ABSTRACT

Title of thesis: Semantically Rich Framework to Automate Knowledge Extraction from Cloud Service Level Agreement
Master's Thesis

Divya Natolana Ganapathy, M.S. CMSC , 2020

Thesis directed by: Dr. Karuna P. Joshi
Department of Information Science

Consumers evaluate the performance of their cloud-based services by monitoring the Service Level Agreements (SLA) that list the service terms and metrics agreed with the service providers. Current Cloud SLAs are documents that require significant manual effort to parse and determine if providers meet the SLAs. Moreover, due to the lack of standardization, providers differ in the way they define the terms and metrics, making it more difficult to ensure continuous SLA monitoring. We have developed a novel framework to significantly automate the process of extracting knowledge embedded in cloud SLAs and representing it in a semantically rich Ontology. Our framework captures the key terms, standards,remedies for non-compliance and roles and responsibilities, in the form of deontic statements and their actors from cloud SLAs. Its mostly built on major cloud SLAs, but could be adapted to other domains as well. In this thesis 'Semantically rich framework to automate knowledge extraction from cloud SLA' , we discuss the challenges in automating cloud services management and how we address these with our framework.

Semantically Rich Framework to Automate Knowledge Extraction from Cloud Service Level Agreement

by

Divya Natolana Ganapathy

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Master of Science
2020

Acknowledgments

I express my gratitude to my advisor Dr. Karuna Joshi for guiding me through the process of my thesis and for providing me with the resource to perform the experiments. I thank her for allowing me to express my thoughts, ideas and build an immense interest for research. I thank Dr. Tim Finin and Dr. Yelena Yesha for reviewing my thesis. I thank all members of CARTA, KnACC and Ebiquity for giving me a platform to learn and make this thesis possible. I thank my husband Jatin Shetty for the constant help and support through this Journey and My parents Mr. Ganapathy and Mrs Shanthy Ganapathy for their guidance and blessings.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Cloud service Provider Service Level Agreements

Organizations are increasingly migrating their data to Cloud-based services to take advantage of Cloud features like rapid provisioning, scalability, ease of use, cost savings, high availability, and platform independence. Cloud service is typically accompanied by a service level agreement (SLA) which defines the minimal guarantees that a provider offers to its customers [1]. During the service negotiation phase [2] providers and consumers agree on the service performance metrics and data security and privacy policies, that are included in the SLA contracts. Components of a service could be procured from multiple service providers; there could be primary and secondary service providers, who will need to agree on the service terms and conditions, thereby adding to the complexity of the SLAs. The SLA specifies service performance metrics like

- Availability timeframe of services,
- Scheduled maintenance times,
- Contingency or business continuity plans,
- Timeframes for notification and recovery following an unplanned service dis-

ruption or a security incident,

- Problem resolution and escalation procedures, etc.

1.2 Motivation

Cloud service contracts are currently managed as large text documents and have to be manually parsed by the consumers to determine which service best suits their needs which is very time consuming and prone to errors. This makes consumers dependent on the cloud service provider‘s performance reports to gauge the value of the cloud service. Moreover, cloud contracts contain rules and policies that are not fully encapsulated by existing performance metrics that cloud providers track. There are no standard definitions for cloud metrics, different providers could have different definitions for the same measure. One of the large federal agencies found that their cloud service provider was tracking the service availability measure as only system availability and not user access. The provider insisted that their service was available and meeting SLA requirements given that it was up and running even if not every valid end user could access it at the same time. As a result, the federal agency had to update the service contract to redefine ‘service availability’, the provider also had to the business process to track availability and user access statistics.

As SLA standards are still in nascent stage, different cloud providers construct their own rules and policies for the service offering and define them as clauses in the cloud legal document. The complexity increases when the documents are written by a third-party service provider. Monitoring service performance based on these documents is time consuming and tedious for the consumers and so often happens only when the provided service fails to live up to the expectations.

There should be a common platform with a common understanding through which information exchange and querying can be done. Therefore, it is important to monitor the SLA continuously. This can be done by automating the cloud service management by making the SLA machine understandable so that the monitoring tool could analyze and make decisions based on the knowledge extracted from the service contracts. We have used a combination of semantic web technologies, text mining techniques and natural language processing to build the framework. The ontologies created in previous work has been used and extended to add extra classes like service credits to depict the actual cost of services. We have also compared the changes that have occurred in the SLA of cloud service providers over the past few years [3]. We have monitored the change in the services provided as well as changes in structure and content of the SLA using modal word frequency and the ontology comparison.

In this paper, we initially discuss the background and related work in this area. In section III, we present our approach towards automating service level agreements and describe the ontology we have developed for the same. We present the results of our analysis in section IV and end with conclusions and future work.

THESIS STATEMENT A semantically rich, natural language processing approach can be used to automatically extract knowledge elements from cloud service providers SLA.

This thesis is organized as follows: In Chapter 2, we present the related work.

In Chapter 1, we give an introduction to the work being done. In Chapter 3, we describe our the domain our thesis data is based on. In Chapter 4 we describe the schema of the ontology, explaining the various classes and their relationships. In Chapter 5 we talk about different approaches, technologies and method used to complete the thesis and the ways we test and evaluate the framework built. In Chapter 7, we summarize those results and suggest areas for future work.

Chapter 2: Related Work

2.0.1 SLA Standards

Baset [1] describes an SLA as consisting of service guarantee that specifies the metrics which a provider strives to meet over a service guarantee time period. Failure to achieve those metrics will result in a service credit to the customer. Availability, incident response time and recovery, and fault resolution time are examples of service guarantees. Service guarantee granularity describes the resource scale on which a provider specifies a service guarantee. For example, the granularity can be on a per service, per data center, per instance, or per transaction basis. SLA also specify the Service guarantee exclusions. Service violation measurement and reporting describes how and who measures and reports the violation of service guarantee, respectively. In recent years, international bodies have been working towards defining standard definitions and terminologies for a Cloud SLA. The International Organization for Standardization's ISO/IEC DIS 19086 standard [4], European Commission's "Cloud Service Level Agreement Standardization Guidelines" and NIST's Special Publication 500-307 on Cloud Computing Service Metrics Description [5], are some of the publications that when finalized will help consumers mandate a fixed SLA format from various cloud providers. That will also help consumers compare and contrast the various cloud services based on their terms of service/SLA metrics. The committee responsible for the standardization and documentation of Service Level Agreements is ISO/IEC JTC 1, Information technology, Subcommittee SC 38, Cloud computing and distributed platforms. The overview, fundamental concepts, and definitions for cloud SLA is in ISO/IEC 19086 which builds on the cloud computing concepts defined in ISO/IEC 17788 and ISO/IEC 17789.ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization [6]. According to ISO/IEC 19086 the cloud SLA should consist of the key characteristics of a cloud computing service and should provide a common understanding between cloud service consumers and cloud service providers. A cloud SLA framework should consist of Cloud Service Agreement (CSA), Cloud Service Level Agreement (SLA), Cloud Service Level Objectives (SLO) and Cloud Service Qualitative Objectives (SQO) [5]. The SLA content areas are as follows:

1. Accessibility: Usability of a product, service, environment or facility by people within the widest range of capabilities.

2. Business continuity: Capability of the organization to continue delivery of products or services at acceptable predefined levels following a disruptive incident.

3. Cloud service agreement: The documented agreement between the cloud service provider and cloud service customer that governs the covered service.

4. Cloud SLA: Part of the cloud service agreement that includes cloud service

level objectives and cloud service qualitative objectives for the covered cloud service.

5. Cloud service level objective (SLO): Commitment a cloud service provider makes for a specific, quantitative characteristic of a cloud service, where the value follows the interval scale or ratio scale.

6. Cloud service qualitative objective (SQO): Commitment a cloud service provider makes for a specific, qualitative characteristic of a cloud service, where the value follows the nominal scale or ordinal scale.

7. Disaster recovery: Ability of the ICT elements of an organization to support its critical business functions to an acceptable level within a predetermined period of time following a disaster

8. Failure notification policy: Policy specifying the processes by which the cloud service customer and cloud service partner can notify the cloud service provider of a service outage and by which the cloud service provider can notify the cloud service customer and cloud service partner that a service outage has occurred. Interval scale: continuous scale or discrete scale with equal-sized scale values and an arbitrary zero.

9. Metric: Standard of measurement that defines the conditions and the rules for performing the measurement and for understanding the results of a measurement.

10. Nominal scale: Scale with unordered labeled categories or ordered by convention and so on.

Tasks in analyzing service level agreements:

1. Comprehending Cloud Service Agreements: Once a cloud service consumer has chosen a cloud service after analyzing the service characteristics and if its requirements are met, then this information is placed in the cloud SLA. Cloud SLA has Cloud service objectives. These objectives give a value to a cloud service characteristic that is understood to have a meaning and a specific level of it needs to be met by the cloud service. A metric does not give specific values but gives a few rules on how to measure a particular characteristic of cloud service.

2. Metrics Measurement: The document gives a model for representing the metrics in a specific way so that measurement tools can be devised. The measured value can be compared to cloud service objective commitments made in the cloud service agreements.

3. Verification of cloud SLA: As the measurements are made with the same metrics that were used to define the characteristics of a service in the cloud SLA,it is possible to compare the measurement result to the cloud service objective commitment. If the values do not match then a remedy will be sought out.

2.0.2 Semantic Web

In cloud-based service environments, there is an exchange of information, queries, and requests between the user and the cloud service provider. This information exchange could be for data or policies followed by the cloud service providers. The handling of heterogeneous policies is usually not present in a closed and/or centralized environment but is an issue in the open cloud. The inter-operability requirement is not just for the data itself, but even for describing services, their service level agreements, quality-related measures, and their policies for sharing data.

One way to solve this would be to use semantic web techniques for modeling service related information. We have used this to automate the monitoring and analysis of cloud service level agreements [2]. The data is annotated with machine-understandable meta-data, which could be queried and used in the correct context in an automated manner. Semantic web technology uses simple languages RDF [7] (Resource Description Language) and OWL [8] for describing the objects and relations, define ontologies and describing the meta-data using these ontologies.

Another approach uses Web Services Definition Language (WSDL) [9] which is a W3C standard for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. However, WSDL does not allow a means to express the policies that the service supports or adheres to. Hence additional proposals like WS-Policy and WSLA have been made to allow for the expression of additional nonfunctional attributes. Turner et al. [10] have proposed a service technology layer for the creation and deployment of web services. They have compared the existing protocols and technology available to implement web services and have also noted gaps that need to be researched.

Web Services Agreement Specification (WS-Agreement) is a web services protocol for establishing the agreement between two parties using an extensible XML language for specifying the nature of the agreement, and agreement templates to facilitate the discovery of compatible agreement parties. WS-Agreement limits the ability to match the agreements to syntactical matching and is also very limited in matching non-functional attributes that define policies on data and security, compliance issues, data quality levels, etc.

Oldham et al. [11] have proposed semantically rich extensions to extend the WS-Agreement. Aiello et al. [12] have proposed a formal definition of 'Agreement' based on WS-Agreement. WS-Negotiation is proposed as an extension of WS-Agreement to allow negotiation capabilities. WS-Negotiation consists of three parts - negotiation message, negotiation protocol, and negotiation decision. In this basic model for WS-Negotiation, the emphasis is on how to negotiate and what to negotiate. It does not demonstrate how a requester‘s enterprise policies can be used to automate the negotiation process. Bui and Gachet [13] have described a broker capable of offering negotiation and bargaining support to facilitate the matching of web services. The negotiation protocol and decision making mechanisms for negotiation have not been described. Skogsrud et al. [14] propose a trust negotiation framework that supports policy lifecycle management for web services. However, this framework is limited to managing the user identity of customers accessing the service. Yao et al. [15] propose flexible strategies for web services negotiation. Their approach does not use policy management to automate service negotiation. There has also been work done on negotiating between web services, however, this work does not relate to the negotiation needed between service provider and consumer.

