SLA data management criteria

Katerina Stamou, Verena Kantere, Jean-Henry Morin
Institute of Services Science, University of Geneva, Switzerland

***Abstract*—Service Level Agreements (SLAs) represent service management contracts that are processed by monitoring and measurement mechanisms for the evaluation of the signatories adherence to the agreed service levels during service execution. The paper discusses SLA data management characteristics that need to be considered in the design of data models for SLA documents. The SLA anatomy is introduced with respect to the Web Service Level Agreement (WSLA) [1] language specification. Furthermore, the paper highlights current obstacles for the integration of automated SLA management in the cloud business setting. The contributed SLA data analysis maps SLA terms to data management attributes according to their operational relevance during the SLA activity. We present an SLA digraph model for the automated SLA formulation and data handling. The SLA digraph is introduced as a programming module that sits on the application layer and communicates with backend data stores for the SLA persistence.**

I. Introduction

In the cloud computing context, IT services are provisioned on-demand by utilizing distributed computing resources and economies of scale. Every customer needs to agree with a Service Level Agreement (SLA) in order to lease a new service. Traditionally, providers define SLAs, in which they guarantee explicit service-level bounds over a predefined, agreed period. SLAs represent contractual terms and conditions between service providers and customers. Their content assures the mutually agreed service levels between a provider and a consumer [2].

SLA contracts describe obligatory service provisioning terms. Such terms encapsulate Quality of Service (QoS) attributes and functional service properties. Furthermore, SLAs may include a multitude of technical and business service-level objectives (SLOs) along with metrics and service parameter endpoints. Thus, SLAs assist with the calculation and measurement of service parameters, which in turn indicate the provider's adherence to the promised service levels.

We assume that SLAs come in machine-readable format and that cloud service providers use automated systems for the management of their offered services. Our view on SLAs exceeds their basic usability as service level instruments. We consider SLA templates as *what-you-see-is-what-you-get (WYSIWYG)* artifacts that customers can use to negotiate and finalize their service selection. SLA templates represent pre-agreed SLAs that typically describe all available service provisioning options. Moreover, we assume that electronic marketplaces and service agencies utilize SLAs and SLA templates to offer customers service-management facilities, where shared SLA data are distributed over remote repositories and processed automatically.

The scope of this work is to introduce SLA data characteristics that impose special storage and data operation structures for SLA documents. Compared to other types of electronic contracts and software licenses, the values of SLA terms usually represent the outcome of multiple backend resource management operations. SLA information needs to be accessed simultaneously by diverse processes (e.g. monitoring, auditing mechanisms) that manage the overall service execution. An appropriate SLA data structure and persistence mode can contribute to efficient service level management.

Currently SLAs for cloud services lack standardization that would provide them with more structured content. Thus, they mostly appear as semi-structured, static information and their utilization by virtual cloud markets is limited. Moreover, the description of offered cloud services is superficial and the provided SLA contracts are inadequate for customers to manage their leased services. Given such facts, it is ambiguous if cloud service customers actually get the service levels that they are paying for. Our work highlights current obstacles that restrict the SLA role in service marketplaces to static, *terms-and-conditions* documents and augments on research challenges that can reverse the SLA utilization into a more dynamic and value-added one.

The paper initially provides a precise analysis of the SLA structure and primary components according to the WSLA language [1] and to the WS-Agreement initiation protocol [3]. To avoid a semi-structured, insufficient SLA representation, the SLA anatomy, formalization and initialization cycle are discussed with respect to the role of language contractual elements including signatory parties and by following the WSLA specification.

The primary contribution of this work is a structured analysis of SLA data management needs. We organize our study by collecting a set of generic data operational attributes. SLA terms are mapped to such attributes according to their operational relevance in the SLA execution. The analysis helps to formulate a view on the data structure that is adequate for SLA data management over distributed web resources. Moreover, the provided study can be used as a generic tool for the design and deployment of SLA data stores.

Given the SLA anatomy and data analysis, we use the WSLA language to build and contribute an SLA digraph data model, where nodes and edges represent the SLA content. We propose that the SLA digraph can be used as a data model for the management of SLA information. The initial model, presented in this work, is implemented as a Python 2.7 programming module.

The paper is organized as follows: Section II examines the

978-1-4799-1293-3/13/$31.00 ©2013 IEEE

SLA structure and initialization cycle. Section III discusses current SLA obstacles that bound the SLA utility in cloud marketplaces as well as research challenges to overcome such obstacles. Section IV focuses on the SLA data analysis with respect to data management characteristics. Section V introduces the proposed digraph model for the data management of SLAs. Finally, Section VI elaborates on criteria and objectives of our experimental study with respect to the SLA digraph model evaluation. We summarize the paper with related research, conclusions and on-going work.

II. SLA ANATOMY AND FORMALIZATION

The distributed computing community has driven research and technical advancements on SLA management to cover immense needs for resource reservation and monitoring of job execution. The WSLA [1] and WS-Agreement [3] language specifications have been used by researchers to express SLAs in a machine-readable format.

