AN ONTOLOGY-BASED EXPERT SYSTEM TO DETECT SERVICE LEVEL AGREEMENT VIOLATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ALPER KARAMANLIOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2017

Approval of the thesis:

AN ONTOLOGY-BASED EXPERT SYSTEM TO DETECT SERVICE LEVEL AGREEMENT VIOLATIONS

submitted by **ALPER KARAMANLIOĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Prof. Dr. Ferda Nur Alpaslan
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Halit Oğuztüzün
Computer Engineering Department, METU _____

Prof. Dr. Ferda Nur Alpaslan
Computer Engineering Department, METU _____

Prof. Dr. Ahmet Coşar
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Pınar Karagöz
Computer Engineering Department, METU _____

Assist. Prof. Dr. Orkunt Sabuncu
Computer Engineering Department, TEDU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ALPER KARAMANLIOĞLU

Signature :

ABSTRACT

AN ONTOLOGY-BASED EXPERT SYSTEM TO DETECT SERVICE LEVEL AGREEMENT VIOLATIONS

Karamanlıoğlu, Alper
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. Ferda Nur Alpaslan

September 2017, 48 pages

In this thesis, an expert system developed with an ontology-based approach to detect Service Level Agreement (SLA) violations is presented. Ontologies represent explicit formal specifications of the concepts in a particular domain and the relationships among them. Expert systems, however, are frequently employed with ontologies because of their reasoning capabilities. The widespread use of SLAs in various areas complicates SLA management and in particular the detection of violations. Although it is necessary to automatically detect SLA violations, developing a different solution for each domain is quite costly. In SLAs of multiple domains, the concepts were examined, and many common concepts have been identified. Identifying familiar concepts in different SLA areas has allowed us to acquire the idea of creating a generic SLA ontology. After generic SLA ontology was created, an expert system was developed using this ontology. The developed expert system is designed to detect SLA violations, check constraints, and make inferences. The developed system has been tested on the SLA data of the telecommunication domain. The results show that the proposed system can correctly detect SLA violations.

Keywords: Constraint Checking, Expert System, Inference, Ontology, Performance Indicator, Semantic Web, Service Level Agreement, Quality of Service

ÖZ

HİZMET DÜZEYİ ANLAŞMA İHLALLERİNİ TESPİT ETMEK İÇİN
ONTOLOJİYE DAYALI BİR UZMAN SİSTEM

Karamanlıoğlu, Alper
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Ferda Nur Alpaslan

Eylül 2017 , 48 sayfa

Bu tezde Hizmet Düzeyi Anlaşması (SLA) ihlallerini saptamak için ontolojiye dayalı bir yaklaşımla geliştirilen bir uzman sistem sunulmaktadır. Ontolojiler belirli bir alandaki kavramların açık resmi özelliklerini ve aralarındaki ilişkileri temsil eder. Bununla birlikte, uzman sistemleri, akıl yürütme yetenekleri nedeniyle ontolojilerle birlikte sıkça kullanılır. Çeşitli alanlarda yaygın SLA kullanımı, SLA yönetimini ve özellikle ihlallerin tespitini zorlaştırmaktadır. Her ne kadar SLA ihlallerini otomatik olarak algılamak gerekirse de, her alan için farklı bir çözüm geliştirmek oldukça maliyetli olmaktadır. Birden fazla alandaki SLA kavramları incelenmiş ve pek çok ortak kavram belirlenmiştir. Farklı SLA alanlarında tanıdık kavramların belirlenmesi, jenerik bir SLA ontolojisi yaratma fikrini edinmemizi sağladı. Jenerik SLA ontolojisi oluşturulduktan sonra, bu ontoloji kullanılarak bir uzman sistem geliştirmiştir. Geliştirilen uzman sistemi, SLA ihlallerini tespit etmek, kısıtlamaları kontrol etmek ve çıkarımlar yapmak için tasarlanmıştır. Geliştirilen sistem telekomünikasyon alanına ait veriler üzerinde test edilmiştir. Elde edilen sonuçlar önerilen sistemin doğru bir şekilde ihlalleri tespit edebildiğini göstermektedir.

Anahtar Kelimeler: Anlamsal Ağ, Çıkarım, Hizmet Düzeyi Anlaşması, Hizmet Kalitesi, Kısıt Denetimi, Ontoloji, Performans Göstergesi, Uzman Sistem

To my family

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to my supervisor Prof. Dr. Ferda Nur Alpaslan for her constant support, encouragement, and guidance throughout my thesis study. It was always a pleasure working with her.

I would like to thank INNOVA IT Solutions Inc for their interest in this study, and all opportunities they have provided us in their company. This work is also supported by TÜBİTAK-TEYDEB Technology and Innovation Funding Programs (Project No: 3150860).

I also would like to thank the members of my thesis examining committee, Prof. Dr. Halit Oğuztüzün, Prof. Dr. Ahmet Coşar, Assoc. Prof. Dr. Pınar Karagöz and Assist. Prof. Dr. Orkunt Sabuncu for their valuable comments and feedback.

I am extremely thankful to Prof. Dr. Yılmaz Kılıçaslan, Prof. Dr. Ali Doğru, Prof. Dr. Göktürk Üçoluk, Dr. Selma Süloğlu and Dr. Hande Çelikkanat for their support and guidance.

I would like to thank my dear friends Mehmet Koça, Anıl Çetinkaya and Murat Öztürk for sharing a lot during this journey. We have always been like four musketeers, and we have never let small problems break our friendship. I know we are always available to help each other when we need it.

I am thankful to my friends, Tuğberk İşyapar, M. Çağrı Kaya, Alperen Dalkıran, Ahmet Rifaioğlu and Ozan Koçak for their technical and moral support that I needed most. Also, I would like to thank my friends Önder Çağlar, Hakan Yılmaz, Fatih Calip, İlhan Yumer and Mahdi Saeedi Nikoo for their support and friendship. I also would like to thank the colleagues from CENG, Alperen Eroğlu, Gökhan Özsarı, Abdullah Doğan, Aybike Şimşek Dilbaz, Ahmet Atakan and Arınç Elhan for the fun times we spent.

During the thesis work, I could not communicate with many of my friends as much as the old times, but they were always in my heart. I hope my friends Burak Koç, Ertan Kabakcı, Deniz Nuş, Çiğdem Maltepe Yılmaz, Gökhan Kısa, Cansu Batan, Furkan Yakışır, Tuğba Akbaşaran, Ecenur Demirci and Ozan Turan understand me.

Last but not least, I would like to thank my family, my parents Beyhan Karamanlıoğlu and Sema Karamanlıoğlu, and my brother Aytaç Karamanlıoğlu for their constant support and unconditional love throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

KPI	Key Performance Indicator
KQI	Key Quality Indicator
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
SHACL	Shapes Constraint Language
SLA	Service Level Agreement
SLO	Service Level Objective
SLS	Service Level Specification
SPIN	SPARQL Inferencing Notation
SWRL	Semantic Web Rule Language
TBC	TopBraid Composer
URI	Uniform Resource Identifier
XML	Extensible Markup Language
W3C	World Wide Web Consortium
WWW	World Wide Web

CHAPTER 1

INTRODUCTION

In this introductory chapter, first of all, brief background information is given. Then the problem is identified, the proposed approach is briefly described, and the contribution of the study is explained. Finally, the outline of the thesis is presented.

1.1 Background

A Service Level Agreement (SLA) can be defined as an agreement between two or more parties, one of which is a customer and the others are service providers. SLAs are contracts that involve separate organizations or different teams within an organization. SLAs have become frequently used agreements to increase accountability and quality as business volume increases in sectors where outsourcing is widespread. These agreements are most often employed in telecommunications, information technology, and healthcare sectors.

Ontologies are constructs that are used to gather information about specific fields of knowledge. Concepts of knowledge domains, and the relations between these concepts can be defined using ontology. Ontologies provide analysis and reuse of domain knowledge. Also, ontologies are used in the majority of the Semantic Web applications. Therefore, it is possible to develop specialized systems in specific areas.

Expert Systems is the type of specialized software systems that aim to solve the problems that human experts are creating better solutions. In fact, the aim of the expert systems to make decisions by imitating the domain expert. The performance of the

Expert System depends on the quality of the Knowledge Base and the Inference Engine. The Knowledge Base is a special database concept that allows the expert system to store facts about the domain. The Inference Engine serves as a mechanism of inference and control.

1.2 Problem Statement

The ability to manage Service Level Agreements as they become widespread in various sectors has become a major challenge. The SLA violations should be detected shortly after the occurrence and the necessary sanctions should be applied quickly. Manual detection of the violation involves many problems. Some of these problems are the high error rate, the excess of reaction time, and loss of work power.

Although manual detection has many challenges, automatic detection has its own difficulties. Because SLAs are used in many different domains, it is costly to develop separate software for each domain. For this reason, it is expected to develop software that can be applied to various domains as much as possible. Database systems are not specific to the domain, so they can not meet this need.

1.3 Approach

In this study, an expert system is developed to detect violations of Service Level Agreements in various areas. The expert system has been established with an ontology-based approach. The SLA ontology is designed to provide a framework for creating SLAs. While creating this framework, concepts defined in SLA field and their relations are taken into consideration. Since the ontology is designed as generic, it is intended to be used in many different knowledge domains.

SLA data is recorded as RDF triples so that semantic operations can be performed on it. In addition to SLA data, Shapes Constraint Language rules and constraints are also recorded as RDF. These data, together with SLA ontology, are all recorded in Jena TDB. Therefore, constraint checking and rule inference are performed.

With the expert system developed by using the proposed ontology, it is desired to determine whether the violation of the SLA occurs. Two different SPARQL queries have been created to monitor and infer SLA violations. The violation monitoring result is sent back as a service message, and the violation inference result is stored in TDB. The developed expert system has been tested on telecommunication data.

1.4 Contribution

This study is important in several aspects.

- A generic SLA ontology is proposed, which can be used directly in many areas and expanded in some specialized areas. Thus, reusability is achieved.
- Storage of SLA data is provided in triple stores. Therefore, many semantic operations can be performed on this data.
- An expert system has been developed to detect SLA violations using the proposed ontology. The expert system provides important contributions to accurately identify violations.
- Constraint checking and inferencing on SLA data is performed using new technologies. So, it is possible to comply with specified constraints and to infer hidden information.
- No actual SLA data was used in past studies. We used real data as well as synthetic data in this study.

1.5 Outline of Thesis

This thesis contains six chapters. The remaining five chapters are organized as follows.

In Chapter 2, some theoretical background information is provided on Service Level Agreement, ontology, the semantic web, and related issues. Also, some technical

background information about ontology development tools, data modeling languages, query language, inference languages, and their integration is given.

Chapter 3 presents related studies on SLA ontologies and SLA violation detection.

In Chapter 4, the proposed SLA ontology is introduced. Firstly, there is an overview of SLA field. After that, object and data properties of the classes and classes defined in the proposed ontology is given. Finally, the construction of the SLA ontology is shown.

Chapter 5, first, describes how SLA data is stored in a triple store as RDF triples. Then, the structure and functioning of the developed SLA Violation Detection System are explained in detail. Simulation and evaluation of the system are then mentioned. Finally, it is specified how to perform constraint checking and rule inference on SLA data stored as triples.

