

# SLA information management through dependency digraphs: the case of cloud data services

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

Michael Georgiou  
Cyprus University of Technology  
Cyprus

**Abstract**—The paper addresses the issue of Service Level Agreement (SLA) data management for the cloud computing domain. We discuss the SLA anatomy and provide an analysis of SLA data management requirements. The analysis highlights SLA data aspects considering the association of SLA terms with service management operations. Our work proposes a simple, structured way to store and manage SLA information through a digraph data model that is modular, extensible and expressive with respect to data operational dependencies. The proposed SLA digraph considers properties of SLA terms and service component dependencies. We sketch a realistic data-service provisioning scenario, where the proposed SLA digraph is applied. We illustrate the mapping of data service attributes into SLA terms and the role of edge properties in the definition of service dependencies.

## I. INTRODUCTION

The cloud computing paradigm has enabled dynamic service management that is performed by cloud providers with respect to time and cost efficiency. Cloud services are provisioned based on Service Level Agreements (SLAs), i.e. contracts between providers and customers that define agreed upon guarantees with respect to service provisioning.

This work considers SLAs as machine-readable documents that are not yet standardized for the cloud computing domain. Such documents are typically semi-structured and their content varies according to service types and diverse business environments. So far, the scientific literature lacks SLA data management approaches that can enhance the automated handling of SLAs and enable their adoption by cloud markets.

The paper proposes the SLA formalization using a digraph model that applies for both the SLA storage and relevant data operations. The discussion begins with a thorough analysis of the SLA anatomy according to the IBM Web Service Level Agreement (WSLA) specification [1]. We then introduce service dependencies and cloud service classes since both influence the SLA content and the flow of service management processes.

A first contribution of our work is the analysis of SLA data management requirements. We map generic data attributes into SLA terms according to their relevance with SLA management operations. Our study highlights the sensitivity of SLA documents with respect to data velocity, intense updates and accuracy. Moreover it signifies the need to correctly address service dependencies in the SLA content. The analysis helps to formulate a complete view on adequate data structures for SLA management over distributed resources. Accordingly, the

provided study can be used as a generic tool for the design and deployment of SLA data stores.

Following, we present our proposed SLA digraph model, which demonstrates a modular, extensible and information-flow aware schema that can be used for SLA data storage and processing. The digraph considers properties of SLA components as key-value pairs that are embedded in the node and edge definition. Dependencies between service components are handled through edge connections and highlight the clear information flow of the SLA content. The SLA digraph is implemented as a Python 2.7 module using the NetworkX [2] scientific library and provides a novel approach to SLA data management for distributed environments.

We apply the proposed SLA data model onto a realistic data-service provisioning scenario, where we signify generic properties that conform uniformly to the description of cloud data services. We demonstrate the conversion of data-service properties into SLA terms using the proposed SLA data model and instantiate a central SLA template for the data-service scenario. Due to space limitations, we present and analyze one service description subgraph of the complete SLA template, which can be found in [3]. The scenario details are simplified to fit the paper demonstration scope.

The discussion begins with the generic SLA formalization, based on the scientific literature. We analyze data management requirements with respect to the SLA content and explain the notion of service dependencies following the work of [4]. We then discuss the anatomy of the proposed SLA digraph, along with advantages and experimentation objectives for the evaluation of the proposed data schema. The paper continues with the data-service provisioning scenario, where we demonstrate the appropriateness of the digraph model for the SLA expression and data management. The paper concludes with relevant research approaches and our on-going work.

## II. SLA FORMALIZATION

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 [5]. SLAs encapsulate Quality of Service (QoS) characteristics and functional service properties. They may include a multitude of technical and business service-level objectives (SLOs) along with metrics that permit the calculation and measurement of the former. Every customer needs to agree with an 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.

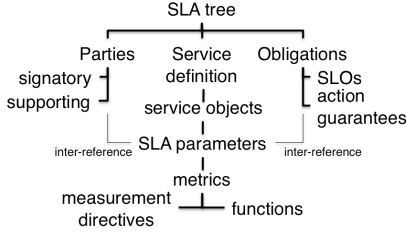


Fig. 1: Web Service Level Agreement (WSLA) structure

#### A. SLA anatomy

In contrast to other forms of contracts for IT services, properties within an SLA have to be monitored during workload execution to audit potential service-level violations or to verify adherence to agreed SLOs. The SLA formalization in a machine-readable format has been initially addressed by the IBM research on utility computing [6] and particularly by the specification of the Web Service Level Agreement (WSLA) [1] language. The Grid Resource Allocation and Agreement Protocol (GRAAP) working group also realized the need for a common language schema to express SLAs. The group introduced the Web Services Agreement (WS-Agreement) specification [7] as a language and a protocol to conduct SLAs.

Both specifications use a tree data structure to represent the SLA information. Tree branches illustrate separate SLA sections and tree leaves inner section terms. XML has been used by both schemas as the standard means to exchange information between web services. Figure 1 illustrates a high level view of the WSLA language specification, according to which the core elements of an SLA are the following:

**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 that include SLA parameters, which in turn contain properties and 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. 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.

