Developing a User-Centric Distributed Middleware for SLA Monitoring in SaaS Cloud Computing Using RESTful Services

Shaymaa Waleed Abdulatteef Al-Shammari

College of Science and Technology

School of Computing, Science and Engineering

University of Salford, Manchester, UK

Submitted in Partial Fulfilment of the Requirements of the

Degree of Doctor of Philosophy

February 2017

Table of Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, this research would not be possible without the blessing of my Lord ALLAH, the most Merciful, who has given me the patience and strength to finish this Ph.D. journey.

I would like to express my gratitude and sincere appreciation to my supervisor, Dr. Adil Al-Yasiri, for his assistance, guidance, and feedback during this study. I would also like to extend my thanks to the members of the School of Computing, Science, and Engineering at the University of Salford, for supporting me in this study.

I gratefully acknowledge the help provided by the Iraqi Ministry of Higher Education in funding this research. I owe a debt of gratitude to the Computer Engineering Department, Al-Nahrain University in Iraq, for the scholarship which allowed me to achieve the Ph.D. Thanks are also due to the Iraqi Cultural Attaché - London for their support during my study.

I would like to sincerely thank my friends, your kindness and support have made the journey more enjoyable, you have lifted me when I needed it the most.

Special thanks to my parents, my father Waleed Al-Shammari, his encouragement has ensured that I continued the path, it is his dream that I have fulfilled. Thanks to my lovely mother, Naheda Al-Bayati, for her love and prayers, which have sustained me throughout the study. Thanks to my brother Hisham for his kindness, and warm gratitude to my sisters, Wasan and Basma, who surrounded me with their love and care. I am eternally grateful for your unconditional love and support.

Declaration

Parts of the research presented in this thesis has been published in the following papers and presentations:

1- Al-Shammari, S., & Al-Yasiri, A. (2014). *Monitoring SLA's QoE of SaaS.* Presented at the Proceeding of the College of Science and Technology Dean's Annual Research Showcase 18 June 2014, University of Salford, MediaCity UK, Manchester, United Kingdom, pp. 38-39, 2014.

2- Al-Shammari, S., & Al-Yasiri, A. (2014). *Defining a metric for measuring QoE of SaaS cloud computing*. Paper presented at the Proceedings of the 15th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNET 2014), Liverpool, United Kingdom, pp. 251-256, 2014.

3- Al-Shammari, S., & Al-Yasiri, A. (2015). *An Approach for Embedding SLA Parameters in REST Services*. Poster and abstract presented at the Dean's Annual Research Showcase 28 June 2015, University of Salford, MediaCityUK, Manchester, United Kingdom, pp. 76-77, 2015.

4- Al-Shammari, S., & Al-Yasiri, A. (2015). *MonSLAR: A Middleware for Monitoring SLA for RESTFUL Services in Cloud Computing*. Paper presented at the IEEE 9th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA), Bremen, Germany, pp. 46–50, 2015.

5- Al-Shammari, S., & Al-Yasiri, A. (2016). *Estimating the QoE Value in SaaS Cloud Computing*. Presented at the Proceedings of the CSE 2016 Annual PGR Symposium (CSE-PGSym 16), University of Salford, Manchester, United Kingdom, p. 8, 2016.

List of Abbreviations

API	Application Programming Interface
ASP	Application Service Provider
CRUD	(Create, Read, Update, and Delete)
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
JSON	Java Script Object Notation
KPI	Key Performance Indicator
LOM	Largest of Maximum
MCDM	Multi Criteria Decision Making
MonSLAR	Monitoring SLA of SaaS using REST
MOS	Message Oriented Middleware
MOM	Medium of Maximum
PaaS	Platform as a Service
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
RPC	Remote Procedure Call
SaaS	Software as a Service
SLA	Service Level Agreement
SLO	Service Level Objectives
SMI	Service Measurement Index

SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
SOM	Service Oriented Middleware
TTP	Trusted Third Party
UDDI	Universal Description Discovery and Integration
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WS-Agreement	Web Services Agreement
WSDL	Web Service Description Language
WSLA	Web Service Level Agreement

Abstract

One of the most important discussions in the cloud computing field is user satisfaction with the associated services. It is important to maintain trusted relationships between clients and providers, for customers who pay subscriptions to receive these services in a timely and accurate manner. Despite the overwhelming advantages of cloud services, clients sometimes have problems in service outage and resource failure. This is due to the failures that can happen in cloud servers, which cause outages to the received services. For example, the failure of Microsoft Office 365 on 18th of January 2016, caused email disruption which lasted for many days. New measures are needed to ensure that the contract signed between the two parties, known as a Service Level Agreement (SLA) has been adhered to. Measuring the quality of cloud computing provision from the client's point of view is, therefore, essential in order to ensure that the service conforms to the level specified in the agreement; this is usually referred to as Quality of Experience.

In recent years, there has been an increase shift in using Simple Object Access Protocol (SOAP) to Representational State Transfer (REST) technology as an alternative technology in cloud applications APIs development. However, there is a penchant in most of cloud monitoring solutions to use SOAP protocol in managing the monitoring process. This trend has drawn the attention to the need for using REST technology in transferring the monitored data between the provider side and the client side.

This thesis addresses the problem of monitoring the quality of Software as a Service from the users' perspective, and the need for developing a lightweight middleware for delivering the monitored data in Software as a Service cloud computing. The aim of this research is to propose a user centric approach for monitoring Software as a Service in cloud computing, and to reduce the overhead caused by the monitoring process.

In order to achieve this aim, a user centric middleware capable of monitoring the Quality of Experience has been developed. The developed middleware is a Service Oriented middleware which uses RESTful web services and provides the monitoring process as an add-on service. A new approach was developed for embedding the SLA parameters in REST services through extending the HTTP messages and exploiting the HEAD and OPTIONS methods to transmit the monitored data and to send notifications about any

SLA violations. This reduces the need to exchange extra monitoring messages between the two parties, and hence reduces the communication overhead.

Furthermore, the estimation of the user satisfaction was implemented by developing a decision making approach to estimate the Quality of Experience value and to predict the effect of the SLA parameters and the Quality of Service (QoS) on the user satisfaction. Fuzzy logic techniques were employed in the decision making process.

The developed middleware is called MonSLAR, for Monitoring SLA for Restful services in SaaS cloud computing environments. The middleware was implemented using the Java programming language, and tested successfully in a cloud environment to prove the proposed solution's capability of transmitting the data using the REST methods, in addition to providing automated and real time feedback. MonSLAR uses a distributed monitoring architecture, which allows SLA parameters to be embedded in the requests and responses of the REST protocol. The proposed middleware was evaluated by measuring the overhead caused by using REST technology in terms of response time and message size and compared to existing techniques. The results revealed that the message size overhead of using REST is approximately five times less than the message size overhead caused by SOAP. Furthermore, the response time overhead of the monitoring process is comparable to the overhead caused by the available monitoring frameworks.

To sum up, the proposed middleware will help to strengthen the relationship between the client and the provider by using real time notifications to the client about any degradation in the cloud services, using a lightweight middleware.

CHAPTER ONE

INTRODUCTION

Cloud computing can be defined as the provision of services by equipping the resources of information technology via the Internet (Sarna, 2010). It is a huge aggregation of resources, which has been considered as an alternative to classical resources, since they are supplied on request (pay-as-you-go) (Chauhan et al., 2011). Cloud computing provides great benefit to customers and developers; for developers, it augments the computing power and the storage capacity to manage their applications, while for customers it ensures the availability of their documents regardless of the status of their machines (Miller, 2008). Cloud computing simplifies access to computing resources distributed over the Internet. In other words, it is a technique of doing computations using shared resources rather than using resources available on one site (Mathur & Nishchal, 2010).

Due to the important role played by cloud computing in individuals' daily needs, it has been considered as the 5th utility along with electricity, gas, water and telephony. Customers of cloud computing systems desire a trustworthy service. Despite the efforts of cloud providers to assure high availability of their services, customers need a guarantee for their rights in case of breach of contract (Buyya et al., 2009).

It is essential to maintain confidence among cloud suppliers and their customers since the customers expect to receive infrastructure services; so it is important to explain the ways of using and delivering these services. Like other services based on subscriptions, the relationship between suppliers and customers must be governed by a Service Level Agreement (SLA) (Chauhan et al., 2011; Firdhous et al., 2013a). Telecom operators first used SLA in the late 1980s within the contract with their customers (C. Wu et al., 2013). In order to avoid violations of the SLA, this agreement should designate the main metrics that are important to evaluate the provision of the SLA's terms (Wieder et al., 2011). An example of the SLA contracts in cloud computing are the services provided by Amazon Storage Services (AWS). Amazon promises an uptime of 99.9% with compensation paid to the client in the case of a violation of the SLA, provided the client submits evidence as proof for the lack of service (Muller et al., 2014).

The outages in the cloud services due to the servers' failure attracted the attention to the importance of using techniques to verify the correct delivery of the received cloud services. An example of this outage is the failure in Microsoft office 365 in 18th of January 2016, which caused an outage in the email services for many days. Another example is the failure in Salesforce in 10th of May 2016, which took days to solve this failure (Tsidulko, 2016).

Although the QoS is concerned with determining the quality of the network with respect to transmitted data, it does not take the users' needs into consideration. Quality of Experience (QoE) is the performance of the network from the user's perspective, in other words, it is the acceptance of services from the customer's point of view. The idea of QoE can become the guiding paradigm in the management of quality in cloud computing (Cicotti et al., 2012; Zeginis & Plexousakis, 2010).

User satisfaction is defined as "a measure of perceived performance relative to expectation" (Larson, 1998), or someone's feeling in a specific situation about factors that affect this situation (Bailey & Pearson, 1983). In other words, QoE measurements can be used as an indication to measure user satisfaction. This research attempts to combine these ideas from a network and software engineering point of view.

It is a common practice for each service oriented architecture to have a middleware that upholds the management of the SLA (Marinescu, 2013).

In SaaS cloud computing, service oriented architectures and web services are used to provide a computing model that is globally accessible. According to Velte et al., a web service is a software designed to support interaction from machine to machine (Velte et al., 2009). Two main types of web services are available: SOAP and REST.

In SOAP protocol the services and the formal mechanism for invoking these services are described in WSDL (Web Service Description Language), which contains the information about the services, the expected QoS to be delivered to the consumer, and details about SLA parameters (Marks & Lozano, 2010). However, this is not available in REST services.

REST is an architectural style used in distributed systems, which supports stateless communication and platform independence (R. T. Fielding, 2000; Marinescu, 2013). Nowadays, there is an apparent shift from SOAP services to REST services, particularly in cloud computing (Shroff, 2010) because of the advantages that REST offers such as simplicity, ease of use, better response time, light weight, and improved server scalability (Velte et al., 2009). Most cloud providers use REST and HTTP to represent their services such as Windows Azure (Microsoft Azure, 2017) since REST is considered a lightweight protocol (Marinescu, 2013). This trend to use REST in cloud computing has drawn attention to finding a way to represent the SLA parameters in this style of architecture and to include the SLA parameters that have emerged. Whereas SOAP has been used as a technique for transferring the monitored data in the monitoring frameworks. However, it is important to mention that the use of a lightweight technique (REST) has led to a reduction of the overhead caused by SOAP. It is important to keep a balance between the monitoring accuracy and the overhead caused (Lu et al., 2016).

1.1 Service Level Agreement

The (SLA) is defined by Jin et al (2002) as “an agreement regarding the guarantees of a web service. It defines mutual understandings and expectations of a service between the service provider and service consumers” (Jin et al., 2002). It is a contract held between the client and the service provider which contains information about the expected levels of services delivered to the clients, the provider’s guarantees for the QoS, the commitments of both parties, and the penalties in case of violating the SLA (Marinescu, 2013). This agreement can help in managing long-term use of service business relationships (Alhamad et al., 2010). The SLA document is composed of the following parts: **purpose**, which explains the causes of building the SLA; **parties**, presents those who are engaged in the agreement; **validity period**, which is the time in which the SLA is active; **scope**, describes the services defined in the SLA; restrictions, defines the steps required to provide the service to the customer; **Service Level Objectives (SLO)**, which specify the levels of services that have to be achieved such as availability and reliability; **penalties**, presents the actions to be taken in case of violating the SLA which can be either a termination of the agreement or reduction in the service’s price; **optional services**, presents the services which may be required by the client as an exceptional; **exclusions**, refers to items not defined in the agreement; and **administration**, which specifies how to manage monitoring the objectives of the SLA (Jin et al., 2002). The common SLA parameters give indication about the operation quality metrics, for instance the SLA parameters of SaaS include scalability, reliability, availability, and usability (Alhamad et al., 2010). An example of SLA document is presented in Appendix A.

SLA plays an important role in maintaining cloud customers' rights through identifying the resources and the services that the clients should get from the cloud. However, the provider has the chance to mislead the customer by providing fewer resources which then permits the providers to increase their earnings by supporting more customers (Ye et al., 2012). Since the cloud services are offered to a wide range of customers, resources in the cloud gained and freed effectively according to clients' demands, means that this resilience in resource provisioning makes the enforcement of an official model difficult to implement (Rak et al., 2013). As a consequence, the QoS parameters should be identified in details (Zhu et al., 2012).

Alhamad, et al. (2010) discussed the significance of an SLA document to manage the relationship between the providers and the consumers in cloud computing. They proposed the main criteria that should be considered at the stage of defining SLAs and classified these criteria according to cloud computing services. The study also emphasized ways to define and monitor SLA parameters and set penalties in case of violation of these documents (Alhamad et al., 2010).

1.2 Problem Statement

Due to the nature of the cloud computing environment, which is varied and complex, it is difficult to ensure that providers will honour their promises with respect to the level of service being supplied to their clients. Therefore, it is essential to specify the contractual obligations between cloud providers and clients in order to stipulate the technical terms of the services provided as well as other legal requirements. The relationship between providers and clients is normally captured in an SLA. SLAs contain a description of QoS parameters that govern how the service level is determined (Zhu et al., 2012), QoS parameters should be identified precisely and quantifiably. Monitoring SLA parameters is usually carried out by cloud providers (Alsulaiman & Alturki, 2012) who leave the detection of the SLA violations to the clients (Cedillo et al., 2015). However, this process should be isolated from the provider to assure the client's trust, for customers cannot rely on the service providers to verify the SLA compliance. In other words, both cloud providers and clients are not suitable for evaluating an SLA or received cloud services (Ye et al., 2012), (Nguyen et al., 2014). One of the challenges of user side management in a cloud environment is the absence of an approach to inform the user about changes in the quality of the delivered services (Rehman et al., 2015). This presents a problem for clients who pay for these services and require the ability to monitor compliance. In this research, the aim is to investigate ways to enable the cloud's clients to verify that the services provided in the cloud conform to the stipulated SLA. To achieve this goal, an SLA driven middleware is presented.

The limitations of monitoring and controlling the parameters of the SLA stem from the difficulty in monitoring them in real time, which means that there is no way for the cloud client to directly monitor the service level using the QoS metrics. The monitoring of SLA forms an essential function in detecting situations where the SLA has been violated and then deciding the best course of action that must be taken as a consequence of the violation.

Although measuring QoE is subjective, it is important to find a way to determine any deviation from it (Brooks & Hestnes, 2010), and to develop a system to monitor the QoE as described in the SLA, taking into consideration the end user's point of view. To date, there has been a lack of research to address the presented problem, Chapter three will provide more details in order to bridge this gap.

1.3 Research Motivation

The research is motivated by the fact that in previous research WSDL has been used in conjunction with SOAP to represent the SLA measurements. WSDL is used to specify the functionality of web services and their methods; hence, it seems logical to extend it for specifying SLA measurements within one document. However, WSDL is a SOAP related technology which cannot be used with REST services; this is in addition to the high overhead associated with WSDL. It is, therefore, important to search for an alternative method that can be used with REST and eliminate the need for technologies like WSDL to manage the monitoring process and represent the SLA parameters. Section 3.4.2 introduces more details about the research.

The motivation for this research is twofold:

1- To increase the ability of the clients' trust in the services provided by SaaS providers, through developing a middleware that implicitly monitors the SLA compliance from the clients' perspectives. The research is motivated by noticing the weakness of the available research in handling this problem. Chapter 3 presents a review of the related work.

2- To extend the methods of REST services to include the SLA parameters which will improve the process of monitoring SLAs in cloud computing as existing REST services frameworks lack an automated SLA management process.

1.4 Research Aim and Objectives

The aim of the research is to propose a user centric approach for monitoring Software as a Service in cloud computing, and to reduce the overhead caused by the monitoring process. The following objectives are relevant to achieve the aim of the research:

1- To gain a detailed understanding of monitoring SLAs and user satisfaction in cloud computing.

2- To develop an approach for lightweight user-centric monitoring of SaaS in cloud computing.

3- To develop an approach to measure client satisfaction with services provided in cloud computing, in terms of QoE value.

4- To evaluate the proposed solution through using simulation experiments to test the performance of the system.

Figure 1-1 shows the research questions and the chapters that present answers to these questions.

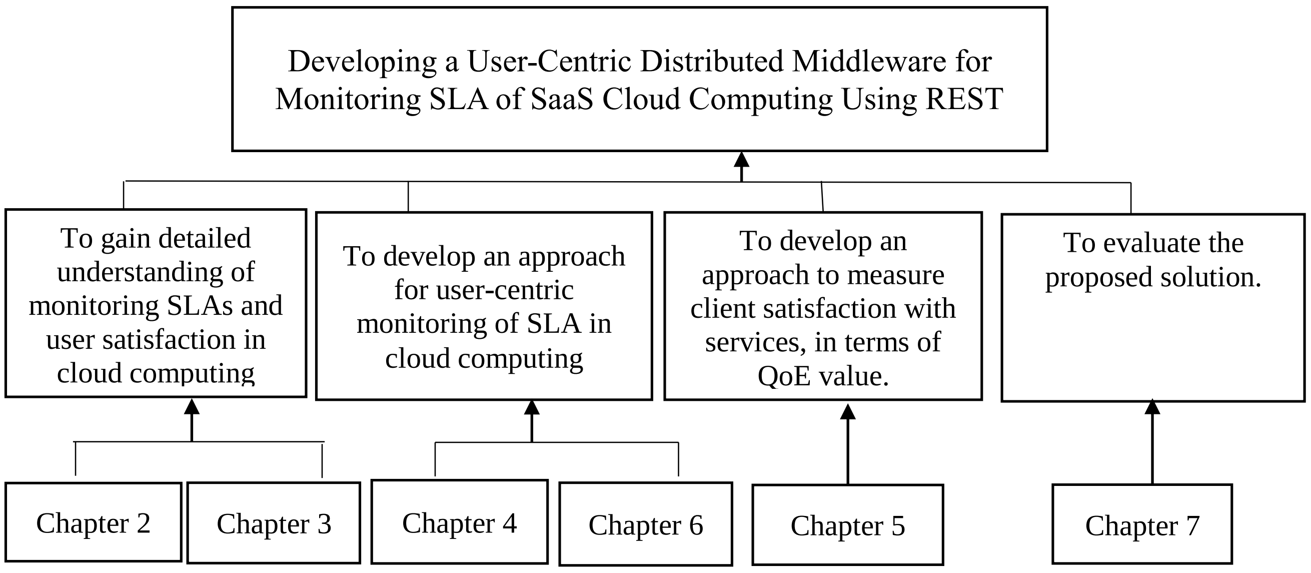

Developing a User-Centric Distributed Middleware for Monitoring SLA of SaaS Cloud Computing Using REST
To gain detailed understanding of monitoring SLAs and user satisfaction in cloud computing
To develop an approach for user-centric monitoring of SLA in cloud computing
To develop an approach to measure client satisfaction with services, in terms of QoE value.
To evaluate the proposed solution.
Chapter 2
Chapter 3
Chapter 4
Chapter 6
Chapter 5
Chapter 7

Figure 1-1 achieving the research objectives in thesis's chapters

1.5 Research Contributions

The main contributions of the thesis are in the field of estimating the QoE in SaaS cloud computing in order to achieve the goal of assuring user satisfaction about the SLA compliance with the promised services. The main contributions are:

1- A new approach to support using REST in monitoring SLA of SaaS. This approach considers embedding SLA parameters in REST services by transferring the monitored data and the estimated QoE value using HEAD and OPTIONS methods of REST architecture and embedding this data in the header of the HTTP messages. This helps in reducing the need for the creation of dedicated messages to transfer the data.

2- An approach for estimating QoE in SaaS cloud computing. The overall user satisfaction about the SaaS service has been considered as an indication of the QoE value. This approach considers estimating the value of QoE as a function for both SLA parameters and QoS. The fuzzy logic engine is used for estimating the value of the QoE based on the collected monitored data from the probes and the negotiated SLA parameters in the SLA document. The proposed approach has been published (Al-Shammari & Al-Yasiri, 2014) and presented in Chapter Five.

3- A novel framework for a user-centric middleware that provides an automated estimation of the QoE value in SaaS cloud computing through employing the approaches introduced in 1 and 2 respectively. The middleware is a service oriented middleware, which provides the monitoring process as an add-on which takes the advantage of REST technology for service binding instead of SOAP technology which has been used so far in the available research. The framework considers two versions of the middleware on both the provider side and the client side without the need for a third party (broker) to manage the estimation process. The proposed middleware has been published (Al-Shammari & Al-Yasiri, 2015).

1.6 Significance of the Study

Taking into consideration the research problem, handling the knowledge gap and the importance of managing, the monitoring process with low overhead has revealed the need for a lightweight monitoring middleware. The study presented a SOM with monitoring services based on REST. The achievements of this thesis could benefit both academic research and the cloud computing industry. The advantage in the academic field includes the new approach for automated user centric monitoring, and the exploitation of REST as a connecting technology in SOM. while in the industry, it is through keeping the confidence between the client and the provider by supplying the client with a facility to check the overall SLA compliance according to user requirements.

1.7 Research Methodology

This section presents the research methodology that has been developed and adopted for this research. According to Creswell (2013), to develop a research, three different concepts should be taken into consideration, which are the philosophical assumptions, the research design, and the research methods used as shown in

Figure 1-2.

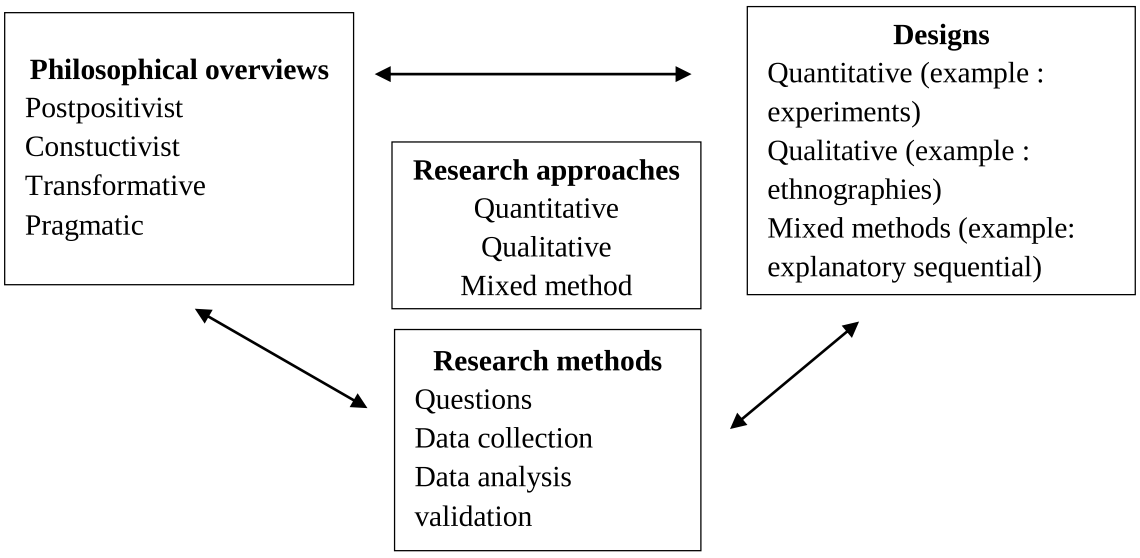


Figure 1-2 The research framework, adapted from Creswell (2013)

The philosophical overview used is the Postpositivist, which is also called the scientific method, as it depends on empirical experiments and measurements. The research methods involve data collection, analysis, and validation, which are depicted in Figure 1-3.

There are three different research approaches: quantitative, qualitative, and mixed methods. The qualitative method includes understanding the person's behaviour and attitude. This method includes using interviews and focus groups and the investigator interpreting the meaning of the collected data, whilst the quantitative method is "an approach for testing objective theories by examining the relationship among variables. These variables, in turn, can be measured, typically on instruments, so that numbered data can be analysed using statistical procedures”. The mixed method involves integrating the quantitative and qualitative methods. The research design includes the strategies of inquiry for each approach (Creswell, 2013) (Dawson, 2002).

The research approach used in this thesis leans to the quantitative approach as it depends on quantitative rigour analysis, and the research design is based on experiments. A scientific research process is used to achieve the goals of the research as shown in Figure 1-3. The main phases of the research methodology are as follows:

1- Investigate previous literature

   This step includes reviewing previous relevant studies, to get a good understanding, investigating the issues related to managing SLA documents in cloud computing in general, as well as estimating QoE, and the available frameworks for managing the cloud environments. This provided good knowledge about monitoring SLAs in the cloud environment.

2- Identify the research problem

   The literature review helped to diagnose the drawbacks found in the current solutions for estimating the QoE in cloud computing. The research started by identifying the requirements for a middleware in cloud computing systems to assure the validity of the SLAs in cloud computing. To achieve this, the research focused on studying the main parameters used to estimate the user satisfaction in SaaS and investigate available SLA monitoring frameworks.

3- Design a new middleware capable of monitoring user’s satisfaction

   Based on the research problem presented in the previous stage, this stage includes presenting a new middleware (MonSLAR), to manage the estimation of the QoE. MonSLAR presents a new approach for extending REST services methods to include the SLA parameters by embedding the QoE and the SLA parameter values in the responses of the REST protocol using the HEAD and OPTIONS methods. MonSLAR is presented in Chapter Four.

4- Develop an approach for estimating the QoE value

   This includes presenting an approach for estimating the QoE value taking into consideration the characteristics of SaaS. QoE is defined as a function of both the

SLA parameters and the QoS values. Fuzzy logic is used to implement the proposed approach, which is presented in Chapter Five.

5- Evaluate the proposed middleware

   The evaluation step includes evaluating MonSLAR to analyse its performance in terms of response time and message size, the acquired results for the message size overhead is compared with the existing monitoring frameworks. The system is evaluated by means of simulation results in comparison with another middleware, in addition to a qualitative comparison between MonSLAR and the available monitoring frameworks. This step also includes a survey study to evaluate the proposed metric, then applying amendments where required based on the evaluation results. The evaluation process is presented in Chapter Seven.

6- Publish the contributions and write up the Ph.D. thesis

   The final stage includes presenting the final verified forms of the metric and the middleware. This step also includes publishing the proposed approaches and writing up the Ph.D. thesis.

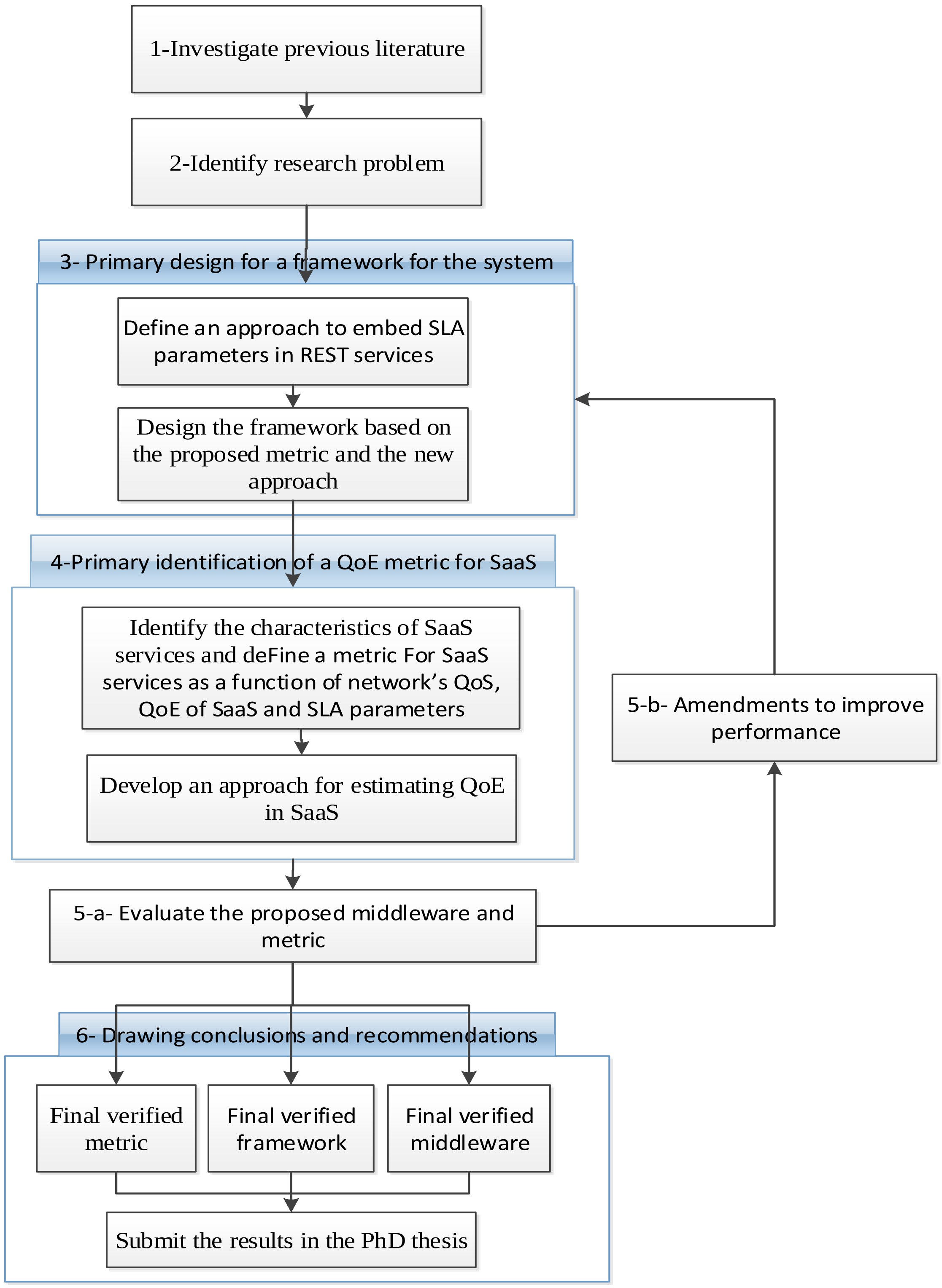

1-Investigate previous literature
2-Identify research problem
3- Primary design for a framework for the system
Define an approach to embed SLA parameters in REST services
Design the framework based on the proposed metric and the new approach
4-Primary identification of a QoE metric for SaaS
Identify the characteristics of SaaS services and deFine a metric For SaaS services as a function of network's QoS, QoE of SaaS and SLA parameters
Develop an approach for estimating QoE in SaaS
5-a- Evaluate the proposed middleware and metric
5-b- Amendments to improve performance
6- Drawing conclusions and recommendations
Final verified metric
Final verified framework
Final verified middleware
Submit the results in the PhD thesis

Figure 1-3 Research process

1.8 Thesis Outline

The thesis is organized into eight chapters as follows:

Chapter Two presents a background to the concepts and technologies related to the thesis research area of cloud computing, web services technologies, together with the middleware architectures.

Chapter Three presents an overview of the literature related to the research subject in order to attract attention to the knowledge gap and the lack of previous research to approve the research problem. The literature is selected to cover the research subject, which is cloud monitoring, QoE measurement, and the SLA oriented monitoring concept.

Chapter Four presents an overview of the suggested solution; the architecture of MonSLAR is presented as the proposed middleware, which is discussed with a description of its characteristics. Furthermore, an approach for using REST methods to manage the monitoring process is presented with a discussion on how it has been used to solve the problem.

Chapter Five presents the approach used for estimating the QoE value, details of the fuzzy logic engine. The main factors used for estimating the user satisfaction are discussed.

Chapter Six presents the implementation of the proposed middleware, the technologies used in the implementation and data formats used in transmitting the data. The chapter discusses the implementation of the REST methods used in the monitoring process. and a detailing of the main RESTful transactions used within the middleware in addition to the representation of the monitored data on the client side.

Chapter Seven introduces the evaluation of the proposed middleware and tests the performance and usefulness of the proposed REST approach. Simulation results are used and a comparison is introduced for the proposed approach using REST with SOAP protocol, in addition to qualitative comparisons with the available monitoring frameworks. The chapter also presents a user study evaluation for the estimation of QoE using the fuzzy logic engine.

Chapter Eight concludes the thesis by summarizing the most important outcomes of the research and highlighting the main points; this chapter also suggests different directions in which this research can be continued.

Appendices are presented to supplement the chapters of the thesis as follows:

Appendix A presents an example of SLA document for SaaS service.

Appendix B shows a study to investigate the effect of using different defuzzification methods on the QoE level.

Appendix C displays the API specification of the proposed middleware.

Appendix D presents the java code of the proposed middleware.

Appendix E investigates the performance of the proposed middleware through studying the overhead caused by the monitoring process in terms of response time.

Appendix F presents the user study survey used for evaluating the use of SaaS-Qual as a model for estimating a QoE value.

CHAPTER TWO

BACKGROUND IN CLOUD COMPUTING

2.1 Introduction

The main feature of cloud computing is the ability to relocate the software and IT industry into services, and formalizing the way that these services are designed and purchased (Armbrust et al., 2010). Accessing these services in the cloud environment is managed using web services such as SOAP and REST (Shroff, 2010), it also requires the use of SOA as an architecture to provide the software components as services. The distributed nature of cloud computing presents the need for using the middleware as a tool to manage its functionalities.

In this thesis, a middleware for monitoring SaaS services in cloud computing is presented. This chapter introduces the main fundamentals related to the subject of the research, the background knowledge is presented to better understand how to manage cloud computing and the main elements used in cloud computing and exploited in this research. Figure 2-1 shows a mind map which summarizes the main topics related to the subject of the thesis and the rationales for delving into the submitted details.

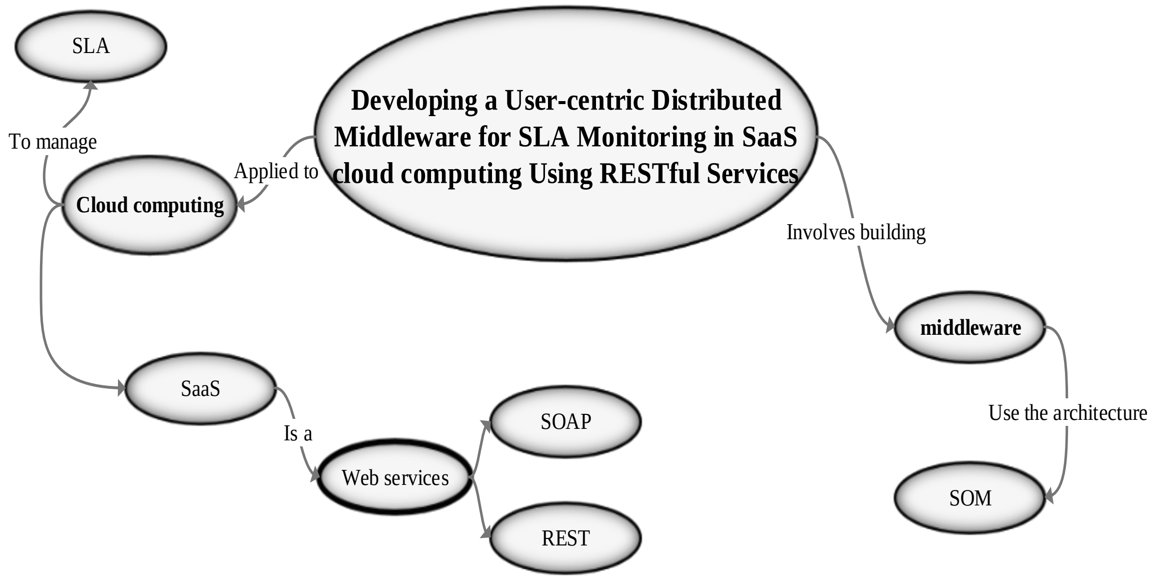


Figure 2-1 Mind map shows the background main fundamentals

2.2 Cloud Computing

Cloud computing is a technique of doing computation through shared resources rather than using resources available on one site. It simplifies the access to computing resources distributed over the internet (Mathur & Nishchal, 2010). These resources are used in a pay-as-you-go way (Vaquero et al., 2008). This way for providing the resources presents many advantages in terms of cost saving and the need for IT maintenance services. Furthermore, the scalability in getting the required resources on demand, which provides unlimited resources' capacities, leads to an improved performance by running the applications on these resources instead of the client's Personal Computer (Velte et al., 2009) (Miller, 2008). Armbrust et al. described the cloud as "the long-held dream of computing as a utility" as it turns most of the IT-industry into as a service delivery (Armbrust et al., 2010).

The main characteristics of cloud computing were presented by Mell and Grance (2010), this includes: network access, i.e. that the services are available and accessed by the client's devices through the network; self-service, where the user can directly customize the received service as needed without the need for interaction with the service provider which helps in saving both time and effort (Voorsluys et al., 2011); elasticity, which refers to the extendibility of the capabilities of the services provided, proportional to the required services; resource pooling, where this feature implies that the resources are pooled in order to serve different users in a high level of abstraction so that those users are unaware of the location of the resources provided; and measured services, where the resources are measurable and controllable in the cloud computing systems, this feature is usually offered on a pay-as-you-go basis (Mell & Grance, 2010).

Cloud computing can be categorized according to possession and administration to four cloud deployment models (Mell & Grance, 2010). First, public cloud, in which services are openly accessible to the public. This type of cloud can be possessed by government, institutions or a mixture of them. Second, private cloud, where information is dominated and secured for special firm by reserving the datacentres for that firm, where many clients are included. Third, hybrid cloud is a combination of the other cloud deployment models like private and public clouds through maintaining the security of the private firm cloud with the ability to expand in case of a heavy workload (Furht, 2010). Finally, community,

in which many enterprises with comparable necessities partake in infrastructure to increase scalability and decrease cost.
The next subsection presents the main cloud models.

2.2.1 Cloud Computing Models

Cloud computing can be defined as the provision of applications and infrastructure resources as services (Xu, 2010). Three main services have been used to classify cloud computing (Mell & Grance, 2010), these services are:

1- Infrastructure-as-a-Service (IaaS), which supplies infrastructure resources as-a-service like storage and computing so that the client has control and can manage the operating systems and the storage services. This kind of services suite the institutions who needs a rapid and cheap extra resources.
2- Platform-as-a-Service (PaaS) is usually used by developers to build and run their applications by providing the ability to build and test the user-created applications in a cloud execution environment. In this kind of service, the client has the ability to control the application deployed but has no control over the operating systems and the storage services.
3- Software-as-a-Service (SaaS) fundamentally refers to the applications provided on the cloud by the service provider to the end user in order to free the client from the burden of installing and maintaining these applications (Mell & Grance, 2010) (Hill et al., 2012).

These three services are the main cloud services, despite the tendency of many providers to use other terms to distinguish their services from the others, such as storage-as-a-service, communication-as-a-service, backup-as-a-service, and servers-as-a-service, etc. (Finn et al., 2012; Rittinghouse & Ransome, 2009).
The next section discusses in more details SaaS, its characteristics, and advantages.

2.3 Software-as-a-Service (SaaS)

The concept of Software-as-a-service (SaaS) was first presented and defined by the Software & Information Industry Association (SIIA) in 2001 as the deployment of applications or services from the central data centre through the internet or LAN, as a kind of paid subscription based network (Software & Association, 2001). It is the delivery of an application to the client as a service, the management of the application delivered to the client and the underlying infrastructure managed by the SaaS vendor (Kavis, 2014). This kind of delivery of software via the internet represents a competitor to the traditional applications installed on the clients' devices (Cusumano, 2010).

Software as a Service first emerged in the 1990s. The application services were hosted by the ASP (Application Service Provider) so that the client can get access to the application through the internet, but this model of the software delivery was unfeasible because of the limited network bandwidth and the slow speed of the internet at that time. In addition, this model for delivering the applications did not support multi clients, in other words, the applications were delivered on a one to one basis by storing the application for each client in the ASP data centre. The use of the cloud computing helped the application providers to host their applications in the cloud to take advantage of the cloud scalability characteristics by managing the increased number of requests and the multi users of the application (Menken & Blokdijk, 2009).

One of the main characteristics of SaaS is the segregation of the software usage from its ownership. The SaaS provider owns the application while the client rents the service on demand (Turner et al., 2003), while in the legacy application systems the user is the owner of the application. Another feature of SaaS is the multi-tenancy support so that many clients can access the same software application (Menken & Blokdijk, 2009). SaaS also considers the management of the commercial software in a network based instead of managing it on the client's side as it is remotely accessed through the internet (Rittinghouse & Ransome, 2009).
SaaS provides many advantages for both the client and the provider. On one hand, SaaS helps to diminish the storage space in the users' machine and helps them to save money through paying on demand instead of buying the application in full. On the other hand, it helps the vendors to reduce privateering and the making of unlicensed copies of the software as it is kept in their hands, SaaS also allows the vendors' to increase their profits by receiving continuous subscriptions from clients instead of a one-off payment for the purchase of the application (Menken & Blokdijk, 2009).

One of the main characteristics of SaaS is the ability to allow multi tenants sharing the same cloud application resources, which is known as multi-tenancy (Cai et al., 2012). Multi-tenancy helps applications providers to reduce the cost of deploying the applications, where an instance of the software application can be used by many tenants. The concept of multi-tenant means that different tenants can use the same application with different configurations and hence an SLA of each tenant can be different (Bezemer & Zaidman, 2010). It is essential to ensure a proper isolation between the tenants in SaaS multi-tenancy environment which includes the data stored in the database. Different architectures can be considered in designing multi-tenancy based on the degree of sharing the resources among the tenants, this could be sharing the application itself, the database, or the infrastructure resources of the cloud (Momm & Krebs, 2011).

Many vendors offer SaaS services, for example, Adobe software is one of the SaaS services available online on a subscription basis, Google also offers Gmail and Google Apps, which are offered for free with restricted features or on a payment basis. Microsoft offers SaaS applications like Microsoft-Office and web emails, whilst Salesforce is one of the well-known SaaS vendors, providing the Customer-Relationship-Management (CRM) applications (Menken & Blokdijk, 2009). According to Gartner, SaaS sales are predicted to be duplicated by 2019 (Gartner, 2015), which is an indication of the importance of this kind of cloud services.
SaaS has been investigated by many researchers. For instance, the problem of adopting SaaS was handled by Tan et al. (2013) through developing a methodology to assess the advantages of adopting SaaS in an organization's business (Tan et al., 2013). While the work of Godse and Mulik (2009) set out to select the best SaaS that satisfies customer requirements through prioritizing the features of the products, this work did not consider the SLA parameters (Godse & Mulik, 2009).

This section presented SaaS as an application service delivered through the internet as a web-based application. This concept is the basis of web services which assume that functionalities provided by organizations are offered as services (Gustavo et al., 2004). In the next section, the main technologies used in cloud computing are discussed.

2.4 Technologies Used in Cloud Computing

This section presents the main technologies that support cloud computing, and used in the development of the proposed solution in this thesis. These technologies play an important role in building, managing, and improving the cloud. The section is subdivided into many subsections to introduce these technologies; to start with, SOA is presented due to its properties which make it suitable for cloud computing. The software components are used as services in the cloud architecture, so that the services are moved easily between the cloud datacentres (Hurwitz et al., 2010). In addition to SOA, web services are discussed. The web services facilitate the communication of the applications over the internet, which in turn help in requesting the service in cloud computing (Shroff, 2010). Finally, due to the heterogeneity of cloud computing, finding a way for managing its resources and services is important which is accomplished using the middleware.

2.4.1 Service Oriented Architecture (SOA)

SOA was defined by Kurbel (2008) as “a software architecture that defines the use of services to solve the tasks of a given software system” (Kurbel, 2008). SOA can be considered as a combination of services; as such in reality, where users are interested about receiving the services more than the software components in charge of implementing these services. The aim of using SOA is to improve the efficiency and productivity of companies; this is due to the fact that one piece of software can be reused as a service which reduces the need for building this service again, where rebuilding the software components causes a redundant functionality (Erl, 2008).

A service in SOA has features that distinguish it from the traditional software components, this includes reusability, automaticity, in addition to its high portability characteristic (Wei & Blake, 2010). The services can be a program code that can be used in distributed systems, these services can be delivered using SOAP or REST as discussed in the next subsections.

The appearance of cloud computing helped in improving the business consideration of SOA (Hurwitz et al., 2010).

The terms of service in Service Oriented Architecture (SOA) is different from that in cloud computing. Whilst in cloud computing it refers to the services and resources available to the client, the service in service oriented technology indicates the software’s function. The use of SOA in cloud computing helps in integrating its components (Yang et al., 2015).

2.4.2 Web Services

Web services were defined by Jin et al. (2002) as “Internet based applications that communicate with other applications to offer business data or functional services programmatically” (Jin et al., 2002). According to (Velte et al., 2009), the web service is

a software designed to support interaction between machines. Web services allow disparate systems to integrate and show the functions that can be invoked over HTTP (Daigneau, 2011). The term 'web services' has been defined by Chavda (2004) as "A programmable application component that can be accessed over the internet and used remotely" (Chavda, 2004). This technology has changed the concept of business on the web by enabling the applications to communicate and provide services to each of the applications and the devices that support web access.
Another comprehensive definition for web services is presented by Papazoglou (2008) as "A platform-independent, loosely coupled, self-contained, programmable web-enabled application that can be described, published, discovered, coordinated, and configured using XML artifacts (open standards) for the purpose of developing distributed interoperable applications". Where 'loose coupling' refers to how far the service is independent of the underlying technologies; 'self-contained' as it implements a distinct function which can be invoked by the client; 'programmatically accessed', as the web services can be invoked and queried at the "code level" which can improve the web service's client efficiency (Papazoglou, 2008)
Web services are considered the basis for allowing the web applications to be deployable and accessible through the internet using a distributed architecture. As it depends on internet standards, they help in developing complex services by combining them (Papazoglou, 2008). It allows integrating the services offered by web applications and illuminates the need for the details of the services' implementation in the interaction process and the sharing of data between the organizations, based on a standardized method regardless of the underlying platform and programming languages (Hill et al., 2012).

Web services are different from web applications. Web services are provided as accessible resources and can be requested by other web services independently from direct human interference (Papazoglou, 2008).

The web services are used in cloud technology to request the services provided in cloud computing (Shroff, 2010). The use of web services presents an advantage by simplifying sharing and reusing common logic with a variety of clients, such as web applications, desktops or laptops. This is because of the use of web standards that are interoperable among the computing platforms, like HTTP, JSON, and XML.

There are two main ways for describing the web services, which are SOAP and REST (Daigneau, 2011). These technologies are presented in the next subsections of this chapter.

2.4.2.1 SOAP

SOAP (Simple Object Access Protocol) is a messaging protocol that uses HTTP (Hyper Text Transfer Protocol) and XML (Extensible Markup Language) to hide the heterogeneity in the distributed-platforms and manage the connection between the service requester and receiver. This is achieved by exchanging XML documents. These exchanged messages are known as SOAP envelopes. The SOAP message is comprised of two main parts: the message header and the message body as shown in Figure 2-2. These messages are used for invoking web services by encapsulating the SOAP request in the transport protocol, this is done by transmitting the SOAP message in the body of an HTTP POST request. SOAP is stateless as it is used with the HTTP (Papazoglou, 2008).

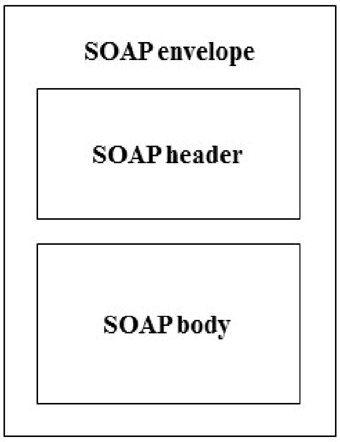
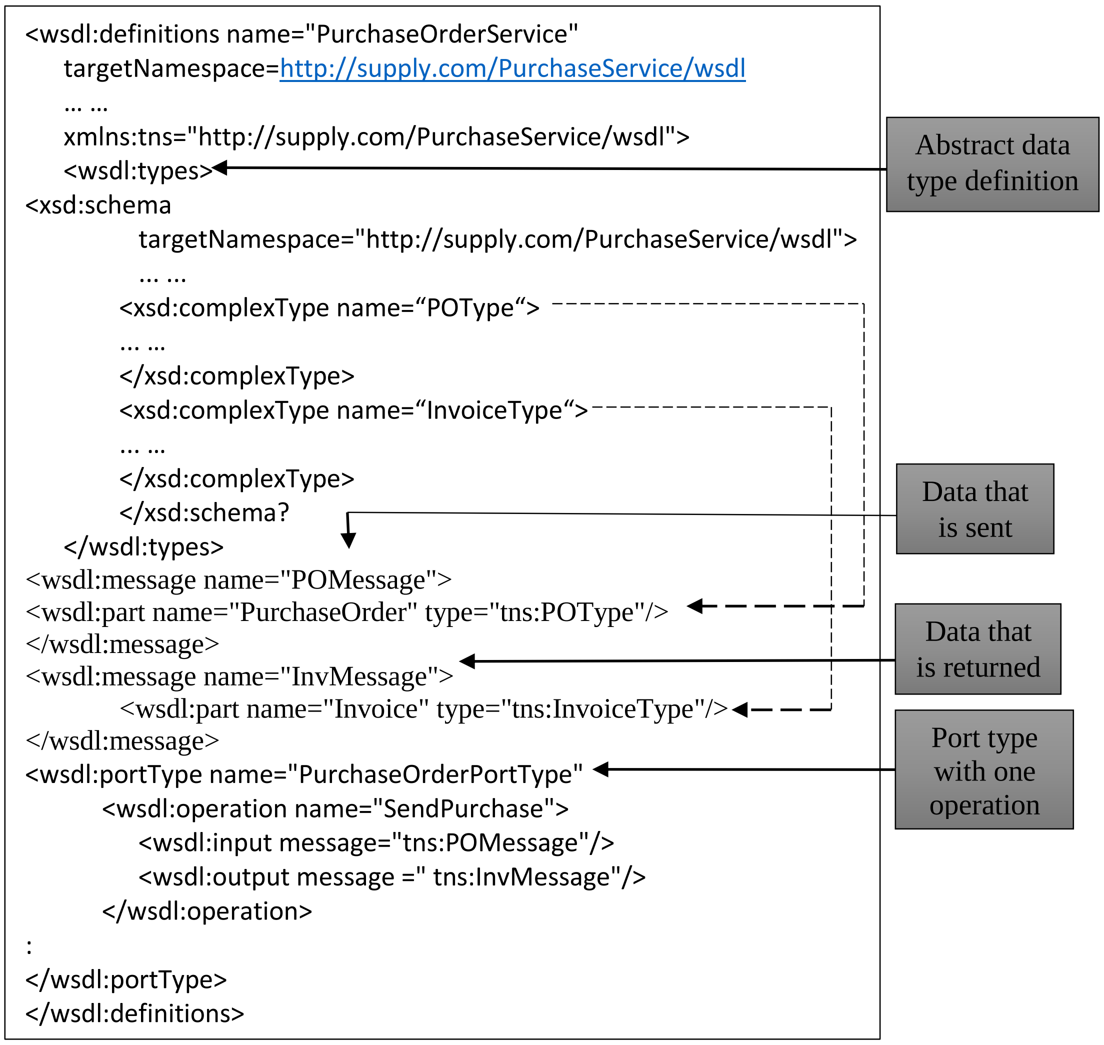


Figure 2-2 SOAP envelop (Gustavo et al., 2004)

SOAP protocol appeared as the technology for web services in the definition of web services by the World Wide Web Consortium (W3C) that defined web services as, “a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages” (Booth et al., 2004).

