TECHNISCHE UNIVERSITÄT DRESDEN

Institut für Systemarchitektur
Fakultät Informatik

Automation of The SLA Life Cycle in Cloud Computing

A Thesis Submitted in Partial Fulfilment of
the Requirements for the Award of the Degree of

Ph.D. (Dr.-Ing.)

from

TECHNISCHE UNIVERSITÄT DRESDEN

by

Waheed Aslam Ghumman *M.Sc.*
Born on 23.01.1981 in Kasur, Pakistan

Scientific Advisers:

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill	TU Dresden, Germany
Prof. Dr.-Ing. habil. Martin Wollschlaeger	TU Dresden, Germany
Prof. Dr. Jörg Lässig	Hochschule Zittau/Görlitz

Dresden, December 06, 2016

DECLARATIONS

1. I, Waheed Aslam Ghumman, hereby assure that I created the present work without inadmissible help of third parties and without use of other auxiliary means than those specified; thoughts that are directly or indirectly taken over from other sources are marked as such.

2. At the selection and evaluation of the material, as well as at the preparation of the manuscript, I did not receive supporting services from any other person or third party except those specified clearly.

3. Additional persons have not been involved at the mental creation of the presented work. In particular, I did not make use of the assistance of a commercial dissertation consultant. Third-parties did neither directly nor indirectly receive benefits of pecuniary value for works that are related to the content of the present dissertation.

4. Up to now, neither in Germany nor in any other country the work has been presented in this or a similar form to another examination agency, and it has not yet been published either except the one mentioned clearly.

5. I confirm that I acknowledge the applicable doctorate regulations of the Fakultät Informatik, Technische Universität Dresden.

Dresden, December 06, 2016 Signature (Waheed Aslam Ghumman)

Dedicated to

my father, mother, sisters, brother, wife and to my son Rafey.

Acknowledgements

First of all, I would like to express my special gratitude and thank to my supervisor Prof. Alexander Schill for continuously motivating me to complete this thesis. His timely guidance, expression of belief in my work and optimistic behavior during our meetings has played a vital role to accomplish this work. I would like to thank Prof. Jörg Lässig for giving me the opportunity of carrying out initial research under his supervision. I would like to thank all of my colleagues at Hochschule Zittau/Görlitz especially Abhishek Awasthi, Markus Ullrish, Nico Dittmann, Andreas Schulz, Jens Heider and Daniel Tasche for their suggestions and for creating a friendly working environment. I would also like to thank Mrs. Romy Jarschel, Dr. Marius Feldmann and Dr. Tenshi Hara for their help and support related to administrative tasks and for making arrangements of research trips.
A special thank to my parents for their hard work throughout their life to support me, to educate me and to sacrifice their leisures for my bright future. Words are not sufficient to express their kindness, love and guidance. I would like to greatly thank my beloved wife Mrs. Qurat Ul Ain Waheed for her selfless support, motivation and love which has been a great influence on my professional work. Finally, I feel truly blessed to have my son Muhammad Rafey Waheed Ghumman in my life who is always a source of amusement and a motivation to work hard.

Thanks to all of you.
Waheed Aslam Ghumman

ABSTRACT

Cloud computing has become a prominent paradigm to offer on-demand services for softwares, infrastructures and platforms. Cloud services are contracted by a service level agreement (SLA) between a cloud service provider (CSP) and a cloud service user (CSU) which contains service definitions, quality of service (QoS) parameters, guarantees and obligations. Cloud service providers mostly offer SLAs in descriptive format which is not directly consumable by a machine or a system. The SLA written in natural language may impede the utility of rapid elasticity in a cloud service. Manual management of SLAs with growing usage of cloud services can be a challenging, erroneous and tedious task especially for the CSUs acquiring multiple cloud services. The necessity of automating the complete SLA life cycle (which includes SLA description in machine readable format, negotiation, monitoring and management) becomes imminent due to complex requirements for the precise measurement of QoS parameters. Current approaches toward automating the complete SLA life cycle, lack in standardization, completeness and applicability to cloud services. Automation of different phases of the SLA life cycle (e.g. negotiation, monitoring and management) is dependent on the availability of a machine readable SLA. In this work, a structural specification for the SLAs in cloud computing (S3LACC in short) is presented which is designed specifically for cloud services, covers complete SLA life cycle and conforms with the available standards. A time efficient SLA negotiation technique is accomplished (based on the S3LACC) for concurrently negotiating with multiple CSPs. After successful negotiation process, next leading task in the SLA life cycle is to monitor the cloud services for ensuring the quality of service according to the agreed SLA. A distributed monitoring approach for the cloud SLAs is presented, in this work, which is suitable for services being used at single or multiple locations. The proposed approach reduces the number of communications of SLA violations to a monitoring coordinator by eliminating the unnecessary communications. The presented work on the complete SLA life cycle automation is evaluated and validated with the help of use cases, experiments and simulations.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Cloud Computing

Cloud computing is a type of computing in which resources are provided on demand as a service over the Internet. The term 'cloud' is generally used to denote an aggregation of objects that visually appears as a single entity and that hides its internal detail. Cloud computing has been defined by many authors, i.e. [1, 2, 3] with slightly varying definitions. L. M. Vaquero *et al.* [3] give the following definition of cloud computing:

> *"Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized service level agreements (SLAs)".*

The earliest known use of term 'cloud computing' dates back to 1996 when it was used in one of the Compaq's internal document. More common initial usage of term 'cloud computing' is considered to be in 2006 when Google and Amazon started describing cloud computing as a new paradigm of accessing software, data and compute services over the web rather than on local machines or desktops. Although, most active research and development work on cloud computing had started in 2007/2008 [4, 5, 6, 7]. However, a standardized definition by the National Institute of Standards and Technology's (NIST), USA was released in 2011 [8] as in the following:

> *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*.

1.1.1 Essential Characteristics

Cloud computing can be described as having the following essential characteristics based on above definitions, recommendations by NIST [8] and common understanding among researchers, cloud providers and users:

On demand service: A cloud service should be available to a cloud service user as and when needed. The user should be independently able to provision cloud services according to its need without human intervention on cloud service provider side.

Broad network access: Cloud computing resources should be accessible to heterogeneous platforms (e.g. mobile phones, laptop, workstation or tablets) over the standard network protocols.

Resource pooling: A cloud service should be able to securely serve multiple cloud service users by using same physical resource and virtually separating the resource on logical level for each user.

Rapid elacticity: The resources in cloud service should be easily scalable rapidly as needed, i.e. cloud service capabilities should be elastic with respect to user's needs to ensure efficient usage of cloud resources.

Measured service: Cloud service should be transparently measurable and billable, or in more general terms pay-as-you-go.

1.1.2 Service Models

Cloud computing services are generally categorized into the following three service models:
Infrastructure as a Service (IaaS): In this most basic cloud service model, computing infrastructure is provided as a service over the Internet, i.e. datacenter infrastructures, storage, virtualized or dedicated servers, network services (e.g. firewalls) or compute servers. So, rather than purchasing costly hardware, computing infrastructure is used as a cloud service and usage payments are made as that infrastructure is used. Examples of IaaS are Amazon Elastic Compute Cloud (EC2), RackSpace, VMware, Google Cloud Storage etc.

Platform as a Service (IaaS): In this cloud service model,

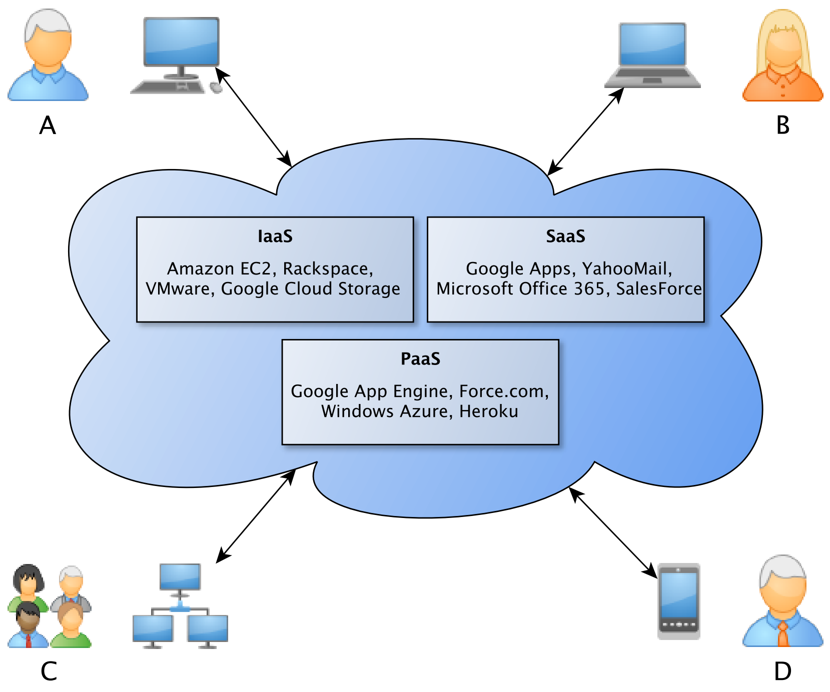

A
B
IaaS
Amazon EC2, Rackspace, VMware, Google Cloud Storage
SaaS
Google Apps, YahooMail, Microsoft Office 365, SalesForce
PaaS
Google App Engine, Force.com, Windows Azure, Heroku
C
D

Figure 1.1: Cloud computing overview

a platform is offered as a service in which cloud user may develop/program, test or deploy a program, an application or a web service. Examples of PaaS are Google App Engine, Heroku, Red Hat's OpenShift etc.
Software as a Service (SaaS): In this service model, a soft-

ware application is offered as a service through a client interface. The service user does not control or manage the underlying infrastructure, i.e. storage or operating system. Examples of SaaS are Google Apps, Salesforce, YahooMail etc.
An overview of cloud computing is shown in Figure 1.1 in which different users (A,B, C and D) may access different cloud services (IaaS, PaaS and/or SaaS) by using different devices or interfaces.

1.2 Service Level Agreements and Legal Contracts

A service level agreement (SLA) is part of a contract between a cloud service provider (CSP) and a cloud service user (CSU) which defines and describes different agreed properties of the service. These properties are usually the quality of service (QoS) parameters that guarantee desired functionality of the cloud service. P. Bianco *et al.* [9] define service level agreements as in the following:

> *"An SLA is part of the contract between the service consumer and service provider and formally defines the level of service".*

The use of service level agreements has been common in IT organizations for many years to identify support requirements for internal monitoring and external customers of IT services [9]. An overview of SLA in cloud environment is shown in Figure 1.2 in which a client gets a cloud service from a cloud service provider based on its requirements and QoS parameters for the service are described in an SLA. Service level agreement is either part of a main service contract or is completely separate document. Service contract is different from an SLA and is a legal binding between two or more parties which outlines service provided, cost, duration, resources, access rights etc. In other words, service contract describes terms of business and legal rights whereas an SLA describes the quality of service and performance measuring parameters e.g. availability percentage, throughput, response time etc.

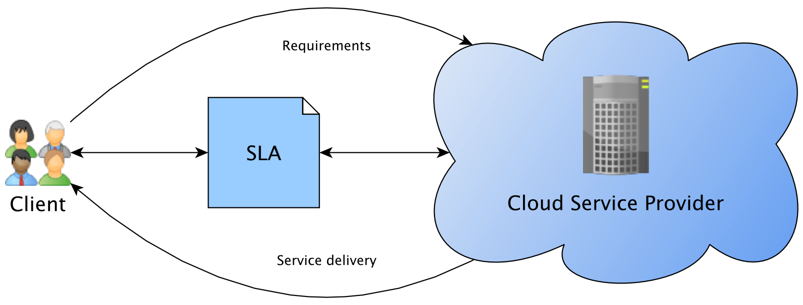


Figure 1.2: Service level agreement between a cloud service user and a cloud service provider

1.3 SLA Life Cycle

Selection and acquisition of different cloud services is triggered by business objectives and needs, i.e. requirement to use a specific cloud service is based on definitive needs of a customer. These requirements also form a basis to define different properties of the desired cloud service or in other words, QoS parameters. These requirements for QoS parameters can also be described as requirements of final SLA for the cloud service. Selection of cloud service provider depends on desired service, its properties (QoS parameters) and budget constraints etc. The selection process may include a negotiation between a cloud service provider and a cloud service user over QoS parameters, budget constraints and other associated properties/requirements of the cloud service. After a successful negotiation process, an SLA is formed to described the agreed QoS parameters. This SLA serves as a basis to monitor real-time usage of the cloud service and also to manage the cloud service if some violation occurs with respect to the SLA. This whole process of defining requirements for cloud service (or requirements for final SLA), negotiation, monitoring, management of the SLA and re-defining/modifying the SLA based on new requirements, is described as SLA life cycle [10, 11]. An overview of SLA life cycle and its different phases is shown in Figure 1.3. SLA life cycle and its different phases are described further in later chapters.

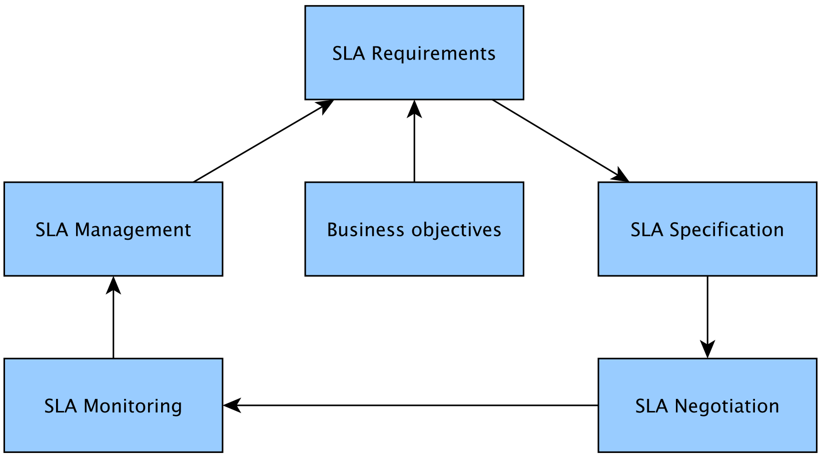


Figure 1.3: An overview of SLA life cycle

1.4 Motivation

In e-business platforms, SLAs are essentially important for the service consumer to monitor and manage the acquired cloud services. Currently, SLAs are available only in descriptive form that is not directly consumable by an external system and requires manual tasks in different phases of SLA life cycle i.e these descriptive SLAs can not be directly used in an automated negotiation algorithm for an efficient and time saving negotiation between a cloud service provider and a cloud service user. The major benefits (i.e. cost effectiveness and rapid elasticity) of using cloud services may be compromised if SLAs are not directly readable by a system. It becomes very time consuming, demanding and laborious task to manually describe the service requirements, using those descriptive requirements to compare with offered cloud services, manually negotiate with cloud service providers based on those descriptive requirements, manually monitor and manage final agreed to SLAs to ensure QoS parameters.

Example 1.4.1 *Consider a network administrator 'ADMIN' who is given a task to find an appropriate cloud based customer relation management (CRM) system for its company. ADMIN is given the requirements such as minimum availability during and outside office hours, budget, storage, number of end users and* *response time. ADMIN looks up for different options and finds out that many reasonable options exist to make a choice. For instance, Amazon Web Services Marketplace[1] alone offers 93 results for cloud based CRMs. All available cloud based CRM solutions come with descriptive SLAs, different prices, user ratings, server options, storage options etc. ADMIN has the following tasks to complete the final purchase of CRM software:*

- *Define all given requirements from its company.*
- *Manually compare different requirement against available options in market with respect to their SLAs, QoS parameters, budget and storage.*
- *Manually negotiate with different available cloud service providers to come up with an appropriate selection.*
- *Manually monitor and manage cloud service with respect to finalized SLA to detect violation, smooth running of service or to ensure other QoS parameters.*

Above example demonstrates that manually selecting, monitoring and managing cloud services with respect to their SLAs can be very time consuming and exhausting task with chances of human errors which can badly affect the output and productivity of the selected cloud services. Also, any future changes in requirements can lead to repetition of complete cycle. Moreover, cloud services may include qualitative parameters such as reliability which requires further processing to transform it to a quantifiable parameter. Many approaches and methods have been proposed and developed which either partly target these problems of manually defining, negotiating, monitoring and management of SLAs for cloud services or they present an SLA life cycle management approach that does not includes all of the requirements discussed above. So, major motivation of this work is to reduce the manual tasks and automate the complete SLA life cycle. This automation of the SLA life cycle includes definition of cloud service QoS

[1]https://aws.amazon.com/marketplace

parameters in system readable format/structure, automated negotiation of SLAs, automated monitoring and management of SLAs. Another major motivation to automate the SLA life cycle is to facilitate the process of future changes in service requirements and to automate this process with minimum manual input.

1.5 Thesis Vision, Objectives and Scope

1.5.1 Vision and Objectives

Cloud computing has been established as a popular paradigm for scalable infrastructure solutions and services. Cost efficiency, rapid elasticity, and timely availability of cloud resources are key features of cloud services making it a propitious choice against traditionally investing huge amounts of money to purchase private hardware and software resources. Currently, cloud services are largely offered with QoS parameters defined in service level agreements. But these SLAs are offered as a plain text document and every cloud provider has different contents for its SLA(s). Furthermore, the SLAs and their formats, the QoS parameters and their definitions are not standardized. These deficiencies of standardization and unavailability of a common framework for digital SLAs lead to new challenges and problems with growing cloud services oriented IT solutions. This work intends to provide an SLA structure that is based on standard SLA definitions. The single SLA structure should be flexible to specify service requirements (from service user) or service offer (from service provider) as well as same SLA structure should be used during negotiation process and afterwards during monitoring and management processes. The desired SLA structure is used as a basis for automating the complete SLA life cycle, i.e. negotiation, monitoring, management and recycling of SLAs. In addition, a web front-end is visioned to serve as a common platform for cloud service users, providers and developers for defining, monitoring and managing SLAs. This web front-end should offer different easy-to-use options to its users such that generating random SLAs (close to real world SLAs), perform multiple predefined tests for automated negotiation and monitoring scenarios to analyze the efficacy of selected negotiation/monitoring method. These analytical tests are anticipated to be conducive in decision making for selecting appropriate cloud provider(s), negotiation method and monitoring method.

1.5.2 Scope

This work on automation of the SLA life cycle includes the following research and development tasks:

- Detailed analysis of state of the art solutions for complete SLA life cycle and its different stages.
- Identification of existing problems and gaps between manual and automated SLA life cycle.
- Definition of a structure to specify requirements for SLA based on required cloud service.
- Both qualitative and quantitative QoS parameters should be processed using the proposed SLA structure.
- Automated negotiation strategy and its implementation for defined SLA structure with analytical comparison to existing negotiation approaches.
- The results of the automated negotiation process are not guaranteed to be optimized with respect to the requirements and available options.
- Usage of same SLA structure to store the finalized SLA on cloud service user side.
- Automated monitoring of QoS parameters as defined in the final SLA. Only those QoS parameters are monitored for which proper definition is available in the finalized SLA.
- Automated management of SLAs and their respective QoS parameters is limited to the scope defined in finalized SLA, i.e. all triggers, events and processes are defined directly in

SLA which are used in automated SLA management (optional part).

- Integration of finalized SLA with other systems is possible as XML data or as a web service (optional part).
- Automated recycling of SLA based on future changes is possible if recycling parameters are defined in the SLA which makes automated re-negotiation and stores finalized SLA for monitoring and management.
- Management of SLAs is limited to such tasks that are possible to be performed as an action from within a programmable function, e.g., collecting the documents required for service credits request in case of service violations or sending an electronic claim to a cloud service provider.
- A web front-end is developed to facilitate user in defining SLAs and making different tests for automated negotiation process based on randomly generated SLAs.
- The web front-end is used for storing SLAs, redefining SLAs and to perform different tests for automated monitoring also.
- Implementation for an automated monitoring is provided only for a specific cloud service e.g. Amazon S3 but presented SLA structure is extensible for other cloud services.

1.6 Summary and Thesis Structure

In this chapter, a brief introduction of cloud computing is presented with its essential properties and most common service models in cloud computing. An overview of service level agreements, legal contracts and their difference is also described briefly. More detail on different components of an SLA and examples are covered in next chapters. An introduction to the SLA life cycle and its different stages is described concisely. The motivation for automating the SLA life cycle is elaborated with an example. The scope, vision and objectives of this work are also outlined precisely

in this chapter.
The remaining thesis work is intuitively divided into different chapters based on different stages of SLA life cycle as in the following:

- Chapter 2 describes the existing approaches and solutions for automation of the overall SLA life cycle and its different stages.
- Chapter 3 defines an SLA structure and its components. All components, their intended use and related examples are included in this chapter.
- In chapter 4, an automated negotiation method for SLAs is presented, implemented and validated using multiple experiments.
- An approach for automated monitoring of SLAs based on proposed SLA structure is formulated, implemented and experimentally analyzed in chapter 5.
- In chapter 6, an implementation of the presented SLA specification and a comparison with existing solutions is given. This chapter summarizes the presented work and also briefly describes the possible future extensions.

Chapter 2

State of The Art

As discussed in Chapter 1, the SLA life cycle consists ot different stages, i.e. specification, negotiation, monitoring, management and recycling. In this chapter, state of the art approaches and techniques are discussed for these stages of the SLA life cycle. Although, many different approaches exist for each of the SLA life cycle stages, only approaches that are most relevant to this thesis work are mentioned and discussed. After describing state of the art for every SLA life cycle stage, an analytical overview is also presented in this chapter. This analytical overview discusses the drawbacks in existing approaches and also leads to requirements formalization for automation the of SLA life cycle presented in next chapters. This chapter is structured as follows. Section 2.1 motivates for the need of SLA specification languages and also discusses the most appropriate specification languages in the web services domain and the cloud services domain. Section 2.2 describes general negotiation techniques that are used in two party or multiple party setups. Section 2.3 explains different SLA negotiation model setups and existing approaches for automated SLA negotiation are also presented. Section 2.4 presents existing techniques for the automated SLA monitoring/management and the SLA life cycle as a whole. Section 2.5 summarizes the contents presented in this chapter.

2.1 SLA Specifications and Languages

Cloud service providers generally offer SLAs in descriptive/natural language format which is not directly consumable by a machine/system. The SLA written in natural language may impede the utility of rapid elasticity in cloud service. Different stages of the SLA life cycle e.g. negotiation, monitoring and management are also dependent on availability of a machine readable SLA. A cloud service user is conventionally responsible itself to monitor and enforce a natural language based SLA by first manually transforming the SLA details and guarantees into a suitable machine readable format. Different languages and specifications have been proposed to represent an SLA as a machine readable format e.g. Web SLA (WSLA) Framework [12], SLAng for defining SLAs in IT services [13][14], Web Services Agreement (WS-Agreement) specification [15], SLA* as part of SLA@SOI project [16], CSLA (Cloud Service Level Agreement) language [17] and a formal language SLAC for SLAs [18]. Among these approaches, WSLA, SLAng and WS-Agreement target specifically SLAs for web service, SLA* deals with services in general and remaining approaches are presented precisely for cloud services. In the following, these approaches are discussed briefly and verbosely in accordance with their relevance to cloud computing, expressiveness and completeness as a SLA specification language. A comparative review of SLA specification languages is also presented by Maarouf *et al.* [19] which we discuss further in later sections.

2.1.1 WSLA, Keller & Ludwig, 2003

IBM research developed the WSLA framework to define and monitor SLAs for web services. The WSLA framework includes an XML schema based language to define the SLAs and a runtime environment to interpret and monitor SLAs. Web services users and providers can define SLAs, specify SLA parameters and their measurement methods, relate the SLA parameters to the other resources and use a monitoring service to automatically enforce the SLAs. A UML (Unified Modelling Language) class diagram

[19] of meta model for the WSLA framework is depicted in Figure 2.1 and is elaborated in the following paragraphs. WSLA classi-

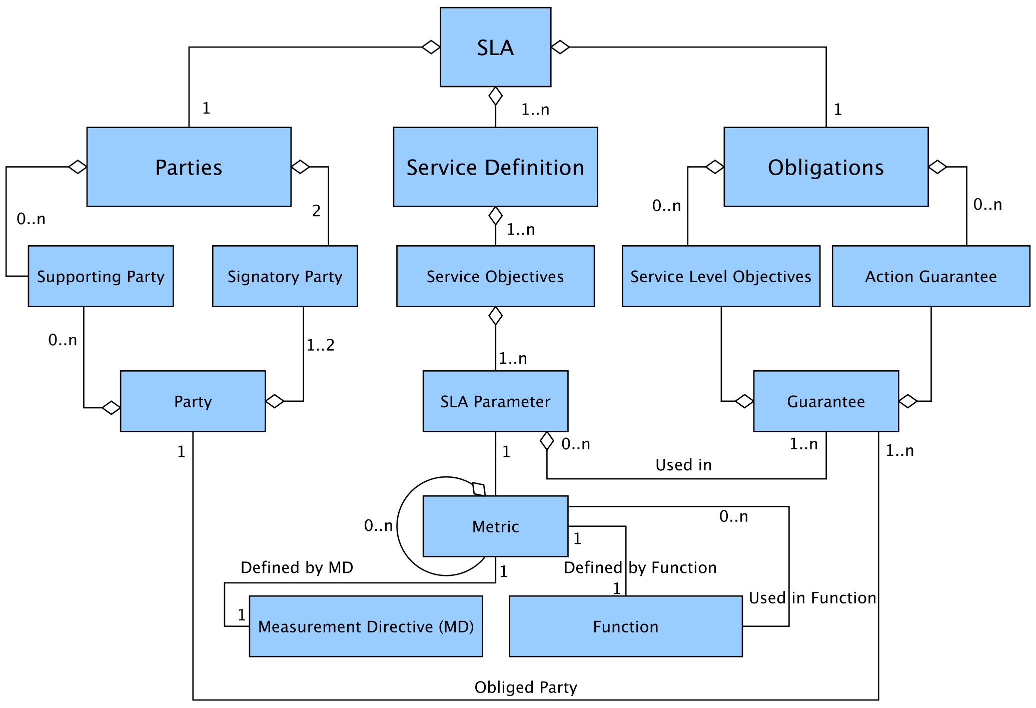


Figure 2.1: WSLA meta model

fies SLA management information into four types, i.e. (i) *resource metrics* (e.g. routers, servers and instrumented applications), (ii) *composite metrics* (e.g., by combining several resource/other composite metrics according to a specific algorithm for aggregated calculations), (iii) *SLA parameters* (to relate the composite metrics provided by a service provider to the service level objectives of service customer for evaluation, e.g. service availability, throughput or response time) and (iv) *business metrics* (to relate SLA parameters to the financial terms on service customer side). Each SLA parameter refers to one metric (resource or composite). The aggregation of metrics (to form a composite metric) is carried out either by defining a *function* (which may include other metrics as operands) or by defining a *measurement directive* (which describes the method of measurement for the metric). WSLA language specification defines the SLA structure in three sections as described below:

- **Parties** section combines information about the signatory parties, i.e. service provider and service customer. This section may also include information about supporting parties to assist, measure, monitor or manage the web service.
- **Service Description** section consists of information about the characteristics of a service together with its measurable parameters. This section defines all service parameters, their relation to the service(s), method of computation or measurement and access protocols for metrics of a managed resource. A measurement service (as part of the WSLA framework) processes the information provided in this section.
- **Obligations** section defines the guarantees and conditions under which these guarantees are valid. The WSLA framework defines two types of obligations, i.e. Service Level Objectives (SLOs) and action guarantees. SLOs define the promises to maintain the state of a service for a certain period of time. Action guarantee is a commitment of a signatory party to perform an action linked with the violation of an SLO. Likewise, obligations of the *parties* involved and the conditions (under which those obligations are valid) are defined in this section, i.e. if a threshold is assigned to an SLA parameter then it might also be required to define the constraints under which this threshold is considered valid. A condition evaluation service (as part of WSLA framework) processes the information provided in this section to assess the violations of SLOs.

2.1.2 SLAng, Lamanna *et al.*, 2003 & 2004

SLAng is an XML schema based language to define service level agreements in IT services and e-business domains. SLAng's general structure of SLAs includes responsibilities of the service provider (*Server*), responsibilities of the service user (*Client*) and their mutual responsibilities (*Mutual*). These responsibilities are expressed in three parts in an SLA: (i) end-point description (e.g. location information of service user and provider), (ii) contractual statements (e.g. contract start/end dates and duration) and (iii) Service Level Specification (SLS) parameters (i.e. QoS parameters and metrics to measure those QoS parameters). SLAng divides the SLAs in six types and two groups (*vertical* and *horizontal*) depending on their service usage type as described below:

- Vertical SLAs represent a service to provide infrastructure support to client
  - Hosting SLA is between a service provider and a host
  - Persistence SLA is between a host and a storage provider
  - Communication SLA is between a application/host and Internet service providers
- Horizontal SLAs represent a partly subcontracted service (of the same type) by the client
  - ASP SLA is between an application/service and an application service provider
  - Container SLA is between two containers (that host applications or services)
  - Networking SLA is between two networking providers

SLAng uses Object Constraint Language (OCL) and UML to represent SLAs with respect to the responsibilities (Server, Client and Mutual), i.e. constraints that are necessary to exist for successful delivery of the service and that are agreed by the parties. SLAng such that constraints should be placed only on mutually visible (to both parties) events. This ensures that monitoring of violated constraints is easily detectable by the affected party. SLAng currently supports multiple types of constraints, i.e. throughput, availability, timeliness (response time), reliability (correctness of response) and data accuracy/consistency. Further conditions about timing of a constraint to be applied (i.e. when a particular constraint is effective) or about state of service (i.e. mutually visible changes in state of the service) may also be applied to enhance the monitoring process.