In Chapter 6, the thesis is concluded, and possible future studies are addressed.

CHAPTER 2

THEORETICAL AND TECHNICAL BACKGROUND

In this chapter, the theoretical and technical background information is mentioned. Semantic languages, query languages, rule engines, and related technologies are described after mentioning Service Level Agreement, semantic web, ontology and associated concepts.

2.1 Theoretical Background

2.1.1 Service Level Agreements

The definition of Service Level Agreement (SLA) is given as "the official commitment between a client and a service provider" [34]. Initially, SLAs were started to be used by the fixed line telecom operators in the late 1980s. In today's world, the usage of SLAs is so common that a company can have more than one SLA in itself, as the company being the service provider and the customers are as clients. In this way, SLAs can be very useful for a company to be able to offer the same quality of service among the different units. SLAs are also useful to assess the difference in a service that is provided by itself and a service gathered by an outside source [12]. The main focus of an SLAs is the outputs received by customers based on the provided services.

SLAs can be categorized according to the levels which they are defined. These categories are Customer based, Service based and Multilevel.

Customer based SLAs can be defined as an agreement that covers all the services

used by a specific customer group. An SLA for regulating billing and payroll system between a service provider and the finance department of an organization can be given an example of this kind of SLAs.

Service based SLAs are the agreements that are used for providing services to customers directly through the service providers. The electronic mailing system used in a company is an instance for service based SLAs.

Multilevel SLAs are the agreements that can address the requirements of a different set of customers in the same SLA.

One of the main purposes of using SLAs by enterprises is to offer a better Quality of Experience (QoE) to the clients domestically and externally [7]. QoE is a term that was defined to introduce some sort of measurement for the quality of a service or a product with respect to their performance, customer satisfaction, overall sales and delivery of these products or services. Thus, QoE allows enterprise to balance the quality level of various products according to their costs and the expectations of the customers. Because of that, it's crucial to offer the necessary distinctions among the various products or services.

Since the quality of a product or service is standardized with QoE and the definition, measured objectives of a product or service is determined via SLA, it can be said that the QoE and the SLA are related. If a SLA doesn't meet the quality mesaures determined with QoE, the SLA must be fixed. Mapping is required to match the measurements from the QoE to the objective measurements of the SLA.

In order to achieve the expected quality level by the QoE, the Key Quality Indicators (KQI) must be included in the SLA. The KQI is achieved through defining, measuring and aggreeing on some Key Performance Indicators (KPI). The relationship between the KQI, KPI and SLA is demonstrated in Figure 2.1

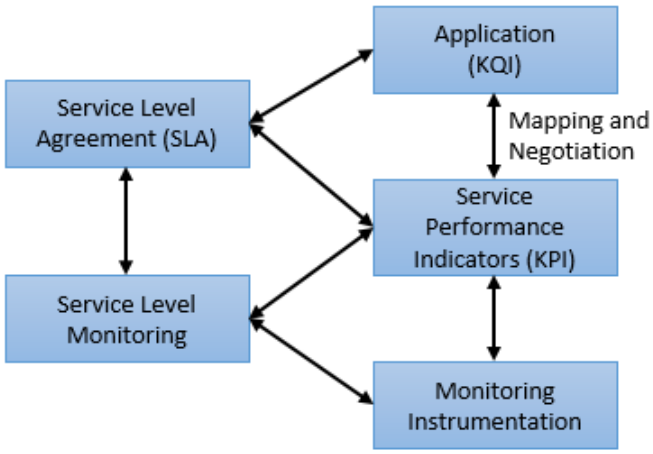


Figure 2.1: KPI, KQI and SLA Relationship. (adapted from [7])

2.1.1.1 SLA Metrics

The metrics used to measure and manage the performance characteristics of the service objects are important factors that make the agreement successful. These metrics, called SLA metrics, provide the ability to manage and measure performance compliance to SLA commitments. SLA metrics provide business continuity because of its contribution to customer satisfaction and confidence.

Metrics can be classified as direct or composite metrics. Direct metrics are derived from managed resources such as middleware, servers, or instrumented applications. Composite metrics are generated by combining direct metrics. Composite metrics are determined by averaging one or more metrics over a period of time using a particular function.

Paschke in [30] proposed three categories for structuring the SLA metrics field.

- service objects under consideration: It consists of basic service objects such as software, hardware, and network.
- ITIL processes: It clarifies responsibilities and procedures by being organized around eleven ITIL management processes.
- automation grade: It specifies the measurability of the metrics. It includes classes

such as measurable, limited measurable, and non-measurable.

2.1.2 Ontology

An ontology is an explicit specification of a conceptualization [18]. This concept, which means to *exist* in philosophy, means to be *represented* in Computer Science. The use of ontology provides many advantages when developing a system for a knowledge domain. A few of these advantages are mentioned below.

- Reusability is possible.
- Ontology is able to infer new information from past knowledge.
- Ontology focuses on meaning.
- Ontologies can be used for many different purposes and applications in various fields.

2.1.3 The Semantic Web

The Semantic Web concept [8] was introduced in 2001 by Berners-Lee et al. According to the authors, the Semantic Web will bring a meaningful structure to the content of future web pages and create an environment in which software agents can smoothly perform advanced tasks for users with page-to-page navigation. Their visions have led to the use of the Semantic Web in many academic studies and company projects. In many academic and commercial studies today, semantic web related concepts are employed.

Many languages and concepts are used in the Semantic Web. Semantic Web Stack is shown in 2.2. The most known and important ones from these concepts are RDF, RDFS, OWL, SPARQL, and SWRL. These concepts are explained in Technical Background subsection. In addition, the place of these concepts in the evolution of the web is shown in Figure 2.3.

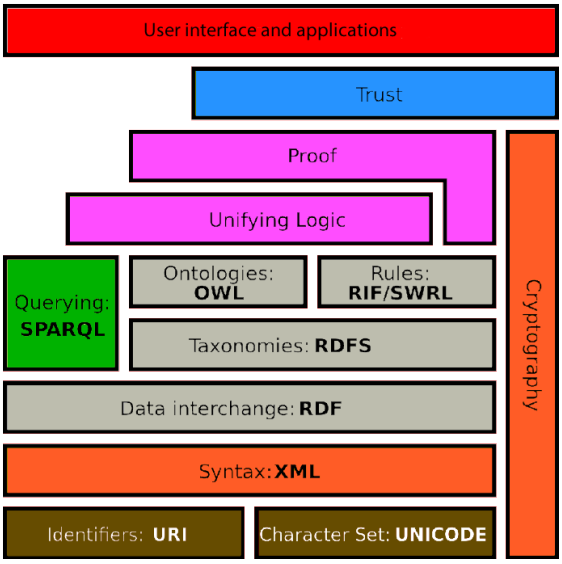


Figure 2.2: Semantic Web Stack. (adapted from [5])

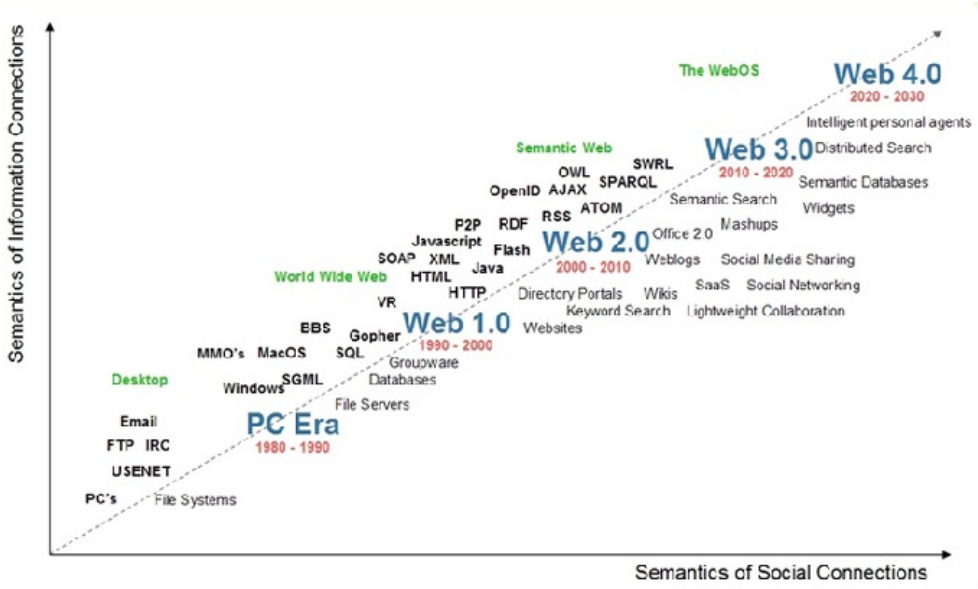


Figure 2.3: The Evolution of the Web. (taken from [6])

2.2 Technical Background

2.2.1 RDF

Resource Description Framework (RDF) [26] is a standard model and general-purpose language designed to represent information on the Web. It provides a formal definition of the meaning of information. It is possible to represent RDFs in different for-

mats such as RDF/XML, N-Triples, Turtle, N3. An RDF triple consists of a Subject, Predicate, and Object. Subject is a resource and can be distinguished by a Uniform Resource Identifier (URL). Predicate specifies the properties of relations and can be identified by URI. Object can be a resource or literal associated with Subject.

2.2.2 RDFS

The RDF Schema (RDFS) [9] extends RDF by providing a data modeling vocabulary for RDF data.

2.2.3 OWL

OWL [11] extends previous web standards such as XML, RDF, and RDFS. Ontologies can be stored in the OWL format. OWL ontologies allow classes, properties, individuals, and data values to be stored as Semantic Web documents.

2.2.4 SPARQL

SPARQL [31] is used to express queries between data sources, regardless of whether the data is structured or semi-structured. It allows developers to query RDF data that can be defined in different formats. The syntactic similarity of SPARQL with SQL provides ease of use. SPARQL's query evaluation mechanism is based on subgraph matching. SPARQL also provides a simple protocol for querying remote databases over HTTP.

2.2.5 Protégé

Protégé [28] is a free and open source knowledge-based framework and ontology editor used to build intelligent systems. Protégé is supported by a large community of government, academia, and institutional users. Using Protégé, it is possible to produce knowledge-based solutions in many domains, especially biomedicine, e-commerce and organizational modeling. Ontologies created using Protégé can be

exported in formats such as RDF, RDFS, OWL and XML Schema.

2.2.6 Topbraid Composer

TopBraid Composer (TBC) [3] is a visual modeling environment based on the Eclipse IDE and the Jena API, developed to create, manage and test ontologies and domain models. Although the TBC has paid versions, there is also a free version. TBC uses D2RQ to provide an interface to relational databases so databases can be treated as a triple store.

2.2.7 SWRL

SWRL [20] is a rule language created for semantic web applications that combines RuleML with OWL. It is suggested that an OWL axioms set be expanded to include Horn-like rules combined with the OWL knowledge base.

2.2.8 Jena Rules

Jena inference support [21] includes RDFS, OWL reasoners, as well as user defined rules. It contains a generic rule engine that can be used for many RDF processing or transformation tasks.