In SOAP protocol, the web service requires to be defined. This is achieved by using WSDL (Web Service Description Language), which provides a description of the web service, the functions performed by this service and the type of data, in addition to the interfaces of the service. The WSDL is represented using XML schema (Papazoglou, 2008). Figure 2-3 shows an example of a WSDL that displays its main parts.

For the web service to be discovered and used, there is a service registry, which consists of the services offered by the providers. This registry is called UDDI (Universal Description Discovery and Integration). UDDI emerged to cover the concept of business, where a registry provides information about organizations and services provided by them, and the interfaces for invoking these services. The process of invoking the web services in SOAP protocol is illustrated in Figure 2-4. WSDL is used in the UDDI to provide information about the description of the service (Gustavo et al., 2004). UDDI is considered as “yellow pages” for the clients to find a WSDL of required web services, this registry is an XML schema composed of business entity, business service, binding template, and a tModel (Richardson & Ruby, 2008). The process of mapping the information of the services in WSDL to the UDDI is depicted in Figure 2-5, this process helps in extracting the required services’ information to be used in the UDDI, in order to manage the interfaces in business’s applications.



```
<wsdl:definitions name="PurchaseOrderService"
    targetNamespace=http://supply.com/PurchaseService/wsdl
    … …
    xmlns:tns="http://supply.com/PurchaseService/wsdl">
    <wsdl:types>
<xsd:schema
            targetNamespace="http://supply.com/PurchaseService/wsdl">
            … …
          <xsd:complexType name="POType">
          … …
          </xsd:complexType>
          <xsd:complexType name="InvoiceType">
          … …
          </xsd:complexType>
          </xsd:schema?
    </wsdl:types>
<wsdl:message name="POMessage">
<wsdl:part name="PurchaseOrder" type="tns:POType"/>
</wsdl:message>
<wsdl:message name="InvMessage">
          <wsdl:part name="Invoice" type="tns:InvoiceType"/>
</wsdl:message>
<wsdl:portType name="PurchaseOrderPortType">
        <wsdl:operation name="SendPurchase">
            <wsdl:input message="tns:POMessage"/>
            <wsdl:output message =" tns:InvMessage"/>
        </wsdl:operation>
:
</wsdl:portType>
</wsdl:definitions>
```

Abstract data type definition
Data that is sent
Data that is returned
Port type with one operation

Figure 2-3 An example shows a WSDL, adapted from (Michael Papazoglou, 2008)

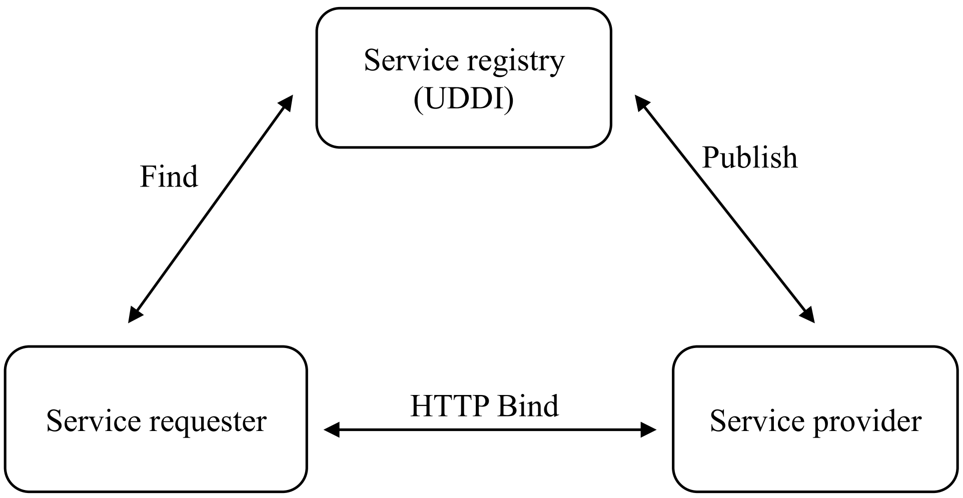


Figure 2-4 SOAP service invocation process (Michael Papazoglou, 2008)

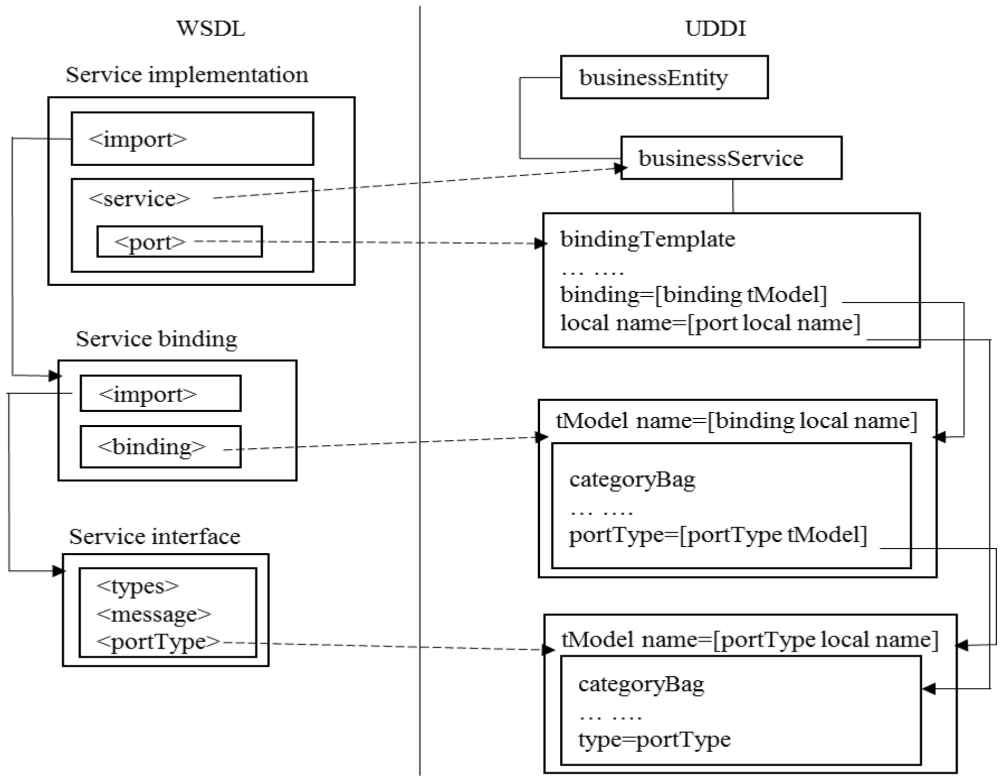


Figure 2-5 depicts mapping WSDL to UDDI (Michael Papazoglou, 2008)

2.4.2.2 REST

REST (Representational State Transfer) is an architectural style used in distributed systems, which was first defined by Fielding (2000) in the dissertation submitted for his doctoral degree, Fielding started to work on REST in 1994 as a guide for developing the architecture of the web while developing the specifications of HTTP/1.0 and the proposed HTTP/1.1. There are six main constraints of the REST architecture as follows: **client-server architecture**, which gives an advantage in terms of scalability and portability through separating the client and server; **statelessness**, which indicates that the state of the requests is not saved which requires that the required information be contained in that request, so that each request contains all the information required for the request to be understood by the server; **cacheable**, which gives the client cache to reuse the response data for similar requests; **uniform interface**, which improves simplicity and decoupling of services in REST; **layered system**, in which it splits the functionalities into hierarchical layers; **on demand code**, which improves the client's ability to request codes on the server side using scripts and applets (R. T. Fielding, 2000). Figure 2-6 shows the communication style in REST architecture. It can be mentioned that it is a client-server architecture and there is no need for a description of the service or a service registry as in

SOAP protocol, as the messages received from the resources in REST includes the metadata related to the received service. This makes the REST easier to change, where any changes on the server side require updating the WSDL to assure retrieving the services on the client side (Bloomberg, 2013).

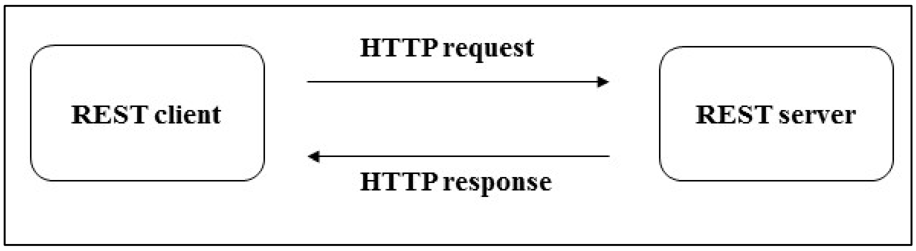
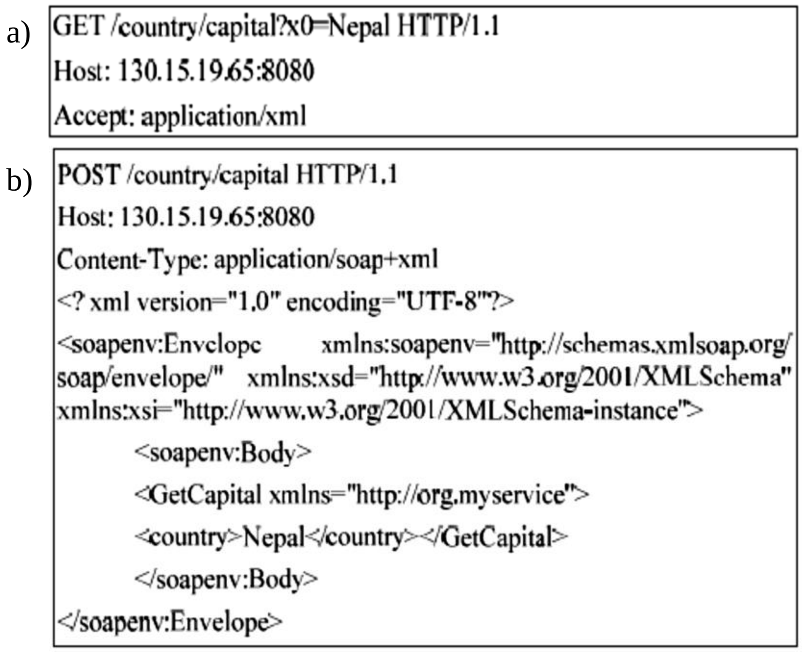


Figure 2-6 REST service invocation process

REST architecture is based on resources and representations. The resource is defined as anything identifiable and accessible using URI (Uniform Resource Identifier) which is used to represent each resource uniquely so the resource is not conflicted with other resources (Papazoglou, 2008). The representation is the data that describe a resource state. The representation data format is also known as media type such as HTML documents (R. T. Fielding, 2000).

REST services are accessed using HTTP methods through supporting CRUD (Create, Read, Update, and Delete) operations (Marinescu, 2013). HTTP messages can be either a request from a client to server or a response in the reverse direction. Each message contains an HTTP header fields and an HTTP message body (R. Fielding et al., 1999). The main HTTP methods that can be used as requests are: GET, this method requests service from the server; POST, sends data to be handled by the provider which results in either creating or updating the service; PUT, used to upload representation for the resource; DELETE, used to delete the resources; HEAD, is similar to a GET request by neglecting the response body, this is important for retrieving metadata without receiving the whole service. This request is used for retrieving the information required for the monitoring process in the proposed middleware. Finally, the OPTIONS method, which returns the HTTP methods supported by the server (Velte et al., 2009), OPTIONS is also used in the proposed middleware to transmit the SLA parameters' values without affecting the requested services. Chapter 4 presents more details about using HEAD and OPTIONS for transmitting the data in the proposed solution.

Figure 2-7 shows the request of services in both the REST and SOAP protocols It can be seen that the GET request in REST is implemented using POST in SOAP in addition to extra XML content being added in the body of the request to manage the communication between the client and the provider (Upadhyaya et al., 2011).


a)

```
GET /country/capital?x0=Nepal HTTP/1.1
Host: 130.15.19.65:8080
Accept: application/xml
```

b)

```
POST /country/capital HTTP/1.1
Host: 130.15.19.65:8080
Content-Type: application/soap+xml
<? xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope        xmlns:soapenv="http://schemas.xmlsoap.org/
soap/envelope/"    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <soapenv:Body>
        <GetCapital xmlns="http://org.myservice">
        <country>Nepal</country></GetCapital>
        </soapenv:Body>
</soapenv:Envelope>
```

Figure 2-7 An example depicts a request message: (a) REST; (b) SOAP web services (Upadhyaya et al, 2011)

REST is considered as one of the architectures that is used in cloud computing which provides a platform and language independent architecture style (Marinescu, 2013). Nowadays, there’s an apparent shift from SOAP services to REST services particularly in cloud computing platforms (Shroff, 2010) because of the advantages that REST offers such as simplicity, ease of use, better response time, and improved server scalability (Velte et al., 2009). Figure 2-8 shows the trend of Google search for the terms ‘SOAP API’ and ‘RESTful API’ between the years 2004 and 2017. The figure reveals a clear move from ‘SOAP API’ to ‘RESTful API’. This trend is due to the characteristics of REST which makes it more preferable in API industry developments (Google Trends, 2016).

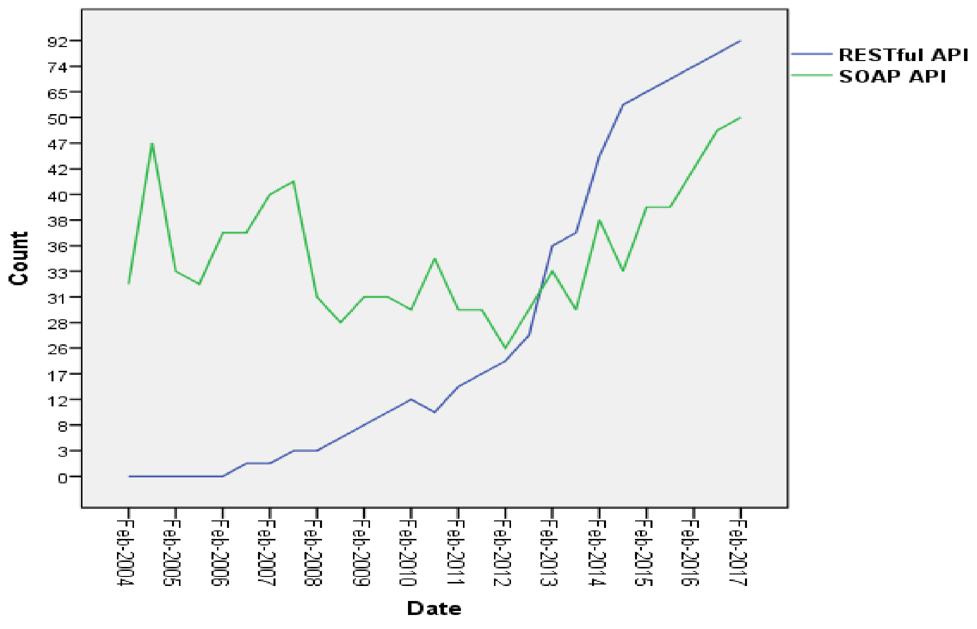


Figure 2-8 Google search trend for RESTful API and SOAP API from 2004 to 2017 (Google-Trends)

Because of the features of the REST architecture, it is mentioned by Kavis (2014) that REST is important in building cloud computing, as the cloud is dependent on a huge number of resources. REST architecture helps in handling these resources independently from the underlying infrastructure (Kavis, 2014).

REST services can be described using WADL (Web Application Description Language) and WSDL2, which provides an XML description for the functionality and the resources of the web services in REST architecture. However, WADL is not considered as important as the WSDL in SOAP web services due to the simplicity of the REST architecture (Richardson & Ruby, 2008).

Han et al. (2009) proposed the use of REST to manage cloud computing instead of SOAP protocol. The authors presented CMS, which is a Cloud Management System that exploits REST architecture in the managing process. The study suggested presenting the managed components as resources in REST. The proposed architecture is composed of a user interface to handle the user requests, a management module to handle the user requests, and the managed elements which are the resources components to be managed in the cloud environment. REST methods such as GET, PUT, POST, PUT, and DELETE were used to manage the resources (Han et al., 2009). The study is considered one of the first studies that considered using REST to manage the cloud, but it failed considering the SLA concept in the management process.

It is important to mention that REST and SOA share the feature of loose coupling, which facilities the development of distributed systems. This is applied in REST by using the uniform interface through the URIs. A good understanding of the REST concepts can assist in building high performance distributed systems (Vinoski, 2007).

2.4.2.3 REST vs SOAP

Several studies have revealed the difference between SOAP protocol and REST architecture, where the most mentioned point is the use of a WSDL as an XML technology to manage the communication between the client and the server side. This characteristic is the reason for the lightweight feature of the REST architecture, which provides better performance to the REST in comparison to SOAP. The researchers Mumbaikar and Padiya presented a comparison study between using SOAP and REST in multimedia-conferencing applications. They discussed the need for handling the XML SAOP messages which are not used in REST. The study showed that SOAP messages required more bandwidth and resource consumption. In conclusion, the researchers found that SOAP protocol added an end-to-end delay of 3 to 5 times the delay caused by REST architecture, and it added a network load 3 times of that caused by REST; their results highlighted the message size in both cases (Mumbaikar & Padiya, 2013).

Another study by Mulligan and Gra (2009) proposed a middleware for independence interaction using SOAP and REST technologies. They mentioned the concept of using resources in REST against the use of XML encoded messages, then, they showed the effect of this difference on the performance in terms of the packet size and the end-to-end delay, they concluded that the REST added less delay and packet size overhead in comparison to the SOAP protocol. Furthermore, the latency caused by synchronous requests in SOAP was higher than that of REST (Mulligan & Gra, 2009).

A similar study by Bora and Bezboruah (2015) compared REST services with SOAP protocol. The authors implemented two services for pharmacological data using SOAP and REST, and compared the performance and stability of the tested services. They argued for the use of client-server architecture in REST and accessing the resources using URIs against the use of an XML WSDL to manage the communication in SOAP. The results were that REST architecture outperforms the SOAP protocol (Bora & Bezboruah, 2015).

The work by Markey and Clynch (2013) studied the retrieval of database items using both SOAP and REST web services architectures. The study considered the sent and received bytes and showed that REST is the optimal selection for web services implementation, especially with using JSON formats against the use of XML in SOAP (Markey & Clynch, 2013).

The research by Mohamed and Wijesekera (2012) introduced a comparative study about using REST and SOAP based web services, and hosting these services in mobile devices. Again, the features of REST and SOAP revealed that more resources were required to parse the SOAP messages, which make REST services to be more efficient and require less resource consumption in comparison with SOAP services. This is due to the increase in the number of requests and the size of the transferred file (Mohamed & Wijesekera, 2012).

Finally, Upadhyaya et al. (2011) proposed converting SOAP based web services into REST services. The study discussed the points to be considered, which involves representing the operations included in the WSDL as resources to be retrieved using CRUD HTTP methods. The study evaluated both of the used services, they mentioned that REST is most popular and used on the web sites of the internet, providing a better performance (Upadhyaya et al., 2011).

2.4.3 Middleware

Middleware has been developed as a model for managing distributed systems, where the name ‘middleware’ indicates its location in the middle layer between the platform operating system and the application layer (Bernstein, 1996). Middleware can be defined as an infrastructure that supports the development and execution of distributed applications (Puder, 2006). Practically, middleware plays an essential role in building any distributed application (Gustavo et al., 2004). It is a software layer that resides between the application and the operating system to hide the distribution and heterogeneity of the underlying hardware details and programming paradigm (Krakowiak, 2007). The heterogeneity types that a middleware can hide and manage are discussed by Puder (2006), where the heterogeneity can be between different programming languages, different operating systems, or among computer architectures (Puder, 2006).

Five main technologies have been used in middleware implementations:

- Remote-Procedure-Call (RPC) middleware provides an abstraction to the interaction procedures, so that the application designer does not need to go through the details of these interactions when calling procedures on other devices.
- Transaction-oriented middleware (TP) supports transactions that guarantee consistent system transition through handling the errors caused wholly or in part by the application failure itself by using transactions to ensure saving the state of the invocations and avoiding the effect of the failure of one invocation on the others. The transaction ensures that a set of invocations within a transaction are guaranteed; this is considered by ACID (Atomicity, Consistency, Isolation, and Durability) characteristics.
- Message-Oriented Middleware (MOM) is the interaction managed by exchanging messages containing the request for service execution and the responses between the clients and the providers, where the message is a structured data involving the parameters of the message in a pair of name and value. This middleware supports the asynchronous communication through the usage of message queues in which the messages sent by the client are stored until the server is able to process them; these queues must be reliable in terms of failure to assure the delivery of the messages to the receiver.
- Distributed-Object Middleware (DOM) supports communication between distributed objects based on the object oriented concept, so instead of invoking procedures, it includes calling methods of objects on the remote machines. CORBA (Common-Object-Request-Broker-Architecture) is a kind of object-oriented architecture for managing applications through the internet (Gustavo et al., 2004).
- The fifth type of middleware is the Service Oriented Middleware (SOM), presents or uses the Service Oriented Architecture (SOA) technology which is a method for providing services to clients' applications or other services through a network. The services in SOA are platform independent and reusable so that they can be used by other software applications to fulfill tasks instead of building new services. In SOM the providers are loosely coupled to the clients (Qilin & Mintian, 2010).

This thesis considers SOM in the design of the proposed middleware (MonSLAR) because of its features of managing the heterogeneous environments in a loosely coupled way through supporting the SOC principles in deploying, managing, and monitoring the services. Chapter four presents more details about the proposed middleware.

In the scope of cloud computing, Marpaung et al. (2013) presented a survey of the middleware solutions available that are used to integrate different cloud applications and services or to manage deploying legacy applications into the cloud. They also mentioned the important role played by the middleware to manage the development of the cloud computing environment (Marpaung et al., 2013).

Many other types of research have been presented about developing a middleware to support different tasks in cloud computing. Brandic et al. (2010) presented C3, which is a middleware for ensuring compliance in cloud computing in terms of trust and privacy. This included defining a compliance agreement for the user requirements by extending the SLA document (Brandic et al., 2010). Although the authors discussed ensuring the privacy of data in the compliance agreement, they lacked the technical details of web services to transmit the data between the client and the provider.

SciCumulus is a middleware proposed by de Oliveria et al (2010) to manage the parallel execution of distributed scientific workflows in cloud computing through collecting the data and hiding the complexity and heterogeneity of the underlying resources of the cloud infrastructure. SciCumulus is composed of three layers, the first one is the desktop layer at the scientist's side; the second layer manages the distributed activities in the cloud, distributing the activities to the cloud instances; and the third layer is the execution layer in the instances of the cloud resources (de Oliveira et al., 2010). This middleware lacked the technical details of web services to transmit the data between the client and the provider.

Presenting a middleware for managing SaaS in cloud computing has been discussed by many researchers. For example, Decat et al. (2015) proposed Amusa, which is a middleware for managing the access of multi-tenants of SaaS to the applications based on their roles. The management considers the policies defined by the SaaS providers and the tenants. The architecture of the proposed middleware contains components for decision making which are based on the results obtained from authentication components, in addition to dashboards to allow the parties to manage the policies' attributes. (Decat et al., 2015) However, the proposed middleware failed to consider monitoring the SLA parameters violation or measuring the user's satisfaction.

Bansal et al. (2016) discussed the concept of middleware to manage multi-tenancy of cloud computing. The study introduced the main parties like the cloud hosts, the clients, and the network communication. The cloud host may contain a middleware component, the authors defined the function of the middleware as "a function that performs a conditional filtering operation or a conditional transformation operation on network traffic". The services provided by the middleware are maintained in a 'per-tenant' policy (Bansal et al., 2016).

Other attempts for developing middleware solutions dedicated to monitoring the cloud services are discussed in section 3.2 Cloud Monitoring.

2.4.3.1 Service Oriented Middleware (SOM)

SOM is a middleware that supports the aggregation of services to manage and develop service oriented applications and based on Service Oriented Computing (SOC). SOC supports the loose coupling services through considering aggregating the components in a network of services which allow the development of distributed applications with high interoperability, code reusability, independence of underlying operating systems and programming languages, in addition to reducing the cost. SOC is the base of web services as they provide services on the internet using standard protocols to support the idea of SaaS. Thus, developing a middleware with SOC capabilities can facilitate the design of service oriented systems, by making software functionalities available as services to the clients to be used later in any design without the need to rebuild these functionalities. These services can be built using different technologies such as SOAP and REST web services. Because of its aforementioned features, SOM is now considered to be the preferred middleware by developers and researchers (Al-Jaroodi & Mohamed, 2012).

Several studies investigating SOM have been presented. Al-Jaroodi and Mohamed (2012) submitted a survey about the available service oriented middleware solutions. This study also discussed the main challenges and requirements for implementing this kind of middleware. This survey emphasized the importance of middleware as a solution to SOC through combining the advantages of both of these technologies. Besides this, to meet the functional and non-functional requirements of the delivered services, they also mentioned that there should be a balance between achieving these requirements and reducing the overall overhead of the system.

Issarny et al. (2011) presented a study about SOM for the future of the internet. They discussed the challenges of the internet's future in terms of scalability, heterogeneity, mobility, awareness and security. They proposed a SOM as a solution for this kind of network which manages connecting services to the clients (Issarny et al., 2011).

Lee et al. (2005) proposed using an SOA to implementing a middleware, which adds a service transparency feature and manages composing them to get complex services from different service providers. The authors suggested using the XML as a technology for managing the WSDL in the middleware through coordinating the connections among the clients, service providers and the brokers (Y.-C. Lee et al., 2005). Although that the study considered designing a middleware taking into consideration the SOA concept, it failed supporting REST web services.

Wohlstadter et al. (2006) proposed Cumulus, which is an SOM for managing interoperability of web services. Cumulus architecture is composed of a client-side, a services' registry and remote middleware services. The middleware allows run time interoperability through attaching the client policies to the Business Process Execution Language (BPEL) and attaching the provider policies to the WSDL. This enables the middleware to select the middleware service from the registry (Wohlstadter et al., 2006). This middleware's functionality was dedicated to the interoperability of web services with the use of WSDL, but it is impractical for cloud solutions or monitors user's services.

The importance of using Service-Oriented-Architecture (SOA) to manage cloud computing has been discussed by many researchers, through creating, organizing, and reusing cloud components. SOCCA was proposed by Tsai et al. (2010), which is an SOA architecture to allow the interoperation among different clouds. The researchers mentioned the importance of using SOA to manage the SLA through separating the roles of the service providers and the cloud provider. The researchers used the WSDL to manage the web services in their implemented prototype (Tsai et al., 2010).

Another SOA middleware was developed by Azeez et al. (2010) to support multi tenancy in cloud computing (Azeez et al., 2010). SOA middleware was proposed by Azeez et al (2010) to manage the multi tenancy in cloud computing. The proposed solution presented to allow sharing the infrastructure of cloud resources among many users in addition to assuring isolating the data of each user. This architecture exploited SOA to present services to the clients in the form of processes, data and security services. This middleware was developed on top of WSO2 Carbon, which is an open source middleware for building scalable servers (Azeez et al., 2010). Torkashvan et al. (2012) proposed using service oriented in cloud computing to add an intelligence as a service layer which allows automating the services in the cloud. The users can get the business or hardware services in an even-driven base (Torkashvan & Haghighi, 2012b). Yang et al. (2015) submitted a study about the use of SOA concept in managing a middleware in cloud computing. The researchers discussed the SOA characteristics as well as mentioning the importance of loose coupling and the reuse of software components in a cloud environment. Although this study presented SOA architecture for the cloud middleware, it did not handle the monitoring of services and the implementation details of the proposed architecture (Yang et al., 2015).

On the other hand, MonSLAR utilizes SOA to provide Monitoring as a service through separating these services from the SaaS providers.

There is different research related to the subject of middleware, however, this thesis focuses on the middleware used for monitoring cloud computing and web services.

2.5 Chapter Summary

This chapter presented an overview of the main technologies related to the subject of the thesis. The chapter is organized into two main directions. The first direction, which is an overall introduction to cloud computing with a focus on the characteristics of SaaS as the main topic of the presented thesis. The second direction, introduced the main technologies used to manage cloud computing. SOA presented as an architecture for supporting services as the main components of the information system. Web services were also introduced as the basis of SaaS applications, with a focus on REST as the technology used to manage the communication in the proposed solution. Finally, the third technology introduced managing the services in a cloud environment by presenting the middleware in addition to an overview of its types, elaborating on the SOM as the middleware type used in this research. In the next chapter, the literature review related to monitoring cloud computing is presented.

CHAPTER THREE

MONITORING USER SATISFACTION IN CLOUD COMPUTING

3.1 Introduction

The monitoring of cloud services from a user's perspective has been the focus of much research. Whereas the SLA is the contract that governs the relationship between the user and the service provider, monitoring the SLA compliance is vital to ensure the user satisfaction with the received services in cloud computing. As a result, the expediency of developing a middleware for monitoring SaaS services in cloud computing can be proved by reviewing the literature related to this subject and by highlighting the lack of reference to it in previous researches. This chapter presents the literature review related to the research topic. Figure 3-1 which summarises the influences from the literature review on the subject of monitoring user satisfaction in cloud computing and the rationale for the proceeding arguments.

This chapter discusses existing research related to monitoring user satisfaction in cloud computing. This presentation of the literature review helps in to clarify the available methods and frameworks employed in monitoring cloud systems in order to overcome their shortcomings in the research.

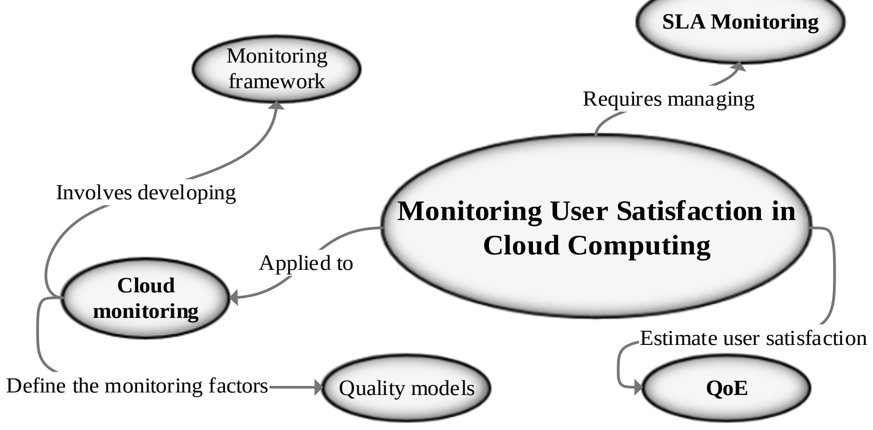


Figure 3-1 mind map shows the influence of the literature review

3.2 Cloud Monitoring

As cloud computing is the delivery of services based on a prescription, it is important to check the delivery of the expected services to the client. Continuous monitoring of cloud computing and the SLA is very important for both cloud providers and clients. It can provide data about the size of the induced workload which helps in identifying ways to prevent violating an SLA's terms (Aceto et al., 2013). This section presents the main studies concerned with monitoring cloud computing.

Lampesberger and Rady (2015) presented a study about the importance of monitoring the client interaction in the cloud. The authors studied the difficulty faced by the client in monitoring the cloud system as a black box, and introduced the monitoring as a method to ensure receiving the required services. They discussed measuring the levels in the service level agreement as an important issue in monitoring cloud computing. The study focused on monitoring XML messages to detect intrusions in the web services (Lampesberger & Rady, 2015). Although the researchers discussed the possible monitoring points in the cloud to be in the hardware, network, middleware, and user level; but they overlooked discussing the use of REST architecture as one of the web service technologies, and measuring the QoE as an instrument for user satisfaction.

Many surveys have been conducted to study the monitoring of cloud computing. The study submitted by Aceto et al., (2013) presented such a survey, the authors highlighted the importance of continuously monitoring the cloud services in order to prevent breaches of SLA through estimating the clients' perceived services, they also discussed the issues related to cloud monitoring and showed that one of the main issues is the importance of notifying the monitored data timely, and provide automatic management for the monitoring process (Aceto et al., 2013).

On the other hand, Gao et al. (2013) discussed the challenges of testing SaaS, and mentioned the importance of defining new standards for testing SaaS. They defined 'testing SaaS' as "different types of validation activities in a test process to assure the quality of SaaS in delivering the specified on-demand function services on a cloud infrastructure". The authors highlighted the importance of considering the multi-tenancy characteristic of SaaS in SaaS monitoring, and the need for considering the QoS of SLA parameters, in addition to checking the usability of the user interface (Gao et al., 2013).

Rehman et al. (2015) presented a state of the art assessment of the available management systems in cloud computing. The authors discussed the main challenges for cloud management, such as the lack of reference in previous research to monitoring cloud services from the user's perspective, the lack in using multi-criteria in estimating the QoS levels in the cloud, in addition to the lack of an automated early warning system for the users altering them for any degradation in QoS (Rehman et al., 2015). These challenges are tackled in the current thesis through presenting MonSLAR with a user-centric monitoring and a dashboard for monitoring QoE.

Incki et al. (2012) presented a survey about testing the software in cloud computing. The authors mentioned the importance of considering an automated monitoring for the cloud environment. They also highlighted the weakness in the research field to provide acceptance tests for the cloud services, where these tests check the achievements of contracts (Incki et al., 2012).

Another survey was submitted by Da Cunha Rodrigues et al. (2016) about cloud monitoring. The study reviewed the available commercial monitoring tools, and mentioned that they are owned by specific cloud providers. The researchers discussed the importance of monitoring cloud computing to manage its resources, besides the need for monitoring the SLA as it is considered one of the main features of cloud computing. They also emphasised considering the cloud model whether it is IaaS, PaaS, or SaaS in the monitoring process; in addition to considering whether the monitored data is from the client or the provider point of view (Da Cunha Rodrigues et al., 2016).

These surveys illustrate that monitoring a user's perception with received services is still an unsolved issue. This issue is solved in this thesis through monitoring the SLA compliance, which is considered as an indicator of the user's satisfaction about received services. A middleware is proposed to manage the monitoring process.

The literature review relating to cloud monitoring is categorised into two different subsections, monitoring frameworks and quality models, as shown in the next two subsections.

3.2.1 Monitoring Frameworks

In recent years, there has been an increasing number of monitoring tools to track cloud resources in both scientific academic research and commercial fields. In this section, a review of the literature related to the monitoring frameworks is presented, with a focus on academic tools, these frameworks are compared in section 3.5. In spite of the availability of variant commercial monitoring tools like CloudWatch from Amazon and AzureWatch from Microsoft Azure SDK (Aceto et al., 2013), they are owned by specific cloud providers, which means that they provide monitoring to those cloud providers and from the providers' perspective (Da Cunha Rodrigues et al., 2016). The presented literature is classified according to the mode of monitoring, and divided into three directions, server-centric, user-centric, and third party monitoring frameworks.

3.2.1.1 Server Centric Monitoring

The first direction of the literature relates to Server-centric monitoring. A considerable amount of research has been published on monitoring frameworks in cloud computing. However, most of the research to date has tended to focus on monitoring the provider side of the cloud. The study presented by Shao and Wang (2011) introduced a monitoring framework for measuring the performance of cloud applications, in order to manage the provision of resources according to performance measurements. The authors proposed using two metrics to measure the performance of the cloud applications: availability and response time (Shao & Wang, 2011). The main limitations of this study are that it failed to take into consideration the client side of the monitoring process, notifications to the user about violations of SLA, or the kind of web services supported by the framework.

Another server-centric monitoring framework is M4CLOUD, presented by Mastelic et al., (2012) to monitor applications in shared resources cloud environments. Taking into consideration the SLA parameters, the monitored data is used to manage the allocation of cloud resources; the researchers submitted an approach to classify the application metrics. This research focused on monitoring the resources of the cloud application, and managing the communication between the agents responsible for collecting the monitored metrics (Mastelic et al., 2012), but it failed either to present a way of delivering the monitored data to the client, or of estimating the overall user satisfaction regarding received services.

Another monitoring framework (JCatascopia) by Trihinas et al. (2014) claimed to provide automatic monitoring for applications in cloud computing, to manage the allocation of cloud resources. The proposed architecture considered collecting the metrics required for the monitoring process from various layers of the cloud. The researchers also focused on the collection of metrics from underlying resources, in addition to the provision of access

to the monitored data using a REST API (Trihinas et al., 2014), but they failed to manage the communication between the client side and the provider side, to provide a user-centric management for the monitoring process, or to notify the client about his/her overall satisfaction about the received cloud services.
None of the above mentioned frameworks provides details about supporting REST architecture or the kind of web services used for transmitting data between the provider and the client. There is also a limitation in that they ignored the client side monitoring.

Povedano-Molina et al. (2013) presented DARGOS, which is an architecture for monitoring multi-tenant cloud computing. They mentioned the importance of adapting lightweight monitoring processes and communications to avoid any additional overhead in the cloud environment. They also mentioned that avoiding using the broker ensures more reliability and robustness as the failure of the broker may affect the whole system (Povedano-Molina et al., 2013). This research focused on collecting the monitored data from the cloud resources using a distributed architecture for monitoring cloud computing. Although a REST API is used, it failed to consider user-centric monitoring for cloud services, and the collected results provide no indication of the overall user satisfaction.

Another middleware has been presented by Cedillo et al. (2015) to monitor SLA compliance in cloud computing and provides reports containing SLA violations. The monitoring process includes measuring the cloud resources and compares their quality with levels specified in the SLA document. The proposed framework consists of two main components, the configurator, to derive a quality model based on the quality requirements; and the monitoring and analysis, which compares the measurements with the requirements to decide the SLA violations (Cedillo et al., 2015). This study was extended by Cedillo et al. (2016) to present a monitoring framework to assess the quality of cloud services. Additional non-functional requirements that are not part of the SLA were taken into account, also the quality of SaaS (Cedillo et al., 2016). Although the middleware presented in the two aforementioned studies comprises an attempt at monitoring SLA parameters violations in cloud computing, they did not present a user-centric control for the monitoring of an SLA. They also fail to mention the supported types of web services in delivering the monitored data.

Smit et al. (2013) introduced MISURE, an architecture which uses streams for monitoring cloud applications in heterogeneous environments. The authors focused on aggregating and collecting the measurements from the cloud resources, and proposed presenting the monitored data as a web service, or in other words Monitoring as a Service (MaaS). The measurements can be retrieved using REST APIs, which can be used later to deliver notifications to the client side (Smit et al., 2013). Although MISURE takes into consideration monitoring the cloud computing resources and collecting the measurements, it does not tackle the problem of providing a measure of user satisfaction based on an SLA, and making a decision based on multi criteria; an adaption is required for this architecture to consider an overall measurement for cloud services.

JTangCMS is another monitoring framework presented by Lu et al., (2016). The researchers mentioned the importance of using a decision making system to handle the collected monitored data. They also mentioned the importance of reducing overheads generated by the monitoring process, which is countered in their study by controlling the frequency of collection of the monitored data (Lu et al., 2016). The middleware employed is a message queue, and the delivery model of data in this research is a push-pull model. Although this research considered a monitoring cloud platform and presenting decision making, it failed to introduce user-centric monitoring for cloud services and to deliver the required data to the user side.

Another middleware by Lee et al., (2012) provided an SOA for an enterprise cloud computing, and proposed a middleware to provide monitoring for the cloud which helps in checking the performance of the system to provide requested services in order to manage resource allocation processes, taking into consideration an SLA document compliance level. The proposed middleware considered the use of a service description language which is an expansion of a WSDL of web services, however, this middleware neither presented user-centric monitoring for cloud services nor discussed how to manage REST services in the monitoring process. Moreover, the middleware was theoretically presented, but without giving any details about its performance evaluation (S.-Y. Lee et al., 2012).

Muller et al. (2012) presented SALMonADA, which is a framework for monitoring the QoS of service based systems. This framework is capable of monitoring the services' QoS and provides violation reports in terms of SLF (Self-Level-Fulfilment), taking into consideration the SLA document (Müller et al., 2012). SOAP protocol was used for invoking the Monitoring-Management-Documents, in addition to WSDL-documents to manage the notifications. This study was extended by Oriol et al. (2015) by proposing SALMon, a monitor for the service based systems through the whole SLA lifecycle. Again, while this study employed a WSDL to transmit and represent the monitored data and reporting the violations to the client, it also failed to present a comprehensive evaluation of user satisfaction (Oriol et al., 2015). However, the delivery of the monitored data is managed in MonSLAR by using REST architecture.

Perez-Espinoza et al. (2015) presented a distributed monitoring architecture for private clouds. The architecture is composed of the following components, a collector to collect the monitored data from the underlying physical and virtual resources; metasensors as the monitoring tools used for monitoring the resources; a distributer to manage the monitored data collected by the collectors; and a visualizer, which is used to allow the users check the monitored data. The collector contains a classifier to check the workload and alerts in case of critical high workloads cases (Perez-Espinoza et al., 2015). The study focused on collecting the measurements from the cloud resources, but it failed in providing an automated user-centric monitoring, in addition to overlooking the SLA document in determining the user satisfaction and SLA violations.

3.2.1.2 User Centric Monitoring

The second direction in this section is the user-centric monitoring frameworks, in which the monitoring process is managed by the client side of the cloud environment. Emeakaroha et al. (2012) proposed CASViD, which is an architecture to monitor and detect the violations in the SLAs of cloud computing applications, through monitoring performance and usage the cloud resources according to the levels specified in the SLA document (Vincent C Emeakaroha et al., 2012). The framework is considered to be user-centric monitoring, as the SLA management is activated by client side requests. This research was extended by Brandic et al. (2015) through proposing an algorithm for determining the intervals between measurements of the applications in multi-tenancy SaaS, this was achieved by considering the cost and the SLA objectives (Brandic et al., 2015). Although violations of the SLA were detected in the two aforementioned studies, but the framework failed to define a way for notifying the client about these violations, and also to declare the web services used for transmitting the data between the client side and the provider side.

GMonE, is a framework developed by Montes et al. (2013) for monitoring cloud environments. The authors discussed the importance of considering user-centric monitoring for cloud services, in addition to the QoS of the SLA parameters (Montes et al., 2013). Although the authors claimed that GMonE provides client-oriented monitoring for the cloud and a GUI access to the monitored data, this was considered in terms of the type of measurements and collection of the data required for the client. However, they provided no information about managing the monitoring process by the client, supporting the REST architecture, or monitoring the overall user satisfaction.

Nguyen et al. (2014) presented a user-oriented monitoring framework for cloud computing. The authors highlighted the importance of distinguishing the role of the cloud user as a consumer of cloud resources, whether it is a client or a provider in the monitoring which affects the required monitored data. They also discussed the fact that the cloud application user is more interested in receiving clear notifications about the decline of a service than the metrics details of used cloud resources. Although the researchers claimed that it is a user-centric monitor, managing the monitoring was accomplished using a trusted third party (Nguyen et al., 2014). The main limitation of this study is the failure to consider SLA compliance in monitoring cloud services.

Serhani et al. (2014) presented a study to check SLA violations in SaaS cloud computing, through measuring the QoS of the received services. The researchers discussed the importance of monitoring the SaaS services for both the cloud clients and providers, claiming that the framework can be both client- or provider-centric (Serhani et al., 2014). However, this study failed to manage an automated monitoring for the SaaS services, and there are no details about the types of web services supported. On the other hand, MonSLAR manages the monitored data for a specific REST service provider, the architecture of MonSLAR also provides more details for supporting the REST architecture in handling the monitored data.

Rehman et al. (2015) presented UCSM, a framework that assists the user in the cloud service selection process. The framework contains monitoring and early warning components. The monitoring process took into consideration collection of the QoS measurements of the cloud services and the users' feedback about the services, to be used later by the other components (Rehman et al., 2015). Although this study considered monitoring the cloud services by considering the users' feedback, it ignored the stated

services' levels in the SLA. Furthermore, this framework lacked an overall estimation for user satisfaction (QoE) based on selected parameters, and provided a poor description for technical details such as the communication mechanisms and the types of supported web services.
Moustafa et al. (2015) presented SLAM, an agent-based framework for monitoring SLA in federated cloud computing. SLAM proposed allowing the user to measure the SLA parameter through mapping it with low level metric, and uses dashboards to provide the monitored data. SLAM allowed the client to evaluate the cloud provider service. This was done by the coordinator component which sends requests to the specific provider and evaluates its performance according to the collected data. The authors claimed that this framework can be used by both the clients and the providers to monitor the cloud services (Moustafa et al., 2015). However, the research failed to provide an automated online monitoring for the cloud service, as well as consideration of the type of web services used in the cloud service, and handling of the REST architecture. This research is considered in Appendix E, through comparing the overhead caused by SLAM with that caused by MonSLAR, as it is the most relevant one to the middleware presented in this thesis.

Another architecture was proposed by Tang et al. (2016) to assess trust in cloud computing based on QoS monitoring and users' feedback. The authors presented a middleware to manage the evaluation process. In their study, trust considered as the expectation of the user about the used service. The architecture proposed providing a list of trusted services to the clients based on the middleware evaluation results and each client SLA requirements, which could help the user in selecting the most trusted service (Tang et al., 2016). Although users' feedback was considered to evaluate the candidate services, the study did not consider measuring the individual user satisfaction of the used service. Little attention has been given to provide an automated monitoring in the proposed architecture and delivering the data to the clients.

The main limitation of the research presented in this kind of monitoring frameworks is the weakness of finding an automated monitoring environment to control SLA violations cases or giving details about the web services used in the monitoring process.

3.2.1.3 Third Party Monitoring

The third direction of this section is third party monitoring. There has been some research proposing using a third party component for monitoring frameworks to manage the monitoring process. For example, Siebenhaar et al. (2013) proposed a monitoring framework that enables estimating the availability of applications in cloud environments from a user's perspective, taking into consideration the measurements on both the client and the provider side. The monitoring process was managed by using a broker, who is in charge of coordinating the monitoring and collecting the data from both the provider and the client side, and measuring the overall availability of the system by considering the availability value in the SLA document (Siebenhaar et al., 2013). The main weakness of this study is the use of a broker in managing the monitoring of the cloud. It also fails to monitor overall user satisfaction about the cloud by focusing on only one parameter (availability).

Katsaros et al. (2011) presented an architecture to manage the monitoring process in cloud computing. The researchers discussed the advantages of REST over SOAP in managing SOA, and proposed using REST to manage the collection of the monitored data from the underlying resources of the cloud in the monitoring process. The monitored data is collected by NEB2REST, which is a developed module as a broker to manage invoking the monitored data, where the monitoring information is provided as the REST resources. (Katsaros et al., 2011). Although this study presented REST to manage the monitoring process, but it failed to present an automated user-centric monitoring for the data, it also failed to present an approach for estimating the user satisfaction or the SLA violation.

Rak et al. (2011) proposed a monitoring framework for cloud computing applications that support mOSAIC components. The proposed framework provides cautions in the case of violating the SLA document through monitoring the quality of cloud resources. An agency was suggested to provide the collected monitored data (Rak et al., 2011). However, this framework is specific to monitoring mOSAIC applications and is not applicable to other applications in cloud computing.

Another framework had been proposed by Ye et al. (2012) in order to verify and detect SLA violations, achieved by presenting a third-party-auditor to check an SLA, which is assumed to be trusted in this study (Ye et al., 2012). In spite of highlighting the importance of splitting the monitoring of the SLA between cloud providers and clients to ensure a trusted relationship, this has been managed by a third party which can at times be considered unreliable. This issue is treated in MonSLAR by managing the monitoring process automatically by the proposed middleware.

Alsulaiman and Alturki (2012) presented a model for SLA monitoring and evaluating the services offered by cloud providers. The study focused on monitoring the QoS of multimedia in the cloud. The researchers mentioned the importance of isolating the monitoring process from a cloud provider to ensure client trust, suggesting the use of an embedded agent independent of the provider side to accomplish the monitoring of the SLA (Alsulaiman & Alturki, 2012). They overlooked notifying the client about violations or discussing the supported types of web services.

Cicotti et al. (2012) presented QoS-MoNaaS, a Quality-Monitoring-as-a-service that provides a solution for evaluating the quality of cloud services and identifying any SLA breaches in cloud computing. The authors suggested presenting the monitored quality to users as a service using a trusted third party (Cicotti et al., 2012). This study was extended by Adinolfi et al. (2012) to consider portable monitoring for the QoS, through making it possible to be ported to different cloud platforms and providing continuous monitoring for QoS (Adinolfi et al., 2012). However, neither of the aforementioned studies considered providing any notifications of violations of an SLA, nor did they consider user-centric monitoring for cloud services.

The study presented by Amato et al. (2012) discussed evaluating an SLA in cloud computing through using a third party broker. This proposal was claimed by the authors to help in selecting providers with best offers of resources in the negotiation time and selection of the provider (Amato et al., 2012). In spite of considering evaluating cloud services in order to manage the relationship between the provider and the client, it focused mainly on evaluating provider services in the negotiation phase and overlooked monitoring the SLA state after service usage, and in addition there was the weakness caused by the use of a broker to manage this process in the form of the increased costs and overheads.

Badidi (2013) proposed a framework for managing an SLA in SaaS. The author mentioned the importance of monitoring the QoS of the provided services in the selection of cloud providers and allocation of the cloud resources, and of ensuring SLA compliance on receipt of services. The management of SLA and monitoring QoS of the web services are achieved by the use of a cloud broker (Badidi, 2013). Although this study discussed the delivery of SaaS as web services and monitoring its QoS, it failed to consider the different kinds of web services and the technical issues of the monitoring process.

Khaddaj et al. (2014) proposed a framework to assure SLA compliance in cloud computing by monitoring the QoS. A broker was suggested to manage the cloud resources allocation process according to the values of QoS defined in the SLA (Khaddaj et al., 2014). Although it is claimed by the authors that the framework introduces a kind of user-centric management for the SLA matching process, the monitoring and the measurement process was managed by the broker, in contrary to MonSLAR, where the user-centric monitoring is controlled by the client side of the middleware.

Aversa et al. (2015) proposed an architecture for monitoring cloud applications taking into consideration the non-functional requirements and the performance of cloud resources. The authors proposed agent-based monitoring to collect the monitored data from cloud resources. The monitoring architecture considers monitoring resources with requirements restrictions to decide the monitoring configurations that are used later in the decision making process, in order to ensure SLA compliance of the provided services (Aversa et al., 2015). Although this research monitors cloud applications based on performance requirements, it lacks considering a user-centric monitoring for the overall satisfaction about the cloud services.

Measuring user satisfaction with cloud services was considered by Hammadi and Hussain (2012) through measuring the provider’s reputation to decide usage continuation. They proposed a framework for monitoring SLA in cloud computing in terms of trust and risk. The study emphasised the importance of monitoring the QoS of cloud services taking into account the SLA. A third party monitor was suggested that used fuzzy logic for measuring provider reputation, assisting in decision making about the continuous use of the service. In their research, fuzzy logic used to measure the reputation based on the recommending user opinions, who decide reputation according to experience with the provider; time delay; and their credibility in the pre-interaction phase; the input to the fuzzy system is not the SLA parameter values (Hammadi & Hussain, 2012). In MonSLAR, measuring the SLA assurance has been used to monitor the provided services through checking the measured SLA parameter values and assessing the services, eliminating the need for a third party; this method is considered more reliable as will be explained in chapter 4. The research presented in this thesis is dedicated to monitoring SaaS services.

You et al. (2015) proposed a framework for SLA management which considers providing an SLM-as-a-service in cloud computing, which was presented by Motta et al. (2014).

This study suggested the use of a third party in the management process of an SLA. The framework includes an agent in charge of monitoring the status of the cloud services in terms of the QoS of the SLA. Although the authors mentioned the use of XML and UDDI in the management process, they failed to give details about supporting REST services, or to consider a user-centric monitoring process (You et al., 2015).

Regarding the literature review related to third party direction, it is important to mention that the use of a broker is considered as a weakness point, because of additional overheads, cost, and reliability issues. These issues can be eliminated by avoiding the use of a broker and defining a new approach to managing the monitoring of the cloud environment, which is handled in this thesis by proposing a client-server monitoring middleware.

