A WSLA-Extension Supporting Service and Contract Composition Modelling

Antonella Longo[1], Marco Zappatore[1], Mario A. Bochicchio[1], Lucia Vaira[1] and Alfredo Cuzzocrea[2]

[1]Department of Engineering for Innovation, University of Salento, Lecce, Italy
{antonella.longo, marcosalvatore.zappatore, mario.bochicchio, lucia.vaira}@unisalento.it
[2]University of Trieste and ICAR-CNR, Italy
cuzzocrea@icar.cnr.it

Abstract:
When dealing with Cloud Computing Services (CCSs) provisioning and usage, different degrees of complexity can be reached, depending on whether service composition is needed to satisfy users' requests. This scenario demands effective ways for modelling CCSs and corresponding Service Level Agreements (SLAs) in order to facilitate service composition, comparison and monitoring. However, current SLA specifications and tools are not well tailored to service composition. In this paper, starting from WSLA, a widely-known SLA description language, we present an extension aimed at modeling contracts and SLAs suitable to support contract owners during service composition and monitoring phases. The proposed WSLA extension is also supported by a tree-graph based tool that can help during SLA and contract composition phases.

Keywords. SLA Design; SLA Composition; SLA Measurement; Cloud Services.

1 Introduction

Modern Information Technology (IT) environments heavily exploits services according to a provide-and-consume paradigm, allowing users to perform a wide variety of tasks. Service terms have to be rendered as contracts that should be effective and flexible from a strategic point of view as well as workable from an operational perspective. In this scenario, Service Level Agreements (SLAs) [1] play a key-role in describing service relevant features by specifying the agreements between providers and consumers with respect to Quality of Service (QoS), service level guarantees, Service Level Objectives (SLOs), compliance to users' requests, service time-sustainability and so on. Each of these aspects has different levels of complexity. SLAs also define measurements and criteria to be used when services fail or deviate beyond specific tolerability thresholds from the agreed-upon contract terms but such specifications may be qualitative and difficult to be transformed in machine-readable elements. SLAs are: a) flexible and potentially language-independent; b) capable of defining both measurable

(e.g., accuracy, availability, latency, cost) and non-measurable aspects (e.g., reputation), therefore can be used for modelling domain information. Normally SLAs refer to single service provisioning and consuming. Cloud Computing Services (CCSs) and Business Process Outsourcing (BPO) represent two typical scenarios where services should be aggregated properly for guaranteeing on-demand access to customizable resources. The available methodologies and frameworks, however, lack in effective formalization of service (and corresponding contracts and SLAs) composition. Therefore, IT managers must a) supervise service functional composition and b) assess precisely Service Levels (SLs) when they design the overall contract for the delivered service. Similarly, consumers usually require rapid service provisioning, reliable resource exploitation, reduced management efforts and frequent interaction with service 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 auditing tools. Selected composing contracts and their specifications should allow users to obtain the terms and conditions of the overall contract in a straightforward, non-domain-specific manner; otherwise, these aspects may severely hinder providers' accountability when services do not match with their defined SLs or when services exhibit failures beyond the allowed guarantees after their delivery.

In this research, we present a descriptive template for contract and SLA composition that extends WSLA. WSLA is one the several formal specifications developed so far to model SLAs and their subject of application so far and it has been selected since it is XML-based and sufficiently general to cover different domain scenarios). Its proposed extension overcome its expressivity limitations for service and contract composition scenarios and allows us identifying accountability issues more efficiently in the Cloud scenario. We also propose a tool for easing the design and visualization phases in service contract composition, named Contract-Aware Service COmposer (CASCO), specifically tailored to IT contracts. We validated our model, in a SLA composition scenario, by using the tool to calculate five Key Performance Indicators (KPIs) typically adopted in the CCSs: availability, response time and Mean Time To Response (MTTR), Mean Time Between Failure (MTBF), Mean Time To Failure (MTTF).