The core elements of the language are further divided into granular sub-elements. In the case of service objects, such information reaches the level of URI sources that can be monitored and measured. In the case of service obligations, the information granularity includes the specification of business level objectives, the definition of evaluation functions and expressions that typically follow first order logic. In addition, the obligations section includes the formalization of penalties in case of service-level violations and the conditions for reimbursements and rewards that represent penalty complements.

An important attribute of both SLA language specifications

is that they permit full automation of the SLA language and manipulation of an SLA document by backend processing systems. Both specifications have been widely adopted in numerous research activities. Moreover, several efforts have defined semantic ontologies [8] to capture and categorize SLA metrics and service attributes with the help of either the Web Ontology Language (OWL) or the Resource Description Framework (RDF).

#### B. Service dependencies

According to [4] service dependencies represent customer/provider relationships that are reflected to the various cooperating components within a distributed service management system. With the term **dependency** the authors define the relationship between a dependent service or application component that requires an operation performed by an antecedent component in order for the former to execute its function. The authors use the terms **antecedent** and **dependent** to express the two counterparts of the relationship.

By default such relationship is directed. The authors in [4] introduce a multi-dimensional framework that captures the main aspects for the definition and management of service dependencies. We summarize such aspects as following:

**Space, locality, domain:** denote the distance between antecedent and dependent components in the service relationship. For example, in cloud IT services this distance can be illustrated as the total time that is required by a remote server to reach one or more backend data repositories or as the network bandwidth that is consumed by a web client to retrieve information from a shared data pool.

**Dependency criticality:** states the importance of a dependency with respect to its timely and accurate satisfaction.

**Dependency formalization:** refers to the level that a dependency is logically formalized to allow for its automatic treatment by a set of computing processes. As the authors in [4] denote, the cost of a well-formalized dependency is much lower in the overall service execution than the one of a poorly formalized dependent relationship.

**Dependency strength:** indicates the intensity of the dependent relationship between the two counterparts. For example, in a graph schema the dependency strength can be reflected through arc-weights between antecedent-dependent node pairs.

**Component type:** defines the nature of the antecedent component with respect to its activity in the service, e.g. a database, a URI, etc.

**Component activity:** the authors distinguish between active and passive antecedent components, where active components can be directly queried, whereas passive ones require the existence of an intermediary to act on behalf of them.

With respect to the **dependency-lifetime**, the authors state that the time-interval, while a dependency is valid, can be dynamic and variant between different dependencies in the same service. For example, a dependency that deals with the time-intervals of data-updates in a shared memory space may be valid for the overall service uptime, while a dependency that defines the schedule for data-migration may be valid for a limited time-period within the same service instance.

Furthermore, the authors define dependency models to analyze aspects of availability management that influence the SLA content. In particular, they highlight:

**Functional dependencies:** 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:** Detailed descriptions of components, whose interoperation represents the offered service. The structural model supplements the functional one and provides explicit descriptions of service elements (technical, configuration installations etc).

Service dependencies and their reflection into an SLA motivate our work towards the establishment of an adequate SLA data model that accommodates service dependencies and allows for their correct and automatic processing.

### C. Service classes

According to the service classification of the cloud-computing paradigm [9], a service can be classified as commoditized, public, private or hybrid. The service class typically illustrates primary service characteristics that may determine quality aspects as perceived by customers. Thus the service class indicates the pricing model that is followed by the service provider.

Commodity services (e.g. Google: mail, docs) are provided either for free or for a very low cost to numerous customers and cover basic functionality with limited QoS attributes. Public services (e.g. Amazon WS) are typically offered for an average price and include basic quality of service (QoS) guarantees. Private services (e.g. Rackspace Private Cloud [10]) are offered for higher prices and are tailored to the needs of specific customer groups with special, made-to-order QoS characteristics. Hybrid service offers bundle together service packages from all aforementioned classes.

During the SLA template formalization, a provider may include alternative options of service provisioning that indicate supported classes and available cost configurations. Moreover, service classes can be determined by the volume and type of dependencies that are defined in a service configuration. The representation and management of service classes within an SLA can be perceived as the interoperation outcome of multiple processes that belong to availability, resource and service management.

## III. DATA MANAGEMENT REQUIREMENTS

We use the term **SLA data management** to enclose 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 (e.g. [1], [7]) 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 occurred.

Accordingly, the measurement of data values has to be depicted in the SLA content during 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.

Since SLAs are not yet standardized and their content is semi-structured, a multitude of SLA terms may have similar management needs or may define additional operations, which can be determined by specific application criteria and desired functionality. Given that SLA documents come in a machine-readable format, we assume that service management operations are processed automatically by backend computing mechanisms.