For our cloud services lifecycle framework, we have used semantic web technologies like OWL instead of WS-Agreement and WS-Negotiation protocol as we were able to more richly define the cloud SLA ontologies, thereby allowing us to incorporate different descriptions of the same SLA measure. Our proposed ontology using protege (built and developed by Stanford [16] is flexible enough to capture knowledge in terms of existing and future SLA standards.

2.0.3 Text Extraction

Researchers have applied a skip gram model to develop word2vec [17] to show the distribution of words in a vector space, find the embeddings and to run a machine learning model on them. Researchers have applied Natural Language Processing (NLP) techniques to extract information from text documents. In Rusu et. al. [18] the authors suggest an approach to extract subject-predicate-object triplets. They generate Parse Trees from English sentences and extract triplets from the parse trees. Etzioni et al. [19] developed the KNOWITALL system to automate the process of extracting large collections of facts from the web in an unsupervised, domain-independent, and scalable manner. Etzioni et al. [19] used Pattern Learning to address this challenge.

Natural language technique uses Noun Phrase extraction for information extraction to create triplets by considering 'Noun Phrase' which would be obtained from part-of-speech taggers. Barker et al. [20] extract key-phrases from documents and show that the noun phrase-based system performs roughly as well as a state-of-the-art, corpus-trained key-phrase extractor.

Documents consist not only of a huge number of unstructured texts but also a vast amount of valuable structured data in the form of tables. Extracting knowledge from structured tables is an ongoing research problem with multiple solutions proposed to handle both general and domain-specific tables. Mulwad et al. [21] proposed a framework that assigns a class to table columns and links table cells to entities, and inferred relations between columns to properties. Bhagavatula et al. [22] created a system called TabEL which works by extracting content using entity linking. We use a modified version of it to improve our knowledge extraction system. Researchers have applied a skip gram model to develop word2vec [23] to vectorize text, find the embeddings and to run a machine learning model on them. Researchers have explored the automated techniques for extracting permissions and obligations from legal documents using text mining and semantic techniques, Kagal et al. [24] formed, proposed a semantic web-based policy framework [25] to model conversation specifications and policies using deontic Logic [26]. Deontic logic has been applied on cloud documents [27], and a comparison of the percentage of deontic statements usage in various company documents has been done.

In our previous work, we have used topic modeling to extract key terms with moderate results. We intend on extending it to small text topic modeling as the traditional methods do not yield great results because of the size of the SLA. We have also described a preliminary knowledge extraction system for cloud SLA documents. In this paper, we extend our framework to include extraction of information from tables and rules in the form of obligations and permissions from cloud SLA documents. which is then automatically populated to the knowledge graph and queried later for results.

The next chapter will give an overview of different kinds of cloud legal documents and the research challenges in analysing them.

Chapter 3: Cloud Legal Documents

3.1 Overview

The Cloud legal documents are signed agreements between the cloud service provider and consumer. These are currently maintained as text documents and include proper metrics and measurements to standardize the procedure of agreements and also to monitor it once the agreement is done.

3.2 Types of Legal Documents

3.2.1 Service Contract documents

Cloud legal documents in this category lay down rules and clauses that specify service functionality, quality and performance metrics. They also specify user access policies and service availability. Documents in this category include the Service Level Agreement, Terms of Service, Customer Agreement, Acceptable Use Policy, etc. A key role of these contracts is to enable efficient management of the cloud service. Metrics listed in these contracts are used by the vendors to assure the consumer of high performance of their services. NIST's special publication 500-307 [5] defines 'metrics as provides knowledge about characteristics of a cloud property through both its definition (e.g. expression, unit, rules) and the values resulting from the observation of the property'. For instance, a customer response time metric can be used to estimate a specific response time property (i.e. response time from customer to customer) of a cloud email service search feature. It also provides the necessary information that is needed to reproduce and verify observations and measurement results. However, one glaring observation after reviewing these contracts was the sheer variety of formats of these documents and the metrics used to track the performance of the same type of service. Due to lack of a standard cloud contract format, we came across SLA documents that were 3 pages long to ones that were over 25 pages in length even though they were offering the same type of computing service.

Components of SLA

In our research, we focus on SLA documents for various cloud services. The main components of an SLA document are:

- Service Commitment: The SLA document specifies the service commitments made by the service provider to its customers.
- Definitions and metrics: The SLA documents also contain definitions and metrics such as 'Availability', 'Region' and 'Uptime/Downtime' which are used to compute the service commitments.
- Rules for computation of service credit corresponding to the commitments by the provider.

- Procedure to request for service credit by the user and payment method by the service provider. These describe the specifications about how a user can apply for a service credit, for example, the request for service credit must include the account information and exact dates of outages of the service.

- Conditions under which the service provider is not liable to its service commitment (exclusions) to the users

- Deontic Expressions: These are modal statements that comprise 'Permission', 'Obligation', 'Dispensation' and 'Prohibition'. These could pertain to the service provider or the customer based on the actor in that statement. It helps the customer to understand the key components like liabilities and rights concerning the subscribed service.

3.2.2 Privacy and Security Data Documents

Cloud legal documents in this category lay down the data security and privacy policies for the service. These documents are usually required by federal and state regulations. Cloud providers either create a single privacy policy for all their services or create multiple privacy policies depending on the service data/category. Privacy policy documents were observed to be similar in content and format across most of the cloud vendors.

3.2.3 Regulatory Compliance documents

Compliance are set of rules, policies or standards formulated by regulatory agencies or standards organizations [5]. Compliance models implement rules and regulations across various components of Information Technology (IT) to make them work harmoniously. Security and privacy compliance models have been proposed for cloud computing security to ensure data protection and user privacy. Some of the proposed models include ISO 27001 [4] , COBIT, etc.

3.3 Research Challenge

Lack of standardization: Due to the lack of standards for cloud service performance, providers often construct their own rules for performance measures and metrics. They define them as 'clauses' in the cloud legal documents, such as TOS, SLAs or privacy policy documents, that are part of the cloud contract. Also, regulatory and compliance bodies have also developed rules and policies that affect the way cloud services can be provided or consumed. Reviewing all these cloud legal documents to ensure the cloud service is meeting the organizational requirements is a labor and time-intensive endeavor for consumers. Also, is often an afterthought when a cloud service fails to live up to its expectations. Hence, continuous cloud service monitoring has been identified as a key open issue by consumers. A critical step in automating cloud service management is to make the cloud SLAs machine-understandable so that monitoring tools can interpret the policy rules and metrics defined in the service contracts.

Co-referencing/cross-referencing: Many legal documents tend to reference other documents or different sections within the document. There is a need to develop techniques to represent and reason over multiple legal documents that co-reference/cross-reference another set of documents and/or sections within the document.

The next chapter explains the components of the ontology being built for the cloud service document.

Chapter 4: Ontology for Cloud Service Level Agreement

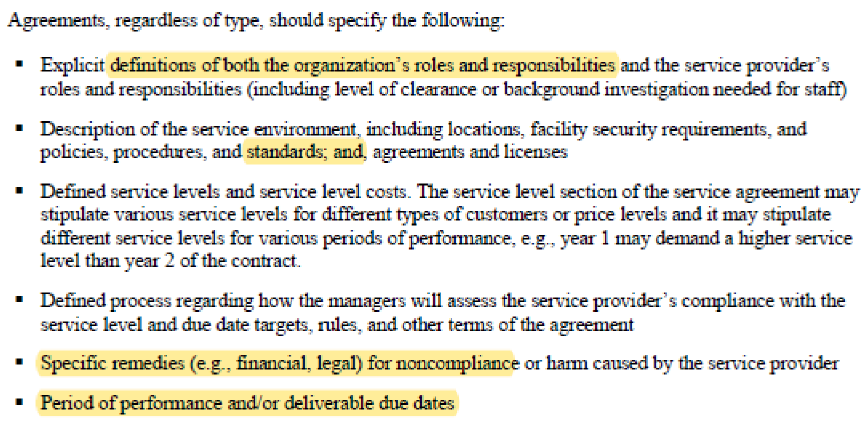
4.1 Introduction

The main goal is to make the SLA machine readable. To achieve this, we collect the key terms and relationships extracted using various techniques, from the SLA, which is used to develop a semantically rich ontology. Ontology is based on knowledge representation language, OWL (Web Ontology Language). The ontology could be queried using SPARQL to obtain the stored relationship. This base of the ontology was built by the combination of two tasks, Key-term and relationship extraction from the text and matching the obtained terms with NIST [5] suggested standards. According to [28] the figure 4.1 shows the essentials to be present in a Service Level Agreement.

4.2 Classes

1. ***Class: CloudSLA***: This is the main class containing subclasses like service name, service description, service delivery, etc.

2. ***Class:DeonticStatements*** :The class contains subclasses "Permission", and "Obligation", . The statements in the SLA of a service provider that indicate


Agreements, regardless of type, should specify the following:

- Explicit definitions of both the organization's roles and responsibilities and the service provider's roles and responsibilities (including level of clearance or background investigation needed for staff)
- Description of the service environment, including locations, facility security requirements, and policies, procedures, and standards; and, agreements and licenses
- Defined service levels and service level costs. The service level section of the service agreement may stipulate various service levels for different types of customers or price levels and it may stipulate different service levels for various periods of performance, e.g., year 1 may demand a higher service level than year 2 of the contract.
- Defined process regarding how the managers will assess the service provider's compliance with the service level and due date targets, rules, and other terms of the agreement
- Specific remedies (e.g., financial, legal) for noncompliance or harm caused by the service provider
- Period of performance and/or deliverable due dates

Figure 4.1: NIST Guidelines for SLA

permission are the DataProperty values of the class Permission. The statements which are Obligation statements in the SLA are the DataProperty values of the class Obligation and so on.

3. ***Class: ServiceStandards*** : The class ServiceStandards has the standards as defined by the service provider. These are the the terms that are mentioned with or without the title 'Definitions' in the SLA document. The data property of this has the frequently used terms and their definitions.

4. ***Class:TermsandAdjustments*** : This class has subclasses like 'ServiceCost', 'PeriodofPerformance' and 'RemedyforNonCompliance'. The class 'RemedyforNonCompliance' contains the values of credits assured to the consumer by the service provider in case the provider is unable to provide the promised service.

4.3 Relationship

Relationships in the Ontology is the relationship between two classes which is called the ObjectProperty or the relationship between an instance and value of that instance called the DataProperty. These relationship has been extracted using various NLP techniques and represented in the Ontology.

1. ***hasDeonticStatements***:This DataProperty represents the relationship between various Service Level Agreements and their DeonticStatements values.

2. ***hasObligationStatements***: This relationship relates the instances of class Obligation to their SLA Obligation Statements.

3. ***hasPermissionStatements***: This relationship relates the instances of class Permission to their SLA Permission Statements.

4. ***hasActors***: This relationship relates the instances of class Obligation and Permission to the DataPropertyValues actors in the Obligation and Permission statements.

5. ***hasRemedyforNonCompliance***: This relates the instances of class RemedyforNonCompliance with the values of credits returned in case of non compliance.

6. ***hasServiceStandards***: This relates the instances of class ServiceStandards with the standardized terms as defined by the SLA.