2.2.9 Inference Engines

An inference engine can be defined as a kind of finite state machine with a loop of three action states [33]. These states are match, select and execute rules. When these stages are over, the cycle is completed.

Two main types of inference engines are forward chaining and backward chaining.

Forward chaining, also called forward reasoning, can be logically defined as the repeated application of modus ponens.

Backward chaining is a method of inference that can be defined colloquially as it works backwards from the goals.

2.2.10 SPARQL Inferencing Notation

A SPARQL-based language, SPIN (SPARQL Inference Notation) [23], has the ability to define constraints and rules for the Semantic Web. SPIN combines concepts of query languages, rule-based systems, and object-oriented languages to define object behavior on data on the web. Therefore, there is no need to use different languages for querying and rules. SPIN components are shown Figure 2.4:

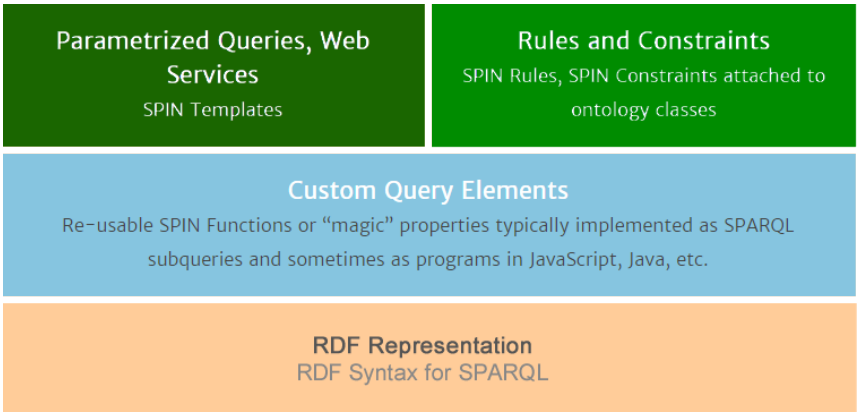


Figure 2.4: The Components of SPIN. (adapted from [4])

RDF Representation: SPIN provides a vocabulary for SPARQL queries to be represented as RDF triples. Ontology models and SPIN queries are stored together in RDF. Storing queries in RDF format makes it easy to share them on the semantic web. SPIN manages namespaces only once; so there is no need to manage every query.

Custom Query Elements: SPIN improves SPARQL expressivity by allowing custom SPARQL functions to be defined and allows to create modular queries. SPIN functions can also be defined in languages other than SPARQL such as Javascript. These functions are stored in RDF format. Using SPIN functions, queries can be simplified and common query patterns can be reused.

Rules and Constraints: SPIN does not require the use of any other rule language, as the rules can be expressed in SPARQL. SPARQL CONSTRUCT or UPDATE requests

are used to create or remove the rules. Thus, new information is inferred from existing data. SPARQL CONSTRUCT or ASK queries or corresponding SPIN Templates are used to specify data quality constraints. These constraints are often used to check the validity of the data. Since SPIN runs directly on RDF, it does not need to be converted to another format to execute the constraints or the rules. This makes the architecture more flexible, faster and simpler.

Parametrized Queries, Web Services: With query templates, higher level domain-specific languages can be defined. Thus, rule designers do not have to be SPARQL experts; they only fill in the blanks to create SPIN queries. SPIN templates are reusable, parameterized building blocks that can be used as RESTful web services.

2.2.11 Shapes Constraint Language

Shape Constraint Language (SHACL) [24] was developed to describe and validate RDF graphs according to a set of specific conditions. The W3C approved the SHACL as an official W3C Recommendation in July 2017. SHACL was significantly affected by SPIN. For this reason, SHACL can be seen as the legitimate successor of SPIN. As in SPIN, RDF syntax is used to define SHACL. In addition, each SPIN feature has an SHACL equivalent. Some of these features are given in Figure 2.5.

SPIN Feature	SHACL Feature
Constraints in SPARQL (spin:constraint, sp:Construct)	Shapes with sh:sparql
Constraints in high-level vocabulary (spl:Attribute, spl:minCount etc)	Shapes with SHACL Core properties (sh:property, sh:minCount etc)
Inference rules in SPARQL (spin:rule, sp:Construct)	Inference rules in SPARQL (sh:rule, sh:construct)
Inference rules in high-level vocabulary (spin:rule, sp:ConstructTemplate	Inference rules with Node Expressions (sh:rule, sh:path, sh:filterShape etc
User-defined SPARQL functions (spin:Function, spin:body)	User-defines SPARQL functions (sh:Function, sh:select)
Magic Properties (spin:MagicProperty)	approx: Triple rules (sh:rule, sh:predicate etc)
JavaScript Support (SPINx) (spinx:javaScriptCode)	User-defines Javascript functions (sh:Function, sh:jsFunctionName
RDF Syntax of SPARQL queries (sp:Select, sp:Filter etc)	Limited to text strings and prefixes (sh:select, sh:prefixes etc)

Figure 2.5: Comparison of SPIN and SHACL Features. (adapted from [1])

The declaration of constraints can be achieved via the Shapes using the constraint components parameters. Constraint components are defined as IRIs, and each must have at least one mandatory parameter. However, the constraint components may have optional parameters. Each of the mandatory and optional parameters specifies a

property.

2.2.12 Apache Jena

Jena [22] is a Java based development framework that allows semantic web and linked data applications to be developed. It has several APIs as well as command line tools. Since it is free and open source, it is preferred in many applications. Jena framework can read and write RDF data in different formats, use OWL API features on ontologies, and perform SPARQL queries on RDF data or OWL ontology. Therefore, Jena is used in many applications to provide integration between ontology, query language and programming language. Jena has the ability to record, query and access in ontologies that can be considered large in size.

Jena has interfaces for modeling semantic structures. Source, Property and Literal interfaces represent the subject of a statement, predicate and literals, respectively.

There are many methods in Jena to write and read RDF as XML. With these methods, an RDF model can be saved and read at any time. Serialization of RDF graphs supported by Jena are RDF/XML, Turtle, and Notation 3.

CHAPTER 3

RELATED WORK

In this chapter, information on the studies on SLA ontologies in the literature is given.

Semantic structures are used in many studies [13, 14, 16, 35, 27, 15, 10] considering service quality. These studies do not propose a general solution enough or focuses on specific areas. For example, Dobson et al. have proposed an ontology for Quality of Service called QoSOn [13] which is particularly focused on the field of service-centric systems.

In the study performed by Paschke [29], a declarative Rule Based Service Level Agreement language is introduced that extends and adapts RuleML to meet the requirements of the SLA domain. The proposed language can be fed into a rule engine to monitor and execute the performance of contract at runtime with a machine readable and interchangeable syntax. The management, interchange, maintenance, and execution of the SLA rules have been simplified using a declarative logic-based approach. Thus, the combination and revision of contract modules and rule sets are intended to be achieved easily.

In the study conducted by Green [17], ontology-based SLA formalization is defined. The ontology described here is presented in many parts. Charging, time unit, temporal, currency, network metrics, violation, entity and SLS are defined as separate ontologies under the generic SLA ontology. The ontologies described here is in OWL format. Various formulas are determined for each ontology. SWRL is used to show the rules and constraints defined according to these formulas.

In [32], the ontology-based SLA Management (OSLAM) proposed by Seo et al.

aimed to manage and guarantee SLAs for IPTV services using SWRLs together with ontologies. The authors analyzed many IPTV PIs from a variety of standard organizations. As a result of this analysis, they extended the DEN-ng model to suggest an IPTV PI hierarchy. The SWRL rules are used to detect SLA violations, to infer hidden relationships between an SLA and a PI, and to find PI value from other PIs. Protégé and Jess have been used for the implementation and testing of the proposed OSLAM architecture. OSLAM aims to enable service providers to prevent SLA violations and to provide high-quality performance to their customers.

Hamadache and Rizou offered an SLA ontology [19] covering the entire service lifecycle based on a QoS ontology representing the QoS model to ensure and improve the evaluation of the services. In the proposed ontology, SLA is the central concept and is directly linked to QoS requirements, Actor, Role, Service, and Feedback. It is suggested that the objective feedback from the service monitoring and the subjective feedback the evaluation of customers be treated symmetrically. The SLA ontology allows a reputation calculation based on customer profiles according to their source.

With the ontology based SLA management approach proposed in [25], it was intended to improve the SLA by taking into account the semantic meaning of SLA concepts and contextual information from the consumers of cloud services. The approach sought to dynamically adapt cloud services and resources to different variations of the consumer context and to meet their requirements using the benefits of ontology representation and inference. Therefore, reliable Quality of Service and compatibility with SLA parameters have been attempted to be provided.

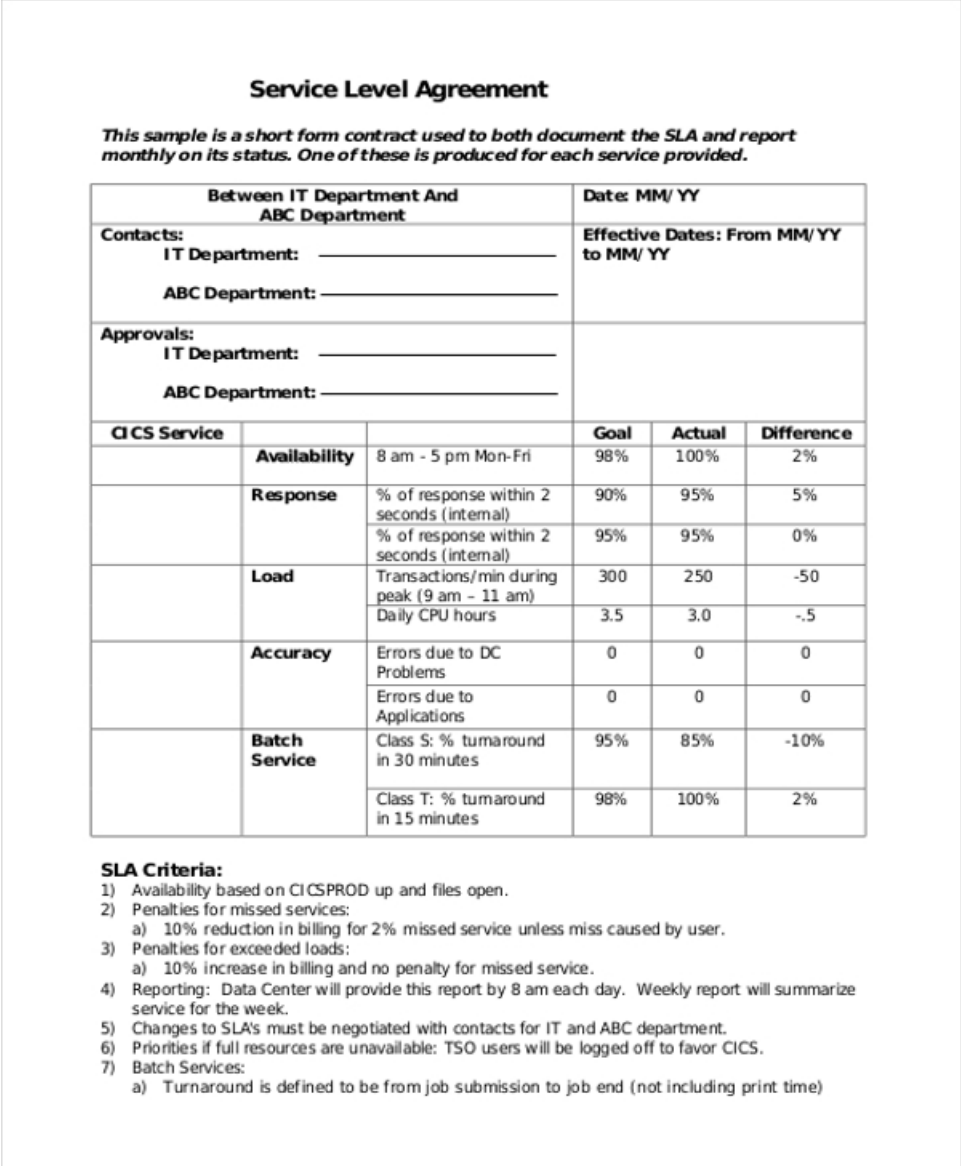
CHAPTER 4

GENERIC SLA ONTOLOGY

In this chapter, the proposed Service Level Agreement (SLA) ontology is explained in detail. First, an overview of SLA field is mentioned. Then, the concepts and the relations between these concepts defined in SLA ontology are given. Finally, the construction of SLA ontology is presented.