We collected generic, data-operational attributes and mapped their significance to core SLA components and operations that are typically addressed during the service and SLA management cycle. Table I illustrates this mapping. The first column of Table I lists 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 relevance of data attributes with respect to indicative SLA management operations.

**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 (kb) range. Moreover, SLA data operations (e.g. customer or provider alerts) can occupy temporal storage. On the other hand, assuming the exploitation of SLAs by public marketplaces, there can be a massive volume of interactive client requests over stored SLA data [11]. A flexible data structure and database schema should allow for alternative persistence modes that efficiently address dynamic SLA data needs and permit the 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 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

Data characteristics	SLA parameter	CMetric/ RMetric	SLO	Action Guarantee	SLA	SLA mgt operations
accessibility, integrity	*	*	*	*	*	measurement, auditing, validation
velocity rate	high	high	high	low	na	real-time monitoring, value computation
replication, staging	*	*			*	real-time guarantee/value evaluation
service dependencies	*		*	*	*	guarantee/value evaluation, alerts
cleanness	*	*	*	*	*	auditing, guarantee/value evaluation
accuracy	*	*	*	*	*	real-time monitoring, auditing, evaluation
ownership, authenticity	*	*	*	*	*	auditing, guarantee/function definition
heterogeneity	*	*	*	*	*	function definition, metric evaluation

TABLE I: Data management characteristics - SLA terms/operations mapping. CMetric: Composite Metric, RMetric: Resource Metric, SLA: complete SLA document, na: non applicable.

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. 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 or 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 description is affected by resource demand. As observed by [12] 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. 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 replicate or migrate into new data repositories where a different data format might be expected.

**Service dependencies:** The evaluation of SLA parameter values may depend on the measurement of one or more metrics. SLA component dependencies need to be reflected in the SLA data structure and equivalently 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. 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 resource measurement correctness. Data conformity helps the service provider to preserve promised quality-of-service (QoS) levels and assures customers with respect to measurement of service consumption.

**Data authenticity, ownership:** During the SLA initiation cycle as defined by the WS-Agreement specification [1], 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:** Currently, 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 but compliant SLA formats that can be processed by available 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.

#### IV. SLA DIGRAPH DATA MODEL

This section proposes the SLA formulation and management as a digraph. We first summarize immediate advantages

of the graph data structure for the representation and management of SLAs. We then use WSLA [1] as a semantic reference to construct the digraph model. We discuss attributes of the proposed data model with respect to nodes, edges and their properties. An initial version of the SLA digraph implementation is presented as a Python 2.7 module along with our current experimentation objectives for the digraph model evaluation.

#### A. Graph management advantages

A graph data structure provides immediate advantages with respect to SLA data manipulation. First of all, the SLA formalization as a directed graph enables the hierarchical ordering and mapping of SLA components into nodes and edges. Semi-structured information can thus be handled in a structured way with respect to data retrieval and management operations. Moreover, graph components can be handled separately and combined dynamically.

Information modularity is an important advantage for SLA management. SLA data can be volatile and their values may influence the values of nearby nodes. With modularity we refer to the structural separation of SLA components and to their dynamic combination through graph traversals over multiple node-sets. A modular structure and a granular, while customizable, data schema is supported by graph traversals and co-ordinated data operations over multiple node-sets.

The SLA modeling as a property digraph allows for data flow clarity, while it enhances the semantic richness of the provisioned schema. According to [13], a property graph represents a multi-relational graph, where nodes and edges contain attributes that designate associated key/value properties. Levels of information granularity may differ within the same SLA graph. For example, the graph may include subgraphs for the information assortment. An SLA graph can be extensible to allow for the integration of additional nodes or subgraphs. Thus, such modeling can be easily adjusted into any application or business requirements. Instead of handling SLA information through schemaless log files and diverse data structures, a single graph or a graph pool can be used to store and manage all relevant SLA data.

The SLA formalization as a graph also enables the classification of service information dependencies and their automatic retrieval through regular path queries (RPQs). The ordering of dependencies can be achieved with their representation as node properties, thus directly associating them to service components. Additionally, dependencies can be treated as triggers in the form of edge properties that invoke specific operations during the service and SLA lifecycle.

Furthermore, the graph data structure permits the orchestration of SLA operations as a workflow that is scheduled to process dynamic SLA data-series (e.g. value measurements, updates, alert notifications). Especially when information has to be processed concurrently, a workflow strategy can adequately organize the scheduling and successful execution of data tasks.