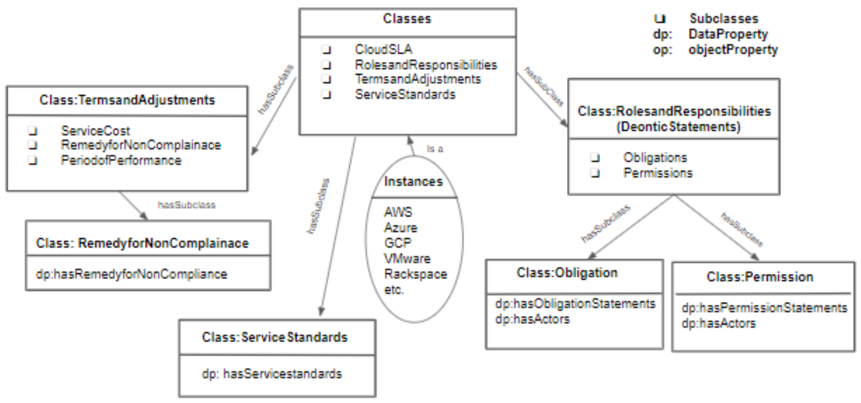


Figure 4.2: Cloud Service Level Agreement Ontology

We found that the most time-consuming process of cloud service consumption is the service negotiation process. Semantic web [29] usage for automation of this process itself is a performance improvement over tedious manual effort. The negotiation would involve discussion and agreement that the service provider and consumer have, regarding the service delivered and its acceptance criteria. A specification is laid out by the consumer as to what service they require in the RFS (Request for Service). The service provider would then layout the SLA which should be agreed upon by both the provider and the consumer. SLAs define the service data, delivery mode, agent details, quality metrics and cost of the service. While negotiating the service levels with potential service providers, consumers can explicitly specify service quality constraints (data quality, cost, security, response time, etc.) that they require. There could be instances where a service provider would provide a service that contains components provided by multiple providers. The SLA should clearly state this relationship and responsibilities. There could be a primary and secondary service provider, the primary provider would interface with the consumer and hence would be responsible for the composition of the services. The primary provider will negotiate the service metrics with the secondary provider. The service metrics have been defined in the ontology that was created using OWL. The Service Level Agreement class consists of properties that are common across all cloud applications. The class SLA has the property hasCost has a class Service Cost which has properties like Service Cost, Pricing Model, etc. SLA has property hasAvailability which has a class Service Availability which has properties like Availability and Terms of availability [2]. These are very helpfully in automating the service consumption and negotiation process as this would help in a clear comparison of various cloud service providers.

The cloud service consumers will have a service contract which contains SLA that can be stored as instances in the proposed ontology [2] and managed as RDF graphs, as shown in Figure 1, which can be automatically queried using SPARQL [30]. Any updates to the SLA would be stored as an RDF graph [7]. This was done so that the consumers can keep a track of SLAs of all the services that they have been using over the years and use that information to plan the future. This ontology is available in the public domain and can be accessed. This ontology forms part of the integrated ontology that we developed for our integrated cloud life cycle framework. The ontology has been extended to include hasExclusions property in Class Service Rebate which would determine the terms which cause the customers to not be able to get a service refund, this includes the procedure for a refund and the time taken to file the request. A property 'Availability Zones' is added as this one, from the research, seems to be is an important factor in calculating the availability of a service, especially when a refund is to be made when service is unavailable.

In the next chapter we talk about the methodology, steps, technologies and techniques used to build and populate the NIST standard based SLA ontology

Chapter 5: Methodology for Building the Framework

5.1 Overview

Through this research, we aim to enable the consumer to manage multiple cloud services. Our framework has a few elements as below to achieve this overall goal. This is done using text mining, Natural Language Processing and Semantic Web techniques on the available SLAs. We aim to have a system that would maintain knowledge about various terms and rules in the SLA obtained by SLAs compliance and regulatory policies, contracts, privacy documents, etc.

5.2 Data-set Used for the Experiment

Data used is from publicly available Service Level Agreement documents of popular cloud service providers like VMware vCloud Air [31], Amazon Web Services(AWS) [32], Microsoft Azure cloud [33], Google Cloud Platform [34] and IBM Softlayer [35] etc. Most of these are compute SLA, however, the framework works fairly well even on other SLA which follow the NIST guidelines and has arranged the documents according to the extractor in the framework.

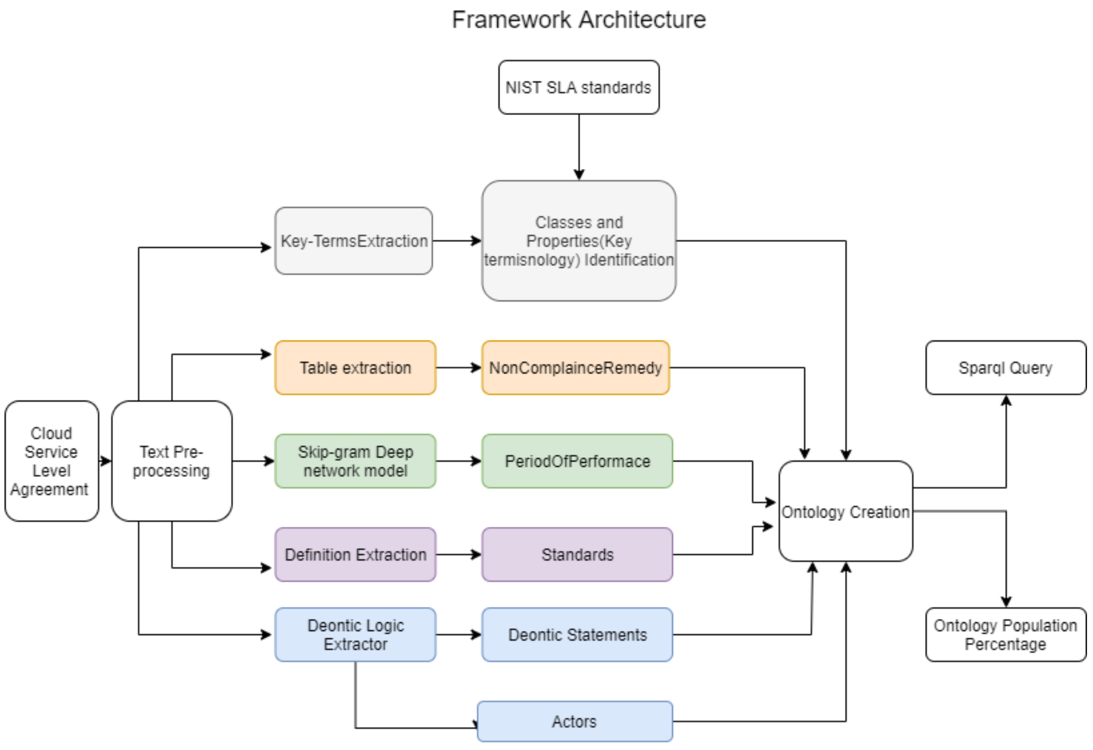


Figure 5.1: Architecture to analyse Cloud Service SLA

5.3 SLA Ontology Creation

We begin by downloading publicly available SLA documents of various cloud service providers. We extract the key items from these documents according the NIST standards suggestions. An ontology is built with the base as suggested by NIST, then the extracted text is populated onto this ontology. The extracted text is populated onto the ontology as triplets. The various aspects of the NIST standard becomes the main classes and subclasses of the ontology. Each service provider is considered an instance of the above built classes and subclasses. The value or the actual sentences corresponding to the NIST standard in each provider's SLA becomes the data Property value a class/subclass and in turn of the instance. Once the SLA terms are identified we save them as an RDF graph which is machine-understandable. Once this is done it could be used to automate the monitoring of SLA compliance. We calculated the percentage population for each item to get a better comparison between different provider's SLA. Once this is done key components can be queried from the ontology using Apache Jena Fuseki.

The SLA terms identified in the terms of service document can be mapped to the ontology when a service provider is selected. The system identifies, for example the term 'availability', this term could now be mapped to the class 'Service Availability' in the cloud SLA. We have been able to add terms and values to the ontology in an automated fashion with no manual intervention. Different key terms could form classes in the ontology with a relationship between them. Adding these terms as classes and establishing the relationship between them in the graph could be done automatically with the help of a script that uses the library 'Owlready'. The main classes in the ontology are 'CloudSLA', 'DeonticStatements', 'SLAStandards', 'TermsandAdjustments', with subclasses. The service providers which are instances are 'AWS', 'GCP', 'Azure', 'VMware' the main four. Once the ontology was completely built its was tested with other SLAs like 'IBMcloud', 'Alibaba Cloud', 'Oracle', 'Rackspace' etc. for checking the capturing capacity of the framework.

Once the SLA terms are mapped we parse the text again to find the SLA measures corresponding to that term. For instance, AWS compute promises to return 10 % of credits if the availability is Less than 99.99% but equal to or greater than 99.0%. This detail can be obtained by querying the remedy for compliance for the instance AWS. The definition of the term 'Healthy Backend Instances' as defined by GCP can be obtained by querying the standards(definitions) for the instance GCP. Similarly the user could query the obligations or permission imposed by any specific service provider or query all of them to compare and contrast. All of this querying could be done using Apache Jena Fuseki by just a few lines of simple SPARQL query.

One challenge that was faced was the usage of different terminology by different providers. The term 'availability' could also be termed as 'uptime'. Understanding the context of these kind of words and their level of similarity is easy for humans by difficult for the machine. A vectorizing the text and applying skip gram helped capture the context of these words. Further analysis can be done by checking the closeness of terms and clustering them into groups. Similar words from different documents are expected to be close to each other in the vector space. We have also addressed this issue by adding complete descriptions of these measures in our ontology. We store the SLA instance in the form of an RDF graph using the SPARQL CONSTRUCT. This SLA graph is stored in a Fuseki graph store [36], which can be easily queried for continuous SLA monitoring since it is in a machine-readable format. For our prototype, we used the SLAs for the same service type and so all vendors' SLAs fit our framework. In our ongoing work, we are going to explore other service domains.

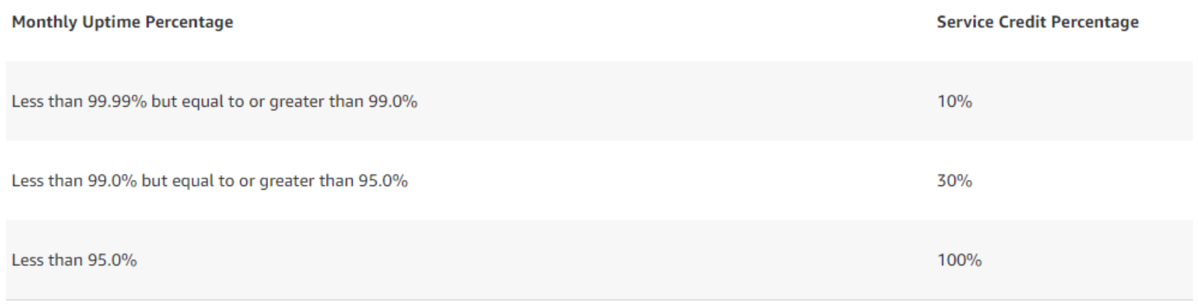
5.4 Text Extraction

The text from the SLA documents could be extracted in two ways.

5.4.1 Definition Extraction

Legal documents define the terms that are repeatedly used in them. This is done to achieve clarity without repetition. Some rules that are followed are that the legal terms should not be defined if used just once or if they conflict the accepted meaning. Also, they should be placed where they are most easily found, preferably quoted before they are used. SLAs contain definitions related to the provider's operation, the framework helps in extracting those terms making it easier for decision making.

To extract these definitions we tokenize the document into sentences, these sentences are parsed using the CMU Link parser [37] and regular expression [38]. Here the sentences would be split into noun phrase X and verb phrase Y. Y could even be a complex sentence on its own. X and Y are connected by a few filler words like 'is', 'means' etc. Pattern matching is used to match the sentences with this required pattern. If a sentence matches this pattern it gets picked up by the extractor, an example sentence would be *"A "Service Credit" is a dollar credit, calculated as set forth above, that we may credit back to an eligible account"*. Here in this sentence *'Service credit'* would be the X in the sentence and *'a dollar credit, calculated as set forth above, that we may credit back to an eligible account'* would be the Y. We then add this sentence to the RDF with the obtained relation between



Monthly Uptime Percentage	Service Credit Percentage
Less than 99.99% but equal to or greater than 99.0%	10%
Less than 99.0% but equal to or greater than 95.0%	30%
Less than 95.0%	100%

Figure 5.2: Remedy for Non Compliance Table in AWS

them, which is that X is a key term that is defined as Y. This forms the dataProperty value of the class 'SLAStandards' as these are the standards set by the SLAs.