4.1 An Overview of SLA Field

It is known that the main concepts of SLA field are SLA, Service Level Specification (SLS), and Service Level Objective (SLO). SLA is a formal agreement between a service provider and a customer that includes a service provider's commitment to the customer. SLS is a frame definition that contains all of the metrics, thresholds, and calculation formulas required for SLA metrics. SLO is a definition that contains metric and threshold. Metrics can be defined as KQI, which is meaningful to the customer, or KPI, which is meaningful low-level metric type for the service provider. Threshold, on the other hand, is the expected target value for metric measurements to reach. SLAs can contain many criteria. The example shown in Figure 4.1 contains various criteria.



Service Level Agreement

This sample is a short form contract used to both document the SLA and report monthly on its status. One of these is produced for each service provided.

Between IT Department And ABC Department			Date: MM/YY		
Contacts: IT Department: _____ ABC Department: _____			Effective Dates: From MM/YY to MM/YY		
Approvals: IT Department: _____ ABC Department: _____					
CICS Service			**Goal**	**Actual**	**Difference**
	Availability	8 am - 5 pm Mon-Fri	98%	100%	2%
	Response	% of response within 2 seconds (internal)	90%	95%	5%
		% of response within 2 seconds (internal)	95%	95%	0%
	Load	Transactions/min during peak (9 am – 11 am)	300	250	-50
		Daily CPU hours	3.5	3.0	-.5
	Accuracy	Errors due to DC Problems	0	0	0
		Errors due to Applications	0	0	0
	Batch Service	Class S: % turnaround in 30 minutes	95%	85%	-10%
		Class T: % turnaround in 15 minutes	98%	100%	2%

SLA Criteria:

1) Availability based on CICSPROD up and files open.
2) Penalties for missed services:
   a) 10% reduction in billing for 2% missed service unless miss caused by user.
3) Penalties for exceeded loads:
   a) 10% increase in billing and no penalty for missed service.
4) Reporting: Data Center will provide this report by 8 am each day. Weekly report will summarize service for the week.
5) Changes to SLA's must be negotiated with contacts for IT and ABC department.
6) Priorities if full resources are unavailable: TSO users will be logged off to favor CICS.
7) Batch Services:
   a) Turnaround is defined to be from job submission to job end (not including print time)

Figure 4.1: Service Level Agreement Example. (taken from [2])

4.2 Concepts Defined in SLA Ontology

The literature survey we have conducted revealed that there are other common concepts related to SLA, SLS, and SLO. All the classes defined in the proposed SLA ontology are described below.

- The central concept in SLA ontology is *SLA* class. SLA class is associated with the Product, Party, and SLS classes.

- *Product* specifies information about the product.
- *Party* specifies parties in SLAs. It is associated with PartyType.
- *PartyType* determines the type of party. There are party types such as supplier, customer, and operational unit.
- *SLS* is one of the most important classes. It is associated with SLSType and ServiceType.
- *SLSType* specifies the type of SLS. There are SLS types such as Operational Level Agreement (OLA), Customer, and Supplier.
- *ServiceType* defines the type of service. The Public Switched Telephone Network (PSTN) is one of the common service types.
- *SLO* is another important class. It is associated with SLS, Metric, Threshold and Penalty classes.
- *Metric* is associated with MetricType class.
- *MetricType* specifies the metric type. There are metric types such as KPI, KQI, and DPI.
- *MetricMeasure* is associated with Metric class.
- *Threshold* is associated with Criteria and Comparator classes.
- *Comparator* specifies the comparator type of Threshold. There are comparator types such as greater and equal, lower and equal, lower, greater, and equal.
- *Criteria* associates with CriteriaType class.
- *CriteriaType* determines the criteria type. There are criteria types such as region, type of business, and access type.
- *Penalty* defines the penalty that should be applied in case of a violation. Penalties may be imposed such as compensation for lost earnings, lost fees, repayment of fees, termination, and combinations thereof.
- *Breached* indicates whether or not a violation occurs.

4.3 Relationships Defined in SLA Ontology

The literature survey showed that some classes are related to each other. The information obtained about these relations are listed as follows.

- An SLA is associated with exactly one Product.
- An SLA must have at least one SLS.
- An SLA must have a Party.
- Each SLS must have at least one SLO.
- A Metric belongs to at least one SLO.
- A Threshold can have more than one Criteria.
- A Comparator may belong to more than one Threshold.
- An SLS must have a SLSType.
- An SLS must have a ServiceType.
- A Criteria must have a CriteriaType.
- A Party must have a Criteria Type.
- A SLO must have a Penalty.
- A MetricMeasure must have a Metric.

Relationships between classes are defined using object and data properties.

Object properties establish a relationship between two individuals. They link individuals from a domain to a range. Many of the object properties defined here are based on *has-a* and *part-of* relations. In addition to these, there are object properties that we define. All object properties defined in the proposed SLA ontology are listed below.

- *associateWithProduct* defines the association between SLA and Product classes.
- *isSLAtoSLS* is the part-of relationship from SLA class to SLS class.

- *isSLAtoParty* is the part-of relationship from SLA class to Party class.
- *isSLOtoSLS* is the part-of relationship from SLO class to SLS class.
- *hasMetric* is the has-a relationship from SLO class to Metric class.
- *hasThreshold* is the has-a relationship from SLO class to Threshold class.
- *hasComparator* is the has-a relationship from Threshold class to Comparator class.
- *hasCriteria* is the has-a relationship from Threshold class to Criteria class.
- *hasSLSType* is the has-a relationship from SLS class to SLSType class.
- *hasServiceType* is the has-a relationship from SLS class to ServiceType class.
- *hasMetricType* is the has-a relationship from Metric class to MetricType class.
- *hasCriteriaType* is the has-a relationship from Criteria class to CriteriaType class.
- *hasPartyType* is the has-a relationship from Party class to PartyType class.
- *isMeasureOf* defines a relationship between MetricMeasure and Metric classes.
- *hasPenalty* is the has-a relationship from SLO class to Penalty class.

Data properties define relationships between individuals and data values. Data properties may vary in different areas. Therefore, the data properties described here should be considered as samples. The data properties defined in the proposed SLA ontology are listed below.

- *value_as_float* belongs to Threshold class. It represents the floating value of the thresholds.
- *value* belongs to Criteria class. Criteria values are string values.
- *name* belongs to Comparator, MetricType, CriteriaType, SLSType, Metric, Breached, and ServiceType classes. It represents name information for these classes. It can be any string value.

- *firstname* belongs to Party class. It defines the first name that the party has. It can be any string value.
- *surname* belongs to Party class. It defines the surname that the party has. It can be any string value.
- *title* belongs to Party class. It defines the title that the party has. Titles can be any string value.
- *serial_number* belongs to Product class. It determines the serial number of the Product. Product serial numbers are string values.
- *code* belongs to Metric class. It specifies the code of the Metric. Metric codes are string values such as "ORDER_COMPLETION_TIME_KQI", "FAULT_COMPLETION_TIME_KQI", and "SERVICE_AVAILABILITY".
- *desc* belongs to Metric class. It includes the description of the Metric. Metric descriptions are string values such as "Order Completion Time KQI", "Fault Completion Time KQI", and "Service Availability".

4.4 The Construction of SLA Ontology

SLA ontology is created using Protégé tool according to the classes, object properties and data properties we have defined. SLA ontology was constructed using Protégé tool and exported in OWL format. In Figure 4.3, the constructed ontology is visualized with OntoGraf, a component of Protégé. Here only the directions of the object properties between classes and classes are shown. In Figure A.1 (see Appendix A), object and data properties are also specified along with the classes.

Since there are no *is-a* relationships among the concepts, all the classes defined in the ontology are at the same level in the class hierarchy. The class hierarchy created in Protégé is shown in Figure 4.2.

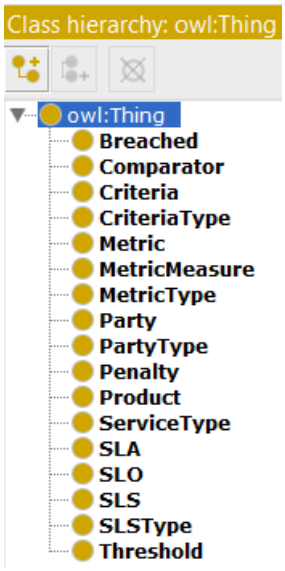


Figure 4.2: Class Hierarchy.