#### B. WSLA into a property digraph

We use the WSLA specification [1] as a semantic reference to construct the SLA data model as a directed graph. We first

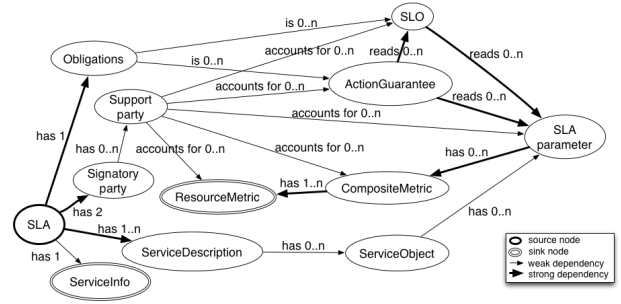


Fig. 2: SLA skeleton as a property digraph

transform primary WSLA language components into element sets by taking into account their cardinalities and relationships. The following list 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  $i \in [1, n]$ .
- $CMetric_i \supseteq RMetric_i, CMetric_j, Function_j$ , where  $CMetric$  and  $RMetric$  represent a set of composite and resource metrics,  $i \in [1, n]$  and  $j \in [0, n]$ .
- For items [a..f]  $i, j$  indicate the cardinality of element subsets and  $n \in \mathbb{N}$ .

Given that the native WSLA language is formulated as a tree, where the sequence of SLA components follows a semantic hierarchy from very generic to granular elements, we express the SLA data model as a digraph that can be perceived as a superset of the tree representation. We define the SLA data model as following:

**Definition:** An SLA is expressed as a directed graph  $SLA_g = (N, E)$ , where  $N$  represents a set of nodes  $N \supset n_i \in SLA_g$  and  $E$  a set of ordered node-pairs  $e_i = (n_i, n_{i+1})$ , where  $E \supset e_i \in SLA_g$ . Given a directed edge  $e_i$  in  $SLA_g$ ,  $n_i$  represents the predecessor node of  $n_{i+1}$ , thus denoting a direct path from  $n_i$  to  $n_{i+1}$ .

Figure 2 illustrates the skeleton graph of the proposed SLA data model. The directed structure highlights the logical flow of service and guarantee dependencies in the provisioning plan. The SLA 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 [7] that contains a similar section named "Context". The service-information node denotes SLA attributes 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.

The "SLA" node represents the source and the graph contains in total two sink nodes. The digraph nodes and edges contain properties. A node or edge property is formed as a key-value pair that is embedded in the node or edge definition. A property key signifies attributes that can be either data content or management specific. **Content properties** describe information that complete the SLA content formalization (e.g. name, type, unit). **Management properties** are graph oriented and indicate traversal patterns and operational characteristics (e.g. assignment of weights, dependencies, labeling). Properties of both types can be predefined or handled dynamically, assuming the usage of a dynamically-typed programming language.

The categorization between content and management properties allows for alternative graph traversal strategies with respect to the application needs. Moreover such ordering supports the analysis of SLA data storage and management characteristics. Alternative graph representations will also yield reasonable SLA structures, given business domain constraints or application specific needs.

In Figure 2 dependencies are classified into strong and weak ones. **Strong dependencies** are denoted with a thick edge connection, where the head of the directed edge indicates the antecedent node that is subject to frequent updates and required for the definition or measurement of the dependent node. Thus, in Figure 2 outgoing edges of dependent nodes reveal the dependent relationship with their antecedents. **Weak dependencies** indicate semantic inter-relations between nodes without interfering with their operational or business tasks. They are signified by a thin edge. The digraph representation of Figure 2 highlights the logical flow between service dependencies and guarantees in the SLA content.

We summarize and classify identified dependencies between nodes as following:

- i. Both service provider and customer may delegate part of the service operation management to external supporting parties. This relationship can be characterized as an operational dependency between signatories and third parties.
- ii. A SLA primarily consists of one or more service descriptions and applicable guarantees. This formulation can be described as an intermix of functional, structural and business dependencies that are amalgamated to formalize one or more SLA templates.
- iii. A service definition consists of service objects, which in turn accumulate SLA-parameter sets. A SLA parameter is calculated by a composite metric, which is defined by one or more resource metrics. We identify such relationships as structural dependencies.
- iv. Obligations represent guarantees in the form of either service level objectives (SLOs) or action guarantees. The definition of obligations typically depends from the definition and value range of SLA parameters. Additionally, the definition of action guarantees is often lead by the formulation and described conditions in one or more SLOs. Relationships between description and obligation objects can be described as business-oriented and technical, structural dependencies.

In the model declaration we address dependencies using weights, which are assigned to edges according to the dependency strength and dependency criticality (Section II-B).

The value criticality is SLA, time and application specific. In graph terms, it is depicted in the edge weights, which in turn signify desired traversal paths. The NetworkX library comes in hand with several graph algorithms (e.g. shortest path, breadth first search) that utilize the assigned weight values for graph traversals.

### C. Implementation and experimentation objectives

A graph approach potentially broadens the SLA utilization as it introduces new manipulation challenges that generate business value in a cloud service economy. The SLA graph representation allows for multiple uses given alternative application scenarios. Edge and node properties become of great importance as they indicate how to define the graph in terms of efficient path classification and optimal query processing. For example, vertices can be declared and ordered according to SLA operations (e.g. special vertices for SLA violations and signatory compensations). Yet, such approach also brings up challenges with respect to its evaluation and practice as an efficient SLA data schema.