5.4.2 Extraction of Remedies for Non Compliance

On various occasions the cloud service providers fail to provide the promised service duration or uptime. A good SLA should have a means to deal to with this non compliance. The Service provider should have clear information of how it would provide a remedy for this situation by giving back credits or extra service or some other remedy useful to the consumer. For example,AWS in their [32] specifies the exact amount of credits that would be given back to the consumer in case of AWS's failure to provide the promised service. This is usually in the form of a table. Various legal documents contain high-quality details in the form of tables. For example, the AWS SLA contains the table 5.2

This is very convenient for humans to understand. Unfortunately, its equally hard for it to be machine-understandable. To deal with this we use the Beautiful-Soup library in python. BeautifulSoup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide various ways of navigating, searching and modifying the parse tree. After parsing the table we end up with a dictionary of elements which is the contents of the table cells. To keep the framework more generic we have used the approach of extracting remedies for non compliance only from tables. Some SLA‘s not having remedies for non compliance populated in the ontology does not mean that they do not have remedies of non compliance, it only means that they do not have in a clear table format which is the case with several popular cloud provider SLA.

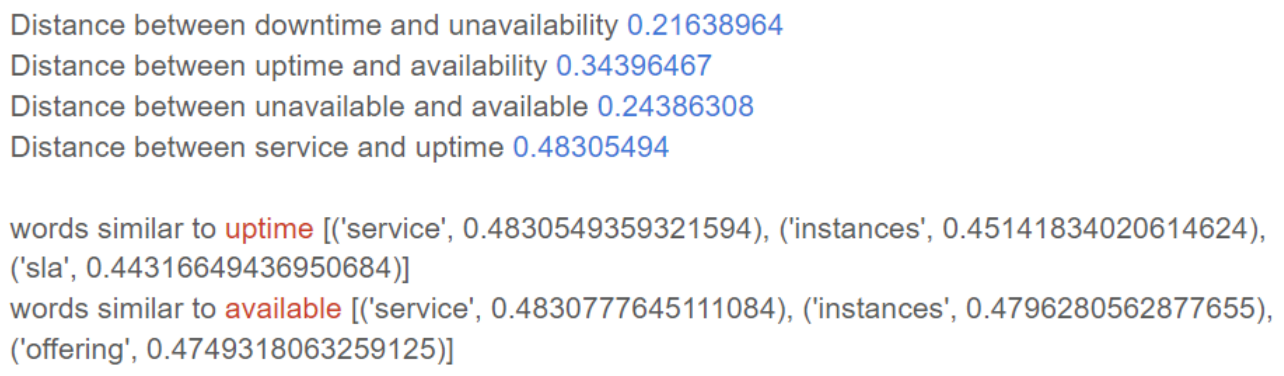
This extracted data is encoded as RDF statements. The RDF statements are then added to the knowledge base. These details are populated as the dataProperty of the class ‘RemedyforNonCompliance’,which is subclass of class ‘TermsandAdjustments’. Once we get the dictionary with data in it, the table headers are used to associate semantics to different cell data. Over the years the way the website displays their SLA has changed. Some have hyperlinks from the main SLA page to various other pages, some have the whole document on one single page. Some prefer to keep the data in PDF format, for this we follow a different approach of extraction as it would no longer follow the rules of HTML. We strove to keep the system reusable by not using any data or documents from a local copy but instead use real-time http links to service providers. This is done so that the same prototype could be reused even when the providers change their SLA content as long as the update is on the same link.

5.4.3 Extraction of Word Context

While dealing with a text document like the SLA, we often come across different terms being used to refer to the same concept across different service provider. It is relatively easy for humans to understand that the term ‘availability’ used by AWS is same as term ‘uptime’ used by GCP. However, it is not that trivial for the machine to understand this. To make the machine understand these words as similar words, we build a 3 layer deep network, which uses skip gram model to understand the context of the words. The words in SLA are the inputs to the first(input) layer of the model. The output layer predicts the context of the words using the word vectors. The hidden layer in the deep network model gives the word embedding of every word. We use cross entropy as the loss function.To minimize this loss function we use gradient descent. When the embeddings are plotted on a 2d plot we see that the values for the word ‘availability’ and ‘uptime’ are quite close to each other. This distance between various terms could be generated by adding more text to the input of the model. This kind of knowledge will help the machine understand the different terms for the same usage across various SLA.

5.4.4 Extraction of Roles and Responsibilities from SLA

We explored the use of Modal Logic to extract rules present in the cloud service SLA documents. This method of analyzing will help the customers to understand their rights and obligations towards the services that they buy and use. It helps in understanding the document by giving it a structure to look at. One big challenge



```
Distance between downtime and unavailability 0.21638964
Distance between uptime and availability 0.34396467
Distance between unavailable and available 0.24386308
Distance between service and uptime 0.48305494

words similar to uptime [('service', 0.4830549359321594), ('instances', 0.45141834020614624),
('sla', 0.44316649436950684)]
words similar to available [('service', 0.4830777645111084), ('instances', 0.4796280562877655),
('offering', 0.4749318063259125)]
```

Figure 5.3: Distance results from the Skip Gram Model

doing this was that every vendor has different vendor-specific terminology. Hence, we used a generic grammar-based extraction procedure.

We first extract sentences containing modal logic and then further extract sentences with deontic expression. Deontic logic is a field of philosophical logic that deals with obligation, permission, and related concepts. This means the sentences containing the words "must", "should", "will" impose obligations on the actor in the sentence. If the customer is the actor in such sentences she is responsible for the thing being talked about in the sentences has a few obligations.

On the other hand if the actor in this kind of a sentence is the service provider (*"Amazon **will** provide 99.8 % availability"*) then the service provider has obligations to be followed. This kind of extraction is extremely worthwhile when purchasing or monitoring services. Researchers have used deontic principles in the past to analyze policy and legal documents. Work done by Travis D.Breaux [39] utilized semantic web and text mining approaches to extract obligations and permissions from privacy policies. We also used similar techniques to extract deontic rules from cloud SLA documents. We have considered SLA documents from cloud service providers like Google Cloud Platform, Amazon Web Services, IBM Softlayer, HP Cloud Compute, Microsoft Azure Cloud and VMware vCloud Air Services.

The Steps followed to extract modal words are as below:

Using Parts of Speech Tagging for extracting modalities: The SLA documents are first sentence tokenized, then every sentence is passed through a Stanford POS tagger. This Parts of Speech tagger tags every word present in every sentence into a part of speech. Every word gets tagged an NN if it is a noun, MO if it is a modal verb, a VB if it is verb, CD for cardinal digit and so on. We will make use of this tag to form a general grammar rule. Our grammar rule extracts sentences that match the rule.

The output of this was all the sentences in SLA documents that have modal verbs in them. Modal verbs are verbs that indicates modalities like 'will', 'must', 'may' etc. This gives us the knowledge of what category among 'permission', 'obligation', 'dispensation' and 'prohibition' does the sentence belong to. The frequency of appearances of modal verbs in an SLA document is captured to give more inference. This was done in the year 2014 and then once in 2019, to observe the trend of how every cloud service under this experiment has started incorporating more modal words usage into their SLA.

We also observed from the extracted sentences with modal verb, that we have two kinds of actors in the SLA documents. The noun and pronoun part of the modalities are used to define the actors associated with the modality. One would be

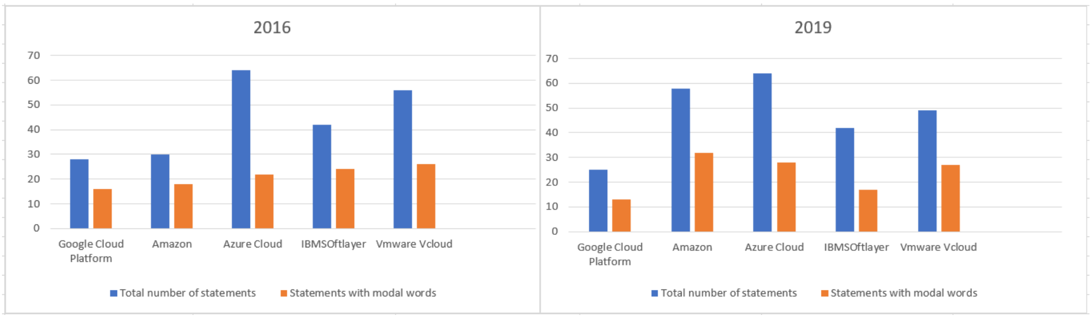


Figure 5.4: Comparison of total number of sentences to number of sentences containing modal words in SLA

the 'Service Provider' and other the 'Customer'. The sentences extracted represent obligations and permissions and mapping each Obligation/Permission with the actor will give us a clearer picture.

Comparison of SLAs across multiple service providers over the past few years: We have compared the number of statements in an SLA vs the number of statements containing modalities as shown in the figure 5.4.

We have also compared the results from 2016 with the ones from 2019 to see if the companies have changed the structure of their SLA to meet the new regulations. Statements containing modalities are useful for customers as they can easily compare their rights and obligations across multiple providers, which then helps them in picking the right provider for their needs. One thing to be noted is that the extreme difference in the AWS graph from 2014 to 2019 is because of their change in pricing and SLA structure. AWS was providing exclusive SLA for EC2 which they have now chosen to merge with other AWS products like EBS and ECS for some reason. And also, HPcloud has made is slightly more difficult to access the SLAs as compared to 2016 with their new structuring of the website and SLA. We also analyzed the key term usage in SLA changes that have happened over the years, we see that the importance given to the words like 'instance creation' has increased compared to terms like 'request' as shown in figure **??** which shows the change in the trend of application of cloud services.

Extracting Obligations and Permissions from SLAs: We extracted and separated rules that are permissions and obligations from statements containing modalities.

1. Permissions/Rights: Permissions are expressions that describe rights or authorizations for an entity/actor.

2. Dispensations: Dispensations describe optional or non-mandatory statements.

3. Obligations: Obligations define the responsibilities that an entity/actor must perform.

4. Prohibitions: Prohibitions specify the conditions or actions which an entity is not permitted to perform

The categorization of sentences from SLA into the above categories resulted in the below findings

Example Sentence:

Google ***will*** *make a determination in good faith based on its system logs, monitoring reports, configuration records, and other available information.*

Type: Obligation

Such sentences are extracted from the SLA with the help of the modal words and then populated under the respective class in the cloud SLA ontology (the statement in this example would get populated as a DataProperty value of the subclass Dispensation under the class DeonticStatements).

Once we have extracted the deontic statements, we further analyse this to extract the actor out of this statement. This is done as below. Thinking in grammatically and looking at the structure of the sentence. The first approach is to extract the nouns(would be the actor) from the statements, this would work perfectly in a statement like the one shown above and the actor would be Google. However, if the statement has more than one noun like below.

Example Sentence:

*At our discretion, **we may** issue the Service Credit to the credit card **you** used to pay for the billing cycle in which the Unavailability occurred.*

Actor: CUSTOMER

Actor: SERVICE PROVIDER

Then both the pronouns ('we' and 'may') would be returned as actors. Another approach to solve this problem would be to use dependency parsing.

"Dependency parsing is the task of extracting a dependency parse of a sentence that represents its grammatical structure and defines the relationships between "head" words and words, which modify those heads [40]."