3.2.2 Quality Models

Definition of quality models has been investigated by many researchers as an indicator of quality, many years before the commencement of the cloud computing concept. This section introduces the various models which are in existence, and a comparison among these models is presented in section 3.5. The quality model (SERVQUAL), which was presented by Parasuraman et al. (1988) identified quality gaps in the consideration of service quality. However, this model was developed in the late eighties and at the time, cloud computing was not established as a possible source of service gap. Whilst this model is therefore useful as an indicator of service level gaps for more traditional service companies, it is not relevant to the development of SLAs in cloud computing. Five quality dimensions were defined in this model which are: reliability, assurance, responsiveness, tangibles, and empathy (Parasuraman et al., 1988).

Defining a quality model for cloud computing services has been investigated by many researchers. The Service Measurement Index (SMI) model is the first and most generally applicable in this field, which comprises a set of Key Performance Indictors (KPIs) identified by the Cloud Services Measurement Initiative-Consortium (CSMIC) to offer a comparative evaluation for the cloud services, taking into consideration the main QoS requirements of the cloud user. This model is a standard that helps organisations in measuring the cloud services based on their demands (CSMIC, 2011).

A similar model was proposed by Lee et al. (2009) to evaluate SaaS in cloud computing according to the features of SaaS. This model was defined taking into consideration the main features of SaaS which are: reusability, availability, scalability, reliability, efficiency, and assessment (J. Y. Lee et al., 2009). However, the main drawback of this research is the weakness in determining the quality dimensions which were heavily dependent on the literature. This study would have been more original if the authors had considered user opinion in the decision process.

CLOUDQUAL is another quality model proposed by Zheng et al. (2013) for cloud computing services taking into consideration the QoS. The authors proposed six dimensions for the proposed quality model which are: usability, availability, reliability, responsiveness, security, and elasticity. Testing this model considered the cloud storage services (Zheng, 2013). However, this study failed to determine the weight of the effect of each of the quality dimensions on the overall user satisfaction, which is required in MonSLAR to estimate the QoE value in the decision making process of the proposed fuzzy logic engine.

Furthermore, SaaS-Qual was presented by Benlian et al. (2011), which is another quality model used as an instrument to expect the continuity of using SaaS by the customers. This model represents the most appropriate model to be used in predicting QoE in SaaS cloud computing. The significance of SaaS-Qual model emerges from the need in this research to define the main factors that affect the user satisfaction with the received services, where these factors are proposed in SaaS-Qual. In addition to the fact that specifying these factors were based on SaaS experts which make them more realistic and suitable for the proposed solution in this thesis.
The next subsection presents more details about SaaS-Qual model, which is the used quality model in this thesis.

3.2.2.1 SaaS-Qual Model

SaaS-Qual was presented by Benlian et al. (2011) as an instrument to predict the continuity of using SaaS by the customers. The authors specified six metrics to measure SaaS, which are rapport, responsiveness, reliability, flexibility, features, and security. These factors are outlined here:

a) Responsiveness: Consists of all aspects of the SaaS provider’s ability to ensure an availability and a performance of a SaaS-delivered application (e.g., through professional disaster recovery planning or load balancing) as well as the responsiveness of support staff (e.g., 24-7 hotline support availability) being guaranteed.
b) Reliability: Comprises all features of the SaaS vendor’s ability to perform the promised services timely, dependably, and accurately (e.g., providing services at the promised time, provision of error-free services).
c) Flexibility: Covers the degrees of freedom customers have to change contractual (e.g., cancellation period, payment model) or functional/technical (e.g., scalability, interoperability, or modularity of the application) aspects in the relationship with the SaaS vendor.
d) Security: Includes all aspects to ensure that regular (preventive) measures (e.g., regular security audits, usage of encryption, or antivirus technology) are taken to avoid unintentional data breaches or corruption (e.g., through loss, theft, or intrusions).
e) Features: Refers to the degree the key functionalities (e.g., data extraction, reporting, or configuration features) and design features (e.g., user interface) of the SaaS application meet the business requirements of a customer.
f) Rapport: Includes all aspects of the SaaS provider’s ability to provide knowledgeable, caring, and courteous support (e.g., joint problem solving or aligned working styles) as well as individualized attention (e.g., support tailored to individual needs).

These factors have been used as a base in monitoring SaaS cloud services. Deciding the quality dimension was based on a comprehensive study which relied on a literature review, in addition to interviews with experts in SaaS who were account managers in SaaS companies, focus groups with information system managers, in addition to a pilot study (Benlian et al., 2011). Their study defined the effect of each quality dimension on the overall user satisfaction of the services as shown in Figure 3-2, which helps with the decision making in the fuzzy logic engine of MonSLAR (see section 5.3). For these reasons and the fact that it is one of the most cited quality models, and considering that determining the factors was based on SaaS experts, SaaS-Qual has been considered in estimating the QoE value in this thesis.

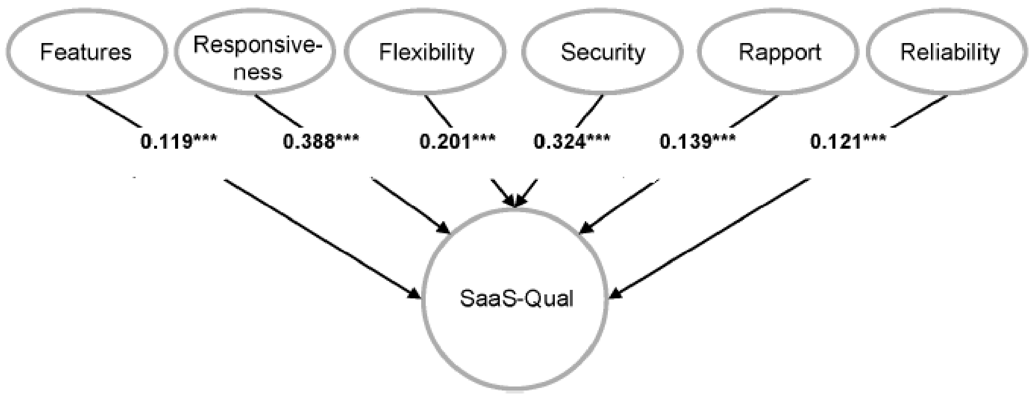


Figure 3-2 weighted combination of the six factors of SaaS-Qual (Benlian et al., 2011)

3.3 QoE Monitoring

QoE was defined by the European Network on Quality of Experience in Multimedia Systems and Services as “the degree of delight or annoyance of the user of an application or service”, which is affected by achieving the results expected from using a service taking into consideration a user’s feeling and thinking of a specific situation (Le Callet et al., 2012). QoE represents a measure of the level of received service (Matulin & Mrvelj, 2013). It can also be defined as the network performance from the user’s perspective when using the service, while the Quality of Service (QoS) relates to the factors that affect the quality of the network while transmitting the data in terms of packet loss and bandwidth, but without considering the user’s point of view (Rifai et al., 2011).

According to Varela et al. (2014), “QoS is defined from a system’s perspective - characteristics of a telecommunications service-, whereas QoE is entirely defined from user’s perspective – degree of delight or annoyance of a person”. The authors stated that QoS is a different concept from QoE, while they interfere as QoE depend on QoS, but QoE provides better perception for the network performance. The study asserted that QoE includes both technical and psychological features; it also discussed that QoE measurement can be used for network management in terms of SLA management (Varela et al., 2014).

Two basic approaches are available to measure the QoE value in networks. The first approach considers using subjective or qualitative tests that can be interpreted using human understanding, which are performed by real customers in a test panel. However, this approach is considered time-consuming and expensive, as it requires a large number of people to conduct the test and get the result, in addition to the varying views of the users, which affect the accuracy of a result. The users of a service are asked to fill in a questionnaire; Mean Opinion Score (MOS) is used to determine the level of the QoE.

The second approach considers using objective tests; this approach is based on the use of measurable parameters to assign quantitative values to the performance that a customer perceives; the objective test is used to acquire values comparable to those of the real users of the services. Five models are available for the objective measurements of QoE: **media layer**, the input in this model is the media signal; **packet layer**, the data used here is the data from the IP header only; **bistream** layer, where the input is both the encoded bistream information and the packet header; **planning**, “includes the quality planning parameters of networks or terminals”; and the **hybrid model**, which is a combination of the previous models. According to Matulin and Mrvelj (2011), the hybrid models use both quantitative and qualitative inputs for the evaluation of QoE (Matulin & Mrvelj, 2013) (Fiedler et al., 2010; Rifai et al., 2011). A hybrid-objective model is used in this thesis.

Estimating the value of QoE has been the subject of many researchers for many years. The IQX-hypothesis is a model developed by Fiedler et al. (2010) to define an exponential relationship between QoS and QoE. The proposed relationship is shown in Figure 3-3, where three different regions were defined, taking into consideration two threshold values (x1) and (x2). In the first region (QoS-disturbance <= (x1)), represents the optimal service perceived by the client, because of the small value of QoS-disturbance; in the second region ((x1) <= QoS-disturbance >= (x2)), the increase of the disturbance in QoS causes a degradation in the QoE value; and in the third region (QoS-disturbance >= (x2)), the clients perceive a low quality service, which represents a low value of QoE (Fiedler et al., 2010). This model can be the base for an objective estimation of the QoE value, however, it does not consider the cloud services, and hence the SLA is not considered in the estimation process of QoE. This model is adapted to present a metric for estimating QoE in SaaS services (See Chapter Five).

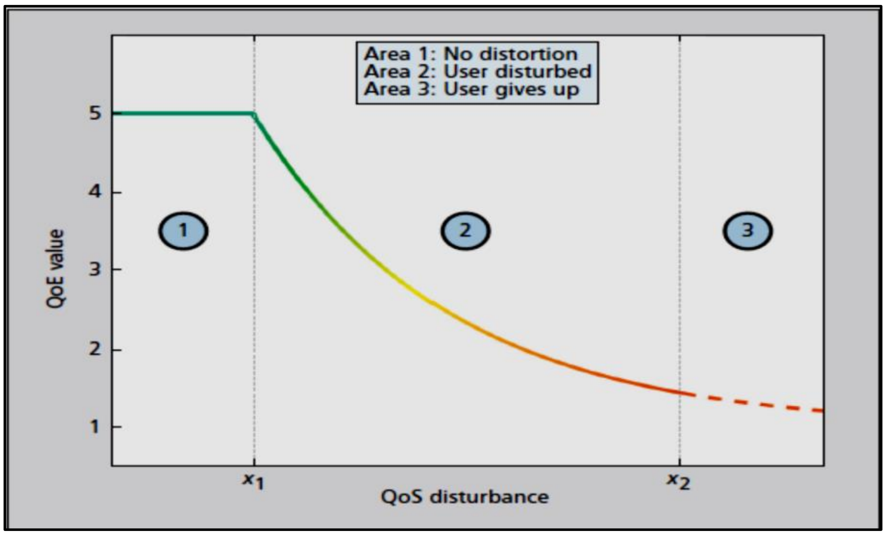


Figure 3-3 The relationship between QoS disturbance and QoE value (Fiedler et al., 2010)

Recently, researchers have paid attention to the use of a QoE as a measure for the quality of the services paid for by a user. Reichl and Zwickl discussed using QoE in pricing the services; in other words, charging users according to their preferences. They proposed the use of the concept of "price/quality" and classifying the users according to the tariff they are willing to pay for the service's QoS level (Reichl & Zwickl, 2015). The idea of considering a user-centric or QoE monitoring for the SLA was discussed by Varela et al. (2015), who proposed presenting the SLA as an Experience Level Agreement (ELA). The authors defined the ELA as "a special type of SLA designed to establish a common understanding of the quality levels that the customer will experience through the use of the service, in terms that are clearly understandable to the customer and to which he or she can relate." In other words, receiving the service with guaranteed QoE. The researchers argued that although the SLA parameters present information about the performance of the service, they fail to represent the perceived user experience. The study discussed the need for an agreement upon metrics and quantifiable measurements, in addition to an automatic monitoring architecture to manage the ELA assurance, by suggesting the use of quality scales such as star-rates to reflect the user experience instead of the usual QoS quality levels (Varela et al., 2015). This concept, as presented in their study is comparable to the concept presented in this thesis about monitoring the QoE of SLA. While their research emphasised the importance of defining a new agreement to guarantee the QoE value, this thesis proposes estimating the value of QoE based on SLA parameter values.

The importance of considering QoE in cloud computing is also discussed by Kafetzakis et al (2012). The researchers presented QoE4CLOUD, a framework for managing resources in cloud computing, taking into consideration the actual perception of received services (QoE). This research considered handling three quality dimensions, which are QoS, QoE, and Quality of Business (QoBiz) that represents the cost of the perceptual service. The QoE is measured continuously using an agent on the cloud side, taking into consideration the SLA. The authors claimed that considering this framework can improve cloud resources usage though using a unified metric (Kafetzakis et al., 2012). Although this study can be considered a reference to the importance of QoE in a cloud, it fails to determine the main parameters and the analysis method that can be taken into account in measuring the QoE.

Hasan et al. (2013) discussed the importance of defining a quantified value for the QoE instead of a subjective one, where the latter is not adequate to manage the SLA. The authors proposed an exponential relationship between QoE and SLA. They suggested solving the problem of virtual machine and resources' allocation through predicting SLA violations, in order to improve the QoE and minimize the SLA violations (Hasan & Huh, 2013). Although this study proposed a way for quantifying the QoE, this reflects resource allocation which is not adequate to predict the QoE of SaaS in the cloud, as it depends on other factors related to the SaaS characteristics.

Many studies have emerged on the subject of monitoring QoE as a measure of the performance of cloud services. Safdari and Chang (2014) presented a review of QoE in cloud computing. The study emphasized the importance of considering the network QoS and the SLA in addition to the monitoring of the datacentres in assessing QoE. The authors also mentioned that the SLA is not enough to express cloud service user satisfaction, and that quantifying the value of QoE is an open issue. The authors also presented an expert survey by which they identified six main KPIs that affect the success of IT service in the cloud, these KPIs were usability, performance, security, accuracy, data portability, and scalability (Safdari & Chang, 2014).

There were many efforts for estimating the QoE value of services provided in cloud computing, taking into consideration the network performance, such as (Casas et al., 2012) (Casas, Seufert, et al., 2013) (Casas, Fischer, et al., 2013) (Jarschel et al., 2013). Casas et al. (2012) studied the QoE in YouTube and Facebook, the authors proposed a user-centric monitoring of QoE taking into account real time and different network conditions. The study considered a lab experiment by asking the participants to use the application and provide their opinion about the received service using the MOS (Casas et al., 2012).

Casas, Seufert et al. (2013) proposed measuring the QoE of remote virtual desktop services of cloud computing. The study, based on the use of lab experiments, investigates the effect of network performance in terms of bandwidth and delay in terms of the response time on the QoE. In other words, it discussed the effect of QoS on the QoE value. The participants were asked to evaluate virtual desktop cloud services using the MOS scale for different network conditions (Casas, Seufert, et al., 2013).

Casas, Fischer et al. (2013) proposed measuring the QoE in the cloud storage and file sharing services. The study evaluated the QoE in terms of the network QoS, through the use of lab tests and asking the participants to use the cloud storage service with different file sizes and network conditions. The results of the study were evaluated using the MOS scale, and the acceptance which shows whether the users' continued using the service with the network conditions considered in the test (Casas, Fischer, et al., 2013).

These studies showed that the QoE is affected by the QoS of the network, where user satisfaction decreased in the case of declining network performance. The common theme in the work presented in the aforementioned three papers is the use of subjective lab tests, and evaluating the QoE using the Mean Opinion Score (MOS) as a measure to identify the degree of user satisfaction. However, the main drawbacks of these researches were, firstly, they overlooked considering the SLA document's parameters in the measurement of the QoE; and secondly, although these studies contribute to this field of the research in terms of showing the effect of the QoS in monitoring the QoE of cloud services, they concentrated on the network performance instead of the cloud provider service itself.

Casas and Schatz (2014) presented a survey about measuring QoE measurements in cloud computing. The authors highlighted the importance of QoE and considered the QoE to be a guiding paradigm for developing applications in cloud computing, where the degradation in the levels of the perceived service quality leads the clients to reject the service. The study introduced a detailed evaluation of different cloud services which are remote desktop services, storage, telepresence, and video streaming It also considered checking the difficulty of accomplishing the test and interacting with the cloud service by the participants as a result of network degradation (Casas & Schatz, 2014).

Furthermore, Jarschel et al. (2013) presented another subjective study to measure QoE in cloud gaming. The authors studied the effect of the network QoS on the QoE value, packet delay and packet loss were chosen as the QoS parameters. To achieve this, the users were asked to play online games and to rate the perceived cloud services in terms of the MOS, which is used to estimate the QoE value. The obtained results showed that the QoE is affected by the QoS, where the degradation in QoS caused the QoE to decrease (Jarschel et al., 2013). Again, the study overlooked considering the effect of the SLA parameters; it also concentrated on the network performance instead of the cloud provider's service.

Tao et al. (2013) emphasized the importance of using QoE as an indication of the perceived quality of the application in cloud computing. They argued that the network QoS is not enough to estimate QoE, and proposed defining factors to measure QoE of the cloud applications depending on experiential marketing dimensions like think, feel, and sense of the received services (Tao et al., 2013). Although this research shed the light on consideration of the user perception in measuring QoE, it lacked the details of the monitoring process or the measurement of QoE.

Zhang et al. (2014) proposed a framework for ranking and selecting services in cloud computing reliability. The authors suggested using QoE for ranking the cloud services before selecting them, to help the user in the selection process. The study introduced a middleware to manage the QoE monitoring process; the middleware uses agents on both the customer and the cloud side to collect the clients' feedback and the QoS parameters, then, by comparing the monitored QoE value with the SLA parameters this can help in estimating the reliability of the services (Zhang et al., 2014). Although the study considered QoE estimation in a pre-use of the service, it failed to estimate the user satisfaction after using it or specifying the parameters which affect the user satisfaction about SaaS.

The study presented by Shin and Huh (2015) highlighted the importance of measuring the QoE in cloud computing as an indicator of SLA compliance. The research focused on considering the services provided by different providers and hence different SLAs in an inter-cloud architecture. A broker was proposed to receive the users' requests and estimating the aggregated QoE value (Shin & Huh, 2015). Despite attempting to estimate the QoE of the inter-cloud architecture, the study overlooked defining a metric for the QoE for SaaS. Defining a metric for QoE is handled in this thesis by presenting a middleware to manage the measurement of QoE without a need to a broker role. Another drawback that has been noticed in their research, that the authors proposed calculating QoE as a function of QoS for each parameter multiplied by a factor, but they failed in defining a way for defining the weight of these factors (which is difficult to manage in an automated way for different SLA cases).

3.3.1 Measuring QoE

As the QoE of SaaS is considered an estimation of the software quality from user's perspective, estimating a value of software metric requires a method to estimate its value. According to the study presented by Gray and McDonell (1997), which introduced a comparison of the evaluation techniques of software metric. Their study presented a set of techniques which are: **Least square regression**, this technique is based on assessing the coefficients' values; **Robust regression**, which is based on increasing the robustness of the model by modifying the error measure of the least square method; **Neural networks**, this is one of the techniques used for software model design, which is based on learning things using back-propagation, in order to make decisions similar to human's decision; **Fuzzy system**, this method is based on mapping the linguistic and numerical values, and using rules to make decisions based on experts' opinions; **Hybrid Nero-Fuzzy systems**, this method is based on combining the neural and fuzzy systems; **Rule based systems**, this method is based on set of rules activated by facts, this method is different from fuzzy rule method through allowing only true or false values for the "antecedents and consequents", without considering degrees of these values; **Case-based reasoning**, this method considers making decisions based on previous observations about project's requirements, in a way similar to experts' decision making based on their knowledge; and **Regression trees and Classification trees,** these methods are based on training the rules based on known dataset. Gray and McDonell compared these techniques by introducing a set of criteria that describes the main modelling attributes, the results achieved by their study revealed that fuzzy logic engine outperforms the other techniques as shown in Table 3-1 (Gray & MacDonell, 1997). Note that the underlined values mean achieving the top level. Fuzzy system is the method used in this thesis to estimate the QoE value.

Table 3-1 Comparison of software metric evaluation techniques (Gray & MacDonell, 1997)

Technique	Model free	Can resist outliers	Explains output	Suits small data sets	Can be adjusted for new data	Reasoning process is visible	Suit complex models	Include known facts
Least Square Regression	No	No	Partially	No	No	Yes	No	Partially
Robust Regression	No	Yes	Partially	Partially	No	Yes	No	Partially
Neural Networks	Yes	No	No	No	Partially	No	Yes	Partially
Fuzzy Systems	Yes	Partially	Yes	Yes	Partially	Yes	Yes	Yes
Hybrid Nero-Fuzzy Systems	Yes	Partially	Yes	Partially	Partially	Partially	Yes	Yes
Rule Based Systems	No	N/A	Yes	N/A	N/A	Yes	Yes	Yes
Case-Based Reasoning	Yes	Partially	Yes	Partially	Yes	Partially	Yes	No
Regression Tress	Yes	Yes	Yes	Partially	Yes	Partially	Yes	Partially
Classification or Decision Trees	Yes	Yes	Yes	Partially	Yes	Partially	Yes	Partially

3.3.2 Using Fuzzy Systems in Measuring QoE

The fuzzy logic system has been used by many researchers for measuring QoE value. Fuzzy logic was defined by Zadeh as "… a precise logic of imprecision and approximate reasoning" (Zadeh, 2008, p. 2751). Fuzzy systems are used for estimation and decision making (McNeill & Thro, 1994); they are based on linguistic rules that make computers' reasoning closer to that of humans (Jantzen, 2013). A fuzzy system is composed of the following components: the Fuzzifier, which is used to map crisp data into fuzzy data; the Inference, which combines the rules; and the DeFuzzifier, which is used to map fuzzy data into crisp data (Mendel, 1995). Figure 3-4 shows the main components of the fuzzy logic system.

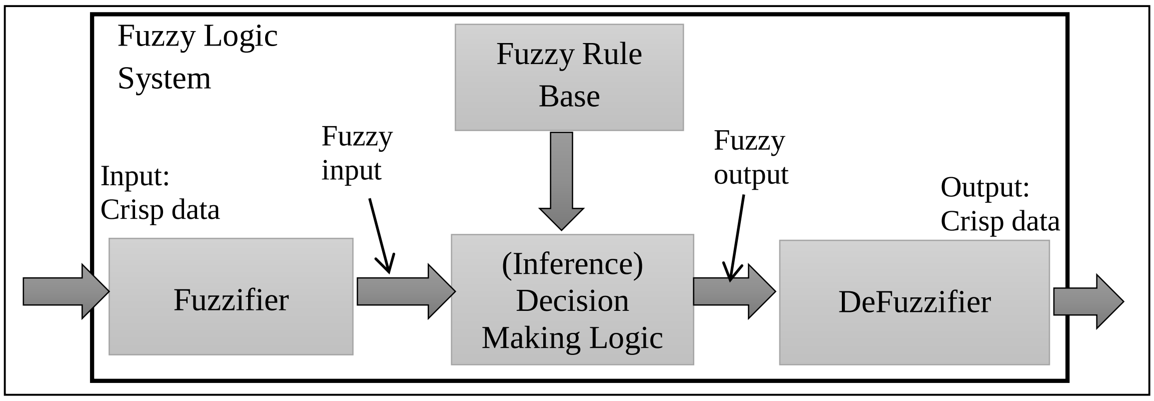


Figure 3-4 Fuzzy system elements (Mendel, 1995)

Evaluating user satisfaction in cloud computing has been handled by many researchers. Several of these studies considered fuzzy logic in measuring QoE in monitoring cloud computing. For example, Pilevari et al. (2013) proposed a model for evaluating user satisfaction in cloud computing. They used fuzzy logic to estimate the value of QoE. The study included deciding the attributes that affect user satisfaction. The selection process of the attributes and defining the membership function of the fuzzy engine were determined by using the literature and expert opinion about a cloud service provided by an Internet Service Provider (ISP) company in Iran (Pilevari et al., 2013). The main weakness of this study is ignoring the SLA parameters in measuring the QoE value; furthermore, the selected criteria were not specified for SaaS.

Another model was proposed by Alhamad et al. (2011) to evaluate trust in e-learning applications in IaaS cloud computing. In this research, trust was defined as a factor for managing the relationship between the client and provider. Trust was considered vague and depended on subjective factors; therefore, the study used fuzzy logic in the evaluation process. The authors defined trust in terms of: availability, scalability, security, and usability. The parameters were defined by domain experts, while the fuzzy rules were defined using online surveys. According to the study, the weights of the fuzzy parameters are decided by the service user, but this process is impractical and not useful to define a generic metric, as it is difficult to adjust these values for each user (Alhamad et al., 2011). However, this research did not consider SLA document parameters in evaluating trust value.

Baliyan and Kumar (2013) submitted a study to assess the quality of SaaS in cloud computing. They proposed a quality model for SaaS; the quality attributes were selected based on literature review. A fuzzy logic model was used for evaluating the quality of SaaS (Baliyan & Kumar, 2013). Although this research was dedicated to SaaS, the quality attributes are not dedicated for measuring the QoE, and the study also failed to consider SLA parameters in estimating the quality.

Another study was presented by Samet et al. (2016) to evaluate the QoE of video services in cloud computing. The authors mentioned the importance of monitoring the multimedia service from users' perspective, they decided the factors affect user satisfaction based on the video characteristics such as buffering time and QoS like packet delay. Fuzzy logic was used as an approach for evaluating the QoE value, the authors discussed the benefit of using the fuzzy approach because of the difficulty of assigning crisp values for some factors like application's easiness of use (Samet et al., 2016). The main weakness of this study is overlooking the SLA document, where the SLA is a main factor in cloud services, the study gives poor details about the generation of the membership functions of the proposed fuzzy engine.

3.4 SLA Monitoring

Managing an SLA has attracted the attention of many researchers who proposed managing the relationship between the cloud provider and cloud consumer through mentioning the importance of considering the SLA parameters in the process of monitoring a cloud computing service. This section introduces an overview of the related work in the area of managing and monitoring an SLA.

3.4.1 Managing SLA Monitoring Process

The first direction of this section is the management of SLA monitoring. The importance of monitoring an SLA as a way of deciding the reputation of a service provider was discussed by Rana et al. (2008), this is essential to manage a relationship between a client and a provider and to determine the violations of an SLA if any. The authors defined three types of monitoring modules, these modules are: a model on the customer's site to help clients in deciding their trust in a service provider; a trusted independent third party (TTP) which can monitor all the communications; or a trusted module that is included in the provider side which helps to monitor the internal state of the service provider (Rana et al., 2008). Whilst their study presented an architecture for monitoring the SLA compliance using a third trusted party, it focused on defining penalties in case of violating an SLA terms. Their research might have been more interesting if the authors had paid more attention to defining approaches for monitoring the user satisfaction of SLA. Another weakness of this study is the use of a third party, which is considered unreliable, with additional cost and overhead. The monitoring framework proposed in this thesis considers developing a middleware that has a combination of both a client and a server side monitoring to accomplish full monitoring of the system.

A monitoring framework had been proposed by Comuzzi et al. (2009) to monitor SLAs taking into account the relationship between monitoring and establishing SLAs. The researchers mentioned the importance of a historical monitored data of QoS when

establishing or renegotiating an SLA between a client and a provider in case of changing the SLA terms, however, the study was dedicated to server-side monitoring without considering customer perspective or managing user-centric monitoring (Comuzzi et al., 2009).
Another architecture is proposed by Kertesz et al. (2009) for managing the allocation of resources in virtualised cloud environments by considering the QoS parameter values specified in an SLA document at the negotiation phase. This management process requires a monitoring service of provider resources (Kertesz et al., 2009). Although the researchers mentioned the significance of monitoring an SLA, their focus was on the resource allocation process, and the solution failed to present an approach for estimating user satisfaction with services nor notifying the user about any violations in the SLA document.

Brandic et al. (2009) presented a framework for managing an SLA through introducing a middleware that uses mappings to handle unmatched SLA templates of cloud providers and customers, with SLA mapping accomplished using WSDL. Although this framework considered monitoring SLA parameters after the negotiation process, it overlooked the presentation of user-centric monitoring or informing the client about any lack in the received services (Brandic et al., 2009).

Emeakaroha et al. (2010) introduced LoM2HiS, a framework to manage an SLA in cloud computing through mapping resource metrics to SLA parameters (Vincent Chimaobi Emeakaroha, 2012; Vincent C Emeakaroha et al., 2010). Although the authors claimed that this framework helps in detecting violations in an SLA through controlling the resources allocated to the selected services, the main contribution of the study is to map the low level monitored metrics to high level SLA parameters with a focus on collecting the measurements on the server side, ignoring user-centric monitoring.

Several studies revealed the need for monitoring services provided in the cloud. Torkashvan and Haghighi (2012) proposed the CSLAM framework to manage SLAs in cloud computing. The researchers highlighted the importance of managing an SLA life cycle in the cloud and of considering monitoring the SLA parameters as part of it; this is achieved by adapting WSLA with some extensions to fit a cloud computing environment. However, the study did not give enough details about how to notify the client side about

SLA violations or managing the transformation of the monitored data between the client and provider (Torkashvan & Haghighi, 2012a).

Another framework was presented by Motta et al. (2013) to monitor the quality of SLA in cloud computing. The authors mentioned the importance of considering the following approaches in managing the monitoring of the quality: the cloud’s resources on the provider side in terms of capacity and quality; the authority of a third party mediator, proposed to manage the monitoring process; in addition to the SLA negotiation and QoS analysis in the client side (Motta et al., 2013). Their study provides limited technical details of the web services used in the monitoring process.

A proposal was submitted by Firdhous et al. (2013) to monitor services provided in the cloud in which the researchers suggested developing monitoring techniques to monitor the QoS in cloud computing, and evaluating these services taking into consideration the SLA parameters values (Firdhous et al., 2013b). Although the research presented a theoretical description of the research problem, it failed to present details about implementing the monitoring process or techniques for managing the communication between the provider side and the client side.

Many studies have been presented to manage an SLA in cloud computing taking into consideration the QoS of the received services (Badidi, 2013), (Motta et al., 2014), (Mosallanejad et al., 2014) and (Khaddaj et al., 2014). However, these studies did not give sufficient detail regarding management of the communication between the provider side and the client side and the types of web service.

Mosallanejad et al. (2014) proposed SH-SLA, which is a hierarchal self-healing SLA management system in cloud computing which includes QoS monitoring and detecting SLA violations. The proposed self-healing system takes into consideration the effect of the relationship between the different layers in clouds (Mosallanejad et al., 2014). In spite of considering the monitoring of the different types of services in the cloud and detecting the SLA violations, the weakness of this study is ignoring the user-centric monitoring of an SLA and communication management details.

Motta et al. (2014) presented a Service Level Management (SLM), a framework to manage an SLA in cloud computing. The study proposed the use of a third party in controlling the SLA life cycle such as the negotiation, compensation, billing, and monitoring, and a monitoring agent was suggested as one of the SLM’s components to analyse the cloud services based on the status of the QoS of provided services and the contracted SLA document (Motta et al., 2014).

Wu et al. (2015) presented SLARMS, which is a framework for managing the SLA in SaaS that detects SLA violations. The proposed study considered that the decision making of the management process is used to manage the resource allocation in the cloud environment taking into account the SLA requirements of the client (L. Wu et al., 2015). Although this study considered SLA management, it failed to provide detailed information about the web services used to manage the monitoring process.

Another framework was presented by Anithakumari and Chandrasekaran (2015) to manage SLA in cloud computing. The study considered monitoring the SLO to check breaches of the SLA. The authors claimed that monitoring the SLA parameters helps in estimating the required resources to be allocated, which in turn can be used for predicting the future violations in cloud computing (Anithakumari & Chandrasekaran, 2015). Although the framework managed the SLA in the cloud, it failed to describe supporting REST architecture or SOAP protocol.

Geebelen et al. (2012) presented a middleware to help SaaS providers in customizing the services according to user requirements. Although the study emphasised the importance of considering the QoS of SLA for each tenant in SaaS, it was dedicated for the services’ customization and configuration instead of estimating the user satisfaction and SLA violations, it also failed to provide detailed description for managing the web services in the monitoring process (Geebelen et al., 2012)

Managing the SLA of SaaS was introduced by Cheng et al. (2009). The authors mentioned the importance of considering the SLA parameters in monitoring the performance of SaaS, which is used to manage the resources’ allocation based on the SLA requirements. The study emphasised considering the concept of multi-tenancy in the monitoring process, in addition to the requirements of each client as presented in the SLA. However, it considered using SOAP protocol in monitoring the web services (Cheng et al., 2009).

Khan et al. (2016) proposed a framework to monitor SLA compliance in SaaS cloud computing taking into consideration the QoS. The authors mentioned the importance of monitoring the responsibility and managing the relationship among the SaaS service provider, client, and “cloud facilitator” who is in charge of providing the infrastructure resources; this requires considering the QoS of both SaaS provider and IaaS provider in detecting SLA’s violations. They also mentioned the significance of issuing a warning detailing the parties responsible for the quality degradation (Khan et al., 2016). Although the study handled the monitoring of SLA taking into consideration the QoS, but it failed considering the subjective parameters in estimating the QoE or user satisfaction about the received services.

The middleware developed in this thesis (MonSLAR) shares with the aforementioned studies the concept of SLA management and monitoring in cloud computing. However, MonSLAR focusses on SLA monitoring and the presentation of a user-centric monitoring for user satisfaction of SaaS.

3.4.2 Representing the SLA Parameters in Web Services

The second direction in the SLA management field is the representation of the SLA parameters and the monitored parameters of the SLA in web services in general and cloud computing specifically. Two main frameworks have been used to define the SLA specifications in web services: WSLA (Web Service Level Agreement) which is introduced by IBM (Keller & Ludwig, 2003), and WS-Agreement (Web Service - Agreement) which is introduced by a working group of Open Grid Forum (OGF) as a protocol for defining the services in an SLA (Andrieux et al., 2007). These frameworks provide standardised agreement templates (Wieder et al., 2008). These standards use an XML based language schema for defining the agreement structure and define how the SLA parameters can be implemented in the system and how their functions can be handled in the SLA negotiation process; they help in automating the monitoring process of the SLA (Bianco et al., 2008). Then, many attempts have been made in the field of the SLA parameters monitoring and data management; these attempts were dedicated to SOAP protocol web services, such as extending WSDL to include SLA parameters as in (D'Ambrogio, 2006). Another study submitted by Torkashvan and Haghighi (2012) proposed a new language to represent SLA parameters based on WSLA language (Torkashvan & Haghighi, 2012a), while other researchers proposed embedding the SLA parameters and the monitored data in the SOAP messages (Zulkernine et al., 2008).

D’Ambrogio (2006) discussed the problem of presenting the QoS parameters in the WSDL of SOAP web services. The author proposed a model for extending the WSDL to include the QoS data, this was accomplished by introducing the Q-WSDL meta-model for the extended XML document of WSDL. According to the researcher, this model is considered a base for representing the QoS of an SLA document to manage the web services (D'Ambrogio, 2006).

Zulkernine et al. (2008) proposed monitoring the SLA in SOAP web services and sharing the monitored data and violation reports between a provider and a client. The authors introduced PM, a middleware for monitoring the web services which sends the monitored data from the web service provider side to the client side and the SLA data from the client side to the provider side by embedding this data in the header of the SOAP messages (Zulkernine et al., 2008). This research will be compared to MonSLAR in the evaluation chapter (section 7.3.1), as it introduces an approach for embedding the SLA monitored data in SOAP messages, which is the most relevant for the approach used in this thesis; comparing MonSLAR with PM gives a clear indication of the difference between using REST and SOAP messages used in sharing the data.

Although the research in the second direction added contribution to the way of representing the QoS of an SLA in SOAP web services, there is a failure to represent these values in REST web services because WSDL is not a REST technology. This thesis presents a new approach for representing and transforming the SLA parameters in REST services, through extending the HTTP messages and embedding the monitored parameters values in the HEAD and OPTIONS REST methods. Thus, it illuminates the need for the use of additional technology such as WSDL.

3.4.3 Managing SLA using REST

The third direction in this section and the most relevant one to the approach presented in this thesis, is the use of REST to manage SLAs. Many researches handled the concept of using REST and HTTP in managing the SLA in web services in general and cloud computing specifically. An approach was submitted by Blumel et al. (2011) to manage the SLA document in cloud computing using a REST-based architecture by providing electronic based contracts, and allowing users to create and modify SLAs by considering these SLAs as the resources of the REST architecture. The proposed approach was developed upon WS-Agreement XML documents (Blumel et al., 2011).

Another attempt is introduced by Kübert et al. (2011) to implement the WS-Agreement using RESTful web services. The authors claimed that this study can help in managing the SLA life cycle in an automated way, and discussed the fact that SLAs managed in previous researches using SOAP and the need to manage them using REST. They investigated presenting the specifications of WS-agreements in REST, this includes: identifying and representing the SLA resources (Kübert et al., 2011).

The researchs presented by Blumel et al. (2011) and Kübert, et al. (2011) offered to manage the SLA life cycle and focused on the negotiation and creating of an electronic contract. However, it would be difficult to employ these methods in representing the measured SLA parameters in an automated user-centric monitoring middleware, as these methods are specified for managing the SLA creation and deletion rather than managing monitoring the SLA parameters or transmitting the violations in the SLA documents. In addition, there is the advantage taken from the approach presented by MonSLAR to present their REST methods as add-on services for the SaaS provider services, where the monitored data are presented as resources in MonSLAR.

Amato et al. (2014) presented a study which uses REST to manage the dynamic negotiation of an SLA in cloud computing. The study proposed a negotiation interface for the SLA to help in allocating the cloud IaaS resources in an automated way taking into consideration the SLA parameters. In this study, a cloud broker agency was proposed to manage the negotiation process, by selecting the resources required and monitoring them after the execution of the service; however, this study focused on the negotiation phase and ignored the monitoring process. REST messages were used to send requests and manage the agreement with or refusal of the services (Amato et al., 2014), but this research did not consider an SLA monitoring or violation detection.

3.5 Chapter Summary

This chapter reviewed literature related to the research subject of the thesis, in order to identify knowledge gaps in previous research with a view to support the current research problem. The literature reveals that many studies have been introduced in the field of monitoring cloud computing.

The chapter began by introducing the cloud monitoring concept and presenting its related literature review. The cloud monitoring section included two sub sections, the monitoring frameworks and the available quality models. The monitoring frameworks are categorised into three main categories, Table 3-2 summarises the main directions considered in the monitoring framework section. These directions are:

1- Server-centric frameworks, where the monitoring process is managed by the cloud provider.
2- User-centric frameworks, where the monitoring process is managed by the cloud’s client side.
3- Third-party frameworks, where the monitoring process is managed by an independent third party component.

It is important to mention that most of the presented monitoring frameworks focus on monitoring cloud resources, and the transmission of monitored data between monitoring agents and providers, but very little on how to deliver these data to the client side. It is also worth noting that these frameworks consider the SOAP protocol in managing the monitoring process, through exploiting the WSDL document in exchanging the monitored data. However, these frameworks are inefficient in managing the cloud REST services. The literature reveals a weakness in available research in tackling this issue.

The chapter also summarized the available quality models; this helped in choosing the most suitable model for measuring the QoE of SaaS in cloud computing, taking into consideration the SaaS user requirements. Table 3-3 summarizes the available quality models.

The chapter also presented the previous research in the subject of QoE measurement, while showing that QoE had been used as a measure of user satisfaction; it also revealed a lack in defining a measure for QoE of SaaS. Most of the related research either ignored the concept of cloud computing, or ignored the effect of the SLA on the QoE value.

Then, SLA-based monitoring in cloud computing was presented in Section 3.4. This section is divided into three main subsections, which are:

1- The proposed SLA management frameworks.
2- Representation of the SLA parameters in the monitoring process.
3- The use of REST to manage the SLA document.

The researches presented in this section highlighted the importance of managing the SLA in cloud computing, and showed the need for an approach to represent the SLA parameters and the monitored data in the REST architecture. Table 3-4 summarises the main directions considered in this section in presenting the SLA oriented Monitoring.

Table 3-2 summarises the work related to the monitoring frameworks

The reference	Mode of monitoring	Purpose	Limitations
(Shao & Wang, 2011)	Server-centric	monitoring framework to measure the performance of cloud applications	Lacked managing the communication between the client side and the provider side, or providing user-centric monitoring
(Mastelic et al., 2012)	Server-centric	M4CLOUD, a framework for monitoring cloud applications	Lacked managing the communication between the client side and the provider side, or providing user-centric monitoring
(Trihinas et al., 2014)	Server-centric	JCatascopia, a framework to monitor cloud applications	Lacked managing the communication between the client side and the provider side, or providing user-centric monitoring
(Povedano-Molina et al., 2013)	Server-centric	DARGOS, an architecture for monitoring resources in cloud computing	Lacked managing the communication between the client side and the provider side
(Cedillo et al., 2015) (Cedillo et al., 2016)	Server-centric	Presented a middleware to monitor SLA violations in cloud computing	Lacked providing a user-centric monitoring
(Smit et al., 2013)	Server-centric	MISURE, an architecture for monitoring cloud applications' resources	Lacked providing a measure for user satisfaction based on SLA
(Lu et al., 2016)	Server-centric	JTangCMS, a monitoring framework uses a decision making for the monitored data	Lacked providing a user-centric monitoring
(S.-Y. Lee et al., 2012)	Server-centric	A middleware to monitor the cloud computing using WSDL	Lacked providing a user-centric monitoring and supporting REST in the monitoring process

(Müller et al., 2012) (Oriol et al., 2015)	Server-centric	SALMonADA, a framework for monitoring the QoS of service based systems using SOAP.	Lacked supporting REST in the monitoring process
(Perez-Espinoza et al., 2015)	Server-centric	Presented a monitoring architecture for private clouds.	Overlooked presenting an automated user-centric monitoring and SLA management.
(Vincent C Emeakaroha et al., 2012)	User- centric	CASViD, a framework to detect violations in SLAs of cloud applications	Lacked the details of web services used in transmitting data between the provider and the client
(Montes et al., 2013)	User- centric	GMonE, a framework to monitor cloud computing	Lacked supporting REST in transmitting data between the provider and the client
(Rehman et al., 2015)	User- centric	UCSM, a framework assists users in making cloud service decision	Lacked supporting REST in transmitting data between the provider and the client, and ignoring SLA in the monitoring
(Nguyen et al., 2014)	User- centric	user-oriented monitoring for cloud computing	Lacked considering SLA compliance in the monitoring
(Siebenhaar et al., 2013)	Third-party	monitoring framework to measure availability of cloud applications	Using a broker, and overlooked estimating an overall user satisfaction about provided services
(Rak et al., 2011)	Third-party	A monitoring framework for the mOSAIC applications in cloud computing	Not applicable to other cloud applications
(Ye et al., 2012)	Third-party	A framework for monitoring and detecting SLA violations in cloud computing	Using a broker
(Alsulaiman & Alturki, 2012)	Third-party	model to monitor and evaluate the cloud services	Using a broker, overlooked notifying the user and the details of supported web services

(Cicotti et al., 2012)	Third-party	QoSMoNaaS, a framework for monitoring cloud services and detecting SLA violations.	Lacked providing a user-centric monitoring, and notifying clients about violations
(Amato et al., 2012)	Third-party	Evaluating the SLA in cloud computing in the negotiation phase	Using a broker, overlooked monitoring services after usage
(Badidi, 2013)	Third-party	A framework manages the SLA in SaaS cloud computing	Lacked detailing the technical issues about web services used in the monitoring process
(Khaddaj et al., 2014)	Third-party	A framework monitors the QoS and check SLAs' violations	Using a broker
(Aversa et al., 2015)	Third-party	An architecture for monitoring cloud applications taking into consideration the non-functional requirements and the performance of cloud resources.	Lacked providing a user-centric monitoring for the overall user satisfaction about the cloud services
(Hammadi & Hussain, 2012)	Third-party	A framework for monitoring SLA compliance in cloud computing	Using a broker
(You et al., 2015)	Third-party	A framework provides SLM-as-a-service using the WSDL	Using a broker, and lacking supporting SaaS-services
(Katsaros et al., 2011)	Third-party	NEB2REST, an architecture uses REST to manage invoking the monitoring services	Failed in providing an automated user centric monitoring, and SLA violation detection.
(Moustafa et al., 2015)	user-centric	SLAM, a framework for monitoring cloud computing	Lacked providing online automated monitoring, it also lacked supporting REST
(Serhani et al., 2014)	user-centric	A study to check SLAs' violations in cloud computing.	Lacked managing automated monitoring for SaaS

Table 3-3 summarises the main quality models

The quality model	**Purpose or principle**	**Drawback**
SERVQUAL (Parasuraman et al., 1988)	Measure users' perception about the quality of services in the retail businesses	The model does not consider the concept of SLA in cloud computing
SMI CSMIC (2011)	Evaluate cloud services	Very general cannot be used for SaaS services
(J. Y. Lee et al., 2009)	Quality model for evaluating SaaS	Weakness in determining the quality dimensions
CLOUDQUAL (Zheng, 2013)	Quality model for evaluating cloud services	Lack in determining the weights of each parameter which is required for the fuzzy engine in MonSLAR
SaaS-Qual (Benlian et al., 2011)	Quality model used for measuring usage continuous in SaaS	-

Table 3-4 summarises the related work to SLA oriented Monitoring

The reference	**Supported web service**	**Purpose**	**Limitation**
WSLA (Keller & Ludwig, 2003)	SOAP	Presenting a standardized template to represent monitored QoS in the SLA	The model does not support REST
WS-Agreement (Wieder et al., 2008)	SOAP	Presenting a standardized template to represent monitored QoS in the SLA.	The model does not support REST
(Torkashvan & Haghighi, 2012a)	SOAP	Represent SLA parameters based on WSLA	The model does not support REST
(Zulkernine et al., 2008)	SOAP	Embedding monitored data in SOAP messages	The model does not support REST
(D'Ambrogio, 2006)	SOAP	Extending WSDL to include the QoS	The model does not support REST
(Blumel et al., 2011)	REST	Manage the SLA document and presenting SLAs as resources of REST	The method does not support representing the measurements of the SLA parameters
(Kübert et al., 2011)	REST	Implement WS-Agreement using REST	The method does not support representing the measurements of the SLA parameters

(Amato et al., 2014)	REST	Using REST to manage the negotiation of SLA	Overlooked managing the monitoring process and transmitting the measurements

To conclude, regarding SLA management, SOAP protocol and WSDL documents have been widely used in previous research, but little research has considered using REST. It is important to define a metric for measuring the QoE in SaaS.

According to the presented literature review in this chapter, it can be seen that none of the available frameworks supported the estimation of QoE in REST services of SaaS cloud computing based on a QoS monitoring.

In the next chapter, the proposed solution for the research problem is presented. The solution introduces a new user-centric middleware for monitoring SaaS services using REST architecture. The proposed middleware considers embedding the monitored SLA parameters in the REST methods without the need to use the WSDL document, or to exchange extra messages to handle the monitoring process. This kind of middleware helps in bridging the gap in previous studies of both providing user centric monitoring and supporting REST technology.

CHAPTER FOUR

THE PROPOSED MIDDLEWARE

4.1 Introduction

The previous chapters have shed light on the importance of proposing a user-centric approach for monitoring SaaS. With a view to avoiding an SLA violation, the main parameters should be determined in this agreement to evaluate the fulfillment of SLA terms. This chapter presents MonSLAR, a framework to measure QoE through monitoring SLA parameters using REST services in SaaS cloud computing. REST is an element of MonSLAR, which is a lightweight framework as it is based on the REST protocol for implementing the monitoring process.

As discussed in the previous chapter, delivering information about the monitored data to the cloud user is very important in maintaining a trusted relationship between the provider and the client. However, delivering this data adds extra overhead to the monitored cloud environment, due to the use of the SOAP protocol in previous works. As discussed in Chapter One, a motivation for this thesis is embedding the monitored data in the REST messages instead of using SOAP protocol messages to transmit them, by exploiting the REST architecture methods in the transmission of the monitored data.

Based on the knowledge that SOA implies the reuse of services in different environments, the services provided in MonSLAR are reusable, loosely coupled and platform independent. Taking into consideration that REST and SOA share the feature of loss coupling, which facilities the development of distributed systems, this is considered in REST by using the uniform interface through the URIs, as a full understanding of the REST concepts can assist in building high performance distributed systems (Vinoski, 2007). SOMs that have been developed so far, build services on top of web services that use SOAP and WSDL technology, while REST is used in MonSLAR to manage the web services.

This design can provide a scalable capability to the middleware as new services can be added to the system that can be managed using REST methods. The use of REST as a mechanism for transmitting the data helps in reducing the overhead and improving the performance of the system.

4.2 REST Methods

Rest services are based on HTTP protocol; so, managing the communication between clients and servers is done by exchanging messages using CRUD methods. REST uses HTTP methods to administer the web resources. These methods are:

- **GET**, which is used by the client to the server to receive a representation of the particular resource;
- **DELETE**, this method is utilized to remove and destroy a resource;
- **POST**, is used to create a new web resource by sending a representation of that resource by the client;
- **PUT**, is used to modify a resource.

Two additional methods can be employed by the client to scout an API;

- **HEAD**, is similar to GET method, but to retrieve the headers without the representation of the requested resource, which reduces the bandwidth of the exchanged signals;
- **OPTIONS**, to explore the supported methods by a resource. Both HEAD and OPTIONS are considered safe methods (Richardson et al., 2013).

The features of the last two methods and the fact that they are not used to retrieve resources' representations have been the motivation for this research to exploit them in exchanging the monitoring data in the proposed middleware.

4.3 The Proposed Solution – Monitoring SLA using REST (MonSLAR)

In this research, a new framework is submitted to measure QoE in terms of SLA parameters and the network's QoS. This framework includes embedding the SLA parameters in REST services and takes advantage of the requests sent by the user to transfer the SLA parameters without affecting the original requests by including these data in the header fields and reduces wasting extra signals in sending files to check SLA parameters.

The proposed middleware is introduced to measure user satisfaction with cloud services by measuring QoE as a function of both SLA parameters and the network's QoS (Al-Shammari & Al-Yasiri, 2014). The proposed middleware offers an automated monitoring process to check service levels and to compare them with their determined values in the signed SLA. The middleware has the ability to collect the required metrics and use them to calculate QoE and decide user satisfaction levels according to the QoE's estimated value.

The proposed middleware is designed by taking advantage of the cloud computing architecture and exploits the client and provider parties for the monitoring process without the need for a third party to manage it.

In this research, the term 'framework' is used to describe a software environment designed to simplify system management (Bernstein, 1996).

To ensure implementing safe methods on the server, REST's HEAD and OPTIONS methods are used to manage the monitoring process, as these two methods are considered safe (Velte et al., 2009). In regard to using PUT and POST to manage the SLA parameters, an approach has been suggested to create special services dedicated for middleware management purposes; this ensures the expected response from the SaaS provider is received. Although the HEAD and OPTIONS methods are part of the REST HTTP specification, they are not usually used in the request-response cycle within an SOA. By using these methods in the monitoring process, safety and backward compatibility are ensured, and at the same time, the system remains compliant with the REST protocol.

Detecting user satisfaction has been considered in two ways; first, the system helps the client to evaluate the level of services offered by the provider before invoking the service from the selected provider and, second, it allows client notification about violations in the delivered services. Monitoring the services in a pre-interaction before choosing the service and after signing the contract is important to maintain confidence between the client and the provider (Rehman et al., 2015). MonSLAR will help to sustain trust between the client and the provider and gives the client flexibility in selecting providers.

The main functionalities and actors in the proposed middleware are presented in a UML use-case diagram, as shown in Figure 4-1. The main actors are the service client, who is the user of the cloud service looking to monitor an SLA in terms of QoE; and the second actor is the SaaS provider, who is in charge of providing the SaaS services that are measured to check the SLA compliance. The figure also summarises the main functionalities of the middleware. It allows clients to '*invoke the service*' which includes requesting the service by the client and receiving the response from the SaaS provider. The middleware also facilitates the functionality '*monitor QoE*' which includes aggregating the main metrics, calculating SLA parameters and the measurement of user satisfaction. In the case of a violation, this functionality is extended to allow '*notify for violation*' which includes deciding the violation level, sending a notification and updating the user's satisfaction.