The skeleton digraph is implemented as a Python 2.7 module using the Networkx [2] scientific library. The SLA skeleton is defined as a DiGraph. In the model definition, the "Obligations", "Signatory party" and "ServiceDescription" nodes are declared as subgraphs of the SLA skeleton graph. Nodes and edges in the digraph represent the SLA content and 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 service details are important for SLA management since they enable the accurate execution of the service contract. By default all nodes and edges come in hand with a unique identifier (uid) that as defined as a random integer value.

The "Obligations" node represents a single node in the skeleton digraph. We define it as a subgraph since it may include numerous service level objectives ("SLO") and action guarantees ("ActionGuarantee") that may not apply uniformly for a given SLA instance. The "ServiceDescription" node may have many instances in a given SLA graph. Each service description instance is defined as a separate subgraph of the skeleton graph and may include numerous "ServiceObjects", "SLAparameters" and metric nodes. Similarly, the "Signatory party" node has exactly two instances in the SLA graph. The party instances are defined as customer and provider subgraphs and may include any key-value paired information relevant to each signatory. Every signatory may have multiple support parties that are obliged in the service/SLA management.

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 supportive of graph operations, can enable the graph management from the application layer ( e.g. through a RESTful web interface). The SLA graph manipulation may involve asynchronous computational tasks to achieve the simultaneous handling of data updates and



conditions evaluation. Moreover, it may require simultaneous HTTP operations with respect to client-server architectures and data-retrieval requests [11].

The term query expressiveness is used to denote the easiness, clarity and flexibility to define and execute query expressions over a graph when compared to a human language and to similar querying techniques. The data quality and semantic accuracy of the SLA digraph can be justified with respect to the type of queries that are processed through a graph-supportive DBMS. In this context, the process of evaluating the query expressiveness, accuracy and presumably semantic correctness can be supported by a comparative mapping between natural language and graph query formulation. Moreover, a comparison with existing approaches for SLA manipulation (e.g. XQuery) can justify the appropriateness of the graph data structure for distributed service management.

## V. DATA SERVICE SLA TEMPLATE

We consider a data service provisioning scenario, where the provider offers deployed relational databases. Customers interact with options of available service options through a web interface that is connected to the provider's service management system. The web interface communicates to prospective customers available levels of service provisioning.

The data provider offers database and schema models that implement alternative configurations in terms of data sharing, administration and storage partitioning. During the service selection and SLA negotiation phase, customers choose the level of database isolation. The cloud infrastructure of the illustrated scenario consists of server pools that are joined through a central service management system. Each server pool consists of servers that host one or more database types according to pre-configured multi-tenancy service types. Servers are deployed to operate for the optimized execution of diverse workload tasks. In this context, optimized execution refers to the distribution of computing resources that handle the processing of active workloads. It also refers to the scaling-out strategy that each cloud pool is configured to deploy (e.g. utilizing processing power from inactive servers).

In the scenario, the cloud infrastructure implements two types of database architectures: i) **share nothing**, where active database instances and persisted data are kept on the same server; ii) **share disk**, where one or more database instances are running on each server and data are shared among all available servers in the cloud pool. In the following paragraphs, we discuss service properties, map them into SLA terms and illustrate the representation of dependencies and service parameters through the property digraph model.

### A. Data service properties

We list and summarize properties that apply homogeneously to data services. Such properties typically influence the cost, performance, security and data isolation of the provisioned service types.

The term **existing workload** denotes a running workload that is processed in the cloud infrastructure, remotely from customer local premises.

The term **complementary workload** describes workloads that have similar performance characteristics. Complementary workloads run on the same server cluster, regardless of service or database type. Mixing non-complementary workloads may lead to missed SLA objectives, outages and poor consolidation of customer-specific deployment.

The **database type** determines the level of service operations offered by the cloud provider. In the illustrated scenario we differentiate between **production** and **development** database. A production database has strict operational requirements that are related to its response time, availability targets (based on resource capacity) and security objectives. A development database has relaxed requirements with respect to service level targets.

Customer **workload requirements** represent a combination of read/write transactions that are processed by the provisioned database instance. In the scenario we assume that direct negotiation is abstracted and customers can specify applicable service level values through a web interface [11]. Customer requirements primarily deal with transaction performance, especially with respect to execution time and processing throughput (e.g. transactions per second).

The **workload duration** is typically measured in seconds or milliseconds and represents the total workload execution time. It is described by a timeframe during which a planned number of transactions is executed.

The **transaction volume** indicates the number of processed transactions per second or millisecond. It represents the accumulative transaction number (read/write) that is processed during the workload duration from all active users.

The combination of workload duration and transaction volume signifies the **transactions per second/millisecond** (tps/tpm) metric that indicates the number of transactions executed in any workload. Transactions must be processed in a predefined timeframe. Workloads with the same tps can be merged and executed on the same server cluster. The tps metric is used to define QoS parameters for every workload.

The **total workload IO** is measured in MB per second (MBps). Customer and data-service provider estimate the workload IO volume and agree on the service cost and configuration parameters. Otherwise, the workload IO value is determined by the workload IO consumption during run time.