1. The sentences with modal words are first determined.

2. From these sentences the 'ROOT' words like 'apply','comply' etc,. are captured.

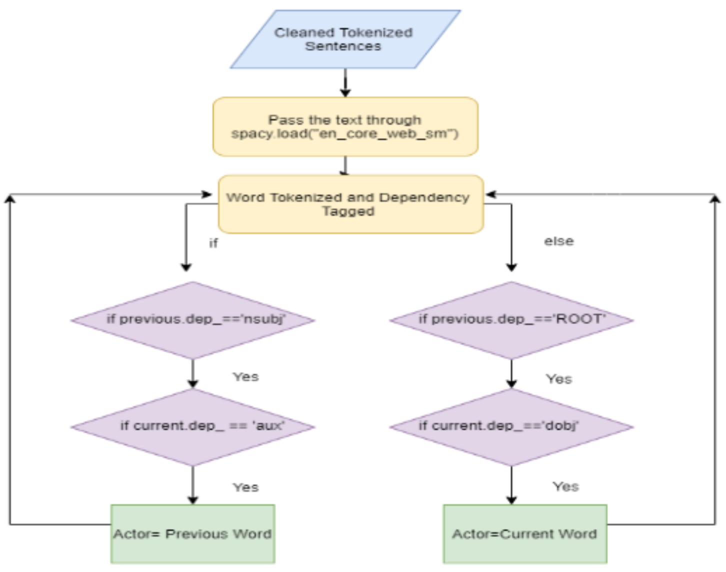

Cleaned Tokenized Sentences
Pass the text through spacy.load("en_core_web_sm")
Word Tokenized and Dependency Tagged
if
else
if previous.dep_=='nsubj'
Yes
if current.dep_ == 'aux'
Yes
Actor= Previous Word
if previous.dep_=='ROOT'
Yes
if current.dep_=='dobj'
Yes
Actor=Current Word

Figure 5.5: Actor extraction from dependency parsed statements

3. Check if the relationship between the 'ROOT' word and the modal is 'aux', (An aux (auxiliary) of a clause is a function word associated with a verbal predicate that expresses categories such as tense, mood, aspect, voice or evidentiality) [41]

4. If yes, then extract the nominal subject ('nsubj') [42] of the 'ROOT' word.

5. Tag the 'nsubj' as the 'Actor' in the statement.

Example Sentence:

*At our discretion, **we may** issue the Service Credit to the credit card **you** used to pay for the billing cycle in which the Unavailability occurred.*

Dependency parsing: *prep(issue-7, At-1) poss(discretion-3, our-2) pobj(At-1, discretion-3) punct(issue-7, ,-4)* ***nsubj(issue-7, we-5) aux(issue-7, may-6) ROOT(issue-7, issue-7)*** *det(Credit-10, the-8) compound(Credit-10, Service-9)* *dobj(issue-7, Credit-10) prep(issue-7, to-11) det(card-14, the-12) compound(card-14, credit-13) pobj(to-11, card-14) nsubj(used-16, you-15) relcl(card-14, used-16) aux(pay-18, to-17) xcomp(used-16, pay-18) prep(pay-18, for-19) det(cycle-22, the-20) compound(cycle-22, billing-21) pobj(for-19, cycle-22) prep(occurred-27, in-23) pobj(in-23, which-24) det(Unavailability-26, the-25) nsubj(occurred-27, Unavailability-26) relcl(cycle-22, occurred-27) punct(issue-7, .-28)*

'ROOT' : 'issue'

'aux' : 'may'

'nsubj' : 'we'

Actor: we (Service Provider)

Once the deontic statements and the actors for each statement are captured from every SLA under test. We populate it onto the ontology as a data property of the class 'Obligations' and 'Permissions'.

5.5 Knowledge Graph Auto-population

Our end goal is to have a knowledge base with SLA term definitions and measures of all cloud service providers. This could be queried later to get the required information. To populate the ontology we developed an automated system that would fetch the key terms and its relationship from the SLA and populate it onto the knowledge graph. This is done in the following steps:

- Fetch and process the SLA of a service provider
- Extract the key terms from the SLA using various Text Mining and Natural

Language Processing techniques.

- Obtain the Noun Phrase and the Verb Phrase using the Stanford parser.
- Write the triples extracted, onto a JSON file as key-value pairs
- Then the JSON file is read and the key-value pairs are populated onto the knowledge graph. The terms now become the classes in the knowledge graph and the relationship fills in as Object Property and Data Property. This is achieved using a python library 'Owlready'

For example, we extract the permission statements obtained from AWS SLA with the help of the modal verbs like 'may', 'can', 'could' etc. These permission sentences are written onto a JSON file with AWSPermission as the key and the permission statements as the value. The JSON file is then read and used to populate the knowledge graph. This is done by making the term 'AWSPermission' as an instance of the class 'Permission' under the SuperClass 'DeonticStatements'. The value in this case (the permission statements) would then be populated as the DataProperty value of the instance 'AWSPermission'. The relationship between the key and the value (AWSPermission and Permission Statements in AWS SLA) would be 'hasPermissionStatements'. The relationship being populated onto the knowledge graph is as shown in figure 4.2

5.6 Query the Ontology

Once the ontology was created we can query the ontology using SPARQL [30]. SPARQL is a semantic query language with which we can query the data stored in the knowledge graph in RDF format. We used the Apache Jena Framework [36] for running the SPARQL queries. One with basic SPARQL query knowledge would be able to extract the information from the ontology. We could further build on this by making a Question Answering System, where we could process the questions. Extract the key terms from these questions(apply Natural Language Processing techniques), link the entities, and extract information from the corresponding knowledge graph. The extracted information is displayed in the form of text, graph or tables. The first step to doing this would be to build a back-end to this system would be to create ontology like the one that we have done, which uses OWL to represent the cloud SLA in the public domain. The user can either select a cloud service name to analyze the properties or type in a natural language query. The purpose of this framework is to present to the user, main components of an SLA such as definitions, metrics, permissions, and obligations which are stored in the knowledge graph and provides a user the ability to query and reason over them.Build a SPARQL query using those key terms. Here the user can compare the permissions of various service providers side by side, similarly, the obligations, definitions and non compliance remedies and make a calculated decision on choosing the service provider.

In the next chapter we talk bout the results and evaluation

Chapter 6: Evaluation and Validation

6.1 Ontology Evaluation

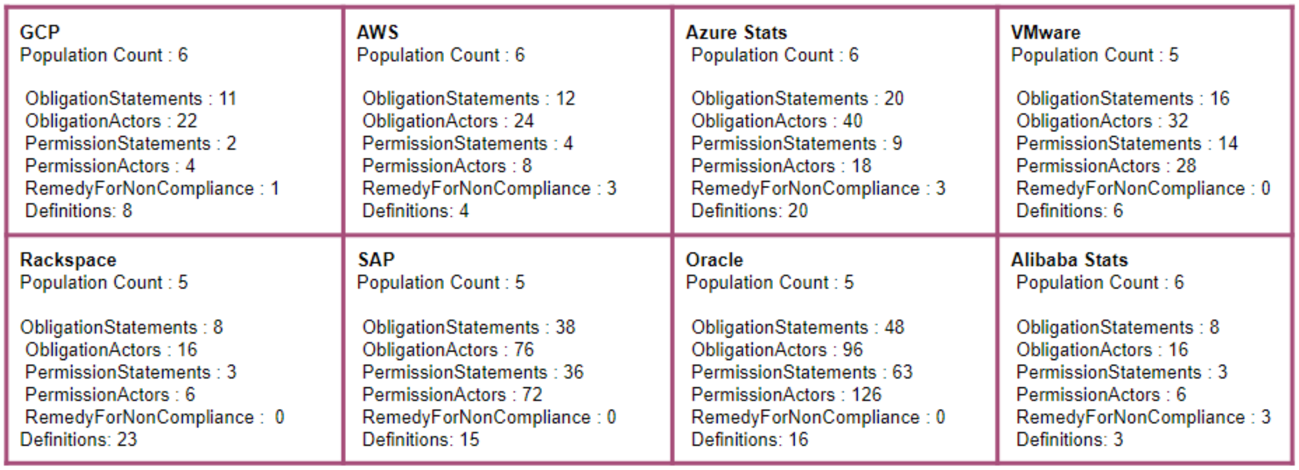
6.1.1 Ontology Population Count

Every Population into the ontology is recorded. Each Populated class increases the total count by one. We have also captured the number of entries for each class population for a particular SLA. The ontology population capacity has been captured for every instance as shown in 6.1. We see that the class population value for VMware is 0 , now that does not mean that VMware does not have remedy for non compliance , it only means that the remedies are not present in the SLA in the format that we are setting up in the framework. VMware's Remedy for non compliance is scattered around in the entire text and not placed in a table format.

6.1.2 SPARQL Query

The ontology that is build can be queried to obtain various results based on the user's requirement. Below are a few test cases, with their respective SPARQL query and results.

1. Case 1: Provide the Obligations Statements with Actors that talk about 'Ser-



GCP	AWS	Azure Stats	VMware
Population Count : 6	Population Count : 6	Population Count : 6	Population Count : 5
ObligationStatements : 11	ObligationStatements : 12	ObligationStatements : 20	ObligationStatements : 16
ObligationActors : 22	ObligationActors : 24	ObligationActors : 40	ObligationActors : 32
PermissionStatements : 2	PermissionStatements : 4	PermissionStatements : 9	PermissionStatements : 14
PermissionActors : 4	PermissionActors : 8	PermissionActors : 18	PermissionActors : 28
RemedyForNonCompliance : 1	RemedyForNonCompliance : 3	RemedyForNonCompliance : 3	RemedyForNonCompliance : 0
Definitions: 8	Definitions: 4	Definitions: 20	Definitions: 6

Rackspace	SAP	Oracle	Alibaba Stats
Population Count : 5	Population Count : 5	Population Count : 5	Population Count : 6
ObligationStatements : 8	ObligationStatements : 38	ObligationStatements : 48	ObligationStatements : 8
ObligationActors : 16	ObligationActors : 76	ObligationActors : 96	ObligationActors : 16
PermissionStatements : 3	PermissionStatements : 36	PermissionStatements : 63	PermissionStatements : 3
PermissionActors : 6	PermissionActors : 72	PermissionActors : 126	PermissionActors : 6
RemedyForNonCompliance : 0	RemedyForNonCompliance : 0	RemedyForNonCompliance : 0	RemedyForNonCompliance : 3
Definitions: 23	Definitions: 15	Definitions: 16	Definitions: 3

Figure 6.1: Ontology Population Count

vice Credits'.

2. Case 2: Compare the credits refunded by Cloud Providers if the availability is around 95%

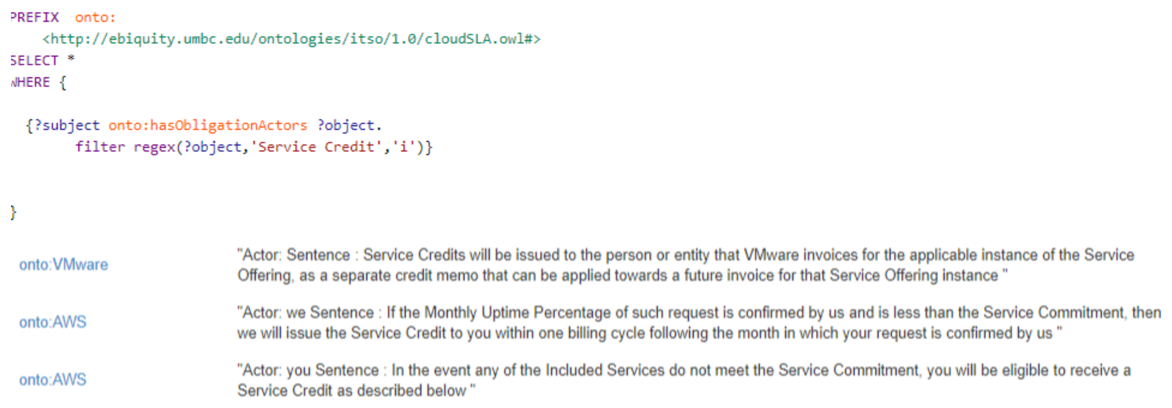
3. Case 3: Looking for Permission with respect to the 'Availability'

4. Case 4: Find the Obligations and refunds provided by AWS

6.2 Actor Evaluation

We see there there are 3 correct categories of Actors Service Provider, Customer and Neither. An Example of all the three is as below. The framework is incorrectly extracting Actors for a few statements, this has also been recorded. This could be because of the complexity of the statement or that the statement does-not follow the extraction structure.