The GRAAP working group proposed the WS-Agreement as a language and a protocol to conduct SLAs. The WSLA language has been led by IBM research on utility computing. According to [1], an SLA complements a service description. An important attribute of both language specifications is that they enable the automated parsing and manipulation of the SLA document by backend processing systems.

In contrast to other contracts for IT services, properties within an SLA have to be monitored during workload execution to audit potential service-level violations and verify adherence to the agreed SLOs. In the literature, SLAs are often perceived as automated processes that merely assist the monitoring and scheduling of computational tasks.

Both WSLA and WS-Agreement specifications use a tree data structure to represent the SLA information. Tree branches illustrate separate SLA sections and tree leaves inner section terms. XML is used by both schemas. Figure 1 illustrates a high level view of the WSLA language anatomy, according to which the core elements of an SLA are the following:

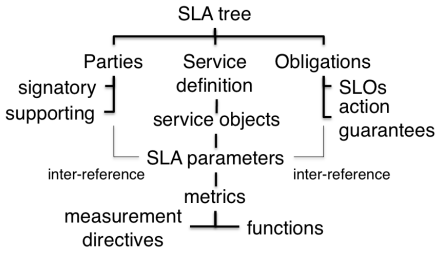

Fig. 1: Web Service Level Agreement (WSLA) structure

Parties: Signatory parties consist of one service provider and one service customer. Supporting parties represent third parties that operate on behalf of either or both signatories.
Service definition: Service objects represent description terms and include SLA parameters that indicate quantitative as well as qualitative metrics.
Obligations: A provider defines guarantees in the form of obligations either as service level objectives (SLOs) or as action guarantees. SLOs represent measurable targets that a provider promises to fulfill during service execution. SLO values can be verified via monitoring and auditing. Action guarantees signify tasks that the provider, one or more supporting parties or, in some cases, the customer will take to establish the promised service levels of one or more SLA parameters. A well-defined description of service and resource parameters is a prerequisite for the specification of QoS attributes in the form of either SLOs or action guarantees.

The core elements of the language are further divided into granular sub-elements. In the case of service objects, the granularity of information reaches the level of URI sources that can be monitored and measured. In the case of service obligations, the flow of the information content involves business level objectives, as well as SLO evaluation functions and expressions that typically follow first order logic.

Moreover, obligations may include the formalization of penalties in case of service-level violations, as well as of reimbursement conditions and rewards that represent penalty complements. The granularity of guarantee description depends on the information content and on the complexity of the guarantee evaluation. A guarantee can be defined for a service atom (i.e. a standalone service or service component) or for a service bundle. Moreover, a guarantee may be generic enough and apply to the overall service provisioning (e.g. customer authentication/authorization options, etc).

A. SLA initiation cycle

In addition to the SLA language structure and semantics that both specifications [1], [3] provide, the WS-Agreement defines a protocol for the SLA initiation between contracting parties. Figure 2 summarizes the basic steps in the information exchange:

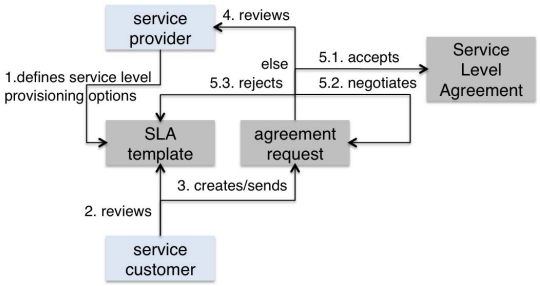

Fig. 2: SLA initialization lifecycle

1) A provider depicts their provisioning availability, capacity and service leasing options in one or more SLA templates. An SLA template represents a pre-instantiated agreement that is prepared by a service provider for customer consideration. The SLA template describes the agreement content that a provider is willing to accept during communications with customers. Thus, it provides a holistic view on a provider's resource capacity and provisioning plan.
2) Both specifications suggest the use of customer and provider templates for the exchange of counter offers in

the process of agreeing on service levels. SLA templates are typically accessible to customers through a service registry that pulls data from designated repositories. To decide which provisioning is more suitable for their needs, customers review SLA templates and proceed with either agreement initiation or negotiation with one or more providers.

3) After reviewing SLA templates, a customer will select and customize the one that best suits their computational needs and will send it back to the service provider as an agreement request.
4) On receiving the agreement request, a provider will review it and estimate whether the request falls within the existing provisioning capacity.
5) After reviewing the agreement request, a provider can either accept it and initiate the agreement or negotiate with the customer on service-level terms or reject the request.
6) Upon agreement initiation, the SLA is subject to the agreed monitoring and auditing services until the execution of the computational task is complete.

III. SLA ROLE IN VIRTUAL ECONOMIES

As described in Section II, SLAs control the measurement and auditing of available resource parameter values. The SLA definition provides an explicit view on how the provisioning of a service is planned. It also indicates precise bounds of service levels that a provider can supply. Providers use SLAs to control the levels of resource consumption and service availability.