The paper is organized as follows. Section 2 analyzes existing works on contract representation and SLA management, Section 3 and 4 present the CASCO tool, its design and implementation. Conclusions are presented in Section 5.

2 Theoretical Background

2.1 SLA for SOAs and WSs

A SLA referring to a specific service should consider: a) description of agreeing parties, QoS and obligations; b) matching with required QoS; c) collected metrics (and when and by whom); d) penalties for failures or unmatched SLs; e) responses to non-deliveries; f) providers' responsibilities; g) effects of technology re-alignments.

SLAs are enforced by management policies, thus ad-hoc frameworks are needed to ensure that applications and services meet the required SLs at deployment, operation,

support and maintenance stages [2] and they should be machine-readable, in order to support service discovery, selection and processing with respect to QoS and SLs. This originates numerous language specifications to define SLAs. Some of them adopt a combination of high-level modeling languages and knowledge representation techniques, such as: SLAng [3], where SLAs are defined by mixing natural and object-constraint languages; RBSLA [4], where SLAs are provided as RuleML constructs; PANDA [5], where SLAs are managed by a multi-agent system primarily referred to Enterprise Resource Planning systems (ERPs).

Despite their high-level of expressivity, those approaches are not so suitable for machine processing and for theintegration with Web Services description languages (e.g., WSDL [6]). Thus, XML-based solutions were proposed, such as the Web Service Level Agreement (WSLA) framework [7]. Its main elements are: subjects, services and obligations (which comprise SLOs and action guarantees). Its first prototype was introduced into WSTK [8], an integrated IBM environment for WS design, implementation and management. Although WSLA allows defining, negotiating, deploying, monitoring and enforcing SLAs, it has some limitations: 1) it refers to two-party contracts only; 2) partner responsibilities are described in terms of parameters and metrics, without considering their actions; 3) corrective management actions are not well supported. Subsequently, the WS-Agreement [9] protocol was proposed to facilitate: 1) agreement contextual definition, 2) service attributes definition, 3) customers support (by allowing the XML formalization of service requirements), 4) providers support (by allowing resource monitoring). However, neither third parties' contributions nor service composition monitoring could be modeled. The Cremona architecture [6] by IBM adopted this protocol but neither decision-making capabilities nor support workflow were provided for managing service compositions.

Modeling and monitoring aspects related to service chains and networks in many business ecosystems are usually based on graph models: BPMN [10] and BPEL [11] models represent a viable alternative **Errore. L'origine riferimento non è stata trovata.**for technicians and domain-skilled professionals, but the majority of the stakeholders does not benefit from adequate modeling tools.

Therefore, a standardized model would enable to apply templates to every type of contracts and SLAs, and to categorize contract terms in different services domains. However, despite their usefulness, SLA standardization policies are still in their infancy, thus providing IT professionals with just static or semi-structured information rather than structured contents during the entire service lifecycle [12] (e.g., pre-instantiated, active and terminated SLAs, metrics, service descriptions, guarantees). Moreover, SLA data belong to several domains and the presence of various vocabularies of provisioning terms determines semantic and structural SLA heterogeneity. This hinders the comparison and the efficient SLA manipulation in distributed service markets.

3 The WSLA Model Extension

Contracts for CCSs often result from the composition of underpinning service contracts, thus their specifications refer to those of composing contracts. Starting from

them, final providers need to automatically obtain the terms of the overall contract, after defining composition rules. A service-based approach of this problem enables to derive the final contract from service composition rules, and contract specifications from the application of these rules on the parameters of each low-level contract and service. Consequently, services from different providers can be used and composed [13] but service contracts (and SLAs) must be properly defined.

A composed contract is obtained from the integration of other contracts, signed with different providers [14], and its terms and conditions can be derived by applying composition rules to the composing contracts. Similarly, this implies that a provider of a composite contract is able to extract and feed clients with the correct terms from the component contracts. The provider has also to manage responsibilities of underpinning providers if necessary (i.e., service-oriented approach to Contract Management).