Actor: Service Provider *"Actor: VMware Sentence : VMware will review the*



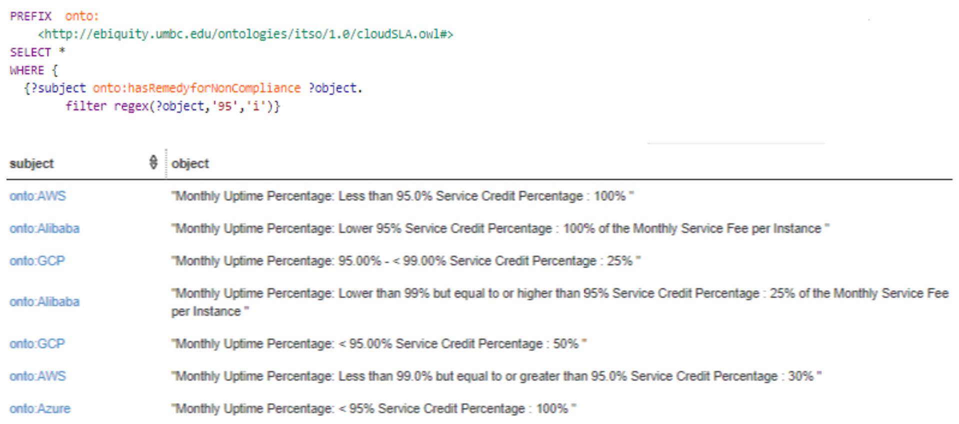
```
PREFIX  onto:
    <http://ebiquity.umbc.edu/ontologies/itso/1.0/cloudSLA.owl#>
SELECT *
WHERE {

  {?subject onto:hasObligationActors ?object.
        filter regex(?object,'Service Credit','i')}


}
```

onto:VMware	"Actor: Sentence : Service Credits will be issued to the person or entity that VMware invoices for the applicable instance of the Service Offering, as a separate credit memo that can be applied towards a future invoice for that Service Offering instance "
onto:AWS	"Actor: we Sentence : If the Monthly Uptime Percentage of such request is confirmed by us and is less than the Service Commitment, then we will issue the Service Credit to you within one billing cycle following the month in which your request is confirmed by us "
onto:AWS	"Actor: you Sentence : In the event any of the Included Services do not meet the Service Commitment, you will be eligible to receive a Service Credit as described below "

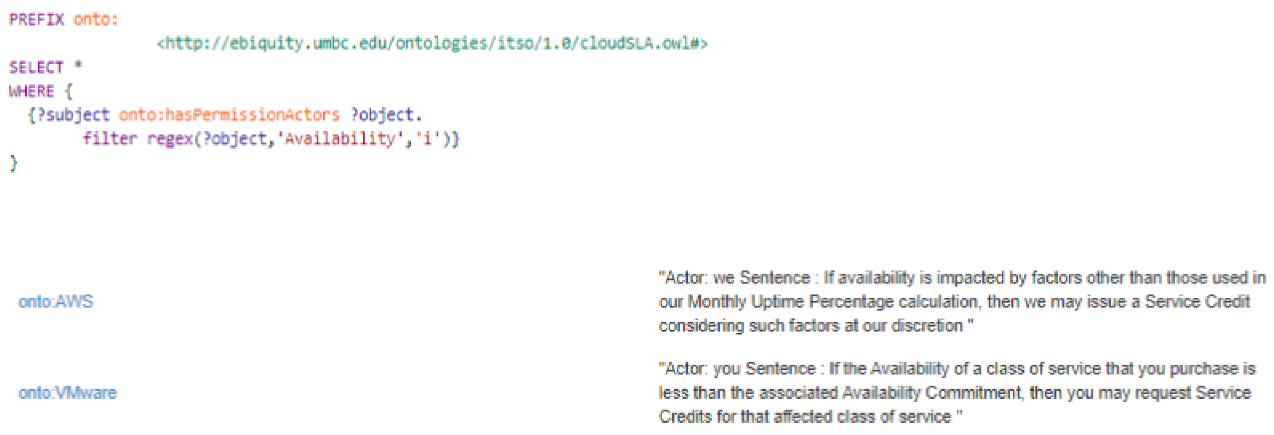
Figure 6.2: Case 1: Provide the Obligations Statements with Actors that talk about ‘Service Credits’.



```
PREFIX  onto:
    <http://ebiquity.umbc.edu/ontologies/itso/1.0/cloudSLA.owl#>
SELECT *
WHERE {
  {?subject onto:hasRemedyforNonCompliance ?object.
        filter regex(?object,'95','i')}
```

subject	object
onto:AWS	"Monthly Uptime Percentage: Less than 95.0% Service Credit Percentage : 100% "
onto:Alibaba	"Monthly Uptime Percentage: Lower 95% Service Credit Percentage : 100% of the Monthly Service Fee per Instance "
onto:GCP	"Monthly Uptime Percentage: 95.00% - < 99.00% Service Credit Percentage : 25% "
onto:Alibaba	"Monthly Uptime Percentage: Lower than 99% but equal to or higher than 95% Service Credit Percentage : 25% of the Monthly Service Fee per Instance "
onto:GCP	"Monthly Uptime Percentage: < 95.00% Service Credit Percentage : 50% "
onto:AWS	"Monthly Uptime Percentage: Less than 99.0% but equal to or greater than 95.0% Service Credit Percentage : 30% "
onto:Azure	"Monthly Uptime Percentage: < 95% Service Credit Percentage : 100% "

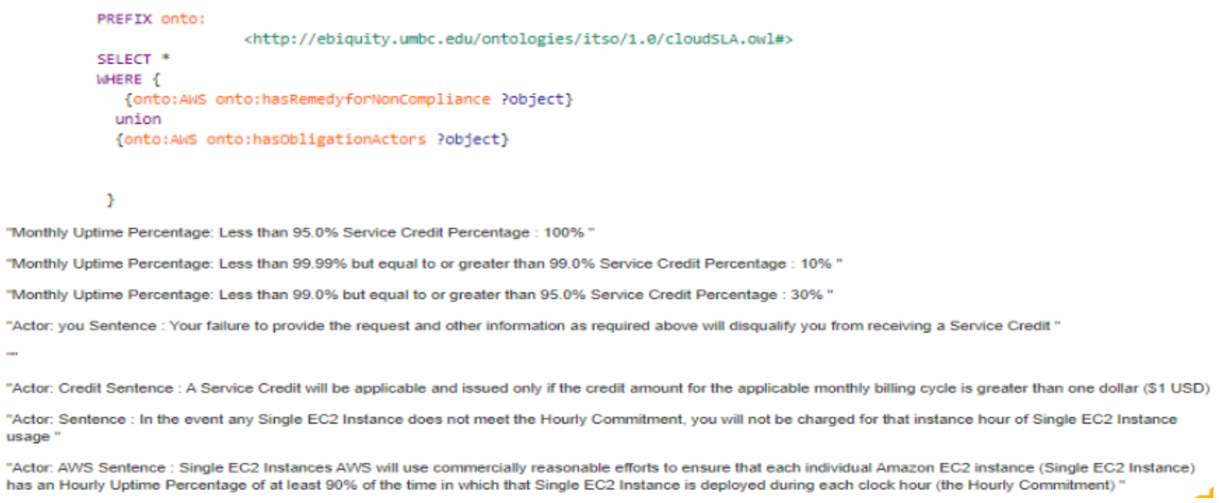
Figure 6.3: Case 2: Compare the credits refunded by Cloud Providers if the availability is around 95%



```
PREFIX onto:
              <http://ebiquity.umbc.edu/ontologies/itso/1.0/cloudSLA.owl#>
SELECT *
WHERE {
  {?subject onto:hasPermissionActors ?object.
        filter regex(?object,'Availability','i')}
}
```

onto:AWS	"Actor: we Sentence : If availability is impacted by factors other than those used in our Monthly Uptime Percentage calculation, then we may issue a Service Credit considering such factors at our discretion "
onto:VMware	"Actor: you Sentence : If the Availability of a class of service that you purchase is less than the associated Availability Commitment, then you may request Service Credits for that affected class of service "

Figure 6.4: Case 3: Looking for Permission with respect to the 'Availability'



```
PREFIX onto:
              <http://ebiquity.umbc.edu/ontologies/itso/1.0/cloudSLA.owl#>
SELECT *
WHERE {
   {onto:AWS onto:hasRemedyforNonCompliance ?object}
  union
  {onto:AWS onto:hasObligationActors ?object}


 }
```

"Monthly Uptime Percentage: Less than 95.0% Service Credit Percentage : 100% "

"Monthly Uptime Percentage: Less than 99.99% but equal to or greater than 99.0% Service Credit Percentage : 10% "

"Monthly Uptime Percentage: Less than 99.0% but equal to or greater than 95.0% Service Credit Percentage : 30% "

"Actor: you Sentence : Your failure to provide the request and other information as required above will disqualify you from receiving a Service Credit "

""

"Actor: Credit Sentence : A Service Credit will be applicable and issued only if the credit amount for the applicable monthly billing cycle is greater than one dollar ($1 USD)

"Actor: Sentence : In the event any Single EC2 Instance does not meet the Hourly Commitment, you will not be charged for that instance hour of Single EC2 Instance usage "

"Actor: AWS Sentence : Single EC2 Instances AWS will use commercially reasonable efforts to ensure that each individual Amazon EC2 instance (Single EC2 Instance) has an Hourly Uptime Percentage of at least 90% of the time in which that Single EC2 Instance is deployed during each clock hour (the Hourly Commitment) "

Figure 6.5: Case 4: Find the Obligations and refunds provided by AWS

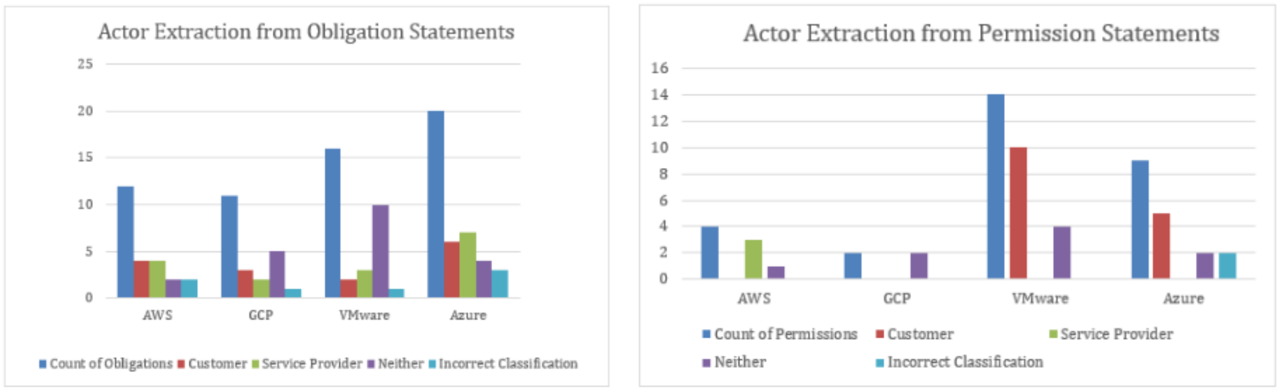


Figure 6.6: Actors Distribution Plot

request and issue a Service Credit when VMware validates the SLA Event based on VMware's data and records "

Actor: Customer *"Actor: you Sentence : Service Credits will not entitle you to any refund or other payment from AWS "*

Neither: *"Actor: Sentence : Capitalized terms not defined in this SLA will have the meanings specified in the Terms of Service "*

Incorrect Extraction: *"Actor: Credit Sentence : If your subscription term for the Service Offering expires or is terminated prior to the issuance of a Service Credit, the Service Credit will become void as of the date of the expiration or termination "* No actor should be extracted for the above statement.

6.2.1 Actor distribution plot

We plotted the distribution of the different categories of actors across the obligation and permission statements of various SLA. En example of the distribution is as shown in 6.6

We see that some sentences have been incorrectly classified, an example of incorrectly classified statement is in

Azure:

"Actor: reseller Sentence : If you purchased a Service from a re seller, you will receive a service credit directly from your re seller and the re seller will receive a Service Credit directly from us "

"Actor: you Sentence : For these Services, any Service Credit that you may be eligible for will be credited in the form of service time (i.e., days) as opposed to service fees, and any references to Applicable Monthly Service Fees is deleted and replaced by Applicable Monthly Period. "

The reason of this incorrect classification could be the complexity of the statement and also the limitation of the framework to understand third part and reseller as actors.