Currently, SLAs hardly appear in cloud marketplaces. The promotion of IT offers to customers primarily relies on high-level service descriptions that are typically static and that do not allow for any processing. The role of SLAs is peripheral and they are often materialized by documents of "terms-and-conditions" that usually do not involve functional service aspects. On top, SLAs lack standardization, which would inevitably lead to more structured, rule-based SLA formulation.

We view SLAs and their derivatives as dynamic information that is updated at frequent time intervals. In [4] we demonstrated how SLA templates can be utilized as customer drivers for service selection by a third-party service (e.g. virtual marketplace, service-aggregator) since SLA templates enclose all details on how services are to be provisioned. Our experimentation showed that the SLA content modularity enables the rapid retrieval of all inclusive information through queries that are performed in one or more databases. This fact allows for the SLA manipulation in multiple ways, which in [4] we exercised by treating SLA template sections as facets.

A. Cloud SLAs challenges

In the scientific literature, SLAs are hardly viewed as end-user documents, but merely as automated processes. In contrast, cloud marketplaces treat SLAs as static documents. It is thus challenging to find the right equilibrium between these two orthogonal aspects and combine machine-readable with user-friendly SLAs into a uniform process that can be used by both backend systems and front-end web services.

In the cloud business setting, diversified services are offered. Description characteristics and provisioning guarantees vary considerably, even if they describe similar services in different contexts. Providers from various business domains use customized terminology to describe service parameters, metric functions and guarantee definitions. In service descriptions, terms like "availability", "throughput" and "performance" are usually included in ambiguous ways, which may be confusing for service customers. Various vocabularies of provisioning terms represent a primary cause for SLA heterogeneity.

On a wide scale, SLA semantic and structural heterogeneity represents a challenge because it complicates the SLA template comparison and hinders any attempt to efficiently manipulate SLAs in distributed service markets. SLA content heterogeneity addresses issues that deal with research opportunities for data management, information retrieval and language processing. Moreover, it highlights the need for SLA standardization that would allow for classification of key performance indicators (KPIs) or would mandate the inclusion of specific functions per business domain.

In the cloud computing setting, SLA formulation highly depends on resource availability that is prone to customer-demand variations [5]. Hence to manipulate SLA templates, the template content must be processed dynamically. As the information depth of SLAs is unbounded, frequent data updates may cause performance delays in the data exchange between customers and providers. Yet, a reasonable storage schema for SLA templates is under question. On one hand, one may argue that since SLA templates represent dynamic information objects, they should not be stored at all. Instead, they should be kept in-memory for as long as they are valid and then be immediately replaced. On the other hand, a modular SLA data model can help with the preservation of SLA templates for longer time periods. Moreover, it can facilitate frequent content updates according to provider resource capacity and service availability. Our research centers on the latter aspect.

According to [3] and [1], SLAs represent nested tree structures. On the business setting, SLA documents may include heterogeneous characteristics and they are typically unbounded in terms of length and content. A challenge for the SLA manipulation is to select a data structure that facilitates quick traversing within nested information paths. For example, a modular structure provides isolation between SLA components and allows for categorization of SLA terms. Moreover, an efficient data structure enables the exploitation of finer grained information, while preserving the necessary SLA data interconnectivity. Finally, an adequate SLA data model applies to diverse types of SLAs and can be accomodated by various management schemas according to application and infrastructure specifics. The rest of the paper concentrates on SLA data requirements that indicate a satisfactory SLA data structure for the cloud computing setting.

IV. SLA DATA ANALYSIS

Under the term **SLA data management** we enclose all SLA data operations that take place before, during and after service execution. Such data include pre-instantiated, active and terminated SLAs as well as fine-grained service elements (e.g. resource metrics, service descriptions and provisioning guarantees), whose combination assembles the SLA content.

In a service economy, the SLA formalization using a structured, well-defined schema (like [1], [3]) extends SLA manipulation opportunities towards business-oriented operations. However, an SLA data model is required to support the automated management of processes with respect to real-time data monitoring and auditing. Compared to other types of service contracts (e.g. terms-and-conditions, software licences) the values of SLA terms need to be monitored and measured during service execution to verify that SLOs are met and that no service violations have occured.

Accordingly, the measurement of data values has to be depicted in the SLA content in frequent time intervals. The requirement for real-time data updates particularly applies in the cloud computing setting, where services are exchanged on demand and business relationships may enclose financial responsibilities (e.g. violations of SLA terms may trigger customer re-imbursements or impair a service provider's reputation). Furthermore, nested SLA information may include interdependencies between diverse components or component sets (e.g. a change in an SLA parameter value may affect respective SLO values). Such dependencies need to be addressed by the SLA data model as they relate to data correctness.