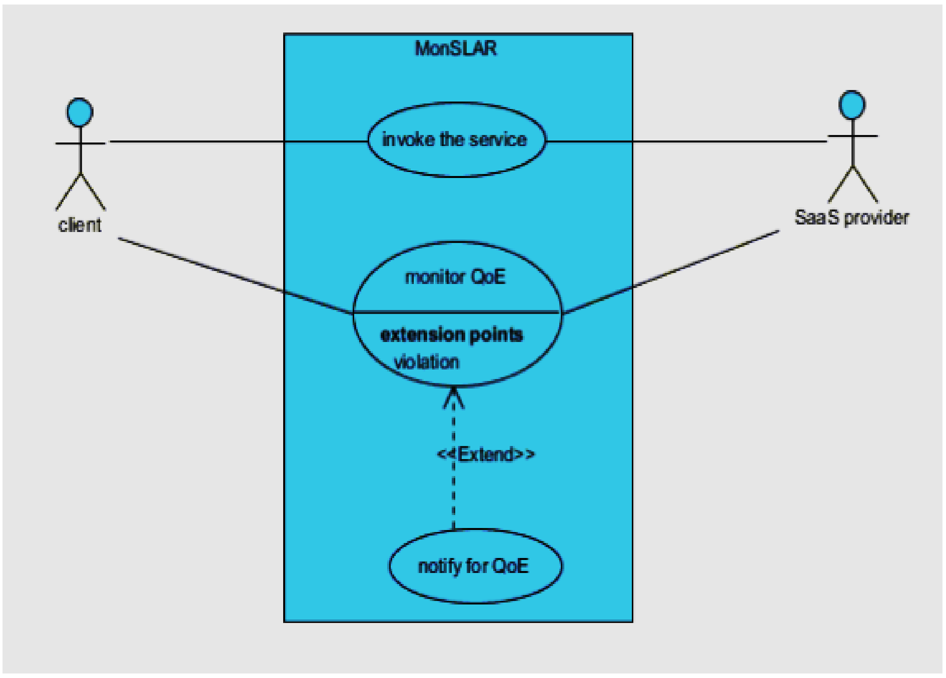


Figure 4-1 A use case model describes the main actors in MonSLAR

4.4 Architecture of MonSLAR

The proposed architecture presents two parts of the middleware on both the server and the client sides, as shown in Figure 4-2. Each part exploits the REST protocol to manage the monitoring process. The term 'architecture' here is used to describe the basic elements of the proposed middleware and the relationships among them (Bass et al., 2013).

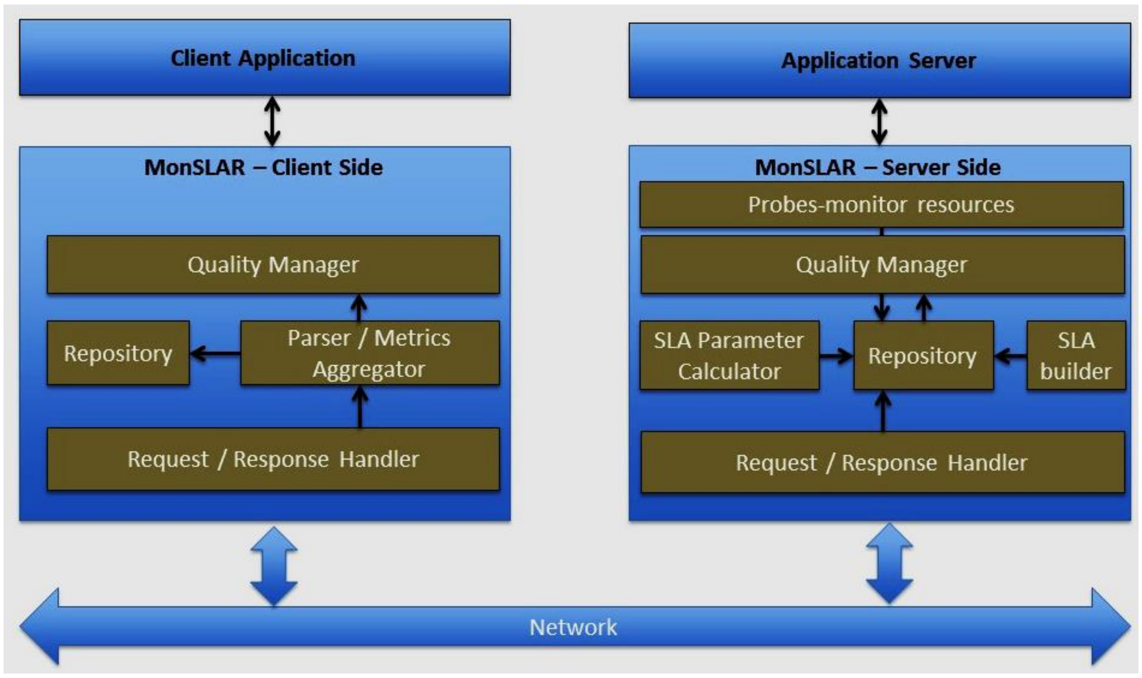


Figure 4-2 MonSLAR's architecture

The client side version is composed of the following components: (1) *Request/Response handler*, used to dispatch requests, receive responses and check the status of these responses to differentiate the errors caused by the network's failure from those caused by the service provider; (2) *Parser/Metrics aggregator*, this component is used to extract the main metrics to be used later for calculating the values of the measured SLA parameters; (3) *Repository* is used to store the collected data; and (4) *Quality Manager*, is used to decide user satisfaction with respect to the monitored services.

On the provider side, the main components are: (1) *Request/Response handler* which is used to handle client requests and invoke the relevant service; (2) *SLA parameter calculator*, which is used to calculate SLA parameters depending on the received metrics from the client side; (3) *SLA builder*, used to create the SLA document; (4) *Repository*, used to store parameter values that are used in the decision making process of the quality manager and to generate the violation reports; (5) *Probes-monitor resources* are agents used to monitor the underlying resources of the provider; and (6) *Quality Manager* is used to decide the violation depending on the SLA parameter values stored in the repository. The provider side middleware analyses the collected data and provides an automated estimation of the QoE value or the user satisfaction which is displayed to the user.

MonSLAR handles three types of request as described in Figure 4-3: *Service*, *Monitoring*, and *Management* requests. According to the type of request, three scenarios are specified to describe how the middleware handles the REST protocol methods and the SLA parameters management. The first type of request (Service request) is the client's request used to invoke a service from the provider. This request is passed (by the middleware) to the SaaS provider to be handled. As shown in the diagram, this request uses the usual REST methods (HTTP methods) for invoking the services.

The second type of request (Monitoring request) deals with three types of monitoring: the first is used to allow the client check the level of services provided by the SaaS provider before invoking them, and retrieve QoE value after using the service; this is done using a HEAD method. The second monitoring request is invoked by the client side middleware to update the server side with the measurements of the client side by sending PUT or POST methods' requests. The third monitoring request involves using OPTIONS method to retrieve the parameters' measurements values of the used service levels.

The third type of request (Management requests) includes permitting the middleware on the client side to query the SLA threshold values of the SLA parameter specified in the SLA document using the OPTIONS method. This function is used in two cases: the first is to update the client side middleware with the threshold values of the SLA parameters after the SLA negation process and creating the SLA document on the provider side, the second is to update the client side middleware in case of updating the SLA document; the management request is important in keeping the monitoring environment up-to-date.

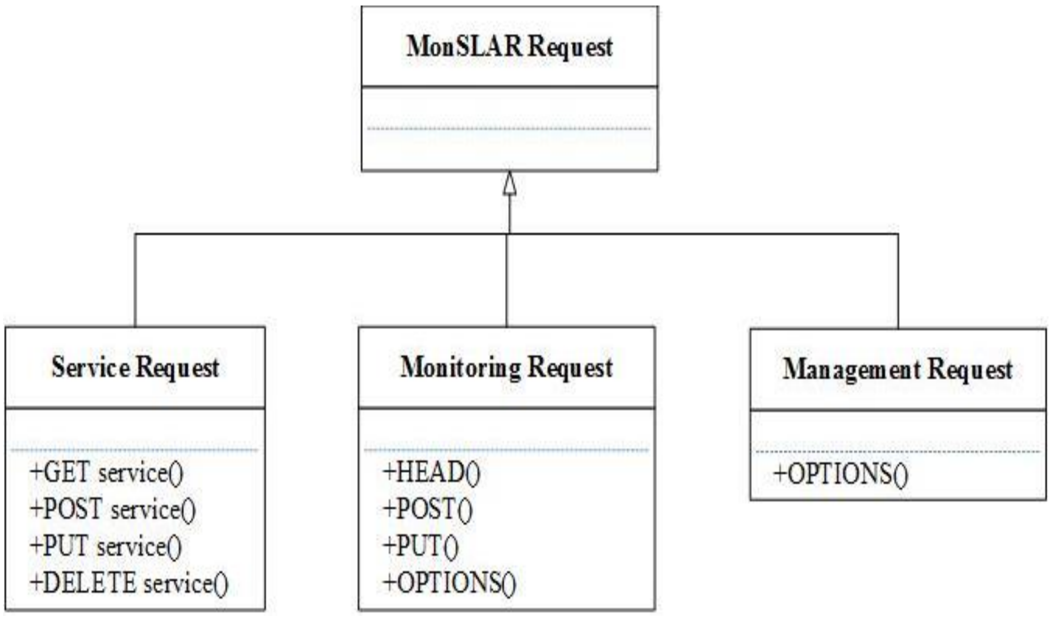


Figure 4-3 A UML model for requests' types in the proposed middleware

As described in the previous paragraph, the proposed framework only uses REST services and HTTP protocol methods to implement the monitoring process, collect the metrics, and exchange the monitored data on both sides between the client and the provider. This approach adds little overhead to the service provision because no additional documents need to be transferred between the client and the provider. Furthermore, no additional brokers or agents are required to manage and control the process.

The three types of request mentioned earlier supported by MonSLAR are presented in the next subsections.

4.4.1 Service Request

Figure 4-4 depicts the scenario of handling the service requests by MonSLAR. In this scenario, the middleware in the client side passes the service request to the SaaS provider that replies by providing the requested resources; the HTTP methods that represent this request are the GET, PUT, POST, and DELETE methods.

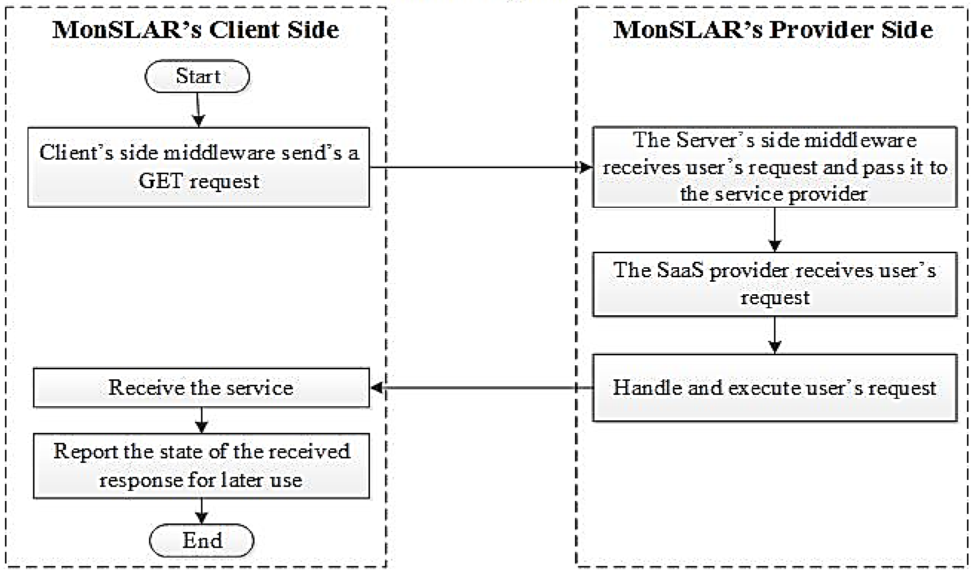
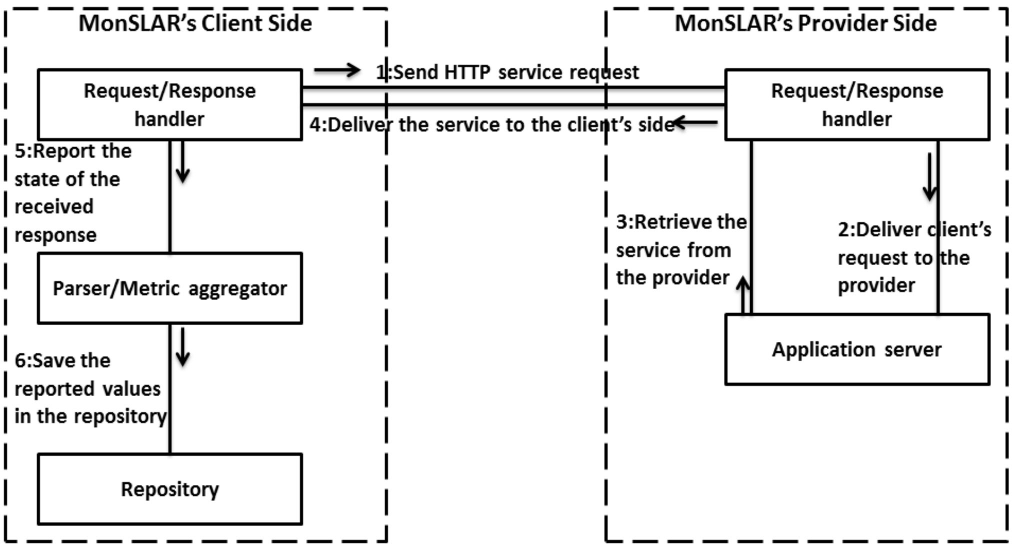

MonSLAR's Client Side
Start
Client's side middleware send's a GET request
Receive the service
Report the state of the received response for later use
End
MonSLAR's Provider Side
The Server's side middleware receives user's request and pass it to the service provider
The SaaS provider receives user's request
Handle and execute user's request

Figure 4-4 Handling the service request in MonSLAR

In Figure 4-5, a collaboration diagram is used to describe how REST service methods (GET, PUT, POST, and DELETE) are used to request services. After receiving the service response, it is checked by the client middleware and the main metrics are saved in the repository to be used later in deciding any violations.



MonSLAR's Client Side
MonSLAR's Provider Side
1:Send HTTP service request
Request/Response handler
4:Deliver the service to the client's side
Request/Response handler
5:Report the state of the received response
Parser/Metric aggregator
6:Save the reported values in the repository
Repository
3:Retrieve the service from the provider
2:Deliver client's request to the provider
Application server

Figure 4-5 Using REST methods in the MonSLAR's service request (collaboration diagram)

4.4.2 Monitoring Request

Figure 4-6 (a, b) describes handling (monitoring request-A) by MonSLAR; in this scenario, the request is handled by the server side middleware, taking into consideration the use of HEAD method as illustrated previously in Section 4.4. Figure (4.6-a), illustrates the monitoring process before invoking the service by sending an HTTP HEAD request which is handled entirely by the middleware. In this scenario, the middleware replys to the HEAD request by embedding the number of previous violations in the HTTP header. These values are useful for the client to evaluate the quality of services provided by the SaaS provider to its customers, where the number of previous violations provides a proactive monitoring of the service and considered as a warning to the client about the provided services. Figure (4.6-b) explains using the HEAD method to transmit the value of QoE as an indicator of the user satisfaction with used services, HEAD method has been used to retrieve the value of QoE in this request. As stated by the HTTP protocol specification, a HEAD response will only contain the header part of the response.

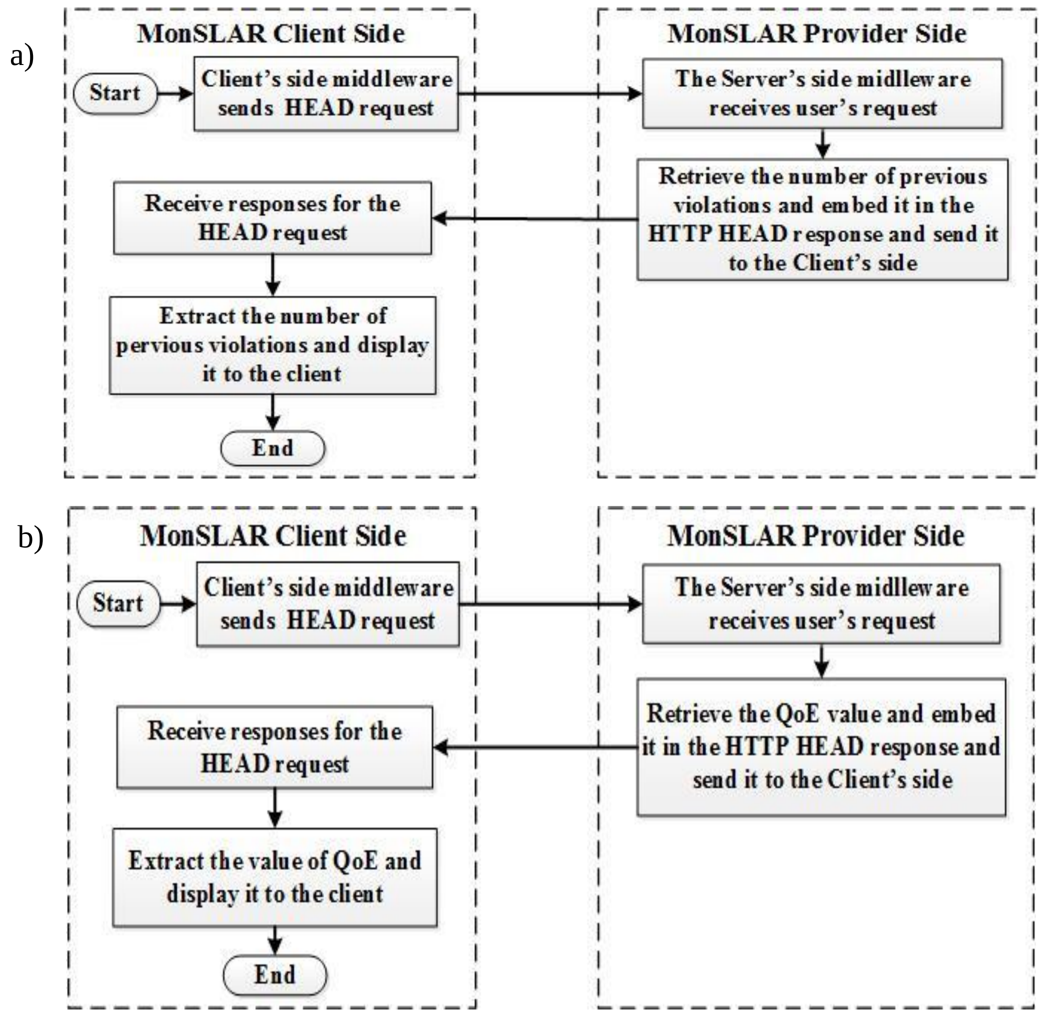
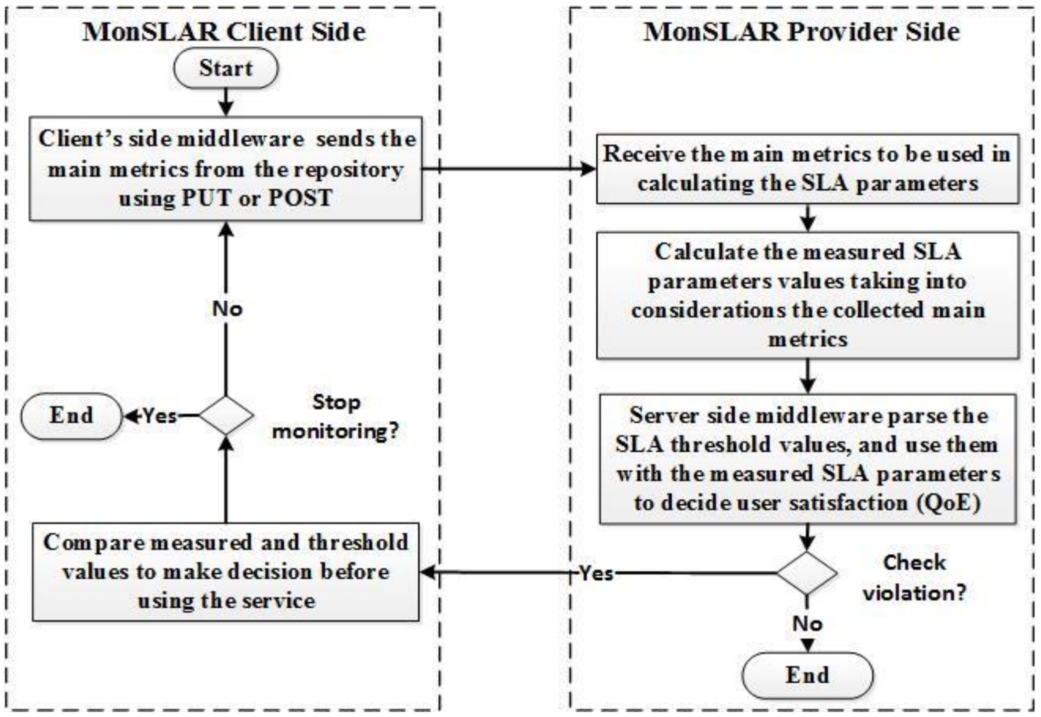

a)
MonSLAR Client Side
Start
Client's side middleware sends HEAD request
Receive responses for the HEAD request
Extract the number of pervious violations and display it to the client
End
MonSLAR Provider Side
The Server's side midllew are receives user's request
Retrieve the number of previous violations and embed it in the HTTP HEAD response and send it to the Client's side
b)
MonSLAR Client Side
Start
Client's side middleware sends HEAD request
Receive responses for the HEAD request
Extract the value of QoE and display it to the client
End
MonSLAR Provider Side
The Server's side midllew are receives user's request
Retrieve the QoE value and embed it in the HTTP HEAD response and send it to the Client's side

Figure 4-6 Handling (monitoring request-A) in MonSLAR using HEAD method: (a) Retrieve number of previous violation; (b) retrieve QoE value

Figure 4-7 introduces (monitoring request-B), the monitoring process includes sending metric updates from the client to the provider. This occurs in order to report client satisfaction to the provider; for example, whether a request is successful or a request has generated an error. This allows the client to check the violation cases while using the service, and report such violations through the middleware components. The middleware uses a database as a service on the provider side to be used for saving the parameters values and updating their values sent by the client side middleware.


MonSLAR Client Side
Start
Client's side middleware sends the main metrics from the repository using PUT or POST
No
End
Yes
Stop monitoring?
Compare measured and threshold values to make decision before using the service
MonSLAR Provider Side
Receive the main metrics to be used in calculating the SLA parameters
Calculate the measured SLA parameters values taking into considerations the collected main metrics
Server side middleware parse the SLA threshold values, and use them with the measured SLA parameters to decide user satisfaction (QoE)
Yes
Check violation?
No
End
Figure 4-7 Handling (monitoring request-B) in MonSLAR to detect violations in the received services

Using OPTIONS to manage monitoring request-C is shown in Figure 4-8, in this request the updated values of the measured parameters are retrieved from the repository of the server side to give the client a detailed information about the enhancement or degradation of the service if needed; this is achieved by delivering the values of the parameters used in estimating the QoE value.

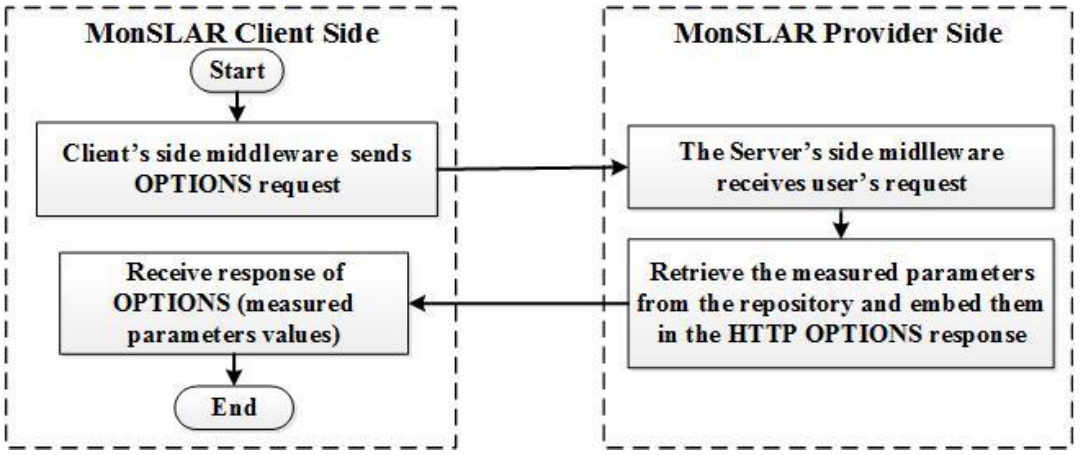

MonSLAR Client Side
Start
Client's side middleware sends OPTIONS request
Receive response of OPTIONS (measured parameters values)
End
MonSLAR Provider Side
The Server's side midllew are receives user's request
Retrieve the measured parameters from the repository and embed them in the HTTP OPTIONS response
Figure 4-8 Handling (monitoring request-C) in MonSLAR to retrieve the measured parameters using OPTIONS method

Figure 4-9 presents a collaboration diagram for using REST in MonSLAR's (monitoring request-A). In Figure (4.9-a), MonSLAR's client side sends an HTTP HEAD request asking for the number of previous breaches of SLAs by this provider which will be embedded in the HTTP HEAD response. The user decides whether to use this service or not depending on the number of previous violations. Figure (4.9-b) presents a detailed description for retrieving the QoE value using HEAD.

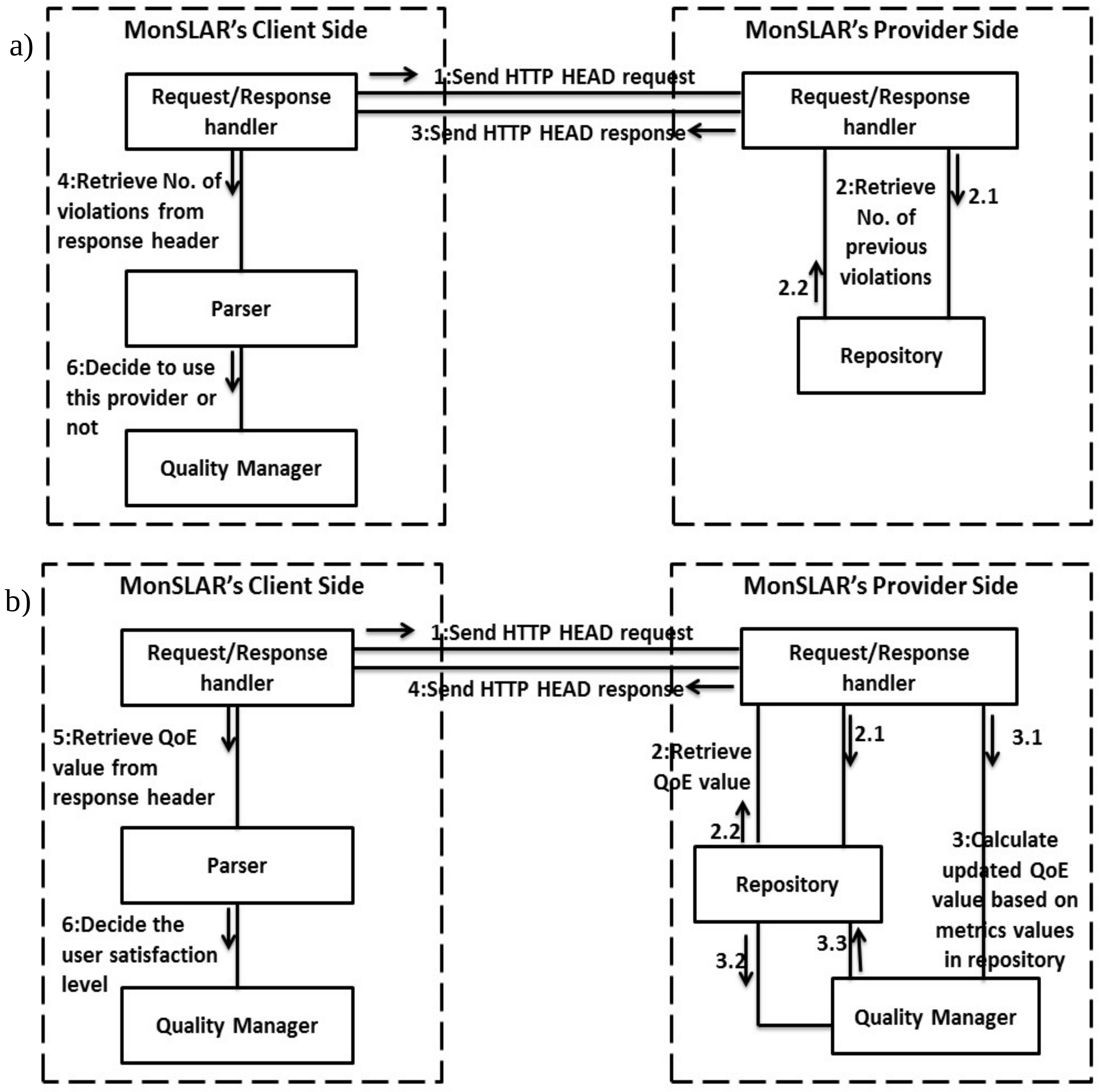


Figure 4-9 Using REST services in the MonSLAR's (monitoring request-A) using HEAD - collaboration diagram: (a) Retrieve previous violations; (b) Retrieve QoE value

Figure 4-10 presents a collaboration diagram for the (monitoring request-B). In this request, the values of the metrics to be used in calculating the SLA parameters should be sent from the client side to the provider side. To do this, the middleware uses a database as a repository service, and the PUT and POST methods are used for updating the database with the sent parameter values.

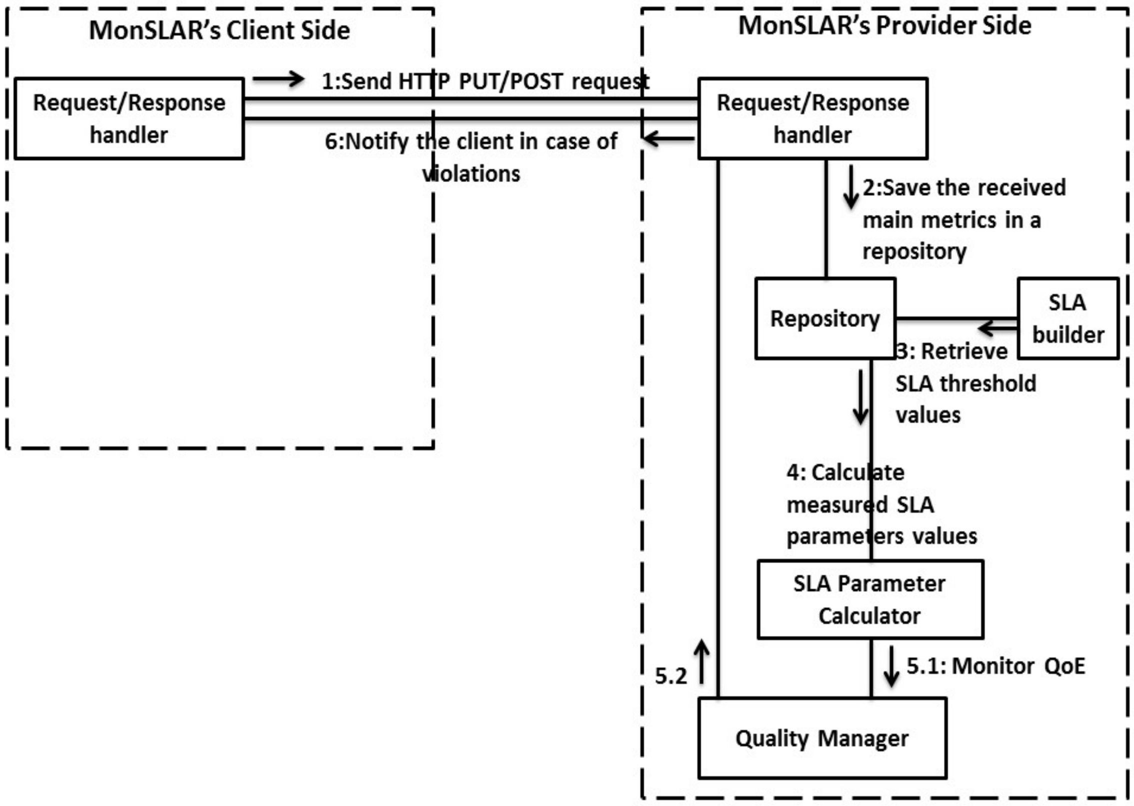

MonSLAR's Client Side
MonSLAR's Provider Side
Request/Response handler
1:Send HTTP PUT/POST request
6:Notify the client in case of violations
Request/Response handler
2:Save the received main metrics in a repository
Repository
SLA builder
3: Retrieve SLA threshold values
4: Calculate measured SLA parameters values
SLA Parameter Calculator
5.1: Monitor QoE
5.2
Quality Manager

Figure 4-10 Using REST services in the MonSLAR's (monitoring request-B) using PUT&POST - collaboration diagram

Figure 4-11 introduces a detailed illustration for using OPTIONS method in (monitoring request-C).

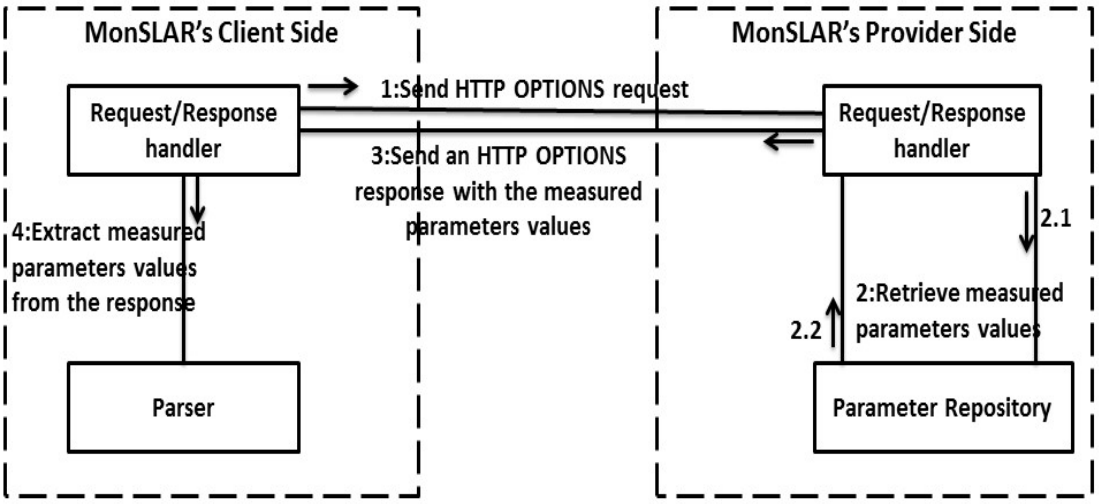

MonSLAR's Client Side
MonSLAR's Provider Side
Request/Response handler
1:Send HTTP OPTIONS request
3:Send an HTTP OPTIONS response with the measured parameters values
Request/Response handler
4:Extract measured parameters values from the response
Parser
2.1
2:Retrieve measured parameters values
2.2
Parameter Repository

Figure 4-11 Using REST services in the MonSLAR's (monitoring request-C) using OPTIONS - collaboration diagram

4.4.3 Management Request

Figure 4-12 demonstrates handling the management requests by MonSLAR. This process includes embedding the values of the parameters specified in the SLA and updating the middleware with the new values in case the SLA document has been updated. As stated earlier, the HTTP method used in this request is the OPTIONS method.

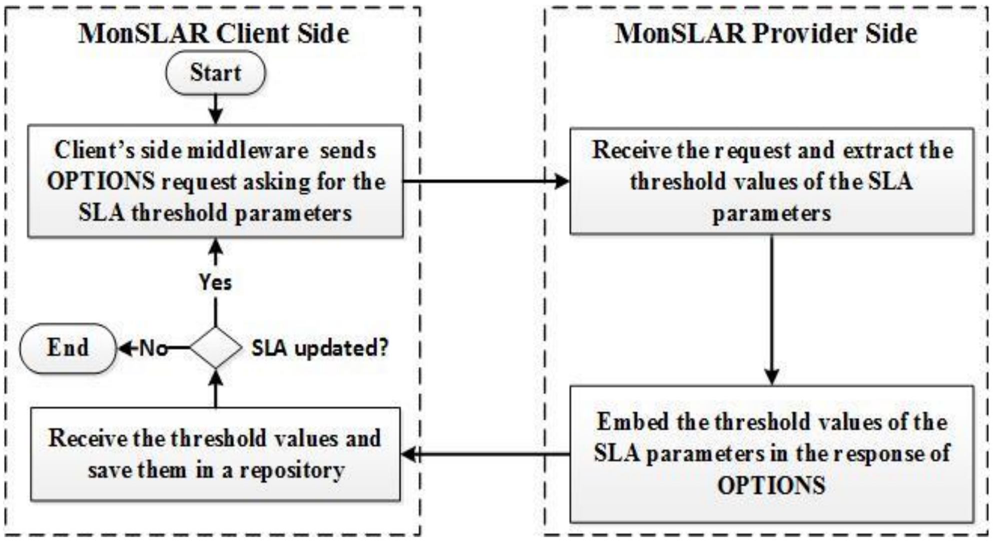

MonSLAR Client Side
Start
Client's side middleware sends OPTIONS request asking for the SLA threshold parameters
Yes
End
No
SLA updated?
Receive the threshold values and save them in a repository
MonSLAR Provider Side
Receive the request and extract the threshold values of the SLA parameters
Embed the threshold values of the SLA parameters in the response of OPTIONS

Figure 4-12 Handling the management request in MonSLAR

Figure 4-13 presents a collaboration diagram for using the HTTP OPTIONS method to retrieve the SLA threshold parameters specified in the SLA document to update the client's side with their values in case of any update.

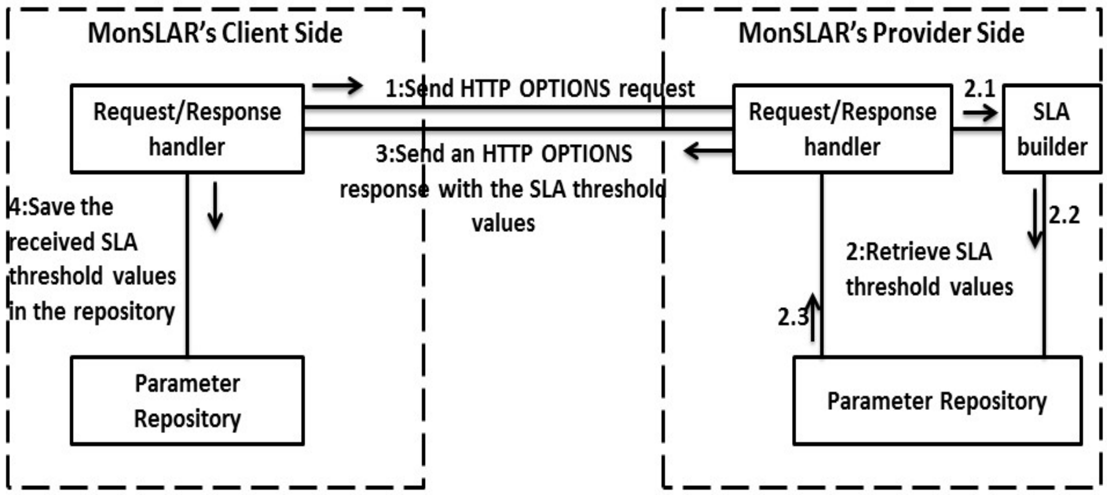

MonSLAR's Client Side
Request/Response handler
4:Save the received SLA threshold values in the repository
Parameter Repository
1:Send HTTP OPTIONS request
3:Send an HTTP OPTIONS response with the SLA threshold values
MonSLAR's Provider Side
Request/Response handler
2.1
SLA builder
2.2
2:Retrieve SLA threshold values
2.3
Parameter Repository

Figure 4-13 Using REST services in the MonSLAR's management request (collaboration diagram)

4.5 Embedding Monitored Data in REST Services

As discussed in Section 3.4, the SLA parameters and measurements of QoS are represented in SOAP web services by exploiting the WSDL document to involve this kind of data. However, WSDL documents cannot be used in REST web services, and because of the spread of REST in the current implementation of cloud computing environments, these are motivations for defining a new approach for implementing the SLA parameters and measurements in the REST architecture. This approach is used in MonSLAR to manage the SLA document and QoS measurements.

The framework exploits existing HTTP methods to represent the SLA parameters and to exchange monitoring data by defining custom HTTP headers for the number of violations and for each SLA parameter with its name and value as shown in Figure 4-14 and Figure 4-15. This simplifies the process of exchanging SLA parameters between the provider and the client. In the case where the client is not interested in asking for a service and the client side middleware needs to update the middleware environment or to retrieve the number of violations or QoE in MonSLAR, the middleware can send an HTTP HEAD or OPTIONS request which is useful in retrieving the required data.

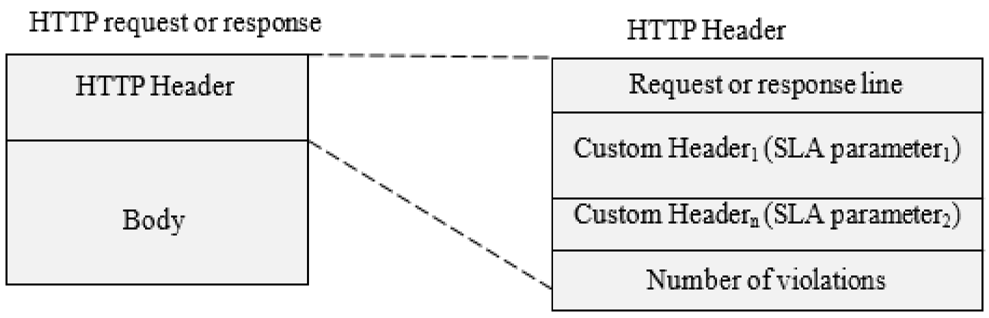


Figure 4-14 Extending HTTP headers to include SLA parameters

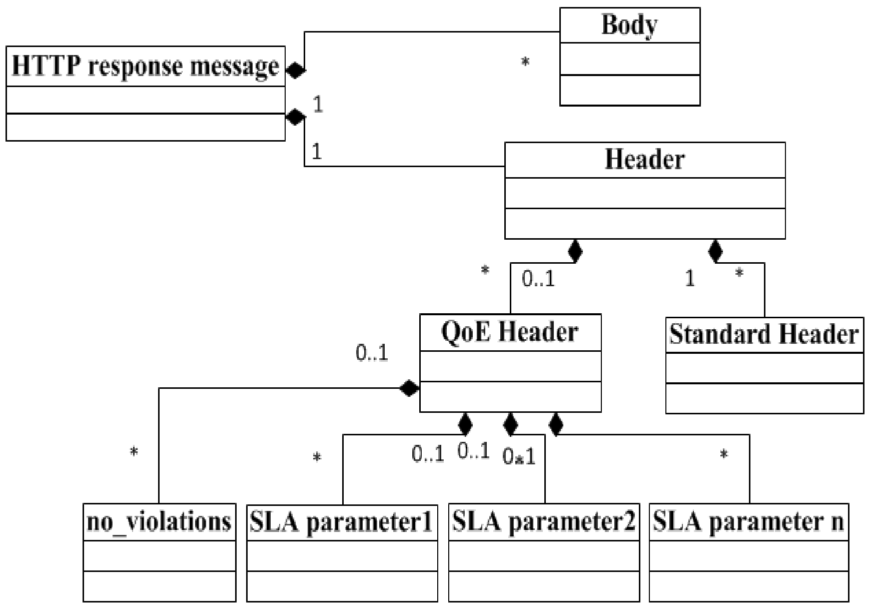


Figure 4-15 A UML diagram for embedding SLA parameters and monitored data within the HTTP message

4.6 Deploying MonSLAR in Multi-tenancy of SaaS

The concept of multi-tenancy of SaaS is supported by MonSLAR. According to Leymann, et al. (2014), designing SaaS includes three levels of multi-tenancy: shared component, tenant-isolated component, and dedicated component (Leymann et al., 2014). These levels were taken into consideration in deploying MonSLAR in multi-tenancy. These cases have been presented in order to show the feasibility of handling the multi-tenancy concept by MonSLAR.

The first case considers the shared component and tenant-isolated component as shown in Figure 4-16. The figure shows that the application server contains different application instances for each tenant, with a shared database that supports the metadata of SaaS tenants. In this case, MonSLAR is deployed in the application server with a database for MonSLAR, the isolation among the tenants in the database is managed through the use of a dedicated tenant-id.

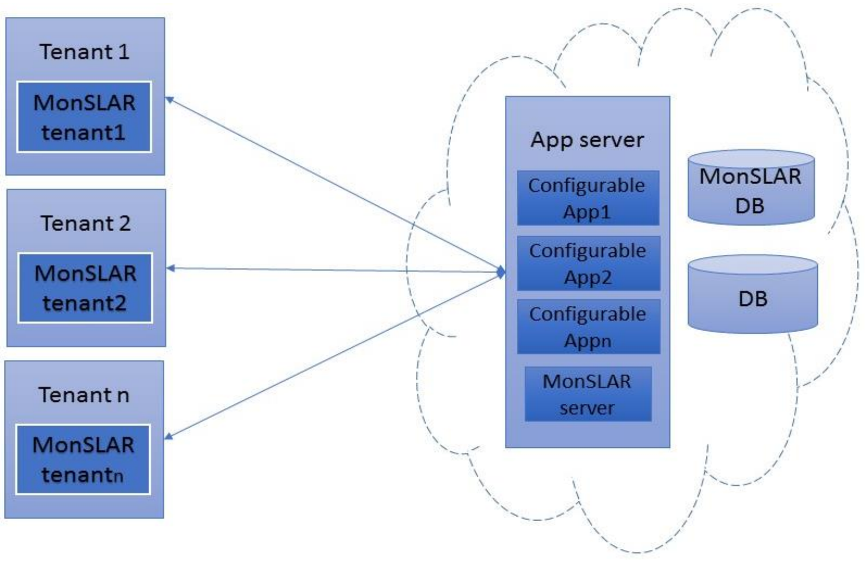


Figure 4-16 MonSLAR deployment in multi-tenancy- (shared component, tenant-isolated component)

Figure 4-17 depicts supporting the dedicated component multi-tenancy by MonSLAR. The figure shows that each tenant has its own application with a separated database for each application to save the data related to the specific tenant. MonSLAR, in this case, is deployed at the server side of each tenant with a database (MonSLAR DB) to save the measurements and the data related to the user satisfaction.

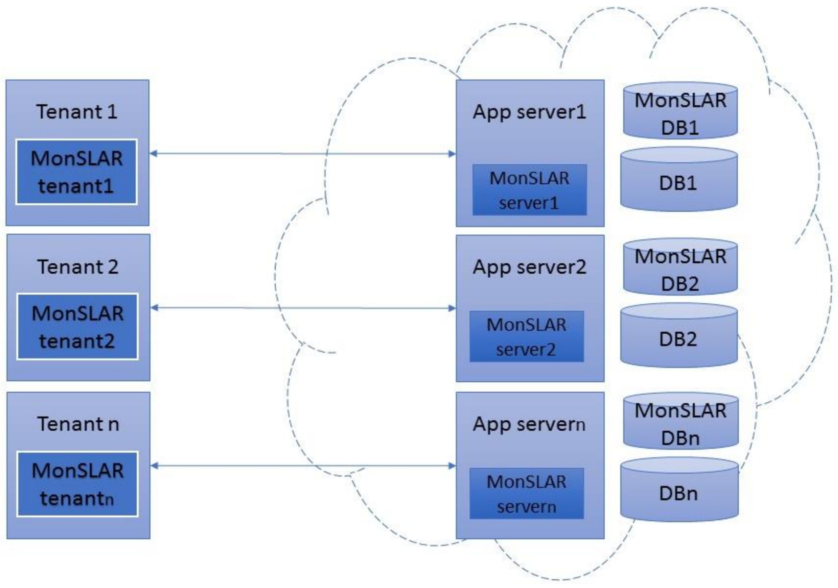


Figure 4-17 MonSLAR deployment in multi-tenancy - (dedicated component)

4.7 Chapter Summary

This chapter introduced MonSLAR as a proposed solution for monitoring the SLA in SaaS cloud computing taking into consideration the client's perception. The proposed middleware provides the following features:

1- SOM: MonSLAR is an SOM middleware which supports features like loose coupling, reusability for the monitoring service and being platform independent. This feature allows the provision of the monitoring process as an add-on service.
2- Using REST: REST technology is used for managing the communication between the client side and the provider side, which is an alternative to using SOAP technology as proposed in existing solutions. This includes finding ways for embedding the monitored data in REST messages and reduces the need for extra messages to exchange this data. HEAD and OPTIONS methods are suggested for transmitting the monitored data. The use of REST technology adds a lightweight characteristic to the proposed middleware.
3- User-centric: The use of a user-centric monitoring gives the user more control over the management of the monitoring process, which helps maintain confidence between the client and the service provider.

These aforementioned characteristics of the proposed middleware are tested and evaluated in Chapters Six and Seven respectively.

The following chapter presents the use of a fuzzy logic engine as an approach for estimating user satisfaction in terms of QoE value.

CHAPTER FIVE

ESTIMATING QoE OF SaaS

5.1 Introduction

Chapter Four presented a description of the middleware for monitoring an SLA in cloud computing. It is important and yet challenging in a cloud monitoring system to use a QoE estimation method, which can take advantage of a large amount of collected monitored data, and express it as essential information to the user (Lu et al., 2016), taking into consideration the difficulty of understanding and analysing the measurements in cloud computing by the user (Shao & Wang, 2011). This chapter introduces an approach for estimating QoE in SaaS services based on the monitored data. Monitoring the quality of cloud computing services from a user's perspective is gaining more recognition in the research field, the relationship between the client and the provider is managed by a signed contract, which is expressed as an SLA. From the user's point of view, it is very important to estimate the value of a QoE as a measure of a user's satisfaction with the perceived services. However, there is a lack of general definition of a holistic metric for estimating the QoE of SaaS services in cloud computing, and this omission highlights the importance of finding a unified and general measure for QoE. To achieve this goal, the SaaS-Qual model reviewed in section 3.2.2.1 is used as a measure of user's satisfaction in cloud computing.

This chapter presents a design of a fuzzy logic inference engine to estimate the value of QoE through considering the SaaS-Qual as a quality model for estimating user satisfaction in SaaS cloud computing; it also discusses the possibility of using it as a unified metric for QoE in SaaS. By mixing the concept of estimating user satisfaction based on the parameters of SaaS-Qual with the concept of measuring QoE in terms of SLA parameters and the QoS of the network, this helps to define a new estimation of a QoE.

5.2 Defining a Metric for Estimating QoE

As SaaS is an internet based application, it is important to take into consideration the impact of the network as well as the impact of running the application of SaaS on the provider's side in defining QoE for SaaS. Similar to end-to-end QoS for a network, QoE for SaaS can be defined as a function of three quantities based on the time taken to provide the service to the client. This can be explained by defining three regions depending on the level of delivering the service to the client as shown in Figure 5-1. The three regions are denoted as T_P, T_X, and T_C where:

- T_P: preparation time, the time consumed at the server side preparing the application to be used by the client.
- T_X: transmission time, the time delay of the network transmission, which represents the QoS of a network.
- T_C: consuming time, the time delay using the application by the client.

Thus, QoE is not only related to the consumption of the service at the client's side, but also the effect of the network's operating conditions. Therefore, according to this model and in case the cloud provider is not responsible for the delivery of the network, it would be unfair to penalise the cloud provider in the case of SLA violations due to degradation in the network's QoS.