The **average execution time** (seconds or milliseconds) of workload transactions is a property metric that is driven merely by the **workload class**. In the scenario, the provisioning infrastructure supports two workload classes: the **OLTP** that is characterized by a large number of short on-line transactions (insert, update, delete) and the **OLAP** that is characterized by relatively low volume of transactions. Queries of the OLAP class are often very complex and involve aggregations.

The **max number of concurrent users** represents the maximum number of users that are connected in the system concurrently. This configuration property is used to control user sessions and to avoid system overloads.

The **max memory per user connection** is measured in MB and signifies the maximum memory that is required in each database session.

The **max execution time** (seconds or milliseconds) of workload transactions is a property metric that is used by the provider to determine the workload peak time. If the gap between the average and max transaction execution time is small, then the workloads are complementary, otherwise they are not.

#### B. Ordering of service properties by provisioning type

The data provider employs three primary service types, whose properties represent customizable or fixed SLA terms. Available service types are categorized as following:

**Separate Database/Separate Schema (ST1)**, where customer workloads are executed on an isolated database that contains data from a single customer only. The primary characteristic of the ST1 provisioning type is the database isolation offered to tenants. Customers have full control over their leased database and their data are physically isolated from the data of other tenants. Every leased DB instance is considered as a separate server node (physical or virtual). Tenants of the ST1 type can choose the operating system platform that will host their database. An additional charge can be added to the total service price, if the selected platform includes extra software licenses e.g. Windows Server, RedHat Linux etc.

**Shared Database/Separate Schema (ST2)**, where multiple customers execute their workloads in the same database and their data are stored in different DB schemas. Database resources are shared, thus their assignment depends from customer-provider arrangements. Tenants have no control over the DB administration and configuration, but they fully manage their schemas. Customer data are isolated from the data of other tenants.

**Shared Database/Shared Schema (ST3)**, where multiple customers execute their workloads in the same database and their data are stored under the same schema. Tenants have no control over the database or the provided schema and their data are only partially isolated from data of other tenants. We classify and demonstrate in Tables II and III service properties that apply to each service configuration.

The first column of Table II lists indicative service properties of resource and system characteristics as well as database, schema characteristics and operations. Listed properties can be extended according to custom application and system criteria. The rest of table-columns represent the three available service provisioning types and their association with listed properties. In Tables II, III the (★) symbol indicates that a property is included in the description of a provisioning type.

All three provisioning types signify configurable options regarding database, schema or table operations such as automatic DB shutdown/start-up, on-demand DB/schema/table backups, etc. A customer is provided with the SLA option to determine time-schedules for data recovery, e.g. recover an accidentally dropped table, rollback a single transaction etc.

Tenants of all service types have configuration options with respect to security and availability guarantees (Table III). A customer can ensure through the SLA that all DB connections as well as schemas, tables and table columns are encrypted. Tenants may select to receive auditing and security reports in real time. Such reports specify controls with respect to

Resource/System characteristics	ST1	ST2	ST3
OS platform: Windows, Unix, Linux (32/64)	★		
CPU (family type/frequency)	★		
Processor (nr/frequency)	★		
<b>Database/Schema</b>			
DB instance/Schema nr.	★	★	
DB memory (MB)	★		
DB memory per connection (MB)		★	★
DB/Schema size (MB/GB)	★	★	
DB/Schema growing ratio	★	★	
DB/Schema administration	★	★	
<b>DB/Schema operations</b>			
DB shutdown/start-up	★		
On-demand DB/schema/table backups	★	★	★
DB/schema/table restores	★	★	★
DB/Schema/table error fixes	★	★	★
DB admin option	★		

TABLE II: Service properties I

Security	ST1	ST2	ST3
DB connection encrypted	★	★	★
Data encrypted	★	★	★
Tables/columns encrypted	★	★	★
DB/schema backup encrypted	★	★	
DBA access control	★	★	
DB/schema auditing/alert reports	★	★	★
Security profile (per schema)	★	★	
<b>Availability</b>			
Server/data (provider guarantee)	★	★	★
Stand by option DB/schema	★	★	
Back up option DB/schema	★	★	
Downtime during DB/schema migration option	★	★	
Scheduled downtime	★	★	★

TABLE III: Service properties II

when and how data and system modifications may occur. Thus customers can be informed regarding scheduled service downtimes and SLA incidents, i.e. service-level violations.

Additionally, tenants can create security profiles for each application schema that they preserve in the database. Security profiles enable tenant control over database resources and schema security options (e.g. failed login attempts, password lock time, password complexity), if such are applicable.

In the scenario, **availability** refers to the database continuous operation over a predefined time interval. Availability properties are classified into two categories: server and data availability. Server availability specifies conditions of server failures, while data availability deals with data processing malfunctions. The central service management system implements an adequate scale-out strategy to stay protected against server failures (e.g. timed-out servers are replenished by new ones that have already been active or are activated after the server failure).