To support our claim that a graph data model is adequate for the SLA data management needs, we collected generic, data-operational attributes and mapped their significance to core SLA components that are typically met in any SLA. Table I illustrates this mapping. The first column of Table I lists the selected data management characteristics. SLA components are mapped to data attributes according to their required operations (e.g. monitoring, auditing, alerting) for successful service management. The last column of Table I illustrates the significance of data attributes to the overall SLA document. Since SLAs are not yet standardized and their content is semi-structured, a multitude of SLA terms may have similar or additional management needs, which can be determined by specific application criteria and desired functionality.

Data volume: refers to the size of information to be stored and processed. In the case of SLAs and SLA components, the size is negligible, thus there is no direct need for storage allocation or big data transfers. The volume per service object (assuming the hierarchical inclusion of SLA parameters, composite and resource metrics) as well as the size of data per guarantee definition typically lies in the kilobyte range. Moreover, SLA data operations (e.g. customer or provider alerts) can occupy temporal space instead of disk storage. On the other hand, and assuming the exploitation of SLAs by public marketplaces, there can be massive volume of interactive client requests over stored SLA data. Thus, a flexible data structure and database schema should allow for alternative persistence modes that efficiently address dynamic SLA data needs and that allow for parallel processing of real-time data queries. As there is no direct mapping with the depicted SLA information, this attribute is not illustrated in Table I.

Data characteristics	**ST1**	**ST2**	**ST3**	**ST4**	**SLA**
accessibility, integrity	⋆	⋆	⋆	⋆	⋆
velocity rate	high	high	high	low	na
replication, staging	⋆	⋆			⋆
service dependencies	⋆		⋆	⋆	⋆
cleanness	⋆	⋆	⋆	⋆	⋆
accuracy	⋆	⋆	⋆	⋆	⋆
ownership, authenticity	⋆	⋆	⋆	⋆	⋆
heterogeneity	⋆	⋆	⋆	⋆	⋆

TABLE I: Data management characteristics- SLA terms mapping. ST1: SLA parameter, ST2: composite or resource metric, ST3: service level objective (SLO), ST4: action guarantee, SLA: complete SLA document, na: non applicable.

Data accessibility, integrity: The definition of resource and composite metrics typically involves numerous data sources. Composite metrics can be calculated by the accumulation of resource metrics from distributed access points (e.g. URIs, query interfaces). SLA parameters are computed using composite metric values. Typically, a logical expression asserts the value range that specifies an SLA parameter. Data accessibility and data-value integrity are of great importance for the correct processing of SLA templates . The information description has to be granular enough to enclose metric dependencies, service access points (SAP) and unique resource identifiers (URIs). Such information enables resource and service management operations, e.g. monitoring processes that control the resource consumption and inform scheduling processes on scale in/out needs, auditing processes that trigger alerts on possible service level violations, etc.

Data velocity rate: SLAs contain terms whose values may change dynamically during service execution. Such terms are typically represented as resource, composite metrics and SLA parameters, which are assembled under the definitions of service objects and represent the holistic service description. Moreover, the information included in the SLA service/resource description is affected by resource demand. As observed by [5] there are time intervals when the consumption load increases and thus a provider has to either scale out or, vice versa, apply a more stringent provisioning plan. SLA guarantees depend very much on the value ranges of SLA parameters, thus are also volatile and subject to changes according to resource consumption and availability. Thus, to support value-added business operations the SLA data schema

must allow for rapid information retrieval and volatile data management.

Data replication, staging: SLA parameter and metric values may affect the definition of other SLA terms, like SLOs or action guarantees. In the cloud setting, such information is accumulated or processed directly from distributed resources, thus data may have to move among diverse repositories or to be transformed into different formats. With respect to SLA guarantees, data staging and replication may depend from provider-customer arrangements and from the provisioned service type or SLA class. For example, an agreed SLA may consist of smaller SLAs that have limited time-spans and combine schedules of parallel and sequential service execution tasks. On completion of an SLA subset, active SLA information may have to be replicated or to migrate into new data repositories where a different data format might be expected.

Service inner-dependencies: The evaluation of SLA parameter values may depend on the measurement of one or more metrics. With the term *service dependency* the authors in [6] define the relationship between an antecedent service or application component that requires an operation performed by a precedent component in order for the former to execute its function. The authors use the terms **antecedent** and **precedent** to express the two counterparts of the dependent relationship. By default such relationship is directed.

The authors in [6] differentiate service dependencies between functional and structural. The former represents generic service dependencies and principal service components to which all other service models are bound (e.g. service type, service name, service purpose (end-user, backend)). Structural dependencies illustrate detailed descriptions of components, whose interoperation represents the offered service. The structural model complements the functional one and provides explicit descriptions of service elements (technical, configuration installations etc). SLA component inner-dependencies need to be reflected in the SLA data structure and database schema since they affect operations that influence the overall SLA processing.

Data cleanness: With respect to cloud services, the SLA service/resource description may define the maximum number of database instances, tenants or disk space that is allocated per virtual machine (VM) and VM instance; moreover it may describe the virtualization platform and CPU characteristics like processor power and type. The SLA service description depicts the overall provisioning capacity, thus resource availability. Such data along with SLA guarantees should be publicly readable to help customers project their service requirements given available resources. Thus, elimination of data noise and preservation of data cleanness represent necessities for the processing of SLA information.