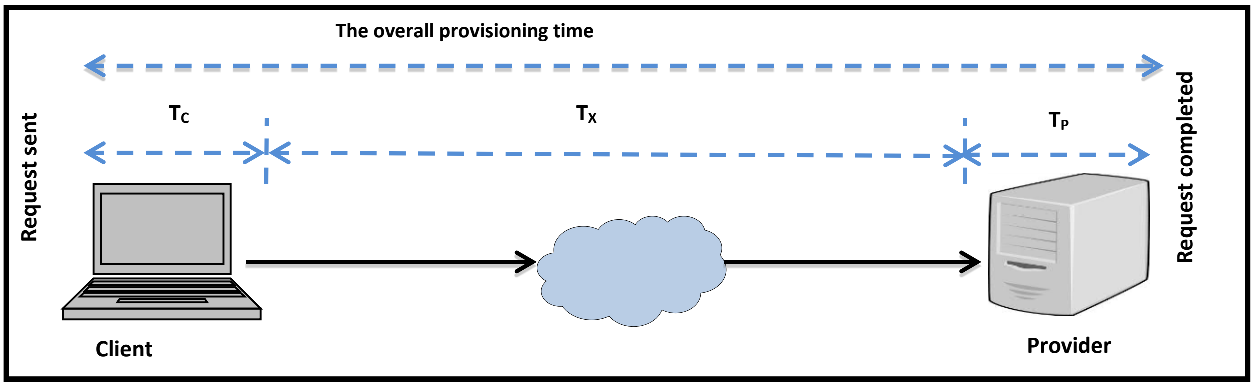


Figure 5-1 End-to-End QoE for SaaS

By measuring these KPIs with corresponding QoS parameters and aggregating the values of QoE taking into consideration values of the SLA parameters determined in the SLA, any violation of the SLA terms can be determined.

Knowing that QoE is a function of QoS from Eq.5.1 (Fiedler et al., 2010), in other words, $QoE_{network}$ is a function of the $QoS_{network}$ as shown in Eq.5.2

$QoE = f(QoS)$ ………………………………………………………….. (5.1)

$QoE_{network} = f(QoS_{network})$ ………………………………………………….. (5.2)

In order to define QoE of SaaS, which is an internet based application, it is important to consider the QoE for the application itself as well as the QoE of the underlying network. Eq.5.3 is specified to take into account the QoE for the software provided to the client as well as the QoS of the network as shown in Eq.5.4

$QoE_{SaaS} = f(QoE_{SW}, QoE_{network})$ ……………………………………………. (5.3)

$QoE_{SaaS} = g(QoE_{SW}, QoS_{network})$ …………………………………………….. (5.4)

Assuming that QoE_{SW} represents the negotiated SLA parameters in the SLA signed by SaaS provider and client, Eq.5.5 can be shown as:

$QoE_{SaaS} = F(SLA_{parameters}, QoS_{network})$ ……………………………………… (5.5)

The above equations stand in the case that the SaaS provider is the same as the network provider [SaaS provider = Network provider]; in this case the SLA will be achieved for both providers, in other words, the defined parameters will satisfy the network's QoS as well as the SLA parameters for the specified SaaS (Al-Shammari & Al-Yasiri, 2014).

When the SaaS provider is not the same as the network provider [SaaS provider ≠ Network provider], the SLA parameters are not expected to cover the network's operating conditions, and hence a decision support technique needs to be developed in order to tackle this dissimilarity.

Therefore, it is important to develop an approach for measuring QoE_{SaaS} in terms of $SLA_{parameters}$, and $QoS_{network}$, which represents function F. This approach is explained in more details in Section 5.4.

5.3 Estimating QoE in Terms of User Satisfaction

The success of cloud computing services depends mainly on the clients' satisfaction about the provided services taking into consideration the QoS defined in the SLA (Badidi, 2013). As discussed in Section 5.2, QoE was defined as the overall satisfaction about received services in a cloud computing environment, and can be considered as a function of both SLA parameters and a $QoS_{network}$ (Al-Shammari & Al-Yasiri, 2014). In other words, measured SLA parameters as functions of a QoS can be used to estimate the QoE or user satisfaction. However, in order to estimate the value of QoE of SaaS services in terms of user satisfaction, it is important to define the main criteria to quantify this value. To do so a unified and general model is required which considers the main characteristics of SaaS services in cloud computing, such a model to be used as an index for benchmarking SaaS services in cloud computing. As mentioned previously in the literature review, many models have been proposed as an attempt to estimate the services in cloud computing in general and SaaS services specifically. SaaS-Qual is considered as one of the most cited pieces of research in this field (see Section 3.2.2.1). SaaS-Qual introduced a set of six factors that affect usage continuance of SaaS, in addition to the weights of each of these parameters. These weights helped in the decision making and building the rules of the fuzzy engine.

A methodology is used to study SaaS-Qual as a model for estimating QoE value as shown in Figure 5-2. This includes studying the effect of the SaaS-Qual parameters on QoE by presenting a fuzzy logic engine (see section 5.4). Then, these results were analysed and amendments are used to adjust the proposed fuzzy engine as presented in Chapter 7 (see section 7.5).

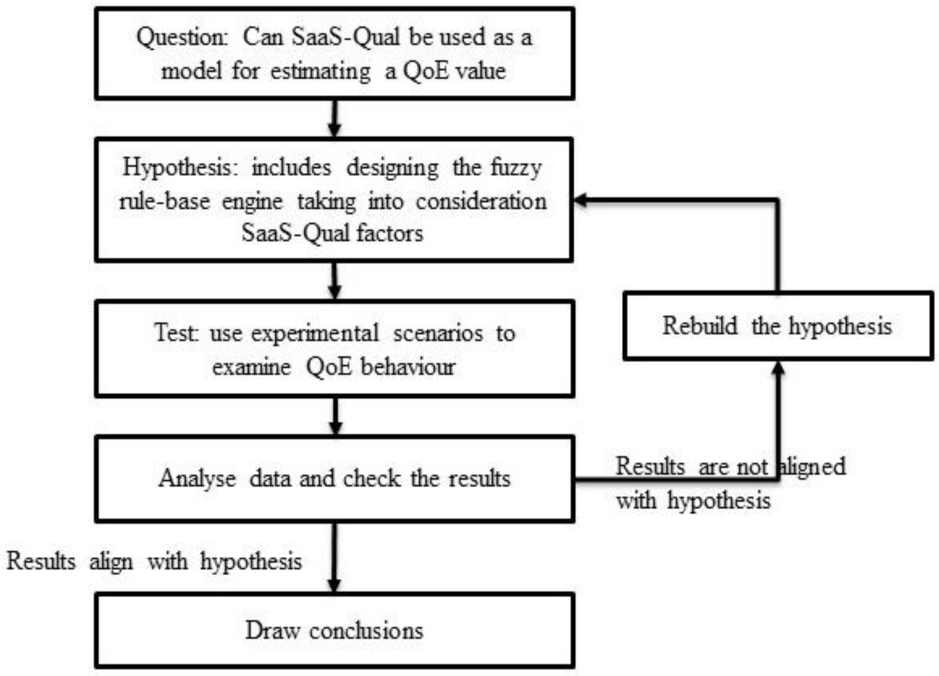

Question: Can SaaS-Qual be used as a model for estimating a QoE value
Hypothesis: includes designing the fuzzy rule-base engine taking into consideration SaaS-Qual factors
Test: use experimental scenarios to examine QoE behaviour
Analyse data and check the results
Rebuild the hypothesis
Results are not aligned with hypothesis
Results align with hypothesis
Draw conclusions

Figure 5-2 Methodology for studying SaaS-Qual to estimate QoE

5.4 Estimating QoE Value Using Fuzzy Logic

As discussed in section 5.3, SaaS-Qual is proposed as a model for estimating QoE. An approach is required to estimate the value of QoE taking into consideration its parameters weights. A rule-based Mamdani fuzzy logic inference engine is used to estimate the QoE value, where fuzzy logic was defined by Zadeh as "… a precise logic of imprecision and approximate reasoning" (Zadeh, 2008, p. 2751). Fuzzy systems are used for estimation and decision making (McNeill & Thro, 1994), it is based on linguistic rules which make the computers' reasoning closer to that of human (Jantzen, 2013). The choice of fuzzy logic is based on the characteristics of the QoE as a subjective perception of the user to the provided services (Hobfeld et al., 2012), in addition to the difficulty of deciding a user's satisfaction based on the changes in the values of the SLA parameters. For example, there is no specific value that can be used to describe that the user is satisfied or dissatisfied with the overall perceived services in SaaS. Furthermore, estimating the value of QoE based on SaaS-Qual model involves a level of uncertainty in the form of vagueness and impreciseness, where the value of QoE is affected by a set of different parameters with different priorities; fuzzy logic was chosen for its ability of handling uncertainty (Ross, 2009). This uncertainty is not described by a mathematical model that can be used for calculating the QoE value based on the SLA parameters. It is the best approach used for handling uncertain and imprecise values, which is the case for estimating user satisfaction, based on the SLA parameters. Users are usually uncertain about their satisfaction about measured SLA parameters, in addition to the fact that it is difficult to decide the exact value of the measured SLA parameter that can be considered acceptable or not. Because of the aforementioned reasons and the fuzzy logic characteristics, fuzzy logic is the best approach that can work efficiently for this type of application for estimating the QoE value in MonSLAR.

The proposed fuzzy system accepts the deviation of the measured values of the SLA parameters from the threshold values as inputs, (where the deviation is the weighted difference between the measured value and the required value as specified in the SLA document); it then generates the fuzzy rules according to the weights of each SLA parameter as decided in the SaaS-Qual model (Benlian et al., 2011). The result of this process is the measured QoE value, which is used as an indicator for the user satisfaction and violations in the SLA document. The architecture of the fuzzy logic engine is shown in Figure 5-3.

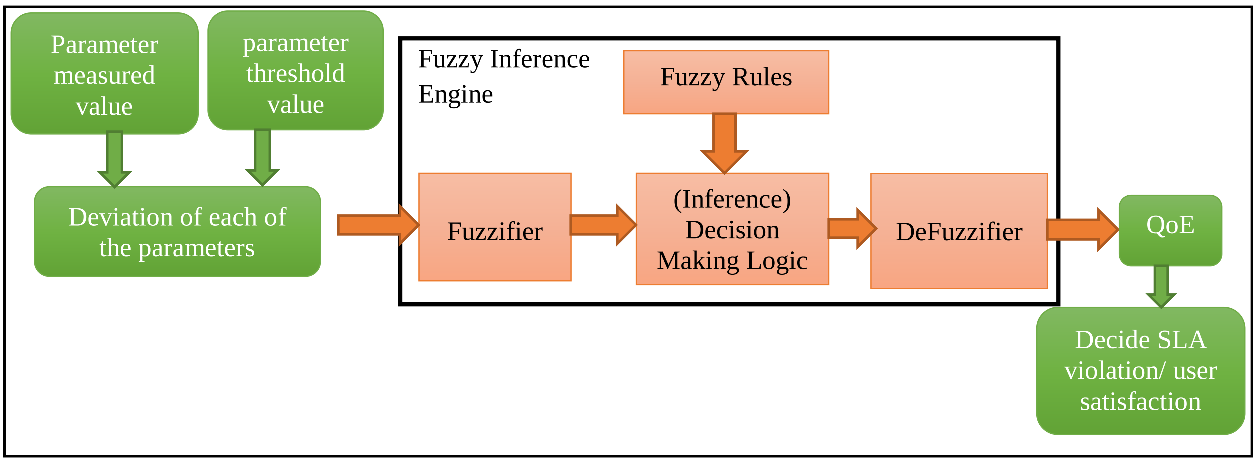

Parameter measured value
parameter threshold value
Deviation of each of the parameters
Fuzzy Inference Engine
Fuzzy Rules
Fuzzifier
(Inference) Decision Making Logic
DeFuzzifier
QoE
Decide SLA violation/ user satisfaction

Figure 5-3 The proposed fuzzy logic engine, adapted from (Mendel, 1995)

The deviation value is used as an input to the fuzzy engine, to make sure that both the actual measured values and the threshold values of the SLA parameters are considered in the estimation of the QoE; this helps to detect a violation in the SLA document. The deviation value of each of the parameters can be obtained by using the following formula:

$$\text{Parameter deviation} = \left|\frac{\mathit{Parameter\ measured} - \mathit{parameter\ threshold}}{\mathit{parameters\ threshold}}\right| * 100\% \quad \ldots\ldots\ldots\ldots \quad (5.6)$$

Algorithm 5-1 presents a pseudo code that describes the procedure used for evaluating QoE in terms of SLA parameters.

```
Algorithm 5-1: Estimating QoE value
Input: SLA threshold parameters, SLA measured parameters.
Output: QoE value.
1   Initialize algorithm variables
2       Retrieve SLA parameters threshold values from SLA document
3       Retrieve updated SLA measured parameters values from the database
4       For each value of measured parameters
5           Calculate parameter deviation using equ.5.6
6       End for
7       Map parameters deviation values into membership functions.
```

```
8        Apply fuzzy rules to input membership functions
9        Defuzzify output fuzzy set.
10   return QoE
```

5.4.1 The Input/Output Design of the Fuzzy Inference Engine

In order to map the crisp input data to the linguistic variables in the fuzzy engine, the membership functions for the input parameters and the output QoE are defined, where the membership function is “a curve to describe how the variable in fuzzy is mapped in a degree of membership between 0 and 1” (Pilevari et al., 2013). According to (Ross, 2009), different methods can be used to assign the membership functions’ as values in fuzzy systems. These methods are: (1) intuition, which is derived from humans’ ability to derive the membership functions based on their understanding and intelligence; (2) inference, in this method, knowledge and facts are used to implement a deductive reasoning; (3) Rank ordering, preferences is identified by research comparisons in this method; (4) Neural networks, this method includes using training data to train a neural system; (5) Genetic algorithms; and (6) Inductive reasoning, this method is not appropriate when the used data is dynamic (Ross, 2009). In this research, the fact that there is no specific value for the levels of the high-deviation and low-deviation of the SLA parameters was the motivation for adopting the intuition method because of its suitability to solve this kind of questions. Opinions of experts in the field are considered in defining the membership functions in the proposed fuzzy engine.

A membership function with a range of input values [0.0, 100.0] is used. This range is specified depending on the values of the deviation of the parameters obtained by equ.5.6, three linguistic terms are used to describe the input to the fuzzy engine which are (high-deviation, fair, and low-deviation) as shown in Figure 5-4. The membership function of the output is trapezoidal, the range used is [0.0, 5.0], which considers five linguistic terms (bad, poor, fair, good, and excellent) as shown in Figure 5-5. This range is considered realistic as the QoE is usually measured using survey studies, these studies involves asking the users about quality of services in means of scalers values (MOS).

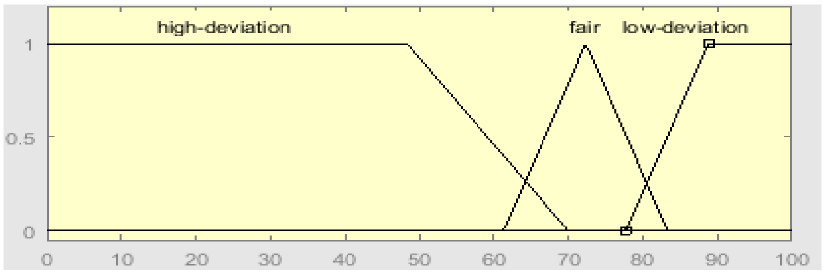


Figure 5-4 Membership function for the fuzzy input

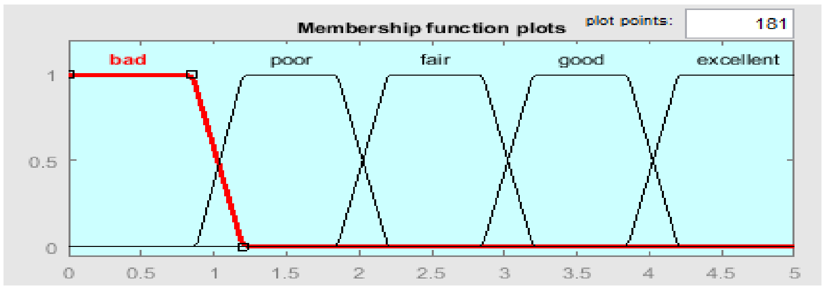


Figure 5-5 Membership function of QoE (output)

5.4.2 The Inference Rules (Rule Selection)

Fuzzy rules are submitted in the form of (IF … THEN), which represents human empirical knowledge. In the proposed fuzzy engine, the rules are derived based on the submitted study by Benlian et al, 2011. The rules are derived taking into consideration the weights of the parameters (see Section 3.2.2.1). The generated fuzzy rules take into consideration all the possible values (high-deviation, fair, and low-deviation) for the six inputs, in other words (3^6 = 729 rules). Figure 5-6 introduces an excerpt of the generated fuzzy rules used for estimating a QoE value.

IF Features **IS** high-deviation **AND** Responsiveness **IS** high-deviation **AND** Flexibility **IS** high-deviation **AND** Security **IS** high-deviation **AND** Rapport **IS** high-deviation **AND** Reliability **IS** high-deviation **THEN** QoE **IS** Bad

IF Features **IS** high-deviation **AND** Responsiveness **IS** high-deviation **AND** Flexibility **IS** high-deviation **AND** Security **IS** fair **AND** Rapport **IS** fair **AND** Reliability **IS** fair **THEN** QoE **IS** Poor

IF Features **IS** high-deviation **AND** Responsiveness **IS** high-deviation **AND** Flexibility **IS** high-deviation **AND** Security **IS** low-deviation **AND** Rapport **IS** low-deviation **AND** Reliability **IS** fair **THEN** QoE **IS** Fair

IF Features **IS** low-deviation **AND** Responsiveness **IS** low-deviation **AND** Flexibility **IS** low-deviation **AND** Security **IS** Good **AND** Rapport **IS** high-deviation **AND** Reliability **IS** high-deviation **THEN** QoE **IS** Good

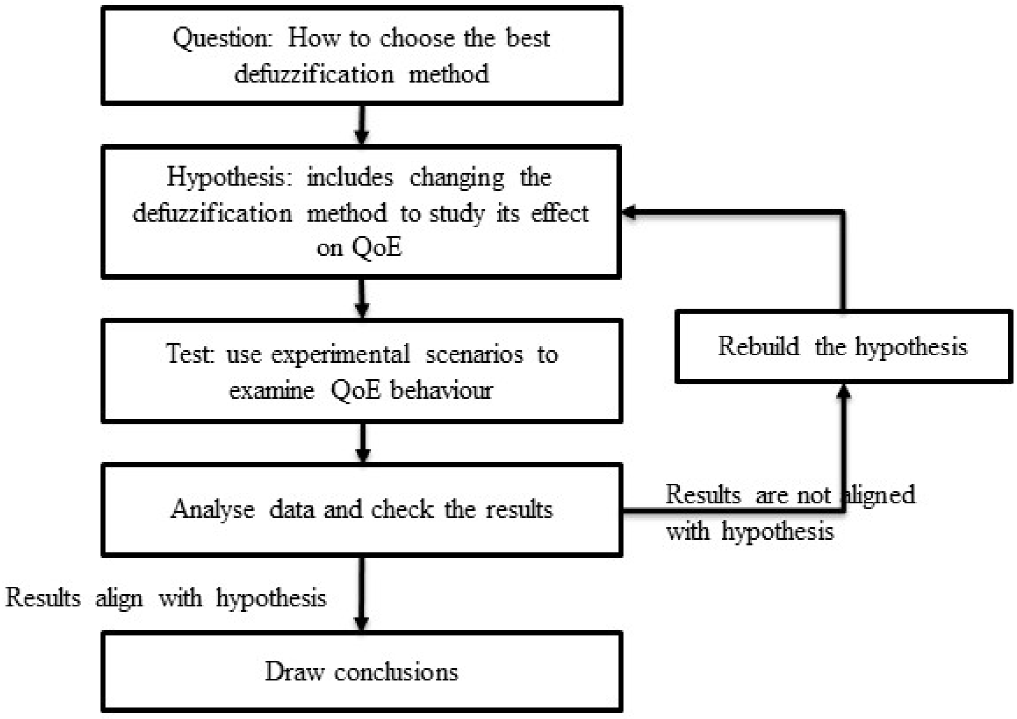
IF Features **IS** low-deviation **AND** Responsiveness **IS** low-deviation **AND** Flexibility **IS** low-deviation **AND** Security **IS** low-deviation **AND** Rapport **IS** low-deviation **AND** Reliability **IS** low-deviation **THEN** OoE **IS** Excellent

Figure 5-6 Excerpt of the fuzzy rules

5.4.3 The Defuzzification Method

Selecting the defuzzification method was based on a methodology to study the effect of the different defuzzification methods and decide the best method for the proposed fuzzy engine, a similar study presented to study the effect of defuzzification methods on Fuzzy results (Naaz et al., 2011).

In this study, many experimental scenarios have been studied to observe the effect of changing the defuzzification method on the estimated QoE value. Five different scenarios have been used to examine the behaviour of the QoE measurement system as a result of changing the defuzzification method Figure 5-7. The centroid method, Mom (Medium of Maximum), LOM (Largest of Maximum), and SOM (Smallest of Maximum) are used as defuzzification method for the first, second, third, fourth, and fifth scenarios respectively. Please refer to Appendix B for more details.



Question: How to choose the best defuzzification method
Hypothesis: includes changing the defuzzification method to study its effect on QoE
Test: use experimental scenarios to examine QoE behaviour
Rebuild the hypothesis
Analyse data and check the results
Results are not aligned with hypothesis
Results align with hypothesis
Draw conclusions

Figure 5-7 Methodology for implementing the defuzzification method

According to Ross (2009), the criteria for selecting the defuzzification method are: **continuity**, which means that small changes in input values should not cause a significant change in an output value; **disambiguity**, which means the output to have one single value; **plausibility**, which means to have a high degree of a result function;

computational simplicity, which means a less computational time needed to do computations; and **weighted method**, which depends on the problem itself (Ross, 2009).

The used method for the defuzzification is the centroid method. Eq.5.7 is used to calculate the output of the defuzzification process.

$$y = \frac{\int \mu_i(x) \bullet x dx}{\int \mu_i(x) dx} \quad \text{……………………………..……….} \quad (5.7)$$

Where: y is the output, µi(x) is the assembled membership function.

The other methods caused the value of the QoE value to drop, while in the centroid method, the best results of QoE were achieved. This is because of the shape of the used membership functions, in addition to having six input parameters. These reasons caused the result of the fuzzification process (inference system) to have shape characteristics, which may not suit the other defuzzification methods. For example: in the Maxima method: there should be one single maximum point, the weighted average method where membership functions should be symmetrical.

5.5 Effect of the SaaS-Qual Factors on the QoE Value

The behaviour of the fuzzy inference engine is described by using surface plots to show the effect of each two of the SLA parameters on the QoE value. The significance of the diagrams is to show the importance of the six parameters in estimating the user satisfaction about SaaS. Given the difficulty of including all the six parameters in the same diagram, the study considered each of two parameters in a separate diagram to highlight the different results in each case.

Figure 5-8 and Figure 5-9 show the effect of both (Responsiveness, Features) and (Security, Features) parameters' pairs on the QoE value respectively. It is apparent that in these two figures, the Responsiveness and Security have a higher effect on the QoE than does Features parameter especially in the [low-deviation low-deviation] and [fair, fair] pairs, which cause the QoE value to reach 1.5.

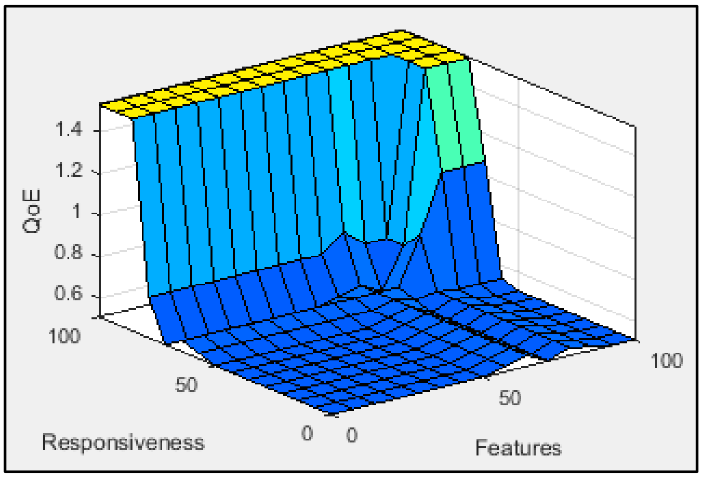


Figure 5-8 Effect of Features and Responsiveness

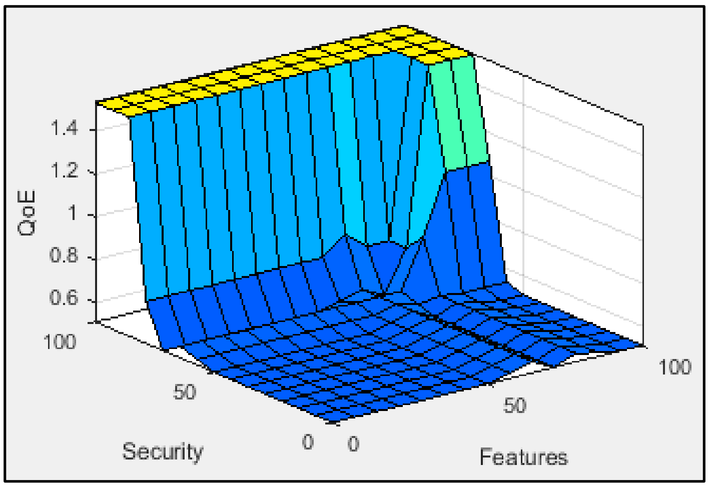


Figure 5-9 Effect of Features and Security

Figure 5-10 shows that both Flexibility and Features have a comparable effect on the QoE value, where the increase in each of these parameters causes a clear increase in the QoE value, especially in the [low-deviation low-deviation] pair. However, the effect of the Features parameter is higher on the [fair, fair] pair.

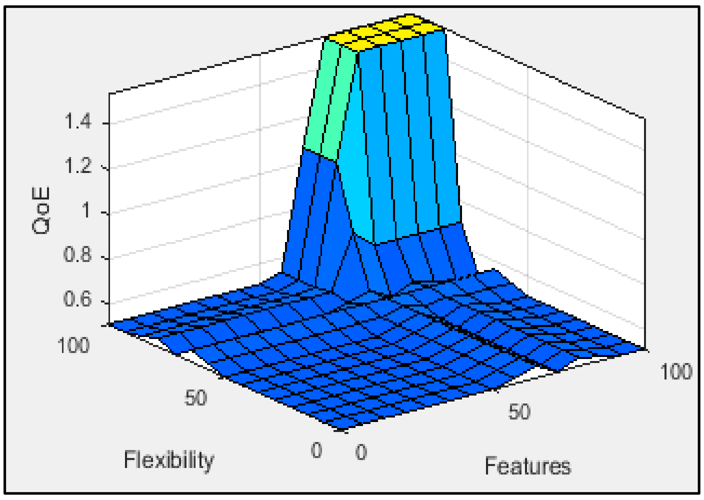


Figure 5-10 Effect of Features and Flexibility

Figure 5-11 and Figure 5-12 show that (Features - Rapport) and (Features - Reliability) pairs cause the maximum value for QoE to reach (0.56). In these figures, there is a clear fluctuation in the value of QoE. While the [high-deviation, high-deviation] pairs caused the QoE to be 0, the [fair, fair] pair increased the QoE value to reach 0.56 and to decrease again in the [low-deviation, low-deviation] pair to reach 0.54. The reason for this is not clear, but it may have something to do with the membership functions of the fuzzy input.

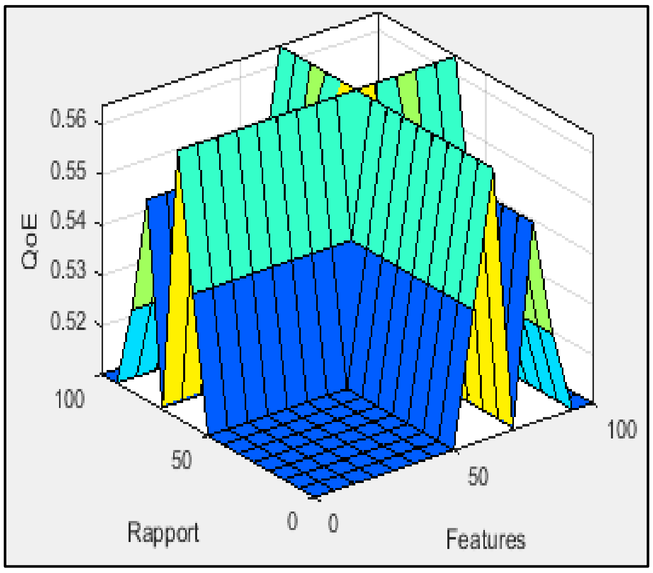


Figure 5-11 Effect of Features and Rapport

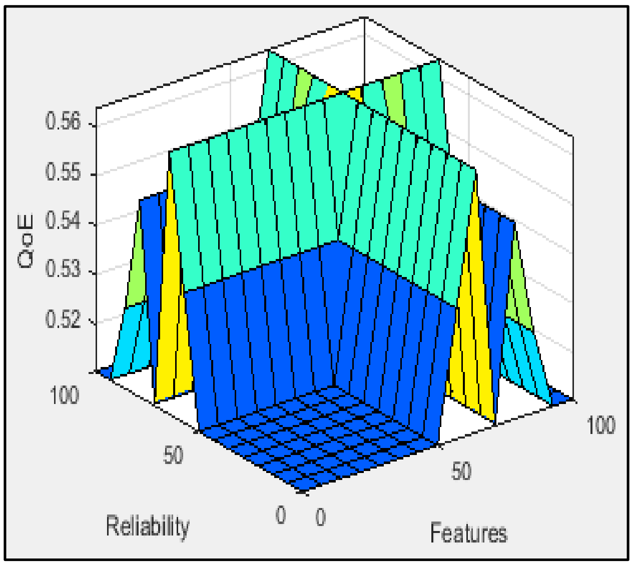


Figure 5-12 Effect of Features and Reliability

Figure 5-13 shows that both Responsiveness and Flexibility parameters have comparable effects on the value of QoE especially in the [low-deviation, low-deviation] pair which cause the QoE value to be 2.5. However, Responsiveness shows a higher effect on the [fair, fair] pair of the input membership function.

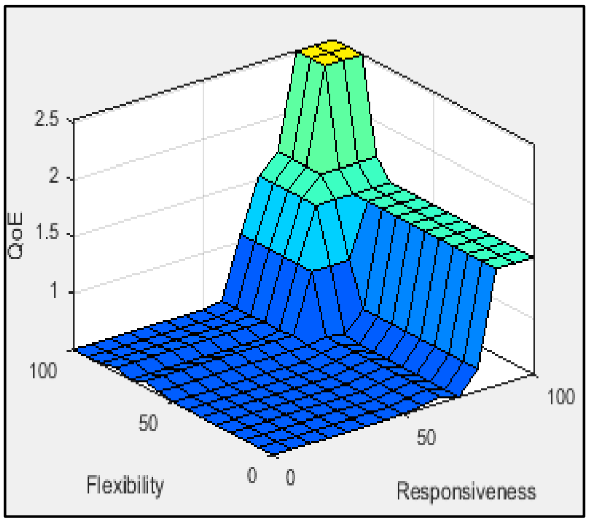


Figure 5-13 Effect of Flexibility and Responsiveness

Similarly, Figure 5-14 shows similar behaviour for the Rapport and Responsiveness parameters on the QoE value, which have the same influence on QoE.

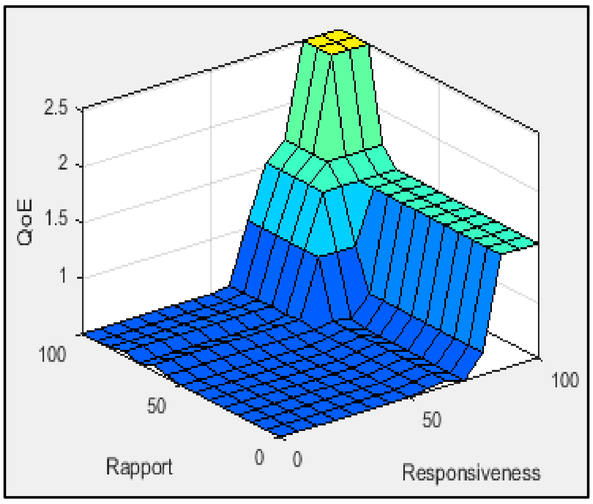


Figure 5-14 Effect of Rapport and Responsiveness

Figure 5-15 shows the effect of both Security and Responsiveness on the QoE value, it is clear that the pair of [high-deviation, high-deviation] of the fuzzy input results in a neglected value of QoE value ≈ 0, while their fair range causes a significant increase in QoE to reach 1.5. The highest values of both of them cause a dramatic increase in QoE value to reach 2.5.

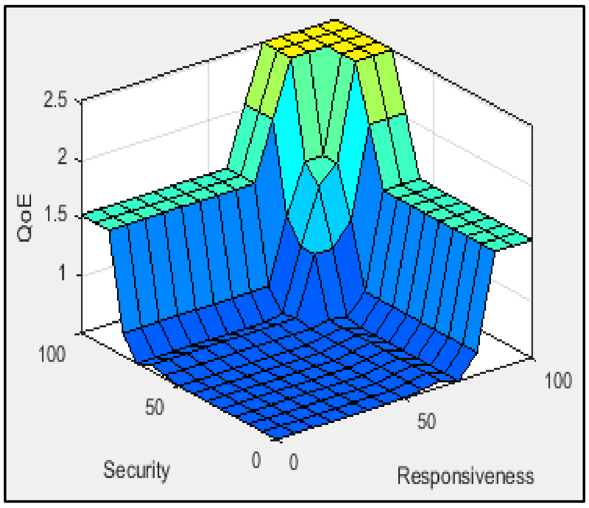


Figure 5-15 Effect of Security and Responsiveness

Figure 5-16 shows the effect of both Reliability and Responsiveness on the QoE value, it is clear that the pair of [high-deviation, high-deviation] like in the other parameter pair cases, results in a neglected value of the QoE value, but the [fair, fair] pair causes a dramatic increase in the output especially Responsiveness which has a clear effect in this region of the plot, all in all, the highest level achieved for QoE is 1.5.

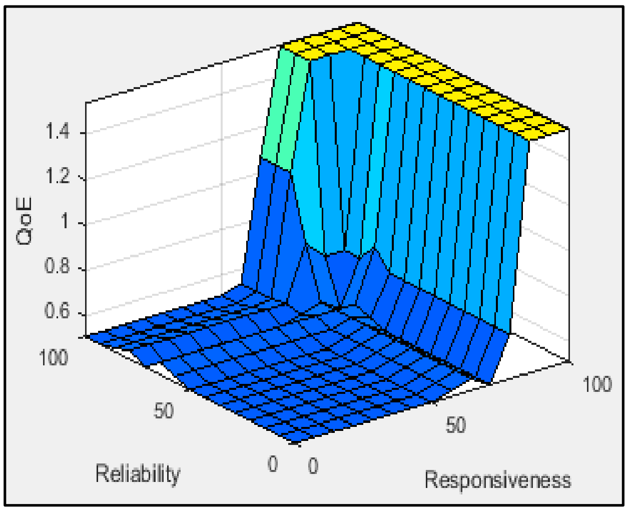


Figure 5-16 Effect of Reliability and Responsiveness

Figure 5-17 shows that both Security and Flexibility parameters have comparable effects on the value of QoE especially in the [high-deviation, high-deviation] and [low-deviation, low-deviation] pair which cause the QoE value to be 0 and 2.5 respectively. However, security shows a higher effect on the [fair, fair] pair of the input membership function to raise the QoE value to 1.5.

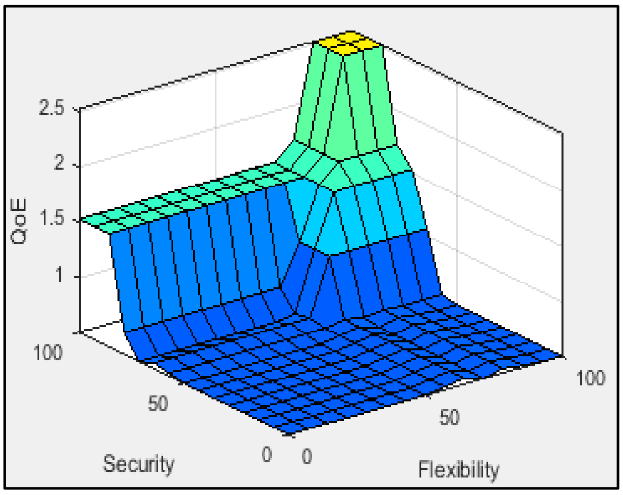


Figure 5-17 Effect of Security and Flexibility

The effect of the (Rapport, Flexibility) and (Reliability, Flexibility) parameters pairs on the QoE is depicted in Figure 5-18 and Figure 5-19 respectively. Where each pair has a comparable effect on the QoE value. The highest achieved value is 1.5, while dropping the input values causes a decline in the results to reach 0, but an observed rise is mentioned in the [fair, fair] especially for the Reliability and Rapport parameter.

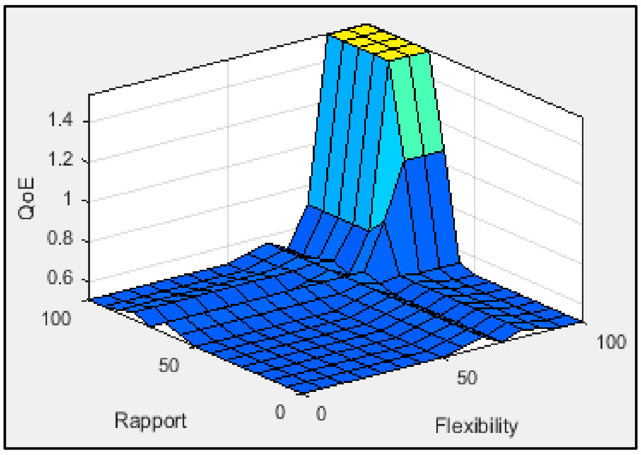


Figure 5-18 Effect of Rapport and Flexibility

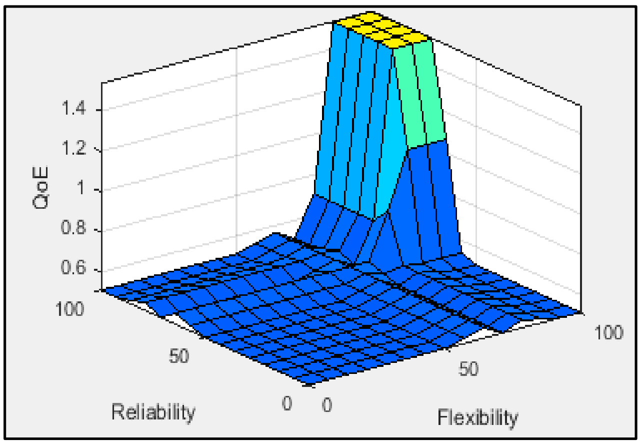


Figure 5-19 Effect of Reliability and Flexibility

Similar to the previous last two figures, the maximum achieved value of QoE is 1.5 as a result of the effect of (Rapport, Security) and (Reliability, Security) parameters' pairs in Figure 5-20 and Figure 5-21 respectively. However, Security causes a significant increase in the QoE value in the [fair, fair] pair.

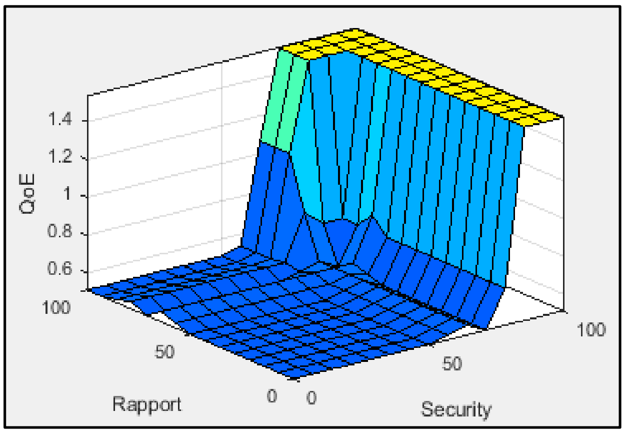


Figure 5-20 Effect of Rapport and Security

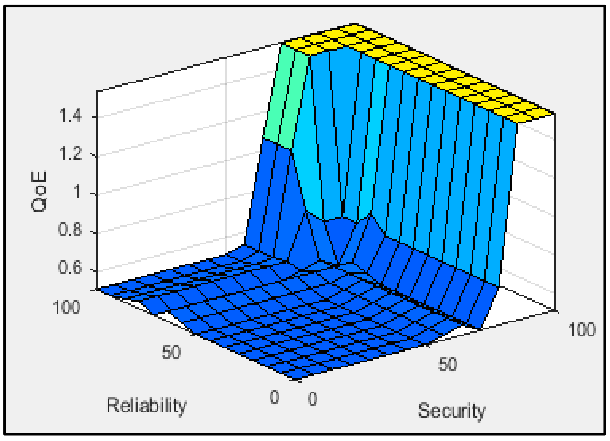


Figure 5-21 Effect of Reliability and Security

The same level of QoE is achieved in Figure 5-22, which shows the effect of Reliability and Rapport on the QoE value.

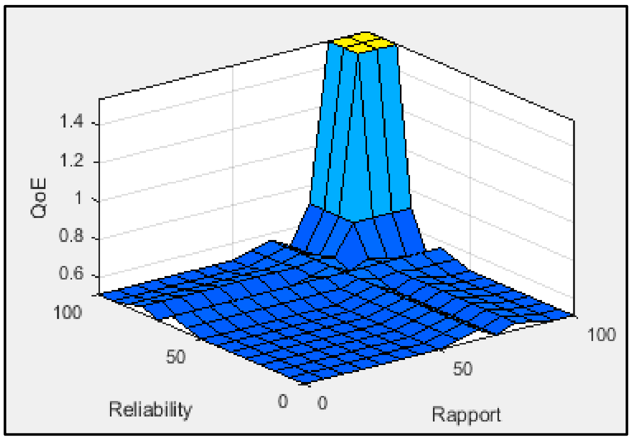


Figure 5-22 Effect of Reliability and Rapport

What is interesting in these figures is that any pair of the parameters does not cause the QoE value to reach its maximum level of 5. This means that its value does not depend on one or two parameters but on the other input parameters as well. It is important to note that the results reflect the users' preferences in the negotiation process of the SLA document through defining the (weights) of the SLA parameters.

5.6 Discussion

In this study, comparing the effect of each two of the SaaS-Qual model parameters reveals the relationship between them and the QoE value. The results showed that these combinations of the parameters are the most effective on the user satisfaction or the QoE value (Flexibility, Responsiveness), (Security, Responsiveness), (Rapport, Responsiveness), (Security, Flexibility). While these combinations of the parameters are the less effect on the QoE value or the user satisfaction (Features, Rapport), (Features, Reliability).

These results may explain the exponential relationship between QoE and QoS for some of the parameters, like flexibility. However, the observed low value of QoE for the reliability parameter contradicts this concept. This result contradicts the importance of the response time of web services users, who expect to receive the services on time.

As expected, the higher deviation between the measured and threshold SLA parameters causes a decline in the user satisfaction and as a result on the QoE value to drop to zero.

5.7 Chapter Summary

This chapter studied the estimation of the QoE of SaaS in cloud computing. A relationship was introduced to define the QoE of SaaS based on the SLA parameters and the network's QoS. The QoE value has been estimated using a fuzzy logic rule based on an inference engine.

Fuzzy logic was used as an approach for measuring the QoE value in SaaS cloud computing through studying the influence of the SLA parameters' degradation and network's status on a user's satisfaction. Applying the deviation of the measured to the threshold values of the SLA parameters as an input to the fuzzy engine helped in reflecting realistic results on the QoE value. The rules of the proposed fuzzy engine were selected based on SaaS-Qual as a model for evaluating a QoE taking into consideration the weights of these parameters. The chapter also presented the effect of SaaS-Qual parameters on the QoE.

This study can be a base for future studies to evaluate a QoE using a holistic unified metric. The next chapter introduces the implementation of the proposed middleware in both the client and provider sides and the main techniques used in the implementation.

CHAPTER SIX

MonSLAR IMPLEMENTATION

6.1 Introduction

The architecture of the proposed middleware and the mechanism for the monitoring process are presented in chapters Four and Five respectively. MonSLAR has the ability to monitor an SLA from a user perspective with the ability to keep the client up-to-date about any violation. This chapter presents the implementation of MonSLAR, which includes a description of the implementation of the main components of MonSLAR on both the client side and provider side, which manages communication using REST technology; and the file formats used for transmitting the data, the delivery and the display of the monitored data on the client side. The implementation of MonSLAR aims to achieve the main functionalities for the proposed monitoring process mechanism.

6.2 Message Flow of Monitoring in MonSLAR

MonSLAR provides an automated real time monitoring for the services and the SLA parameters. The communications between the client side and the provider side are accomplished using REST technology.

MonSLAR middleware has two main responsibilities, the first one is invoking and delivering SaaS services and the second is managing the monitoring process, which involves both measuring the QoE value (user satisfaction) and delivering the monitored measurements of the SLA parameters values.

Figure 6-1 presents a sequence diagram that shows the steps of the monitoring process in the proposed middleware. When the user uses a SaaS service, the MonSLAR-client side sends two types of requests to the MonSLAR-server side. The first one is to invoke the SaaS service from the SaaS provider and delivers this service to the client side through MonSLAR-client; the second one is the monitoring request.

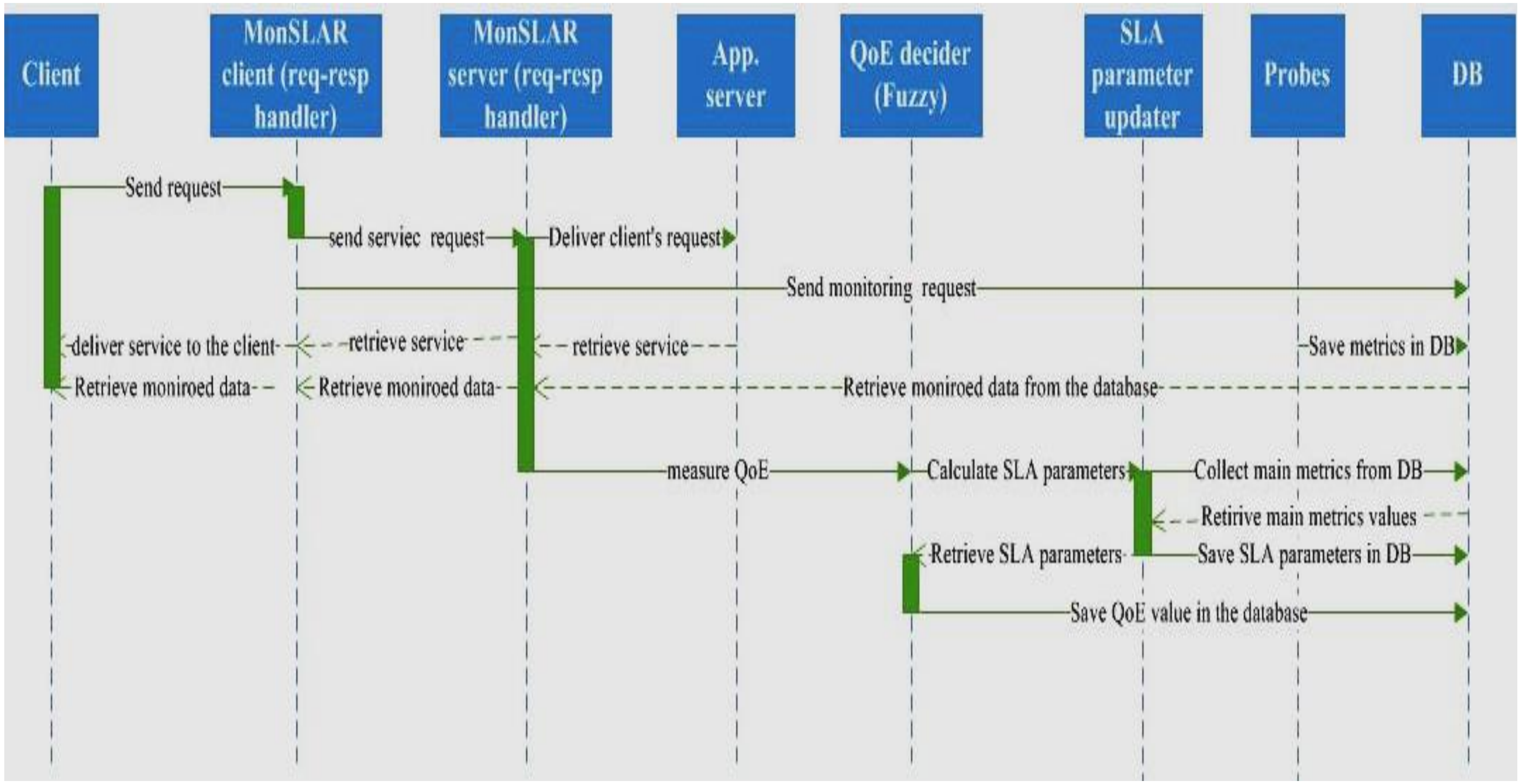


Figure 6-1 Sequence diagram describes the message flow of monitoring in MonSLAR

In the monitoring request, the MonSLAR-server side retrieves all the monitored data from the database and delivers it to the client through the MonSLAR-client side. This, in turn, will use this data to show the user satisfaction and the SLA breaching state in the form of dashboard; the monitoring request also activates the QoE decider component, which will activate the SLA parameter updater component that updates the parameters' values using the main metrics in the database, which were already measured by the probes. These values are used to calculate the QoE value, which is saved in the database. The user will be able to retrieve this information whenever refreshing the dashboard. This process ensures a real-time collection for the monitored data. Updating the SLA parameters' values is done by computing the average for those parameters values. Algorithm 6-1 illustrates the process of updating the measured values.

For each parameter value p= [p_1 p_2 p_3 … p_n], n: is the number of measured valued.

The overall value of each parameter after the update process is as below:

$$P_{new} = \frac{\sum_0^n P_n}{n} + P_{n+1} \quad \text{……………………………………………..……… (6.1)}$$

$$P_{new} = \frac{(\sum_0^n P_n) + P_{n+1}}{n+1} \quad \text{..…………………………………………..………… (6.2)}$$

```
Algorithm 6-1: updating the values of the measured parameters
Input: Measured parameters' values.
Output: Updated parameters' values.
1  For each received measured parameter ($P_{n+1}$) do
2      Retrieve the old value of the parameter ($P_n$) from the database
3      Calculate the value of $P_{new}$ as in equ.6.2
4      Save the value of $P_{new}$ in the database
5  EndFOR;
6  END
```

In MonSLAR, the monitored metrics are collected from the probes on the server side, which will then be used for calculating the SLA parameters' values and measuring the QoE value. This provides a trusted and reliable way for the measurement as the metrics are collected in an automated way directly from the probes.

In the implementation of MonSLAR, it is assumed that the SLA document is available and agreed on between the client and the service provider.

6.3 Deploying MonSLAR in SaaS Providers and Clients

The architecture of MonSLAR as an SOA facilitates the process of exploiting it and helps in retrieving the measurements and the monitoring process in the form of services. This concept enables customizing the collaboration between heterogeneous environments and integrating software components from different technologies with each other.

Both SaaS clients and providers can use the proposed middleware. MonSLAR's API provides the main monitoring functionalities that can be used by the cloud developer to use MonSLAR in monitoring the cloud service. On the other hand, the clients can use it in the form of Add-ons in the web browsers of the clients, which enables them to retrieve the monitored QoE values. The API specification of MonSLAR is introduced in Appendix C.

6.4 MonSLAR's Components Implementation

This section illustrates the implementation of the main components used in MonSLAR (see Section 4.4) for both the server side and the client side. A prototype of MonSLAR is implemented entirely using java programming language (Java Eclipse Kepler EE IDE for web developers), Apache Tomcat 7 server, and the JAX-RS REST API to manage the REST communications.