2.1.3 WS-Agreement, Andrieux *et al.*, 2007

WS-Agreement is an XML based language and a web services based protocol for specifying agreement/contract between two parties, generally a service provider and a service consumer. WS-agreement is presented by Grid Resource Allocation Agreement Protocol (GRAAP) working group of the Computer Area of the Open Grid Forum (OGF). The WS-Agreement specification consists of the following three parts:

- A structure to specify an agreement (as shown in Figure 2.2)
- A structure to specify an agreement template (as shown in Figure 2.3)
- A set of port types (agreement factory, pending agreement factory, agreement, agreement Acceptance) and operations (i.e. to exchange resource information between ports or destroy resources) to manage the complete life cycle of an agreement, i.e. creation, monitoring and expiration of an agreement

WS-Agreement specification facilitates a service consumer (agreement initiator) to discover the required services. A service provider (agreement responder) publishes available agreement templates. Each agreement template describes service name, context, service description terms (functionality that will be delivered under the service), guarantee terms (assurances on service quality) and agreement creation constraints (rules that must be followed to create an agreement). An overview of service discovery, offer and agreement after acceptance under WS-Agreement specification is shown in Figure 2.4. An agreement initiator sends a request to an agreement responder to get available agreement templates (which describe the available service, its variations, quality parameters, guarantees etc). The agreement responder sends back a list of agreement templates. The agreement initiator makes an offer based on the agreement templates and sends it to agreement factory service which evaluates the offer. Result of the evaluation may be immediate acceptance or rejection depending on offer or

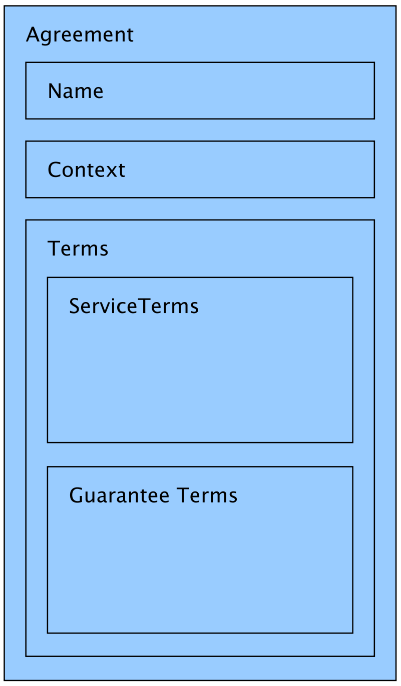


Figure 2.2: Agreement structure

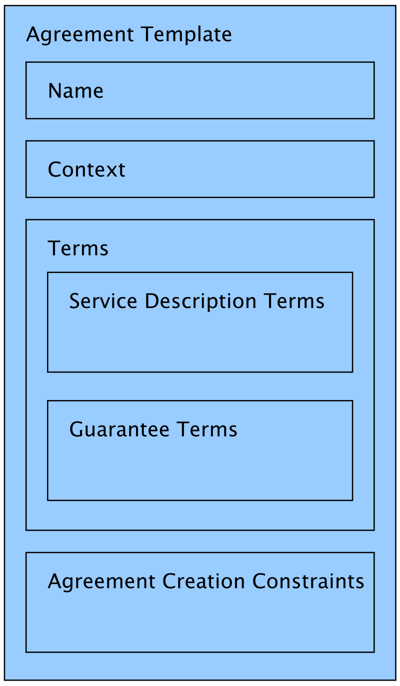


Figure 2.3: Agreement template structure

decision is deferred. In case of acceptance, the agreement responder sends a new agreement. If the decision is deferred then an instance of pending agreement is sent to the agreement initiator and once a decision has been made then agreement state is changed to either *observed* (if offer is accepted) or *rejected* (if decision is rejected). WS-Agreement specification supports only one round of negotiation, i.e. agreement initiator makes an offer which is either accepted or rejected. This deficiency is further covered by WS-Agreement Negotiation [20] which extends the WS-Agreement specification to allow negotiations and re-negotiations.

2.1.4 SLA*, Kearney *et al.*, 2010

SLA* is proposed as part of SLA@SOI European project to generalize the XML based web services standards WSLA, WS-Agreement and WSDL. SLA* is an abstract syntax to formalize SLAs and SLA templates (collectively termed as SLA(T)s) for services in general rather than only for web services. An SLA template is a structure

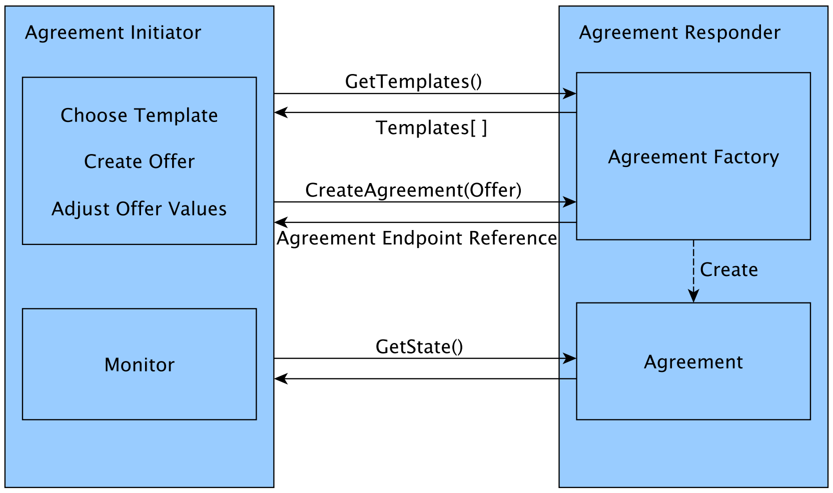


Figure 2.4: Overview of the service discovery, offer and agreement

which includes information about a service, its QoS parameters, guarantees, rules, party (i.e. a service provider) and constraints whereas an SLA is an agreement made between two parties (a service provider and a service user) based on the SLA template after negotiating over the offer by service user. SLA* is developed with an idea of language independence, better expressiveness and easy extensibility. In SLA*, contractual obligations of parties are enforced through constraints on actions of the parties. SLA(T)s structures (SLAs and SLA templates) are represented in terms of entities. An entity is a collection of *key/value* attribute pairs where each attribute represents some property of that entity. A *key* is a property/attribute name e.g. *party.role* and *value* is an unordered set of expressions representing contents of that attribute including constraints. An overview of SLA(T) structure is shown in Figure 2.5. An SLA template consists of a template ID and a version number which is used as a reference in final SLA. Interface declaration represents an obligation on one of the SLA parties to provide an interface where an interface defines a particular operation (e.g. messaging, reporting). In other words, obligations of parties to report or send a message about an event or operation is declared through interface declarations and this information is accessible to other party through SLA. Variable declarations represent values or expressions and are used for convenience purpose.

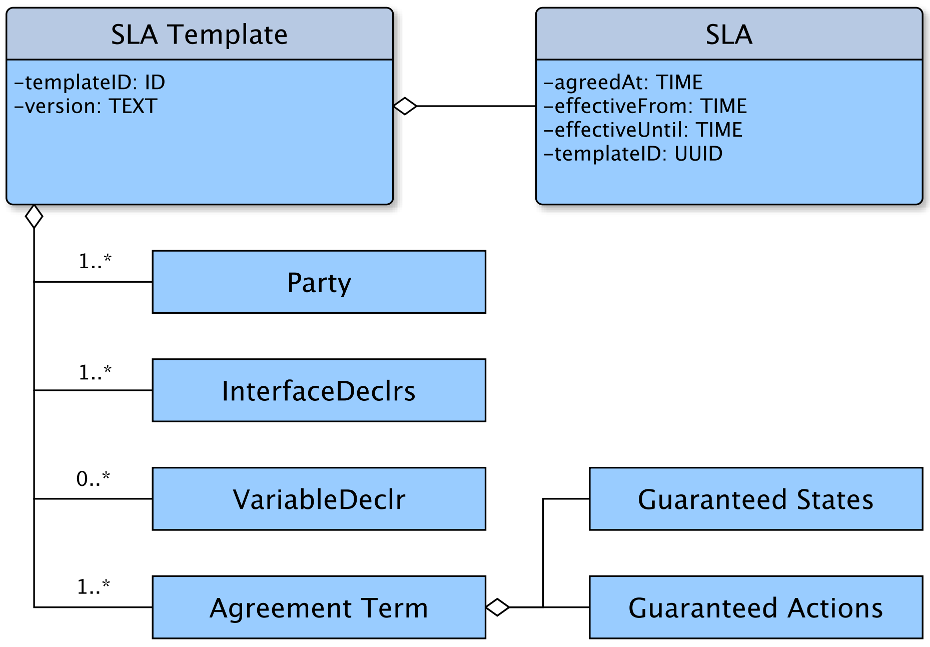


Figure 2.5: Structure of SLA and SLA Template in SLA*

In SLA*, variables declarations are also used to specify domain related *alternatives* for an *option* e.g. '> 1 and < 5' or 'in the time interval $7:00$ to $15:00$'. Apart from interface declarations (used to define functional obligations), agreement terms are used to represent non-functional (QoS) obligations. SLA* agreement terms may contain pre-conditions or constraints that must exist before an agreement term becomes applicable. An agreement term may contain multiple guarantees of one of the two kinds, i.e. states and actions. A guaranteed state is a constraint in which a party is obligated to maintain a particular state of the service e.g. 'response time $<$ 8ms'. A guaranteed action is a constraint in which a party is obligated to perform a specific action within a possible deadline if a defined pre-condition to perform the action exists. SLA* syntax is implemented as Java API, an XML schema and BackusNaur Form (BNF) Grammer.

2.1.5 CSLA, Kouki *et al.*, 2014

CSLA language is developed with main focus on dealing with QoS uncertainty and performance fluctuations in cloud services. The

CSLA is based on the Open Cloud Computing Interface (OCCI)[21] and reference architecture of cloud computing by National Institute of Standards and Technology, USA [22] [17]. CSLA defines an SLA in three sections:

- Validity of the agreement, i.e. duration of agreement
- Parties of two types, signatory parties (i.e. service provider and service customer) and supporting parties (i.e. trusted third parties)
- Agreement template contains the following five elements:
  - Cloud service definition, i.e. SaaS, PaaS or IaaS. Each service can be defined in different modes of delivery (i.e. normal mode and degraded mode) to handle the uncertainty and fluctuation in a cloud service.
  - Parameters define variables that are used in other sections of the agreement template and refer to distinct elements such as *metric*, *monitoring* and *schedule*.
  - Guarantees define the obligations of one or more parties and are comprised of four elements: *scope* (defines the services covered in the guarantee), *requirements* (define the pre-conditions that must exist to run the scoped services or to fulfill the guarantee), *terms* (each guarantee term defines one or more *objectives*/SLOs evaluated by constraints using metric, comparator and a threshold) and *penalties* (to define compensations in case of violations in guarantees). A priority is defined for each SLO to accommodate the QoS preferences of service user. The *metric* is calculated according to predefined *monitoring* in a particular *schedule* (period).
  - Billing: CSLA supports two types of billing, i.e. *pay-as-you-go* and *all-in-package*.
  - Termination defines the termination date/time for the service.

CSLA specification can be defined in different programming languages and is not XML restricted. Also, CSLA language can be used for different types of a cloud services (i.e. SaaS, IaaS, PaaS). Similar to degradation options in cloud service definition, QoS degradation definition is also possible in CSLA by defining *fuzziness* (acceptable margin around a threshold) and *confidence* (the percentage of compliance or reliability) to deal with uncertainty and unpredictability at QoS preferences level.

2.1.6 SLAC, Uriarte *et al.*, 2014

SLAC is presented as specification language particularly for cloud computing and is claimed to support main cloud deployment models. SLAC is based on WS-Agreement and inherits many features and structures from it. SLAC supports multi-party agreements including a broker, business aspects (pricing schemes) of SLAs and SLA management. A software framework[1] is also developed which uses SLAC to specify, evaluate and enforce SLAs. The formal syntax of SLAC is defined in Extended Backus Naur Form (EBNF) [23]. The semantics of SLAC is formalized as Constraint Specification Problem (CSP) which enables the verification of consistency of terms in an agreement at negotiation step and also validation of service characteristics at SLA enforcement step. SLAC defines the SLA in three parts:

- Description of the contract
- Specification of contract terms
- Guarantees for the contract terms

The *description* of an SLA consists of a unique identifier and at least two parties (service provider and service user) with *party name* (optional) and its role(s). Multiple roles for a single party enables the definition of such scenarios in which a service provider might also be a service user to some other party and an optional party name supports the definition of an SLA template in which

[1]https://code.google.com/archive/p/slac-language/

a party name is not yet mentioned. The *terms* of an agreement define the properties of the service as well as its value(s). Each SLA must contain one or more terms. A term can be a *metric* or group of terms (composed of multiple terms to define complex characteristics of a service). Each term also contains a party name (responsible to fulfill that term) and the contractors of the term (one or more) which enables the definition of parties with different roles, i.e. broker or service provider. A *metric* can be of three types:

- Numeric metric can be a value constrained by open or closed intervals or an expression. A unit is also attached to each metric e.g. milliseconds.
- Boolean metric can contain *true* or *false* values.
- List metric contains a list of values.

SLAC predefines all metrics (including list metrics along with list values) and their methods of measurements in the context of the cloud domain. *Guarantees* for the contract terms define the obligations of the responsible party (to take a defined action) in case of a particular event. An SLA defined in SLAC (using EBNF) can be checked for consistency at design time and a monitoring system uses the well defined SLA at run time to enforce and evaluate service guarantees. SLAC is extended to fulfill business aspects (i.e. pricing schemes and negotiations) by dividing SLA management into the following three phases:

- Information phase of an SLA, in which details about the service, service users, service providers and brokers is collected;
- Agreement phase, in which parties finalize contract terms through negotiation. This phase also includes finalizing the price models (i.e. flat or variable prices);
- Settlement phase, in which SLAs are enforced and evaluated through a monitoring service

SLAC elaborately defines flat and variable pricing schemes in an extended language specification to facilitate practical pricing models computing. The evaluation and experimentation of SLAC is performed mainly for IaaS scenarios, however, it is envisaged by the authors that SLAC can also be used to define SLAs for other cloud service models also, i.e. SaaS or PaaS.

2.2 Negotiation Techniques and Preliminaries

Negotiation is a process of communication between two or more parties to reach an agreement on a given issue/objective. In service computing, the most common negotiation arrangement consists of an offer from one party and acceptance (or counter offer) from another party in a cyclic pattern until an agreement is made. This arrangement may involve an adjudicator or negotiation agent to intercede the negotiation process. This negotiation model is termed as *alternating offer protocol* and is based on Rubinstein's bargaining model [24]. Rubinstein's model has the following characteristics:

- Two parties;
- Complete information is available about negotiation environment;
- Unlimited number of iterations, i.e., negotiation process continues until an agreement is made;
- Alternating offers, i.e. first party makes an offer in first round, offer can be accepted or rejected, if offer is rejected then opposite party makes a counter offer in second round, first party may accept or reject that offer from opposite party, if offer is rejected then first party makes another offer in third round (better than first round offer) and it continues until one party accepts offer from other party;
- Delays are costly (in terms of time, money or some other factor).

Negotiation is a routine process in Service Oriented Architectures (SOA) and multi-agent systems. Parties in SOA or agents may have to negotiate in different scenarios to optimize their interests or activities leading to their goals. Generally, different negotiation strategies are required to be adopted in different setups for an optimal agreement between agents or parties [25]. A *concession* is an amount that one party concedes with an intention of reaching an agreement during negotiation process. In simple setups (consisting of two parties), three common concession tactics exist [26] as described below. The concession tactics are commonly parameterized by the value β.

- **Conceder** ($\beta > 1$), in which a party makes a great concession during the start of the negotiation process. A party that follows this concession strategy soon reaches its final limit of acceptable value. Although this strategy increases the chance of a successful agreement sooner, however, it also favors the opponent party as giving great concessions at start time means to move towards a setting which is more favorable to the opponent party.

- **Linear** ($\beta = 1$) concession strategy makes an equal amount of concession in each round of negotiation process. This strategy is considered moderate as chances of reaching a common ground (win-win scenario) are high in this strategy. This strategy is, however, not very supportive in time bound setups as slow concession may increase time to reach an agreement.

- **Boulware** ($\beta < 1$) is a concession strategy in which a negotiating party maintains its initial offer value during almost all negotiation rounds until deadline is approaching and makes a great concession towards the end of the negotiation process. This strategy may result in a favorable agreement for negotiating party, however, due to no concession at all during initial rounds of negotiation process, a negotiating party may end up with no agreement due to time constraints.

Time dependent tactics represent a concession approach in which a party concedes more as deadline is approaching. Resource dependent tactics give a concession depending on the availability, demand and consumption of a resource. As resources become more scarce, the concession amount increases and vice versa. Behavior dependent tactics are applied where opponent's concession strategy and willingness to make an agreement are apparently known. The benefit of using each of these concession tactics is measured through a utility function. The utility function returns a value representing the more favorable or less favorable agreement made based on initial values of the negotiating party and the value on which an agreement is made, i.e. as negotiating party makes concessions during negotiation process, the overall utility of the possible agreement decreases.

2.3 SLA Negotiations

Cloud services are adopted as cost effective and time saving solutions for an infrastructure, software or a platform. Different quality issues may arise (related to an acquired cloud service) due to a degraded service, delay in response, packet loss, provisioning time or some other QoS parameter. These issues may affect the desired business objectives and may cause different types of losses to a party. As the cloud service market has grown rapidly, selection of an appropriate service according to preset business objectives and negotiation process becomes a time consuming and complex task. Moreover, performing these tasks (i.e. negotiation and provider selection) manually may affect other stages of an SLA life cycle and different functions in an organization linked with the acquired cloud service. Cloud services are usually automatic in their elastic functionality, i.e. a cloud service may scale up or down automatically depending on load. Similarly, acquisition of new services without manual intervention is also a mandatory requirement in most of the cases [27]. As QoS parameters of a cloud service are defined in an SLA, so it requires an automatic negotiation process for an SLA for timely and effective acquisition of a cloud service. Many approaches has been presented to automate negotiation for

SLAs [28][29][26][30][31][32][33][34][35][27][36]. We discuss only a few of these approaches in the following sections.

2.3.1 SLA Negotiation Model Setups

Cloud services are available in different variations, i.e. a provider may offer cloud services under fixed terms (no negotiation over QoS parameters and price is possible) while another provider may offer negotiable cloud services. Moreover, cloud services may also involve a service broker (a negotiator/mediator between a service user and service provider(s)) to negotiate on behalf of the user . Considering these scenarios and different requirements, a negotiation model may be one of the following types:

- **Fixed model** $(1-n)$ represents a setup in which one service user is selecting from n number of service providers offering their services on fixed terms, i.e. QoS parameters and price of the offered service are not negotiable.
- **Variable model** $(1-n)$ **with no broker** represents a setup (depicted in Figure 2.6) in which one service user negotiates directly with n number of service providers over price and QoS parameters.
- **Variable model** $(1-n)$ **with broker** is same as *variable model* $(1-n)$ with a difference that a broker or mediating party negotiates on behalf of the service user with service providers over QoS parameters and price (as depicted in Figure 2.7).
- **Variable model** $(m-n)$ represents a setup in which m number of service users are competing/bidding for different cloud services offered by n number of providers. In this setup more than one service broker may exist and all service providers offer negotiable services. Also, a service provider may be a service user of some other cloud service.
- **Mixed model** represents the most flexible model in which all of the above setups can be combined, i.e. some service

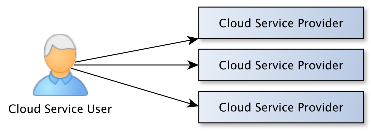


Figure 2.6: Variable model $(1-n)$ with no broker

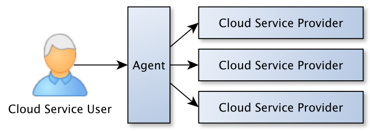


Figure 2.7: Variable model $(1-n)$ with broker

providers may offer fixed term services while others may offer negotiable prices.

2.3.2 Concurrent Negotiations in Cloud-Based Systems, Siebenhaar *et al.*, 2012

In this work [26], an SLA negotiation approach is presented for multiple cloud providers across multiple tiers, i.e. user tier (service user and broker), service tier (service provider and service user of other resource service provider) and resource tier (resource service provider). This approach enables combining services from multiple providers and negotiating with them concurrently through different entities. An overview of the negotiation architecture is shown in Figure 2.8. Each negotiating entity in this architecture

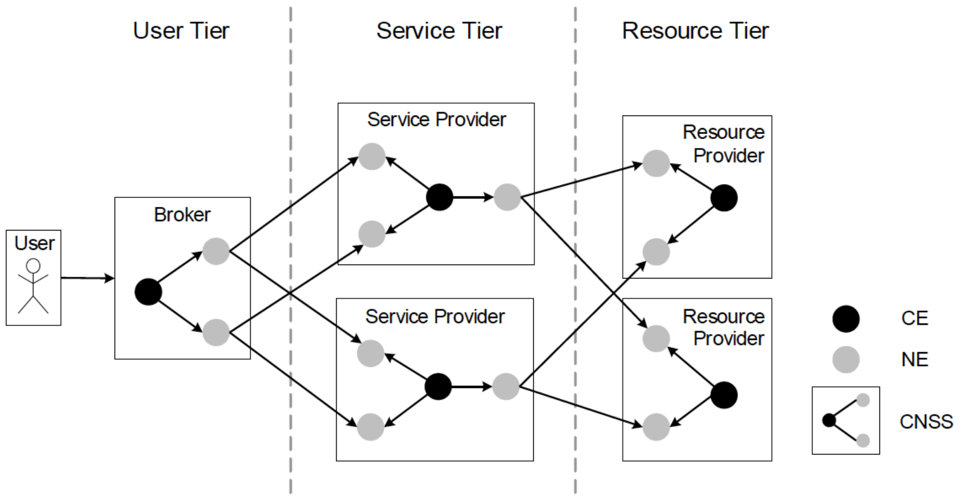


Figure 2.8: An overview of concurrent negotiation architecture

is represented in a Cloud Negotiation Support System (CNSS). A user initiates a request of a single service or combined services (depending on its functional requirements and business objectives) to the broker's CNSS. Then the broker or coordinating entity (CE) evaluates the request, selects appropriate service providers and creates the required number of negotiating entities (NE) to start the negotiation process. In the service tier, a service provider may need infrastructure services from another service provider (in resource tier) to fulfill needs of its service users (in user tier). Service provider sends its request for infrastructure services to CE which creates further NEs to negotiate with different resource providers. NEs of service/resource requester communicate with the NEs of service/resource provider, respectively. NEs in service provider are aware of available service levels and their QoS parameters which are used at the time of negotiation with NEs of user tier. Similarly, NEs of resource tier are aware of available resources and their QoS parameters. This information is used while negotiating with NEs of service tier. Each of these negotiation tasks is run concurrently by responsible NEs and the result of the concurrent negotiation process is passed to CE. The CEs of user tier and service tier evaluate the results and select most appropriate service provider and resource provider, respectively. This negotiation architecture uses linear, conceder and Boulware tactics for experiments with their effect on consumer's and provider's utility.

2.3.3 Optimal Negotiation of Service Level Agreements for Cloud-Based Services through Autonomous Agents, Yaqub *et al.*, 2014

In this work [35], a near-optimal SLA negotiation strategy is presented for cloud computing environments. The negotiation setup uses Rubinstein's alternating offer protocol [24] where two participating parties are negotiating over different issues (SLA parameters) by exchanging bids (offers and counter offers). A bid b is defined as $b = \{x_1, ..., x_N\}$ where x is chosen (offered) value of N issues. For example, if we have two issues $x_1 =$*availability* and $x_2 =$*response time* with best possible values of 99.9 and $0.4ms$, respectively then we may start our first bid with best possible (desired) values, i.e. $b = \{99.9,\ 0.4\}$. The worth of the bid b is given by a linear additive utility function $u(b)$ as:

$$u(b) = \sum_{i=1}^{N} w_i V_i(x_i)$$

where w_i is the weight (preference/priority factor for a QoS parameter) for an issue x_i such that

$$\sum_{i=1}^{N} w_i = 1$$

and $V_i(x_i) = eval(x_i)/max(eval(x_i)) \in [0,1]$ representing ratio of selected value for an issue in a bid b to the best possible value of that issue e.g. if the issue $x_1(availability)$ has value of 95 in a bid then $V_1(x_1) = 95/99.9 = 0.95$. The evaluation $(V_i(x_I))$ along with weight (w_i) for an issue (x_i) enables a service user or service provider to set its priorities with respect to service requirements. It is argued in this work that autonomous negotiation process is complex and computationally expensive. Generally, a negotiation strategy is modeled as a 3-tuple such that $S = (B,\ M_o, A)$ where B is bidding function to generate a counter offer based on opponent's offer, M_o is the opponent's assessed modes (concession strategy) based on received bid and A is the acceptance function which decides to accept opponent's bid. The computation of S based on its functions is a complex task and a computationally inexpensive negotiation strategy is proposed in this work termed as 'Reactive Exploitation' (RE). According to this negotiation strategy (RE), a party generates its bid based on a *quasi tit-for-tat* policy, i.e. a concession is given if the opponent is also observed to do the same. This negotiation strategy is outlined as in th following:

- The negotiation process is restricted by time constraints;
- A party starts negotiation process with minimal concession in first bid;
- Opponent accepts bid (which is quite unlikely due to low concession and opposite interests) or generates a counter offer

(bid);

- If a counter offer is received then from second bid onwards, first party generates its new bid based on the mean value of opponent's last *l* many bids and remaining time in negotiation process;
- First party maintains record of best counter offer received so far and latest generated bid;
- The RE strategy utilizes maximum time in most cases to exploit maximal possible options;
- An agreement is made by selecting the best acceptable bid received in the process.

RE is experimented for PaaS environment against different negotiation agents, i.e. Boulware, Linear Conceder and TheNegotiator [37]. The results of experiments show that RE performs at par with state of the art opponents.

2.3.4 An Autonomous Time-Dependent SLA Negotiation Strategy for Cloud Computing, Dastjerdi & Buyya, 2015

This negotiation framework presents a time dependent strategy (as described in Section 2.2) to automate the SLA negotiation process computing environments. This approach is motivated to help in increasing the profits of cloud service providers by testing different work loads and market scenarios under time dependent tactics. The model setup used in this approach includes a scenario in which a service user wants to find an IaaS provider with the following requirements:

- Hard disk (functional requirement and fixed)
- CPU (functional requirement and fixed)
- Memory (functional requirement and fixed)
- Cost (non-functional requirement and negotiable)

- Availability (non-functional requirement and negotiable)
- Deadline (non-functional requirement and fixed)

As cloud service user and service provider have opposite priorities, e.g. a user wants to maximize service availability and minimize cost whereas a service provider wants to get maximum profit by offering minimum services. In other words, if overall utility of SLA agreement is increasing for one party then overall utility for opposite party is decreasing. Moreover, users have time constraint (fixed deadline) with an assumption that if a service is not acquired by a particular time then service user is not able to satisfy its end users. A client negotiation service (NS) on service user's side negotiates with different service providers on behalf of the user and this NS is also able to measure the reliability of an offer from a service provider. Similarly, a NS on provider side deals with different service requests from service users. The negotiation framework in this approach works as in the following:

- A service user (client)sends its service requirements (e.g. CPU, memory, storage and other QoS parameters) to client NS.
- Client NS starts discovering appropriate service providers depending on functional and non-functional requirements submitted by client.
- Client NS starts negotiation process with NSes of the discovered providers by using time dependent tactics and makes an initial offer to provider's NS.
- Provider's NS evaluates the client's offer by considering its available resources, functional requirements and QoS parameters of requested service. If provider accepts the offer then a reliability factor is also sent to client's NS along with acceptance message, otherwise the provider generates a counter offer according to its negotiation tactics.
- Client NS may receive acceptance or counter offer. If a counter offer is received by NS then client's NS evaluates the offer by again considering deadline and other requirements.

- This process of offer and counter offer is repeated until an agreement has been made or deadline approaches.