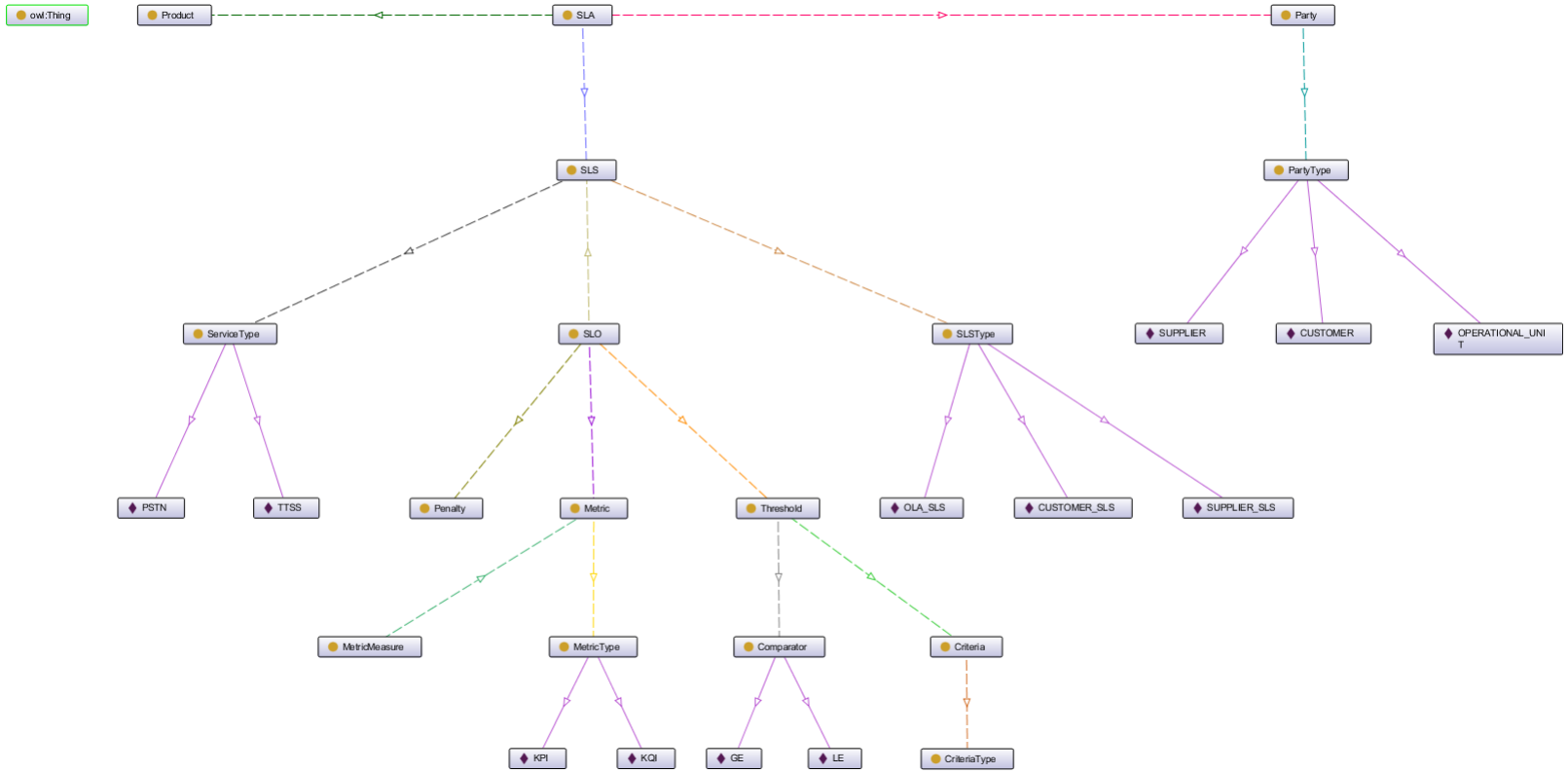


Figure 4.3: The Constructed SLA Ontology.

CHAPTER 5

SLA VIOLATION DETECTION SYSTEM

In this chapter, firstly, it is explained how to store SLA information in triple stores. Later, the system of SLA violation detection that we give the name SLAVIDES is introduced and the simulation and evaluation of the system have been realized. Finally, constraint checking and rule extraction on SLA data is demonstrated.

5.1 SLA Violation Detection System

5.1.1 Storing SLA Information in Triple Stores

SLA information should be stored as RDF triples so that semantic querying, inferencing or constraint checking can be performed. RDF triples can be stored in triple stores specially designed to store them. Most of the triple stores can usually be queried with SPARQL.

With graph databases, data can be recorded as linked, but these databases do not have a standard query language. Thus, RDF triple store is preferred when recording data.

TDB is a component of Apache Jena and is used for RDF storage and querying. It supports all of the Jena APIs. TDB can be used as a high-performance RDF storage in a single machine. The triple data employed in this study are stored in TDB.

Recording SLA information in TDB is performed in three phases. The first phase is to record the metric data. Then SLS data is recorded. Finally, SLA data is included in TDB.

Table 5.1: Incoming Message for Metric Insertion

604;KQI;Order Completion Time KQI;ORDER_COMPLETION_TIME_KQI; Order Completion Time KQI, 11080;KQI;Fault Completion Time KQI;FAULT_COMPLETION_TIME_KQI; Fault Completion Time KQI, 12050;KQI;Service Availability;SERVICE_AVAILABILITY; Service Availability.

Table 5.2: Triple Representation of Metric Values

Subject	**Predicate**	**Object**
odtuMetric:604	odtu:type	"KQI"
odtuMetric:604	odtu:name	"Order Completion Time KQI"
odtuMetric:604	odtu:code	"ORDER_COMPLETION_TIME_KQI"
odtuMetric:604	odtu:desc	"Order Completion Time KQI"
odtuMetric:11080	odtu:type	"KQI"
odtuMetric:11080	odtu:name	"Fault Completion Time KQI"
odtuMetric:11080	odtu:code	"FAULT_COMPLETION_TIME_KQI"
odtuMetric:11080	odtu:desc	"Fault Completion Time KQI"
odtuMetric:11080	odtu:type	"KQI"
odtuMetric:11080	odtu:name	"Service Availability"
odtuMetric:11080	odtu:code	"SERVICE_AVAILABILITY"
odtuMetric:11080	odtu:desc	"Service Availability"

There is no object property that connects to a class other than Metric and MetricType classes. For this reason adding, updating or removing metrics will not cause major problems as it will not change the whole structure. In this phase, only information about metrics is included in TDB. The message shown in the Table 5.1 contains information about three different metrics. The incoming message is not in the RDF format. Therefore, it is necessary to parse the message, convert it into triple format and save it in TDB.

The data up to the "," character are given for the first metric. All metric values are separated by "," characters. All the information about a metric is separated by ";" characters. Taking these special characters into consideration, this message is parsed and the records are saved as Subject, Predicate, Object, as in Table 5.2.

When adding SLA data, data related to Product and Party are added besides SLA. Leaving some fields empty does not cause a problem. A sample message is shown in Table 5.3 to add three SLAs. Party, Product and SLA information are separated by

Table 5.3: Incoming Message for SLA Insertion

```
1250;Metin;Takak; ;CUSTOMER,
1257; ; ;Innova I.T Solutions;CUSTOMER,
1577;Alper;Karamanlioglu; ;CUSTOMER,
||
3250;2123122222,
3300;3123122224,
3301;4121122012,
&&
3801;3100;1250;3250,
3805;3100;1577;3300,
3903;6400;1257;3301.
```

special characters. It will then be parsed and recorded as RDF triples.

The most challenging task when recording SLA information is to store SLS values. The information for all classes except Metric, SLA, Product and Party are considered as SLS-linked. The reason for this is that the SLS has all the details of the service part of SLA. Deleting or updating SLS data may cause many problems. For this reason, these operations should be carried out considering the classes to which they are connected.

5.1.2 Architecture of the Developed Expert System

SHACL rules and constraints, SLA ontology and SLA data are stored in TDB. The incoming service message is parsed and queries are generated to monitor and infer. The query response is sent as a JSON message. The rule engine is applied when necessary. The system has components that fulfill these tasks. The system architecture in which these components are shown in Figure 5.1.

5.1.3 SLA Violation Monitoring

The information describing the product to be detected whether SLA violation has occurred is sent to SLA Violation Detection System (SLAVIDES) as service messages. In the content of these messages sent to the system, the incoming message ID, the product ID, the measurement values of the metrics and the information of the crite-

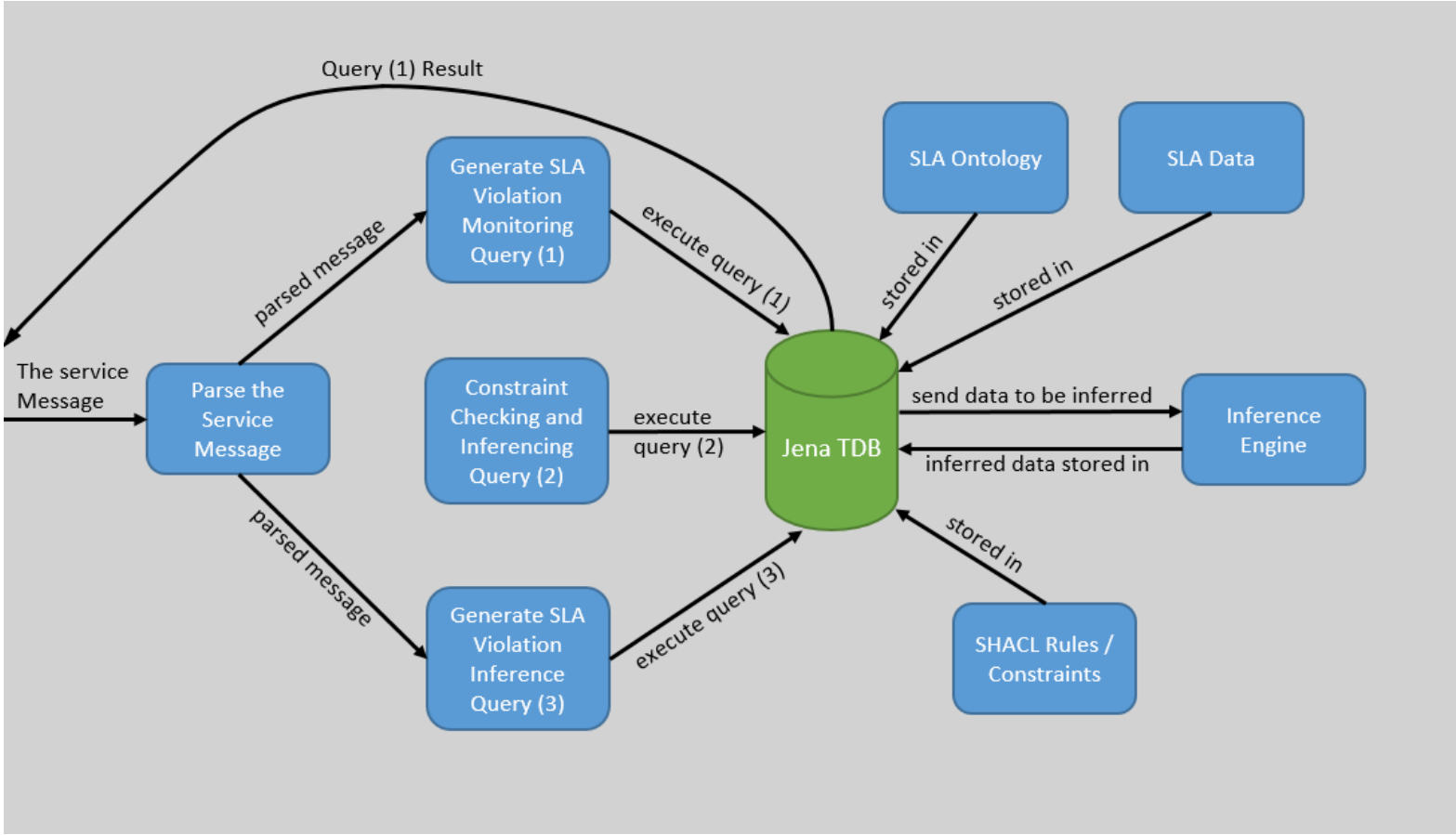


Figure 5.1: The Architecture of the Developed Expert System.