6.4.1 Server side MonSLAR

As described in chapter 4, the server side consists of the following components:

(a) Request/Response handler. This component processes the HTTP requests and responses, distinguishing the type of request (service, monitoring, or management request).

(b) SLA parameter calculator. This component is used to calculate the SLA parameter values based on the metrics.

(c) Repository. This component represents a database used to store the monitored data and the estimated values of the QoE for each client in addition to the SLA parameter values. The MySQL server is used to implement the repository as a database management system (MySQL, 2017), taking advantage of relational database management to assure accuracy in collecting the data. The values stored in the repository are considered a backup which is used later to notify the user about any degradation in the received service. It introduces a fast retrieval of the data by the other methods in the developed middleware. By considering the multi-tenant concept in SaaS, which allows different clients to use the same provider's application environment, an isolation should be taken into account when saving the values of the metrics and QoE value. These are specified for each user, in addition to the time of monitoring (time stamp), which gives an indication about the history of violations for both the cloud client and the provider. The isolation among clients in the shared database is considered by using a client-id field in the database which ensures retrieving the measurements for the specific client. The client-id is retrieved for each session. Figure 6-2 introduces an excerpt from MonSLAR's code for retrieving the value of QoE for the specific client.

```
Integer currentUserId = (Integer) request.getSession().getAttribute(
                                "currentUserId");
if (currentUserId != null) {
        File file = new File ("QoEstimator.fcl");
        FuzzyQoE fq = new FuzzyQoE();
        fq.RunFuzzy();
        Double value = new DBConnect().getQoE(new Integer(currentUserId));
```

Figure 6-2 java code depicts isolating the retrieved measurements based on the client_id

(d) Quality Manager. This component represents the QoE estimator tool used to estimate the value of QoE as an indicator of user satisfaction, which is implemented using the jFuzzyLogic open source library and supports the implementation of the Fuzzy Inference System (FIS) (Cingolani & Alcalá-Fdez, 2013). The use of this library allowed a comprehensive implementation for the middleware's components. This method returns the value of the QoE which is retrieved later by the HEAD method. The decision in this unit is made based on the metrics' values retrieved from the database repository.

(e) The probes. In this research, it is supposed that the measured data for the quality of service is collected from trusted probes and provides the correct database for the measured data. The probes are the monitoring tools available in the cloud. It is also assumed that the interval time for retrieving the measurements from the probes is agreed between the provider and the client.

(f) SLA builder. This component performs the SLA parameters extraction process by using an XML parser and save the values in the repository to be used later in the QoE estimation process.

6.4.2 The Client Side MonSLAR- System Front End

The client side of MonSLAR aims to support reporting the data to the client. This section introduces the dashboard used to visualize the user satisfaction to the user. The client side of MonSLAR consists of the following components:

(a) Request/Response handler. The implementation of this component includes the use of AJAX requests to call the REST methods that are used to retrieve the monitored data. The client's side monitoring requests are as follows:

HEAD method: [*URL: Base_URL/qoe]*
OPTIONS method: [*URL: Base_URL/measure]*

(b) The parser/metric aggregator. This component is used to monitor the current state of the received responses for the requested services, which includes Ajax instructions to measure the response time and the state of the received services that help in estimating the reliability and availability. The collected metrics present the base of the calculated SLA parameters to be used in estimating user satisfaction.

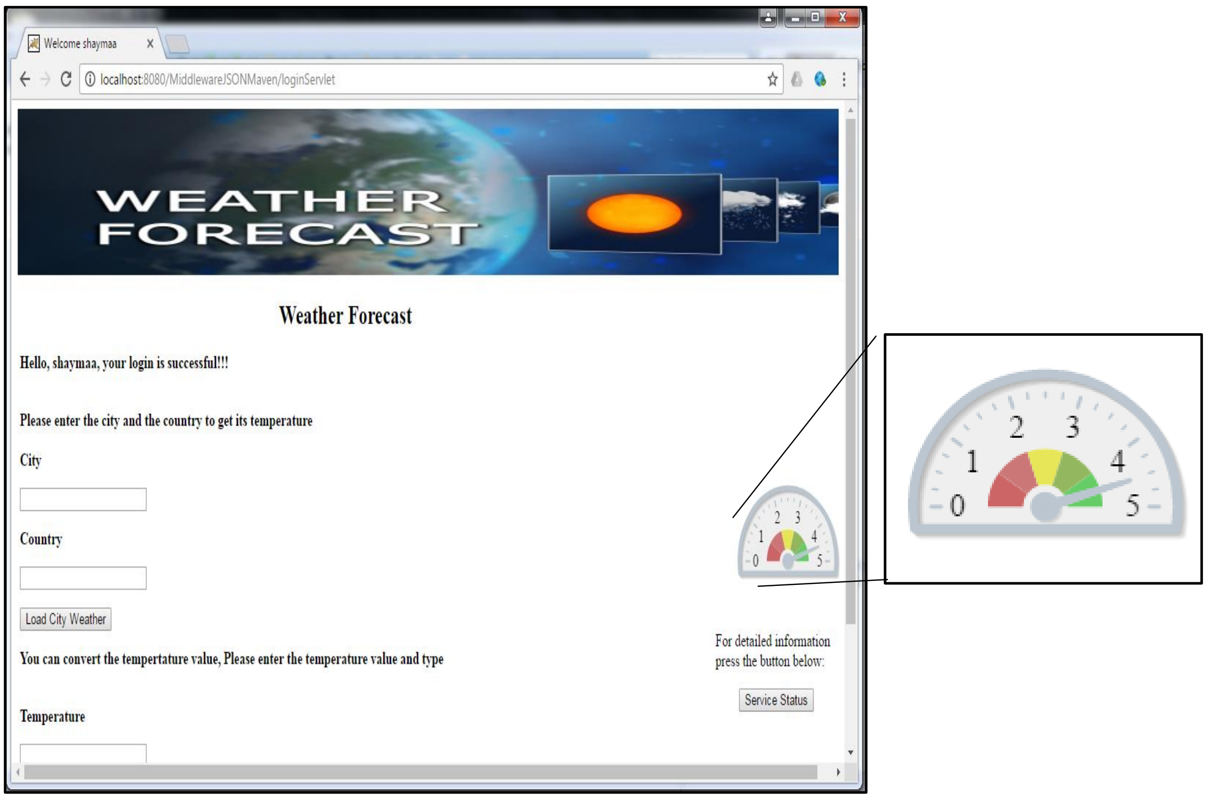
(c) Repository. the data collected from the different machines on the client side are stored in the cookies, which can be accessed later by MonSLAR's server side to be used in the decision making process.

(d) Quality Manager. This component interprets the QoE value that is retrieved from the provider side to show the level of user satisfaction about received services using a dashboard representation as shown below.

These requests are sent automatically when the user loads the SaaS service page. The retrieved data is displayed to the client in a dashboard, which gives a visualized and meaningful display of the relevant information that the user needs to know about the monitored data. Examples include the average value for each measured SLA parameter and the QoE value, in addition to the status of the users' satisfaction for this provider. This information is going to be displayed on one page, which gives the user more control and better administration of the SaaS application. The data is retrieved for the current user from the MySQL database

The user interface for the dashboard uses a display chart. A bar chart is used to show the measured values of the SLA parameters, while a meter chart is used to display the value of QoE. The meter max value is set to 5 taking into consideration the maximum value considered in the fuzzy logic system. The red colour is used to indicate service degradation, while green is used to show a proper status of the used services. The graphs created are based on the data gathered in real-time.

The user has the chance to use the SaaS service and at the same time monitors the SLA violation state by getting a dashboard with monitoring facilities and information about the user satisfaction status, as well as the values of the real measurements for the SLA parameters. Figure 6-3 describes how the user will be able to use the service and to get a dashboard that shows the QoE value. The user can also get detailed data about the factors that caused the QoE value by pressing the button "service status", as shown in Figure 6-4.


Welcome shaymaa

localhost:8080/MiddlewareJSONMaven/loginServlet

WEATHER FORECAST

Weather Forecast

Hello, shaymaa, your login is successful!!!

Please enter the city and the country to get its temperature

City

Country

Load City Weather

You can convert the tempertature value, Please enter the temperature value and type

Temperature

For detailed information press the button below:

Service Status

Figure 6-3 The SaaS service web page with MonSLAR's dashboard

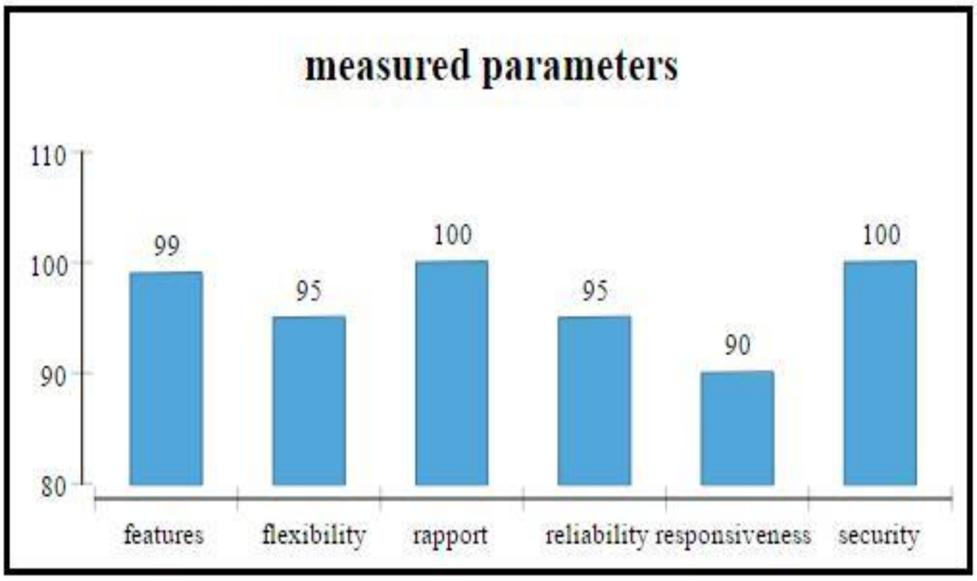


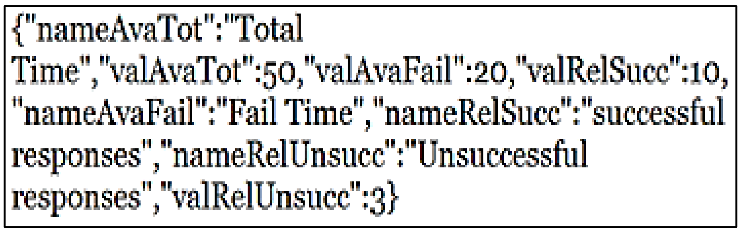
Figure 6-4 MonSLAR's dashboard with the monitored data

6.5 Implementing MonSLAR Requests

As discussed in Chapter Four, MonSLAR deals with three types of requests, service, monitoring, and management. These requests are initiated by Uniform Resource Locators (URLs) used to access the REST methods. This section introduces the communications and interfaces designed using REST technology by focusing on the implementation of the monitoring requests as they handle the management of the monitoring process in the proposed solution.

6.5.1 The Service Request

A REST service has been implemented for a weather forecast service. In this section the client sends a REST request asking for the service, the request could be any of the REST methods (GET, PUT, POST, and DELETE). After receiving the service on the client side, the main metrics are collected in the response, as explained in Section 6.4.2 (the parser, metric aggregator). The measured metrics are saved in a (JavaScript Object Notation) JSON file (Bray, 2014) as shown in Figure 6-5. This file will be sent using a POST method in the monitoring request-B presented in the following subsection. The use of JSON provides a lightweight transmission for the monitored data.



```
{"nameAvaTot":"Total
Time","valAvaTot":50,"valAvaFail":20,"valRelSucc":10,
"nameAvaFail":"Fail Time","nameRelSucc":"successful
responses","nameRelUnsucc":"Unsuccessful
responses","valRelUnsucc":3}
```

Figure 6-5 An example for saving the metrics in JSON file

6.5.2 The Monitoring Request

This section depicts the implementation of the monitoring requests in MonSLAR. REST technology offers a URI based technique to retrieve the information by sending an HTTP request. This facility is used to retrieve the monitored data as resources using both HEAD and OPTIONS methods.

In monitoring request-A, the client’s side middleware sends an HTTP HEAD request asking for the QoE value of that provider. This value will be included in the response of the HTTP response in a name-value pair. The value of QoE is retrieved from the FuzzyQoE() method. The java code for the HEAD method is presented in (Appendix D, **Error! Reference source not found.**).

The HTTP request for the HEAD method is shown in Figure 6-6. Figure 6-7 presents an example of the response to the HTTP HEAD method, which includes the value of the QoE in the monitoring request-A. It is obvious, from the response that the required information is embedded in the header and the HEAD method does not return a message body in the response. The retrieved data from this method is used in the dashboard to represent the user satisfaction about the SaaS service. If an error occurs in retrieving the data from the database, the client receives a message that there is no data to be displayed.

```
HEAD / MonSLAR /api/monitor/qoe HTTP/1.1
Referer: http://localhost:8080/MonSLAR/index.html
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
Host: localhost:8080
Content-Length: 0
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: JSESSIONID=316E698A14960970CD1F90F4F32D7DB6
```

Figure 6-6 HTTP message request of the HEAD method

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
QoE: 4.52
Content-Length: 0
Date: Tue, 19 Apr 2016 11:52:29 GMT
```

Figure 6-7 HTTP message response of HEAD method

For monitoring request-B, the metrics measured on the client side are embedded in a JSON file and sent to the provider side using the POST method. The retrieved data is saved in the repository, to be used later for measuring the value of QoE in the decision maker on the provider side. The Java code for the POST method is presented in Appendix D, **Error! Reference source not found.**.

To manage the connections of monitoring request-C of MonSLAR, the OPTIONS method is used to retrieve the values of the metrics required to evaluate the user satisfaction and the SLA compliance, transmitted in a JSON file format. This contains the metrics' values and names pairs measured by MonSLAR, and are received on the client side. The Java code for the OPTIONS REST method is introduced in Appendix D, where the method ReadMeasuredParameters() is used to retrieve the measured parameters values from MonSLAR's database on the provider side. The Java code for the POST method is presented in (Appendix D, Figure D- 3).

The HTTP request and response for the OPTIONS method are shown in Figure 6-8 and Figure 6-9 respectively, the data retrieved from this method is used in the user side dashboard to show the measured values of the SLA parameters as in Figure 6-4.

```
OPTIONS /MonSLAR/api/options/measure HTTP/1.1
Referer: http://localhost:8080/MonSLAR/index.html
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
Host: localhost:8080
Content-Length: 0
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: JSESSIONID=C977E6F8406CFEB3525273453BFDB465
```

Figure 6-8 HTTP message request for the OPTIONS method (monitoring request-C)

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
data:
[{"Value":99,"Name":"features"},{"Value":90,"Name":"responsiveness
"},{"Value":95,"Name":"flexibility"},{"Value":30,"Name":"security"},{"
Value":100,"Name":"rapport"},{"Value":95,"Name":"reliability"}]
Content-Length: 0
Date: Mon, 18 Apr 2016 17:04:22 GMT
```

Figure 6-9 HTTP message response for the OPTIONS method (monitoring request-C)

6.5.3 The Management Request

This section discusses the implementation of the management request in MonSLAR. To manage the connections of this request, the OPTIONS method is used to update the client side of the middleware with the SLA parameters in case of any updates. The pair of parameters' names and values are transmitted in JSON file format. The java code for the OPTIONS method is illustrated in (Appendix D, **Error! Reference source not found.**).

Figure 6-10 shows the HTTP request of the OPTIONS method, while an example of OPTIONS response is depicted in Figure 6-12.

```
OPTIONS /MonSLAR/api/sla HTTP/1.1
Referer: http://localhost:8080/MonSLAR/index.html
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
Host: localhost:8080
Content-Length: 0
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: JSESSIONID=C977E6F8406CFEB3525273453BFDB465
```

Figure 6-10 HTTP message request for the OPTIONS method (management request)

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
data:
[{"Value":98,"Name":"reliability"},{"Value":99,"Name":"availability"}]
Content-Length: 0
Date: Mon, 18 Apr 2016 17:04:22 GMT
```

Figure 6-11 HTTP message response for the OPTIONS method (management request)

6.6 Chapter Summary

This chapter discussed the implementation of MonSLAR with a description of the main methods used for transmitting the monitored data and delivering it to the client side. Technologies, such as JSON are used to provide a lightweight implementation. The use of the dashboard helps in representing the monitored data in a human readable format and simplifies the process of retrieving the data on the client side in an intuitive way. The communications between the client side and the provider side were implemented and managed using REST through providing the monitored data as services can be accessed using URLs specified for each method.

The next chapter presents the system testing and evaluation which is used to check that the thesis objectives have been achieved successfully.

CHAPTER SEVEN

SYSTEM TESTING AND EVALUATION

7.1 Introduction

This chapter presents the test and evaluation of the proposed system. The evaluation is submitted as proof that the objectives presented in Chapter One were achieved and to evaluate to what extent the proposed middleware achieved the aim of the study, which implied building a lightweight monitoring framework using REST technology.

The proposed middleware is tested to check that the system's tasks are working correctly. Then the middleware is evaluated. The first evaluation includes a quantitative evaluation to check the performance of the proposed middleware. The performance of MonSLAR is evaluated in terms of the overhead caused by the monitoring process. The overhead is measured taking into consideration the overhead added to the message size. The message size overhead evaluation is compared to PM middleware (please refer to Chapter Three). The second evaluation approach includes a qualitative study of the operations and features of the proposed middleware, in comparison to existing monitoring frameworks which were presented in Chapter Three.

The third evaluation approach evaluates the proposed fuzzy engine; this includes a test to check the values acquired from SaaS-Qual in measuring the QoE value. Then the proposed metric is validated using a user-study.

7.2 MonSLAR Test

This section discusses an algorithm for testing the functionality of MonSLAR. This is achieved by using a scenario to check the middleware completeness, which examines whether the system covers all the specified tasks or not. These tasks are (1) to estimate QoE based on SLA parameters, and (2) to deliver measurements using HEAD and OPTIONS RESTful methods.

7.2.1 Experiment Objectives

The objective of the experiment is to test the monitoring functionality of MonSLAR.

7.2.2 Experiment Setup

The proposed middleware was implemented and tested in a cloud environment.

The simulation environment is composed of two parts, the first section represents the client side machine which is Intel core i7, 3.6 GHZ processor, 16 GB of RAM, windows 7 64 bit, a software testing tool for measuring the response time of the received responses; while the second section of the testbed is the cloud computing implementation of the proposed solution.

DigitalOcean (DigitalOcean) was used as a public cloud service to deploy MonSLAR. The simulation environment for the virtual machine was 512 MB Memory / 20 GB Disk / LON1 - Ubuntu 16.04.1 x64. Docker (Docker, 2017) was used as a SaaS container builder to deploy the application on the cloud virtual machine. The SaaS container contained Apache Tomcat Server as the application server, in addition to the MonSLAR extension. The requests received by the SaaS containers are managed by MonSLAR, which is in charge of delivering these requests to the SaaS application server. MonSLAR is deployed on this machine, and the test achieved by accessing the application in the cloud environment. The results showed that MonSLAR executed and tested correctly, this is tested by checking the HEAD and OPTIONS methods of the received responses. Figure 7-1 shows the experiment testbed architecture used in the evaluation process.

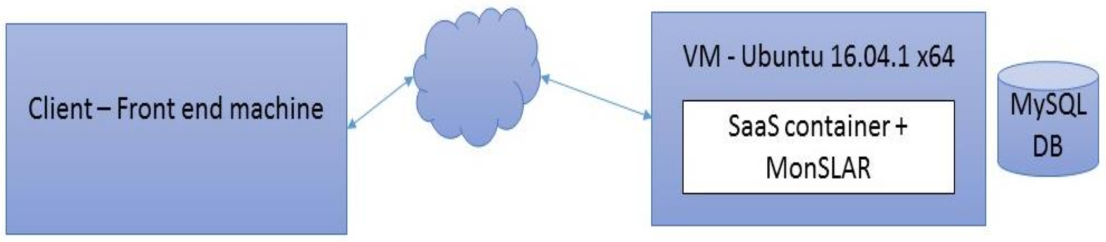


Figure 7-1 Experiment testbed architecture

Algorithm 7-1 is used to check the completeness of MonSLAR. This algorithm presents the pseudo code for the steps utilized for the delivery of the monitored data using the HEAD and OPTIONS methods. The algorithm initializes the used variables and checks the headers' fields of both HEAD and OPTIONS requests in lines (1-4), then according to this value sets the value of completeness in lines 5 and 7.

Algorithm 7-1: Checking the completeness of functionalities of MonSLAR

Input: List of MonSLAR responses
Output: completeness status

```
1   Initialize algorithm variables, set completeness = false
2   For each response of MonSLAR do
3       Check the QoE and parameters in headers of HEAD and OPTIONS
        requests
4       If (QoE header of HEAD or OPTIONS request = empty) then
5           completeness ← false;
6       else
7           completeness ← true;
8       end if
9   EndFOR;
10  Return completeness
11  END
```

7.2.3 Experiment Results

The proposed middleware was tested using a REST web application developed in the laboratory for the purpose of testing MonSLAR. The experiment was repeated 150 times in order to guarantee meaningful results, and the completeness status field was checked. The result value was 150 true and 0 false in the completeness field. This test helped to check the ability of the proposed middleware to present a user-centric monitoring and delivery set of data using REST. More results related to the system behaviour in terms of response time overhead is shown in Appendix E.
The next section introduces a quantitative evaluation of the proposed middleware.

7.3 Quantitative Evaluation of MonSLAR

Each monitoring process adds an additional overhead to the systems which it uses. This is caused by the time required to do the monitoring, in addition to the size of the data transmitted in this process. However, it is important to keep this value as low as possible. This section presents the quantitative evaluation, which is used to evaluate the overhead caused by MonSLAR in terms of the message size. Then, comparing its performance characteristics with other monitoring frameworks found in the literature review.

7.3.1 Message Size Overhead

In this section, an evaluation for the proposed middleware (MonSLAR) is considered, by comparing the usage of REST technology with the usage of SOAP technology in order to show the lightweight properties of REST technology in transferring the monitored data. MonSLAR is compared with PM (Zulkernine et al., 2008), PM is a performance monitoring middleware used to monitor the SLA of web services. The reason for choosing this middleware is because it delivers the monitored data to the client side by embedding the monitored data in the SOAP message header. Whilst in MonSLAR, the monitored data is delivered to the client side but embedded it in the REST header. Thus the use of PM in the evaluation provides the opportunity to compare the message size in both cases.

7.3.1.1 Experiment Objectives

The objective of the experiment is to evaluate the performance of MonSLAR in terms of the message size overhead.

7.3.1.2 Experiment Description

The experiment includes studying the size of the response message used in transferring the monitored data in MonSLAR. To achieve this target, the amount of data added to the header size is investigated for OPTIONS request which is used to exchange the parameters values mainly in the monitoring process (please refer to Chapter Four).

In order to compare the response messages of PM and MonSLAR, it is important to mention that PM delivers the data in the SOAP header while MonSLAR sends the data in the header of the REST (OPTIONS) method. As both SOAP and REST are based on HTTP protocol, the overhead caused by the HTTP message used to transfer the SOAP envelop should be taken into consideration in the comparison process. Figure 7-2 illustrates the difference between the compared SOAP and REST messages. While PM's monitored data is sent in the SOAP message which is in the HTTP body, MonSLAR data is sent in the OPTIONS header with no body entity.

Figure 7-3 shows a comparison between the response SOAP header message used in PM and the REST header message used in MonSLAR OPTIONS method.

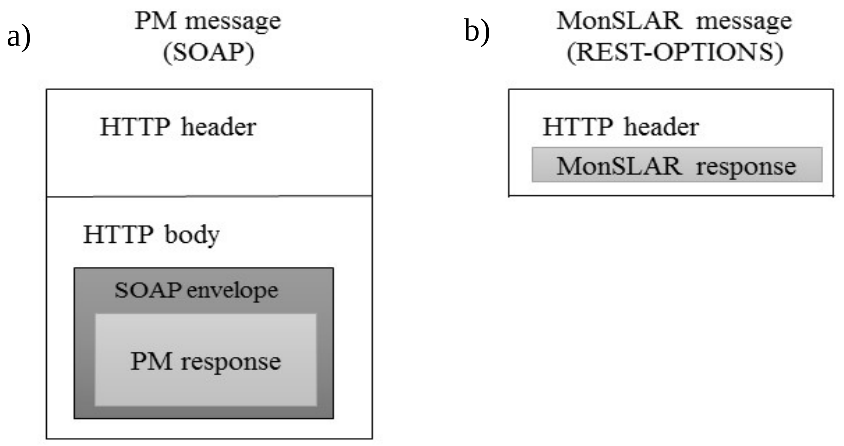
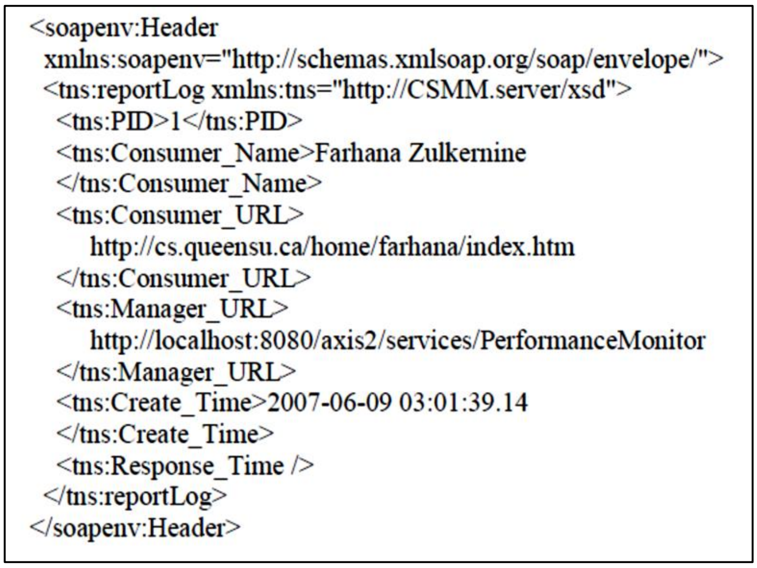


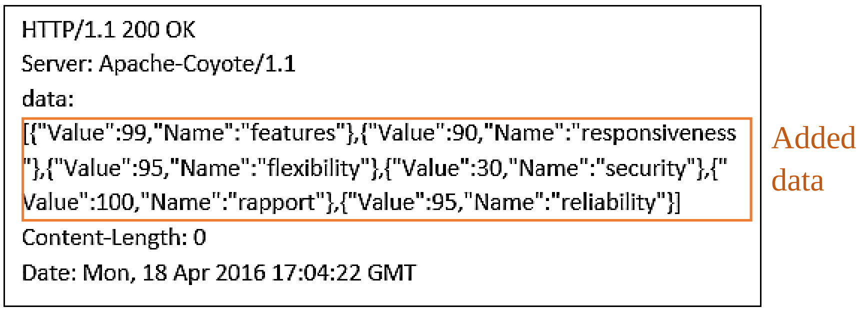
Figure 7-2 Illustrates representing the monitored data in the response message: (a) PM; (b) MonSLAR

a)



```
<soapenv:Header
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <tns:reportLog xmlns:tns="http://CSMM.server/xsd">
  <tns:PID>1</tns:PID>
  <tns:Consumer_Name>Farhana Zulkernine
  </tns:Consumer_Name>
  <tns:Consumer_URL>
     http://cs.queensu.ca/home/farhana/index.htm
  </tns:Consumer_URL>
  <tns:Manager_URL>
     http://localhost:8080/axis2/services/PerformanceMonitor
  </tns:Manager_URL>
  <tns:Create_Time>2007-06-09 03:01:39.14
  </tns:Create_Time>
  <tns:Response_Time />
 </tns:reportLog>
</soapenv:Header>
```

b)



```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
data:
[{"Value":99,"Name":"features"},{"Value":90,"Name":"responsiveness
"},{"Value":95,"Name":"flexibility"},{"Value":30,"Name":"security"},{"
Value":100,"Name":"rapport"},{"Value":95,"Name":"reliability"}]
Content-Length: 0
Date: Mon, 18 Apr 2016 17:04:22 GMT
```

Figure 7-3 Response message content: (a) PM (Zulkernine et al., 2008); (b) MonSLAR OPTIONS

The message size was investigated for PM and MonSLAR; algorithm 7-2 was used to check the message size overhead (bytes). The message size was tested using File.length() method in java. The algorithm computes the message size taking into consideration different parameter values (0 – 6), in order to check the overhead caused by transferring the parameters in each case. The reason for choosing this number of parameters is because the proposed estimation for the QoE value is based on six different parameters (please refer to Chapter Five). The results obtained for each case are discussed in Section 7.3.1.3.

Algorithm 7-2: Checking the message size overhead

Input: MonSLAR OPTIONS response, PM response
Output: message size (bytes)