6.2.2 Correctness of Actor extraction by survey

We evaluated the captured actors by randomly selecting 10 deontic statements. These statements were sent to 5 individuals to identify the actors in them if there are any.The statements sent for evaluation were

1. AWS will use commercially reasonable efforts to make the Included Services each available for each AWS region with a Monthly Uptime Percentage of at least 99.99%, in each case during any monthly billing cycle (the Service Commitment).

2. In the event any Single EC2 Instance does not meet the Hourly Commitment, you will not be charged for that instance hour of Single EC2 Instance usage.

3. For all Virtual Machines that have two or more instances deployed across two or more Availability Zones in the same Azure region, we guarantee you will have Virtual Machine Connectivity to at least one instance at least 99.99% of the time.

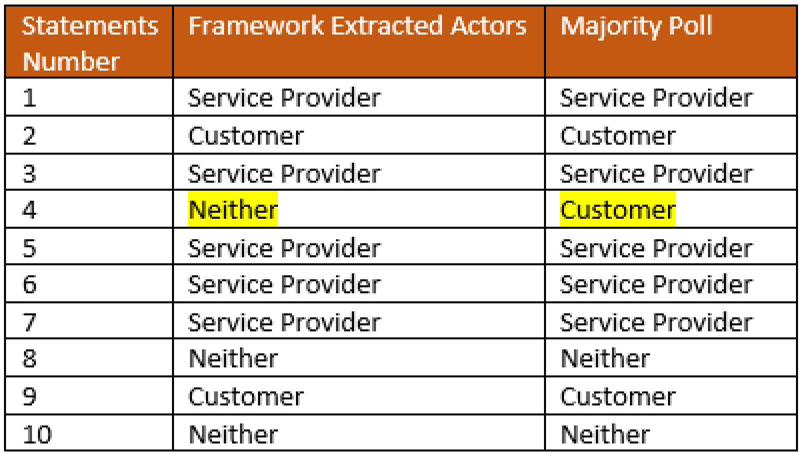
4. If the Agreement authorizes the resale or supply of Google Cloud Platform under a Google Cloud partner or reseller program, then all references to Customer in this SLA mean Partner or Reseller (as applicable), and any Financial Credit(s) will only apply for impacted Partner or Reseller order(s) under the Agreement.

5. If a dispute arises with respect to this SLA, Google will make a determination in good faith based on its system logs, monitoring reports, configuration records, and other available information, which Google will make available for auditing by Customer at Customers request.

6. At our discretion, we may issue the Service Credit to the credit card you used to pay for the billing cycle in which the Unavailability occurred.

7. A “Service Credit” is a dollar credit, calculated as set forth above, that we may credit back to an eligible account

8. The self-service console, available at https://vchs.vmware.com, cannot successfully authenticate a simulated user for more than five (5) consecutive minutes.



Statements Number	Framework Extracted Actors	Majority Poll
1	Service Provider	Service Provider
2	Customer	Customer
3	Service Provider	Service Provider
4	Neither	Customer
5	Service Provider	Service Provider
6	Service Provider	Service Provider
7	Service Provider	Service Provider
8	Neither	Neither
9	Customer	Customer
10	Neither	Neither

Figure 6.7: Evaluation of the framework but checking the correctness of actors capturing

9. You may not unilaterally offset your Applicable Monthly Service Fees for any performance or availability issues.

10. Capitalized terms used but not defined in this SLA will have the meaning assigned to them in the Agreement.

The results obtained are as shown in Figure 6.7 We notice that there is a mismatch in the majority poll and the extracted actor for statement 4, and this is because of the complexity of the statement. Though there is no well defined actor in statement 4, (according to dependency parsing rule), human beings could read, understand the statement and decipher that the actor is ‘Customer’ which the system is failing to do. This is expected while building the system.

In the next chapter we discuss conclude the thesis and discuss about the future work

Chapter 7: Conclusions and Future Work

7.1 Conclusions

Currently, the cloud service SLA is text documents that require manual effort to read through and choose the right service. We have extracted the text from these documents and built a publicly available automatic extraction and monitoring of cloud service level agreements system, which is done with the help of semantic web technologies and text mining techniques involving the usage of OWL and RDF graph. In this paper, we have described in detail the cloud SLA ontology that we have developed. We have also described the prototype that we have developed to illustrate how the SLA measures can be automatically extracted from legal terms of service or customer agreement document, that are available in the public domain.

For our initial study, we selected four providers that are the main providers of cloud-based services. In the following work we have identified and extracted modalities from SLA documents. Using which we have extracted the modal verbs and their associated actors. With this information, we then classify the modal verbs as permissions and obligations. Each permission/obligation is associated with specific actors. The actors here are 'Customers' and 'Service Providers'. Doing this, we have eased the process of finding the most suitable service provider for every customer. The customers now have the clarity of what their rights are and what are they obliged to do.

7.2 Future Work

In our future work, we intend to include more information about the SLA like compliance, which could be done by adding a new class to the base ontology. We also intend on making the ontology more involved so that it can capture the complexity of the sentences which were incorrectly classified and tagged. We are planning on expanding the number of service providers we are checking. We also intend to do the automation across domains rather than just cloud service providers. We also intend on exploring newer small text modelling techniques for better capture of important topics from the SLA as the traditional topic modeling does-not give satisfactory results due to the small size of the SLA. Our question and answering system is closed domain for now and we intend on making it open domain. As part of future work, we will also expand the data-set for our system to include all legal documents associated with cloud services like terms of service, privacy policy, and service level agreements. This would also include a streamlined auto service selection process.

Appendix A: Python code to Automate Cloud SLA

```
from nltk import sent_tokenize

import preProcessingUtils as utils

import OntologyUtils as ontUtils

cloudServiceProviders = \

[

    'GCP',

    'AWS',

    'Azure',

    'VMware',

    'Alibaba',

]

print("\nProcessing the following SLAs : \n")

utils.displayProviders(cloudServiceProviders)

for eachProvider in cloudServiceProviders:

    print("Processing - ",eachProvider)

    soup = utils.getSoup(eachProvider)

    text = utils.getTextFromSoup(soup,eachProvider)
    tokenized = utils.customTokinizer(text)

    ontUtils.getDefinitions(eachProvider,tokenized)

    ontUtils.getDeonticLogic(eachProvider,sent_tokenize(text))

    ontUtils.getActors(eachProvider)

    ontUtils.getNonComplianceRemedy(soup,eachProvider)

print("Populating Ontology\n")

print("Loading OWL \n")

onto = ontUtils.loadOwl()

# print("Clearning existing instances in OWL")

# ontUtils.clearInstances(onto,['CloudSLA','RemediesforNonCompliance'])

print("Creating data properties \n")

ontUtils.createDataProperties(onto)

print("Creating a struction in the ontology \n")

ontUtils.createOntStructure(onto)

print("Creating instances \n")

populationStatStr = ""

for eachProvider in cloudServiceProviders:

    populationStatStr += ontUtils.createInstances(eachProvider,onto)


utils.writePopluationStats(populationStatStr)

onto.save()

print("Ontology updated and saved!")
```

```
#CODE FOR PRE-PROCESSING TEXT



from urllib.request import urlopen

from bs4 import BeautifulSoup

from nltk import word_tokenize, PunktSentenceTokenizer

import os

import shutil


slaURLDict = \

    {

        'GCP' : 'https://cloud.google.com/compute/sla',

        'AWS' : 'https://aws.amazon.com/compute/sla/',

        'Azure' : 'https://azure.microsoft.com/en-us/'

               'support/legal/sla/virtual-machines/v1_8/',

        'VMware':'https://www.vmware.com/'

                 'support/vcloud-air/sla.html',

        'IBM':'https://cloud.ibm.com/docs/overview?topic=overview-slas',

        'Alibaba':'https://www.alibabacloud.com/help/doc-detail/42436.htm'


    }


def displayProviders(cloudServiceProviders):
    for eachProvider in cloudServiceProviders:

        print("{0} : {1} ".format(eachProvider,slaURLDict[eachProvider]))




def getSoup(provider):

    page = urlopen(slaURLDict[provider])

    pageRead = page.read().decode('utf-8', 'ignore')

    page.close()

    soup = BeautifulSoup(pageRead, 'lxml')

    # soup = BeautifulSoup(pageRead, 'html.parser')


    return soup




def getTextFromSoup(soup,provider):

    if provider=="GCP":

        rawText = soup.find_all("div", class_="devsite-article-body clearfix")

        text = ' '.join(map(lambda p: p.text, rawText))

    elif provider=="AWS":

        rawText = soup.find_all("div", {"class":"lb-col lb-tiny-24 lb-mid-24"})

        text = ' '.join(map(lambda p: p.text, rawText))

    elif provider=="Azure":

        rawText = soup.find_all("div", {"class": "section"})
        text = ' '.join(map(lambda p: p.text, rawText))

    elif provider == "VMware":

        rawText = soup.find_all("div", {"class": "container-fluid"})

        text = ' '.join(map(lambda p: p.text, rawText))

    elif provider == "Alibaba":

        rawText = soup.find_all("div", {"class": "col-lg-7 col-md-7'

        'col-sm-8 col-xs-12 doc-content"})

        text = ' '.join(map(lambda p: p.text, rawText))

    elif provider == "IBM":

        rawText = soup.find_all("div")

        text = ' '.join(map(lambda p: p.text, rawText))

    else:

        rawText = soup.find_all("div", class_="devsite-article-body clearfix")

        text = ' '.join(map(lambda p: p.text, rawText))

    return text



def customTokinizer(text):

    custom_token = PunktSentenceTokenizer(text)

    tokenized = custom_token.tokenize(text)

    return tokenized


def tokenizeWords(text):
    print (text)

    print("Tokenizing words...")

    words = word_tokenize(text.decode().lower())

    return words


def createFolders(folders):

    for eachFolder in folders:

        dir = eachFolder

        if os.path.exists(dir):

            shutil.rmtree(dir)

        os.makedirs(dir)


def removeDuplicates(theString):

    theList = []

    for line in theString.split('\n'):

        if line not in theList:

            theList.append(line)


    theMerged = "\n".join(theList)

    return theMerged


def textFormater(text):

    if isinstance(text, list):
        text = "\n".join(text)
        text = text.replace("\n", " ")
        text = text.split(". ")
    else:
        text = text.replace("\n", "")
        text = "\n".join(text.split(". "))

    return text

def writePopluationStats(str):
    file = open('PopulationStats.txt','w')
    file.write(str)
    file.close()

def CheckPopulationCount(PermState_Count,PermAct_Count,
OblState_Count,OblAct_Count,NonComp_Count):
    count = 0
    if PermState_Count > 0:
        count +=1
    if PermAct_Count > 0:
        count +=1
    if OblState_Count > 0:
        count +=1
    if OblAct_Count > 0:

        count +=1

    if NonComp_Count > 0:

        count +=1

    return count


#CODE FOR ONTOLOGY AUTO POPULATION



import spacy

from owlready2 import *

import owlready2 as owl

import preProcessingUtils as utils

deonticDict = \

    {

        'Obligation' : ['must', 'should', 'have to', 'will'],

        'Permission' :  ['can', 'may', 'could']

    }

owlPath = "C:\\Users\divya\Desktop\Thesis\\3_3_2020\cloudSLA_Final.owl"

actors = ['aws', 'customer', 'we', 'you', 'amazon',

        'google', 'gcp', 'microsoft', 'azure', 'vmware',

        'oracle', 'alibaba','Alibaba Could', 'us']
```

```
def getDefinitions(provider, tokenized):

    tokenized = utils.textFormater(tokenized)

    fileName = "Definitions_" + provider + ".txt"

    writeBack = ""

    file = open(fileName, 'w')

    for i in tokenized:

        r1 = re.findall("[a-zA-Z\ ":\"()]+(?:is"

        "calculated|refers|means|is a|is"

        "defined|mean)+[\na-zA-Z" "0-9!@#$&()\\-`.+,/\:"\*%]+", i)

        if len(r1) != 0:

            writeBack += str(r1) + '\n'

    writeBack = utils.removeDuplicates(writeBack)

    file.write(writeBack)

    file.close()




def getDeonticLogic(provider,text):

    text = utils.textFormater(text)

    for deonticType in deonticDict:

        file = open(deonticType+"Statements_" + provider + ".txt", 'w')

        wordMerge = ""

        counter = 0

        deonticCount = 0
        for s in text:
            counter += 1
            if any(word in s for word in deonticDict[deonticType]):
                deonticCount += 1
                wordMerge += str(s) + '\n'
        wordMerge = utils.removeDuplicates(wordMerge)
        file.write(wordMerge)
        file.close()

def getActors(eachProvider):
    nlp = spacy.load("en_core_web_sm")
    for deonticType in deonticDict:
        f = open(deonticType+'Statements_'+eachProvider+'
        .txt', 'r', encoding='utf-8', errors='ignore')
        text = f.readlines()
        fw = open(deonticType+'Actors_'+eachProvider+'.txt','w')
        for sent in text:
            doc = nlp(sent)
            Actor = ""
            for previous, current in zip(doc, doc[1:]):
                nouns = ['NNP', 'NN', 'NNS', 'NNPS']
                if ((previous.dep_) == 'nsubj') & ((current.tag_) == 'MD'):
                    Actor = previous
                elif ((previous.dep_) == 'ROOT') & ((current.dep_) == 'dobj'):
                    Actor = current
            fw.writelines("Actor: " + str(Actor) + "     "
            + "Sentence : " + str(sent) + '\n')

        fw.close()
        f.close()

def getNonComplianceRemedy(soup,provider):
    mergedText = ""
    table = soup.find('table')
    table_rows = table.find_all('tr')
    newonw = open("NonComplianceRemedies_" + provider + ".txt", 'w')
    for tr in table_rows:
        count=0
        th = tr.find_all('th')
        td = tr.find_all('td')
        for i in td:
            if "%" in i.text and provider!='VMware':
                if count == 1:
                    mergedText += '              Service
                    Credit Percentage : '+i.text + '\n'
                    count=0
                else:
                    mergedText += 'Monthly Uptime Percentage: '  + i.text
                    count+=1

    newonw.write(mergedText)
    newonw.close()

def loadOwl():
    onto_path.append(owlPath)
    onto = get_ontology(owlPath)
    onto.load()
    return onto

def createDataProperties(onto):
    with onto:
        class hasRemedyforNonCompliance(DataProperty):
            domain = [onto.RemediesforNonCompliance]
            range = [str]
    with onto:
        class hasObligationActors(DataProperty):
            domain = [onto.Obligation]
            range=[str]
    with onto:
        class hasPermissionActors(DataProperty):
            domain=[onto.Permission]
            range=[str]
    with onto:
        class hasObligationStatements(DataProperty):
            domain=[onto.Obligation]
            range=[str]
            pass
    with onto:
        class hasPermissionStatements(DataProperty):
            domain=[onto.Permission]
            range=[str]
            pass



def createOntStructure(onto):
    with onto:
        class Statments(Thing):
            pass
    with onto:
        class CloudSLA(Thing):
            pass
    with onto:
        class Standards(Thing):
            pass
    with onto:
        class TermAndAdjustments(Thing):
            pass
    with onto:
        class RemediesforNonCompliance(Thing):
            pass
    with onto:
        class DeonticStatements(Thing):
            pass
    with onto:
        class Obligation(DeonticStatements):
            pass
    with onto:
        class Permission(DeonticStatements):
            pass

def createInstances(provider,onto):
    DefState_Count = 0
    PermState_Count = 0
    PermAct_Count = 0
    OblState_Count = 0
```

```
OblAct_Count = 0

NonComp_Count = 0

instance = onto.CloudSLA(provider)

instance.is_a.append(onto.CloudSLA)

instance.is_a.append(onto.Permission)

instance.is_a.append(onto.Obligation)

instance.is_a.append(onto.RemediesforNonCompliance)

instance.is_a.append(onto.Standards)

defStatFile = open("Definitions_" + provider + ".txt", 'r')

permStatFile = open("PermissionStatements_" + provider + ".txt", 'r')

obligStatFile = open("ObligationStatements_" + provider + ".txt", 'r')

permActorFile = open("PermissionActors_" + provider + ".txt", 'r')

obligActorFile = open("ObligationActors_" + provider + ".txt", 'r')

nonCompRemFile = open("NonComplianceRemedies_" + provider + ".txt", 'r')

instance.hasObligationStatements = [""]

instance.hasPermissionStatements = [""]

instance.hasPermissionActors = [""]

instance.hasObligationActors = [""]

instance.hasRemedyforNonCompliance = [""]

instance.hasStandards = [""]

for defStat in defStatFile.readlines():

    DefState_Count += 1

    instance.hasStandards.append(defStat)
```

```
for oligStat in obligStatFile.readlines():

    OblState_Count += 1

    instance.hasObligationStatements.append(oligStat)

for permStat in permStatFile.readlines():

    PermState_Count += 1

    instance.hasPermissionStatements.append(permStat)

for permAct in permActorFile.readlines():

    PermAct_Count += 1

    instance.hasPermissionActors.append(permAct)

for obligAct in obligActorFile.readlines():

    OblAct_Count += 1

    instance.hasObligationActors.append(obligAct)

for nonCompRem in nonCompRemFile.readlines():

    NonComp_Count += 1

    instance.hasRemedyforNonCompliance.append(nonCompRem)

populationCount = utils.CheckPopulationCount(PermState_Count,PermAct_Count,

OblState_Count,OblAct_Count,NonComp_Count)

populationStatStr = ('\n######### {} Stats ######### \n '

      '------- Population Count : {} ------- \n '

      'ObligationStatements : {} \n '

      'ObligationActors : {} \n '

      'PermissionStatements : {} \n '

      'PermissionActors : {} \n '
```

```
            'RemedyForNonCompliance : {} \n\n'
            .format(provider,populationCount,OblState_Count,
            OblAct_Count,PermState_Count,PermAct_Count,NonComp_Count))
    return populationStatStr


def clearInstances(onto,fClass):
    with onto:
        class fClass(Thing):
            pass
    for eachClass in onto.get_instances_of(fClass):
        owl.destroy_entity(eachClass)
    onto.save()
```