Table 5.4: Sample Service Message

```
{
MESSAGE_ID: 1114
PRODUCT_ID : 3250
MEASUREMENT {
METRIC_ID : 604 MEASURE : 3
METRIC_ID : 11080 MEASURE : 3
}
REGION : 1.Region
TYPE_OF_BUSINESS : Activation
ACCESS_TYPE : ME
}
```

ria are included. SLAVIDES obtains this information and ensures that a SPARQL query is generated to detect the violation. The system executes the generated query on the triple data stored in TDB. The contents of a sample service message are shown in Table 5.4. The SPARQL query generated for this message is as shown in Figure 5.2. After executing this query, the query result in JSON format is shown in Figure 5.5. The query result is returned as a service message along with SLA violation information and other required information.

Each service message can contain multiple metric measurements. Therefore, it is possible that more than one violation may occur for each message. SLAVIDES can process a single message as well as have the ability to process multiple messages. Initially, incoming service message parsed and the list of messages is identified. The results are then combined by performing separate queries for each message.

In addition, many criteria can be defined within each message. The violation is detected by the query generated by considering these criteria and metrics. The measured values and threshold values are compared according to the comparator and it is determined whether or not the violation occurs.

The SPARQL query that monitors the SLA Violation is produced as follows.

- The prefixes to be used in the query are determined. Many of the classes defined in the SLA ontology are used when constructing queries. So, each of these classes must be defined as a prefix when starting to construct the query. Thus,

```
PREFIX odtu: <http://www.semanticweb.org/odtu#>
PREFIX odtuSLA: <http://www.semanticweb.org/odtu#SLA:>
PREFIX odtuSLS: <http://www.semanticweb.org/odtu#SLS:>
PREFIX odtuSLO: <http://www.semanticweb.org/odtu#SLO:>
PREFIX odtuProduct: <http://www.semanticweb.org/odtu#Product:>
PREFIX odtuMetric: <http://www.semanticweb.org/odtu#Metric:>
PREFIX odtuParty: <http://www.semanticweb.org/odtu#Party:>
PREFIX odtuMessage: <http://www.semanticweb.org/odtu#Message:>
PREFIX odtuServiceType: <http://www.semanticweb.org/odtu#ServiceType:>
PREFIX odtuThreshold: <http://www.semanticweb.org/odtu#Threshold:>
PREFIX odtuComparator: <http://www.semanticweb.org/odtu#Comparator:>
PREFIX odtuCriteria: <http://www.semanticweb.org/odtu#Criteria:>
PREFIX odtuMetricType: <http://www.semanticweb.org/odtu#MetricType:>
PREFIX odtuCriteriaType: <http://www.semanticweb.org/odtu#CriteriaType:>
PREFIX odtuSLSType: <http://www.semanticweb.org/odtu#SLSType:>
PREFIX odtuBreached: <http://www.semanticweb.org/odtu#Breached:>
SELECT DISTINCT ?MID ?KPI ?SERIAL_NUMBER ?FIRSTNAME ?SURNAME ?MEASURE ?THRESHOLD ?COMPARATOR ?BREACHED WHERE {{
odtuMessage:1111 odtu:value ?MID.
?SLA  odtu:associateWithProduct odtuProduct:3250.
odtuProduct:3250  odtu:serial_number ?SERIAL_NUMBER.
?SLA odtu:isSLAofParty ?Party.
?Party odtu:firstname ?FIRSTNAME.
?Party odtu:surname ?SURNAME.
?SLA  odtu:isSLAofSLS ?SLS.
?SLO  odtu:isSLOofSLS ?SLS.
?SLO  odtu:hasMetric ?M.
?M odtu:name ?KPI.      VALUES (?M){ (odtuMetric:604)  (odtuMetric:11080) }
?SLO  odtu:hasThreshold ?ThresholdList.?CR0 odtu:value "1.Region".
?ThresholdList odtu:hasCriteria ?CR0.
?CR1 odtu:value "Activation".
?ThresholdList odtu:hasCriteria ?CR1.
?CR2 odtu:value "ME".
?ThresholdList odtu:hasCriteria ?CR2.
{ SELECT ?ThresholdList (COUNT(distinct ?X) as ?count) WHERE { ?ThresholdList odtu:hasCriteria ?X. }
  GROUP BY ?ThresholdList HAVING (COUNT(distinct ?X) =3)}
?ThresholdList odtu:value_as_float ?THRESHOLD.
odtuMessage:1111 odtu:hasAmetric ?M.
?M odtu:metricmeasure1111 ?MEASURE.
?ThresholdList odtu:hasComparator ?COMP.
?COMP odtu:name ?COMPARATOR.
OPTIONAL {odtuBreached:Plus odtu:name ?BREACHED FILTER ((?THRESHOLD <= ?MEASURE && ?COMPARATOR = "GE")||
        (?THRESHOLD >= ?MEASURE && ?COMPARATOR = "LE"))}.
OPTIONAL {odtuBreached:Minus odtu:name ?BREACHED FILTER ((?THRESHOLD > ?MEASURE && ?COMPARATOR = "GE")||
        (?THRESHOLD < ?MEASURE && ?COMPARATOR = "LE"))}.      } }
```

Figure 5.2: SLA Violation Monitoring Query.

Table 5.5: Returned SLA Violation Message


```
{
"head": {
"vars": [ "MID" , "KPI" , "SERIAL_NUMBER" , "FIRSTNAME" ,
"SURNAME" , "MEASURE" , "THRESHOLD" , "COMPARATOR" ,
"BREACHED" ]
} ,
"results": {
"bindings": [
{
"MID": { "type": "literal" ,
"datatype": "http://www.w3.org/2001/XMLSchema#integer" , "value": "1111" } ,
"KPI": { "type": "literal" , "value": "Order Completion Time KQI" } ,
"SERIAL_NUMBER": { "type": "literal" , "value": "2123122222" } ,
"FIRSTNAME": { "type": "literal" , "value": "Metin" } ,
"SURNAME": { "type": "literal" , "value": "Takak" } ,
"MEASURE": { "type": "literal" ,
"datatype": "http://www.w3.org/2001/XMLSchema#float" , "value": "3.0" } ,
"THRESHOLD": { "type": "literal" ,
"datatype": "http://www.w3.org/2001/XMLSchema#float" , "value": "5.0" } ,
"COMPARATOR": { "type": "literal" , "value": "LE" } ,
"BREACHED": { "type": "literal" , "value": "+" }
}
]
}
}
```

it is possible to use abbreviated forms of URIs.

- The SELECT command defined in SPARQL is used to monitor whether an SLA violation has occurred. Parameters to be sent back as service messages should be defined here.
- Multiple metrics are included in the query using the VALUES command. The number of metrics is not specific and can be any value.
- Criteria are produced dynamically according to the incoming message. The number or order of the criteria in the message may be different.
- By using the OPTIONAL / FILTER commands, it is possible to compare measured values by considering the comparison type.

5.1.4 SLA Violation Inference

The SLA violation inference query differs from the monitoring query in that the CONSTRUCT structure is used instead of the SELECT structure. Here, the CONSTRUCT structure is used to establish the *odtu:violated* relationship between SLA and Breached classes. Thus, when other conditions are met, new information is extracted. There is no need to use some classes' prefixes as there is no visualization here. SLA Violation Inference Query is shown in Figure 5.3.

5.2 Simulation of the Proposed System

As an example, a step-by-step simulation of how the system should work is shown. At each step, information about the SLA field is also given.

1) Each SLA must be associated with a Product. As shown in Figure 5.4, SLA number 3801, which is associated with Product number 3250 specified in the service message, should be selected.

```
PREFIX odtu: <http://www.semanticweb.org/odtu#>
PREFIX odtuSLA: <http://www.semanticweb.org/odtu#SLA:>
PREFIX odtuSLS: <http://www.semanticweb.org/odtu#SLS:>
PREFIX odtuSLO: <http://www.semanticweb.org/odtu#SLO:>
PREFIX odtuProduct: <http://www.semanticweb.org/odtu#Product:>
PREFIX odtuMetric: <http://www.semanticweb.org/odtu#Metric:>
PREFIX odtuMessage: <http://www.semanticweb.org/odtu#Message:>
PREFIX odtuThreshold: <http://www.semanticweb.org/odtu#Threshold:>
PREFIX odtuComparator: <http://www.semanticweb.org/odtu#Comparator:>
PREFIX odtuCriteria: <http://www.semanticweb.org/odtu#Criteria:>
PREFIX odtuBreached: <http://www.semanticweb.org/odtu#Breached:>
CONSTRUCT {?SLA odtu:violates ?BREACHED.}WHERE {{
odtuMessage:1111 odtu:value ?MID.
?SLA  odtu:associateWithProduct odtuProduct:3250.
?SLA  odtu:isSLAofSLS ?SLS.
?SLO  odtu:isSLOofSLS ?SLS.
?SLO  odtu:hasMetric ?M.
?M odtu:name ?KPI.
VALUES (?M){ (odtuMetric:604)  (odtuMetric:11080) }
?SLO  odtu:hasThreshold ?ThresholdList.
?CR0 odtu:value "1.Region".
?ThresholdList odtu:hasCriteria ?CR0.
?CR1 odtu:value "Activation".
?ThresholdList odtu:hasCriteria ?CR1.
?CR2 odtu:value "ME".
?ThresholdList odtu:hasCriteria ?CR2.
{ SELECT ?ThresholdList (COUNT(distinct ?X) as ?count)
  WHERE { ?ThresholdList odtu:hasCriteria ?X. }
  GROUP BY ?ThresholdList HAVING (COUNT(distinct ?X) =3)}
?ThresholdList odtu:value_as_float ?THRESHOLD.
odtuMessage:1111 odtu:hasAmetric ?M.
?M odtu:metricmeasure1111 ?MEASURE.
?ThresholdList odtu:hasComparator ?COMP.
?COMP odtu:name ?COMPARATOR.
OPTIONAL {odtuBreached:Plus odtu:name ?BREACHED
FILTER ((?THRESHOLD <= ?MEASURE && ?COMPARATOR = "GE")||
(?THRESHOLD >= ?MEASURE && ?COMPARATOR = "LE"))}.
OPTIONAL {odtuBreached:Minus odtu:name ?BREACHED
FILTER ((?THRESHOLD > ?MEASURE && ?COMPARATOR = "GE")||
(?THRESHOLD < ?MEASURE && ?COMPARATOR = "LE"))}.      } }
```

Figure 5.3: SLA Violation Inference Query.

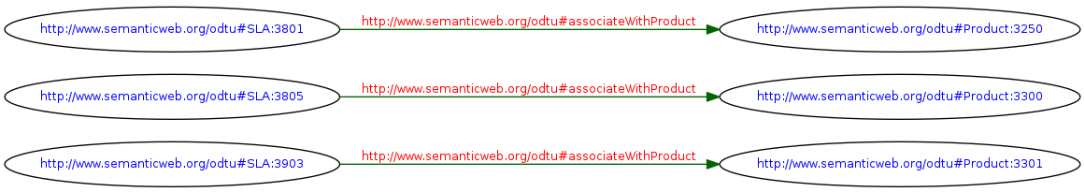


Figure 5.4: SLA Transition from Product.