Data accuracy: As guarantee information enables both provider and customer to evaluate the levels of service provisioning, the provider needs to ensure the information accuracy in terms of real-time updates and correct resource measurement. Data conformity helps the service provider to preserve promised quality-of-service (QoS) levels and assures customers with respect to service consumption.

Data authenticity, ownership: During the SLA initiation cycle (Fig. 2), SLA information is exchanged and modified by the prospective signatories. Thus, a mechanism needs to guarantee the data transfer integrity and privacy among the contracting entities. Moreover, providers formulate guarantees that define value range limits for SLA parameters and configuration options for service objects. As the reputation and business prosperity of a provider primarily depends on the accomplishment of promised and agreed service levels, the provider has to ensure the authenticity, non-repudiation and ownership of publicly readable SLA information.

Data heterogeneity: As discussed in Section III, cloud SLAs lack standardization that permits a more structured content and data schema. Additionally, the variety of services offered in the cloud context requires alternative SLA forms and compliant formats that can be processed by existing web platforms. Illustrated SLA terms, including the overall SLA document are subject to service type and business criteria with respect to their formulation, machine interpretation and operational processing.

V. SLA DIGRAPH MODEL

We use the WSLA specification [1] as a reference to construct a directed graph for the SLA data model. We first transform primary WSLA language components into element sets by taking into account their cardinalities and relationships. Table II illustrates identified element sets. We use a subscript to denote their cardinality in any SLA graph instance.

SLA$\supset$ $\{Signatory_2$, $Obligations_1$, $ServiceInfo_1$,$\{ServiceDescription_i\}\}$, where $i \in [1,n]$
$Signatory_2 \supset SupportParty_i$, where $i \in [0,n]$
$Obligations \supseteq SLO_i, ActionGuarantee_i$, where $i \in [0,n]$
$ServiceDefinition_i \supseteq ServiceObject_i$, where $i \in [1,n]$
$ServiceObject_i \supset SLAparameter_i$, where $i \in [1,n]$
$CompositeMetric_i \supseteq ResourceMetric_i, CompositeMetric_j, Function_j$, where $i \in [1,n]$ and $j \in [0,n]$
i,j indicate the cardinality of element subsets and $n \in \mathbb{N}$.

TABLE II: SLA language components - Set representation

The digraph follows the guidelines of the WSLA specification with respect to language components. We add an additional node, the service-information one, to include data that identify overall service properties, which are not related with obligations of any involved party. Motivation for the inclusion of such node is derived from the WS-Agreement specification [3] that contains a similar section named "Context". The service-information node denotes SLA attributes

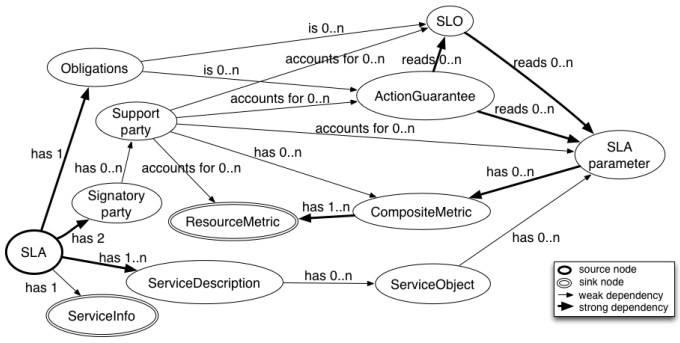

Fig. 3: SLA representation as a digraph

like the location of the provider infrastructure and customer stored data, the legislation that the service provider adheres to, as well as service monitoring and configuration options for customers, if such exist.

Figure 3 illustrates the generated SLA skeleton digraph. The proposed model consists of 12 nodes that are interconnected through 19 edges. The assembly of nodes and edges represents the SLA skeleton with respect to primary SLA language elements. The "SLA" represents the source node and the graph contains in total two sink nodes. The graph model provides a simple but powerful way to order semi-structured SLA data into structured context. Moreover, it can be extended accordingly to depict service specific properties and metrics, as well as terminology requirements and metric functions of particular business settings.

By definition, a digraph contains nodes and directed edges, which in our context represent the SLA content and are used as indicators of **unidirectional dependencies** in the graph information flow. Nodes and edges in the digraph are accompanied with properties like description, measurement functions as well as measurement type and unit.

For example, edges that connect SLA parameters to composite or resource metrics, include a property on the measurement and evaluation schedules of the respective metrics. Such property is needed in order to schedule the value update of an SLA parameter with respect to the metric values that are read and used. Such service details are important for SLA management since they enable the accurate execution of the service contract. A node or edge property is defined as a key-value pair that is embedded in the node or edge definition.

Furthermore, in Figure 3 a thick line denotes edges that indicate **strong dependencies** between nodes, while a thin line denotes **weak dependencies**. A strong dependency signifies operations between a set of nodes that are connected through a set of directed edges (e.g. the content of the antecedent node requires the precedent node for its definition or measurement). Weak dependencies indicate semantic interrelations between nodes.