The outcome of above negotiation process is an SLA if negotiation is successful. This approach assumes that negotiating parties are unaware of opponent's concession tactics and also investigates risk of malicious negotiation attempts under this setup, i.e. a malicious client may submit arbitrary offers to service provider to gain knowledge of provider's concession tactics and resource utilization preferences. The malicious client may use this information to acquire an actual service from the same provider faster than other clients. This scenario may cause profit loss for a provider and also deprive other clients from acquiring scarce resources in a competitive environment. On the other hand a provider may also use offer information from client's NS to maximize its profits in future transactions.

2.4 SLA Monitoring, Management and SLA Life Cycle

Monitoring of cloud services is a task of great importance for both service provider and service user [38]. Quality assurance is mandatory for cloud services to realize the underlying business objectives. SLAs contain different QoS parameters (e.g. availability, response time or throughput) as part of service contract. These QoS parameters must be observed continuously for preferred usage of cloud services according to QoS requirements. A complex SLA may contain multiple QoS parameters and guarantees by service provider. Manual and continuous monitoring of these QoS parameters and guarantees is usually not possible and it may also hinder the timely actions to rectify a problem/degradation in a service. Also, if a violation with respect to SLA occurs, then an accurate measure/action must be taken as defined in the SLA (generally within a deadline). Automatic monitoring of QoS parameters and guarantees with respect to the SLA is a much needed functionality in cloud services [39]. Multiple approaches have been presented to facilitate the automated monitoring task in cloud computing environments [40][41][42][43][44][45][46]. A detailed review of cloud service monitoring is presented by Aceto *et al.* [38] and by Hussain *et al.* [47]. In the following sections, we discuss a most relevant monitoring approach, few SLA management approaches and SLA life cycle as a whole.

2.4.1 Low Level Metrics to High Level SLAs - LoM2HiS Framework: Bridging the Gap Between Monitored Metrics and SLA Parameters in Cloud Environments, Emeakaroha *et al.*, 2010, 2012

LoM2HiS (Low-level resource Metrics to High-level SLAs) framework [40] is presented as a solution to map low level resource metrics (e.g. system uptime and downtime) to high level SLA parameters (e.g. availability) for detection of SLA violation threats. LoM2HiS is developed as part of FoSII (Foundations of Self-governing ICT Infrastructures) infrastructure. FoSII is divided in two core parts, i) enactor component (self management part of the deployed services) and ii) LoM2HiS (monitoring component which provides information to enactor component). It is assumed in this approach that negotiation process has already been completed and SLA is stored in repository for service provisioning. Future SLA violation threats are detected by defining tighter thresholds than real SLA objective thresholds. However, threat thresholds are assumed to be predefined in this work and no particular component is defined for this purpose. A step by step functioning of LoM2HiS framework is shown in Figure 2.9.

The monitoring process of LoM2HiS starts after negotiated SLA is stored in repository. In step 1, service provider stores mapping of low level metrics to SLA parameters. As service user requests the service (step 2), the runtime monitor loads related SLA from the SLA repository (step 3). Infrastructure resources are the network resources and hosts in a datacenter for cloud services. Monitoring agents in infrastructure resources measure raw metrics (of service usage) which are accessed by host monitor (step 4). These raw

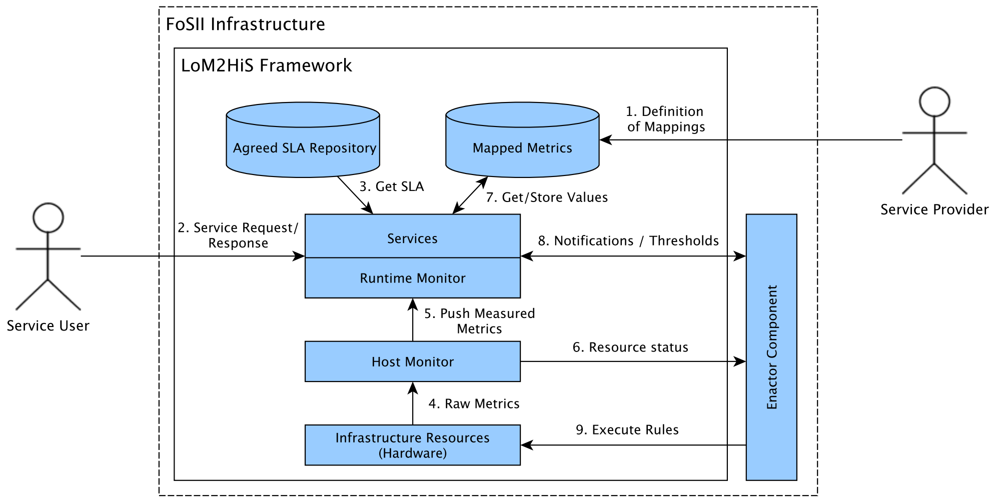


Figure 2.9: LoM2HiS Framework Architecture

metrics are analyzed by host monitor to extract metric-value pairs and periodically sends it to runtime monitor (step 5) as well as to enactor component (step 6). The runtime monitor uses an XML parser to extract the SLA parameters and their values available in the agreed SLA document. In step 7, runtime monitor gets related mappings from repository and compares them with measured metrics (received from host monitor) to build an equivalent of the agreed SLA objectives. The result of this comparison is stored back to mapped metrics repository. The runtime monitor also notifies the enactor component if future violation threats occur (step 8). The enactor component takes decision about changes in service to prevent future violation threats (step 9). As a result, service is scaled up/down or any other preventive measure is taken on run time to continue fulfilling service objectives. Different components of implemented LoM2HiS framework and FoSII framework exchange messages using Java Messaging Service (JMS) API and communication model among components is based on queuing mechanism. An extended version of this approach is also presented in 2011/2012 [42].

2.4.2 SLA Management and SLA Life Cycle in General

SLA management of cloud services includes tasks such as preparing claims in case of service violations, updating SLA parameters if requirements change or performing an action triggered due to a monitoring event. Shu *et al.* [48] present an approach for life cycle based SLA management for web services. An SLA management platform is presented in [48] to define SLAs for web services, registration of SLAs, monitoring and mapping of provider supplied parameters to service user's QoS parameters. In a most recent survey, Faniyi *et al.* [49] present an overview of SLA management for cloud services in which it is argued that cloud SLAs have still not standardized enough to be automatically deployed. It is also concluded in [49] (based on detailed analysis), majority of approaches related to SLAs have considered between one to three SLA parameters. Rak *et al.* [41] base their work for SLA monitoring on the mOSAIC API [50] (which offers development of inter-operable, portable and provider independent cloud applications). In [51], mOSAIC API is used as basis for user-centric SLA management. Maarouf *et al.* [52] present a model for the SLA life cycle management in a more recent paper where different phases of the SLA life cycle are discussed and modelled using UML (unified modeling language) diagrams. However, this work does not includes any SLA specification itself.

2.5 Summary

In this chapter, a detailed review of state of the art for different phases of cloud SLAs life cycle is presented. The SLA specification can be considered as a basis for automating the whole SLA life cycle, so different options for are discussed with reference to the web services and the cloud services. The negotiation of SLAs is an important phase of the SLA life cycle which requires cooperation of a CSP, a CSU and a third party (agent) in some cases. The negotiation process requires a common definition of the SLA parameters among the participating parties. Different SLA negotiation setups and approaches are also discussed in this chapter. Likewise the SLA monitoring phase becomes of high importance due to performance and QoS requirements. Most of the existing approaches wither concentrate on limited parts of the SLA life cycle or they lack in meeting the available standardization guidelines for the SLAs.

Chapter 3

Structural Specification of SLAs in Cloud Computing (S3LACC)

In previous chapter, an overview of state of the art in specification languages and models is described with an analysis of existing shortcomings. It is discussed in earlier chapters that automation of the SLA life cycle is dependent on transformation of a descriptive SLA to a machine readable SLA. This chapter introduces a cloud computing domain specific model to specify SLAs in a machine readable format. The proposed model is titled S3LACC (Structural Specification of SLAs in Cloud Computing). The design of S3LACC is rationalized considering the following standards/guidelines:

- Cloud computing service metrics description (draft) by National Institute of Standards and Technology (NIST), US Department of Commerce (2014)[1]
- Cloud service level agreement standardization guidelines by European Commission (2014)[2]. These guidelines are presented by Cloud Select Industry Group, Subgroup on Service Level Agreement (C-SIG-SLA) which is also working in close collaboration with Cloud Computing Working Group of International Organization for Standardization (ISO) for

[1]NIST Special publication 500-307, available online at: http://www.nist.gov/itl/cloud/upload/RATAX-CloudServiceMetricsDescription-DRAFT-20141111.pdf

[2]Available online at: https://ec.europa.eu/digital-single-market/news/cloud-service-level-agreement-standardisation-guidelines

preparing international standard for SLAs (under ISO/IEC 19086 project)[3]. The SLA standard (ISO/IEC 19086) by ISO is still under development.

3.1 Preliminaries, SLOs and Metrics

An SLA is a binding document containing explicit information about the cloud service, its QoS parameters, measurement metrics, guarantees, rules of service and any other information that is important for at least one of the users of an SLA. In this section, a basic overview of QoS parameters, metrics and preliminary definitions (important to an SLA specification) is presented.
A **Service Level Objective (SLO)** is a mean to measure the level of a cloud service [10]. An SLO can be quantitative (e.g. availability) or qualitative (e.g. reliability). A qualitative SLO has a value that is described in a descriptive form. An SLO should have the following characteristics [53]:

- Attainable
- Repeatable
- Measurable
- Understandable
- Meaningful
- Controllable
- Affordable
- Mutually acceptable

A **metric** is a method or scale to measure an SLO e.g. the SLO *availability* is generally measured with the metric *percentage*. An SLO contains a service level, metric, measurement period,

[3]ISO/IEC JTC 1/SC 38 - Cloud Computing and Distributed Platforms:
http://www.iso.org/iso/home/store/catalogue_tc/catalogue_tc_browse.htm?commid=601355

measurement type and location [54]. The QoS parameters (non-functional requirements of a service) are also referred as SLOs [12]. However, depending on an SLA, the SLO may define functional requirement of a service also e.g. *minimum required disk space.* An SLA may contain different types of SLOs, i.e. performance related SLOs (e.g. availability, response time or throughput), security related SLOs (reliability, authentication type, authentication level, session expiry time or data transfer protocol), data related SLOs (e.g. backup frequency or restore availability period).

3.2 Requirements for SLA Specification

The precise definition of an SLA is important for cloud service users and providers [54]. Likewise, an admissible SLA specification should fulfill some definitive requirements. In the following subsections, fundamental requirements for an SLA specification are discussed.

3.2.1 Composition

An SLO may contain one or multiple metrics to measure its service level. For example, the *availability* SLO might be composed of multiple metrics e.g. availability percentage, number of outages, maximum duration per outage, outage type (regional, global) or measurement period etc. Composition of different metrics for a single SLO is complex, i.e. [35] uses *performance* as a composite metric which is defined by combining *response time* and *throughput.* In most of the SLA specifications, minimum or maximum values for an SLO are linked directly in that SLO and are computed manually. However, in case of the SLOs which contain multiple metrics, it can be difficult to assign a single value (or score) to represent required values of each metric. Moreover, a single value assigned to multi-metric SLO can not effectively prioritize individual metrics in that SLO.

Example 3.2.1 *A cloud service user Bob is planning to acquire a new cloud based infrastructure service. Apart from other func-*

tional and non-functional requirements, Bob sets requirements for the SLO availability as specified in Table 3.1: Bob has discov-

Metric name	**Acceptable value**	**Desired value**
Availability percentage	95%	100%
Number of outages	10	0
Maximum duration per outage	30 mins	1 min

Table 3.1: Example of requirements for the *availability* SLO and its multiple metrics

ered few service providers, who offer almost similar and negotiable cloud based infrastructure services. Bob's priority (generally selected from the interval $[0, 1]$*) for service availability is very high due to his business objectives and is set to 0.5. For an automated negotiation process. Bob is following an optimal negotiation strategy as described in Section 2.3.3. Bob wants to combine all of the information related to availability SLO along with other service requirements in a machine readable format such that generating bids automatically during negotiation process is possible according to Bob's negotiation strategy. Representing all of this information is not a straight forward task and requires a flexible language which can accommodate multiple metrics based SLOs including their priority levels, metrics values, negotiation information and measurement period.*

An SLO may also include another SLO as part of it as discussed above with the reference to [35] or an SLA may include another sub-SLA of a supporting service.

3.2.2 Common Template for Service User and Service Provider

Mostly, specification languages for cloud SLAs target one of the entities, i.e. service users or service providers. Automation of the SLA life cycle as a whole and specifically automation of the negotiation, monitoring and management requires a common SLA template so that different parties may share specified information in a mutually understandable and agreed format. Having a common template can also eliminate different additional tasks like transformation from one format to an other or sending meta-data in different communications during negotiation or monitoring phases. This can reduce network communication load significantly in scenarios where multiple cloud services are dynamically provisioned.

3.2.3 Dependency

A cloud SLA may contain different service parameters, guarantees, rules and obligations. These components of the SLA are usually linked and one metric's value may be used as an input in computation of an other metric. For example, *availability percentage* metric is computed as follows:

$$Availability\ percentage = \frac{service\ actually\ available + planned\ downtime}{Total\ service\ time} \times 100$$

In above calculation, the parameter (*service actually available*) might not be a straight forward calculation and may include a condition e.g. service is considered unavailable only if *service response time* is greater than 10 seconds for at least 40% of requests sent during 3 minutes continuously (*time period*) and total number of requests are not higher than allowed limit (*throughput*) at any point of time during those 3 minutes. In this example, the *availability percentage* metric is dependent on other metrics including *planned down time*, *service response time*, *time period* and *throughput*. Similarly, different parts of an SLA may depend on other parts, i.e. an SLO may be dependent on an other SLO or on a metric from an other SLO, a guarantee may include observed values from different metrics, an action as a result of SLA violation may include resetting values of different SLOs.

3.2.4 Scope of The SLA Specification

The SLA specification presented in this work is targeted to define a standard model to represent an SLA for the cloud services. This specification is useful for both cloud service provider and cloud service user. A standardized SLA model (common to the both service provider and service user) is helpful to automate different states of the SLA life cycle, i.e. negotiation, monitoring and management. This work defines in detail the complete structure of the cloud services based SLA which is suitable for automation and can be used to represent all important information in an SLA especially such information which is relevant to ensure the quality of a service. However, this work does not define all possible SLOs and metrics to measure those SLOs, i.e. any well defined SLO and its metric(s) can be represented using SLA specification presented in this work but the definition of possible SLOs/metrics for different cloud services is out of scope of this work.

3.3 Proposed SLA Specification - S3LACC

The SLA life cycle starts with definition of business objectives for a targeted service. After the service requirements are finalized based on the business objectives, service discovery process starts. The discovered services are defined in SLA templates. Each negotiating party sets its negotiation priorities. The negotiation process is either finalized with an agreed SLA if successful or with no outcome if failed. In the following sections, a detailed description of the S3LACC is given.

3.3.1 S3LACC Overview

An SLA template is a document which consists of service description, obligations, QoS parameters (SLOs), metrics to measure those SLOs and guarantees. The SLA template is a common document among the participating parties and is a basis for the negotiation process. Mostly, an SLA template is a different document than the final agreed SLA. However, in S3LACC, an SLA template and final SLA are combined into a single structure. To achieve this common structure, the SLA parameters ($P_s = \{P_1, ..., P_n\}$ such that $P_i \in P_s$ and $1 \leq i \leq n$) are divided in the following three types:

- **Template parameter** represents such information which is part of an SLA template only and is denoted by a P_i^T. The template parameters are not negotiable. An SLA with template parameters is generated by a service provider based on a mutually known format.

- **Negotiation parameter** represents such parameter which contains information about automated negotiation process and is denoted by P_i^N.

- **Agreement parameter** contains the agreed value of a negotiated parameter and is denoted by P_i^F.

- **Mix parameter** contains such information which may belong to different phases of the SLA transformation in different SLAs and is donated by P_i^ρ where $\rho \in \{T,\ N,\ F\}$.

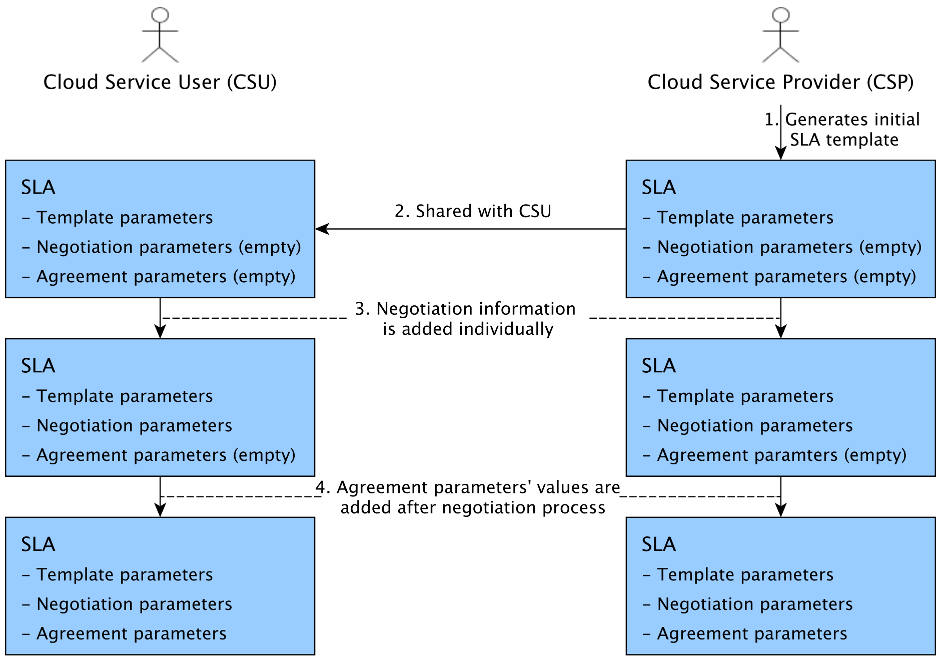

Cloud Service User (CSU)
Cloud Service Provider (CSP)
1. Generates initial SLA template
2. Shared with CSU
3. Negotiation information is added individually
4. Agreement parameters' values are added after negotiation process
SLA
– Template parameters
– Negotiation parameters (empty)
– Agreement parameters (empty)
– Negotiation parameters
– Agreement paramters (empty)
– Agreement parameters

Figure 3.1: S3LACC transformation process from SLA template to the final SLA

The categorization of SLA parameters enables specification of all information in a single SLA and is adapted in different phases of the SLA life cycle. Negotiation parameters and agreed values of the negotiated parameters are added to the SLA template by the CSP and the CSU individually. An overview of this transformation from SLA template to the final SLA is shown in Figure 3.1. An SLA in S3LACC is composed of service description, one or more SLOs, zero or more guarantees, zero or more obligations and zero or more notes (containing such explanatory information and clauses which are not related to QoS parameters). A UML representation of the relationships between the SLA and its different parts is shown in Figure 3.2. Detail of each SLA part is given in the following subsections. An SLA in S3LACC is composed of the following parts:

- Service description
- SLOs
- Guarantees
- Obligations
- Notes

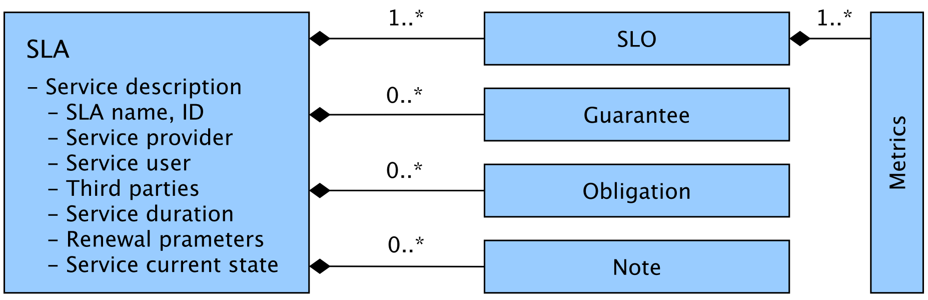


Figure 3.2: UML representation of SLA structure in S3LACC

3.4 Service Description

Service description is composed of the following parameters:

- SLA name/identifier ($SLAName^T$) is a unique name and/or a unique identifier assigned to the SLA and is mutually known to all parties.

- Service provider ($ServiceProvider^T$) represents the name of the CSP and is available in the SLA template.
- Service user ($ServiceUser^N$) represents the name of the cloud service user and contains empty value in the SLA template.
- Third parties list with roles ($ThirdParties^\rho$) is a list of parties involved in a cloud service other than the CSP and the CSU, e.g. broker or an external monitoring service provider. This list is represented as $ThirdParties^\rho = [TP_1^\rho, ..., TP_m^\rho]$ where $m \geq 0$, $\rho \in \{T,\ N,\ F\}$ and $TP_i^\rho = \langle ThirdParty\ Name_i,\ Role_i \rangle$ where $0 \leq i \leq m$. A third party information may be added as negotiation parameter TP_i^N (e.g. broker or negotiation agent) or as a final agreement parameter TP_i^F (e.g. a third party for SLA monitoring).
- Service duration ($ServiceStartDateTime^F$, $ServiceEndDateTime^F$) represents the start date and end date of the service.
- Service renewal parameters ($ServiceRenewalParameters^F$) contain information about automatic renewal of the cloud service on a preset date/time or based on a precondition. Automatic renewal may involve automatic renegotiation also. Renewal parameters are the following:
  - Precondition ($ServiceRenewalCondition^F$) represents a boolean expression that must be evaluated to *true* before the service is renewed. If the precondition is empty then service is renewed automatically on preset date/time.
  - Renegotiate on renewal ($RenegotiateOnRenewal^F$) is a boolean which represents whether a renegotiation is required on service renewal or not. *Renegotiation parameters* are defined in Section 3.5.1.
  - Service renewal date/time ($ServiceRenewalDatTime^F$) is the time-stamp on which serviced is renewed.
  - Renew service ($ResetService()^F$) is a function that resets all agreement parameters (P_i^F) to their initial values or

to renegotiated values.

- Service current state ($ServiceCurrentState^F$) contains the current state of a cloud service. It may contain one of the intuitive values {*Starting*, *Stopping*, *Stopped*, *Started*, *Terminated*}.

3.5 Service Level Objectives (SLOs)

The performance of a cloud service is characterized by defining Service Level Objectives (SLOs). As discussed in Section 3.1 and Section 3.2, an SLO may depend on another SLO, may be a qualitative SLO (*e.g. reliability*) or quantitative SLO (*e.g. response time*). It is also discussed in Section 3.2.1 that a single SLO may contain more than one metrics. As cloud services are becoming more complex due to service composition or cloud federations, an SLO may be comprised of both qualitative and quantitative information. For example, an SLO, *reliability of benchmark data* in a cloud based bioinformatics data analysis service may be described as: benchmark data should be *highly reliable* (qualitative) with *chances of loss of data* during computations being less than 0.5% (quantitative). One way to represent such an SLO can be to divide one SLO in two separate SLOs, i.e. *reliability degree* SLO (qualitative) and *chances of loss of data* (qualitative). In S3LACC, qualitative and quantitative properties of SLOs are intuitively shifted to their metrics and a single SLO may contain one or multiple metrics (qualitative, quantitative or mix of both). Functioning and further detail of quantitative and qualitative metrics are discussed in Section 3.5.1. An SLO contains the following parameters under S3LACC:

- SLO ID ($SLOID^T$) contains the unique identifier of the SLO.
- Name ($SLOName^T$) contains the SLO name.
- SLO weight ($SLOWeight^\rho$) is a value from the interval $[0, 1]$ to represent the priority of an SLO. This value is used at the

time of negotiation to generate and evaluate the negotiation bids.

- Metric list ($M_s^\rho(SLOID) = \{M_1^\rho, ..., M_l^\rho\}$ where $l \geq 1$), represents list of metrics IDs associated with the $SLOID$.
- SLO list ($SLO_s^\rho = \{SLO_1^\rho, ..., SLO_k^\rho\}$ where $k \geq 0$), is a list of SLOs used to combine one or more SLOs as sub-SLOs to meet composition requirements.

3.5.1 Metrics

A metric $M_i^T(SLOID)$ can be one of the following value types:

- Numeric
- Date/time
- Range of numeric or date/time values
- Boolean (*true* or *false* represented as 1 or 0, respectively)
- Qualitative/fuzzy

Metric Ratio Type

Definition 3.5.1 *Directly Proportional Metric: A metric is directly proportional metric if its utility value increases with its increasing value.*

An example of directly proportional metric is *availability percentage*, i.e. utility value of a service with *higher* percentage is also *higher.*

Definition 3.5.2 *Inversely Proportional Metric: A metric is inversely proportional metric if its utility value increases with its decreasing value.*

An example of inversely proportional metric is *duration per outage*, i.e. utility value of *longer* duration outage is *lower.*
Based on metric value types, a metric is one of the following two types:

A **quantitative metric** represents a metric which contains a numeric value, boolean value, date/time or range of numeric or date/-time values.
A **quantitative metric** represents a metric which contains value type in set $\{\nu\} \setminus \{qualitative/fuzzy\}$. A **qualitative metric** in S3LACC has value type of qualitative/fuzzy. All possible descriptive values of the qualitative metric are defined as a well ordered set ($X_s = \{X_1, ..., X_j\}$ where $j \geq 1$, $X_i \in X_s$ and $1 \leq i \leq j$) with respect to the utility level ($U(X_i)$) of each descriptive value (X_i) such that $U(X_i) < U(X_{i+1})$ and $U(X_i) = i/j$. Semantically, utility level of a descriptive value X_i represents its worthiness level. A *qualitative metric* in S3LACC is automatically processed by converting its descriptive values to their numeric utility levels.
Metric Parameters A metric is comprised of the following parameters:

- Metric ID ($MetricID^{\rho}$) is a unique metric identifier
- Name ($MetricName^{\rho}$)
- Unit of measurement ($MetricUnit^{\rho}$)
- Negotiation parameters
- Renegotiation parameters
- Monitoring parameters

The **negotiation parameters** of a metric are the following:

- Negotiable ($IsNegotiable^{T}$) is a boolean value which describes whether metric is part of the negotiation process or not. A *false* value is used when a metric is defined for monitoring or management purposes only.
- Mandatory ($IsMandatory^{N}$) is a boolean value which is set by the CSP and the CSU in their respective template as part of the negotiation strategy. If a *true* value is assigned to this parameter then negotiation requirements (as restricted by *desired value* and *acceptable value*) for the metric must be fulfilled otherwise the negotiation process is unsuccessful. If a

false value is assigned to this parameter then it represents that negotiation requirements are preferred to be fulfilled, however not mandatory.

- Weight ($MetricWeight^N$) is value from interval $[0, 1]$ representing the priority/importance level of the metric. A weight at SLO level and at metric level facilitates to prioritize an SLO and metrics within an SLO separately.
- Desired value ($DesiredValue^N$) represents best possible single value or range of values for the metric. Depending on negotiation policy, this value is usually the starting value in the negotiation process.
- Acceptable value ($AcceptableValue^N$) represents the reserve value (or worst possible value that is acceptable) during the negotiation process.
- Agreed value ($AgreedValue^F$) represents the final value that is agreed between the CSP and the CSU after negotiation process.
- Deadline ($Deadline^T$) is the maximum number of negotiation rounds or time limit allowed for the negotiation process for the metric. This parameter is part of the SLA template and set by the CSP. However, a CSU may set a different value if required but not exceeding the value set by the CSP.
- Concession values ($CV_s^N = \{\langle D_1,\ CV_1\rangle, ..., \langle D_q,\ CV_q\rangle\}$) is an ordered set (with respect to deadline D_i) such that $\langle D_i,\ CV_i\rangle \in CV_s^N$, CV_i is concession value and $1 \leq i \leq q$. If this set contains only one tuple ($\langle D_1,\ CV_1\rangle$) then $D_1 = Deadline^N$ which means that in every negotiation round an equal amount of concession value CV_1 is applied to generate a new bid value for the metric. If more than one tuples are present in the set CV_s then concession value CV_i is applied until deadline D_i. After D_i, the concession value CV_{i+1} is applied until deadline D_{i+1} and so on. These values can be used to preset a negotiation strategy, i.e. conceder, Boulware, linear or a custom

concession strategy can be defined by varying the values in this set.

- Negotiation strategy ($NegotiationStragety()^N$) is a function which dynamically fills the negotiation parameters in set CV_s and may depend on opponent's negotiation strategy, number of competitors, demand/supply and/or any other factor. This function is used to implement any type of automated and dynamic negotiation strategy by modifying the set CV_s on runtime.

Renegotiation parameters define a set of new values or expressions for the specified parameters of a metric for which negotiation information is to be updated for the renegotiation process. This set is defined as:
$RNP_s^F = \{RNP_1^F, ...RNP_r^F\}$ where $r \geq 0$, $RNP_i^F \in RNP_s^F$, $0 \leq i \leq r$ and $RNP_i^F = \langle RNPID, ResetParameterName, NewValueOrExpression\rangle$. RNP_s^F enables to automate the renegotiation process in case of service failure or SLA violation by linking the $RNPID$ to the violation rules defined in section 3.6.
Monitoring parameters are the following:

- Computation formula ($ComputationFormula^F$) is a well formed mathematical expression to compute the value of a metric which may include the observed/calculated values of other metrics in the same SLA, constants and/or variables containing values from the metrics of other SLA(s), web service(s), database value(s) or any other internal and/or external data source.