In order to model this scenario, we introduce three levels of conceptual abstraction: 1) the *Contracts Layer*, which collects service-provisioning agreements; 2) the *Services Layer*, which gathers the composing services; 3) the *Resources Layer*, which contains IT infrastructure components. The links across levels represent dependencies between contracts and services and between services and resources respectively. The links across the same level represent resources topologically composed to deliver a certain service. Thus, different levels of internal visibility of its composing sub-services are possible for a given service, depending on which kind of details are available about them. For instance, sub-services can be described by using their external parameters only (e.g., availability, response time, MTTF, etc.) or by exposing their internal details, thus enabling more precise monitoring. We decided to consider the Services Layer and the Resources Layer as a unique layer at this stage of our research, in order to strictly focus on highlighting the issues arising in service and SLA aggregation. In this way, we just consider contracts in terms of basic composing elements, thus forwarding the subsequent step of service-to-resource mapping to further research developments.

The aspects described so far cannot be fully described by current SLA models and frameworks, therefore we propose a WSLA extension that introduces SL management aspects, according to [15]. Our template presents contracts as composed by three sections: 1) the Agreement Info, which handles general contract information (e.g., name, description, start and expiry date); 2) the *Agreement Parties*, which models contracting parties and *third parties* (i.e., entities providing goods or services in underpinning contracts that have to support service delivery to final customers); 3) the *Agreement Services*, which manages services information for 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. In turn, each SLA exhibits specific features, such as reference period, description, report date and SLA specification. SLAs specifications usually refer to multiple parameters; therefore, we decided to include the following properties:

- Cost (established for a SL with specific parameters);
- Start/End Date;
- Sample Period (used during SL monitoring to make decisions about service times and to improve quality);
- Service Level Objective (SLO): the SL threshold value;

- Working Window: time span for SL generation;
- Penalty: the set of algorithms used when providers fail to meet the agreed SLs;
- Service Level Indicator (SLI): it includes units of measure and algorithms for SLA calculation and service monitoring.

The contract template described so far can be modeled as a directed tree graph (i.e., digraph) where nodes represents template components whilst oriented edges are the available properties. Edges are labeled with corresponding cardinalities (i.e. how many nodes having those properties can be repeated). This approach allows representing data about a single contract, even if with a great level of detail. Therefore, composition topologies and rules have been introduced as additional elements to WSLA, for describing SLA and contract composition. Topologies offer an intuitive way of creating service chains matching users' requests. We adopted four compositional patterns: series (each service starts once the previous one stops), parallel (services work simultaneously), loop (several cycles are needed to complete a service), condition (service execution is oriented by an initial choice) [16]. Topologies can be further composed to create complex systems (in a way similar to circuit elements are cascaded in circuit theory) and the final service is obtained by recursively choosing the available topology patterns.

Fig. 1A depicts the contract template while Fig. 1B and 1C show the composition and rule model, respectively. Listing 1 enlists the formal definitions for all the entities mentioned so far, obtained through sets language.