Bibliography

[1] S. A. Baset, "Cloud slas: present and future," *SIGOPS Operating Systems*, 2012.

[2] K. P. Joshi, Y. Yesha, and T. Finin, "Automating cloud services life cycle through semantic technologies," *Services Computing, IEEE Transactions on*, vol. 7, no. 1, pp. 109–122, 2014.

[3] S. Mittal, K. P. Joshi, C. Pearce, and A. Joshi, "Automatic extraction of metrics from slas for cloud service management," in *IEEE International Conference on Cloud Engineering*, 2016.

[4] "International organization for standardization's iso/iec dis 19086." [Online]. Available: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=67545

[5] "Cloud computing service metrics description." [Online]. Available: http://www.nist.gov/itl/cloud/upload/RATAX-CloudServiceMetricsDescription-DRAFT-20141111.pdf

[6] "Federal risk and authorization management program (fedramp)." [Online]. Available: http://csrc.nist.gov/groups/SMA/ispab/documents/minutes/2012-02/feb3_fedramp_ispab.pdf

[7] "Resource description framework (rdf)." [Online]. Available: http://www.w3.org/RDF/

[8] "Owl web ontology language." [Online]. Available: http://www.w3.org/TR/owl-features/

[9] E. Christensen and G. Meredith, "Web services description language (wsdl) 1.1," 2001.

[10] M. Turner, D. Budgen, and P. Brereton, "Turning software into a service." *Computer.*, vol. 36, no. 10, pp. 38–44, 2003.

[11] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour, "Semantic ws-agreement partner selection," in *Proceedings of the 15th international conference on World Wide Web.* ACM, 2006, pp. 697–706.

[12] 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, 2007, p. 216.

[13] T. Bui and A. Gachet, "Web services for negotiation and bargaining in electronic markets: Design requirements and implementation framework," in *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on.* IEEE, 2005, pp. 38–38.

[14] H. Skogsrud, B. Benatallah, and F. Casati, "Trust-serv: model-driven lifecycle management of trust negotiation policies for web services," in *Proceedings of the 13th international conference on World Wide Web.* ACM, 2004, pp. 53–62.

[15] Y. Yao, F. Yang, and S. Su, "Flexible decision making in web services negotiation," in *Artificial Intelligence: Methodology, Systems, and Applications.* Springer, 2006, pp. 108–117.

[16] "Protege." [Online]. Available: https://protege.stanford.edu/

[17] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality."

[18] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik, and D. Mladenic, "Triplet extraction from sentences," in *Proceedings of the 10th International Multiconference" Information Society-IS*, 2007, pp. 8–12.

[19] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Unsupervised named-entity extraction from the web: An experimental study," *Artificial intelligence*, vol. 165, no. 1, pp. 91–134, 2005.

[20] K. Barker and N. Cornacchia, "Using noun phrase heads to extract document keyphrases," in *Advances in Artificial Intelligence.* Springer, 2000, pp. 40–52.

[21] Z. S. V. Mulwad, T. Finin and A. Joshi, "Using linked data to interpret tables," in *Proceedings of the First International Conference on Consuming Linked Data-Volume 665. CEUR-WS. org*, 2010, pp. 109–120.

[22] T. N. C. S. Bhagavatula and D. Downey, ""tabel: Entity linking in web tables," in *The Semantic Web-ISWC 2015. Springer*, 2015, p. 425–441.

[23] "Word2vec." [Online]. Available: https://en.wikipedia.org/wiki/Word2vec

[24] L. Kagal and T. Finin, "Modeling conversation policies using permissions and obligations," *Autonomous Agents and Multi-Agent Systems*, 2006.

[25] R. Sipoš, D. Mladenić, M. Grobelnik, and J. Brank, "Modeling common real-word relations using triples extracted from n-grams," in *The Semantic Web*. Springer, 2009, pp. 16–30.

[26] "Deontic logic." [Online]. Available: https://plato.stanford.edu/entries/logic-deontic/

[27] A. Gupta, S. Mittal, K. P. Joshi, C. Pearce, and A. Joshi, "Streamlining Management of Multiple Cloud Services," in *IEEE International Conference on Cloud Computing*. IEEE Computer Society, June 2016, p. 8.

[28] "Guide to information technology security services." [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-35.pdf

[29] S. Kumar and R. B. Mishra, "An approach to multi-attribute negotiation between semantic web services," *International Journal of Web Engineering and Technology*, vol. 5, no. 2, pp. 162–186, 2009.

[30] "Sparql 1.1 overview." [Online]. Available: http://www.w3.org/TR/sparql11-overview/

[31] "Vmware service level agreement." [Online]. Available: https://www.vmware.com/support/vcloud-air/sla/

[32] "Amazon service level agreement." [Online]. Available: https://aws.amazon.com/compute/sla/

[33] "Microsoft azure sla." [Online]. Available: https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/v1_8/

[34] "Google compute engine sla." [Online]. Available: https://cloud.google.com/compute/sla

[35] "Service level agreement for ibmsoftlayer." [Online]. Available: https://www.softlayer.com/sites/default/files/sla.pdf

[36] "Jena apache fuseki: serving rdf data over http,." [Online]. Available: http://jena.apache.org/documentation/serving_data/index.html

[37] "Link grammar." [Online]. Available: http://www.link.cs.cmu.edu/link/

[38] "Regular expression." [Online]. Available: https://docs.python.org/3/library/re.html/link/

[39] T. D. Breaux and A. I. Anton, "Analyzing goal semantics for rights, permissions, and obligations," in *RE'05: Proceedings of the 13th IEEE International Requirements Engineering Conference (RE'05)*. IEEE Computer Society, August 2005, pp. 177–186.

[40] “Dependency parsing.” [Online]. Available: http://nlpprogress.com/english/dependency_parsing.html

[41] “Auxiliary dependency.” [Online]. Available: https://universaldependencies.org/u/dep/aux_.html

[42] “Nominal subject dependency.” [Online]. Available: https://universaldependencies.org/u/dep/nsubj.html