- Monitoring schedule (MS^F) is a set which contains different monitoring schedules at which monitoring of the metric is performed:
  $MS_s^F = \{MS_1^F, ..., MS_t^F\}$ where $t \geq 0$, $MS_i^F \in MS_s^F$, $1 \leq i \leq t$, $MS_i^F = \langle MSStartDate_i,\ MSEndDate_i,\ MSStartTime_i,\ MSEndTime_i,\ MSFreq_i,\ StoreLocation\rangle$ and $MSFreq_i \in \{ms, ss, mm, hh, dd, mm, yy\}$.
  $MSStartDate_i$, $MSEndDate_i$, $MSStartTime_i$ and

$MSEndTime_i$ are start and end dates and times, respectively, at which monitoring schedule MS_i^F of a metric starts and ends. $MSFreq_i$ is the monitoring frequency which contains one of the value from the set $\{ms, ss, mm, hh, dd, mm, yy\}$ to represent monitoring of the metric every millisecond, second, minute, hour, day, month or year, respectively. This flexible monitoring schedule technique allows to define the different monitoring schedules for different weekdays, for different months of the year or for a particular season to accommodate the dynamic requirements of cloud service monitoring. $StoreLocation$ contains the data storage location where the monitored value is stored.

3.6 Guarantees / Obligations

An SLA guarantee is an agreed commitment by a cloud service provider to maintain a certain service level. Guarantees are defined with respect to the agreed values of metrics in the SLOs. A guarantee has the following parameters:

- Guarantee ID ($GuaranteeID^\rho$) is a unique identifier.
- Guarantee precondition ($GuaranteePrecondition^\rho$) is a combination of one or more boolean expressions (containing observed/calculated value(s) of the metric(s), variable(s) and/or constant(s) joined by boolean operators (AND, OR, NOT).
- Guarantee action ($GuranteeAction()^\rho$) is a function that performs predefined tasks (e.g. automatically logging of the specific information, changing $ServiceCurrentState_F$ or preparing a service claim document).

Obligations are also defined as guarantees with similar parameters as *guarantees*, i.e. $ObligationID^\rho$, $ObligationPrecondition^\rho$ and $ObligationAction()^\rho$. Obligations are different from guarantees in such a way that obligation may not depend on observed/calculated metric values but rather may depend on external conditions e.g.

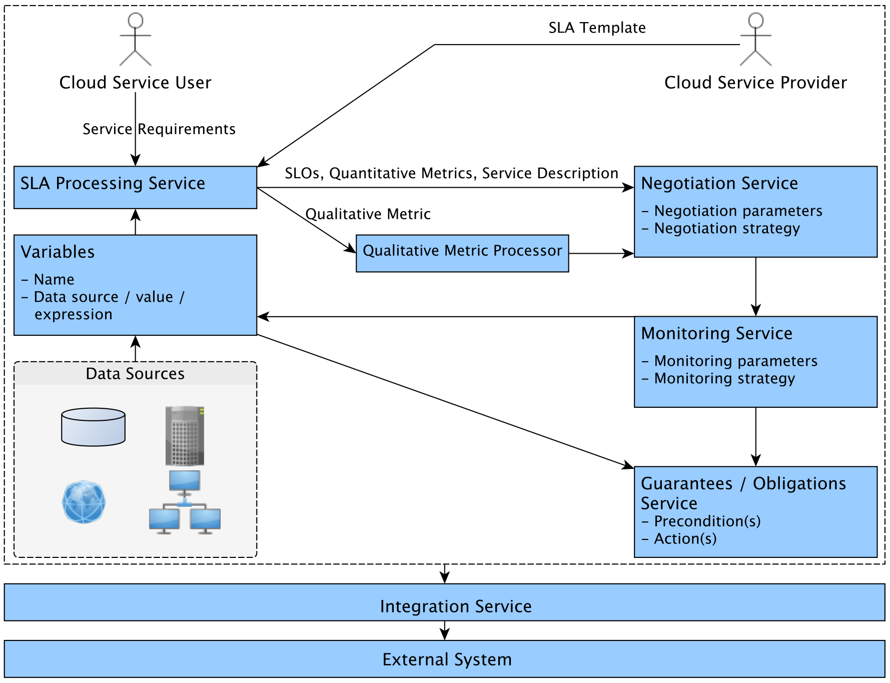


Figure 3.3: An overview S3LACC framework

a cloud service user may be obliged to inform the cloud service provider two hours in advance if further resources are required compared to what is agreed in the SLA.

3.7 S3LACC Framework

S3LACC framework briefly gives an overview of the S3LACC's usage in a cloud environment. An overview of S3LACC framework is depicted in Figure 3.3. Service requirements come from the CSU which starts a provider discovery process and may involve a broker/third party as a support service party. It is assumed that all CSPs for the same service have the similar SLA templates. The *SLA processing service* selects the shortlisted CSPs, negotiation parameters are added to the SLA template and any qualitative metrics are transformed to their quantitative utility levels by *qualitative metric processor*. A custom negotiation strategy may also be embedded in the SLA as described in above sections. The *negotiation service* may involve a broker to mediate the negotiation process. After the negotiation process, agreed values from the SLA are communicated to the *monitoring service* com-

ponent. The *monitoring service* reads real-time metric values from the specified locations on the specified time schedule. Variables are stored separately which contain up-to-date data values from different sources. Each variable contains a particular data value from a specific data source. These variable values are used as input in metric computation formula and also in condition expressions (e.g. in *GuaranteePrecondition* or in *ObligationPrecondition*). The *guarantees/obligations service* checks for service violations or obligations. An integration with external system is achieved through an integration service which transforms the SLA data to XML format.
The SLA management tasks are performed as actions in a guarantee or an obligations component. A use case in the following section illustrates SLA management tasks in guarantees' and obligations' actions.

3.8 Use Case

As a proof of concept, we transform a precise descriptive SLA of a cloud based customer relation management (CRM) service S1 (assumed) to the S3LACC based SLA. Let's consider the following scenario for the service S1:

A company ABC has its offices throughout the country and requires S1 to be used by its employees (S1 users). ABC requires that S1 should have availability from 95% *to* 100%. *S1 may have* 2 *to* 6 *outages per month with maximum duration of* 10 *minutes per outage. The S1 users should be authenticated using one of the protocols {TACACS+, RADIUS, DIAMETER, Kerberos, OpenID}, arranged in ascending order of priority. S1 users should be authenticated within* 5 *seconds after submitting the login information. The cloud service providers CSP1 and CSP2 offer S1. ABC receives SLA template from CSP1 and CSP2, adds negotiation parameters according to its objectives and starts an automated negotiation process with the CSP1 and CSP2. An agreement is made with the CSP1 after negotiation process. According to the final SLA, the following terms are agreed. If monthly availability of the*

Service description:
$SLAName^T = S1\_SLA$, $ServiceProvider^T = CSP1$, $ServiceUser^N = ABC$
Variables:
var $x_1 = PlannedDowntime = 5 \times 5 = 25mins$
var $x_2 = TotalServiceTimeAgreed$ =<Total agreed service time in the month>
var $x_3 = ActualAvailability$ = <Monitored value>
var $x_4[\,] = DurationPerOutageInTheMonth$ =<Array of outage durations>
var $x_5 = MonthlyServiceCost = x_3 \times$ <price per unit>
var $x_6[\,] = UserAuthenticationTimes$ =<Array of monitored values>
var $x_7 = MonthlyLoginRequests$ =<total number of login requests>
var $x_8 = AccountBalance$ =<External value from accounting system>
SLOs:
$SLOID^T = SLO_1$, $SLOName^T = Availability$, $M_s^T(SLO_1) = \{M_1, M_2, M_3\}$
$SLOID^T = SLO_3$, $SLOName^T = Authentication$, $M_s^T(SLO_3) = \{M_4, M_5\}$
Metrics:
$MetricID^T = M_1$, $MetricName^T$ =Availability percentage
$MetricID^T = M_2$, $MetricName^T$ = Number of outages per month
$MetricID^T = M_3$, $MetricName^T$ = Duration per outage
$MetricID^T = M_4$, $MetricName^T$ = Authentication protocol
$MetricID^T = M_5$, $MetricName^T$ = Average authentication time per user

$MetricID^T$	M_1	M_2	M_3	M_4	M_5
$IsNegotiable^N$	*true*	*true*	*true*	*true*	*true*
$IsMandatory^N$	*true*	*true*	*true*	*true*	*true*
$Weight^N$	0.40	0.15	0.15	0.15	0.15
$AcceptableValue^N$	95%	6	10 mins	TACACS+	5 sec
$DesiredValue^N$	100%	2	1 min	OpenID	1 sec
$AgreedValue^F$	96%	5	6 mins	OpenID	3 sec
$Deadline^N$	20	4	10	5	5
CV_s^N	$\{\langle 0.001, 15\rangle, \langle 0.007, 20\rangle\}$	$\{\langle 1, 4\rangle\}$	$\{\langle 1, 10\rangle\}$	$\{\langle 1, 5\rangle\}$	$\{\langle 1, 5\rangle\}$
$Computation\ Formula^F$	$\frac{x_3 + x_1}{x_2} \times 100$	$Length(x_4)$	x_4	None	$\frac{Sum(x_6)}{x_7}$

Table 3.2: Example of S3LACC based SLA for the scenario described in Section 3.8 (Part 1/2)

```
**Guarantees:**
$GuaranteeID^F = G_1$, $GuaranteePreondition^F = M_1 < 96\%$
- $GuaranteeAction()^F\{$
      var $ClaimAmount = 0.2 \times x_5$;
      Send claim to CSP1 of $ClaimAmount$;
      Send logged information to CSP1 for monthly service unavailability;
  $\}$
$GuaranteeID^F = G_2$, $GuaranteePreondition^F = M_5 > 3$ seconds
- $GuaranteeAction()^F\{$
      var $ClaimAmount = 0.07 \times x_5$;
      Send claim to CSP1 of $ClaimAmount$;
  $\}$
**Obligations:**
$ObligationID^F = O_1$,
- $ObligationPreondition^F = x_8 <$ reserve amount $AND$ bill is due tomorrow
- $ObligationAction()^F\{$
      Inform CSP1 about delay in payment
  $\}$
```

Table 3.3: Example of S3LACC based SLA for the scenario described in Section 3.8 (Part 2/2)

S1 is less than 96% *then CSP1 will reimburse* 20% *of the monthly service cost. S1 may have up to* 5 *outages per month with maximum duration of* 5 *minutes per outage. ABC is responsible to provide logged information of service unavilability (date, time, duration). If average user login time is more than* 3 *seconds then CSP1 is liable to reimburse* 7% *of the monthly service cost. ABC is responsible to inform CSP1, one day in advance if ABC wants payment of the S1 to be delayed.*

Above SLA description is transformed to the S3LACC based SLA as shown in Table 3.2 and Table 3.3. In this use case, only relevant SLA parts of the S3LACC are included in this description. SLO_1 (*Availability*) contains three metrics (with metric IDs M_1, M_2 and M_3) and SLO_2 (*Authentication*) contains two metrics (a qualitative metric M_4 and and a quantitative metric M_5). Assumed negotiation and agreement parameters are described for each metric in a nested table for illustration. The metric parameter $Weight^N$ is set by the ABC according to its priority (supposed) for each metric. The negotiation process for M_4 starts from highest priority authentication protocol (i.e. *OpenID*) and if not accepted by a CSP then a lower priority protocol is suggested in next round. CV_s^N represents the list of tuples containing concession value up to a specific deadline during the negotiation process, e.g. $\{\langle 0.001, 15\rangle, \langle 0.007, 20\rangle\}$ for M_1 indicates that 0.1% concession is given on initial preferred value of 100% until 15 rounds of the negotiation process. From 16th round and until 20th round, 0.7% concession is offered. Variables (containing values of different sources) are used in computation formulae of some metrics.

3.9 Summary

In this chapter, we have proposed a specification for SLAs in cloud computing (S3LACC) with a good trade-off between complexity and expressiveness . Our specification targets specific requirements of cloud domain such as complex dependencies among different metrics and composition of different metrics in one SLO (through metric lists). Current approaches lack standardized structure definition according to international standards for the cloud computing SLAs. Also, support for automation of the complete SLA life cycle is generally ignored in most of the SLA specification languages and models. S3LACC meets all of these requirements by defining an intuitive SLA structure which can be used to implement almost all types of negotiation strategies and monitoring policies for an automated SLA life cycle. Also, renegotiations in case of QoS violations and automated recycling of the SLA is possible using S3LACC. Qualitative parameters are an important part of the cloud SLAs which are easily definable using the S3LACC.
Note: This SLA specification (S3LACC) has been published in the following paper:
Waheed Aslam Ghumman, Alexander Schill. *Structural Specification for the SLAs in Cloud Computing (S3LACC).* In Proceedings of the 13th International Conference on Economics of Grids, Cloud, System and Services (GECON), Springer Publishing, September 2016, Athens, Greece.

Chapter 4

Automated SLA Negotiation

The SLA negotiation between a cloud service provider (CSP) and a cloud service user (CSU) is a pivotal process in the SLA life cycle. An SLA may include many negotiable quality of service (QoS) parameters with each QoS parameter having multiple options. Different combinations of possible selections grow exponentially as number of QoS parameters and their options grow. Manually negotiating for all of the QoS parameters considering their various options can be an erroneous and diligent task with lower utility. It is gainful for both of the CSU and the CSP to automate the negotiation process with minimum human interaction. In previous chapter, a structural specification (S3LACC) is presented to formally represent the SLAs for the cloud services in a machine readable format. The SLA described in the S3LACC is utilized as a basis in this chapter to include a negotiation strategy. The negotiation process may involve a broker or agent to mediate the negotiation process. Also, negotiation setups may differ with respect to the number of CSUs and the CSPs involved in the negotiation process. Each party (a CSU or a CSP) prepares its offer with respect to its business objectives which are quantized by assigning different utility levels (using a utility function) to the different values of an SLA parameter. The party receiving the offer evaluates the received offer using its offer evaluation function. A party may accept an offer or reject an offer with or without a counter offer. Subsequent offers usually include a concession (calculated using a concession computation function) on initially offered value to make an offer acceptable to the opponent. The automated negotiation process can be divided into the following sub tasks:

- Negotiation protocol (Section 4.1)
- Negotiation strategy (Section 4.2)
  - Concession computation function
  - Offer generation using utility function
  - Offer evaluation function

4.1 Negotiation Protocol

The CSU's requirement of a new cloud service leads the CSU to discovery of the CSPs offering such a cloud service. After shortlisting the appropriate CSPs, the CSU gets an SLA template from the relevant CSPs. The SLA template contains the SLA parameters including their values as a basis for the negotiation process. The CSU starts negotiation process with each shortlisted CSP separately. The CSU makes first offer to a CSP using the SLA template, business objectives and requirement constraints. Acquisition of new services without manual intervention is also a mandatory requirement in most of the cases [27]. As QoS parameters of a cloud service are defined in an SLA, so it requires an automatic negotiation process for an SLA to timely and effectively acquisition of a cloud service. The negotiation protocol used in this work is based on Rubinstein's bargaining model [24] as described in chapter 2, with few alterations. The negotiating parties (the CSU and the CSP) exchange the offers using Rubinstein's alternating offer protocol, however, contrary to Rubinstein's model, the negotiation process in this work is time bound with limited rounds of negotiation. Generally, the SLA negotiations are bound by number of negotiation rounds (termed as deadline). However, fixed number of rounds as a deadline approach may not be suitable for time critical cloud services. For example, consider a case where a CSP requires to search for a new cloud service for sudden increase in its web based utility service. If fixed-rounds are used as a deadline during the negotiation in this example then there might be a chance that SLA negotiation negotiation process with different CSPs is delayed due to network delays, computation time or due to complex negotiation strategies. Any delay in the negotiation process might be costly for the CSU. Conversely, a deadline based on amount of time ensures that either agreement is made during the given time or negotiation process is terminated at the end of the deadline. In this work, amount of time is used as a deadline. In realistic scenarios, it might not be possible to enforce a single deadline (same amount of time) for all CSPs and CSUs, so there are higher chances that each CSP has its own deadline for negotiation process, mentioned in the SLA template. On the other hand, a CSU may have a different requirements for the negotiation deadline according to its business objectives. So, a CSU may set a different deadline than the deadline set by the CSP in the template. However a CSU is not allowed to set the deadline greater than the deadline of the CSP. The CSU is required to communicate its shorter deadline to the CSP. Let T_{max}^{CSU} and T_{max}^{CSP} represent the maximum time for the negotiation process individually set by the CSU and the CSP, respectively such that $T_{max}^{CSU} \leq T_{max}^{CSP}$ and T_{max} is the maximum time for the complete negotiation process then T_{max} is the minimum of the T_{max}^{CSU} and T_{max}^{CSP}, i.e.:

$$T_{max} = min(T_{max}^{CSU}, T_{max}^{CSP}) \tag{4.1}$$

Let
T_o^a be the time to generate an offer by a party a,
$T_t^{a \longrightarrow b}$ be the time to travel an offer, acceptance, rejection or confirmation from a to the other party b (offer receiver),
T_e^b be the evaluation time taken by the party b to make a decision (acceptance, rejection or counter offer),
T_{co}^b be the time taken by b to generate a counter offer, and
T_e^a be the time taken by a to evaluate the response (counter offer) of b,
then total round trip time $T_{rtt}^{a \longleftrightarrow b}$ of generating an offer by a and

getting its response or counter offer from b is following:

$$T_{rtt}^{a\longleftrightarrow b} = T_o^a + T_t^{a\longrightarrow b} + T_e^b + T_{co}^b + T_t^{b\longrightarrow a} + T_e^a \tag{4.2}$$

The maximum number of complete negotiation rounds NR_{max} is the floor of the division of the total time allowed for negotiation by the time taken for one round trip of the negotiation process, i.e.

$$NR_{max} = \left\lfloor \frac{T_{max}^a}{T_{rtt}^{a\longleftrightarrow b}} \right\rfloor \tag{4.3}$$

Let NR_{act} be the actual number of rounds completed during the negotiation process including the decision round of acceptance or rejection for an offer or counter offer, where $NR_{act} \leq NR_{max}$, then total time for the negotiation process (T_N) is

$$T_N = \begin{cases} T_{rtt}^{a\longleftrightarrow b} \times NR_{act} - T_{co}^b - T_e^a + T_t^{a\longrightarrow b} & if\ b\ accepts\ offer \\ T_{rtt}^{a\longleftrightarrow b} \times NR_{act} - T_o^a - T_e^b + T_t^{b\longrightarrow a} & if\ a\ accepts\ counter \\ & offer \\ T_{rtt}^{a\longleftrightarrow b} \times NR_{act} - T_{co}^b - T_e^a & if\ b\ rejects\ offer \\ T_{rtt}^{a\longleftrightarrow b} \times NR_{act} + T_t^{a\longrightarrow b} & if\ a\ rejects\ offer \\ T_{max} & if\ negotiation \\ & process\ timesout \end{cases} \tag{4.4}$$

If deadline of the negotiation process is based on number of negotiation rounds then user's control over time taken per negotiation round is lost which may increase negotiation duration significantly. The negotiation protocol (bound by maximum time allowed for the negotiation process) is shown in Fig. 4.1. The negotiation process of offers and counter offers continues until acceptance by one of the parties, rejection by one of the parties or T_{max} is reached. If an acceptance is sent by one of the parties then other party sends a confirmation of acceptance to complete the SLA agreement. If a confirmation is not sent until T_{max} then agreement is not made and negotiation process is considered void. The two step acceptance protocol (i.e. acceptance and confirmation) is beneficial for both CSU and CSP. The CSU may evaluate offers of other CSPs after

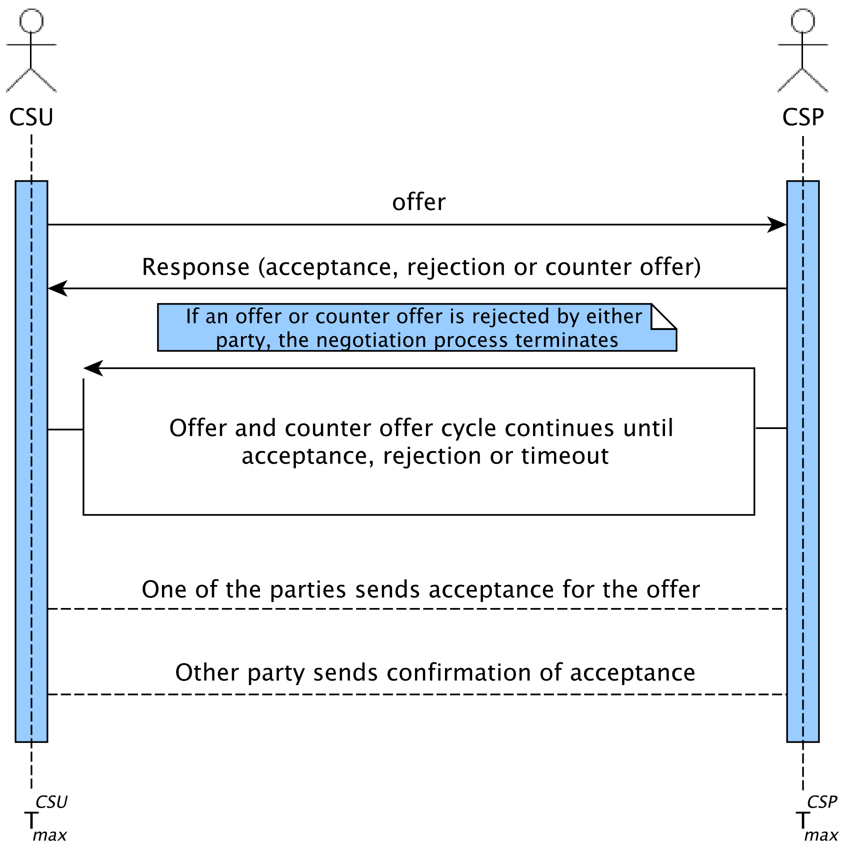

CSU
CSP
offer
Response (acceptance, rejection or counter offer)
If an offer or counter offer is rejected by either party, the negotiation process terminates
Offer and counter offer cycle continues until acceptance, rejection or timeout
One of the parties sends acceptance for the offer
Other party sends confirmation of acceptance
T_{max}^{CSU}
T_{max}^{CSP}

Figure 4.1: Negotiation protocol based on alternating offer model

receiving an acceptance from the CSP and before making the final selection. On the other hand, if an acceptance is sent by the CSU then the CSP may evaluate offers of other CSUs before making its final decision. A similar negotiation protocol is used by Dang *et al.* [55] where acceptance and confirmation are termed as pre-accept and accept, respectively. Similarly, pre-reject and reject are used as a two phase rejection protocol in [55]. In this work, rejection is only one step process and confirmation of rejection is not required. This restriction of single step rejection enforces a very useful rule that a party must send a counter offer if an offer is not accepted to ensure the fair play, i.e. a party sending the rejection should not expect a new offer from its opponent.

4.2 The Flip-Flop Negotiation Strategy and its Building Blocks

In previous section, the negotiation protocol is explained. In this section a concurrent negotiation strategy is presented for the model setup $1-n$ without broker, i.e. one CSU negotiates with n number of CSPs over SLA parameters. Before describing the algorithm for the negotiation strategy, different components of the negotiation strategy are formalized in the following subsections:

4.2.1 Time Based 3D Linear Utility Function

A utility function is used to rate the worthiness of a generated offer or of a counter offer received from the other parties. Generally, utility function $U(x)$is based on single parameter *value* x, i.e. $y = U(x)$. In the following, utility function $U(x)$ is derived where $x \in \{offer,\ counter\ offer\}$.
Let M_a be a metric to measure an SLO. M_a has a best value V_b (most desired value of the M_a) and a worst value V_w (also referred as a reserve value or a worst acceptable value). The utility level of V_b is u_{max}, utility level of V_w is u_{min} and $\Delta u = u_{max} - u_{min}$. Let V be a value of M_a in the interval $[V_w, V_b]$ then the utility of the

value V is given by the function $u(V)$ as below:

$$u(V) = u_{min} + \Delta u \times \left(\frac{V - V_w}{V_b - V_w} \right) \tag{4.5}$$

The equation 4.5 gives a linear utility function of any given value V of a metric M_a. The utility level may increase or decrease with increasing value of the metric depending on its ratio type (directly proportional or inversely proportional). If utility level increases with increasing value V then its directly proportional metric and if the utility level decreases with increasing value V then its inversely proportional metric. In other words, if $V_b > V_w$ then its directly proportional metric and if $V_w > V_b$ then its inversely proportional metric. In most of the related work, either two types of utility functions are defined separately for two types of metric ratio types or a partial function is defined to calculate utility level depending on the metric ratio type, e.g. in [36]. However, it is not necessary to define two separate functions or a partial function for calculating the utility level. The utility function defined in the equation 4.5 covers both types of metrics.
An SLA contains one or more SLOs. Let $M_s = \{M_1, ...M_m\}$ be a set of negotiable metrics where $m \geq 1$ and $SLO_s = \{SLO_1, ..., SLO_n\}$ be a set of SLOs, where $n \geq 1$. Each $SLO_i \in SLO_s$ is measured by a unique subset of metrics $M_i \subset M_s$ where $1 \leq i \leq n$. If SLO_i does not contain any negotiable metric then $M_i = \varnothing$. An offer $O_k^{a \longrightarrow b}$ (also termed as a *bid* in literature) from the party a to the party b is comprised of the selected values of all negotiable metrics, i.e. $O_k^{a \longrightarrow b} = \{V_{1,k}^{a \longrightarrow b}, ...V_{m,k}^{a \longrightarrow b}\}$ where $O_k^{a \longrightarrow b} \in O_s^{a \longrightarrow b}$, $O_s^{a \longrightarrow b} = \{O_1^{a \longrightarrow b}, ..., O_p^{a \longrightarrow b}\}$ and $1 \leq k \leq p$. Similarly, $CO_s^{b \longrightarrow a} = \{CO_1^{b \longrightarrow a}, ..., CO_q^{b \longrightarrow a}\}$ are counter offers from the party b to the party a. Each metric M_j also contains a weight w_j representing its priority, where $1 \leq j \leq m$. The selected value of metric M_j in a kth offer from a to b is $V_{j,k}^{a \longrightarrow b}$. The overall utility $U(O)$ of an offer O is computed as below (using equation 4.5):

$$U(O_k^{a \longrightarrow b}) = \sum_{j=1}^{m} w_j \times u(V_{j,k}^{a \longrightarrow b}) \tag{4.6}$$

A similar utility function (with few variations) is used by the different SLA negotiation approaches, e.g. in [35]. The difference in the utility function defined in the equation 4.6 and in the [35] is that the overall utility in equation 4.6 is computed using the individual utility (V_j) of each metric M_j whereas overall utility in [35] is computed using the real selected value of each metric.
As discussed in Section 4.1, the negotiation protocol is time dependent rather than depending on the number of negotiation rounds. An earlier agreement is in favor of both CSP and CSU. The utility may decrease if an agreement is made towards the end of the negotiation process. The utility functions given in the equations 4.5 and 4.6 are two dimensional, i.e. the value V_j and its utility $u(V_j)$. Considering this fact, computation of the overall utility should also include the effect of time elapsed during the negotiation process. So, equation 4.5 is modified to compute a time dependent three dimensional utility function $u(V_{j,k}^{a\longrightarrow b}, T_\epsilon)$, time as a third dimension) for the value $V_{j,k}^{a\longrightarrow b}$ of a metric M_j in the equation 4.7 as in the following:

$$u(V_{j,k}^{a\longrightarrow b}, T_\epsilon) = u_{min}^{T=0} + \Delta u^{T=0} \times \left(\frac{V_{j,k}^{a\longrightarrow b} - V_w}{V_b - V_w} \right) - (\lambda_T \times T_\epsilon) \quad (4.7)$$

where T_ϵ is the unit number of time elapsed during the negotiation process, $u_{min}^{T=0}$ is the utility for the value V_w at the start time, i.e. $T = 0$, $u_{max}^{T=0}$ is the utility for the value V_b at time $T = 0$, $\Delta u^{T=0} = u_{max}^{T=0} - u_{min}^{T=0}$ and λT is the depreciation factor to determine the decrease in the initial utility of V_j due to passage of time. Similarly, the overall utility function $U(O)$ (equation 4.6) is modified to include the effect of the time elapsed during the negotiation process as below:

$$U(O_k^{a\longrightarrow b}, T_\epsilon) = \sum_{j=1}^{m} w_j \times u(V_{j,k}^{a\longrightarrow b}, T_\epsilon) \quad (4.8)$$

Example 4.2.1 (3D Linear Utility Function) *Let's consider an example metric availability percentage including the effect of time*

T_ϵ	**Availability Percentage**					
	100	**99.9**	**99.8**	**99.7**	**99.6**	**99.5**
0	1	0.9	0.8	0.7	0.6	0.5
1	0.9	0.8	0.7	0.6	0.5	0.4
2	0.8	0.7	0.6	0.5	0.4	0.3
3	0.7	0.6	0.5	0.4	0.3	0.2
4	0.6	0.5	0.4	0.3	0.2	0.1

Table 4.1: Example data for the *availability percentage* metric with effect of T_ϵ on utility level

on its utility level as given in the following:
$u^T_{min} = 0 = 0.5$ *(minimum utility of the availability percentage value* V_j *at time* $T = 0$*)*
$u^T_{max} = 0 = 1$ *(maximum utility of the availability percentage value* V_j *at time* $T = 0$*)*
$\Delta u^{T=0} = 1 - 0.5 = 0.5$ *(difference in the minimum and maximum utility values at* $T = 0$*)*
$V_w = 99.5$ *(worst acceptable value of the metric availability percentage)*
$V_b = 100$ *(best possible value of the metric availability percentage)*
$\lambda T = 0.1$ *(depreciation factor, i.e. utility decrease by* 10% *with passage of every unit of time)*
By using above data along with the different values of the availability percentage and time in the equations, different values of the utility level are obtained as shown in table 4.1, e.g.:

$$u(99.8, 3) = 0.5 + 0.5 \times \left(\frac{99.8 - 99.5}{100 - 99.5}\right) - (0.1 \times 3) = 0.5$$

The availability percentage is given in the second row from column two to seven. Elapsed unit time (T_ϵ*) is given in first column from row three to row seven. The utility level of availability percentage with respect to elapsed time is given in the cells from* $(2, 3)$ *to* $(7, 7)$ *where first elements in* $(2, 3)$ *and* $(7, 7)$ *are column numbers and second elements are row numbers. The graphical representation of this three dimensional data is shown in Figure 4.2.*

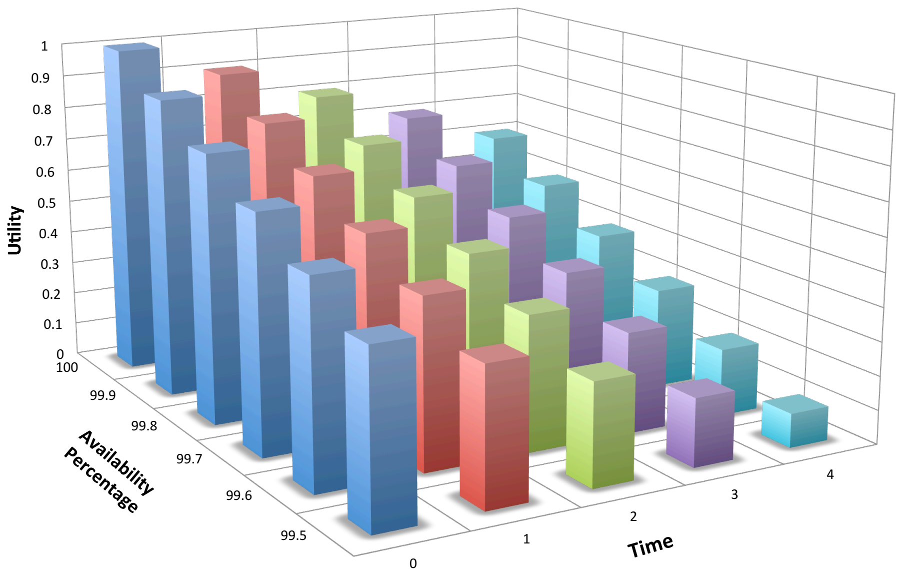


Figure 4.2: Example of time based 3D linear utility function

4.2.2 Concession Computation using Polynomial Interpolation

Concession is the amount which is applied on the initial value of the metric and it reduces the utility of a negotiable SLA parameter. By allowing the concession, a party is moving towards its opponent's preferences. By allowing an inaccurate concession amount without due consideration of the negotiation environment factors (e.g. opponent's concession pattern, deadline or overall utility) may result in a lesser useful agreement. In this section, a concession computation method is presented which is based on opponent's concession pattern using the polynomial interpolation. The estimation of the opponent's concession for later offers is used as a basis to compute CSU's own concession value. The process of the concession computation is explained in the following.