Definitions
$Agreement := \{AgreementParties_1, AgreementInfo_1, AgreementServices_i\}, \quad with\ i \in [1,n]$
$AgreementParties := \left\{Contractor_1, CustomerM_1, \{CustomerS_i\}, \{3rdParty_j\}\right\}, \quad with\ i \in [0,n], j \in [0,m]$
$AgreementServices := \{\{Service_i\}\}, \quad with\ i \in [0,n]$
$Service := \{\{SLA_i\}, GuaranteeCalendar_1, ServiceInfo_1\}, \quad with\ i \in [0,n]$
$GuaranteeCalendar := \{\{IntervalDayYear_i\}\}, \ with\ i \in [1,n]$
$SLA := \{\{SLASpecification_i\}\}, \ with\ i \in [1,n]$
$SLASpecification := \{\{Penalty_i\}, SLI_1, SLASpecificationInfo_1, WorkingWindow_1\}, \quad with\ i \in [0,n]$
$WorkingWindow := \left\{\{IntervalDayWeek_i\}, \{IntervalHour_j\}\right\}, \quad with\ i \in [1,n], j \in [1,m]$
$Penalty := \{\{Event_i\}\}, \ with\ i \in [0,n]$
$Event := \{Algorithm_1\}$
$Composition := \{\{Service_i\}\}, \ with\ i \in [1,n]$
$Service := \{\{Composition\}\}, \ with\ i \in [0,1]$
$OutputService := \{\{Composition_i\}\}, \ with\ i \in [0,1]$
$ServiceCompositionTopology := \{\{SLACompositionRule_i\}\}, \quad with\ i \in [1,n]$
$SLA := \{\{SLACompositionRule_i\}\}, \quad with\ i \in [0,n]$
$SLACompositionRule := \left\{\{Series_i\}, \{Parallel_j\}, \{Condition_k\}, \{Loop_l\}\right\}, with\ i,j,k,l \in [0,1] : i+j+k+l = 1$

Listing 1. Contract template (formal definition)

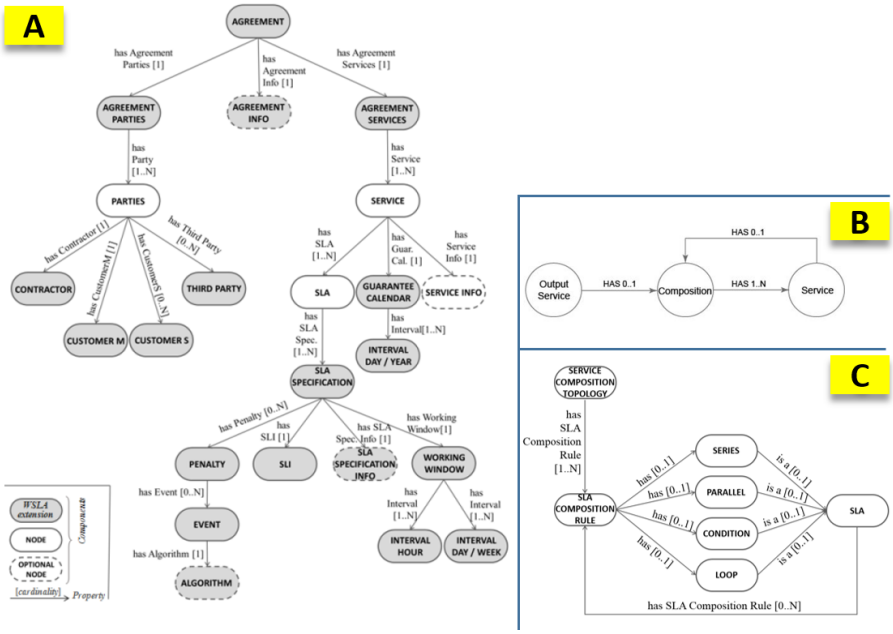

Fig. 1. Contract template as a digraph (A); composition topology (B) and rule (C) models.

4 Tool Description

The main goal of CASCO is to test the model proposed in Section 3 and to help customers and providers in composing SLAs 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 the SLAs of composed services starting from elemental ones. During the execution phase, the system comes in handy since it offers an engine that is able to calculate the result of composite SLAs. The tool has been designed starting from identifying stakeholders for a generic service providing/consuming firm: both managers (general, technical and business, department, human resources, service and contract) and administration personnel have been considered. Different tasks have been identified during the subsequent goals elicitation. For instance, a service manager can use the tool 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 companies or final users. The tool exploits a 3-layer architecture (Fig. 4) to clearly separate data management, business logic and user interface components. Moreover, the application complies with the widely adopted Model-View-Controller (MVC) pattern. 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 XML files de-

scribing agreements, topologies and rules. The Presentation module manages user operations and parses input, in order to generate the XML files (Fig. 3). The *Manager* module is further divided into four subsections. The *Parser* allows extracting and manipulating data from XML files to store them in the graph database (DB). The *Agreement Manager* handles contracts and the operations that have to be performed on them. The *SLA Compositor* creates composed services, defines corresponding measurement metrics and stores output services into the DB. The *SLA Control* allows accessing services information and parameters for resources and services monitoring.

