2012. *Neo4jPHP*. (Online) Available at: https://github.com/jadell/neo4jphp (Accessed Nov. 2014).

Allen, P., 2006. *Service Orientation - Winning Strategies and Best Practices*. Cambridge Univ. Press.

Allen, E. & Seaman, C., 2007. Likert Scales and Data Analyses. *Quality Progress*, pp.64-65.

Andrieux, A. et al., 2007. GFD-R-P.107 *Web Services Agreement Specification*. Memo. (GRAAP) WG.

Anon., 2007. http://www.panda-project-com/ *PANDA: Collaborative Process Automation Support using Service Level Agreements and Intelligent Dynamic Agents in SME Clusters*. IST-Panda Reseach Project.

Baker, S. & Dobson, S., 2005. Comparing Service-Oriented and Distributed Object Architectures. In *Int. Symp. on Distributed Objects and Applications*., 2005.

Bang-Jensen, J. & Gutin, G.Z., 2008. *DIgraphs: Theory, Algorithms and Applications*. 2nd ed., Springer.

Ben Mabrouk, N. et al., 2009. QoS-aware Service Composition in Dynamic Service Oriented Environments. *Middleware 2009 - ACM/IFIP 10th Int. Conf.*, 2009.

Bochicchio, M.A., Longo, A. & Giacovelli, S., 2013. SARA: a Tool for Service Levels-Aware Contracts. In *IFIP/IEEE Int. Symp. on Integrated Network Management*, 2013.

Bodenstaff, L., Reichert, M. & Jaeger, M.C., 2008. Monitoring Dependencies for SLAs: the MoDe4SLA Approach. In *IEEE Service Computing Conference*., 2008.

Boley, H., Tabet, S. & Wagner, G., 2001. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *SWWS-2001*, 2001.

Cardoso, J. et al., 2004. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics*, 1, pp.281-308.

Christensen, E., Curbera, F., Meredith, G. & Weerawarana, S., 2001. http://www.w3.org/TR/wsdl *Web Service Description Language (WSDL) 1.1*. W3C Note.

Cortellessa, V. & Grassi, V., 2007. A Modeling Approach to Analyze the Impact of Error Propagation on Reliability of Component-based Systems., 2007.

Dijkman, R.M., Dumas, M. & Ouyang, C., 2008. Semantics and Analysis of Business Process Models in BPMN. *Inform. and Software Tech.*, 50(12), pp.1281-94.

IBM Corp., 2006. *IBM Web Services Toolkit (WSTK)*. (Online) (2.3 (discontinued)) Available at: http://www.alphaworks.ibm.com/tech/webservicestoolkit.

Keller, A. & Ludwig, H., 2002. RC22456 *The WSLA Framework: Specifying and Monitoring Service Level Agreements*. Technical Paper. IBM Research.

Lamanna, D., Skene, J. & Emmerich, W., 2003. SLAng: a Language for Defining Service Level Agreements. *FTCDS 2003*, 2003.

Lodi, G., Panzieri, F., Rossi, D. & Turrini, E., 2007.

SLA-Driven Clustering of QoS-Aware Application Servers. *IEEE Trans. on Software Eng.*, 33(3), pp.186-97.

Ludwig, H., Dan, A., Kearney, R. & Heights, Y., 2004. *Cremona: an Architecture and Library for Creation and Monitoring of WS Agreements*. Res. Rep. IBM.

Neo Technology, Inc., 2014. *Neo4j*. (Online) (2.1.6) Available at: http://www.neo4j.org/ (Nov. 2014).

OASIS, 2014. *OASIS -Advancing open standards for the information society*. (Online) Available at: https://www.oasis-open.org/ (Accessed November 2014).

Paschke, A., 2008. Knowledge Representation Concepts for Automated SLA Management. *Decision Support Systems*, 46(1), pp.187-205.

Piccinelli, G., 1999. HPL-1999-84 *Service Provision and Composition in Virtual Business Comm.*. Tech. Rep.

Pivotal SW, Inc., 2014. *Spring Data Neo4J*. (Online) http://projects.spring.io/spring-data-neo4j (Nov. 2014).

Saeedi, K., et al., 2010. Extending BPMN for Supporting Customer-Facing Service Quality Requirements. In *IEEE Int. Conf. on Web Services*., 2010.

Stamou, K., Kantere, V. & Morin, J.H., 2013.10. SLA Data Management Criteria. In *IEEE Big Data*., 2013.10.

Stamou, K., Kantere, V. & Morin, J.H., 2013.6. SLA Template Filtering: a Faceted Approach. In *4th Int. Conf. on Cloud Computing, GRIDs and Virtualization*., 2013.6.

Telemanagement (TM) Forum, 2008. *SLA Handbook Solution Suite v2.0*.

Ul Haq, I. & Schikuta, E., 2010. Aggregation Patterns of Service Level Agreements. *FIT'10*, 2010.

Ward, C., et al., 2002. A Generic SLA Semantic Model for the Execution Management of e-Business Outsourcing Contracts. In *3rd Int. Conf. on e-Commerce and Web Technologies*., 2002.

Xiang, F., Tevfik, B. & Jianwen, S., 2004. Analysis of Interacting BPEL Web Services. In *WWW '04*, 2004.

Zheng, Z. & Lyu, M.R., 2010. Collaborative Reliability Prediction of Service-Oriented Systems. In *10th ICSE*. Cape Town, South Africa, 2010. ACM.