The first offer from the CSU is generated with the most preferable values of all negotiable metrics which gives the maximum utility. For the initial four rounds of the negotiation process, the CSU uses the concession value as specified in the negotiation parameters of M_j (see Section 3.5.1). Let $T_s = \{T_0, T_1, ..., T_{max}\}$ be a set of unit time values in increasing order such that $T_u \in T_s$, $0 \leq T_u \leq T_{max}$. The first concession $(\zeta_1^{b \longrightarrow a}(M_j))$ on metric M_j from the opponent

b to a is $\zeta_1^{b\longrightarrow a}(M_j) = V_{j,2}^{b\longrightarrow a} - V_{j,1}^{b\longrightarrow a}$, i.e. value of metric M_j in the counter offer 2 subtracted by the value of the metric M_j in the counter offer 1. Similarly, the set of concession value for the metric M_j is $\zeta_{s,j}^{b\longrightarrow a} = \{\zeta_{1,j}^{b\longrightarrow a}, ..., \zeta_{q-1,j}^{b\longrightarrow a}\}$ for the set of points in time $T_s(CO^{b\longrightarrow a}) = \{T(CO_2^{b\longrightarrow a}), ..., T(CO_q^{b\longrightarrow a})\}$ at which the counter offers are received, respectively. The opponent's concession function $(\alpha_j(T_u))$ for the metric M_j on the basis of its last three concession values, received in the counter offers (using polynomial interpolation method) is computed as given in the following equation:

$$\alpha_j(T_u) = \sum_{\substack{\iota=v \\ 0\leq v\leq q}}^{\iota+2} \left(\prod_{\substack{\kappa=v+1 \\ \kappa\neq\iota}}^{v+3} \frac{T_u - T(CO_\kappa^{b\longrightarrow a})}{T(CO_\iota^{b\longrightarrow a}) - T(CO_\kappa^{b\longrightarrow a})} \right) \zeta_{\iota,j}^{b\longrightarrow a} \tag{4.9}$$

Semantically, equation 4.9 gives a function to compute the opponent's concession value at time T_u using the last three concession values and the points of time at which last three counter offers were received. Using equation 4.9, opponent's concession is predicted for the remaining negotiation process considering the expected number of negotiation rounds using equations 4.2 and 4.3, i.e. $\{\alpha_j(T_u), \alpha_j(T_u + T_{rtt}^{a\longleftrightarrow b}), ..., \alpha_j(T(CO_q^{b\longrightarrow a}))\}$. By evaluating the opponent's expected concession values in all of the remaining counter offers (using equation 4.9), it is possible to predict the expected final value $V_{j,q}^{b\longrightarrow a}$ for a metric M_j in the final counter offer $CO_q^{b\longrightarrow a}$ that opponent is expected to offer. Let $V_{j,w}$ is the worst acceptable value (reserve value) of the metric M_j then the user's concession UC is set as given in the following:

$$UC = \gamma_j(T_u) = \begin{cases} \dfrac{V_{j,k}^{a\longrightarrow b} - V_{j,q}^{b\longrightarrow a} - \theta}{NR_{rem}} & if\ V_{j,w} \leq V_{j,q}^{b\longrightarrow a} \\ \dfrac{V_{j,k}^{a\longrightarrow b} - V_{j,w} - \theta}{NR_{rem}} & if\ V_{j,w} > V_{j,q}^{b\longrightarrow a} \end{cases} \tag{4.10}$$

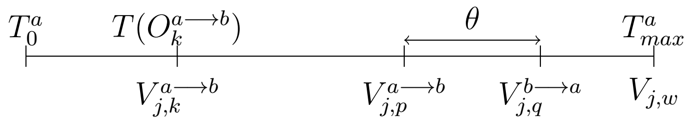

T_0^a
$T(O_k^{a\longrightarrow b})$
θ
T_{max}^a
$V_{j,k}^{a\longrightarrow b}$
$V_{j,p}^{a\longrightarrow b}$
$V_{j,q}^{b\longrightarrow a}$
$V_{j,w}$

Figure 4.3: Concession computation $\gamma_j(T_u)$ if $V_{j,w} > V_{j,q}^{b\longrightarrow a}$

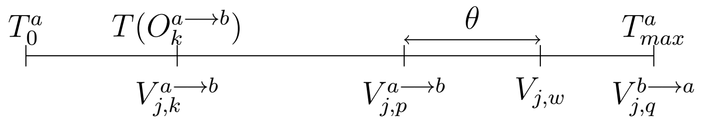

T_0^a
$T(O_k^{a\longrightarrow b})$
θ
T_{max}^a
$V_{j,k}^{a\longrightarrow b}$
$V_{j,p}^{a\longrightarrow b}$
$V_{j,w}$
$V_{j,q}^{b\longrightarrow a}$

Figure 4.4: Concession computation $\gamma_j(T_u)$ if $V_{j,w} \leq V_{j,q}^{b\longrightarrow a}$

where NR_{rem} are the remaining number of rounds in the negotiation process, i.e. until the last counter offer is received and $NR_{rem} = NR_{max} - NR_{act}$ and θ is the amount that is subtracted further from the expected final counter offer value $(V_{j,q}^{b\longrightarrow a})$ from the opponent to keep a gap between $V_{j,q}^{b\longrightarrow a}$ and the last expected offer $(V_{j,p}^{a\longrightarrow b})$ of the user , which is also useful to hide the exact reserve value $V_{j,w}$ from the opponent. This gap (θ) is added in the final offer $(V_{j,q}^{b\longrightarrow a}$) if agreement is not reached until the final offer. This method dynamically changes user's concession by predicting the opponent's expected offer using polynomial extrapolation. The representation of the values in equation 4.10 is shown in Figures 4.3 and 4.4 for the first and the second case of the equation 4.10, respectively.

4.2.3 Flip-Flop Negotiation Strategy

As initial offer from the CSU contains best possible values of all metrics, it generates highest utility for the CSU. Subsequent offers and counter offers yield lesser utility. On the other hand, selection of suitable values for each metric is a complex task. The number of possible combinations of different values of the negotiable metrics grows very rapidly as number of metrics (in general literature called as issues) and their acceptable values increase. For instance, if number of negotiable metrics is m and each metric has χ number of possible values then total numbers of possible offers are χ^m. In realistic scenarios, each metric has different number of possible values. The total number of possible offers (NO_{max}) for m number of metrics with each metric having different number of possible values is computed as given in the following equation:

$$NO_{max} = \prod_{j=1}^{m} NV_j \tag{4.11}$$

where NV_j is the number of possible values in metric M_j. Each offer may have different utility and different offers may have same utilities due to difference of time at which they are selected. In a limited amount of negotiation time, it is hard to optimize the negotiation result by sending every possible combination as an offer starting from the highest utility offer and moving towards the lowest utility offer. So, an intelligent strategy is required to select the appropriate offers during the negotiation process. In this section a negotiation strategy is presented which selects the appropriate value of each metric by considering the opponent's concession pattern and by utilizing the time based 3d utility function. The main idea is to reach an early agreement and to increase the utility level due to time. The negotiation strategy is outlined in the following:

1. The negotiation process starts by generating the user's (CSU) offers using the concession value as set in metric's negotiation parameters (as defined in the SLA specification presented in chapter 3) until $NR_{act} = 4$;

2. The counter offers from the opponent (CSP) and the points in time (when they are received) are recorded;

3. Opponent's concession is estimated based on the first four counter offers using polynomial extrapolation (equation 4.9);

4. User's concession (UC) is determined according to the equation 4.10;

5. User increases UC by a factor δ i.e. $UC' = UC + \delta$ such that estimated number of negotiation rounds are decreased by 1. The motive to increase user's concession from UC to UC' is to reach an early agreement and to achieve higher

utility due to time (as a same metric value has higher utility at earlier agreement time). For instance, if UC is determined with respect to NR_{rem} then:

$$UC' = \frac{V_{j,q}^{b\longrightarrow a} - \theta - V_{j,k}^{a\longrightarrow b}}{NR_{rem} - 1} \tag{13}$$

and $\delta = UC' - UC$. This process of increasing the concession is termed as *flip*. The δ value is also pushed to a stack (*ConcessionStack*) to keep track of increased concessions values;

6. If opponent keeps its concession rate constant (same as it was in previous counter offer) or decreases the concession value (indicating that opponent is following a greedy strategy), then user recalculates UC using equations 4.9 and 4.10. If *ConcessionStack* is not empty then a value (σ) is popped from the stack and UC is subtracted by the σ i.e. $UC'' = UC - \sigma$. This process of decreasing the concession value is termed as *flop*. The intuition behind *flop* step is to recover the loss that was made in previous offer with a motive to reach an agreement and which was not possible due to opponent's greedy strategy;

7. After *flop* step, the user attempts another *flip* step while preparing the next offer and δ is moved to the *ConcessionStack*;

8. If opponent responds with an increased concession such that percentage increase is equal to or greater than user's increase percentage then user keeps on repeating *flip* step until opponent's concession becomes again constant or opponent's concession is decreased resulting in a *flop* step. The condition of comparing percentage increase (of the user's concession and percentage increase of the opponent's concession) ensures that agreement is still reached at equal or better value (despite increased concession) than it was expected with initial concession UC;

9. The *flip-flop* process continues until an agreement is reached

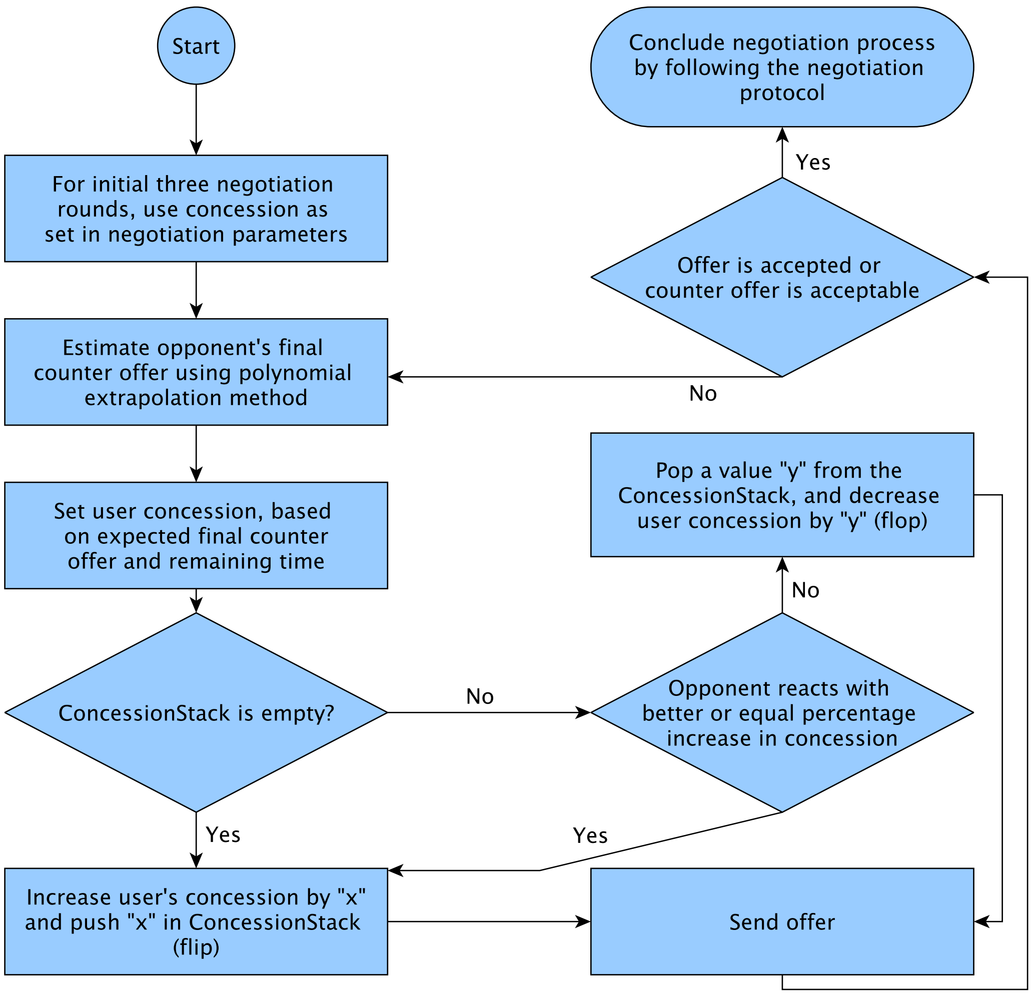

Start
For initial three negotiation rounds, use concession as set in negotiation parameters
Estimate opponent's final counter offer using polynomial extrapolation method
Set user concession, based on expected final counter offer and remaining time
ConcessionStack is empty?
Yes
No
Increase user's concession by "x" and push "x" in ConcessionStack (flip)
Send offer
Opponent reacts with better or equal percentage increase in concession
Yes
No
Pop a value "y" from the ConcessionStack, and decrease user concession by "y" (flop)
Offer is accepted or counter offer is acceptable
No
Yes
Conclude negotiation process by following the negotiation protocol

Figure 4.5: Flip-flop negotiation strategy flowchart

or negotiation process times out.

Above process is depicted in Figure 4.5. The *flip-flop* negotiation strategy is very safe to be used such that even in worst case no loss is made to user as any increase during the *flip* process is reversed in the *flop* step if *flip* was not useful to the user. On the other hand, if opponent responds positively then, with an increase in concession with the same percentages by the user and the opponent, makes a win-win situation, i.e. both reach at the same agreement value as it was expected with initial concession rates but in lesser number of rounds. The acceptance or rejection of a counter offer is determined based on the condition that if counter offer from the opponent has better utility than the user's next offer and it is better than or equal to reserve value then the counter offer is accepted.

4.3 Multi-Provider Concurrent Negotiations

In above sections, negotiation method for a single user and single provider is presented. However, in realistic scenarios, a CSU generally has more than one choices with similar cloud services. Let $P_s = \{P1, ..., P_\rho\}$ be the list of providers and $P_i \in P_s$ is a provider from the list. In the following, the negotiation process with multiple providers using above negotiation strategy is given:

1. A concurrent negotiation coordinator *CNC* manages the multi-provider concurrent negotiations. The tasks of the *CNC* include starting a negotiation service *NS* concurrently for each provider P_i, evaluating the accepted offers or counter offers and making decision of the final provider selection

2. A CSU gets SLA templates from all of the shortlisted CSPs

3. Deadline for the negotiation process is set according to the smallest of (i) all CSPs' deadlines or (ii) CSU's deadline

4. All providers are informed about the negotiation deadline if its shorter than any of them

5. *CNC* starts an instance of *NS* concurrently for each provider P_i

6. Each *NS* initiates offer generation using the rules and strategies discussed in Section 4.2 for a single CSP

7. If a CSP_i accepts an offer sent by an NS_i, then the NS_i communicates acceptance to the CNC. The CNC collects information from all of the other *NSs* (which are still in the negotiation process) about their expected overall utility from the predicted agreement value. As this model is based on 3D utility function which considers time as an important factor, so

the NS_i (reporting earlier acceptance from the CSP_i) is compared with all other *NSs* which are still in process with respect to their expected overall utility. If the NS_i has greater utility than the other (that are still in process) then an acknowledgment of acceptance is sent to the corresponding CSP_i and negotiation process with all of the remaining CSPs is terminated

8. If an NS_i gets a counter offer from a CSP_i such that the counter offer has greater or equal utility than the next offer to be sent by the NS_i then the NS_i reports that counter offer to the *CNC* to make a decision for acceptance of the counter offer. The *CNC* compares the expected utilities of the remaining *NSs* and the counter offer is accepted if all remaining *NSs* have lesser expected utility at agreement time. After an acknowledgment of the acceptance from the CSP_i is received, the negotiation process is concluded and all other *NSs* send a termination message to their corresponding CSPs.

4.4 Experimental Verification

We have implemented the *flip-flip* negotiation strategy (using Java) for multi-providers (opponents) and single user to negotiate concurrently in real-time. All experiments are performed on a MacBook Pro (Mid 2010) 2.66 GHz Intel Core i7 with 4GB memory. Example data is generated randomly (but with controlled limits) for different experiments. All metrics are considered to be inversely proportional metrics with user perspective i.e. increasing the value of a metric decrease the utility level. Experiments are performed with same data using *flip-flop* negotiation strategy and without using *flip-flop* negotiation strategy. Five metrics are included in the experiments. For each CSP, random data is generated for each metric and worst value for each metric in CSPs' data is kept lesser than then worst value for the same metric in user data. The reason behind this step is to ensure that agreement is possible between user and the CSP. If the CSP's worst acceptable

	V_b	V_w	UC	$u_{max}^{T=0}$	$u_{min}^{T=0}$	**Weight**	λ_T
Metric 1	39	79	1.33	1	0.66	0.14	0.12
Metric 2	72	95	0.77	1	0.69	0.18	0.1
Metric 3	65	86	0.70	1	0.56	0.2	0.11
Metric 4	37	64	0.90	1	0.54	0.15	0.1
Metric 5	53	103	1.67	1	0.66	0.13	0.12

Table 4.2: Example metric data for Experiment 1

	$T_{max}(ms)$	θ	$T_{RTT}^{a\longleftrightarrow b}(ms)$	$T_t^{a\longrightarrow b}$
NS_1	115000	14	3000	1
NS_2	121000	19	5000	1
NS_3	133000	20	4000	2
NS_4	117000	20	5000	3
NS_5	130000	19	6000	3
NS_6	146000	17	3000	1
NS_7	140000	14	2000	2
NS_8	102000	12	4000	1
NS_9	146000	13	2000	3
NS_{10}	140000	19	4000	2

Table 4.3: Negotiation Service (NS) data for Experiment 1, one NS is created for one provider

value is higher than the user's worst acceptable value then agreement is not possible in automated negotiation environment until a much intelligent strategy is applied which can make human like decisions i.e. even agreement is not possible and the CSP has free resources then CSP may accept an offer lesser than its worst acceptable value. We elaborate here results of three different experiments (all performed with completely different data). The data for user metrics (used in Experiment 1) is shown in Table 4.2. For Experiment 1, ten CSPs are included in the negotiation process and one NS is created for one CSP to negotiate concurrently with each CSP. The data for the NSs used in Experiment 1 is shown in the Table 4.3. The CSP's concession strategy is based on a random approach such that if user increases its concession to reach an agreement (*flip*) then there are 66.66% chances that CSP also increases the concession by the same or greater percentage than the percentage increase in user's concession and 33.33% chances are that CSP decreases its concession in its counter offer adopting

	V_b	V_w	*Concession*
Metric 1	137	61	3.8
Metric 2	134	79	2.75
Metric 3	120	73	2.35
Metric 4	96	59	1.85
Metric 5	168	88	4

Table 4.4: Data for first CSP (NS_1) for Experiment 1

the greedy approach. The input data for only first CSP included in (Experiment 1) is shown in the Table 4.4 as an example. The comparison of the utility levels (achieved using the *flip-flop* negotiation strategy and without using it) is shown in Fig. 5.3 for the Experiment 1. These utility levels represent the *agreement utility* or the utility of the offer or counter offer that is accepted by the CSP or by the user, respectively. It can be noted in Fig. 5.3 that the *flip-flop* negotiation achieved higher or equal utility in 90% of the cases in Experiment 1. A similar comparison for Experiment 2 is shown in Fig. 5.4 where the *flip-flop* generates better or equal utility level in 100% cases. The difference in percentage of higher or equal utility level is due to the difference in the data that is used in these two experiments and due to the CSP's concession strategy (with 33% chances of adopting greedy strategy). However, despite these mixed concession strategies from the CSPs, the *flip-flop* negotiation strategy performs better due to its safe concession approach, i.e. in one step, concession is increased (*flip*) to reach an early agreement but if, as a response, the CSP reacts with greedy approach then the *flip-flop* strategy reverts back the increase in concession during the previous offer by decreasing the concession in the current offer (*flop*). Fig. 4.8 shows the comparison of time to reach the agreement with each CSP in Experiment 2. The time to reach an agreement using the *flip-flop* negotiation strategy is lesser or equal for all of the cases in Experiment 2. The Fig. 5.5 shows the results of Experiment 3 where utility levels for the agreements made using the *flip-flop* and without using the *flip-flop* negotiation strategy are shown. Based on the experimental results, it can be claimed that *flip-flop* negotiation

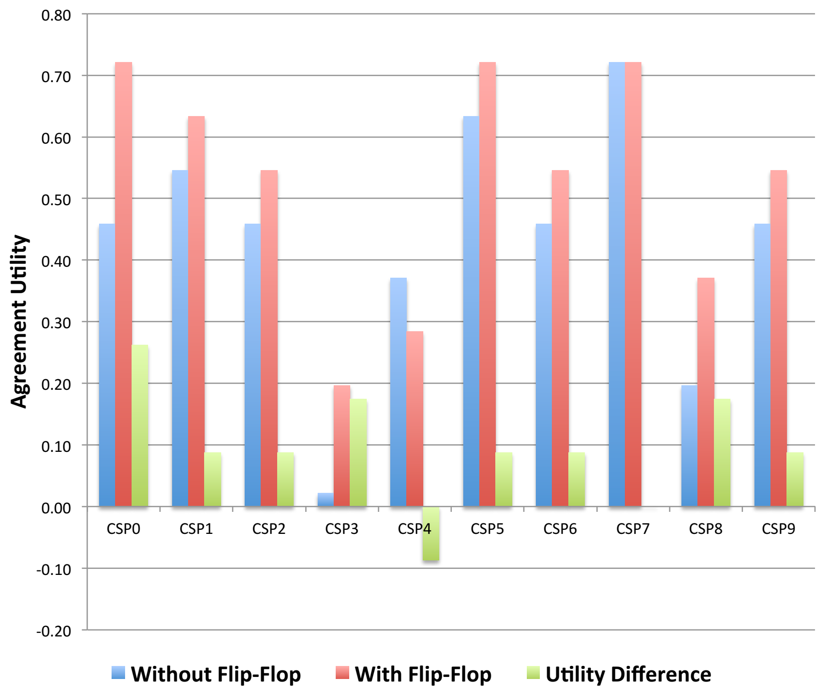


Figure 4.6: Experiment 1: comparison agreement utility for negotiation using flip-flip and without flip-flop

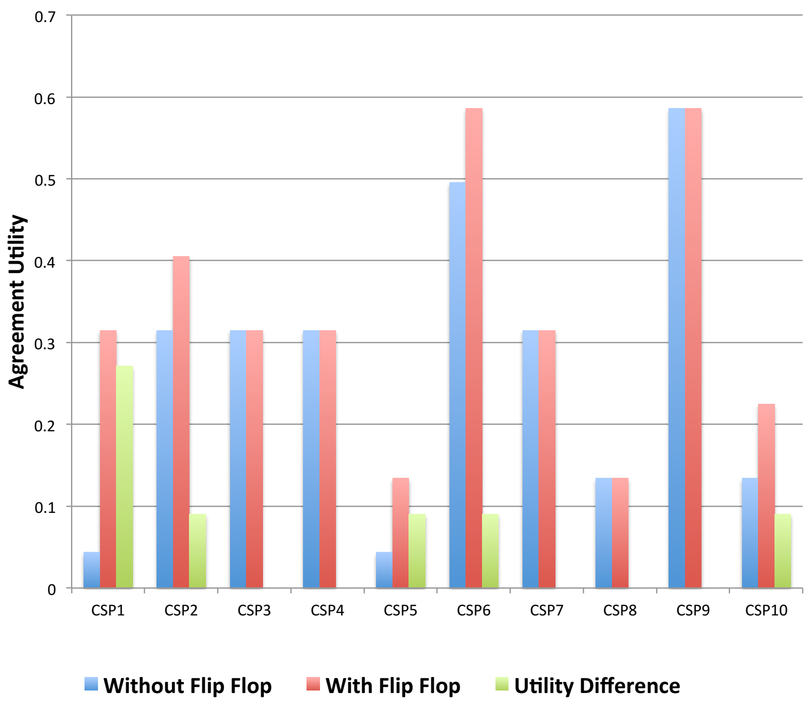


Figure 4.7: Experiment 2: comparison agreement utility for negotiation using flip-flip and without flip-flop

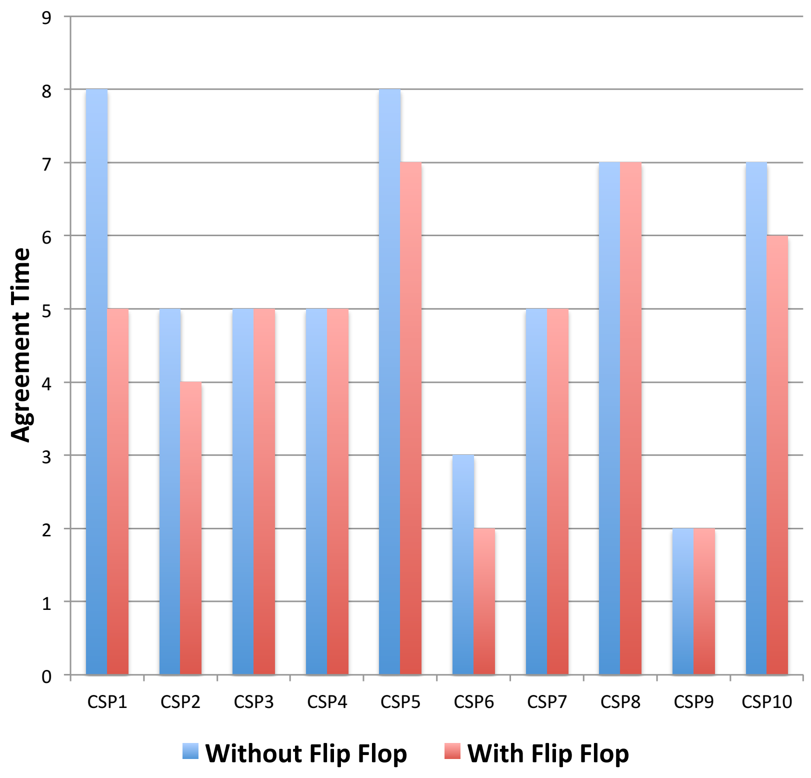


Figure 4.8: Comparison of time taken to reach an agreement for Experiment 2

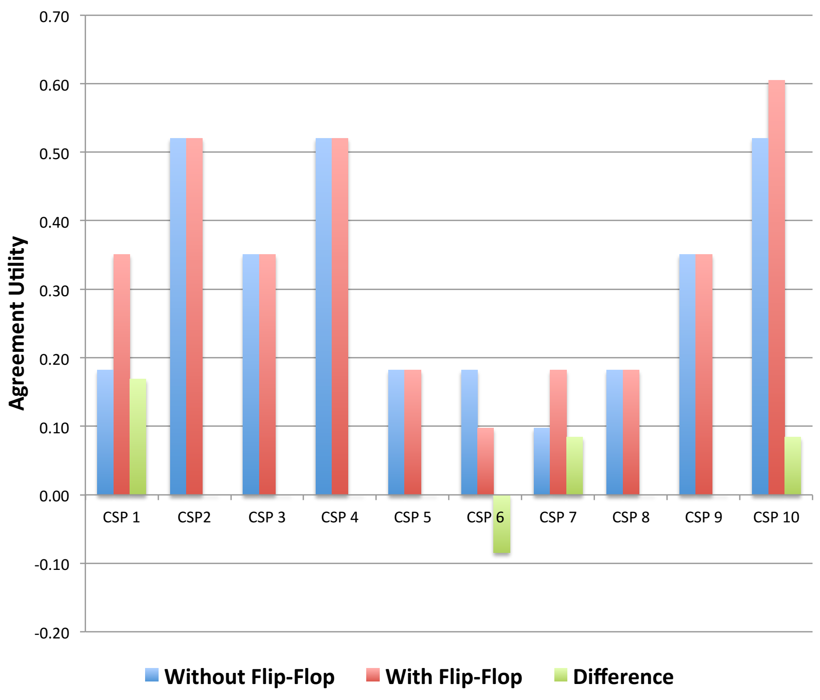


Figure 4.9: Experiment 3: comparison agreement utility for negotiation using flip-flip and without flip-flop

strategy performs better in most of the cases and is very useful for such environments in the cloud services SLA negotiation where opponent's negotiation strategy or concession rate are unknown. Also, the *flip-flop* negotiation strategy is very useful for such cloud environments where time is critical during the negotiation process as the *flip-flop* negotiation strategy decreases the time to reach an agreement which is beneficial for both CSU and CSP. The CSP's benefit of reacting positively to a *flip-flop* negotiation strategy is that the CSP may allocate its resources efficiently to its customers rather than keeping them reserved until a negotiation process is concluded. In realistic cases, the communication time between a CSU and a CSP is a bottleneck to any efficient negotiation strategy. So, if number of negotiation rounds increase due to a greedy negotiation strategy then time delays can be costly.