From an implementation point of view, the digraph model has been defined into a NoSQL graph DBMS (Neo4j v.1.9.6/v.2.0.0 [17]), in order to store SLA and contracts information as well as composition topologies and rules. The tool has been developed in PHP with HTML5 compliancy for the front end in order to be natively compatible with mobile devices, and in Java for the back end. We used Spring-data-neo4j (v2.3.4) [18], a framework for mapping classes with the elements stored in Neo4j, and Neo4jPHP (v.0.1.0) [19], a PHP library for accessing Neo4j database. The tool has been used to evaluate and assess SLAs during the design phase of service composition.

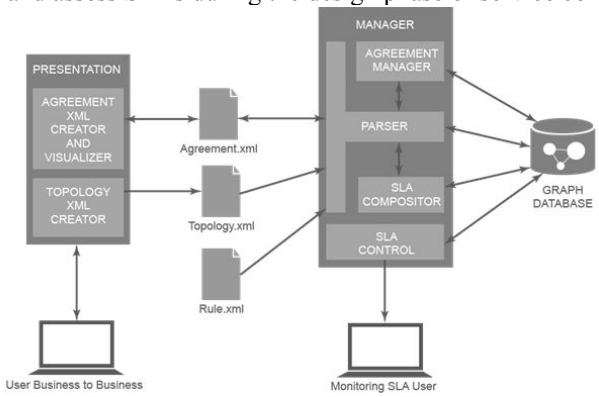

Fig. 2. The CASCO tool architecture in the large

5 Applying The Model To A Real Scenario

5.1 SLA Metrics

Given a specific topology, services are composed and SL specifications are calculated accordingly. To decide the acquisition of a service as part of a final one, a manager needs to know which constraints the final service undergoes. Indeed, this is necessary in order to evaluate the feasibility, the sustainability, the compatibility of the composite service against the requirements. 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, in order to perform a suitable SLA composition, metrics must be the same and their measurement units must be commensurable and compatible (e.g., all

measures have to be time-related or all measures must be percentages, etc.). We considered five metrics exhibiting a great relevance in cloud scenarios to demonstrate the feasibility of the proposed model by using our ad-hoc developed tool. The selected metrics are: Availability, Response Time, Mean Time To Failure (MTTF), Mean Time Between Failures (MTBF), Mean Time To Repair (MTTR).

Fig. 3 introduces some relevant quantities used in calculating the adopted metrics. We refer to *Service Downtime (TSD),* as the time period during which a specific service is not available (i.e., 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, $t_{u,i}$). Similarly, *Service Uptime (TSU)*, is the time interval during which a specific service is working (i.e., the time range between the *i-th* occurrence of the service start, $t_{u,i}$, and the *i+1-th* occurrence of its failure $t_{d,i+1}$). The period between two failures (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*), that is: *TBF* = *TTR*+*TTF*.

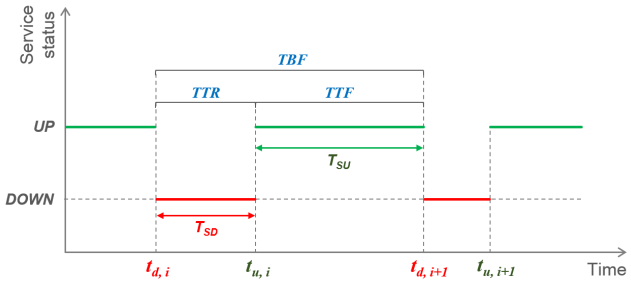

Fig. 3. Significant temporal quantities in service status

Availability: it describes how long a service S_A is available within its temporal window of analysis. Being *TAW* the duration of this temporal window and *TSD* the service downtime period, the availability *A* is expressed as in (1):

$$A(S_A) = (T_{TW} - T_{SD})/T_{TW} \quad (1)$$

Response Time: it quantifies how long it takes to a service S_A to answer client's interrogation. If we have N interrogations from client, each with its own response time $rt^{S_A}(t)_i$, the overall response time *RT* is the average of the distinct responses *rt*, as in (2):