```
1    Initialize algorithm variables, set OPTIONS_size=0, PM_size=0
     For i ← 1 to 6 do
2        If PM message
             Add (parameter_name, value), (HTTP header) to the message
             Compute message size (bytes)
3            PM_size (i) ← message size (bytes)
             Return PM_size
         Elseif OPTIONS message
             Add parameter_name, value to the message
             Compute message size (bytes)
3            OPTIONS_size (i) ← message size (bytes)
             Return OPTIONS_size
8        End if
9    EndFOR
10   END
```

7.3.1.3 Experiment Results

Figure 7-4 presents a comparison between the overhead added to the message response's header in bytes, for MonSLAR and PM middleware respectively. It is clear that the overhead added by PM is approximately five times larger than the overhead caused by MonSLAR. This overhead is due to the use of an XML based format for the parameters in the SOAP message header in addition to the HTTP message header added in case of PM, while the data is transmitted in a JSON format in the HTTP header in MonSLAR. These results support the previous research about the performance of REST in comparison to SOAP (please refer to section 2.4.2.3).

The next section presents a qualitative evaluation of the proposed solution.

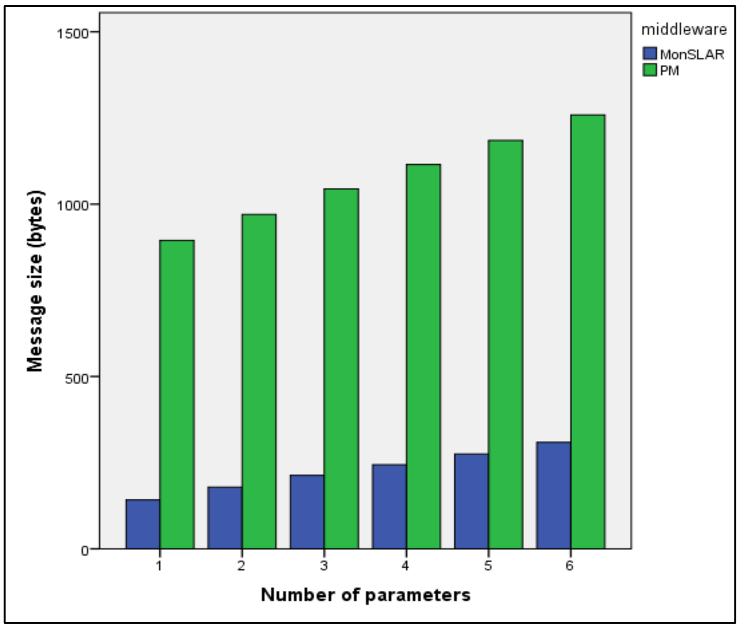


Figure 7-4 Message size overhead caused by varying no. of parameters for MonSLAR compared with PM

7.4 Qualitative Evaluation of MonSLAR

In this section, a qualitative evaluation is considered to evaluate MonSLAR by comparing its features with existing monitoring frameworks from the literature. The evaluation criteria are chosen as a proof for the user-centric objective set for the proposed middleware (MonSLAR). **Error! Reference source not found.** presents the eight main c riteria metrics used in the qualitative evaluation. The table also introduces the possible values for each quality metric and the baseline of each criterion, while Table 7-2 presents a comparison among MonSLAR and the available monitoring frameworks (from the literature), which have been presented in Chapter Three.

The criteria's metrics are chosen by taking into consideration the requirements of achieving a user-centric monitoring of an SLA in SaaS, which is presented in the study of MonSLAR.

The first criterion is the communication architecture; this criterion investigates whether a middleware was used to manage the communication and monitoring process between the provider side and the client side. Two values are considered for this criterion: 'Yes' for using middleware, or 'No' if no middleware is used to manage the communication. Using a middleware is considered the baseline, because of its characteristics for managing the heterogeneity among the different systems. The top value is to use a SOM, because of the features of service oriented to provide reusability and loose coupling to the overall structure. Using SOM in this thesis helped in managing the overall monitoring process through providing the monitored data as services by taking advantage of REST the technology.

The second criterion is checking user satisfaction. It considers three different values: either considering a ‘Check user satisfaction’ which is used to notify the user or manage the cloud resources according to the estimated value; or it could be that a ‘no Check for user satisfaction”; or a ‘comparison’ is used between measured values and SLA parameters to check violation. According to the previous studies, measuring the user satisfaction about the received services is an important issue, this can be achieved by measuring QoE as the perceived services by the end user. For this reason, the baseline is the use of comparison to check the provided services; the top value is to check user satisfaction with the used services.

The third criterion is the notify-ability; this criterion considers three different values. Its value can be either a dashboard with an automatic notification ability; or a dashboard with manual notification, which requires user interaction to retrieve the results; or no dashboard is used in a monitoring framework. The baseline for this criterion is set to the use of manual check dashboard, while the top value is the use of automatic notify-ability. Notify-ability helps the user in deciding the overall user satisfaction about received services. It also helps in saving time and effort in checking each SLA parameter and comparing that to a threshold value set in an SLA document, which is the function of the fuzzy logic engine in the proposed middleware.

The fourth criterion is the mode of monitoring, this criterion considers three different values which are: a server-centric monitoring, in which the monitoring process is managed by the server side; a user-centric monitoring, in which the monitoring process is managed by the client side; and a third party monitoring, in which the monitoring process is managed by a third party broker. A user–centric is chosen as a baseline and the top value for this criterion, which gives more control to the client side. The use of a user-centric monitoring removes the need for a broker to manage the monitoring process. In terms of monitoring architectures, the use of a broker as a third party is considered a source of failure and service outage. As the architecture of MonSLAR is developed by taking advantage of the cloud computing architecture, this provides high availability for MonSLAR through avoiding the use of a broker in comparison with the other monitoring architectures that use brokers to implement the monitoring process or delivering the monitored data. MonSLAR assures higher availability, by avoiding the denial of services caused by a broker failure.

The fifth criterion is the interaction type, which is used for delivering the monitored data to the client side. Three different values are considered, either using a web services-REST technology, or a web services-SOAP protocol, or no web services are used to deliver the monitored data. The baseline is set to the use of web service (either SOAP or REST) to manage the interaction. REST technology is considered the top value, because of the characteristics of REST to have low overhead values compared to SOAP protocol (see section 2.4.2.3), in addition to the trend of the cloud services to be managed by REST because of its simplicity and ease of use. This takes into consideration the difficulty of including the monitored data in a REST technology in comparison with SOAP. However, MonSLAR handles this problem by embedding the monitored data in HEAD and OPTIONS methods which add more flexibility in managing the monitored data and delivering it to the client side.

The sixth criterion is SLA oriented, which has two possible values, either ‘Yes’, where SLA parameters are taken into consideration in the monitoring process, or ‘No’, in which the monitoring process does not consider the SLA parameters. SLA parameters are considered an important issue in monitoring cloud environment, where SLA parameter values represent the agreed levels of services between a client and a cloud provider. This consideration could be a good indication of services’ levels achievement, where SLA parameters represent threshold values to evaluate received services’. ‘Yes’ is considered the baseline and the top value for this criterion.

The seventh criterion is the real-time measurement. This criterion accepts two values, either ‘Yes’, which manages to deliver online real time measurements to the client-side; or ‘No’, which does not consider real time measurements of the data. The baseline is set to ‘No’ that considers the use of monitored data regardless of the real time measurement, while the top value considers delivering real time measurements; this assures more control of the monitored data and the violation detection process. The use of real time data gives a realistic indication about breaching the SLA contract.

The eighth criterion is the automatic detection through providing automatic detection of SLA violations by automatically monitoring the measured data. Two values are considered for this criterion: ‘Yes, for the ability to provide an automatic detection of SLA violation; or ‘No’, where no automatic detection is used in a monitoring process. The baseline value and the top value are set to having the ability of automatic detection; again this characteristic helps in keeping a monitoring process unbiased to any of the parties in the system.

Table 7-1 Quality Criteria

Metric	Description	Possible values	Baseline	Top value
Communication architecture	The type of communication used to manage the monitoring process (a middleware used to manage communication between provider and client?)	- Yes - No -Not mentioned	Yes	SOM
Check user satisfaction	The ability to check the user satisfaction and decide SLA violation	- Check user satisfaction (CUS) - No Check for user satisfaction (NCUS) - Comparison	CUS	CUS
Notify-ability	The ability to provide automatic notifications to the user: a dashboard used to notify the user automatically, or the dashboard requires the user to take manual action to get data.	- Manual check (MC) - Automatic Notification (AN) - No Dashboard used (ND)	MC	AN
Mode of Monitoring	This criterion indicates which party is in charge of managing the monitoring process	- User centric - Server centric - Third party	User centric	User centric
Interaction type	The interaction type used to deliver the services to the client side.	- web services-REST - web service-SOAP - (-) No web services were used to deliver the measurements to the client side	Web services	REST
SLA oriented	The ability of considering the SLA parameters in the monitoring process.	- Yes - No	Yes	Yes
Real time measurement	The ability of the monitoring system to provide real time measurements of the used services.	- Yes - No - (-) not mentioned	No	Yes
Automatic detection	The ability of the system to provide automatic measurements of the used services and automatic detection of the SLA violation	- Yes - No	Yes	Yes

Table 7-2 Comparing MonSLAR to the other monitoring frameworks

Reference	**Notify-ability**	**SLA oriented**	**Check user satisfaction**	**Mode of Monitoring**	**Interaction type**	**Communication Architecture**	**Real time measurement**	**Automatic Detection**
MonSLAR	AN*	Yes*	CUS*	User-centric*	REST*	Yes*	Yes*	Yes*
SLAM (Moustafa et al., 2015)	MC	Yes*	NCUS	User-centric*	-	Not mentioned	Yes*	No
DARGOS (Povedano-Molina et al., 2013)	MC	No	NCUS	Server-centric	REST*	No	No	No
(Serhani et al., 2014)	MC	Yes*	Comparison	User-centric*	-	No	Yes*	No
M4CLOUD (Mastelic et al., 2012)	ND	Yes*	CUS*	Server-centric	-	Not mentioned	Yes*	Yes*
CASViD (Vincent C Emeakaroha et al., 2012)	ND	Yes*	CUS*	User-centric*	-	Not mentioned	Yes*	Yes*
(Rak et al., 2011)	MC	Yes*	NCUS	Third party	-	No	Yes*	No
JCatascopia (Trihinas et al., 2014)	ND	No	NCUS	Server-centric	REST*	No	Yes*	Yes*
GMONE (Montes et al., 2013)	MC	No	NCUS	User-centric*	-	No	Yes*	No

LoM2HiS (Vincent C Emeakaroha et al., 2010)	ND	Yes*	Comparison	Server-centric	-	No	Yes*	Yes*
(Shao & Wang, 2011)	ND	Yes*	CUS*	Server-centric	-	No	Yes*	Yes*
SALmonADA (Müller et al., 2012)	MC	Yes*	CUS*	User-centric*	SOAP	Yes*	Yes*	Yes*
UCSM (Rehman et al., 2015)	MD	No	CUS*	User-centric*	-	No	Yes*	No
(Cedillo et al., 2015)	MD	Yes*	Comparison	Server -centric	-	No	Yes*	Yes*
JTangCMS (Lu et al., 2016)	ND	No	CUS*	Server-centric	-	No	Yes*	Yes*
(S.-Y. Lee et al., 2012)	ND	Yes*	CUS*	Server-centric	SOAP	Yes*	Yes*	No
(Ye et al., 2012)	MD	Yes*	CUS*	Third party	-	-	-	No
(Siebenhaar et al., 2013)	ND	Yes*	CUS*	Third party	-	No	Yes*	Yes*
(You et al., 2015)	MD	Yes*	CUS*	Third party	-	-	Yes*	No
QoSMONaaS (Cicotti et al., 2012)	ND	Yes*	Comparison	Third party	-	Not mentioned	Yes*	No

* indicates achieving the top value for the specified criterion.

Underlined values indicate achieving the baseline level.

7.5 The Fuzzy Engine Test

7.5.1 SaaS-Qual Test

In order to study the SaaS-Qual as a model to estimate the QoE value, experimental scenarios were used to examine the QoE behaviour (Section 5.3, Figure 5-2). The proposed fuzzy engine was tested by feeding a set of different input data values to check the result of the QoE value. Test data is used for these reasons: (1) Availability of the data, where the used dataset represents measurements of parameters used for real cloud users and the difficulty of obtaining this data in real experiment; (2) Ease of storage and reproduction of the data; and (3) Repeatability, where the experiments can be repeated with the generated data. A random input data for the deviation input to the fuzzy engine was considered to examine the QoE value for these different values. A sample of the results of the QoE obtained by testing different random values of SLA parameters is shown in Table 7-3. This table gives an indication of the estimated QoE value as a result of changing the input parameters' values.

In an attempt to generate more realistic data for the input measured parameters values, a random data set was generated taking into consideration the standard deviation and the mean of the trusted feedback dataset available in the Cloud Armor web project (Cloud Armor, n.d.). As the values in this dataset reflect the user's opinion for each parameter, this helps as they can be considered as the deviation of the measured parameters from the SLA threshold values. The values' range provided in this dataset was [1, 5] as it represents users' opinions in Likert scale, this required using a scale to convert it to [0, 100] to make it compatible with the proposed fuzzy input membership functions.

Table 7-3 Test results of QoE values

Features	Responsiveness	Flexibility	Security	Rapport	Reliability	QoE
70	52	11	88	100	48	2.525
46	40	71	68	76	98	1.535
93	95	90	80	90	95	4.464
80	90	84	100	24	60	3.525
89	50	92	22	90	76	2.525
26	86	58	90	27	80	2.929
39	57	39	46	28	30	0.536

7.5.2 Validating the Proposed Metric

7.5.2.1 Purpose of the Study

As the QoE refers to the user satisfaction or dissatisfaction about a service, a questionnaire survey is conducted to validate the proposed metric and the effect of the SaaS-Qual parameters' weights on the overall QoE value. The purpose of the study is to gather information about user satisfaction with SaaS services. This survey allowed checking the possibility of using the SaaS-Qual quality model as a measure for the QoE. The survey included a questionnaire to study factors influence the QoE of SaaS services like email and Google documents. Two phases are considered in the design of the questionnaire's questions. The first phase included a pilot study, which was held to design the questionnaire's questions before collecting the main data of the research. The pilot study included sending the questionnaire to 10 different participants and discussing the results with researchers from the field, to ensure that the questions are understandable, and to derive the final version of the questionnaire; in addition to checking the applicability of the questionnaire to this kind of study. The second phase of the study included sending the questionnaire to 150 students at the University of Salford. The reason for choosing this population is to ensure that all the participants have an experience with the email as a kind of SaaS services as the participants are intended to be general users of the SaaS services. The number of received responses were 100; 27 of the received responses are not included in the analysis. Excluding these cases was because of the inaccurate results obtained due to the same repeated results for all the questions in the study for a particular participant. As a result, the actual number of answers included in the analysis is 73. The reasons for selecting this population are: (1) because of their knowledge about the technology of email as a kind of SaaS service; (2) the simplicity of gathering the data in the form of a laboratory experiment. The questionnaire survey is presented in Appendix F.

The questions in the questionnaire were designed to allow the researcher to evaluate the rules of the fuzzy logic inference engine. Six questions were used to ask about the effect of each of the six parameters on user satisfaction, in addition to the other four scenarios to evaluate some cases from the fuzzy rules table with different levels of each parameter (bad, medium, and high). Ten different scenarios were chosen so that the questions are not too long for the participants, and at the same time gives an indication of the main effects of the SLA parameters on the user satisfaction. Linguistic terms were used to represent the values of the SLA parameters and the QoE, to ensure an easy understanding of the questions and to avoid the misunderstanding of the numerical values by the participants.

In the questionnaire, the participants needed to express their opinions using a Likert scale. A Likert scale of five points ranging from 1 to 5 was used with the following values: strongly satisfied, satisfied, neutral, dissatisfied, and strongly dissatisfied). This scale has the advantage that it reflects the linguistic terms used in the fuzzy logic membership function so that it was neither too hard for the researcher to express in the design of the survey nor too hard for participants to understand and answer. The participants were asked about their opinions about the effect of each of the SaaS-Qual parameters and their satisfaction about the perceived cloud service. The questionnaire was designed and implemented using the online survey tool (SurveyMonkey, 2016) to provide more visual interaction with the users in addition to the ease of use and collection of the study data. Data analysis of the obtained results were accomplished using SPSS.

7.5.2.2 Study Process

Figure 7-5 illustrates the process of validating the proposed metric. The process started by generating random dataset taking into consideration the ten different scenarios of the user study. The same data was used in the fuzzy engine and the user study to compare the results obtained in each case. The numerical input data were mapped to equivalent expressive linguistic terms, taken into consideration the fuzzy input membership function (section 5.4.1, Figure 5-4). The results of the study were then expressed using linguistic values comparable to the fuzzy results so that strongly dissatisfied, dissatisfied, neutral, satisfied, and strongly satisfied mapped to bad, poor, fair, good, and excellent respectively. Following this, the generated dataset was fed to the fuzzy engine, and the computed QoE values were reported. Finally, the results were compared and analysed.

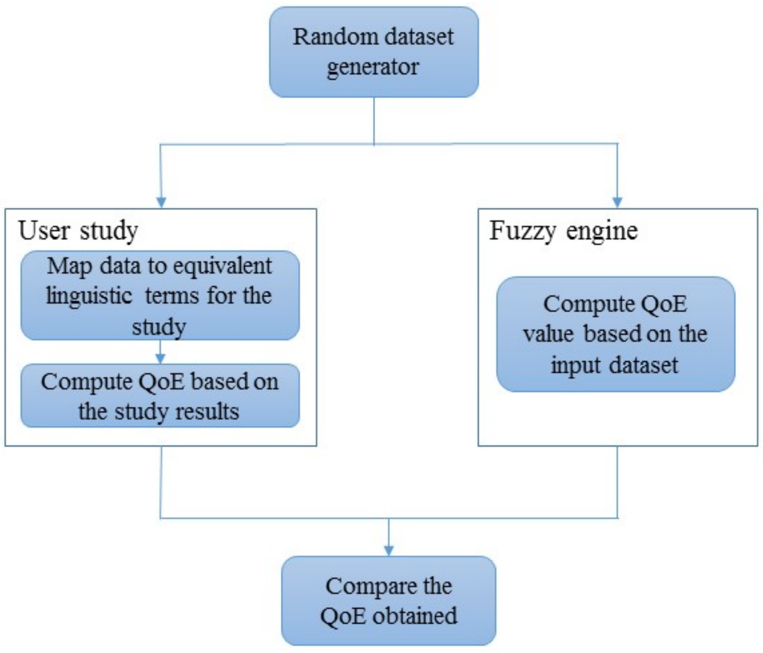

Random dataset generator
User study
Map data to equivalent linguistic terms for the study
Compute QoE based on the study results
Fuzzy engine
Compute QoE value based on the input dataset
Compare the QoE obtained

Figure 7-5 Process of validating the proposed metric

7.5.2.3 Study Results

The first question was about the effect of declining the Responsiveness parameter on the QoE value. The results are shown in Figure 7-6, which shows that 39.7% of the participants were satisfied with this scenario. Equivalent numeric values were fed to the fuzzy engine to estimate the QoE value; the result was 3.525 that is equal to good; a similar result has been achieved in the user study.

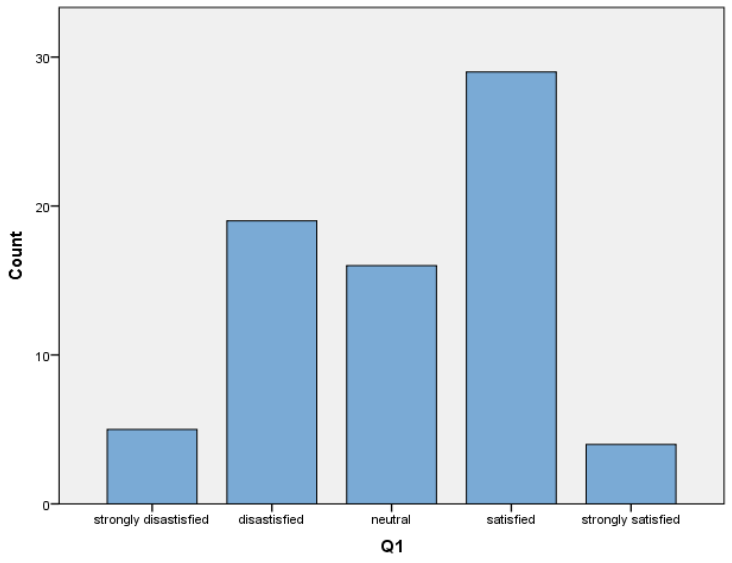


Figure 7-6 participants' results for Responsiveness parameter

The second question was about the effect of declining the Features parameter on the QoE value. The results are shown in Figure 7-7, with 43.8% of the participants choosing neutral. Equivalent numeric values were fed to the fuzzy engine to estimate the QoE value, the result of the fuzzy engine was 4.52 that is equal to excellent; while the result obtained in the user study was fair.

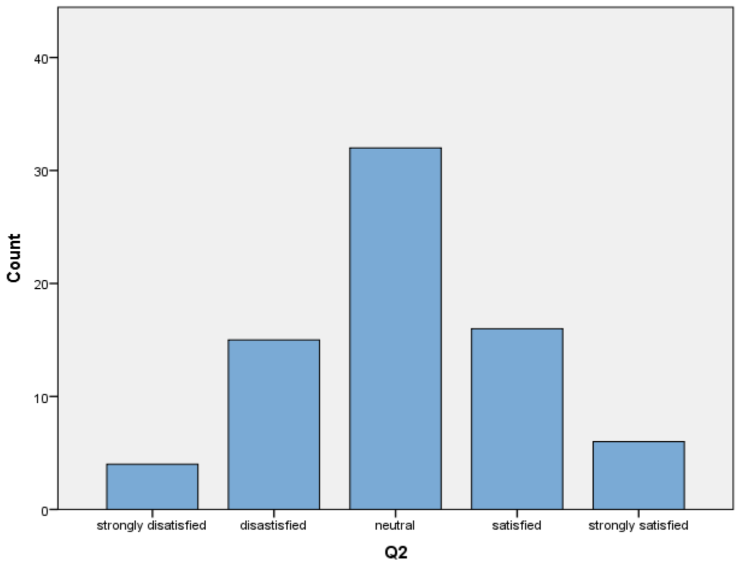


Figure 7-7 participants' results for Features parameter

The third question investigated diminishing the Security parameter on the QoE value. The results are illustrated in Figure 7-8 with 41.1% of the participants strongly dissatisfied with the scenario. The results obtained from inputting similar data to the fuzzy engine reported that the QoE value was 3.525 which is equivalent to the good term, but the result obtained in the questionnaire was bad. The results achieved from the survey reflects the high effect of this parameter on user satisfaction.

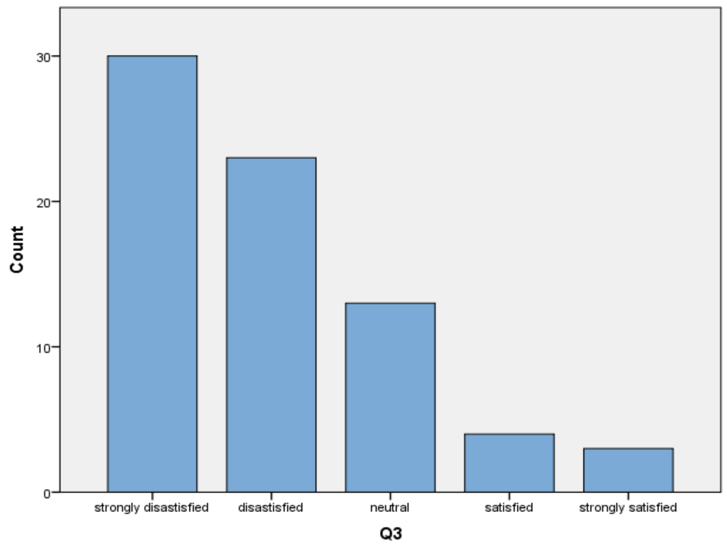


Figure 7-8 participants' results for Security parameter

The influence of declining the Rapport parameter on the QoE value was studied in the fourth question. The results are as shown in Figure 7-9, with 42.5% of the participants had been satisfied. On the other hand, the results obtained from running the fuzzy engine with the same input data produced a QoE value of 4.52 which is excellent. The results obtained in the survey were good.

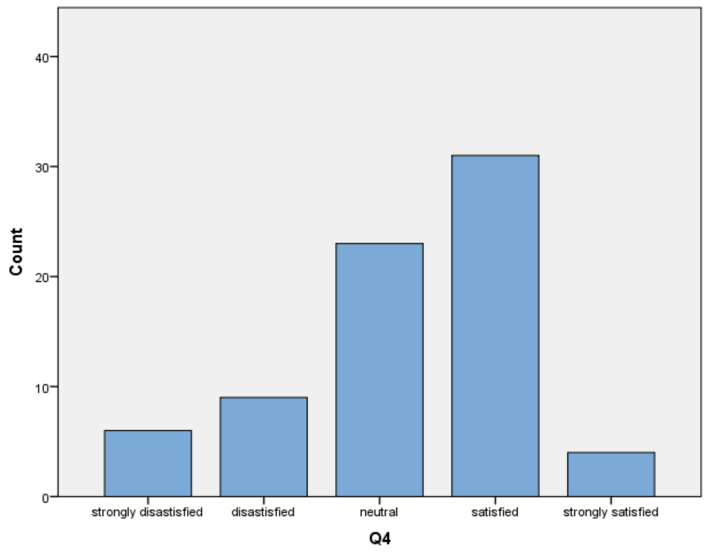


Figure 7-9 participants' results for Rapport parameter

Exploring the effect of dropping the Flexibility parameter on the QoE value was considered in the fifth question. The results are presented in Figure 7-10, with 37% of the participants being satisfied with this scenario. Nevertheless, the results acquired by computing the QoE value using the fuzzy engine was 4.514 which is excellent; the results obtained in the survey were good.

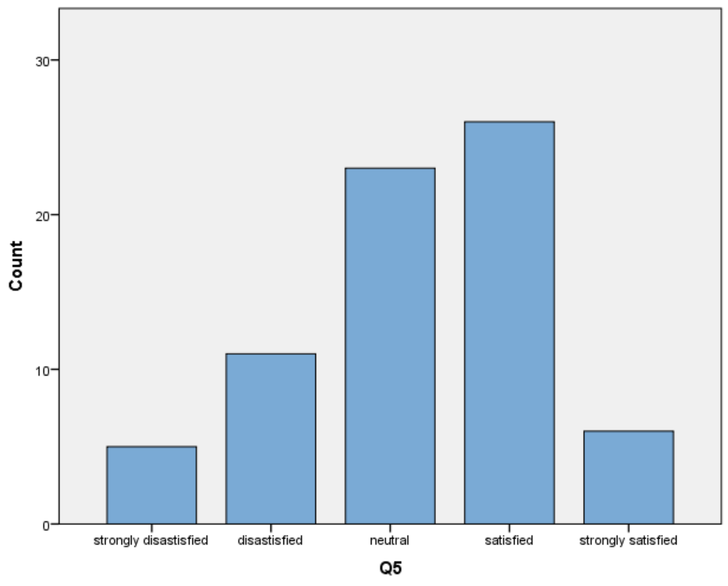


Figure 7-10 participants' results for Flexibility parameter

The sixth question was about the impact of declining the Reliability on the QoE value. The results are shown in Figure 7-11, with 52.1% of the participants were dissatisfied with this scenario. Simulating the fuzzy engine with similar input data resulted in QoE value equal to 4.52 which is excellent. However, the result obtained in the survey was poor, which is an indicator of the high impact of this factor on QoE.

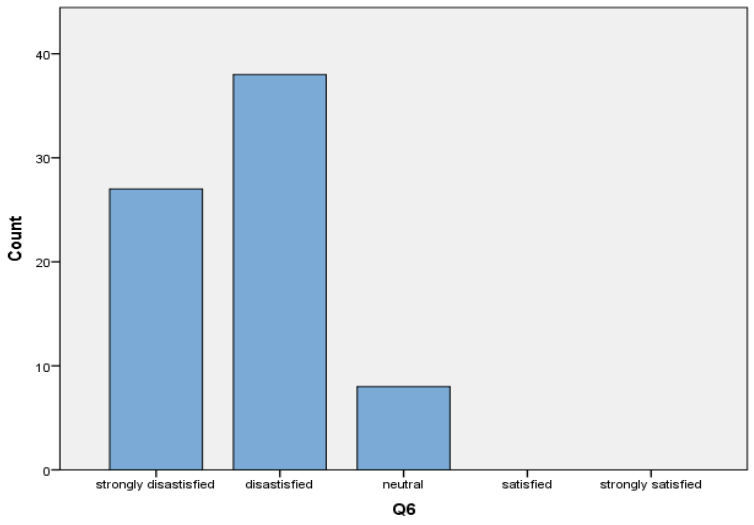


Figure 7-11 participants' results for Reliability parameter

The seventh question considered a combination of the parameters values on the QoE value. The input values of the parameters Responsiveness, Reliability, Flexibility, Security, Features, and Rapport were set to bad, good, medium, medium, medium, and medium respectively. The results are depicted in Figure 7-12, with 37% of the participants were dissatisfied. While the results acquired by simulating the fuzzy engine with this set of data resulted in a QoE value of 2.525 which is fair. On the other hand, the results obtained in the survey were poor.

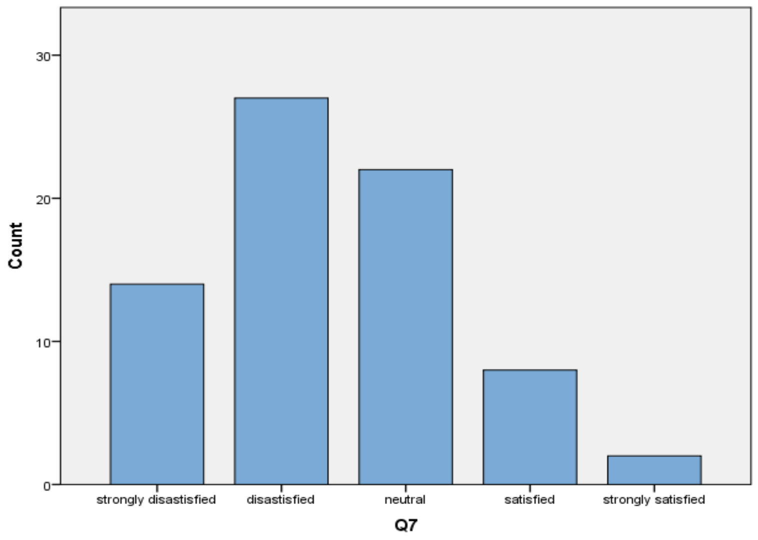


Figure 7-12 participants' results for the seventh question

The eighth question introduced another combination of the parameters values to find their effect on QoE. The values of Responsiveness, Reliability, Flexibility, Security, Features, and Rapport were considered good, medium, good, good, good, and bad respectively. The results are shown in Figure 7-13, with 43.8% of the participants were dissatisfied. While the results of the fuzzy engine were 4.508 which is excellent, the result obtained in the survey was poor.

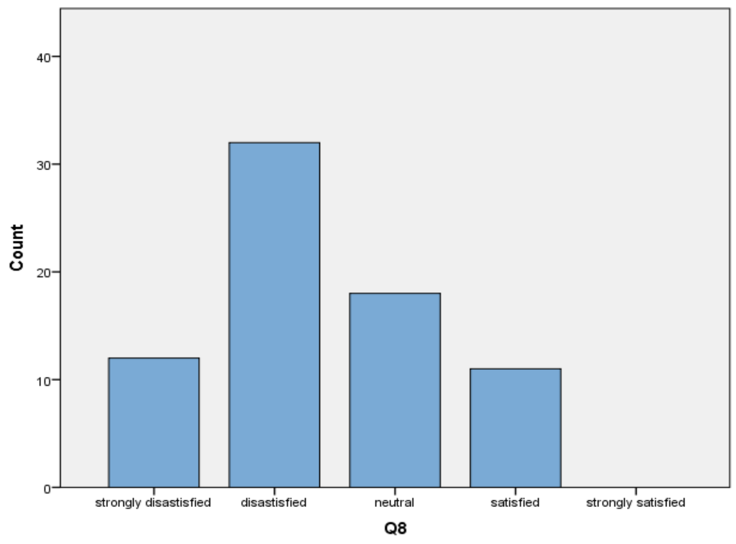


Figure 7-13 participants' results for the eighth question

The ninth question studied the effect of considering the values of Responsiveness, Reliability, Flexibility, Security, Features, and Rapport parameters to good, good, medium, good, bad, and bad respectively. The results are presented in Figure 7-14, with 37% of the participants were dissatisfied. While the results obtained from the fuzzy engine showed that QoE was 2.525 which is good, the result obtained in the survey was poor.

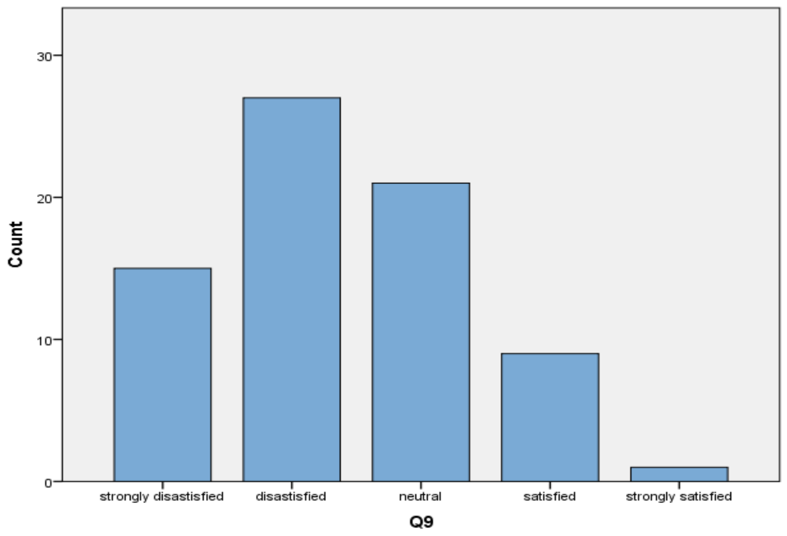


Figure 7-14 participants' results for the ninth question

The tenth question considered a combination of the parameter values on the QoE value. The parameters were selected as: Responsiveness: medium, Reliability: medium, Flexibility: good, Security: bad, Features: good, and Rapport: good]. Figure 7-15 shows the results of the study, with 52.1% of the participants dissatisfied. Whilst the results obtained from the fuzzy simulation was 3.525 which is good, the result obtained in the survey was bad.

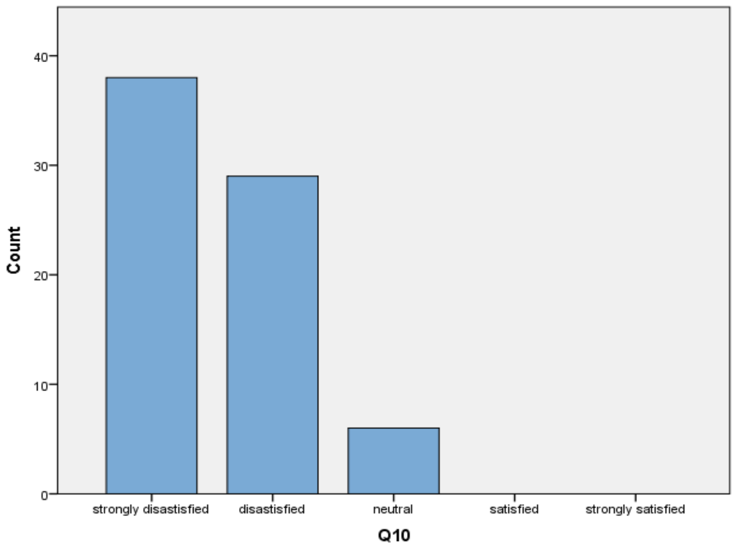


Figure 7-15 participants' results for the tenth question

The overall obtained results are summarized in Table 7-4, which shows a comparison between the results achieved by the fuzzy engine and the results obtained by the user study. This table gives a good indication of the difference between the expected results and the realistic data.

Table 7-4 Comparison between the results of Fuzzy decision and the survey study

Case	Input Data												Fuzzy Output		Study Output	
	Responsiveness linguistic	**Responsiveness numeric**	**Reliability linguistic**	**Reliability numeric**	**Flexibility linguistic**	**Flexibility numeric**	**Security linguistic**	**Security numeric**	**Features linguistic**	**Features numeric**	**Rapport linguistic**	**Rapport numeric**	**Fuzzy results linguistic**	**Fuzzy results numeric**	**Study results**	**Study results**
1	Bad	30	Good	88	Good	90	Good	95	Good	93	Good	99	Good	3.525	Satisfied	Good
2	Good	98	Good	92	Good	89	Good	90	Bad	40	Good	94	Excellent	4.52	Neutral	Fair
3	Good	99	Good	99	Good	92	Bad	22	Good	90	Good	91	Good	3.525	Strongly dissatisfied	Bad
4	Good	90	Good	96	Good	100	Good	89	Good	95	Bad	34	Excellent	4.52	Satisfied	Good
5	Good	100	Good	90	Bad	15	Good	99	Good	88	Good	95	Excellent	4.514	Satisfied	Good
6	Good	95	Bad	8	Good	88	Good	100	Good	97	Good	90	Excellent	4.52	Dissatisfied	Poor
7	Medium	73	Bad	20	Medium	75	Medium	70	Medium	73	Good	94	Fair	2.525	Dissatisfied	Poor
8	Good	98	Good	89	Good	95	Good	90	Bad	22	Medium	74	Excellent	4.508	Dissatisfied	Poor
9	Bad	50	Good	91	Medium	73	Good	95	Bad	18	Good	100	Fair	2.525	Dissatisfied	Poor
10	Good	92	Medium	74	Good	93	Bad	21	Good	89	Medium	75	Good	3.525	Strongly dissatisfied	Bad

7.5.3 Adjusting the Fuzzy Engine Based on the User Study

Based on the methodology presented in (section 5.3, Figure 5-2), the results obtained from the user study were used for adjusting the design of the fuzzy logic engine. This adjustment involved modifying the fuzzy rules to match the users' opinions, this modification then used to change the parameter weights according to the new rules.

In order to decide the final form of the rules of the fuzzy inference engine, the adapted system behaviour was discussed with an expert. This section explains the behaviour of the adjusted system by introducing the surface diagrams of each pair of SaaS-Qual parameters. The most striking results to emerge from this study is the increase of the QoE level especially for the security and reliability parameters.

The first set of figures are presented in Figure 7-16 and Figure 7-17, which shows the effect of (Responsiveness – Features) and (Flexibility – Features) parameters' pairs respectively. The figures reveal a comparable behaviour in both cases, but the Responsiveness parameter has a higher effect on the [fair fair] pair. The maximum value of QoE is 1.5 in the [low-deviation low-deviation] pair.

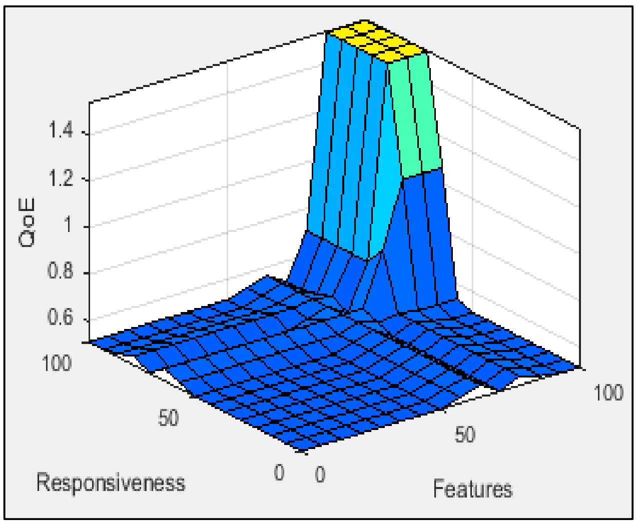


Figure 7-16 Effect of Features and Responsiveness

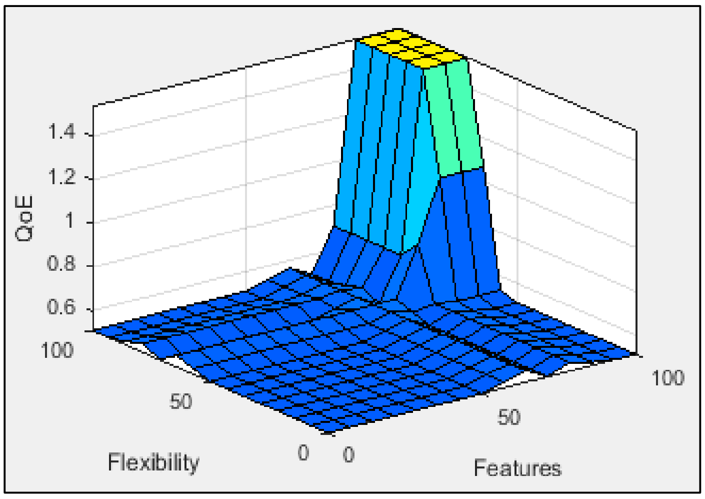


Figure 7-17 Effect of Features and Flexibility

The influence of the (Security - Features), (Security - Responsiveness), and (Security – Flexibility) parameters' pairs is shown in Figure 7-18, Figure 7-19, and Figure 7-20 respectively. These pairs cause an increase in the QoE value to reach 3.5. As the figures depict, Security has a higher effect on the QoE than Features, Responsiveness, and Flexibility. This is obvious for the fair and low-deviation of Security to cause the QoE value to reach 3.5. However, the QoE value is higher in the [low-deviation low-deviation] pair in Figure 7-18 than this obtained in the other figures, and this is because the Features parameter has higher priority than Responsiveness and Flexibility according to the user study.

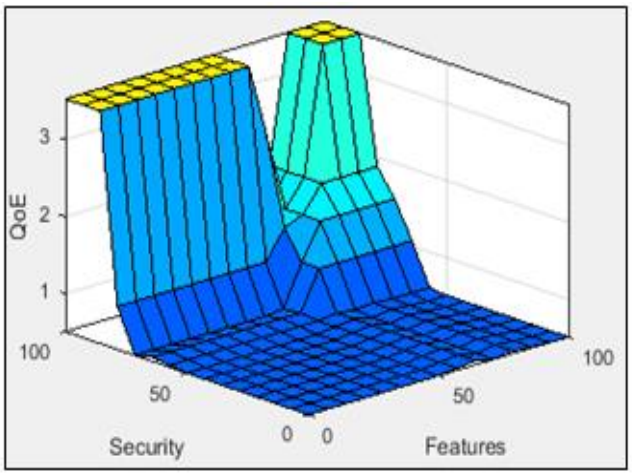


Figure 7-18 Effect of Features and Security

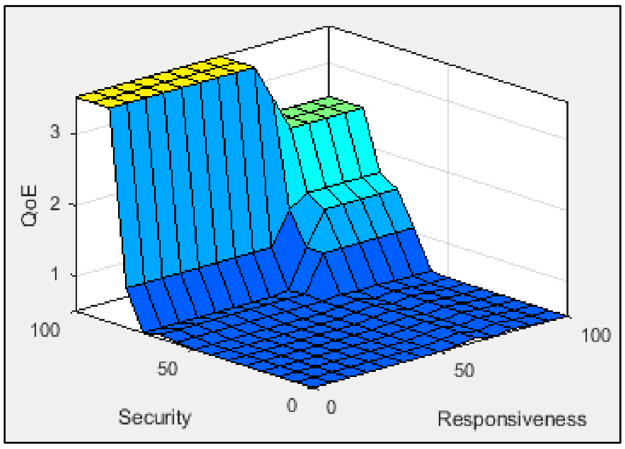


Figure 7-19 Effect of Security and Responsiveness

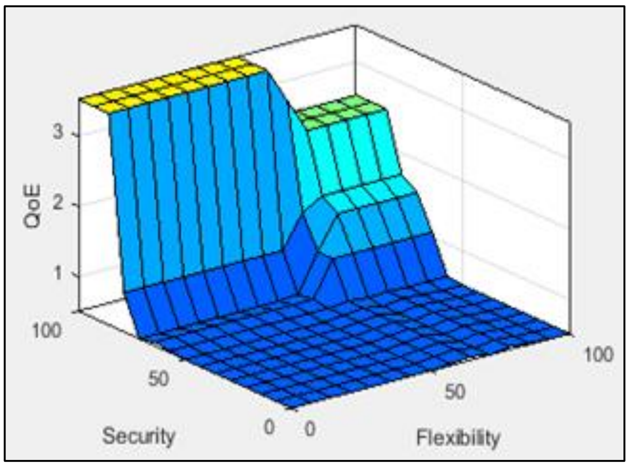


Figure 7-20 Effect of Security and Flexibility

Similar behaviour can be seen in Figure 7-21, which shows the effect of Security and Rapport on QoE. It is clear that the Rapport causes to decline QoE value to 2.5 due to the low impact of this parameter.

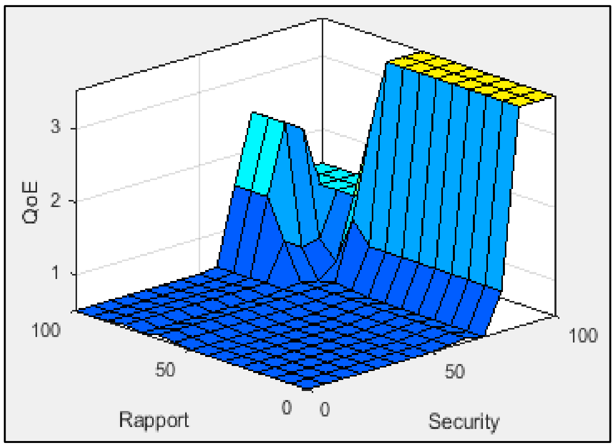


Figure 7-21 Effect of Rapport and Security

Figure 7-22 and Figure 7-23 show the effect of (Reliability – Features) and (Responsiveness – Reliability) on QoE respectively. These parameters cause the QoE value to reach 2.5 in the [low-deviation low-deviation] pairs. This is due to the high impact of Reliability. The figures reveal that Reliability has a higher effect on QoE than Features and Responsiveness.

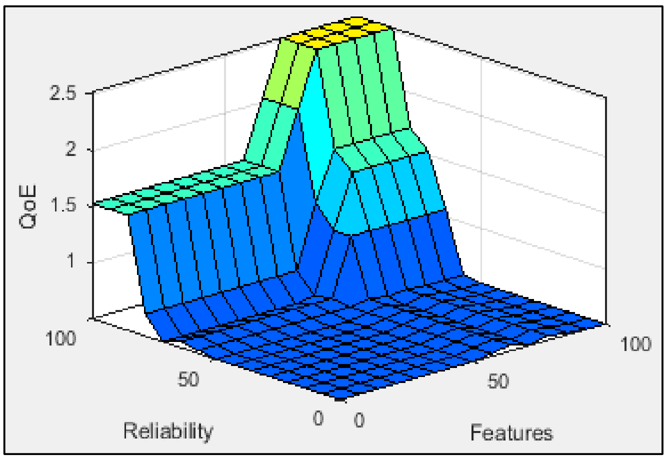


Figure 7-22 Effect of Features and Reliability

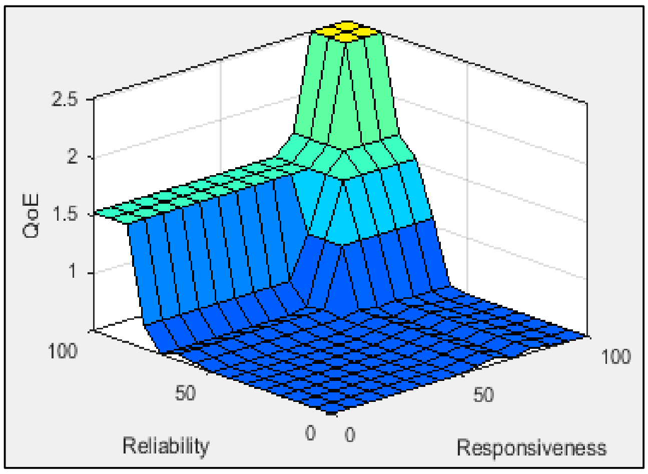


Figure 7-23 Effect of Reliability and Responsiveness

Figure 7-24 presents the influence of Rapport and Features on QoE. The maximum obtained value of QoE, in this case, is 1.5 in the [low-deviation low-deviation] pair, with neglected value of QoE value otherwise.

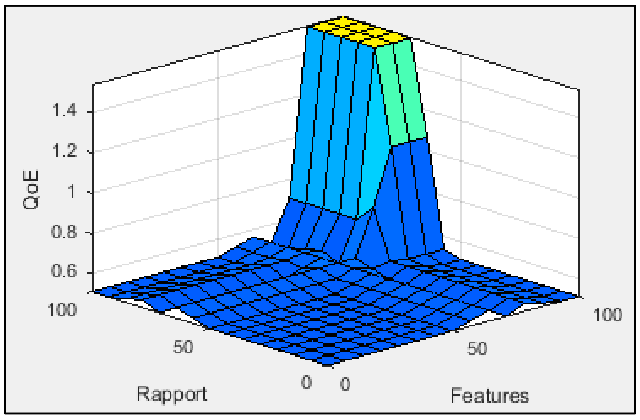


Figure 7-24 Effect of Features and Rapport

From the data obtained in Figure 7-25, Figure 7-26, and Figure 7-27, it can be seen that the adjusted fuzzy engine resulted in the lowest value of QoE which is 0.56. The reason for this behaviour is the fact that the user study considered Flexibility, Responsiveness, and Rapport as the least effective parameters on user satisfaction or QoE.

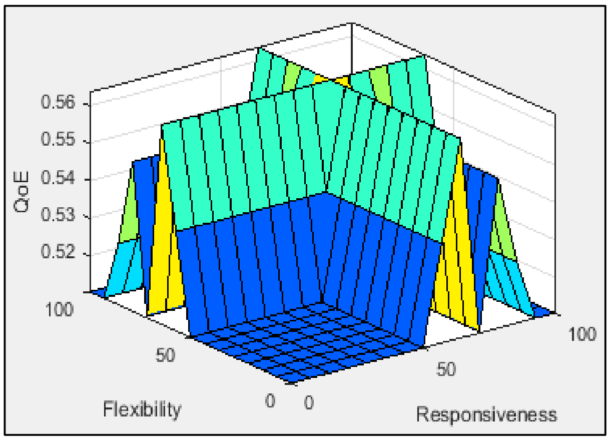


Figure 7-25 Effect of Flexibility and Responsiveness

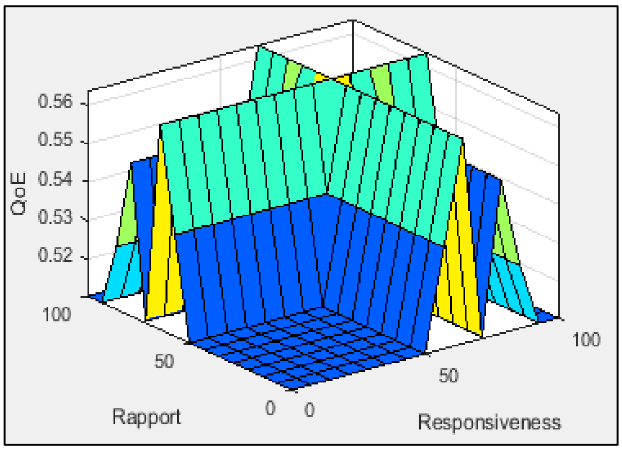


Figure 7-26 Effect of Rapport and Responsiveness

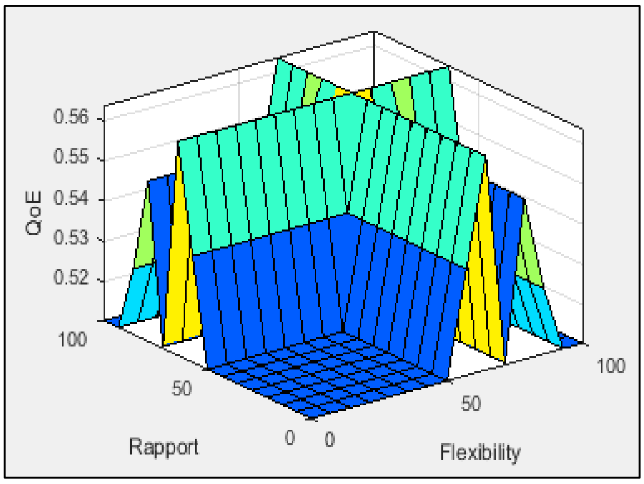


Figure 7-27 Effect of Rapport and Flexibility

Figure 7-28 and Figure 7-29 show the effect of (Reliability - Flexibility) and (Rapport - Reliability) parameters' pairs on QoE respectively. The diagrams reveal that the maximum achieved value is 1.5 in the [low-deviation low-deviation] pair. It is also clear that the effect of Flexibility is higher in [fair fair] pair in Figure 7-28, due to the higher priority of Flexibility in comparison to the Rapport.

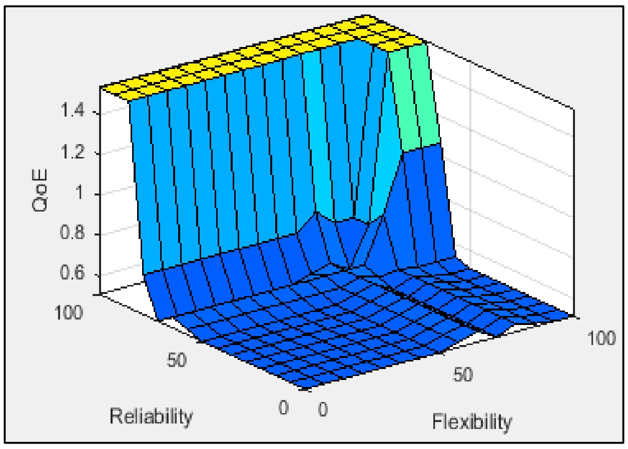


Figure 7-28 Effect of Reliability and Flexibility

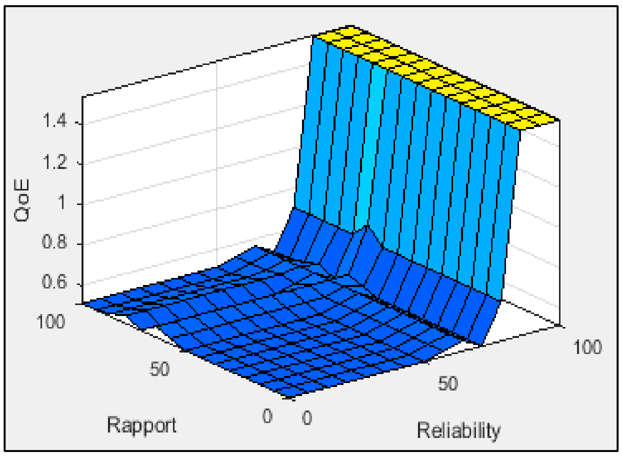


Figure 7-29 Effect of Reliability and Rapport

Figure 7-30 presents the effect of Reliability and Security on QoE, which shows a clear trend of increasing the level of QoE to reach 4.5 approximately in the [low-deviation low-deviation] values for this pair of parameters. The influence of the Security is higher than Reliability, this is obvious in the [low-deviation high-deviation] pair where the QoE value is 3.5.

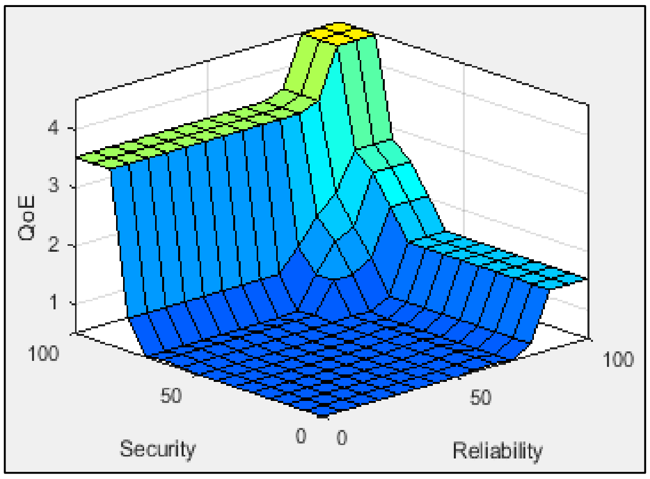


Figure 7-30 Effect of Reliability and Security

7.5.3.1 Testing the Modified Fuzzy Engine

The adjusted fuzzy engine was tested again by applying the same data used in the user study to check the produced results. Table 7-5 presents a comparison between the results obtained from the modified fuzzy engine and the user study. It can be noticed from the acquired results that the fuzzy engine results match the users' opinions.

The next section introduces a discussion for the evaluation of the proposed middleware and the metric validation, in addition to the overall conclusion obtained from the presented studies.

Table 7-5 Comparison between the results of the adjusted Fuzzy engine and the user study

Case	Input Data												Fuzzy Output		Study Output	
	Responsiveness linguistic	**Responsiveness numeric**	**Reliability linguistic**	**Reliability numeric**	**Flexibility linguistic**	**Flexibility numeric**	**Security linguistic**	**Security numeric**	**Features linguistic**	**Features numeric**	**Rapport linguistic**	**Rapport numeric**	**Fuzzy results linguistic**	**Fuzzy results numeric**	**Study results**	**Study results**
1	Bad	30	Good	88	Good	90	Good	95	Good	93	Good	99	Good	3.525	Satisfied	Good
2	Good	98	Good	92	Good	89	Good	90	Bad	40	Good	94	Fair	2.525	Neutral	Fair
3	Good	99	Good	99	Good	92	Bad	22	Good	90	Good	91	Bad	0.504	Strongly dissatisfied	Bad
4	Good	90	Good	96	Good	100	Good	89	Good	95	Bad	34	Good	3.525	Satisfied	Good
5	Good	100	Good	90	Bad	15	Good	99	Good	88	Good	95	Good	3.525	Satisfied	Good
6	Good	95	Bad	8	Good	88	Good	100	Good	97	Good	90	Poor	1.533	Dissatisfied	Poor
7	Medium	73	Bad	20	Medium	75	Medium	70	Medium	73	Good	94	Poor	1.463	Dissatisfied	Poor
8	Good	98	Good	89	Good	95	Good	90	Bad	22	Medium	74	Poor	1.781	Dissatisfied	Poor
9	Bad	50	Good	91	Medium	73	Good	95	Bad	18	Good	100	Poor	1.531	Dissatisfied	Poor
10	Good	92	Medium	74	Good	93	Bad	21	Good	89	Medium	75	Bad	0.523	Strongly dissatisfied	Bad

7.6 Discussions

To conclude, the presented experiments showed that MonSLAR indicates a clear improvement in terms of the message size overhead, where it introduces much better performance than PM because of the use of REST technology in comparison to the SOAP protocol. The study revealed that the message size overhead of MonSLAR is approximately five times less than the message size overhead caused by using SOAP in previous research (PM); this is due to the use of XML format in SOAP protocol. The results produced in this study corroborate the findings of a great deal of the previous research in this field (section 2.4.2.3). These results are consistent with those described by Mumbaikar and Padiya (2013) that confirmed the increase in the message size to be five times less in REST technology in comparison to SOAP (Mumbaikar & Padiya, 2013).

Moreover, there are similarities between the attitudes expressed by MonSLAR and those presented by Markey and Clynch (2013). Their study showed the decrease in the message size due to the use of REST technology in comparison with SOAP technology especially in the case of using JSON files, this reduction in message size in REST was two times less from that in SOAP (Markey & Clynch, 2013).

Furthermore, the study presented by Mulligan and Gra (2009) revealed that there was a reduction in the packet size for the case of using REST in comparison to SOAP technology, the results of their study showed that the packet size in REST is approximately two times less than packet size in SOAP protocol (Mulligan & Gra, 2009). It is important to be mentioned that their study considered the CRUD methods using GET, PUT, POST, and DELETE methods; while in MonSLAR, embedding the data in the header of OPTIONS method was investigated.

These results also accord with the study presented by Mohamed and Wijesekera (2012), who discussed the difference between the message size of SOAP and REST. They introduced a sample for each case showed that the payload's message size overhead in REST is twenty-five times less than in SOAP protocol (Mohamed & Wijesekera, 2012). This is due to sending the data in the message payload. Again, the data in MonSLAR was sent within the header of the message in REST OPTIONS method.

At the same time, MonSLAR makes an enhancement to the monitoring process by delivering the monitored data to the client side in addition to the SaaS service using REST architecture, by embedding the monitored data in the response of the SaaS REST service. MonSLAR also provides information about overall user satisfaction using a decision making tool. The experiments also explored the behaviour of the monitoring system, which revealed that the response time overhead with and without using MonSLAR is comparable.

The qualitative evaluation revealed that MonSLAR outperforms the other monitoring frameworks in the research field. This was achieved by comparing the features of MonSLAR with the available monitoring frameworks. These features involved the following: the ability of MonSLAR to detect any SLA violations automatically; in addition to real-time records of the monitored data. Furthermore, an automatic notify-ability feature that helps the client of the SaaS services in controlling the negotiated services. These features save the need for a third party service to achieve the monitoring process. Not to forget the communication architecture feature, which considers the use of a middleware as a tool to manage the monitoring process. MonSLAR uses SOM that provides loose coupling and reuse capabilities to the provided services; besides the interaction type used for managing the web services, MonSLAR uses REST technology, which adds lightweight characteristics and reduces the need for the use of technologies like SOAP to transmit the monitored data.

On the other hand, the user study for validating the SaaS-Qual metric showed that the QoE value depends on the combination of the model's parameters. However, the study tells that some of the parameters have higher effects on user satisfaction than others. In the current study, comparing the results obtained from both the fuzzy engine test with the study survey results indicate that Security has the highest priority for the SaaS users. Another finding was that Reliability has the second highest effect on user satisfaction. Whereas Features parameter was found to have the third highest priority, followed by Responsiveness and Flexibility having the fourth highest priority, while Rapport has the lowest priority among the parameters.

The findings of the user study have implications for adjusting the rules of the proposed fuzzy engine. It is interesting to note that the results obtained from the adjusted system revealed an overall improvement in the QoE level.

One of the issues revealed from these findings was that the SaaS-Qual model parameters can be used to estimate the QoE value, but an adaption is required to adjust the parameters weights according to the users' satisfaction and requirements.

7.7 Chapter Summary

This chapter presented a test for the main functionality of MonSLAR to present user-centric monitoring, in addition to an evaluation of the performance of the proposed middleware. The evaluation methods showed the ability of the middleware to achieve the aim of the research of monitoring the QoE value, and provided user centric monitoring using the REST architecture methods with an acceptable performance in comparison with the monitoring frameworks in the literature review.

Three different approaches have been used to evaluate MonSLAR. Firstly, a quantitative evaluation is used to investigate the overhead caused by MonSLAR in terms of the message size. Secondly, the proposed middleware is evaluated using qualitative study by comparing the main characteristics of MonSLAR with the available monitoring frameworks presented in the literature review. Finally, another evaluation has been introduced to evaluate estimating the QoE value using fuzzy logic; this was achieved by conducting a questionnaire survey.

The next chapter concludes the thesis, discussing the main achievements of the research and proposing a set of recommendations for improvements in future research.

CHAPTER EIGHT

CONCLUSIONS AND RECOMMENDATIONS

8.1 Introduction

Monitoring Cloud services has become a leading driver in assuring the compliance of an SLA and reserving the rights of both the cloud client and server. This thesis introduced a set of approaches and techniques to monitor SaaS in cloud computing. The research presented MonSLAR, a Middleware for Monitoring SLA in SaaS cloud computing using REST technology. This chapter concludes the thesis by discussing the overall results of the study; it reflects on the main objectives of this research and how these objectives were fulfilled, it also discusses how these objectives were combined together to achieve the aim, in addition to the directions that can be taken as a future extension of this research.

8.2 Conclusions

The importance of the proposed middleware appears in the ability to manage the client-provider relationship through providing an autonomic real-time monitoring for SaaS, and introducing REST as the connection technology in SOM instead of relying on a SOAP protocol to achieve this, as SOAP technology adds a considerable amount of overhead to the monitoring process caused by the use of XML messages to transmit the data. This section presents an overall summary of this research. The thesis handled the problem of monitoring an SLA of SaaS from a user's perspective. The main contribution of this research is the development of the proposed middleware. The monitoring of the services from a user's perspective considered estimating a QoE of SaaS by presenting a fuzzy inference rule based engine, the study of estimating a QoE in this thesis can be a base for future studies to evaluate a QoE using a holistic, unified metric.

8.3 Achievements of the Aim and Objectives

The aim of the submitted thesis was to propose a user centric approach for monitoring SaaS in cloud computing and to reduce the overhead caused by the monitoring process. The proposed monitoring approach was successfully developed using REST technology. Four objectives were proposed and achieved successfully to fulfill the aim of the research (see Section 1.4). The progress of developing each objective and the achievements of each objective are as follows:

The **first objective** was to review the related work and define the research problem and the weakness in the research area. This was addressed in Chapter Two and Chapter Three of this thesis.

The **second objective** was to develop an approach for lightweight user-centric monitoring of SaaS in cloud computing and to assure the delivery of the monitored data to the client side. This is achieved by the design of a SOM capable of delivering the monitored data to the client side; this middleware offers a loosely coupled, reusable, and platform independent components that helped in managing the monitoring process between the client side and the server side. The lightweight feature of MonSLAR was achieved by the use of REST technology and proposing an approach for embedding the monitored data in the requests and responses of HEAD and OPTIONS methods; this reduced the need to use dedicated messages for transmitting the monitored data. Exploiting REST in MonSLAR helped to reduce the overhead caused by the monitoring process. The use of REST in the design and the implementation of the proposed middleware were detailed in Chapters Four and Six respectively.

The **third objective** was to develop an approach to measure user satisfaction with services provided in cloud computing in terms of QoE. The monitoring of the SLA and checking the user satisfaction was achieved by estimating the QoE value as an indication of user satisfaction, this was fulfilled by the design of a fuzzy logic system capable of estimating the QoE based on the monitored QoS parameters and SLA parameters. The design of the proposed fuzzy logic system was discussed in Chapter Five.

The **fourth objective** was to evaluate the proposed system to check its performance. This was presented in Chapter Seven. Figure 8-1 illustrates the progression in the research objectives and the use of the methods and techniques to achieve the aim of the research.

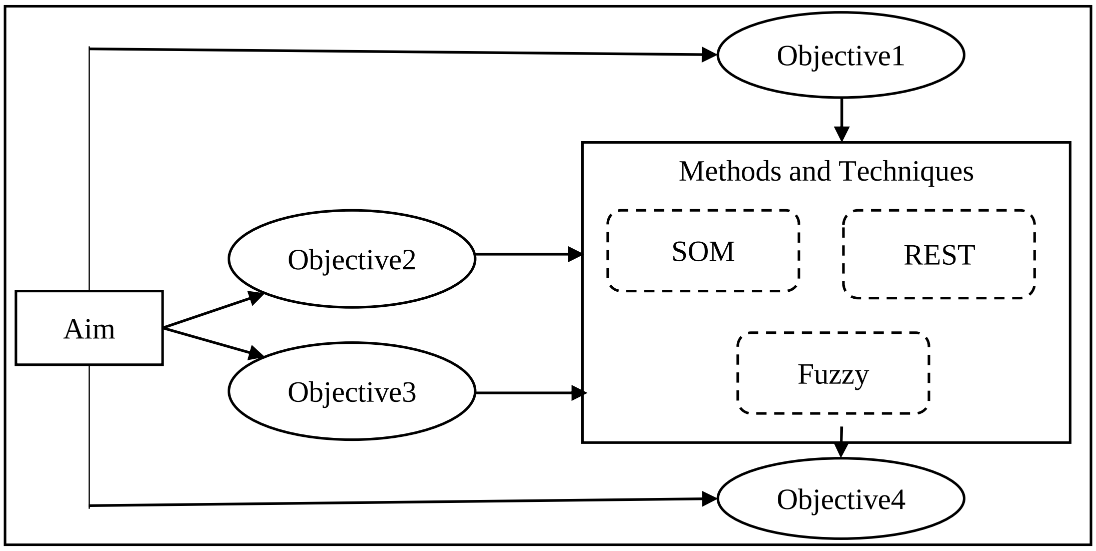


Figure 8-1 The progress of the objectives to achieve the research aim.

8.4 Research Limitations

The main limitation of this study lies in the fact that the measurement of the SLA parameters and the SaaS-Qual factors are out of the scope of this research. The measurements are considered available and stored in a database, as the current study was not specifically presented to define measurements or estimations for these parameters.

8.5 Recommendations for Future Research

This section introduces a set of research directions that can be taken in future research. Future research can follow two main directions, the first direction is related to extending MonSLAR functionalities, while the second direction includes more research on the proposed QoE metric. The next subsections present the recommendations for future research.

8.5.1 Research related to MonSLAR

The first direction of the research considers MonSLAR middleware. The research findings introduce the following insights for future research:

1- Further research might investigate extending the client side of MonSLAR, this extension is proposed to perform the measurements of the QoS that are sent to the provider side of MonSLAR. This extension is related to the monitoring request-b, which includes sending the monitored data from the client side to the provider side using PUT and POST methods.

It is also important to consider how frequently should the measurements in the client side be obtained, this can be decided by the service provider and the client at the first time of using the SaaS service. For example, the Rapport and Features can be measured once a week or once a month by asking the client to complete a form to measure their values; while for Reliability, this can be measured automatically for each received response.

The automation of the POST request activation can be managed either by using a time-based invocation or event-based invocation. Time-based invocation requires making a decision about how often to send this data, while the event-based invocation requires a client action to manage sending this data, which includes using a GUI with a mouse click to activate this process.

It is also important to define a way for mapping the parameter threshold factor values which are not part of the negotiated SLA document, to be included in the decision making process. This can be managed by using a GUI for inputting the threshold values of the parameters that are not defined in the SLA document. The use of GUI allows the user to input these values at the first time of using the service. These values are sent to the MonSLAR-server side to be used later by the fuzzy logic engine to measure the value of the QoE.

2- It would be interesting to investigate the action to be considered in the case of SLA violation. This action could potentially mean terminating the contract between the client and the service provider, or calculating a compensation to be paid to the client as a result of violating the contract.

3- Further experimental investigations are required to perform more evaluations in a cloud environment. This includes checking the effect of increasing the number of clients, and the number of servers' virtual machines and SaaS containers as an indication of the multi-tenancy of SaaS on the performance of the proposed middleware. This exploration can help to check the effect of changing the number of underlying cloud resources like virtual machines on the performance of MonSLAR.

8.5.2 Research related to the proposed QoE metric

The second direction is to extend the research with respect to the applicability of SaaS-Qual as a model for estimating QoE. Considerably, more work will need to be done to estimate each of the SaaS-Qual parameters, this includes studies for the main QoS metrics and formulae to define these metrics.

8.6 Chapter Summary

This chapter has presented an overall conclusion for the thesis. The aim of the research, the objectives, and the techniques that were used by the researcher to achieve the study objectives were addressed; and in final conclusion, the chapter outlined potential future directions which could be adopted as further research work.

REFERENCES

Aceto, G., Botta, A., De Donato, W., & Pescapè, A. (2013). Cloud monitoring: A survey. *Computer Networks, 57*(9), 2093-2115.

Adinolfi, O., Cristaldi, R., Coppolino, L., & Romano, L. (2012). *QoS-MONaaS: a portable architecture for QoS monitoring in the cloud.* Paper presented at the 2012 Eighth International Conference on Signal Image Technology and Internet Based Systems (SITIS), Sorrento, Italy.

Al-Jaroodi, J., & Mohamed, N. (2012). Service-oriented middleware: a survey. *Journal of Network and Computer Applications, 35*(1), 211-220.

Al-Shammari, S., & Al-Yasiri, A. (2014). *Defining a metric for measuring QoE of SaaS cloud computing*. Paper presented at the 15th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNET 2014), Liverpool, UK.

Al-Shammari, S., & Al-Yasiri, A. (2015). *MonSLAR: a middleware for monitoring SLA for RESTFUL services in cloud computing.* Paper presented at the IEEE 9th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA), Bremen, Germany.

Alhamad, M., Dillon, T., & Chang, E. (2010). *Conceptual SLA framework for cloud computing.* Paper presented at the 2010 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST), Dubai, United Arab Emirates.

Alhamad, M., Dillon, T., & Chang, E. (2011). Trust-evaluation metric for cloud applications. *International Journal of Machine Learning and Computing, 1*(4), 416-421.

Alsulaiman, L. A., & Alturki, R. (2012). *Monitoring multimedia quality of service in public Cloud Service Level Agreements.* Paper presented at the 2012 International Conference on Multimedia Computing and Systems (ICMCS), Tangiers, Morocco.

Amato, A., Di Martino, B., & Venticinque, S. (2012). *Evaluation and brokering of service level agreements for negotiation of cloud infrastructures.* Paper presented at the 2012 International Conferece For Internet Technology And Secured Transactions, London, United Kingdom.

Amato, A., Liccardo, L., Rak, M., & Venticinque, S. (2014). SLA-based negotiation and brokering of cloud resources. *International Journal of Cloud Computing, 3*(1), 24-44.

Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., . . . Xu, M. (2007). *Web services agreement specification (WS-Agreement).* Paper presented at the Open Grid Forum.

Anithakumari, S., & Chandrasekaran, K. (2015). *Monitoring and Management of Service Level Agreements in Cloud Computing.* Paper presented at the 2015 International Conference on Cloud and Autonomic Computing (ICCAC), Boston, MA, USA.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., . . . Stoica, I. (2010). A view of cloud computing. *Communications of the ACM, 53*(4), 50-58.

Aversa, R., Panza, N., & Tasquier, L. (2015). *An Agent-Based Platform for Cloud Applications Performance Monitoring.* Paper presented at the 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS), Brazil.

Azeez, A., Perera, S., Gamage, D., Linton, R., Siriwardana, P., Leelaratne, D., . . . Fremantle, P. (2010). *Multi-tenant SOA middleware for cloud computing.* Paper presented at the 2010 IEEE 3rd International Conference on Cloud Computing (Cloud), USA.

Badidi, E. (2013). A framework for software-as-a-service selection and provisioning. *International Journal of Computer Networks & Communications, 5*(3).

Bailey, J. E., & Pearson, S. W. (1983). Development of a tool for measuring and analyzing computer user satisfaction. *Management science, 29*(5), 530-545.

Baliyan, N., & Kumar, S. (2013). *Quality assessment of software as a service on cloud using fuzzy logic.* Paper presented at the 2013 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), Bangalore, India.

Bansal, D., Patel, P., & Greenberg, A. (2016). *Multi-tenant middleware cloud service technology. US Patent Office 20160149813 A1* Retrieved from https://www.google.com/patents/US20160149813

Bass, L., Clements, P., & Kazman, R. (2013). *Software Architecture in Practice*: Addison-Wesley.

Benlian, A., Koufaris, M., & Hess, T. (2011). Service quality in software-as-a-service: developing the SaaS-Qual measure and examining its role in usage continuance. *Journal of Management Information Systems, 28*(3), 85-126.

Bernstein, P. A. (1996). Middleware: a model for distributed system services. *Communications of the ACM, 39*(2), 86-98.

Bezemer, C.-P., & Zaidman, A. (2010). *Multi-tenant SaaS applications: maintenance dream or nightmare?* Paper presented at the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), Belgium.

Bianco, P., Lewis, G. A., & Merson, P. (2008). *Service level agreements in service-oriented architecture environments*. Retrieved from http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA528751

Bloomberg, J. (2013). *The Agile Architecture Revolution: How Cloud Computing, REST-based SOA, and Mobile Computing are Changing Enterprise IT*. Hoboken, NJ: John Wiley & Sons.

Blumel, F., Metsch, T., & Papaspyrou, A. (2011). *A restful approach to service level agreements for cloud environments.* Paper presented at the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC), Sydney, Australia.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004). Web services architecture.   Retrieved from https://www.w3.org/TR/ws-arch/

Bora, A., & Bezboruah, T. (2015). A Comparative Investigation on Implementation of RESTful versus SOAP based Web Services. *International Journal of Database Theory and Application, 8*(3), 297-312.

Brandic, I., Dustdar, S., Anstett, T., Schumm, D., Leymann, F., & Konrad, R. (2010). *Compliant cloud computing (c3): Architecture and language support for user-driven compliance management in clouds.* Paper presented at the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), USA.