4.5 Summary

In this chapter, we have presented an SLA negotiation strategy that is designed specifically for such cloud services which include time as a critical factor during the negotiation process. A two-step acceptance based negotiation protocol is defined to support the *flip-flop* negotiation strategy along with the basic time calculations during the negotiation process. The fine grained approach in this work for negotiation protocol and for the negotiation strategy elaborates the building blocks of the negotiation process. The major contributions of this work include the design of a negotiation strategy for the SLA negotiation that considers the effect of time on the utility during the negotiation process and an efficient method of offer selection (using the polynomial extrapolation) to reach an agreement in lesser time. The experimental results show that such approach is beneficial to both CSU and CSP with respect to time and the utility level of the agreed SLA. The *flip-flop* negotiation strategy estimates the expected final offer from the opponent and tries to reach at the same offer value earlier in time by increasing the concession in a safe mode to mitigate the possible utility loss due to increase in concession. A multi-provider concurrent negotiation method is also presented which uses the *flip-flop* negotiation strategy.

Note: The work presented in this chapter has been published in the following paper:

Waheed Aslam Ghumman, Alexander Schill, Jörg Lässig. *The Flip-Flop SLA Negotiation Strategy Using Concession Extrapolation and 3D Utility Function.* In the proceedings of the 2nd IEEE International Conference on Collaboration and Internet Computing (CIC). November 2016, Pittsburgh, PA, USA.

Chapter 5

Automated SLA Monitoring

5.1 Introduction and Motivation

Successful negotiation process between a cloud service user (CSU) and a cloud service provider (CSP) ends up with an agreed service level agreement (SLA). Mostly, violations of SLA terms are related to agreed service levels to be provided by a CSP. However, adversely, the CSU is responsible for monitoring of SLA violations and to claim service credits based on SLA violations. An SLA for a cloud service generally contains multiple guarantees and QoS (quality of service) parameters. In real world scenarios, a cloud service is measured by precise key performance indicators (KPIs). The number of KPIs are dependent on complexity of a cloud service and objectives of a CSU. Manually monitoring all KPIs to keep track of service parameters as agreed in the SLA can easily be error-prone, time consuming and a laborious task resulting in consumption of human/technical resources. For example, service availability defined in terms of *five nines* (*99.999*%) requires high level of precision while monitoring. So, a fully automated SLA monitoring method is essential to observe KPIs, to assure quality of service and to predict the future performance of the cloud service. In this chapter, a distributed monitoring model for the cloud SLAs is presented which is extensively based on the S3LACC (presented in Chapter 3). It is assumed in this chapter that all KPIs and information relevant to the monitoring process are readily available.

5.2 Distributed Monitoring of the Cloud SLAs

Cloud services offer easy access to contracted resources (software, infrastructure, platform) from more than one locations generally. A central monitoring system might not be sufficient to capture KPIs from different locations. A classical distributed monitoring system of cloud SLAs may be modelled as given in the following:

- Agreed SLA is stored on a central location (also referred as a monitoring coordinator);
- All locations (using the cloud service) report events/information, significant with respect to the quality of service and agreed parameters;
- Monitoring coordinator analyzes the reported events/information and manages the the cloud services accordingly.

Above model is represented in the Figure 5.1 where an arrow between a location and monitoring coordinator indicates communication of events/information from a location to the monitoring coordinator and a response from the monitoring coordinator. The number of events or amount of information reported from a cloud service location to the monitoring coordinator is very significant. Because larger number of reported events may consume network bandwidth and also may increase workload on monitoring coordinator whereas lesser number of reported events may decrease the chances of reporting an important information/event to the monitoring coordinator. Different approaches have been proposed in literature to optimize the distributed monitoring with respect to number of events to be reported. An analysis of similar optimization problem is discussed in [56] where it is referred as *the countdown problem.* The countdown problem is described as in the following:

- There are k number of observers/sites ($S_s = \{S_1, S_2, ..., S_k\}$) on different locations such that $S_i \in S_s$ and $1 \leq i \leq k$;
- Each observer sees some non-overlapping events, i.e. each event is only seen by one observer;

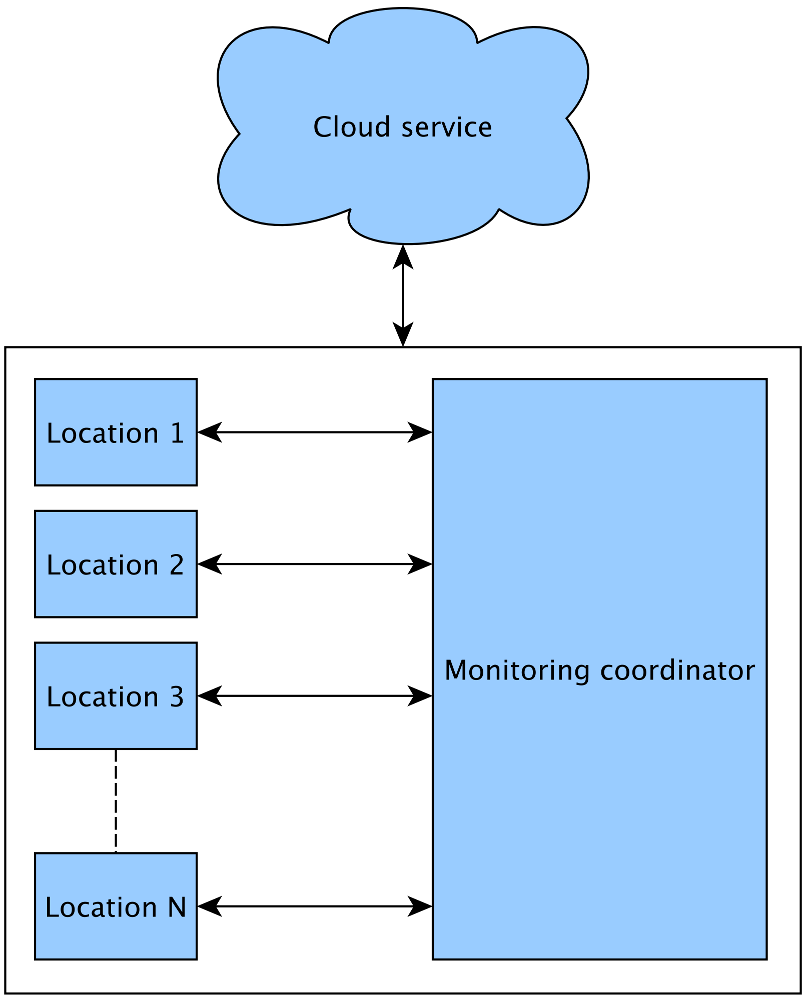


Figure 5.1: Classical distributed monitoring model for cloud SLAs

- Target is to determine when a τ number of events have been seen by an observer;
- If a small amount of information is sent to the coordinator for each unusual event then total number of communications are $\mathcal{O}(\tau)$.

An initial solution for the countdown problem as discussed in [56] proceeds as in the following:

- A limit τ/k is set as a necessary condition in the solution such that at least one observer site should observe τ/k unusual events before the threshold is reached;
- Each site begins observing events with an initial upper bound of τ/k;
- Whenever local count n_i of unusual event for a site S_i has reached the upper bound τ/k, then S_i informs the coordinator;
- The coordinator determines a new upper limit UL for all sites such that $UL = (\tau - N)/k$ where N is cumulative count of reported events so far;
- The coordinator sends a signal to all sites to reset their local counts n_i's to a new upper limit UL;
- This process continues until UL is equal to *1*, and in that case each site starts reporting every unusual event;
- The total number of communications in this method are $\mathcal{O}(k^2 \log \tau/k)$.

Few improvements to above method of distributed monitoring are also discussed in [56] by modifying the reporting rules and/or adding some constraints to the basic countdown problem.

5.3 Distributed Monitoring of Cloud SLAs Using Partial Violations

In this section, a multi-level distributed monitoring approach is presented for cloud SLAs. As discussed in Chapter 3, an SLA may contain multiple service level objectives (SLOs) and an SLO may contain one or more metrics. Moreover, an SLA contains guarantees and obligations to assure the quality of service. Let,
$SLO_s = \{SLO_1, SLO_2, ...SLO_o\}$ be the set of SLOs in an SLA such that $SLO_j \in SLO_s$ and $1 \leq j \leq o$.
$M_s = \{M_1, M_2, ...M_m\}$ be the set of metrics in an SLA such that $M_l \in M_s$ and $1 \leq l \leq m$.
After successful negotiation process, all SLOs and metrics have agreed values along with guarantees and obligations. A violation in the service term is determined by comparing the actual monitored data with the value agreed in an SLA. However, as discussed in above sections, reporting each violation in real time may choke the network bandwidth and holding back critical violations may result in financial, business and/or technical losses. So, it is important that each critical violation event must be reported immediately and a violation event of lesser significance may be reported at a later stage to optimize the communications between a site and the coordinator. To achieve this goal, a partial violation level is assigned to each metric. Each SLO is assigned a minimum number of violations E_j that must occur before the information about violations is communicated to the monitoring coordinator such that $E_j \geq 1$ denotes the minimum number of violations for the SLO_j. Intuitively, an SLO_j with higher partial violation value has lesser E_j and vice versa. Laterally, a violation calculation method is applied to all monitor-able metrics as described in the following:

- Let V_{kw} be the unacceptable value of a metric M_k and V_{kr} be the routine value of the metric M_j such that V_{kr} is better than V_{kw}. Routine value in this context means the value that is generally observed for a metric when the cloud service is running in normal condition.

- The range between V_{kw} and V_{kr} for M_k is marked at p number of distinct points and a partial violation value $PV_n \in PV_s$ (where $PV_s = \{PV_1, PV_2, ..., PV_p\}$, $1 \leq n \leq p$ and $PV_n \in [0, 1]$) is assigned to each point. PV_1 is the value at V_{kw} and PV_p is the value at V_{kr} such that $PV_n < PV_{n-1}$ if the metric is a directly proportional metric. Otherwise $PV_n > PV_{n-1}$, $PV_1 = V_{kr}$ and $PV_p = V_{kw}$, i.e. V_{kw} has higher partial violation value than V_{kr}. Semantically, it represents that as we move closer to the final agreed value of a metric the partial violation value increases. This mapping of the metric values to the partial violation values results in a hash table where each possible value of a metric (key) has its partial violation value.

- The actual observed value of the metric is compared with the partial violation values table and a PV_n is selected based on the corresponding value in the table.

- Each new value of PV_n is added to the previous cumulative sum for that metric and when sum is greater than *1* then one violation is marked for the related SLO.

Above method of assigning the partial violation values to a metric works as a controllable environment where violations are automated to be aggregated until E_j is reached.
The number of communications using the partial violation method is derived as given in the following:

- Let k sites/locations have same number of (total o) SLOs for one cloud SLA and SLO_{ij} represents the $j-th$ SLO at site S_i;

- Let E_{ij} represents the minimum number of violations threshold for $j-th$ SLO at site S_i;

- Let T_{ij} represents the total number of times the threshold E_{ij} is reached for $j-th$ SLO at site S_i;

- Then total number of communications from all of the locations to the monitoring coordinator are $\mathcal{O}(\sum_{i=1}^{k}\sum_{j=1}^{o} T_{ij})$.

Example 5.3.1

A site is monitoring a cloud service and the SLA for that cloud service contains three SLOs ($\{SLO_1, SLO_2, SLO_3\}$) with minimum number violations for each SLO being $\{3, 5, 10\}$, respectively. The SLO_1 contains one metric M_1 with routine value of 99.9 and worst value (below agreed value) of 99.5. Partial violation values are assigned to the M_1 as shown in Table 5.1(a). As monitoring starts,

M_1 value	Partial violation value
99.9	0.00
99.8	0.01
99.7	0.02
99.6	0.08
99.5	1

(a) Metric value to partial violation mapping

SLO ID	Minimum number of violations
SLO_1	3
SLO_2	5
SLO_3	10

(b) Minimum number of violations for each metric

Table 5.1: Example data

the monitored value for M_1 is compared with the mapping Table 5.1(a) and a partial violation value PV_1 is selected based on the monitored value. In next monitoring step, a new value for M_1 is available which is again compared with the Table 5.1(a) to find the PV_2. The PV_2 is added to the previous PV_1 value and aggregated in each monitoring step. If the aggregated $PV \geq 1$ then one violation is aggregated to the current total number of violations for SLO_1 as M_1 belongs to SLO_1 and aggregated partial violation value is reset to 0. If current total number of violations for SLO_1 are greater than or equal to 3 (using Table 5.1(b)) then this site reports total violations along with the required information to the coordinator. This process also enables to internally set alerts for a service with deteriorating performance level by setting appropriate partial violation values for a metric.

5.4 Implementation and Experimental Verification

The distributed monitoring presented in the above sections is based on an assumption that monitoring data is readily available at all distributed sites/locations. A prototype implementation is achieved including that assumption for the Amazon S3 cloud service. As monitoring requirements may vary from user to user based on the business objectives, so the prototype implementation includes a custom interface for Amazon S3 rather than the one provided by the Amazon. This custom implementation (using the AWS SDK for Java[1]) enables to include the S3LACC based automated monitoring as described in the following:

- A final SLA contains the agreed values, guarantees and obligations that are stored on each site/location using the S3 cloud service;
- Whenever a user at any distributed location performs a service task (e.g. upload, download or delete an object) then required KPIs are precisely noted in the custom S3 environment. For example, if a user initiates an object upload request then start time of the upload transaction, upload time and confirmation receipt time might be noted by system to use these values for comparison with the corresponding promised parameters of the agreed SLA;
- Violations are detected and reported according to the procedure described in the Section 5.3.

5.4.1 Experimental Validation Using Monitoring Simulation

The purpose of this automated monitoring process is to reduce the number of communications between a site and the monitoring coordinator. A monitoring simulation program has been written in Java to perform different experiments. The monitoring simulation

[1]https://aws.amazon.com/sdk-for-java/

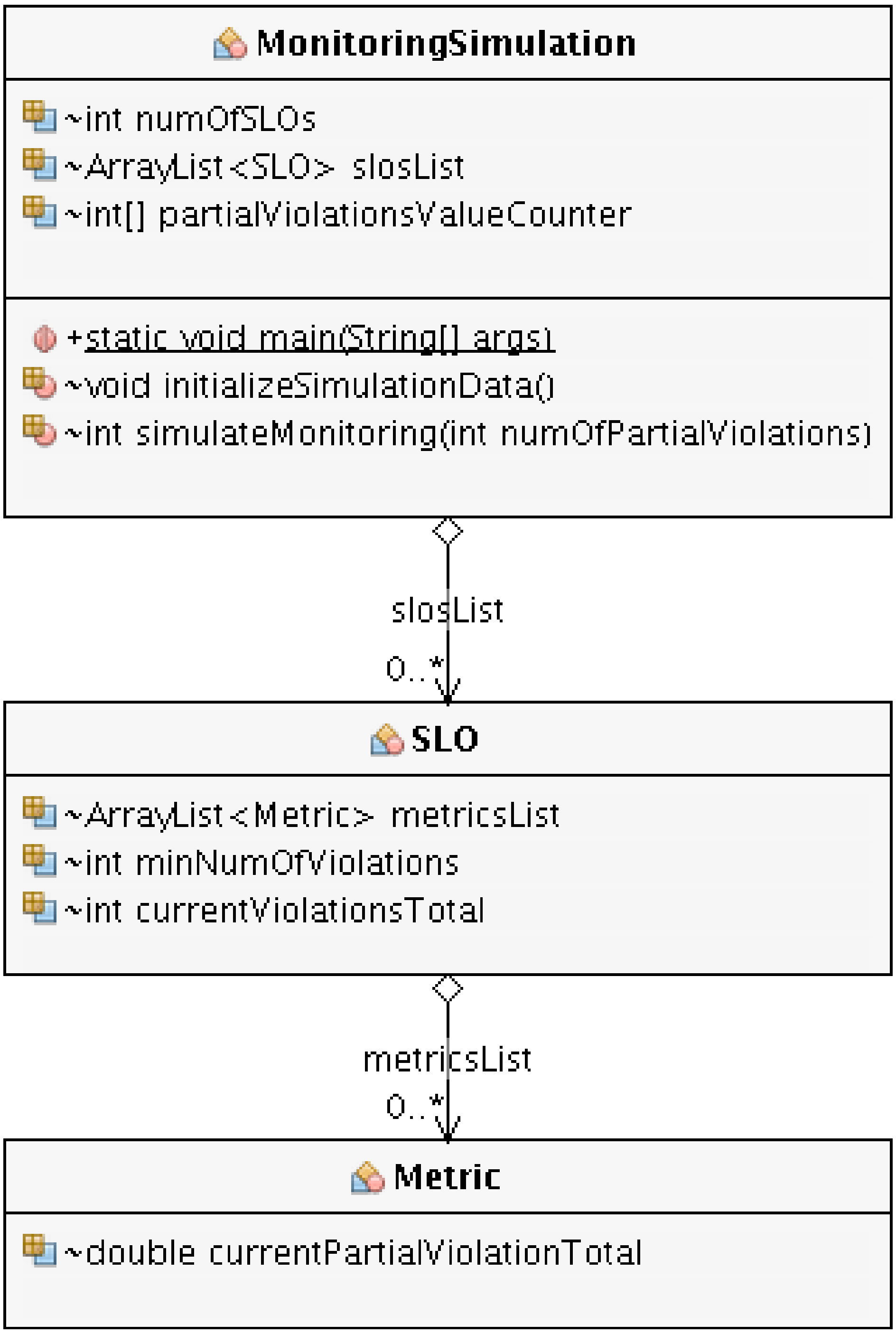

MonitoringSimulation
~int numOfSLOs
~ArrayList<SLO> slosList
~int[] partialViolationsValueCounter
+static void main(String[] args)
~void initializeSimulationData()
~int simulateMonitoring(int numOfPartialViolations)
slosList
0..*
SLO
~ArrayList<Metric> metricsList
~int minNumOfViolations
~int currentViolationsTotal
metricsList
0..*
Metric
~double currentPartialViolationTotal

Figure 5.2: UML representation of monitoring simulation setup

setup is explained in the following and a UML diagram of this setup is shown in Figure 5.2:

- A metric contains a variable *currentPartialViolationTotal* to hold the total of partial violations which initially set to *0*. As value of this variable goes beyond *1*, one violation is reported to the related SLO and variables values is decreased by *1*;
- An SLO contains a list of one to three metrics (number of metrics is random), a variable *minNumOfViolations* (generated randomly from the interval $[1, 10]$) to hold a threshold of minimum number of violations that must occur before communicating violations to the monitoring coordinator and *currentViolationsTotal* to hold the current number of violations for the SLO. The variable *currentViolationsTotal* is reset to 0 after the threshold *minNumOfViolations* has been reached;
- The method *initializeSimulationData()* generates a list of SLOs which includes initialization of metrics list too;
- The *simulateMonitoring()* method takes an input parameter *numOfPartialViolations* which indicates the number of induced partial violations during the monitoring simulation process;
- In *simulateMonitoring()* method, the following process is repeated *numOfPartialViolations* number of times:
  - A SLO S is selected at random and then a metric M is selected at random for that SLO;
  - A partial violation value is randomly generated from the interval $[0, 1]$ and added to *currentPartialViolationTotal* variable of the metric M selected in previous step;
  - If *currentPartialViolationTotal* of M is greater than *1* then one violation is added to *currentViolationsTotal* of S and one communication to the monitoring coordinator is counted.

The above simulation is repeated for multiple experiments by varying the following parameters:

- The variable *numOfPartialViolations* is changed from *20* to *200* by adding *20* in each experiment which results in total of 10 experiments.
- A set of ten experiments is repeated for different number of SLOs, i.e. *4*, *8* and *20*

		Number of partial violations per interval					
SLOs	Total partial violations	[0,.2[	[.2,.4[	[.4,.6[	[.6,.8[	[.8,1]	Number of communi-cations
4	20	3	8	1	4	4	0
4	40	5	3	9	8	15	4
4	60	12	16	11	12	9	5
4	80	21	16	16	10	17	6
4	100	18	24	19	19	20	9
4	120	24	16	16	30	34	13
4	140	33	25	27	23	32	11
4	160	28	27	37	28	40	15
4	180	32	36	34	35	43	16
4	200	38	37	37	48	40	19

Table 5.2: Experiment data and results with 4 SLOs

		Number of partial violations per interval					
SLOs	Total partial violations	[0,.2[	[.2,.4[	[.4,.6[	[.6,.8[	[.8,1]	Number of communi-cations
8	20	4	6	4	2	4	0
8	40	8	6	7	6	13	3
8	60	12	8	13	8	19	6
8	80	18	18	21	12	11	8
8	100	28	13	16	22	21	11
8	120	25	24	24	29	18	12
8	140	35	35	29	23	18	15
8	160	27	30	42	29	32	17
8	180	32	25	42	42	39	23
8	200	33	36	42	43	46	25

Table 5.3: Experiment data and results with 8 SLOs

The Table 5.2, Table 5.3 and Table 5.4 shows the data for three sets of ten experiments using 4, 8 and 20 SLOs, respectively. The

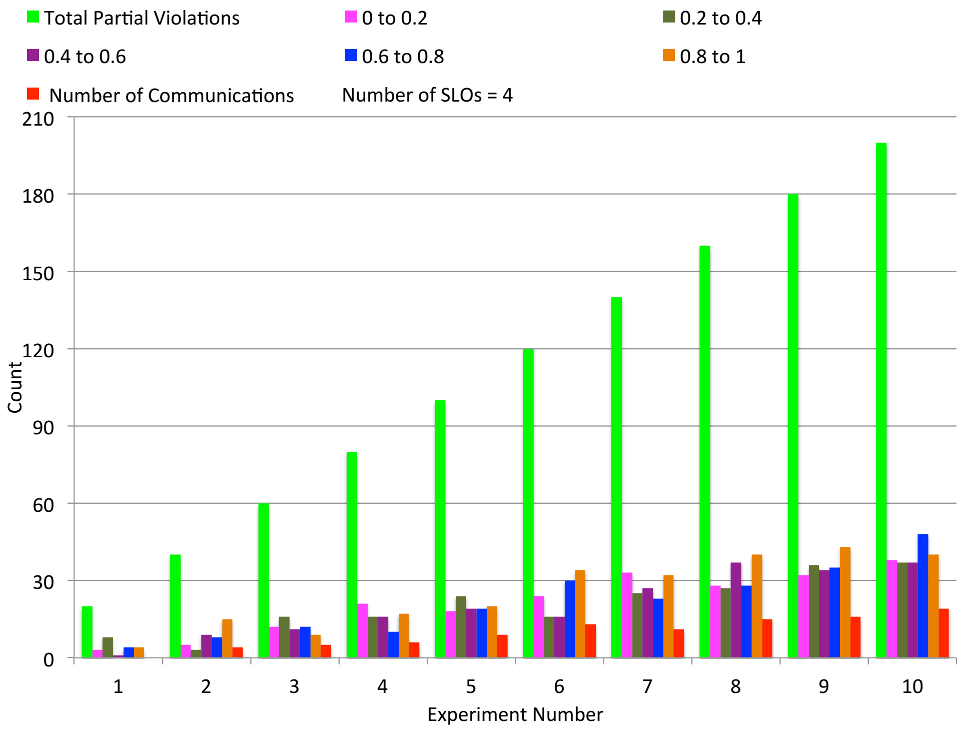


Figure 5.3: Experiment to evaluate the proposed method for number of communications for 4 SLOs

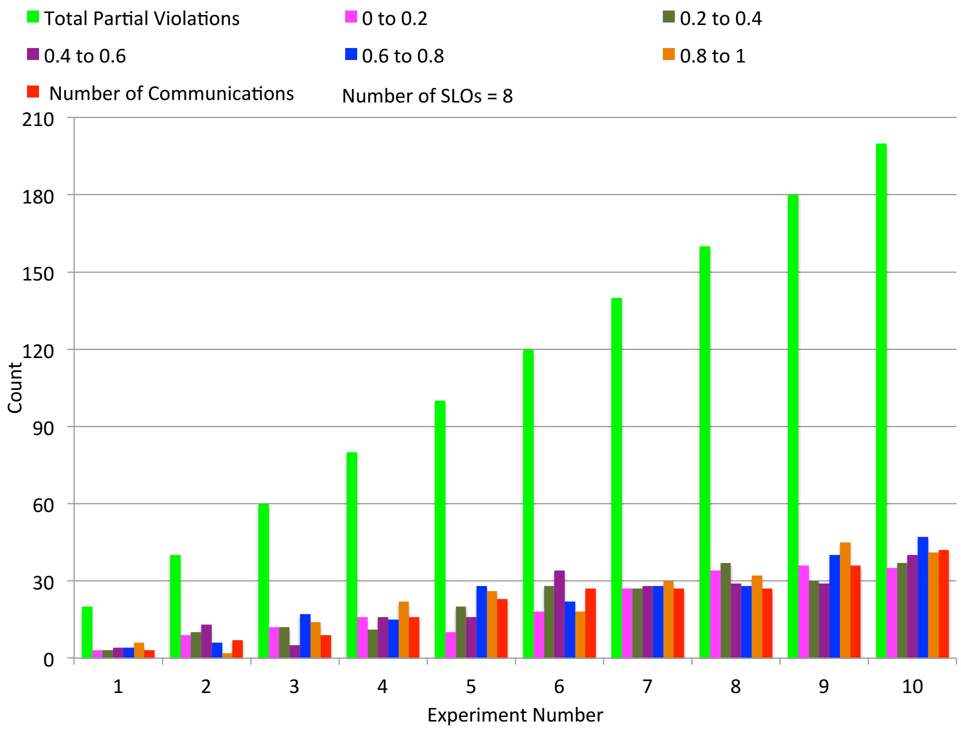


Figure 5.4: Experiment to evaluate the proposed method for number of communications for 8 SLOs

		Number of partial violations per interval					
SLOs	Total partial violations	[0,.2[	[.2,.4[	[.4,.6[	[.6,.8[	[.8,1]	Number of communi-cations
20	20	3	3	4	4	6	3
20	40	9	10	13	6	2	7
20	60	12	12	5	17	14	9
20	80	16	11	16	15	22	16
20	100	10	20	16	28	26	23
20	120	18	28	34	22	18	27
20	140	27	27	28	28	30	27
20	160	34	37	29	28	32	27
20	180	36	30	29	40	45	36
20	200	35	37	40	47	41	42

Table 5.4: Experiment data and results with 20 SLOs

merged column *Number of partial violations per interval* in this table represents the distribution of partial violation values in distinct equal intervals, e.g. the interval $[0, .2[$ contains the number partial violation values that range from 0 up to 0.2 (excluding .2). The *Number of communications* column in Table 5.2 contains the total number of communications to the monitoring coordinator as a result of SLA violations. The data from Table 5.2 is represented graphically in Figure 5.3. Similarly, Figure 5.4 and Figure 5.5 illustrate the results of two sets of experiments for 8 and 20 number of SLOs, respectively.