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

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

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

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):

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

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

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

5.2 Composition Rules

New services resulting from a process of composition [20] exhibit different metrics, according to the composition rules and topologies selected during the design phase. In order to define such rules, let us refer to series and parallel topologies and let us consider two different services S_A and S_B that have to be composed and whose metrics are known before the composition is performed. By applying math relationships borrowed from circuits theory, the indicators specified in (1)-(5) can be derived for parallel and series topologies, as in parallel and series configurations of circuital elements, thus providing us with the corresponding composition rules. Table I resumes the rules for Availability, Response Time and *MTTF* metrics. For the sake of brevity, their mathematical derivation is not reported in this paper.

TABLE I. COMPOSITION RULES FOR THREE METRICS (SERIES, PARALLEL)

Metric	Series Topology	Parallel Topology
Availability	$A_S = A(S_A) \times A(S_B)$	$A_{//} = 1 - [(1 - (A(S_A)) \times (1 - A(S_B))]$
Response Time	$RT_S = RT(S_A) + RT(S_B)$	$RT_{//} = max\{RT_A, RT_B\}$
MTTF	$MTTF_S = 1/((MTTF(S_A)+MTTF(S_B))$	$MTTF_{//} = MTTF(S_A)+MTTF(S_B) - MTTF_S$

5.3 Validation

Complex modeling approaches typically require a multi-year effort to be validated. Both design- and run-time validation activities against functional and non-functional requirements are required. During validation, specific aspects should be checked, such as the suitability of the model to describe properly a significant variety of SLAs, its amenability to be integrated within information systems, its capability to handle managed resources. Therefore, we started with an internal validation. A set of agreements having as metrics the ones described in Section 5.1 has been simulated. Then, the composition rules described in Section 5.2 have been loaded into the tool (within the corresponding XML file, see Section 4). Specific tests have been executed to both evaluate the template feasibility for modeling SLA aspects and the tool usability for IT managers and contract owners. Therefore, 20 Italian users, aging from 30 to 50 y.o. and belonging to technical and juridical areas, have been selected as testers. The candidates have been assigned with three tasks deemed as representative of a concrete deployment scenario for the application. *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 in order to assess their mood against the proposed system and the features it exposes. A 10-point psychometric Likert scale has been used to measure users' responses in evaluating each task, according to the

easiness in retrieving information, navigational and message clarity, ranging from 1 (i.e., worst evaluation) to 10 (i.e., best evaluation). Table II enlists average scores (*AS*) for the features evaluated when performing a specific task and mean execution times (*MET*) for each task (means are computed across all 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 service composition) and do not refer to possible issues in contract and SLA modeling or missing modeling functions.

TABLE II. OVERALL SCORES AND EXECUTION TIMES FROM VOTERS' POOL

Evaluated Feature	AS[1]	Task	MET[2]
Easiness in retrieving information	6.45 / 10	1	42.3[s]
Clarity of navigation and messages	6.8 / 10		
Clarity of agreement registration requests	8.5 / 10	2	183[s]
Easiness of composite service creation	6 / 10		
Clarity of content presentation	9.25 / 10	3	352[s]
Clarity of graphical representation	9,35 / 10		

[1] *AS: Average Score (per feature)* – [2] *MET: Mean Execution Time (per task)*