As discussed in Section IV, SLA components have operational characteristics that require a flexible and dynamic data structure. For example, SLA parameters and SLOs may need to be informed at frequent time intervals on the state or value of other SLA terms. The term unidirectional dependencies is used to signify operational functions between diverse SLA components that require an antecedent - precedent relationship for the evaluation of every pair of SLA terms.

We model this relationship as following: For every pair of SLA nodes (A, B) that are connected through a directed edge labeled x: $A_{value} = function(B_{value})$, while a condition is valid and given that x represents an outgoing edge from A to B. (A, B) may denote any set of SLA terms. We consider (A, B) as distinct element sets, except if A represents a sink node. In such a case, B cannot represent the incoming node of an outgoing edge from A and respectively A may either require to apply some function to itself (e.g. update a previous value, which will inevitably cause a cycle in the graph) or read and update its value directly from a source (i.e. URI, query interface). Moreover, we assume that an SLA data operation is accompanied by a condition that is reflected as an edge property in the graph and that specifies the operation time validity, the obliged party for the operation execution and the required data sources.

For example, in the skeleton graph of Figure 3 a "SLA parameter" and a "CompositeMetric" represent a pair of SLA nodes, where the "SLA parameter" requires the value of the "CompositeMetric" in order to compute its own value. Similarly, the "CompositeMetric" requires the "ResourceMetric" value. Yet, the "ResourceMetric" represents a sink node in the graph, thus data operations with respect to the retrieval and update of its value, may either include a cycle (in case an iterative function is used) or a process may need to read the metric's value from an indicated resource endpoint.

The skeleton digraph is implemented as a Python 2.7 module using the NetworkX [7] scientific library. The SLA skeleton graph is defined as a DiGraph object. The "Obligations", "SignatoryParty" and "ServiceDescription" nodes are defined as subgraphs of the SLA skeleton digraph. The Python module reads lists of ordered service description details as input and returns SLA service description trees that follow the set hierarchy of Table II. SLO and action guarantee descriptions are defined as nodes of the obligations subgraph. Their dependencies on service description nodes are represented through edge weights. Generally, SLA component dependencies are reflected via weights that are assigned to edges according to data content and operation criticality. The latter is subject to the SLA instance, service execution time and SLA application criteria.

For example in the initial Python module implementation, edge weighting is driven by the update frequency rate of SLA terms (e.g. measurement of composite, resource metrics). Moreover, node and edge properties signify specific SLA conditions (qualitative or quantitative), which in turn affect the content value of other SLA components or the overall information flow of an SLA graph instance. For example, given available service provisioning options in an SLA template, a customer can select desired parameters to derive a well-suited service offer. The digraph model eases the automated identification of available SLA term combinations as informa-

tion nodes and their properties can be retrieved by following directed paths. For example, a graph traversal can be designed to return service offer combinations that satisfy customer requirements, while preserving the provider's capacity and availability bounds.

By default, every SLA is associated with a single provider-customer pair to represent the one-to-one agreement relationship and thus the "Signatory party" node has exactly two instances in the SLA digraph. The party instances are defined as customer and provider subgraphs and can include any key-value paired information relevant to each signatory. Customer and provider can be affiliated with one or more third parties that support service operations. Both signatories can be associated with the same third party, following the guidelines of [1].

The module outputs an SLA digraph that preserves the directed relationships of all subgraphs and sugraph elements. The produced digraph can be fed as a JSON dictionary into a graph-aware database or to any applicable DBMS for further processing. Alternatively, NetworkX natively supports the transformation of a graph into its adjacency matrix form. The skeleton graph can be extended to include any type of nodes or edges in any valid graph orientation, e.g. diverse service-description subgraphs with respect to specific application or business needs. The digraph representation highlights the logical flow between service dependencies and guarantees in the SLA content. Alternative graph representations will also yield reasonable SLA structures, given business domain constraints or application specific needs.

VI. Use cases and graph manipulation objectives

The SLA formalization as a property digraph provides important advantages. First of all, it allows for data flow clarity with respect to semantic richness and to hierarchical ordering of elements. The mapping of SLA information into nodes and edges enables the storage and management of data in a structured way. Graph elements can be handled separately and combined dynamically. Moreover, this formalization enables the classification of information dependencies and their automatic retrieval through regular path queries (RPQs).

The directed structure helps to view SLA processing operations as workflows that are scheduled to perform data operations over semi-structured, dynamical data-series (e.g. value measurements, updates, alert notifications). This is particularly important when information has to be processed concurrently and a workflow strategy is needed to handle the efficient execution of data tasks. The realization of the graph as directed permits the orchestration of asynchronous operations that consider dependencies between antecedent and precedent nodes or node-sets.