Experimental results validate the usefulness of the monitoring approach presented in above sections such that total number of communications with the monitoring coordinator are fully customizable to suit the specific functional requirements with respect to different locations, SLOs and metrics. Increasing the number of SLOs has an impact on total number of communications as shown in Figure 5.4 and Figure 5.5. A communication to the monitoring coordinator at one location is totally independent from communications from other locations making this approach suitable for distributed environments. The number of communications are dependent on violation type, i.e. higher partial violation values cause more communications. This method of controlled service violation

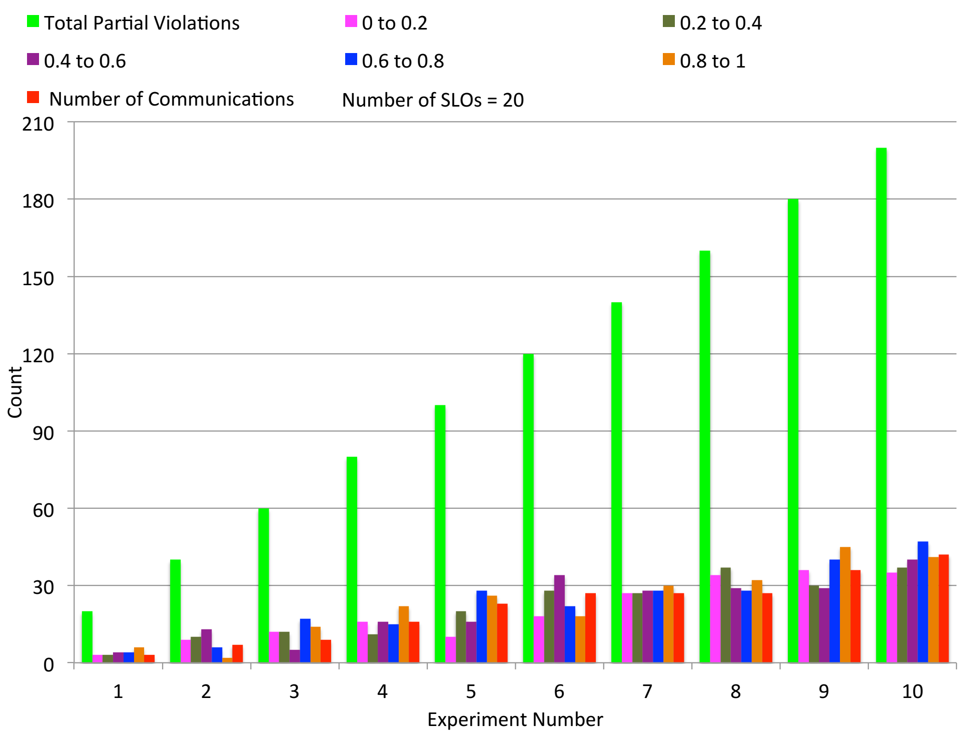


Figure 5.5: Experiment to evaluate the proposed method for number of communications for 20 SLOs

reporting eliminates the problem of randomness as described in the basic solution of the countdown problem in Section 5.2 and is based on the concrete of accurately reporting the service violations that should be reported.

5.5 Summary

In this chapter, a distributed monitoring method is presented for cloud SLAs of such cloud services which are used in different locations and communication of service violations is an important task with respect to precise total number of communications to the monitoring coordinator. The method presented in this chapter not only accurately reports the service violations but also facilitates the coordinator to set different alert levels for different metrics. A prototype is implemented in Java for Amazon S3 cloud service to illustrate the practicability of this method. Experiments enforce the worthiness of this monitoring model.

Note: The distributed monitoring of the cloud SLAs presented in this chapter has been submitted for review at The 11th IEEE International Symposium on Service-Oriented System Engineering (SOSE 2017).

Chapter 6

Implementation, Comparative Analysis and Conclusions

6.1 Introduction

In previous chapters, SLA specification, negotiation, monitoring and management phases are focused individually. This chapter provides implementation information of the combined framework for the complete SLA life cycle automation. Validation of individual phases is provided along with theoretical description or modeling in their respective chapters, i.e. the presented SLA specification (S3LACC given in Chapter 3) is demonstrated with the help of a use case, usefulness of the negotiation strategy is validated by multiple experiments and practicality of the monitoring approach is tested by simulating SLA violations. The SLA management phase is briefly covered in the monitoring phase where any SLA violation is combined with a resulting action. The SLA specification presented in this work is easily extensible due to core SLA elements distinctively structured to meet the specific requirements of the cloud services. Relationship among core elements (i.e. SLA, SLO and metric) is maintained to accommodate custom requirements of different SLAs for the cloud services by extending the basic structure (S3LACC). In Section 6.2, an implementation of S3LACC in Java language is given with explanation about extending the specification for custom requirements. Section 6.3 gives a comparative analysis of features introduced in this work and existing approaches. Finally, Section 6.4 concludes this work with clear contributions and future directions.

6.2 Implementation

In this section, implementation of S3LACC is given in Java language with overall functioning of the complete SLA life cycle implementation using negotiation and monitoring techniques proposed in this work. Figure 6.1 and 6.2 give UML (unified modeling language) representation of the Java classes using S3LACC. Beside

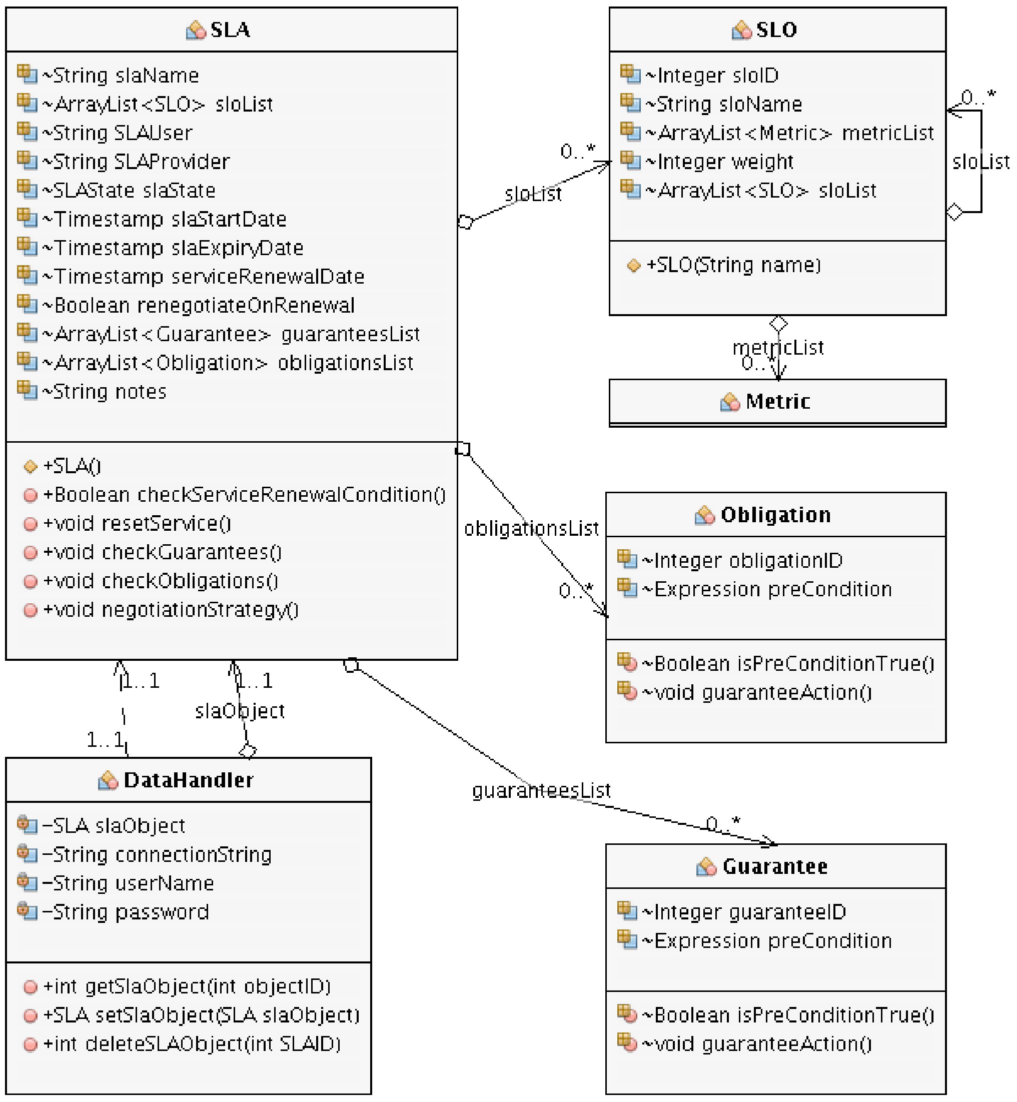


Figure 6.1: UML diagram of SLA Structure in S3LACC (Metric class is collapsed in this diagram due to space limitation, which is expanded in next Figure)

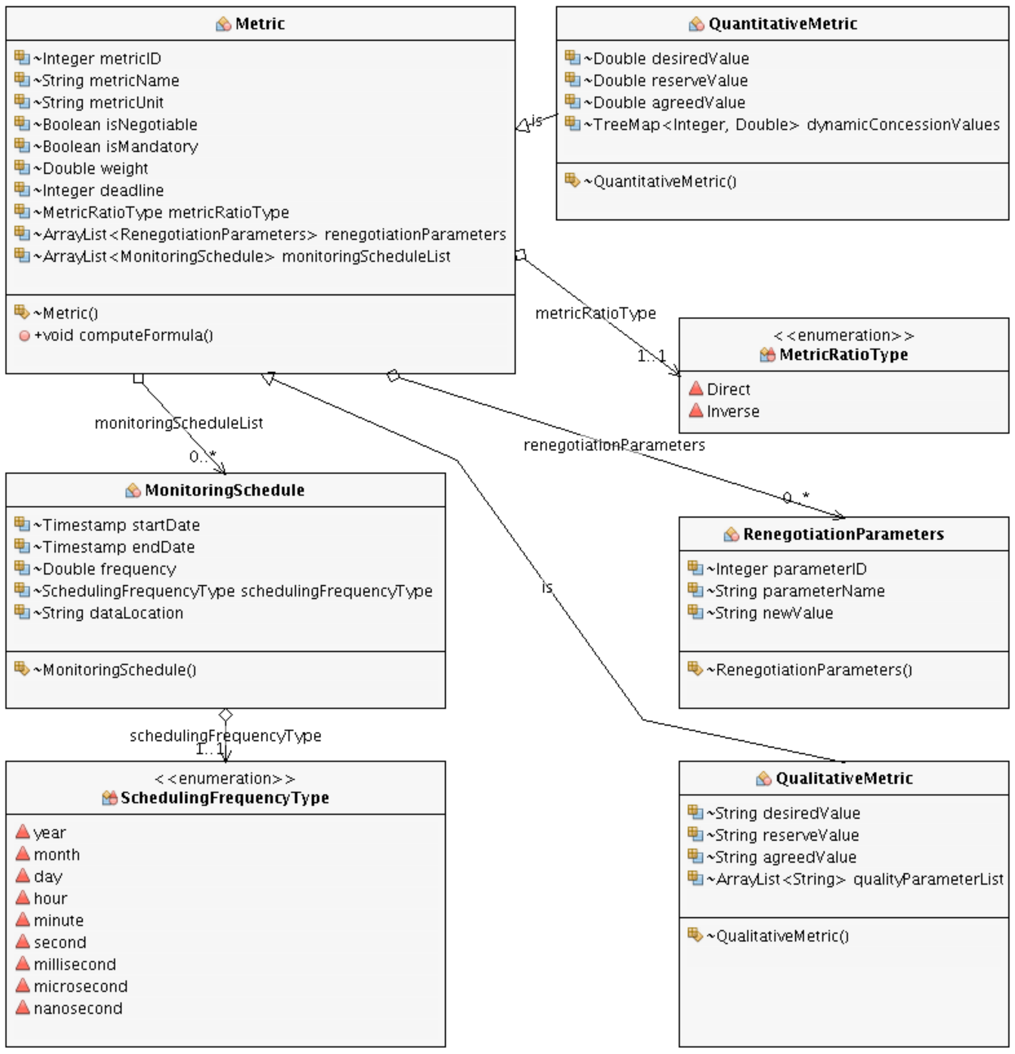


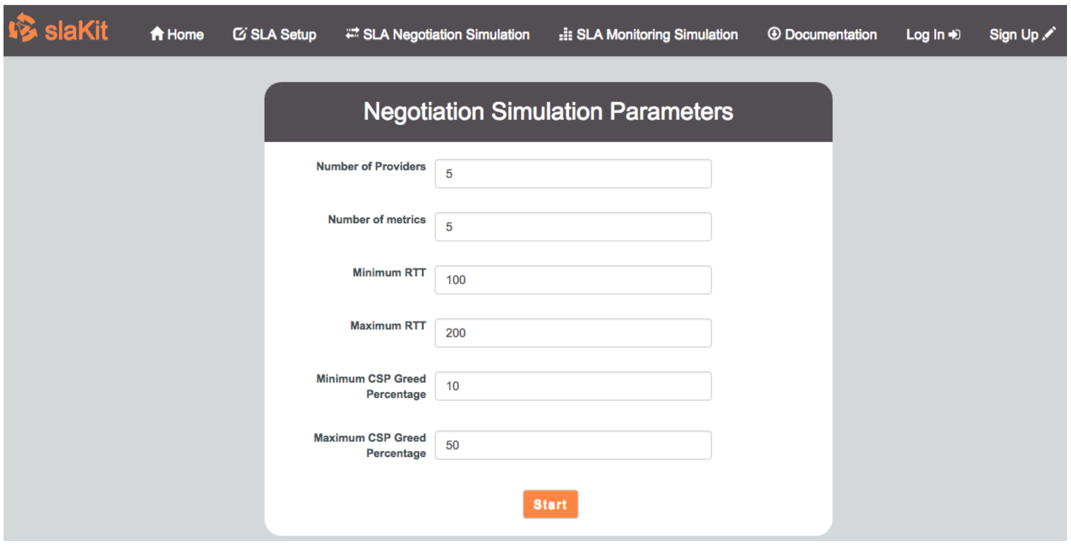
Figure 6.2: UML diagram of SLA Structure in S3LACC with metric and related classes expanded

basic elements of the SLA and methods (in Figure 6.1) as described in Chapter 3, the SLA class contains two methods (*checkGuarantees()* and *checkObligations()*) which continuously wait for triggering events based on the monitoring output. The *guaranteeAction* method of the related *Guarantee* object is performed as its related guarantee condition is evaluated to true. A *guaranteeAction* can implement SLA management activities, e.g., collecting required supporting documents from the specified locations and sending to the service provider for service credits. The *preCondition* in *Guarantee* and *Obligation* classes are implemented as Java's EL

(Expression Language) library which evaluate to true or false and may include input parameters. In Figure 6.2, the *QualitativeMetric* and *QuantitativeMetric* classes extend basic *Metric* class to implement the required functionality of qualitative and quantitative metrics, respectively. A monitoring schedule is assigned to each metric so that different metrics may have varying monitoring frequencies and data sources.

6.2.1 Prototype

A web based prototype is implemented in Java to demonstrate complete functioning of the proposed specification, negotiation and monitoring strategies. The prototype implements the following



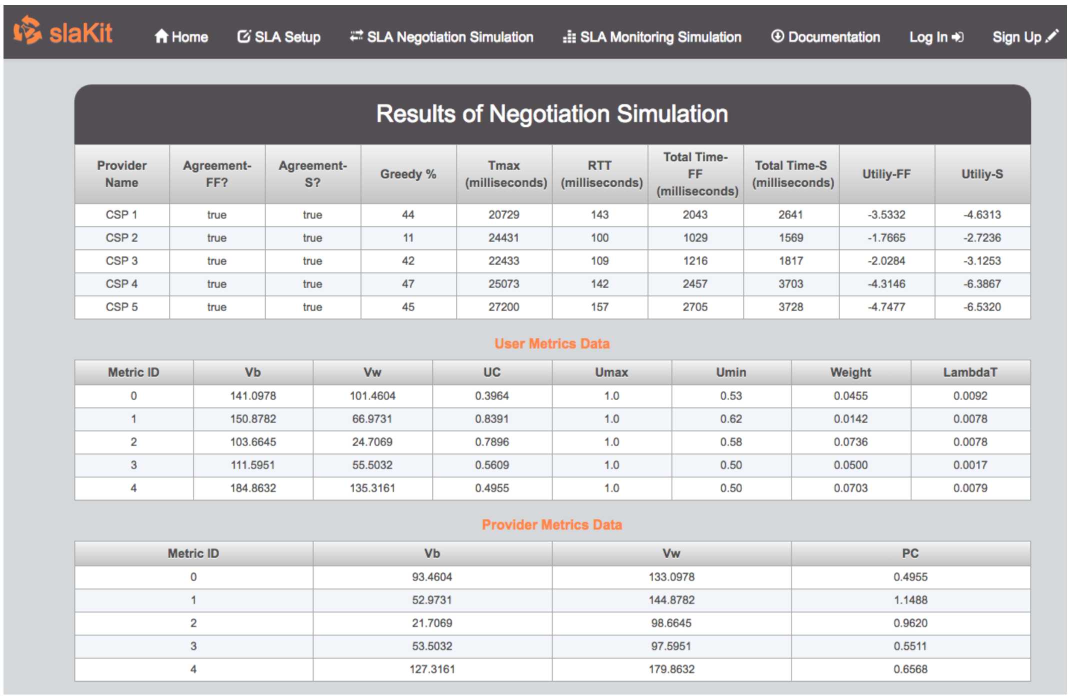
Negotiation Simulation Parameters

Number of Providers	5
Number of metrics	5
Minimum RTT	100
Maximum RTT	200
Minimum CSP Greed Percentage	10
Maximum CSP Greed Percentage	50

Figure 6.3: Web interface for the negotiation simulation experiments to compare the flip-flop negotiation strategy with a simple negotiation strategy.

tasks:

- A secure user login system is maintained to create, store and retrieve the SLAs for testing purposes only.
- A user (after successful login) is able to create/store an SLA in the specified structure as described above. The *DataHandler* class (in Figure 6.1) is used to store and retrieve SLA objects. An SLA object is stored in database by serializing it to a byte



Results of Negotiation Simulation

Provider Name	Agreement-FF?	Agreement-S?	Greedy %	Tmax (milliseconds)	RTT (milliseconds)	Total Time-FF (milliseconds)	Total Time-S (milliseconds)	Utiliy-FF	Utiliy-S
CSP 1	true	true	44	20729	143	2043	2641	-3.5332	-4.6313
CSP 2	true	true	11	24431	100	1029	1569	-1.7665	-2.7236
CSP 3	true	true	42	22433	109	1216	1817	-2.0284	-3.1253
CSP 4	true	true	47	25073	142	2457	3703	-4.3146	-6.3867
CSP 5	true	true	45	27200	157	2705	3728	-4.7477	-6.5320

User Metrics Data

Metric ID	Vb	Vw	UC	Umax	Umin	Weight	LambdaT
0	141.0978	101.4604	0.3964	1.0	0.53	0.0455	0.0092
1	150.8782	66.9731	0.8391	1.0	0.62	0.0142	0.0078
2	103.6645	24.7069	0.7896	1.0	0.58	0.0736	0.0078
3	111.5951	55.5032	0.5609	1.0	0.50	0.0500	0.0017
4	184.8632	135.3161	0.4955	1.0	0.50	0.0703	0.0079

Provider Metrics Data

Metric ID	Vb	Vw	PC
0	93.4604	133.0978	0.4955
1	52.9731	144.8782	1.1488
2	21.7069	98.6645	0.9620
3	53.5032	97.5951	0.5511
4	127.3161	179.8632	0.6568

Figure 6.4: Results of the negotiation simulation based on the parameters provided in the Figure 6.3.

array and similarly retrieved from the database by converting the byte array back to an SLA object.

- A random SLA generator helps a user to create and populate an SLA object completely with random data. This function helps the user to quickly generate a full SLA with test data.

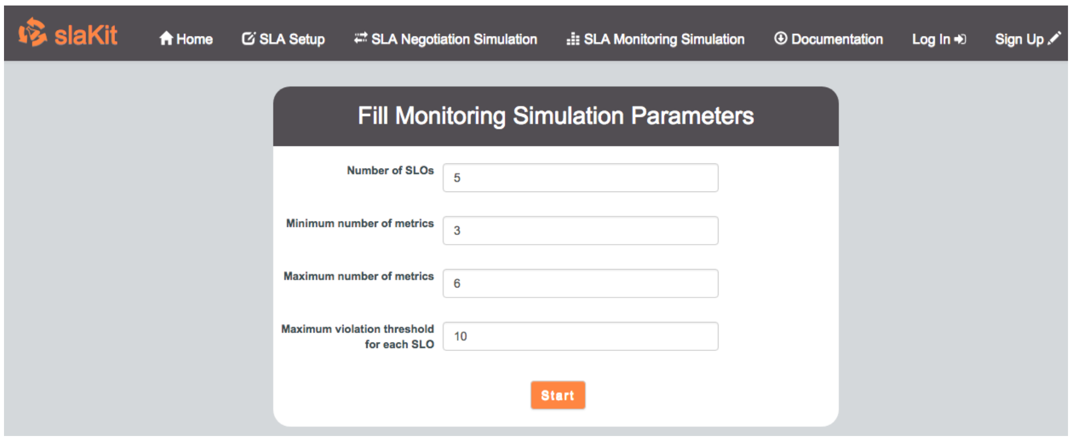
- A user is able to use an existing SLA to test the presented SLA negotiation strategy (in Chapter 4). For this purpose, the user can specify negotiation parameters with the CSU and the CSP perspective. The CSU parameters are simply defined using the *Metric* class (6.2). The user can set the negotiation parameters for the CSP as given in following:

  – Number of CSPs (for concurrent negotiations).

  – Number of metrics to be included in the negotiation process.

  – Minimum round trip time (RTT).

  – Maximum RTT.



Fill Monitoring Simulation Parameters

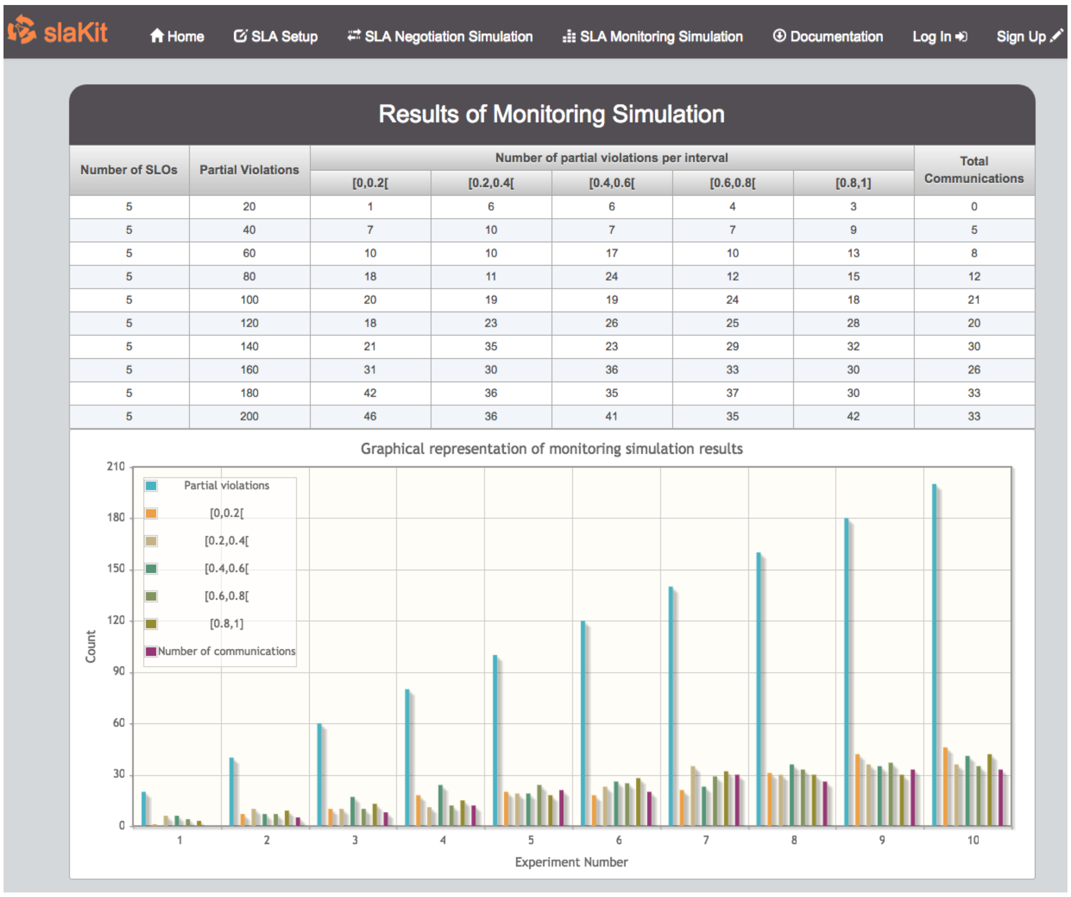
Number of SLOs	5
Minimum number of metrics	3
Maximum number of metrics	6
Maximum violation threshold for each SLO	10

Figure 6.5: Web interface for the monitoring simulation experiments to evaluate the partial violation monitoring method.

- Minimum CSP greed percentage, which simulates a CSP's degree of being greedy during the negotiation process, i.e. if a CSU increases (flip) the concession value to reach an early agreement then degree of greediness determines the chances that a CSP tries to take advantage of the CSU's increase in concession by reducing its own concession.
- Maximum CSP greed percentage.

The simulated negotiation service performs the automated negotiation and returns analytical information to the user. This information includes the outcome of the negotiation process with all of the CSPs, comparison of time taken using the flip-flop negotiation strategy with simple negotiation strategy (which includes greedy approach) and comparison of agreement utilities achieved with and without flip-flop negotiation strategy. This analysis is helpful for a user to evaluate the different settings in negotiation parameters and usefulness of the flip-flop negotiation strategy. Figure 6.3 shows web interface to define the negotiation simulation parameters as described above. Figure 6.4 shows the result of the simulation process as described in the following:

- First table in the Figure 6.4 shows overall result of the negotiation simulation process. In that table, the second and third columns (Agreement FF and Agreement



Results of Monitoring Simulation

Number of SLOs	Partial Violations	Number of partial violations per interval					Total Communications
		[0,0.2[	[0.2,0.4[	[0.4,0.6[	[0.6,0.8[	[0.8,1]	
5	20	1	6	6	4	3	0
5	40	7	10	7	7	9	5
5	60	10	10	17	10	13	8
5	80	18	11	24	12	15	12
5	100	20	19	19	24	18	21
5	120	18	23	26	25	28	20
5	140	21	35	23	29	32	30
5	160	31	30	36	33	30	26
5	180	42	36	35	37	30	33
5	200	46	36	41	35	42	33

Figure 6.6: Results of the monitoring simulation based on the parameters provided in the Figure 6.5.