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. An extension of the widely known, XML-based WSLA framework [7] has been proposed in order to obtain a flexible template for IT service contracts. We aim at overcoming two main issues in managing SLAs: 1) tackling the lack in standard models representing service contracts and their SLAs in service network environments; 2) making SLAs more machine-readable. The former one is obtained by complementing WSLA with composition topologies and rules, extremely useful when dealing with CCSs (typically provided as compositions of multiple, third-party services). The latter one is achieved by modeling our template as a digraph (implemented in a NoSQL graph DBMS). According to the proposed model, we developed a specific tool named CASCO (Contract-Aware Service COmposer) in order to offer SLA evaluation and assessment capabilities to IT professionals during design phase. Five metrics (i.e., *Availability*, *RT*, *MTTR*, *MTTF*, *MTBF*), typically used in CCSs were selected to evaluate the template feasibility in our tool. We have ascertained that the WSLA extension is suitable to handle all the information needed to describe contracts and SLAs for composed services.

In the near future, this research will be widened and improved by: 1) extending the template (SLAs for BPO contracts); 2) proposing an ontological formalization of the template; 3) adopting rule-based policies to perform real-time monitoring of composed services as in Complex Event Processing (CEP) scenarios.

Acknowledgement

The research has been partially supported by Engineering SpA and Essentia SpA.

References

1. Telemanagement (TM) Forum: SLA Handbook Solution Suite v2.0. (2008)
2. Allen, P.: Service Orientation-Winning Strategies and Best Practices. Cambridge Univ. Press, 2006
3. Paschke, A.: Knowledge Representation Concepts for Automated SLA Management. Decision Support Systems 46(1), 187-205 (2008)
4. Boley, H., Tabet, S., Wagner, G.: Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In : First Semantic Web Working Symposiium (SWWS-2001), pp.381-401 (2001)
5. IST: PANDA: Collaborative Process Automation Support Using Service Level Agreements and Intelligent Dynamic Agents in SME Clusters. PANDA Research Project Report (2007)
6. Ludwig, H., Dan, A., Kearney, R., Heights, Y.: Cremona: an Architecture and Library for Creation and Monitoring of WS Agreements. Research Report, IBM (2004)
7. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements. Technical Paper RC22456, IBM Research (2002)
8. IBM Corp.: IBM Web Services Toolkit (WSTK) v2006. Available at: http://www.ibm.com
9. Andrieux, A.: Web Services Agreement Specification. Memo GFD-R-P.107, WG (2007)
10. Dijkman, R. M., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Information and Software Technology 50(12), 1281-1294 (2008)
11. Xiang, F., Tevfik, B., Jianwen, S.: Analysis of Interacting BPEL Web Services. In : 13th Int. Conf. on World Wide Web (WWW'04), pp.621-630 (2004)
12. Stamou, K., Kantere, V., Morin, J. H.: SLA Data Management Criteria. In : IEEE Big Data (2013)
13. Piccinelli, G.: Service Provision and Composition in Virtual Business Communities. HP Technical Report HPL-1999-84, Hewlett-Packard (1999)
14. Bochicchio, M. A., Longo, A., Giacovelli, S.: SARA: a Tool for Service Level Aware Contracts. In : IFIP/IEEE Int. Symp. on Integrated Network Management (2013)
15. Stamou, K., Kantere, V., Morin, J. H.: SLA Template Filtering: a Faceted Approach. In : 4th Int. Conf. on Cloud Computing, GRIDs and Virtualization (2013)
16. Ul Haq, I., Schikuta, E.: Aggregation Patterns of Service Level Agreements. In : 8th Int. Conf. on Frontiers of Information Technology (FIT'10) (2010)
17. Neo4j: Neo4j. Available at: http://www.neo4j.org
18. Spring: spring-data-neo4j. Available at: https://projects.spring.io/spring-data-neo4j
19. Jadell, J.: Neo4jPHP. Available at: https://github.com/jadell/neo4jphp
20. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of Service for Workflows and Web Service Processes. J. of Web Semantics 1, 281-308 (2004)