2) An SLS can belong to more than one SLA. Figure 5.5 shows that SLS number 3100 belongs to SLAs with 3801 and 3805 numbers. SLS number 3100 should be selected because SLS of the SLA number 3801 selected in Step 1 is requested to be added.

Figure 5.5: SLS Transition from SLA.

3) Each SLS can have more than one SLO. Figure 5.6 shows that SLS 6400 has SLOs with 6150 and 10100 numbers, SLS number 3100 has SLOs with 6050, 9050 and 10050 numbers. SLOs numbered 6050, 9050, and 10050 should be selected as the SLOs of SLS number 3100 selected in Step 2 are requested to be selected.

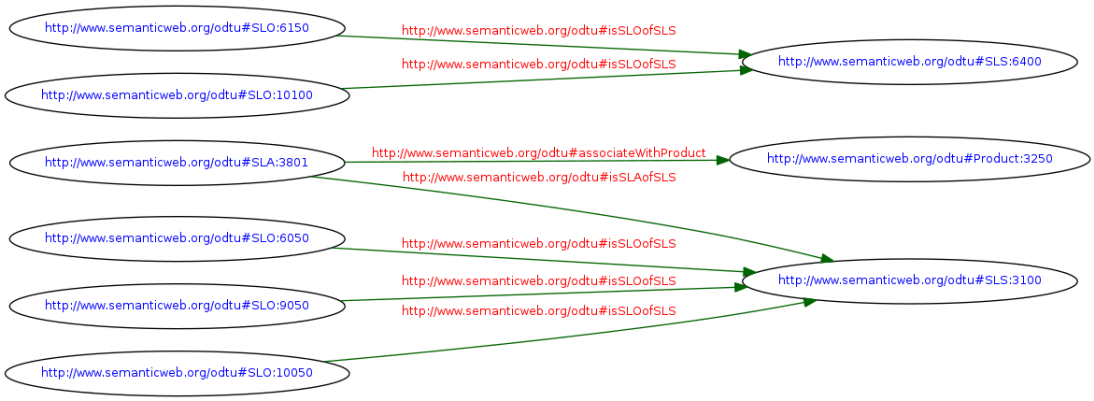


Figure 5.6: SLO Transition from SLS.

4) A metric can belong to more than one SLO. Figure 5.7 shows that Metric number 604 belongs to SLOs with numbers 6050, 6150 and 6200.It is understood from the

sample service message that the selection of SLOs of metrics with 604 and 11080 numbers is requested. Therefore, SLO number 9050 which relates to metric number 11080 and SLO number 6050 which relates to metric number 604 from SLOs 6050, 9050, 10050 selected in Step 3, should be selected.

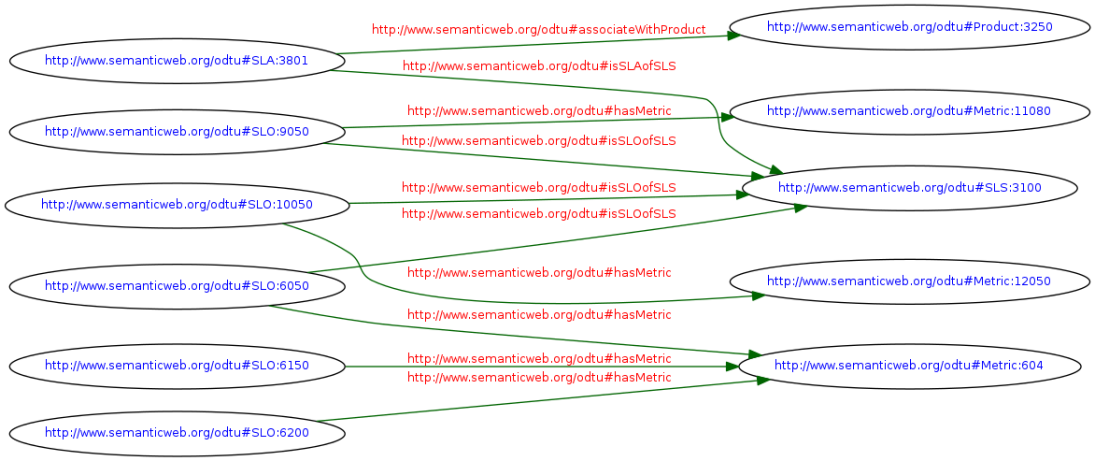


Figure 5.7: Metric Transition from SLO.

5) Each SLO must have a Threshold. Figure 5.9 shows that SLOs with 6050, 9050, and 10050 numbers have Thresholds with 5150, 8050, and 9050 numbers, respectively. SLOs with 6050 and 9050 numbers selected in Step 4 have Thresholds with 5150 and 8050 numbers.

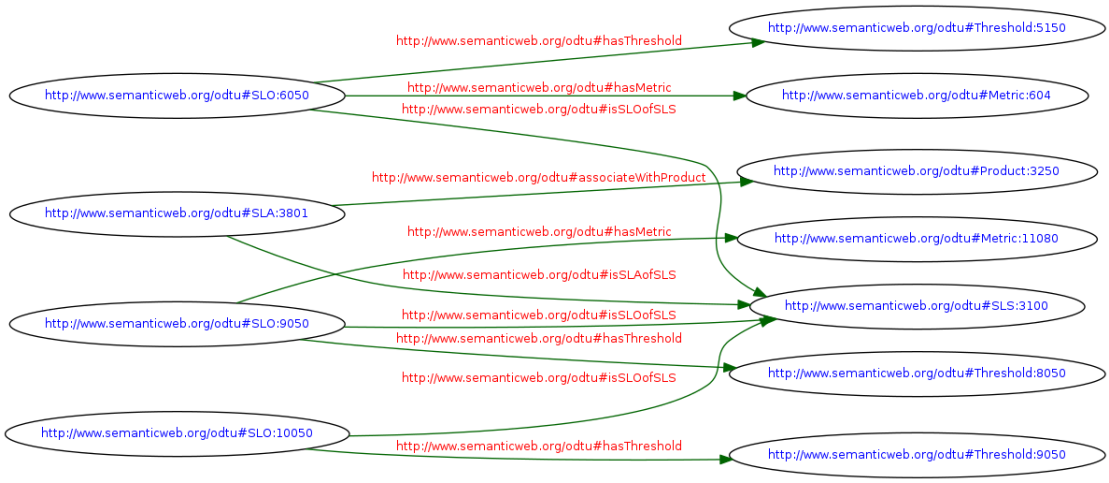


Figure 5.8: Threshold Transition from SLO.

6) Each Threshold must have a Comparator. Figure 5.9 shows that Threshold number 5150 has the LE Comparator and Threshold number 8050 has the GE Comparator.

Then choose the appropriate values according to the criteria and compare these values with the values in the message.

5.3 Evaluation of the Proposed System

The SLA Violation Detection System, developed using the proposed ontology, has been tested on 1036 actual triple data from SLAs in the field of telecommunication. Accordingly, the success of the system was measured by considering 75 different scenarios with different inputs. A group of three people determined that 62 of these scenarios did not constitute violations and 13 of them had one or more violations. It is stated that there are more than one violation in 3 of the violation cases. The developed system achieves the same results as the results of the group in all of the mentioned scenarios.

Other than that, we also wanted to measure the success of the system in a larger dataset, so we produced synthetic data. Synthetic data set consisting of 10000 triple data we produced was recorded in TDB. Again, 75 different service messages have been created. While creating these messages, we did not have the goal of keeping the percentage of actual violations occurring. The same group of people investigated whether these messages constituted violations. Nearly half of the messages have been constituted violations. More than one violation occurred in about half of the messages that have been constituted violations. Thereafter, the violation detection queries were executed on the synthetic data set. It has been determined that much of the violations found by the group of people and the violations that the system identifies are the same. Differences were again reviewed. It has been seen that the system correctly detected the violations, and the differences were caused by human error.

5.4 Constraint Checking and Rule Inference

Constraint checking and rule inference are frequently used features of ontology-based systems. We noticed that there are no common constraints and inference rules in all areas where SLAs are used. Every area has its own set of rules and constraints.

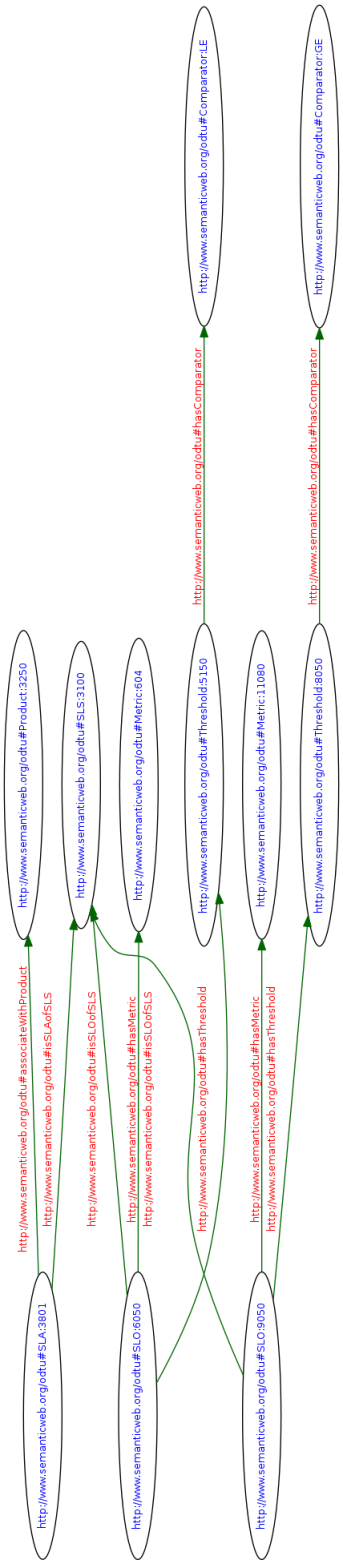


Figure 5.9: Comparator Transition from Threshold.

Therefore, we have shown how sample constraints and inference rules can be defined. To achieve this, Shapes Constraint Language (SHACL), a new technology approved by the W3C, is used.

We employed SHACL's open-source and Jena-based implementation, TopBraid SHACL API, to perform constraint checking and rule inference. TopBraid SHACL API has been developed fully in accordance with SHACL specifications.

Constraints and rules can be easily defined in TopBraid Composer. We then convert it to SHACL format via TopBraid SHACL API. Therefore, they can be stored as RDF triples in TDB.

There are many different constraints and rules in different SLA areas. For example, frequent violations of SLAs in the same product may be due to incorrectly defined KQI values. Such examples may vary for each area and even for each company in the same area.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this chapter, this thesis study is concluded and future studies are mentioned.

6.1 Conclusion

In this thesis, an expert system developed to detect the violations of Service Level Agreement is introduced. This system, called SLAVIDES, has been developed ontology based and therefore has the advantages of using ontologies. The most important of these benefits is to enable reuse of domain knowledge. SLAVIDES also has the ability to perform semantic queries, infer new information from existing information, and check constraints.