Our approach broadens the SLA utilization as it introduces new manipulation potentials that can generate business value in the cloud service economy. The SLA digraph model of Section V is used for our experimental study to help us evaluate the modularity, query expressiveness and clear information flow of the proposed SLA data model. By applying the model over distributed, virtual hosts and by processing SLA information with respect to the aforementioned characteristics, the efficiency of the proposed schema can be examined.

Levels of information granularity may differ within the same SLA graph as the graph may include subgraphs and traversal trees for the information assortment. SLA data can be volatile and their values may influence the values of nearby nodes. With the term **modularity** we refer to the structural separation of SLA components and to their dynamic combination through graph traversals over multiple node-sets.

Walks within a graph typically take place through regular expressions that consider both the graph structure as well as inclusive nodes and edges. According to [8], a property graph represents a multi-relational graph, where nodes and edges contain attributes that designate associated key/value properties. With respect to SLA manipulation, property graphs can prove valuable as they can encapsulate all the information required to thoroughly represent an SLA document. We use the term **query expressiveness** to denote the easyness (when compared to a human language and to similar quering techniques), along with the clarity and flexibility to define and execute query expressions over a graph. Instead of direct regular expressions, a property-graph domain specific language (DSL) can be used to interact with underlying graphs.

Due to the primary SLA usage as a measurement instrument, SLA management requires **clear** and **granular information flow** such that information can be retrieved and processed rapidly. The structured accommodation of the SLA content into nodes and edges permits their direct retrieval. Moreover, a backend DBMS that is graph-aware and thus supportive of graph operations, can enable the graph management from the application layer (e.g. through a RESTful web interface). Generally, the SLA graph manipulation may involve asynchronous computational tasks to achieve the simultaneous handling of data updates and condition evaluation. Moreover, it may require simultaneous HTTP operations with respect to client-server architectures and data-retrieval requests. Finally, it allows to study and examine alternative storage and scheduling strategies with respect to combinations of in-memory and disk data storage.

Thus, we concentrate our current experimentation over two primary use-cases, where the proposed SLA data model applies and which can help us collect knowledge on the previously mentioned graph manipulation objectives. The exercise of these scenarios allows for the evaluation of the SLA digraph as an efficient data model for SLA and service management. Both cases are positioned in the cloud computing setting. For our experimentaion we assume that information exchange is primarily processed through the HTTP layer. Moreover, we consider that computational resources and data repositories are distributed over remote hosts.

Customer aware SLAs: As we demonstrated in [4], a virtual marketplace that operates over HTTP, can use SLA facets as means of service-offer selection features. A customer can adjust desired service provisioning needs, submit their requirements to a remote server and receive back offers that

satisfy their requests. A client - server architecture enables to evaluate the SLA digraph efficiency with respect to scalability and performance throughput over HTTP. The server keeps SLA offers stored in remote repositories. On customer request, the server processes one or more queries to backend, distributed repositories and returns the result to each customer instance. The scenario involves the parallel processing of selection and filtering queries with data updates. In [4] we utilized a relational (MySQL) and a document (MongoDB) database to experiment. Currently, the proposed graph data model is deployed for the Titan database [9] and the Gremlin DSL [10] is used to express the graph data queries.

SLA matching: The SLA Python module of Section V enables the automated and customized SLA generation. In a real-business scenario, provider and customer may negotiate on the desired service levels before agreement initiation. An SLA can be dynamically formulated by combining resources and services from a set of providers. Thus, SLA templates can be generated on-demand. Given provider capacity and availability constraints that can be modeled as graph node and edge properties, as well as customer service criteria, the use-case allows for the simulation of an automated negotiation scenario between two potential signatories. Moreover, a possible candidate for the formalization and respectively computation of provider constraints is the Network Simplex algorithm [11].

Simplex represents a network optimization algorithm that applies smoothly to economic problems, e.g. price setting optimizations. In the SLA and service management context, Simplex can be applied to the digraph model through node and edge properties. For the use-case under consideration, the algorithm can be applied to examine and model the provider overall cost from satisfying customer service objectives. Customer satisfaction and quality assurance are, undoubtedly, criteria that any provider needs to consider as their levels can have consequences to the provider reputation and market survivability. The NetworkX scientific library provides an in-hand Simplex implementation, where given a digraph with weighted nodes and edges, the algorithm returns the minimum network cost flow that satisfies all node demands.

VII. Related work

To our knowledge, limited scientific work focuses entirely on SLA data management. In [12] the authors elaborate on SLA semantic graphs using an in-depth analysis of service guarantees and objectives that are derived from real service offers. The authors define an SLA as a legal contract that specifies the minimum expectations and obligations between a provider and a customer. They refer to their model as the necessary means to design database schemas for the SLA information. Their SLA semantic model is based on UML and is formed as a relationship diagram that indicates the logical information flow among principal SLA elements.

The approach in [12] can be considered complementary to ours. The authors label their identified graph relationships with semantic, unidirectional attributes to describe the primary data flow among business SLA components. Their model includes the notion of service package graphs and resembles a hypergraph of service bundles that contains transition triggers from one service package to another according to predefined conditions. The SLA model in [12] signifies business perspectives related to service provisioning. Our model proposes a modular and extensible approach to structure and manage the SLA information flow.