S) show whether agreement was made or not using the flip-flop (FF) or simple (S) negotiation strategy respectively. The third column (Greedy %) shows the amount of greediness selected at random for the flip-flop negotiation simulation. The next column (Tmax) is the maximum time allowed during the negotiation process. RTT is round trip time selected at random based on the user's input parameters. The next columns (Total Time FF and Total Time S) represent the time taken to conclude the negotiation process using flip-flop (FF) and simple (S) strategy respectively. Utility-FF and Utility-S columns are the final utilities (for the flip-flop and simple negotiation strategy respectively) computed based on the user metrics data (second table in Figure 6.4). The data for the negotiation simulation process is generated randomly and is bound by limits defined by the user. It can be noted in simulation results that lesser percentage of greediness by the provider (by CSP2 in first table in the Figure 6.4) is useful to reduce the overall negotiation process time (Total Time FF) and also results in a better overall utility for both parties (i.e. CSP and CSU). It can also be noted that the flip-flop negotiation strategy (Utility-FF) has better results than the simple negotiation strategy (Utility-S). In the second table in Figure 6.4, the columns, Vb, Vw, UC, UMax, Umin, Weight, LambdaT represent the desired value, worst value, user concession, maximum utility, minimum utility, weight and depreciation factor in utility of a metric due to elapsed time respectively. More detail about these parameters is discussed in previous chapters. The second table represents the metric data that is selected in a controlled random way to be used in the negotiation process. Similarly, in the third table of Figure 6.4, Vb, Vw and PC represent the best value, worst value and provider's concession for each metric respectively. The reason of a better result for the flip-flop negotiation strategy is due to the fact that earlier agreement results in a better utility value, i.e. a lesser percentage of greediness (positive approach) from a CSP motivates the CSU to keep on applying flip steps to reach an early agreement.

- The monitoring service is simulated to enable a user to evaluate a multi-location and precise monitoring approach. A user can specify, number SLOs, minimum number of metrics in each SLO, maximum number of metrics in each SLO and maximum violation threshold for each SLO as shown in the Figure 6.5. The result of this simulation is reported as the number of communications made with the monitoring coordinator as shown in the Figure 6.6. The number of partial violations per interval (first table in the Figure 6.6) are randomly selected to distribute the partial violations in different intervals. The monitoring simulation service executes the given simulation parameters for different number of partial violations, e.g. 20, 40 and 60 etc. The last column in the table (of Figure 6.6) gives the number of communications made due to partial violations. This simulation helps the user to evaluate the presented monitoring strategy for its custom requirements.

A demo of the above prototype is part of the oral presentation of this thesis work. Implementation detail for the web based prototype is out of scope of this work.

6.3 Comparative Analysis

In this section, a comparison of existing approaches for SLA specification monitoring and management is described. First, an overall analysis is given that compares different SLA specification languages with S3LACC and its features with respect to capabilities of SLA negotiation, monitoring and description of qualitative metrics. Qualitative metrics play an important role in cloud SLAs specifically e.g. reliability is a major factor while selecting a cloud service which is a qualitative metric in general and it requires a different method of specification and negotiation than quantitative metric. Table 6.1 shows a brief feature based comparison of WSLA [12], WS-Agreement [15], SLAng [57], SLA* [16], SLALOM [58], Stamou *et al.* [59], Joshi *et al.* [60], CSLA [61], Kotsokalis *et al.* [62] and S3LACC.

In second column of the Table 6.1, target domain (original domain for which the specification was given) is mentioned, next columns show if SLA negotiation, monitoring, management and qualitative parameters are supported by the specification or not. The word *Partial* in the negotiation column represents that either negotiation parameters are partially definable or negotiation strategy is not integrated within the specification. S3LACC enables complete integration of the static and dynamic negotiation parameters.

Source	**Original domain**	**Negotiation**	**Monitoring/ Management**	**Qualitative**
WSLA	Web services	Yes (static)	Yes	No
WS-Agreement	Web services	Yes (static)	Partial	No
SLAng	Internet/web services	No	Only monitoring	No
SLA*	Domain independent	Partial	No	No
SLALOM	IT services	No	No	No
Stamou *et al.*	Cloud data services	No	No	No
Joshi *et al.*	Cloud services	Partial	Yes	No
CSLA	Cloud services	No	Yes	No
SLAC	Cloud services	Partial	Yes	No
Kotsokalis *et al.*	IT services	Yes	Yes	No
S3LACC	Cloud services	Yes	Yes	Yes

Table 6.1: Comparative analysis of S3LACC framework with other approaches

Also, S3LACC enables a user to include any custom negotiation strategy within SLA template. Another feature of S3LACC adds the capability of merging the SLA template and the final SLA as a single document. *Partial* in monitoring/management column represents that either full SLA monitoring is not supported by the specification or a customizable SLA monitoring technique is not possible to integrate using the specification. In the following sections, a detailed comparative analysis is given for the S3LACC specification and also for the presented SLA negotiation strategy (*flip-flop* negotiation) with other approaches.

6.3.1 Comparative Analysis of S3LACC with Related Work

Different languages and specifications have been proposed to represent an SLA in a machine readable format, however, as discussed by SLAC authors [18], most of these specifications are not defined specifically for the cloud services and do not fulfill specific requirements of SLAs for the cloud services e.g. scenarios involving a broker during negotiation process [18]. In [18] a comparison of different SLA specification models and languages is presented with respect to different features such as cloud domain, multi-party, broker support, business metrics, price schemes, syntax, semantics, verification, evaluation and open-source availability. S3LACC includes all of these features except verification. Apart from these features, the S3LACC extends the SLA specification with additional capabilities such as a common template for the CSU and the CSP, static/dynamic negotiation support in the SLA and automated monitoring facilitation. Moreover, S3LACC is designed according to the latest available cloud SLA standards and definitions to support the complete SLA life cycle rather than its isolated phases. All information of the complete SLA life cycle is efficiently bundled in a single SLA. The quantitative and qualitative SLA parameters are possible to be grouped in a single SLO using S3LACC. Another closer approach is presented by Stamou *et al.* [59] which describes SLAs for data services as a directed graph to represent dependencies in SLA data management flow. The SLA directed graph model by Stamou *et al.* is based on WSLA Framework. The structure defined as the graph model in [59] is different from the S3LACC structure, i.e. an *SLO* and a *service object* are different entities in their model. According to [59], a *service object* contains SLA parameters like *transaction time* or *average execution time*, whereas, *SLOs* define limits for these SLA parameters through guarantees or obligations. Kotsokalis *et al.* [62] model SLAs for service computing as binary decision diagrams (BDDs) to automate the SLA negotiation, subcontracting, optimizing the utility and SLA management. An SLA in [62] is defined in terms of *facts*, *conditions* and *clauses* which evaluate to *true* or false, hence an SLA is represented as a boolean function. This boolean function is represented as BDD to eliminate redundancies. However, as discussed in the [62], the proper recognition of *facts* requires additional attention.

6.3.2 Comparative Analysis of Flip-Flop Negotiation with Related Work

Bilateral and multilateral negotiations in context of IT services is a widely studied topic including autonomous agents for mediation and non-agent based approaches [63]. From web services [64] to auction based resource allocation systems [65], different negotiation approaches have been presented based on the negotiation protocols altered to suit the negotiation scenario [66]. The requirements for achieving an equilibrium among all participating parties and optimizing the overall utility for each participant are influenced by the negotiation model, deadline, available resources and system domain [67]. Different techniques have been applied to optimize the negotiation results including game theory (e.g. [68]) and machine learning [69]. The automated SLA negotiation for the cloud services is also an instance of general negotiation model, however with additional constraints and requirements e.g. trust among participants, information sharing about negotiation preferences and level of available resources. In practical scenarios of cloud environments, participating parties (i.e. CSU and CSPs) may not adapt a single negotiation strategy to optimize the social utility (utility level for all participants). Each participant usually strives to optimize its own utility level and negotiation strategies are not disclosed among participants. At a maximum level of information sharing, all CSPs may have same SLA template, SLOs, metrics and negotiation protocol. The same negotiation parameters (negotiation deadline, minimum concession in each negotiation round or maximum time to conclude the negotiation process) can not be enforced to all CSPs in realistic cases. The negotiation strategy presented in this work makes no assumption about the CSPs' negotiation parameters. We use polynomial interpolation to formulate the opponent's concession pattern and then use polynomial extrapolation to estimate the opponent's future offers. Many approaches has been presented to automate negotiation for SLAs, e.g. [28, 29, 31, 32, 33, 27], we discuss a few that are very close to our approach with analytical comparison.

A similar approach presented in [70] uses Gaussian Processes to estimate opponent's future behavior in the negotiation process. A time based function then estimates the best time in future when opponent's offer has maximum utility. This approach presented in [70], however, differs with ours with respect to computation of user's (agent) concession and offer generation. In our work, offer is generated based on the flip-flop process, opponent's reaction, opponent's concession and remaining time whereas in [70], offers are generated based on the target utility (calculated using time based function). A price-and-time-slot negotiation mechanism is presented in [30] to facilitate negotiations for time-slot and prices based on a utility function. Different utility functions are presented in [30] to calculate the price utility of a provider and of an agent (user). Provider's time-slot preferences are based on service demand, temporal ordering of tasks and fitting tasks whereas agent's time-slot preferences are defined using a partial function (different time slots having different utilities). In [26], an SLA negotiation approach is presented for multiple cloud providers across multiple tiers, i.e. user tier (service user and broker), service tier (service provider and service user of other resource service provider) and resource tier (resource service provider). This approach enables combining services from multiple providers and negotiating with them concurrently through different entities. Each negotiating entity in this architecture [26] is represented in Cloud Negotiation Support System (CNSS). A user initiates a request of a single service or combined services (depending on its functional requirements and business objectives) to the broker's CNSS. Then the broker or coordinating entity (CE) evaluates the request, selects appropriate service providers and creates required number of negotiating entities (NE) to start negotiation process. In service tier, a service provider may need infrastructure services from another service provider (in resource tier) to fulfill needs of its service users (in user tier). Each of these negotiation tasks is run concurrently by responsible NEs and result of concurrent negotiation process is passed to CE. Zheng *et al.* [34] present an approach for the cloud services SLA negotiation for a scenario where opponent's strategy is not well comprehended. They [34] use a mixed approach of concession and trade-off (by decreasing the utility of one issue and increasing the utility of the other issues, by keeping the overall utility same). A time-dependent SLA negotiation strategy for CSPs is presented in [36] which automatically adjusts offer depending on the utilization of resources, i.e. conceding more on free resources and lesser on the resource in demand. This approach assumes that negotiating parties are unaware of opponent's concession tactics and also investigates risk of malicious negotiation attempts under this setup, i.e. a malicious client may submit arbitrary offers to service provider to gain knowledge of provider's concession tactics and resource utilization preferences. The malicious client may use this information to acquire an actual service from the same provider faster than other clients. This scenario may cause profit loss for a provider and also deprive other clients from acquiring scarce resources in a competitive environment. On the other hand a provider may also use offer information from client's NS to maximize its profits in future transactions. The time factor discussed in our work is linked with the change in utility due to time. Another closer approach is presented in [35], where a near-optimal SLA negotiation strategy is presented for cloud computing environments. The negotiation strategy presented in [35] uses the utility function based on the weight (priority) of the issue. It is argued in this work that autonomous negotiation process is complex and computationally expensive, so a computationally inexpensive negotiation strategy is proposed in this work [35] termed as 'Reactive Exploitation' (RE). According to this negotiation strategy (RE), a party generates its bid based on a *quasi tit-for-tat* policy, i.e. a concession is given if the opponent is also observed to do the same. The negotiation process is in RE restricted by the time constraints. A party using the RE maintains the record of best counter offer received so far and latest generated bid. The RE strategy utilizes maximum time in most cases to exploit maximal possible options. An agreement is made by selecting the best acceptable bid received in the negotiation process. This approach is however different than our approach such that, we strive to make an early agreement by predicting the opponent's expected final offer and move faster towards that offer by using the flip-flop strategy.

6.3.3 Comparative Analysis of the Proposed Monitoring Approach with Related Work

Monitoring of the cloud services is a task of great importance for both service provider and service user [38]. Quality assurance is mandatory for cloud services to realize the underlying business objectives. Automatic monitoring of QoS parameters and guarantees with respect to the SLA is a much needed functionality in cloud services [39]. Multiple approaches have been presented to facilitate the automated monitoring task in cloud computing environments [40][41][42][43][44][45][46]. A detailed review of cloud service monitoring is presented by Aceto *et al.* [38] and by Hussain *et al.* [47]. LoM2HiS (Low-level resource Metrics to High-level SLAs) framework [40] is presented as a solution to map low level resource metrics (e.g. system uptime and downtime) to high level SLA parameters (e.g. availability) for detection of SLA violation threats. LoM2HiS is developed as part of FoSII (Foundations of Self-governing ICT Infrastructures) infrastructure. FoSII is divided in two core parts, i) enactor component (self management part of the deployed services) and ii) LoM2HiS (monitoring component which provides information to enactor component). It is assumed in this approach that negotiation process has already been completed and SLA is stored in repository for service provisioning. Future SLA violation threats are detected by defining tighter thresholds than real SLA objective thresholds. However, threat thresholds are assumed to be predefined in this work and no particular component is defined for this purpose. An extended version of this approach is also presented in 2011/2012 [42]. A cloud application monitoring approach (termed as mOSAIC monitoring) is presented in [41] by extending the mOSAIC API [71]. This approach [41] presents a monitoring API which offers *connectors* and *drivers* to abstract resource monitoring and acquire monitoring data, respectively. The mOSAIC monitoring also provides tools for developers to build a custom monitoring system. Oliveira *et al.* [44] present an approach to detect deviations in network SLAs named as network traffic anomaly detection engine (TADE). This approach [44] defines different components as part of the architecture to detect and communicate the SLA violations to the related systems. The important issue in SLA monitoring is the number of communications to the coordinator which is ignored in most of the available SLA monitoring approaches. Our approach for distributed SLA monitoring is developed with a main focus on reducing the number of communications in such a way that essential reporting of SLA violations is ensured.

6.4 Conclusions and Scientific Contributions

In this work, we have presented an SLA specification which is specifically designed for the cloud service. The proposed specification considers the latest available standards and specification guidelines to standardize the cloud SLAs. It has been argued by several authors that cloud SLAs require additional features to accommodate the specific requirements e.g., pricing, monitoring and qualitative parameters. The proposed specification (S3LACC) consists a core structure (with most suitable relationship among SLA elements) which is easily extensible to meet the customer specific requirements and it can also be easily modified for future changes. The S3LACC specification targets the complete SLA life cycle whereas most of the specifications lack one or other critical phase of the SLA life cycle. An SLA specification is not very beneficial if one of the SLA life cycle phase is not supported or complete features of the cloud service specific SLAs are not supported. The negotiation strategy presented in this work (*flip-flop* negotiation) enables a CSU and a CSP to conclude the negotiation process in lesser time, hence efficient use of cloud resources is ensured which is an essence of cloud computing. The *flip-flop* negotiation strategy can be easily integrated in the SLA using S3LACC and a CSU can make use of this efficient negotiation strategy without making any changes to the SLA template. Similarly, the monitoring strategy presented in this work enables distributed and continuous monitoring which can be integrated with the S3LACC easily as well. The presented monitoring approach decreases the number of communications made from different service locations towards the monitoring coordinator. Also, this monitoring approach helps a monitoring coordinator to define different monitoring parameters for different locations rather than a global monitoring strategy. The implementation of S3LACC is provided in Java language with a simulation of the *flip-flop* negotiation strategy and a prototype for analytical testing of the presented monitoring approach. The future directions of this work include the extension of S3LACC for CSP perspective and to design a negotiation strategy that enables off-line negotiations to reduce the number of round trips between a CSU and a CSP during the negotiation process. These extensions require special considerations with respect to security and privacy issues as well.

References

[1] H. Takabi, J. B. D. Joshi, and G. J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security and Privacy*, vol. 8, pp. 24–31, Nov 2010.

[2] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl, "Scientific cloud computing: Early definition and experience," in *10th IEEE International Conference on High Performance Computing and Communications*, pp. 825–830, Sept 2008.

[3] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, pp. 50–55, Jan 2009.

[4] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *10th IEEE International Conference on High Performance Computing and Communications*, pp. 5–13, Sept 2008.

[5] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop*, pp. 1–10, Nov 2008.

[6] E. Walker, "Benchmarking amazon ec2 for high-performance scientific computing," *The Magazine of USENIX and SAGE*, vol. 33, no. 5, pp. 18–23, 2008.

[7] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing

for scientific workflows," in *IEEE Fourth International Conference on eScience*, pp. 640–645, Dec 2008.

[8] P. M. Mell and T. Grance, "SP 800-145. The NIST definition of cloud computing," tech. rep., National Institute of Standards & Technology, Gaithersburg, MD, United States, Oct 2011.

[9] P. Bianco, G. Lewis, and P. Merson, "Service level agreements in service-oriented architecture environments," tech. rep., Software Engineering Institute, Carnegie Mellon University, 2008.

[10] L. Wu and R. Buyya, "Service level agreement (SLA) in utility computing systems," *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions, IGI Global*, pp. 1–25, July 2011.

[11] G. Conway and E. Curry, "Managing cloud computing-a life cycle approach," in *CLOSER*, pp. 198–207, Apr 2012.

[12] A. Keller and H. Ludwig, "The WSLA framework: Specifying and monitoring service level agreements for web services," *Journal of Network and Systems Management*, vol. 11, pp. 57–81, Mar 2003.

[13] D. D. Lamanna, J. Skene, and W. Emmerich, "SLAng: a language for defining Service Level Agreements," in *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems, 2003. FTDCS 2003. Proceedings.*, pp. 100–106, May 2003.

[14] J. Skene, D. D. Lamanna, and W. Emmerich, "Precise service level agreements," in *26th International Conference on Software Engineering*, pp. 179–188, May 2004.

[15] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (Ws-Agreement)," in *Open Grid Forum*, vol. 128, p. 216, 2007.

[16] K. T. Kearney, F. Torelli, and C. Kotsokalis, "SLA*: An abstract syntax for Service Level Agreements," in *11th IEEE/ACM International Conference on Grid Computing*, pp. 217–224, Oct 2010.

[17] Y. Kouki, F. A. d. Oliveira, S. Dupont, and T. Ledoux, "A language support for cloud elasticity management," in *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 206–215, May 2014.

[18] R. B. Uriarte, F. Tiezzi, and R. D. Nicola, "SLAC: A formal Service-Level-Agreement language for cloud computing," in *Proceedings of the IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pp. 419–426, Dec 2014.

[19] A. Maarouf, A. Marzouk, and A. Haqiq, "A review of SLA specification languages in the cloud computing," in *10th International Conference on Intelligent Systems: Theories and Applications*, Oct 2015.

[20] D. Battr, F. M. T. Brazier, K. P. Clark, M. Oey, A. Papaspyrou, O. Wldrich, P. Wieder, and W. Ziegler, "A proposal for ws-agreement negotiation," in *11th IEEE/ACM International Conference on Grid Computing*, pp. 233–241, Oct 2010.

[21] A. E. Thijs Metsch, "Open cloud computing interface-infrastructure," in *Standards Track, no. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN)*, Apr 2010.

[22] R. B. Bohn, J. Messina, F. Liu, J. Tong, and J. Mao, "Nist cloud computing reference architecture," in *2011 IEEE World Congress on Services*, pp. 594–596, Jul 2011.

[23] R. B. Uriarte, *Supporting Autonomic Management of Clouds: Service-Level-Agreement, Cloud Monitoring and Similarity Learning.* PhD thesis, IMT Institute for Advanced Studies, Lucca, Italy, Mar 2015.

[24] A. Rubinstein, “Perfect equilibrium in a bargaining model,” *Econometrica*, vol. 50, pp. 97–109, Jan 1982.

[25] N. Matos, C. Sierra, and N. R. Jennings, “Determining successful negotiation strategies: an evolutionary approach,” in *International Conference on Multi Agent Systems*, pp. 182–189, Jul 1998.

[26] M. Siebenhaar, T. A. B. Nguyen, U. Lampe, D. Schuller, and R. Steinmetz, *8th International Workshop on Economics of Grids, Clouds, Systems, and Services (Revised Selected Papers)*, ch. Concurrent Negotiations in Cloud-Based Systems, pp. 17–31. Nov 2012.

[27] A. F. M. Hani, I. V. Paputungan, and M. F. Hassan, “Renegotiation in service level agreement management for a cloud-based system,” *ACM Computing Surveys*, vol. 47, pp. 51:1–51:21, Apr 2015.

[28] V. Stantchev, C. Schröpfer, and D. Petcu, “Negotiating and enforcing qos and SLAs in grid and cloud computing,” in *4th International Conference on Advances in Grid and Pervasive Computing*, pp. 25–35, May 2009.

[29] A. V. Dastjerdi and R. Buyya, “An autonomous reliability-aware negotiation strategy for cloud computing environments,” in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 284–291, May 2012.

[30] S. Son and K. M. Sim, “A price- and-time-slot-negotiation mechanism for cloud service reservations,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, pp. 713–728, June 2012.

[31] S. Son and S. C. Jun, “Negotiation-based flexible sla establishment with SLA-driven resource allocation in cloud computing,” in *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 168–171, May 2013.

[32] L. Wu, S. K. Garg, R. Buyya, C. Chen, and S. Versteeg, “Automated SLA negotiation framework for cloud computing,” in *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 235–244, May 2013.

[33] A. Cuomo, G. Di Modica, S. Distefano, A. Puliafito, M. Rak, O. Tomarchio, S. Venticinque, and U. Villano, “An SLA-based broker for cloud infrastructures,” *Journal of Grid Computing*, vol. 11, pp. 1–25, Oct 2013.

[34] X. Zheng, P. Martin, K. Brohman, and L. D. Xu, “Cloud service negotiation in internet of things environment: A mixed approach,” *IEEE Transactions on Industrial Informatics*, vol. 10, pp. 1506–1515, May 2014.

[35] E. Yaqub, R. Yahyapour, P. Wieder, C. Kotsokalis, K. Lu, and A. I. Jehangiri, “Optimal negotiation of service level agreements for cloud-based services through autonomous agents,” in *IEEE International Conference on Services Computing*, pp. 59–66, June 2014.

[36] A. V. Dastjerdi and R. Buyya, “An autonomous time-dependent SLA negotiation strategy for cloud computing,” *The Computer Journal*, pp. 1–15, July 2015.

[37] A. S. Y. Dirkzwager, M. J. C. Hendrikx, and J. R. De Ruiter, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, ch. TheNegotiator: A Dynamic Strategy for Bilateral Negotiations with Time-Based Discounts, pp. 217–221. Springer Berlin Heidelberg, Apr 2013.

[38] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, “Cloud monitoring: A survey,” *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, 2013.

[39] P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, *Service level agreements for cloud computing.* Springer Science & Business Media, 2011.

[40] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar, "Low level metrics to high level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments," in *2010 International Conference on High Performance Computing and Simulation*, pp. 48–54, June 2010.

[41] M. Rak, S. Venticinque, T. Máhr, G. Echevarria, and G. Esnal, "Cloud application monitoring: The mOSAIC approach," in *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 758–763, Nov 2011.

[42] V. C. Emeakaroha, M. A. Netto, R. N. Calheiros, I. Brandic, R. Buyya, and C. A. D. Rose, "Towards autonomic detection of SLA violations in cloud infrastructures," *Future Generation Computer Systems*, vol. 28, no. 7, pp. 1017–1029, 2012. Special section: Quality of Service in Grid and Cloud Computing.

[43] A. A. Falasi, M. A. Serhani, and R. Dssouli, "A model for multi-levels sla monitoring in federated cloud environment," in *IEEE 10th International Conference on Ubiquitous Intelligence & Computing and IEEE 10th International Conference on Autonomic & Trusted Computing*, pp. 363–370, Dec 2013.

[44] A. C. Oliveira, H. Chagas, M. Spohn, R. Gomes, and B. J. Duarte, "Efficient network service level agreement monitoring for cloud computing systems," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, June 2014.

[45] A. G. García, I. B. Espert, and V. H. García, "SLA-driven dynamic cloud resource management," *Future Generation Computer Systems*, vol. 31, pp. 1–11, Feb 2014. Special Section: Advances in Computer Supported Collaboration: Systems and Technologies.

[46] H. Ludwig, K. Stamou, M. Mohamed, N. Mandagere, B. Langston, G. Alatorre, H. Nakamura, O. Anya, and

A. Keller, “rSLA: Monitoring SLAs in dynamic service environments,” in *13th International Conference on Service-Oriented Computing*, pp. 139–153, Nov 2015.

[47] W. Hussain, F. K. Hussain, and O. K. Hussain, “Maintaining trust in cloud computing through SLA monitoring,” in *21st International Conference on Neural Information Processing*, pp. 690–697, Nov 2014.

[48] S. Zhang and M. Song, “An architecture design of life cycle based SLA management,” in *Proceedings of the 12th International Conference on Advanced Communication Technology*, pp. 1351–1355, Feb 2010.

[49] F. Faniyi and R. Bahsoon, “A systematic review of service level management in the cloud,” *ACM Comput. Surveys*, vol. 48, pp. 43:1–43:27, Dec. 2015.

[50] F. Moscato, R. Aversa, B. D. Martino, T. F. Forti, and V. Munteanu, “An analysis of mOSAIC ontology for cloud resources annotation,” in *Federated Conference on Computer Science and Information Systems*, pp. 973–980, Sept 2011.

[51] M. Rak, R. Aversa, S. Venticinque, and B. Di Martino, *User Centric Service Level Management in mOSAIC Applications*, pp. 106–115. 2012.

[52] A. Maarouf, A. Marzouk, and A. Haqiq, “Practical modeling of the sla life cycle in cloud computing,” in *2015 15th International Conference on Intelligent Systems Design and Applications (ISDA)*, pp. 52–58, Dec 2015.

[53] R. Sturm, W. Morris, and M. Jander, *Foundations of Service Level Management.* Sams publishing, Apr 2000.

[54] S. Frey, C. Reich, and C. Lüthje, “Key performance indicators for cloud computing SLAs,” in *The Fifth International Conference on Emerging Network Intelligence*, pp. 60–64, Sep 2013.

[55] J. Dang and M. N. Huhns, "An extended protocol for multiple-issue concurrent negotiation," in *The National Conference on Artificial Intelligent*, vol. 20, pp. 65–70, July 2005.

[56] G. Cormode, "The continuous distributed monitoring model," *SIGMOD Record*, vol. 42, pp. 5–14, May 2013.

[57] D. D. Lamanna, J. Skene, and W. Emmerich, "Specification language for service level agreements," *EU IST*, vol. 34069, 2003.

[58] A. Correia, V. Amaral, *et al.*, "Slalom: a language for SLA specification and monitoring," *arXiv preprint arXiv:1109.6740*, 2011.

[59] K. Stamou, V. Kantere, J.-H. Morin, and M. Georgiou, "A SLA graph model for data services," in *Proceedings of the Fifth International Workshop on Cloud Data Management*, pp. 27–34, Oct 2013.

[60] K. P. Joshi and C. Pearce, "Automating cloud service level agreements using semantic technologies," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pp. 416–421, IEEE, 2015.

[61] Y. Kouki and T. Ledoux, "Csla: a language for improving cloud sla management," in *International Conference on Cloud Computing and Services Science, CLOSER 2012*, pp. 586–591, 2012.

[62] C. Kotsokalis, R. Yahyapour, and M. A. Rojas Gonzalez, "Modeling service level agreements with binary decision diagrams," in *Service-Oriented Computing: 7th International Joint Conference, ICSOC-ServiceWave Proceedings*, pp. 190–204, Nov 2009.

[63] X. Zheng, P. Martin, and K. Brohman, "Cloud service negotiation: Concession vs. tradeoff approaches," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '12, pp. 515–522, May 2012.

[64] S. Paurobally, V. Tamma, and M. Wooldrdige, "A framework for web service negotiation," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 2, pp. 14:01–14:23, Nov 2007.

[65] "Efficient mechanisms for the supply of services in multi-agent environments," *Decision Support Systems*, vol. 28, no. 12, pp. 5–19, 2000.

[66] N. Jennings, P. Faratin, A. Lomuscio, S. Parsons, M. Wooldridge, and C. Sierra, "Automated negotiation: Prospects, methods and challenges," *Group Decision and Negotiation*, vol. 10, no. 2, pp. 199–215, 2001.

[67] J. S. Rosenschein and G. Zlotkin, "Designing conventions for automated negotiation," *AI magazine*, vol. 15, no. 3, pp. 29–46, 1994.

[68] K. Binmore and N. Vulkan, "Applying game theory to automated negotiation," *NETNOMICS*, vol. 1, no. 1, pp. 1–9, 1999.

[69] J. R. Oliver, "A machine-learning approach to automated negotiation and prospects for electronic commerce," *Journal of Management Information Systems*, vol. 13, no. 3, pp. 83–112, 1996.

[70] C. R. Williams, V. Robu, E. H. Gerding, and N. R. Jennings, "Using gaussian processes to optimise concession in complex negotiations against unknown opponents," 2011.

[71] D. Petcu, C. Craciun, and M. Rak, "Towards a cross platform cloud api," in *1st International Conference on Cloud Computing and Services Science*, pp. 166–169, 2011.