A standby database can provide disaster protection and protection against data corruption. A frequent database backup on pre-agreed time intervals (e.g. daily, every 12 hours) can eliminate data losses and guarantee service continuity. Database availability signifies a series of service description and guarantee terms that the cloud provider offers to customers. Tenants are able to choose the type of DB, schema or data backup (online or offline, export dump, logical backup) and customize or at least be aware of timeframes between DB, schema and data migration.

### C. SLA template

An SLA template represents a pre-instantiated agreement that is reviewed by customers or can be in a one-to-one negotiation between a provider and a customer. We analyze the formalization of an SLA template subgraph that is constructed for the illustrated data-service scenario. The complete template can be reached at [3]. The graph size is indicative and intended for demonstration purposes only. In a real business scenario, an SLA digraph may contain thousands or millions of nodes and respectively connected edges. Our analysis concentrates on the mapping of service properties into SLA terms as well as on the representation of service dependencies through edge labels. Due to space limitations, we exclude from our description node properties, the representation of guarantees and focus exclusively on the formalization of a single description type.

We use the Python module that was described in Section 2 and Graphviz [14], an open source graph visualization software, for the SLA template construction and graph representation. In the scenario the service provider offers three distinct service types, which are mapped into service classes. The service and guarantee description in the SLA is primarily determined by the service type.

The inherent modularity of the graph structure and the ordering of SLA information into nodes and edges allow for numerous extensions (e.g. transformation into a hypergraph, automatic subgraph retrieval). For example the digraph transformation into a hypergraph (i.e. a graph generalization) allows for an explicit distinction between functional and structural SLA components. Thus, key performance indicators and service metrics may represent modular data subgraphs that are combined with structural elements (service type, class) to derive complete service offers.

Data-service characteristics are either service-type oriented or apply homogeneously to any provisioned service. We use the element-set description of Section ?? to map data-service attributes into the appropriate SLA content type. Thus data service characteristics are transformed into edge relationships and nodes that take the form of service objects, SLA parameters, composite and resource metrics.

Granular elements are decomposed into fine-grained service parameters and measurable metrics and mapped into service-description or service-object terms. For example, the three data-service provisioning types represent service-description nodes as their description encloses many of the defined service properties. There is no limit as to what and how many service-objects may describe a service-description node. Service-objects are derived from generic service properties and

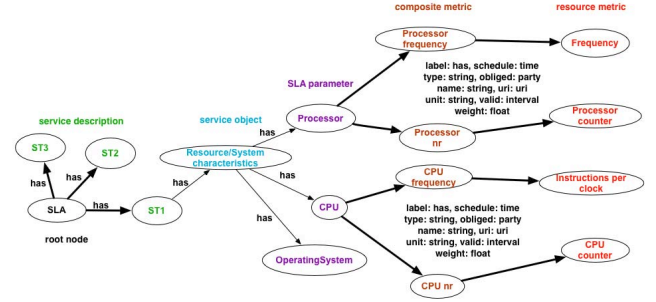


Fig. 3: Service type 1 (ST1) SLA subgraph - Data-service provisioning scenario

may enclose in their description multiple SLA parameters and composite or resource metrics.

The SLA template graph is initiated with the "SLA" as the start node. The "SLA" node is connected with three service description nodes, one for each service type. Every node and edge in the digraph is accompanied with properties that take the form of key-value pairs, which are implemented as nested dictionaries or lists in the SLA term definition. Figure 3 depicts the relationship of the service type 1 (ST1) provisioning model with functional components that describe "Resource and System characteristics".

We represent "Resource and System characteristics" as a service-object node that is connected with three SLA parameters, namely the "Operating System", "Processor" and "CPU" nodes. The "Operating System" node embeds as properties available OS options. The "Processor" and "CPU" nodes are connected with composite and resource metrics that estimate the SLA parameter values with respect to the number of available processors and processor frequency as well as CPU type and CPU efficiency. As described in Table II such components apply to the ST1 provisioning model only.

Composite metrics can use the value of resource metrics for their definition [15]. They include a function as a node property that specifies their value evaluation through a logical expression. The description of resource metrics includes as node properties the metric name, type and unit. It also includes the port or query interface, where the value of a metric is measured. Moreover, both composite and resource metrics contain additional node properties that are specified according to the metric type. By default all nodes and edges in the graph are accompanied by a unique id (uid).

Connecting edges define the measurement and evaluation schedules of composite and resource metric values. By default, the definition of edges includes their labelling name and weight according to the imposed dependency. In Figure 3 indicative edge properties of composite and resource metrics are illustrated. Additional edge properties that address metric (and more generally term) specific requirements, can be declared dynamically. The measurement or evaluation schedule of a metric value signifies operational dependencies that indicate specific timelines to be followed as well as the criticality level for the performed process. The connection between the "SupportParty" node and metrics indicates the responsibility

of the former element with respect to the measurement and update of the latter.