In [4] we prototyped and experimented with a client - server scenario, where the server simulates the role of a cloud service marketplace that uses SLA templates as service offers. In that scenario, customers submitted their service provisioning requirements over HTTP and the marketplace, returned SLA templates that matched customer requests. For our observations in [4] we deployed a MongoDB and a MySQL database to store and manage SLA templates. We measured and reported the overall time of simultaneous query execution and client - server information exchange over HTTP. Moreover, we justified that a modular SLA data schema enhances the data manipulation possibilities.

VIII. Conclusions and on-going work

The paper discussed SLA data management requirements by analyzing the SLA anatomy according to the WSLA specification [1]. SLA language components were introduced and mapped to data management attributes with respect to their operational relevance during any SLA activity. The provided SLA data analysis is a useful tool for the design and deployment of SLA data stores. Additionally, it is generic enough to include new SLA elements as well as data attributes and to adapt into diverse business settings and application criteria.

Next, we introduced our SLA digraph model that follows the WSLA language directives and has been developed as a Python 2.7 module. The programming module uses the NetworkX scientific library [7] . The module expresses core SLA components, like service description, obligations and signatory parties, as subgraphs of the primary SLA digraph. The digraph module sits on the application layer. It can be used by backend systems and diverse applications for the automatic generation of SLA templates. Most importantly, it can be easily extended to consider conditions and parameter costs in the SLA formulation.

Although NetworkX provides an excellent library for the automatic graph generation and manipulation, as it supports multiple graph algorithms, it unfortunately lacks of a native or supportive DBMS solution. Thus, as our research concentrates on SLA data management, we are currently experimenting with the proposed digraph model using the Titan [9] graph database and the Gremlin [10] query language.

Our testing is targeted towards the evaluation of the proposed digraph model with respect to the challenges that were discussed in Section VI. In particular, our current experiments compare the usage of a graph query language like Gremlin with XQuery statements over the native WSLA XML format. Furthermore, the selection of a distributed graph database as

the backend layer enables a real simulation of the first use-case in Section VI. With respect to the second summarized use-case our current plan is to re-use the NetworkX library and possibly adapt the Simplex algorithm to our SLA model needs.

Summarizing, currently the SLA usage by IT service markets is superficial and lacks automation and technical details that would help customers evaluate the service levels that they are paying for. A graph-based, multi-relational approach for the SLA formalization and operations creates new opportunities for SLA-aware, value-added services and enhances the research opportunities around service and SLA management.

ACKNOWLEDGEMENT

This work is supported by the Swiss National Science Foundation (SNSF), grant number 200021E-136316/1

REFERENCES

[1] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck, "Web Service Level Agreement (WSLA) Language Specification," IBM Corporation, 2003.

[2] A. Dan, H. Ludwig, and G. Pacifici, "Web service differentiation with service level agreements," *White Paper IBM Corporation*, 2003.

[3] A. Andrieux *et al.*, "Web Services Agreement Specification (WS-Agreement)," Open Grid Forum, (GRAAP) Working Group, 2005. [Online]. Available: http://www.ogf.org/documents/GFD.192.pdf

[4] K. Stamou, V. Kantere, and J. Morin, "SLA template filtering: a faceted approach," in *Proc. of the 4th International Conference on Cloud Computing, GRIDs, and Virtualization, IARIA CLOUD COMPUTING 2013, Valencia, Spain*, 2013.

[5] M. Bailey, "The Economics of Virtualization: Moving Toward an Application-Based Cost Model," VMware, Nov 2009.

[6] A. Keller, U. Blumenthal, and G. Kar, "Classification and Computation of Dependencies for Distributed Management," in *Proc. of the Fifth IEEE Symposium on Computers and Communications (ISCC 2000)*, ser. ISCC '00. IEEE Computer Society, 2000.

[7] A. Hagberg, D. Schult, and P. Swart, "NetworkX," http://networkx.github.io/, , accessed: March, 2013.

[8] M. Rodriguez, "Property Graph Algorithms," http://markorodriguez.com/2011/02/08/property-graph-algorithms/, accessed: July, 2013.

[9] ThinkAurelius, "Titan Distributed Graph Database," http://thinkaurelius.github.io/titan/, accessed: July, 2013.

[10] ThinkAurelius team, "Gremlin graph query language," https://github.com/tinkerpop/gremlin/wiki, accessed: July, 2013.

[11] J. B. Orlin, "A polynomial time primal network simplex algorithm for minimum cost flows," in *Proc. of the seventh annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '96, 1996, pp. 474–481.

[12] C. Ward, M. J. Buco, R. N. Chang, and L. Z. Luan, "A Generic SLA Semantic Model for the Execution Management of e-Business Outsourcing Contracts," in *Proceedings of the Third International Conference on E-Commerce and Web Technologies*, ser. EC-WEB '02. London, UK: Springer-Verlag, 2002, pp. 363–376.