SLAVIDES processes the incoming service message and generates SPARQL queries for this message to detect the SLA violation. One of these queries is to create a new relationship between SLA and Breach classes. The other is to monitor the SLA violation in the desired format. The generated queries are executed on SLA data stored as triples in Jena TDB. The violation monitoring query response is returned as a service message, and the violation inference query result is recorded in TDB. Besides, constraint checking and inferencing are performed according to SHACL rules and constraints. These rules and constraints are also recorded in TDB in RDF format.

While the proposed ontology is being designed, concepts that are common to the SLAs in various areas are just included in the ontology. By establishing the concepts and the relationships between them, an ontological structure was created for the SLA

field to be understood by even non-specialists.

The developed expert system provides significant contributions in accurately detecting SLA violations. Automatic detection of violations ensures rapid detection of violations, reduces the error rate, and prevents loss of work power. This increases the quality of service and enterprise.

Publishing SLA ontology in formats that are widely used in the Semantic Web increases reusability. In addition, the flexibility of the proposed ontology leads to the possibility of direct use in many areas. Besides, in some areas new ontologies are created by extending SLA ontology when necessary. By the expandability of SLA Ontology, it is possible to create new ontologies based on this ontology.

6.2 Future Work

Since the proposed ontology is generically designed, it can be used satisfactorily in many different areas, but there may be different needs and expectations in Service Level Agreements in some domains. Therefore, it will be useful to test the validity of the ontology and the developed SLA violation detection system by applying it to areas outside the telecommunications sector.

Also, constraint checking and rule inference have been realized assuming that each area has different constraints and inference rules. In the future, the capabilities of the system will be improved if common constraints and inference rules are identified in different areas.

SLA data are stored in Jena TDB as RDF triples and the violation is detected in these data. A performance analysis comparing the same operations in different triple stores will be conducted in another study.

Apart from these, another system is planned to be developed using machine learning techniques in the future to automatically predict SLA violations. The system to be developed will contribute to the prevention of SLA violations and improve the service quality. Furthermore, the system will perform trend analysis taking into account the distribution of SLA data. The system is planned to be developed as Java-based to

facilitate integration into SLAVIDES.

REFERENCES

[1] From spin to shacl. http://spinrdf.org/spin-shacl.html. Accessed: 2017-09-01.

[2] Sample Service Level Agreement service level agreement samples. https://www.sampletemplates.com/business-templates/sample-service-level-agreement.html. Accessed: 2017-09-01.

[3] Topbraid composer. http://www.topbraidcomposer.com/. Accessed: 2017-09-01.

[4] TopQuadrant, Inc spin (sparql inferencing notation). https://www.topquadrant.com/technology/sparql-rules-spin. Accessed: 2017-09-01.

[5] Wikipedia semantic web. https://en.wikipedia.org/wiki/Semantic_Web. Accessed: 2017-09-01.

[6] ZDNet from semantic web (3.0) to the webos (4.0). http://www.zdnet.com/article/from-semantic-web-3-0-to-the-webos-4-0. Accessed: 2017-09-01.

[7] Gregory Bain, Jay Dia, et al. Sla management handbook-volume 4: Enterprise perspective. In *TM Forum*, 2004.

[8] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[9] Dan Brickley and Ramanathan V Guha. Resource description framework (rdf) schema specification 1.0: W3c candidate recommendation 27 march 2000. 2000.

[10] Amir Vahid Dastjerdi, Sayed Gholam Hassan Tabatabaei, and Rajkumar Buyya. A dependency-aware ontology-based approach for deploying service level agreement monitoring services in cloud. *Software: Practice and Experience*, 42(4):501–518, 2012.

[11] Mike Dean, Guus Schreiber, Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L McGuinness, Peter F Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language reference. *W3C Recommendation February*, 10, 2004.

[12] Jianguo Ding. *Advances in network management*. CRC press, 2016.

[13] Glen Dobson, Russell Lock, and Ian Sommerville. Qosont: a qos ontology for service-centric systems. In *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*, pages 80–87. IEEE, 2005.

[14] Glen Dobson and Alfonso Sanchez-Macian. Towards unified qos/sla ontologies. In *Services Computing Workshops, 2006. SCW'06. IEEE*, pages 169–174. IEEE, 2006.

[15] Kaouthar Fakhfakh, Tarak Chaari, and Mohamed Jmaiel. Semantic enabled framework for sla monitoring. 2009.

[16] Kaouthar Fakhfakh, Tarak Chaari, Saïd Tazi, Khalil Drira, and Mohamed Jmaiel. A comprehensive ontology-based approach for sla obligations monitoring. In *Advanced Engineering Computing and Applications in Sciences, 2008. ADVCOMP'08. The Second International Conference on*, pages 217–222. IEEE, 2008.

[17] Les Green. Service level agreements: an ontological approach. In *Proceedings of the 8th international conference on Electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet*, pages 185–194. ACM, 2006.

[18] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[19] Kahina Hamadache and Stamatia Rizou. Holistic sla ontology for cloud service evaluation. In *Advanced Cloud and Big Data (CBD), 2013 International Conference on*, pages 32–39. IEEE, 2013.

[20] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21:79, 2004.

[21] Apache Jena. Reasoners and rule engines: Jena inference support. *The Apache Software Foundation*, 2013.

[22] Apache Jena. A free and open source java framework for building semantic web and linked data applications. *Available online: jena. apache. org/(accessed on 28 April 2015)*, 2015.

[23] H Knublauch, JA Hendler, and K Idehen. Spin sparql inferencing notation. *Retrieved Feb*, 8:2013, 2009.

[24] Holger Knublauch and Arthur Ryman. Shapes constraint language (shacl). *Working Draft (work in progress), W3C*, 2016.

[25] Taher Labidi, Achraf Mtibaa, and Faiez Gargouri. Ontology-based context-aware sla management for cloud computing. In *International Conference on Model and Data Engineering*, pages 193–208. Springer, 2014.

[26] Ora Lassila and Ralph R Swick. Resource description framework (rdf) model and syntax specification. 1999.

[27] Priscilla S Moraes, Leobino N Sampaio, José AS Monteiro, and Marcos Portnoi. Mononto: a domain ontology for network monitoring and recommendation for advanced internet applications users. In *Network Operations and Management Symposium Workshops, 2008. NOMS Workshops 2008. IEEE*, pages 116–123. IEEE, 2008.

[28] Mark A. Musen. The protÉgÉ project: A look back and a look forward. *AI Matters*, 1(4):4–12, 2015.

[29] Adrian Paschke. Rbsla a declarative rule-based service level agreement language based on ruleml. In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, volume 2, pages 308–314. IEEE, 2005.

[30] Adrian Paschke and Elisabeth Schnappinger-Gerull. A categorization scheme for sla metrics. *Service Oriented Electronic Commerce*, 80(25-40):14, 2006.

[31] Eric Prud, Andy Seaborne, et al. Sparql query language for rdf. 2006.

[32] Sin-seok Seo, Arum Kwon, Joon-Myung Kang, and James Won-Ki Hong. Oslam: towards ontology-based sla management for iptv services. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 1228–1234. IEEE, 2011.

[33] Swapna Singh and Ragini Karwayun. A comparative study of inference engines. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 53–57. IEEE, 2010.

[34] Philipp Wieder, Joe M Butler, Wolfgang Theilmann, and Ramin Yahyapour. *Service level agreements for cloud computing*. Springer Science & Business Media, 2011.

[35] Lamia Youseff, Maria Butrico, and Dilma Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. IEEE, 2008.

APPENDIX A

THE PROPOSED SLA ONTOLOGY

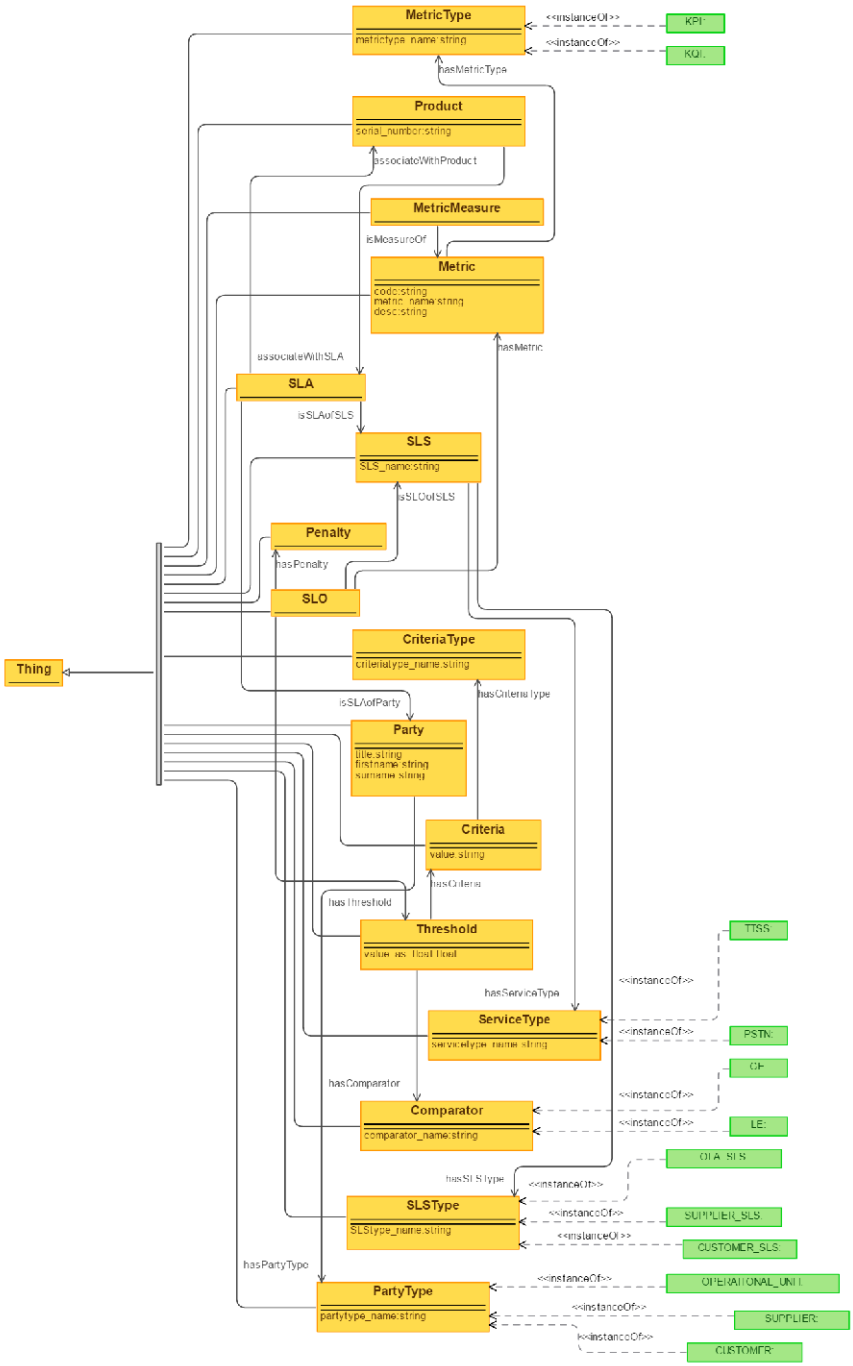


Figure A.1: Proposed SLA Ontology.