The connection thickness signifies the dependency strength between antecedent and dependent nodes. Both node and edge properties indicate alternative SLA traversal paths. For example, the SLA template [3] can adjust edge or node weighting that is defined by data storage operations like frequent updates (e.g. measurement of composite, resource metrics). Moreover, node and edge properties can signify special SLA conditions that enable the automated retrieval of property values from multiple nodes. Accordingly, the design and implementation of the SLA graph can be customized to meet the application criteria and service as well as business environment needs.

## VI. RELATED WORK

In [16] 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 a 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.

We consider the approach in [16] 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 hyper-graph of service bundles that contains transition triggers from one service package to another according to predefined conditions. The SLA model in [16] signifies business perspectives related to service provisioning. Our model proposes a modular and extensible approach to structure and manage data of the SLA information flow.

## VII. CONCLUSIONS AND ONGOING WORK

The paper presented a graph data model for the SLA data management of cloud services. The WSLA specification [1] was used as reference to transform the SLA structure into a property digraph. The digraph structure enables the modular handling of enclosed information, where nodes and edges represent various SLA terms that can be handled separately. The SLA digraph allows for clear information flow. For the completeness of SLA formalization, the illustrated data model considers dependencies between service components as well as content and management properties that are embedded in the SLA digraph definition.

The SLA digraph model of Section 2 is currently used for our experimental study to help us evaluate the modularity, query expressiveness and clear information flow of our proposition. The efficiency of the digraph model is evaluated through its applicability over distributed, virtual hosts, where SLA information is processed dynamically and with respect to identified service dependencies.

Walks within a graph typically take place through regular expressions that consider the graph structure along with inclusive nodes and edges. With respect to SLA manipulation,

property graphs can prove valuable as they encapsulate all the information required to thoroughly represent an SLA document. Instead of direct regular expressions, a property-graph domain specific language (DSL) can be used to interact with underlying graphs. At the time of this writing, the NetworkX library lacks of a backend persistence layer to allow for graph data management operations. The SLA digraph module is currently rebuilt to communicate with the Titan [17] graph database. Accordingly, the Gremlin [18] DSL is used for graph query expressions.

## ACKNOWLEDGMENT

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

## REFERENCES

- [1] H. Ludwig *et al.*, "Web Service Level Agreement (WSLA) Language Specification," IBM Corporation, 2003.
- [2] A. Hagberg, D. Schult, and P. Swart, "NetworkX," <http://networkx.github.io/>, accessed: March, 2013.
- [3] K. Stamou, "SLA digraph, complete RDBaaS template," <http://thinkcloud.unige.ch/SLAgraph>, created: April, 2013.
- [4] 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.
- [5] A. Dan, H. Ludwig, and G. Pacifici, "Web service differentiation with service level agreements," *White Paper IBM Corporation*, 2003.
- [6] M. Buco *et al.*, "Utility computing SLA management based upon business objectives," *IBM Systems Journal*, no. 1, pp. 159–178, 2004.
- [7] 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>
- [8] C. M. Ribeiro, N. S. Rosa, and P. R. Cunha, "An Ontological Approach for Personalized Services," in *Proc. of the 20th International Conference on Advanced Information Networking and Applications, IEEE Computer Society*, vol. 2, 2006, pp. 729–733.
- [9] "SP NIST 500-292 Cloud Computing Reference Architecture," <http://www.nist.gov>, National Institute of Standards and Technology (NIST), Sep 2011, accessed: October, 2011.
- [10] OpenStack, "Virtual Server powered by OpenStack," <http://www.rackspace.com/cloud/private/>, accessed: April, 2013.
- [11] 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.
- [12] M. Bailey, "The Economics of Virtualization: Moving Toward an Application-Based Cost Model," VMware, Nov 2009.
- [13] M. Rodriguez, "Property Graph Algorithms," <http://markorodriguez.com/2011/02/08/property-graph-algorithms/>, accessed: July, 2013.
- [14] Graphviz, "Graph Visualization Software," <http://www.graphviz.org/>, accessed: March, 2013.
- [15] A. Keller and H. Ludwig, "Defining and Monitoring Service-Level Agreements for Dynamic e-Business," in *Proc. of the 16th USENIX conference on System administration*, ser. LISA '02. Berkeley, CA, USA: USENIX Association, 2002, pp. 189–204.
- [16] C. Ward *et al.*, "A Generic SLA Semantic Model for the Execution Management of e-Business Outsourcing Contracts," in *Proc. of the Third International Conference on E-Commerce and Web Technologies*, ser. EC-WEB '02. Springer-Verlag, 2002, pp. 363–376.
- [17] ThinkAurelius, "Titan Distributed Graph Database," <http://thinkaurelius.github.io/titan/>, accessed: July, 2013.
- [18] M. A. Rodriguez *et al.*, "Gremlin graph traversal language," <https://github.com/tinkerpop/gremlin/wiki>, accessed: February, 2013.