Brandic, I., Emeakaroha, V. C., Netto, M. A., & De Rose, C. A. (2015). Application-Level Monitoring and SLA Violation Detection for Multi-Tenant Cloud Services. *Emerging Research in Cloud Distributed Computing Systems*, 157.

Brandic, I., Music, D., Leitner, P., & Dustdar, S. (2009). Vieslaf framework: Enabling adaptive and versatile sla-management *Grid Economics and Business Models* (pp. 60-73): Springer.

Bray, T. (2014). The JavaScript Object Notation (JSON) Data Interchange Format.

Brooks, P., & Hestnes, B. (2010). User measures of quality of experience: why being objective and quantitative is important. *Network, IEEE, 24*(2), 8-13.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems, 25*(6), 599-616.

Cai, H., Reinwald, B., Wang, N., & Guo, C. J. (2012). Saas multi-tenancy: Framework, technology, and case study. *Cloud Computing Advancements in Design, Implementation, and Technologies*, 67.

Casas, P., Fischer, H. R., Suette, S., & Schatz, R. (2013). *A first look at quality of experience in Personal Cloud Storage services.* Paper presented at the 2013 IEEE International Conference on Communications Workshops (ICC), Budapest, Hungary.

Casas, P., Sackl, A., Egger, S., & Schatz, R. (2012). *YouTube & Facebook Quality of Experience in mobile broadband networks.* Paper presented at the 2012 IEEE Globecom Workshops (GC Wkshps), Anaheim, CA, USA

Casas, P., & Schatz, R. (2014). Quality of Experience in Cloud services: Survey and measurements. *Computer Networks, 68*, 149–165.

Casas, P., Seufert, M., Egger, S., & Schatz, R. (2013). *Quality of experience in remote virtual desktop services.* Paper presented at the 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), Ghent, Belgium.

Cedillo, P., Gonzalez-Huerta, J., Abrahao, S., & Insfran, E. (2016). A Monitoring Infrastructure for the Quality Assessment of Cloud Services *Transforming Healthcare Through Information Systems* (pp. 17-32). New York: Springer.

Cedillo, P., Jimenez-Gomez, J., Abrahao, S., & Insfran, E. (2015). *Towards a Monitoring Middleware for Cloud Services.* Paper presented at the 2015 IEEE International Conference on Services Computing (SCC), New York, USA.

Chauhan, T., Chaudhary, S., Kumar, V., & Bhise, M. (2011). *Service level agreement parameter matching in cloud computing.* Paper presented at the 2011 World

Congress on Information and Communication Technologies (WICT), Mumbai, India.

Chavda, K. F. (2004). Anatomy of a Web service. *Journal of Computing Sciences in Colleges, 19*(3), 124-134.

Cheng, X., Shi, Y., & Li, Q. (2009). *A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on SLA.* Paper presented at the 2009 Joint Conferences on Pervasive Computing (JCPC), Taiwan.

Cicotti, G., Coppolino, L., Cristaldi, R., D'Antonio, S., & Romano, L. (2012). *QoS monitoring in a cloud services environment: the SRT-15 approach.* Paper presented at the Euro-Par 2011: Parallel Processing Workshops, France.

Cingolani, P., & Alcalá-Fdez, J. (2013). jFuzzyLogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming. *International Journal of Computational Intelligence Systems, 6*(sup1), 61-75.

Cloud Armor. (n.d.). The Project Website. Retrieved from http://cs.adelaide.edu.au/~cloudarmor/ds.html

Comuzzi, M., Kotsokalis, C., Spanoudakis, G., & Yahyapour, R. (2009). *Establishing and monitoring SLAs in complex service based systems.* Paper presented at the 2009 IEEE International Conference on Web Services, 2009. ICWS, Los Angeles.

Creswell, J. W. (2013). *Research design: Qualitative, quantitative, and mixed methods approaches*. London: Sage.

CSMIC. (2011). Service Measurement Index Version 1.0: CSMIC.

Cusumano, M. (2010). Cloud computing and SaaS as new computing platforms. *Communications of the ACM, 53*(4), 27-29.

D'Ambrogio, A. (2006). *A model-driven wsdl extension for describing the qos ofweb services.* Paper presented at the 2006 IEEE International Conference on Web Services. ICWS'06, Chicago, Illinois.

Da Cunha Rodrigues, G., Calheiros, R. N., Guimaraes, V. T., Santos, G. L. d., de Carvalho, M. B., Granville, L. Z., . . . Buyya, R. (2016). *Monitoring of cloud computing environments: concepts, solutions, trends, and future directions.* Paper presented at the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy.

Daigneau, R. (2011). *Service Design Patterns: fundamental design solutions for SOAP/WSDL and restful Web Services*: Addison-Wesley.

Dawson, C. (2002). *Practical research methods: A user-friendly guide to mastering research*. Newtec Place, UK: How to Books Ltd.

de Oliveira, D., Ogasawara, E., Baião, F., & Mattoso, M. (2010). *Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows.* Paper presented at the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), Miami, Florida, USA.

Decat, M., Bogaerts, J., Lagaisse, B., & Joosen, W. (2015). *Amusa: middleware for efficient access control management of multi-tenant SaaS applications.* Paper presented at the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain.

DigitalOcean. Cloud computing, designed for developers. Retrieved from https://www.digitalocean.com/

Docker. (2017). Docker is the world's leading software containerization platform. Retrieved from https://www.docker.com/

Emeakaroha, V. C. (2012). *Managing Cloud Service Provisioning and SLA Enforcement via Holistic Monitoring Techniques.* (PhD Thesis), Vienna University of Technology.

Emeakaroha, V. C., Brandic, I., Maurer, M., & Dustdar, S. (2010). *Low level Metrics to High level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments.* Paper presented at the 2010 International Conference on High Performance Computing & Simulation, France.

Emeakaroha, V. C., Ferreto, T. C., Netto, M. A. S., Brandic, I., & De Rose, C. A. (2012). *Casvid: Application level monitoring for sla violation detection in clouds.* Paper presented at the 2012 IEEE 36th Annual Computer Software and Applications Conference (COMPSAC), Turkey.

Erl, T. (2008). *Soa: principles of service design* (Vol. Prentice Hall): Upper Saddle River.

Fiedler, M., Hossfeld, T., & Tran-Gia, P. (2010). A generic quantitative relationship between quality of experience and quality of service. *Network, IEEE, 24*(2), 36-41.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). Hypertext transfer protocol–HTTP/1.1: RFC 2616, June.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures.* University of California, Irvine.

Finn, A., Vredevoort, H., Lownds, P., & Flynn, D. (2012). *Microsoft private cloud computing*: John Wiley & Sons.

Firdhous, M., Hassan, S., & Ghazali, O. (2013a). A Comprehensive Survey on Quality of Service Implementations in Cloud Computing. *International Journal of Scientific & Engineering Research, 4*(5), 118-123.

Firdhous, M., Hassan, S., & Ghazali, O. (2013b). Monitoring, Tracking and Quantification of Quality of Service in Cloud Computing. *International Journal of Scientific & Engineering Research, 4*(5).

Furht, B. (2010). Cloud computing fundamentals *Handbook of cloud computing* (pp. 3-19): Springer.

Gao, J., Bai, X., Tsai, W.-T., & Uehara, T. (2013). *SaaS Testing on Clouds-Issues, Challenges and Needs.* Paper presented at the 2013IEEE 7th International Symposium on Service Oriented System Engineering (SOSE), USA.

Gartner. (2015). Market Trends: Future Look at SaaS in the Application Markets. Retrieved from https://www.gartner.com/doc/3172034/market-trends-future-look-saas

Geebelen, K., Walraven, S., Truyen, E., Michiels, S., Moens, H., De Turck, F., . . . Joosen, W. (2012). *An open middleware for proactive QoS-aware service composition in a multi-tenant SaaS environment.* Paper presented at the International Conference on Internet Computing (ICOMP), Athens.

Godse, M., & Mulik, S. (2009). *An approach for selecting software-as-a-service (SaaS) product.* Paper presented at the 2009 IEEE International Conference on Cloud Computing. CLOUD'09, India.

Google Trends. (2016). REST vs SOAP search trend.   Retrieved from https://trends.google.co.uk/trends/explore?q=RESTful%20API,SOAP%20API

Gustavo, A., Casati, F., Kuno, H., & Machiraju, V. (2004). *Web services: concepts, architectures and applications*. Berlin Springer.

Hammadi, A. M., & Hussain, O. (2012). *A framework for SLA assurance in cloud computing.* Paper presented at the 2012 26th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Japan.

Han, H., Kim, S., Jung, H., Yeom, H. Y., Yoon, C., Park, J., & Lee, Y. (2009). *A RESTful approach to the management of cloud infrastructure.* Paper presented at the 2009 IEEE International Conference on Cloud Computing, India.

Hasan, M. S., & Huh, E.-N. (2013). *Maximizing SLA and QoE in Heterogeneous Cloud Computing Environment.* Paper presented at the International Conference on Grid Computing and Applications (GCA), Athens.

Hill, R., Hirsch, L., Lake, P., & Moshiri, S. (2012). *Guide to cloud computing: principles and practice*: Springer.

Hobfeld, T., Schatz, R., Varela, M., & Timmerer, C. (2012). Challenges of QoE management for cloud applications. *Communications Magazine, IEEE, 50*(4), 28-36.

Hurwitz, J., Bloor, R., Kaufman, M., & Halper, F. (2010). *Cloud computing for dummies*: John Wiley & Sons.

Incki, K., Ari, I., & Sözer, H. (2012). *A survey of software testing in the cloud.* Paper presented at the 2012 IEEE Sixth International Conference on Software Security and Reliability Companion (SERE-C), USA.

Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadist, P., Autili, M., . . . Hamida, A. B. (2011). Service-oriented middleware for the future internet: state of the art and research directions. *Journal of Internet Services and Applications, 2*(1), 23-45.

Jantzen, J. (2013). *Foundations of fuzzy control: a practical approach*: John Wiley & Sons.

Jarschel, M., Schlosser, D., Scheuring, S., & Hoßfeld, T. (2013). Gaming in the clouds: QoE and the users' perspective. *Mathematical and Computer Modelling, 57*(11), 2883-2894.

Jin, L.-J., Machiraju, V., & Sahai, A. (2002). Analysis on service level agreement of web services. *HP June*, 19.

Kafetzakis, E., Koumaras, H., Kourtis, M. A., & Koumaras, V. (2012). *QoE4CLOUD: A QoE-driven multidimensional framework for cloud environments.* Paper presented at the 2012 International Conference on Telecommunications and Multimedia (TEMU), Greece.

Katsaros, G., Kübert, R., & Gallizo, G. (2011). *Building a service-oriented monitoring framework with rest and nagios.* Paper presented at the 2011 IEEE International Conference on Services Computing (SCC), USA.

Kavis, M. J. (2014). *Architecting the cloud: Design decisions for cloud computing service models (SaaS, PaaS, AND IaaS)*: John Wiley & Sons.

Keller, A., & Ludwig, H. (2003). The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management, 11*(1), 57-81.

Kertesz, A., Kecskemeti, G., & Brandic, I. (2009). *An SLA-based resource virtualization approach for on-demand service provision.* Paper presented at the 3rd international workshop on Virtualization technologies in distributed computing, Barcelona, Spain.

Khaddaj, S., Arul, J. M., Chung, H.-Y., Ko, H.-Y., Dugki, M., EunmiChoi, V. K. K., . . . James, M. (2014). QoS and SLA in Cloud Computing. *International Journal of Emerging Trends in Computing and Communication Technology, 1*(1), 1-6.

Khan, H. M., Chan, G.-Y., & Chua, F.-F. (2016). *An adaptive monitoring framework for ensuring accountability and quality of services in cloud computing.* Paper presented at the 2016 International Conference on Information Networking (ICOIN), Kota Kinabalu, Malaysia.

Krakowiak, S. (2007). *Middleware Architecture with Patterns and Frameworks*.

Kübert, R., Katsaros, G., & Wang, T. (2011). *A RESTful implementation of the WS-Agreement specification.* Paper presented at the Second International Workshop on RESTful Design, Hyderabad, India.

Kurbel, K. E. (2008). *The Making of Information Systems: Software Engineering and Management in a Globalized World*: Springer Science & Business Media.

Lampesberger, H., & Rady, M. (2015). Monitoring of client-cloud interaction *Correct Software in Web Applications and Web Services* (pp. 177-228): Springer.

Larson, K. D. (1998). The role of service level agreements in IT service delivery. *Information Management & Computer Security, 6*(3), 128-132.

Le Callet, P., Möller, S., & Perkis, A. (2012). *Qualinet white paper on definitions of quality of experience European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003)* Retrieved from https://hal.archives-ouvertes.fr/hal-00977812/document

Lee, J. Y., Lee, J. W., & Kim, S. D. (2009). *A quality model for evaluating software-as-a-service in cloud computing.* Paper presented at the 2009 Seventh ACIS International Conference on Software Engineering Research, Management and Applications. SERA'09, China.

Lee, S.-Y., Tang, D., Chen, T., & Chu, W.-C. (2012). *A QoS Assurance middleware model for enterprise cloud computing.* Paper presented at the 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW), Izmir, Turkey.

Lee, Y.-C., Ma, C.-M., & Chou, S.-C. (2005). *A service-oriented architecture for design and development of middleware.* Paper presented at the 12th Asia-Pacific Software Engineering Conference (APSEC'05), Taipei, Taiwan.

Leymann, C., Fehling, F., Retter, R., Schupeck, W., & Arbitter, P. (2014). *Cloud computing patterns*. London: Springer.

Lu, X., Yin, J., Xiong, N. N., Deng, S., He, G., & Yu, H. (2016). JTangCMS: An efficient monitoring system for cloud platforms. *Information sciences, 370*, 402-423.

Marinescu, D. C. (2013). *Cloud Computing: Theory and Practice*: Newnes.

Markey, P., & Clynch, G. (2013). *A performance analysis of WS-*(SOAP) and RESTful Web Services for Implementing Service and Resource Orientated Architectures*. Paper presented at the 12th Information Technology and Telecommunications (IT&T) Conference, Athlone.

Marks, E. A., & Lozano, B. (2010). *Executive's guide to cloud computing*: John Wiley and Sons.

Marpaung, J. A., Sain, M., & Lee, H.-J. (2013). *Survey on middleware systems in cloud computing integration.* Paper presented at the 2013 15th International Conference on Advanced Communication Technology (ICACT), PyeongChang, Korea (South).

Mastelic, T., Emeakaroha, V. C., Maurer, M., & Brandic, I. (2012). *M4Cloud-Generic Application Level Monitoring for Resource-shared Cloud Environments.* Paper presented at the 2nd International Conference on Cloud Computing and Services Science (CLOSER), Portugal.

Mathur, P., & Nishchal, N. (2010). *Cloud computing: New challenge to the entire computer industry.* Paper presented at the 2010 1st International Conference on Parallel Distributed and Grid Computing (PDGC), India.

MathWorks. (2017). Defuzzification Methods. Retrieved from http://uk.mathworks.com/help/fuzzy/examples/defuzzification-methods.html#zmw57dd0e2380

Matulin, M., & Mrvelj, Š. (2013). State-of-the-practice in evaluation of quality of experience in real-life environments. *PROMET-Traffic&Transportation, 25*(3), 255-263.

McNeill, F. M., & Thro, E. (1994). *Fuzzy logic: a practical approach*. London: Academic Press.

Mell, P., & Grance, T. (2010). The NIST definition of cloud computing. *Communications of the ACM, 53*(6), 50.

Mendel, J. M. (1995). Fuzzy logic systems for engineering: a tutorial. *Proceedings of the IEEE, 83*(3), 345-377.

Menken, I., & Blokdijk, G. (2009). *Saas and web applications specialist level complete certification kit-software as a service study guide book and online course*: Emereo Pty Ltd.

Microsoft Azure. (2017). Microsoft Azure. Retrieved from http://azure.microsoft.com/en-us/

Miller, M. (2008). *Cloud computing: Web-based applications that change the way you work and collaborate online*: Que publishing.

Mohamed, K., & Wijesekera, D. (2012). Performance analysis of web services on mobile devices. *Procedia Computer Science, 10*, 744-751.

Momm, C., & Krebs, R. (2011). *A Qualitative Discussion of Different Approaches for Implementing Multi-Tenant SaaS Offerings.* Paper presented at the Software Engineering (Workshops), Karlsruhe, Germany.

Montes, J., Sánchez, A., Memishi, B., Pérez, M. S., & Antoniu, G. (2013). GMonE: A complete approach to cloud monitoring. *Future Generation Computer Systems, 29*(8), 2026-2040.

Mosallanejad, A., Atan, R., Murad, M. A., & Abdullah, R. (2014). A Hierarchical Self-Healing SLA for Cloud Computing. *International Journal of Digital Information and Wireless Communications (IJDIWC), 4*(1), 43-52.

Motta, G., You, L., Sacco, D., & Sfondrini, N. (2013). *Cloud computing: the issue of service quality: an overview of cloud service level management architectures.* Paper presented at the 2013 Fifth International Conference on Service Science and Innovation (ICSSI), Kaohsiung, Taiwan.

Motta, G., You, L., Sfondrini, N., Sacco, D., & Ma, T. (2014). *Service level management (slm) in cloud computing-third party slm framework.* Paper presented at the 2014 IEEE 23rd International WETICE Conference (WETICE), Parma, Italy.

Moustafa, S., Elgazzar, K., Martin, P., & Elsayed, M. (2015). *SLAM: SLA Monitoring Framework for Federated Cloud Services.* Paper presented at the 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), Cyprus.

Muller, C., Oriol, M., Franch, X., Marco, J., Resinas, M., Ruiz-Cortes, A., & Rodríguez, M. (2014). Comprehensive explanation of SLA violations at runtime. *IEEE Transactions on Services Computing, 7*(2), 168-183.

Müller, C., Oriol, M., Rodríguez, M., Franch, X., Marco, J., & Resinas, M. (2012). *SALMonADA: A platform for monitoring and explaining violations of WS-agreement-compliant documents.* Paper presented at the 4th International Workshop on Principles of Engineering Service-Oriented Systems, Zurich, Switzerland.

Mulligan, G., & Gra, D. (2009). *A comparison of SOAP and REST implementations of a service based interaction independence middleware framework.* Paper presented at the 2009 Winter Simulation Conference (WSC), Austin, Texas.

Mumbaikar, S., & Padiya, P. (2013). Web services based on soap and rest principles. *International Journal of Scientific and Research Publications, 3*(5).

MySQL. (2017). MySQL Community Server.   Retrieved from https://dev.mysql.com/downloads/mysql/5.6.html

Naaz, S., Alam, A., & Biswas, R. (2011). Effect of different defuzzification methods in a fuzzy based load balancing application. *IJCSI, 8*(5), 261-267.

Nguyen, T. A. B., Siebenhaar, M., Hans, R., & Steinmetz, R. (2014). *Role-Based Templates for Cloud Monitoring.* Paper presented at the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC), London, United Kingdom.

Oriol, M., Franch, X., & Marco, J. (2015). Monitoring the service-based system lifecycle with SALMon. *Expert Systems with Applications, 42*(19), 6507-6521.

Papazoglou, M. (2008). *Web services: principles and technology*: Pearson Education.

Parasuraman, A., Zeithaml, V. A., & Berry, L. L. (1988). Servqual. *Journal of retailing, 64*(1), 12-37.

Perez-Espinoza, J., Sosa-Sosa, V. J., Gonzalez, J., & Tello-Leal, E. (2015). *A Distributed Architecture for Monitoring Private Clouds.* Paper presented at the

2015 26th International Workshop on Database and Expert Systems Applications (DEXA), Valencia, Spain.

Pilevari, N., Toloei, A., & Sanaei, M. (2013). A model for evaluating cloud-computing users' satisfaction. *African Journal of Business Management, 7*(16), 1405-1413.

Povedano-Molina, J., Lopez-Vega, J. M., Lopez-Soler, J. M., Corradi, A., & Foschini, L. (2013). DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds. *Future Generation Computer Systems, 29*(8), 2041-2056.

Puder, A. (2006). *Distributed systems architecture: a middleware approach*: Elsevier.

Qilin, L., & Mintian, Z. (2010). *The state of the art in middleware.* Paper presented at the 2010 International Forum on Information Technology and Applications (IFITA), China.

Rak, M., Cuomo, A., & Villano, U. (2013). *A Proposal of a Simulation-based Approach for Service Level Agreement in Cloud.* Paper presented at the 2013 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Barcelona, Spain.

Rak, M., Venticinque, S., Máhr, T., Echevarria, G., & Esnal, G. (2011). *Cloud application monitoring: The mOSAIC approach.* Paper presented at the 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), Athens, Greece.

Rana, O., Warnier, M., Quillinan, T. B., & Brazier, F. (2008). Monitoring and reputation mechanisms for service level agreements *Grid Economics and Business Models* (pp. 125-139). Berlin: Springer.

Rehman, Z.-u., Hussain, O. K., & Hussain, F. K. (2015). User-side cloud service management: State-of-the-art and future directions. *Journal of Network and Computer Applications, 55*, 108-122.

Reichl, P., & Zwickl, P. (2015). The Economics of Quality of Experience: Recent Advances and Next Steps. *Invited Paper, IEEE COMSOC MMTC E-Letter, 10*(3).

Richardson, L., Amundsen, M., Amundsen, M., & Ruby, S. (2013). *RESTful Web APIs*. Sebastopol, Calif.: O'Reilly Media.

Richardson, L., & Ruby, S. (2008). *RESTful web services*. Sebastopol, Calif.: O'Reilly Media.

Rifai, H., Mohammed, S., & Mellouk, A. (2011). *A brief synthesis of QoS-QoE methodologies.* Paper presented at the 2011 10th International Symposium on Programming and Systems (ISPS), Algeria.

Rittinghouse, J. W., & Ransome, J. F. (2009). *Cloud computing: implementation, management, and security*: CRC press.

Ross, T. J. (2009). *Fuzzy logic with engineering applications*. Hoboken, NJ: John Wiley & Sons.

SaaShost.net. (2016a). SaaShost.net. Retrieved from http://www.saashost.net/saashost-services/

SaaShost.net. (2016b). Service Level Agreement. Retrieved from http://www.saashost.net/service-level-agreement-2/

Safdari, F., & Chang, V. (2014). *Review and analysis of Cloud Computing Quality of Experience*. Paper presented at the First International Workshop on Emerging Software as a Service and Analytics, Barcelona.

Samet, N., Leta, A. B., Hamdi, M., & Tabbane, S. (2016). *Real-Time User Experience Evaluation for Cloud-Based Mobile Video.* Paper presented at the 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Crans-Montana, Switzerland.

Sarna, D. E. (2010). *Implementing and developing cloud computing applications*: Auerbach Publications.

Serhani, M. A., Atif, Y., & Benharref, A. (2014). Towards an adaptive QoS-driven monitoring of cloud SaaS. *International Journal of Grid and Utility Computing, 5*(4), 263-277.

Shao, J., & Wang, Q. (2011). *A performance guarantee approach for cloud applications based on monitoring.* Paper presented at the 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW), Munich, Germany.

Shin, Y.-R., & Huh, E.-N. (2015). *QoE metrics aggregation for hierarchical Service Level Agreement in Cross-Layered SLA architecture.* Paper presented at the 2015 Seventh International Conference on Ubiquitous and Future Networks, Japan.

Shroff, G. (2010). *Enterprise cloud computing: technology, architecture, applications*. Cambridge Cambridge University Press.

Siebenhaar, M., Wenge, O., Hans, R., Tercan, H., & Steinmetz, R. (2013). *Verifying the Availability of Cloud Applications.* Paper presented at the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013), Germany.

Smit, M., Simmons, B., & Litoiu, M. (2013). Distributed, application-level monitoring for heterogeneous clouds using stream processing. *Future Generation Computer Systems, 29*(8), 2103-2114.

Software, & Association, I. I. (2001). Software as a service: Strategic backgrounder. *Washington, DC, 31*.

SurveyMonkey. (2016). SurveyMonkey. Retrieved from https://www.surveymonkey.co.uk/

Tan, C., Liu, K., & Sun, L. (2013). A design of evaluation method for SaaS in cloud computing. *Journal of Industrial Engineering & Management, 6*(1).

Tang, M., Dai, X., Liu, J., & Chen, J. (2016). Towards a trust evaluation middleware for cloud service selection. *Future Generation Computer Systems*.

Tao, Y.-H., Wu, Y.-L., Chang, C.-J., & Chang, C.-W. (2013). *Measuring of QoE for Cloud Applications.* Paper presented at the 3rd International Workshop on Intelligent Data Analysis and Management, Taiwan.

Torkashvan, M., & Haghighi, H. (2012a). *CSLAM: A framework for cloud service level agreement management based on WSLA.* Paper presented at the 2012 Sixth International Symposium on Telecommunications (IST), Tehran, Iran.

Torkashvan, M., & Haghighi, H. (2012b). *A service oriented framework for cloud computing.* Paper presented at the 3rd International Conference on Information and Communication Systems, Irbid, Jordan.

Trihinas, D., Pallis, G., & Dikaiakos, M. D. (2014). *JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud.* Paper presented at the 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, IL, USA

Tsai, W.-T., Sun, X., & Balasooriya, J. (2010). *Service-oriented cloud computing architecture.* Paper presented at the 2010 Seventh International Conference on Information Technology: New Generations (ITNG), USA.

Tsidulko, J. (2016, July 27). The 10 Biggest Cloud Outages Of 2016 (So Far). *CRN*. Retrieved from http://www.crn.com/slide-shows/cloud/300081477/the-10-biggest-cloud-outages-of-2016-so-far.htm?itc=refresh

Turner, M., Budgen, D., & Brereton, P. (2003). Turning software into a service. *Computer., 36*(10), 38-44.

Upadhyaya, B., Zou, Y., Xiao, H., Ng, J., & Lau, A. (2011). *Migration of SOAP-based services to RESTful services.* Paper presented at the 2011 13th IEEE International Symposium on Web Systems Evolution (WSE), Williamsburg, VA, USA.

Vaquero, L. M., Rodero-Merino, L., Caceres, J., & Lindner, M. (2008). A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review, 39*(1), 50-55.

Varela, M., Skorin-Kapov, L., & Ebrahimi, T. (2014). Quality of service versus quality of experience *Quality of Experience* (pp. 85-96): Springer.

Varela, M., Zwickl, P., Reichl, P., Xie, M., & Schulzrinne, H. (2015). *From Service Level Agreements (SLA) to Experience Level Agreements (ELA): The Challenges of Selling QoE to the User.* Paper presented at the 2015 IEEE International Conference on Communication Workshop (ICCW), London, United Kingdom.

Velte, T., Velte, A., & Elsenpeter, R. (2009). *Cloud computing, a practical approach*: McGraw-Hill, Inc.

Vinoski, S. (2007). REST Eye for the SOA Guy. *IEEE Internet Computing, 11*(1), 82.

Voorsluys, W., Broberg, J., & Buyya, R. (2011). Introduction to cloud computing. *Cloud computing: Principles and paradigms*, 1-44.

Wei, Y., & Blake, M. B. (2010). Service-oriented computing and cloud computing: challenges and opportunities. *IEEE Internet Computing, 14*(6), 72.

Wieder, P., Butler, J. M., Theilmann, W., & Yahyapour, R. (2011). *Service level agreements for cloud computing*: Springer.

Wieder, P., Seidel, J., Wäldrich, O., Ziegler, W., & Yahyapour, R. (2008). Using sla for resource management and scheduling-a survey *Grid Middleware and Services* (pp. 335-347): Springer.

Wohlstadter, E., Tai, S., Mikalsen, T., Diament, J., & Rouvellou, I. (2006). *A service-oriented middleware for runtime web services interoperability.* Paper presented at the 2006 IEEE International Conference on Web Services (ICWS'06), Chicago, Illinois.

Wu, C., Zhu, Y., & Pan, S. (2013). *The SLA Evaluation Model for Cloud Computing.* Paper presented at the International Conference on Computer, Networks and Communication Engineering (ICCNCE 2013), Beijing, China.

Wu, L., Garg, S. K., & Buyya, R. (2015). *Service Level Agreement (SLA) based SaaS Cloud Management System.* Paper presented at the 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), Australia.

Xu, D. (2010). *Cloud computing: an emerging technology.* Paper presented at the 2010 International Conference on Computer Design and Applications (ICCDA), China.

Yang, J., Zhang, L., & Wang, X. A. (2015). *On Cloud Computing Middleware Architecture.* Paper presented at the 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Krakow, Poland.

Ye, L., Zhang, H., Shi, J., & Du, X. (2012). *Verifying cloud Service Level Agreement.* Paper presented at the 2012 IEEE Global Communications Conference (GLOBECOM), Anaheim, CA, USA.

You, L., Motta, G., & Sfondrini, N. (2015). *SLM as a Third Party Service in Cloud Environment: A Reference Framework.* Paper presented at the 2015 IEEE International Conference on Services Computing (SCC), New York, USA.

Zadeh, L. A. (2008). Is there a need for fuzzy logic? *Information sciences, 178*(13), 2751-2779.

Zeginis, C., & Plexousakis, D. (2010). *Monitoring the QoS of Web Services using SLAs* Retrieved from https://www.ics.forth.gr/tech-reports/2010/2010.TR404_Monitoring_QoS_Web_Services_using_SLAs.pdf

Zhang, Y., Liu, H., Deng, B., & Peng, F. (2014). *A Reliable QoE-aware Framework for Cloud Service Monitoring and Ranking.* Paper presented at the 2013 International Conference on Electrical and Information Technologies for Rail Transportation (EITRT2013)-Volume II, China.

Zheng, X., Martin, P. ; Brohman, K. ; Xu, L.D. (2013). CLOUDQUAL: A Quality Model for Cloud Services. *IEEE Transactions on Industrial Informatics*(99). doi:10.1109/TII.2014.2306329

Zhu, F., Li, H., & Lu, J. (2012). *A service level agreement framework of cloud computing based on the Cloud Bank model.* Paper presented at the 2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE), Zhangjiajie, China.

Zulkernine, F. H., Martin, P., & Wilson, K. (2008). *A middleware solution to monitoring composite web services-based processes.* Paper presented at the 2008 IEEE Congress on services part II. SERVICES-2. IEEE, Beijing, China.

APPENDIX A

SLA EXAMPLE

This appendix presents an example of an SLA introduced by a SaaS provider. This company provides SaaS services like email, office suites and collaboration programmes like Skype (SaaShost.net, 2016a). The agreement introduces in this appendix explains the services levels and the actions to be taken in case of SLA violations.

A.1 SERVICE LEVEL AGREEMENT

This document outlines the service level agreement for USERs provisioned with Hosted Services with SAASHOST.NET (SaaShost.net, 2016b).

Master Service Level Agreement

This document contains the Service Level Agreement for SAASHOST.NET. Please read it carefully as this is the official agreement in force at the present time. The agreement listed below supersedes any other written document you may have prior to today's date. Exhibits to this agreement are also available highlighting additional terms. If you have questions or comments about this agreement, please do not hesitate to contact us.

SLA Objective

THIS SERVICE LEVEL AGREEMENT ("Agreement" or "SLA") shall apply to all Hosted Services provided by SAASHOST.NET expressly as an addendum to the Terms Of Service ("TOS") for each customer/client/consumer/domain/administrator/end user/user ("USER"). SAASHOST.NET is committed to providing a highly available and secure network to support its USERs. Providing the USER with consistent access to Hosted Services is a high priority for SAASHOST.NET and is the basis for its commitment in the form of a SLA. The SLA provides certain rights and remedies in the event that the USER experiences service interruption as a result of failure of SAASHOST.NET infrastructure. The overall service availability metric is 99.999%, measured on a monthly basis.

Term Definitions

For the purpose of this Service Level Agreement, the terms in bold are defined as follows:

Available or Availability

When the USER who's account is active and enabled has reasonable access to the Hosted Service provided by SAASHOST.NET, subject to the exclusions defined in Downtime Minutes below.

Total Monthly Minutes

The number of days in the month multiplied by 1,440 minutes per day.

Maintenance Time

The time period during which the Hosted Service may not be Available each month so that SAASHOST.NET can perform routine maintenance to maximize performance, is on an as needed basis.

Downtime

The total number of minutes that the USER cannot access the Hosted Service. The calculation of Downtime Minutes excludes time that the USER is unable to access the Hosted Services due to any of the following:

(a) Maintenance Time

(b) USER's own Internet service provider

(c) Force Majeure event

(d) Any systemic Internet failures

(e) Enhanced Services

(f) Any failure in the USER's own hardware, software or Network connection

(g) USER's bandwidth restrictions

(h) USER's acts or omissions

(i) Anything outside of the direct control of SAASHOST.NET

SAASHOST.NET Network

The network inside of SAASHOST.NET border routers.

Problem Response Time

The time period after SAASHOST.NET's confirmation of the Service event, from receipt of the information required from the USER for SAASHOST.NET's Support Team to begin resolution and open a trouble ticket in SAASHOST.NET's systems. Due to the wide diversity of problems that can occur, and the methods required to resolve them, problem response time IS NOT defined as the time between the receipt of a call and problem resolution. After receiving a report of fault, SAASHOST.NET shall use a reasonable method to provide USER with a progress update.

Affected Seats

SAASHOST.NET's Hosted Service are provided in a multi tenant architecture where seats of a USER's domain may be extended across numerous servers. USER may obtain remedy only for affected seats residing on the server experiencing Downtime exceeding the SLA.

Maintenance Notices

SAASHOST.NET will communicate the date and time that SAASHOST.NET intends to make the Hosted Services un-Available via the front page of the support web site at least forty-eight (48) hours in advance (or longer if practical). The USER understands and agrees that there may be instances where SAASHOST.NET needs to interrupt the Hosted Services without notice in order to protect the integrity of the Hosted Services due to security issues, virus attacks, spam issues or other unforeseen circumstances. Below are the Maintenance Windows and their definitions:

Emergency Maintenance

These change controls happen immediately with little notification ahead of time; however,
we will post the information to our website soon after or during the change.

Preventative Maintenance

These change controls are when we detect an item in the environment that we need to take action on, to avoid emergency change controls in the future. These change controls, if possible, will usually occur in low peak hours with peak being defined by our network metrics.

Planned Maintenance

These are change control's being done to:

- Support on-going product and operational projects to ensure optimal performance
- Deploy non-critical service packs or patches.
- Periodic redundancy testing.

Where possible planned maintenance will be posted 5-days prior; however, certain circumstances may preclude us from doing so, such as an external vendor issuing a change control to SAASHOST.NET, e.g. the power company alerting us to perform power testing 48 hours ahead of time.

USER Responsibility

Minimum Requirements

The required configurations USER must have to access the Hosted Services include:

- Internet connection with adequate bandwidth
- Internet Browser

Control Panel

The Control Panel is provided to all USERs enabled with Hosted Services at SAASHOST.NET, therefore the USER can manage their own account and services. The USER should use discretion when granting administrative privileges to the Control Panel. For liability purposes The Support Team is not permitted to access nor perform tasks via the USER Control Panel. Mailboxes, services enabled, and storage quota facilitated in the Control Panel are billable and SAASHOST.NET is unable to provide credits due to negligence in the Control Panel. SAASHOST.NET is not responsible for downtime related to negligence in the Control Panel. An example of negligence is service unavailability caused by reaching quota limits set in the Control Panel. Another negligence example is Hosted Services disabled/deleted in error. Please note that in the case of negligence SAASHOST.NET may/may not have the ability to restore data as data restoration is reserved for disaster recovery purposes. If data is lost due to negligence and it is determined that the data or a fraction of the data can be restored, professional service fees may be applied as stated in the SAASHOST.NET Backup and Restoration Policy.

Service Levels - Term of the Service Level Agreement

This Service Level Agreement shall only become applicable to the Hosted Services upon the later of (a) completion of the "stabilization period," as such term is defined in the Statement of Work (if any), or (b) ninety (90) days from the provisioning of Hosted Services.

Measurement

SAASHOST.NET uses a proprietary system to measure whether the Hosted Services are Available and the USER agree that this system will be the sole basis for resolution of any dispute that may arise between the USER and SAASHOST.NET regarding this Service Level Agreement.

Availability is calculated based on the following formula:

$A = (T - M - D) / (T - M) \times 100\%$

A = Availability

T = Total Monthly Minutes

M = Maintenance Time

D = Downtime

Availability	**Credit Amount of Monthly Fee for Affected Seats**
> 97.9% but < 99.999%	5%
> 96.9% but < 97.9%	7%
< 96.9%	9%

Problem Response Time

SAASHOST.NET's failure to meet the Service level metric for Problem Response Time for a month shall result in a Service Level Credit calculated per incident at a credit of 50% of the monthly invoice, up to a maximum Service Level Credit of $200, for the Hosted Service (not including setup, activation fees or other services provided by SAASHOST.NET) per month. The response time per incident will vary upon the degrees defined below:

Category Level	**Criteria**	**Problem Response Time**
1	Unplanned interruption rendering the Services un-Available; no work-around	5 Minutes
2	Unplanned interruption rendering the Services un-Available; work-around available	15 Minutes
3	Services are un-Available for a single User or small percentage of USER affected	4 Hours
4	Intermittent problem	8 Business Hours

Remedy and Procedure

The USER's remedy and the procedure for obtaining the USER's remedy in the event that SAASHOST.NET fails to meet the Service level metrics set forth above are as follows:

To qualify for remedy:

(a) There must be a support ticket documenting the event within 24 hours of the service interruption

(b) USER account must be in good standing with all invoices paid and up to date

The USER must notify SAASHOST.NET in writing within five (5) business days by opening a support ticket and providing the following details:

- Subject of email must be: “Claim Notice – ‘USERDomain’.com” (USER’s primary domain hosted with SAASHOST.NET must be listed in place of ‘USERDomain.com’)
- List the type of Hosted Service that was affected
- List the date the Downtime Minutes occurred
- List user(s) Display Name and E-mail address affected by Downtime Minutes
- List an estimate of the amount of actual Downtime Minutes
- Ticket number of the documented event

SAASHOST.NET will confirm the information provided in the Claim Notice within five (5) business days of receipt of the Claim Notice. If SAASHOST.NET cannot confirm the Downtime Minutes, then the USER and SAASHOST.NET agree to refer the matter to executives at each company for resolution. If SAASHOST.NET confirms that SAASHOST.NET is out of compliance with this Service Level Agreement, the USER will receive the amount of Service Level Credits set forth above for the affected Service level metric and the affected Seats for the affected month. The SLA credit will be reflected in the SAASHOST.NET invoice to the USER in the month following SAASHOST.NET confirmation of the Downtime Minutes. Please note that SLA credits can only be applied to accounts that are in good standing with all invoices paid and up to date.

SLA Exhibits

Exhibits to this Master Service Level Agreement may be available for Hosted Services provided by SAASHOST.NET. The SLA Exhibits for each Hosted Service provides additional terms specific to the Hosted Service. The SLA Exhibits must be agreed to in addition to this Master Service Level Agreement prior to executing use of the Hosted Service.

Service Level Agreement – Exchange Exhibit

This document contains the Service Level Agreement for SAASHOST.NET. Please read it carefully as this is the official agreement in force at the present time. The agreement listed below supersedes any other written document you may have prior to today’s date. If you have questions or comments about this agreement, please do not hesitate to contact us.

SLA Objective

THIS SERVICE LEVEL AGREEMENT (“Agreement” or “SLA”) shall apply to Hosted Microsoft Exchange services provided by SAASHOST.NET expressly as an exhibit to the Master Service Level Agreement (“MSLA”) for each customer/ client/ consumer/ domain/ administrator/ end user/ user (“USER”).

SAASHOST.NET is committed to providing a highly available and secure network to support its USERs. Providing the USER with consistent email access is a high priority for SAASHOST.NET and is the basis for its commitment in the form of a SLA. The SLA provides certain rights and remedies in the event that the USER experiences service interruption as a result of failure of SAASHOST.NET infrastructure. The overall service availability metric is 99.999%, measured on a monthly basis.

Term Definitions

For the purpose of this Service Level Agreement, the terms in bold are defined as follows:

Downtime

The total number of minutes that the USER cannot access the mailbox on the Microsoft Exchange Server. The calculation of Downtime Minutes excludes time that the USER is unable to access the mailbox on the Microsoft Exchange Server due to any of the following:

(a) Maintenance Time;

(b) USER’s own Internet service provider

(c) Force Majeure event

(d) Any systemic Internet failures

(e) Enhanced Services

(f) Any failure in the USER’s own hardware, software or Network connection

(g) USER’s bandwidth restrictions

(h) USER’s acts or omissions; e.g. mailbox inaccessible due to suspension or quota overage

(i) Anything outside of the direct control of SAASHOST.NET; e.g. outage/latency due to Spam Filtering Service outage.

Mail Delivery Time

The time between an email sent from the USER's email interface (containing valid internet connection, header, and address information at our server) to a valid email address inside or outside of the USER domain. SAASHOST.NET is not responsible for undelivered mail that has departed SAASHOST.NET's network, however routed improperly due to recipient policies or configurations.

USER Responsibility

Minimum Requirements

The required configurations USER must have to access the Microsoft Exchange Server include:

- Internet connection with adequate bandwidth
- Internet Browser
- Windows XP SP2
- Outlook 2003 SP2
- DNS settings provided by SAASHOST.NET must be configured in USER's DNS Zone

SAASHOST.NET recommends utilizing the latest Windows operating system, not in beta; and the latest Outlook version, not in beta. Full Access mailboxes are recommended to make use of the complete functionality of Microsoft Exchange and is fully supported by the Support Team. Copies of Outlook and Entourage are made available for Full Access mailboxes by SAASHOST.NET. Comparable operating systems and mail clients to access Email via Full Access/POP3/IMAP/SMTP can be utilized, but may not be supported. Once mail has been extracted from the Microsoft Exchange server via POP3, archiving or any other method, SAASHOST.NET no longer has visibility and may only provide limited support regarding the data.

Mobile Devices

SAASHOST.NET provides USER with access to the Microsoft Exchange server via Windows Mobile (ActiveSync) or through add-on services by use of third party software/servers. Accessing the Microsoft Exchange server via such devices are reliant upon the device hardware, device operating system, and wireless carrier. SAASHOST.NET will make commercially reasonable efforts to ensure Availability and support in configuration, but cannot guarantee accessibility due to the many factors out of SAASHOST.NET's control.

Service Levels - Term of the Service Level Agreement

This Service Level Agreement shall only become applicable to the Services upon the later of (a) completion of the "stabilization period," as such term is defined in the Statement of Work (if any), or (b) ninety (90) days from the MX records change over date.

Mail Delivery Time

The Service level metric for Mail Delivery Time is within 5 minutes or less, 95% of the time measured on a monthly basis, subject to the exclusions defined in Downtime Minutes above. The remaining 5% will be processed, but may take longer than 5 minutes. The delivery time calculation does not include complications from outside forces including but not limited to ISP delays or failures, USER Internet connectivity issues, datacenter collocation failures, blacklisting, spam filtering, systemic Internet failures, DDOS attacks, recipient policies, recipient network, and other foreseen interruptions.

Mail Delivery Time	**Credit Amount of Monthly Fee for Affected Seats**
> 93% but < 95%	3%
> 91% but < 93%	5%
< 91%	7%

Service Level Agreement – SharePoint Exhibit

This document contains the Service Level Agreement for SAASHOST.NET. Please read it carefully as this is the official agreement in force at the present time. The agreement listed below supersedes any other written document you may have prior to today's date. If you have questions or comments about this agreement, please do not hesitate to contact us.

SLA Objective

THIS SERVICE LEVEL AGREEMENT("Agreement" or "SLA") shall apply to Hosted Microsoft SharePoint services provided by SAASHOST.NET expressly as an exhibit to the Master Service Level Agreement ("MSLA") for each customer/client/consumer/domain/administrator/end user/user ("USER"). SAASHOST.NET is committed to providing a highly available and secure network to support its USERs. Providing the USER with consistent connectivity to the SharePoint service is a high priority for SAASHOST.NET and is the basis for its commitment in the form of a SLA. The SLA provides certain rights and remedies in the event that the USER experiences service interruption as a result of failure of SAASHOST.NET infrastructure. The overall service availability metric is 99.999%, measured on a monthly basis.

Term Definitions

Available or Availability

When the CUSTOMER who's account is active and enabled has reasonable connectivity to the Microsoft SharePoint Site provided by SAASHOST.NET, subject to the exclusions defined in Downtime Minutes below. Availability is in regard to connectivity with standard SharePoint functionality to the provisioned SharePoint site and its modification tools. Availability does not refer to customization, installation of templates, mapped drives, importing data from previous SharePoint sites or backups, or use of 3rd party applications.

Downtime

The total number of minutes that the USER cannot access the SharePoint site provisioned on the SASS PROVIDER's network. The calculation of Downtime Minutes excludes time that the USER is unable to access or modify the SharePoint site due to any of the following:

(a) Maintenance Time;

(b) USER's own Internet service provider

(c) Force Majeure event

(d) Any systemic Internet failures

(e) Enhanced Services

(f) Any failure in the USER's own hardware, software or Network connection

(g) USER's bandwidth restrictions

(h) USER's acts or omissions; e.g. disabling SharePoint in the Control Panel

(i) Anything outside of the direct control of SAASHOST.NET; e.g. site inaccessibility due to Browser or DNS caching

(j) Incorrect DNS Settings

USER Responsibility

Minimum Requirements

The required configurations USER must have to access the Microsoft SharePoint Server include:

- Internet connection with adequate bandwidth
- Internet Browser
- Windows XP SP2
- DNS settings provided by SAASHOST.NET must be configured in USER's DNS Zone

Service Levels - Term of the Service Level Agreement

This Service Level Agreement shall only become applicable to the Services upon the later of (a) completion of the "stabilization period," as such term is defined in the Statement of Work (if any), or (b) ninety (90) days from the CNAME configuration date.

Service Level Agreement – CRM Exhibit

This document contains the Service Level Agreement for SAASHOST.NET. Please read it carefully as this is the official agreement in force at the present time. The agreement listed below supersedes any other written document you may have prior to today's date. If you have questions or comments about this agreement, please do not hesitate to contact us.

SLA Objective

THIS SERVICE LEVEL AGREEMENT ("Agreement" or "SLA") shall apply to Hosted Microsoft Dynamics CRM services provided by SAASHOST.NET expressly as an exhibit to the Master Service Level Agreement ("MSLA") for each customer/client/consumer/domain/administrator/end user/user ("USER").

SAASHOST.NET is committed to providing a highly available and secure network to support its USERs. Providing the USER with consistent connectivity to the Dynamics CRM service is a high priority for SAASHOST.NET and is the basis for its commitment in the form of a SLA. The SLA provides certain rights and remedies in the event that the USER experiences service interruption as a result of failure of SAASHOST.NET infrastructure. The overall service availability metric is 99.999%, measured on a monthly basis.

Term Definitions

For the purpose of this Service Level Agreement, the terms in bold are defined as follows:

Available or Availability

When the USER who's account is active and enabled has reasonable connectivity to the Microsoft Dynamics CRM Site provided by SAASHOST.NET, subject to the exclusions defined in Downtime Minutes below. Availability is in regard to connectivity with standard Dynamics CRM functionality to the provisioned Dynamics CRM site and its modification tools. Availability does not refer to customization, installation of templates, plug-ins, importing data from previous Dynamics CRM sites or backups, or use of/mapping to 3rd party applications.

Downtime

The total number of minutes that the USER cannot access the Dynamics CRM site provisioned on the SASS PROVIDER's network. The calculation of Downtime Minutes excludes time that the USER is unable to access or modify the Dynamics CRM site due to any of the following:

(a) Maintenance Time

(b) USER's and/or User's own Internet service provider

(c) Force Majeure event

(d) Any systemic Internet failures

(e) Enhanced Services

(f) Any failure in the USER's and/or User's own hardware, software or Network connection

(g) USER's and/or Users bandwidth restrictions

(h) USER's and/or User's, acts or omissions; e.g. disabling Dynamics CRM in the Control Panel

(i) Anything outside of the direct control of SAASHOST.NET; e.g. site inaccessibility due to Browser caching

USER Responsibility Minimum Requirements

The required configurations USER must have to access the Microsoft Dynamics CRM Server include:

- Internet connection with adequate bandwidth
- Internet Browser
- Windows XP SP2

Control Panel

The Control Panel is provided to all USERs/domains enabled with services at SAASHOST.NET, therefore the USER can manage their own account and services. The USER should use discretion when granting administrative privileges to the Control Panel. For liability purposes The Support Team is not permitted to access nor perform tasks via the USER Control Panel.

Service Levels - Term of the Service Level Agreement

This Service Level Agreement shall only become applicable to the Services upon the later of (a) completion of the "stabilization period," as such term is defined in the Statement of Work (if any), or (b) ninety (90) days from the Dynamics CRM site is provisioned.

Measurement

SAASHOST.NET uses a proprietary system to measure whether the Services are Available by sending “pings” to servers in the data center at regular intervals and by monitoring the running services on the system. The USER agrees that this system will be the sole basis for resolution of any dispute that may arise between the USER and SAASHOST.NET regarding this Service Level Agreement.

APPENDIX B

DEFUZZIFICATION METHODS RESULTS

B.1 Testing Defuzzification Methods

This appendix presents a study to investigate the effect of different defuzzification methods on QoE level resulted from the proposed fuzzy engine. The compared methods are: centroid, bisector, MOM (Mean of Maximum), LOM (Largest of Maximum), and SOM (Smallest of Maximum). Figure B- 1 depicts these methods on an aggregated membership function in Fuzzy logic.

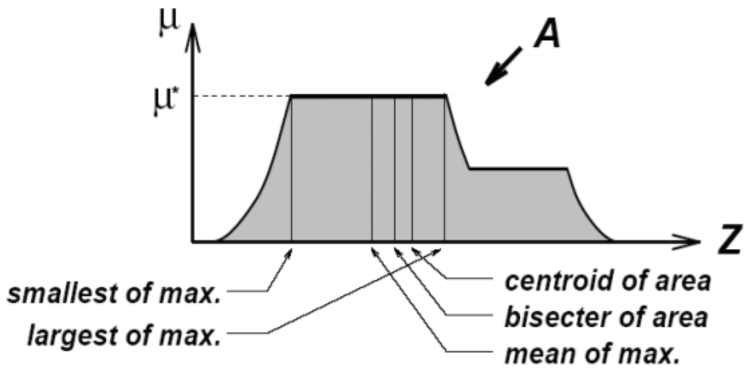


Figure B- 1 effect of defuzzification methods on an aggregated fuzzy membership function (Naaz et al, 2011)

Table B- 1 introduces a numerical comparison of five different defuzzification methods by comparing the value of QoE value in each case. The table reveals that the results obtained by centroid, bisector, and MoM are comparable. The results obtained from SOM method were lower than the three aforementioned methods. On the other hand, the worst results were acquired in LOM method which caused the OoE value to drop to zero. In general, centroid method revealed the best results in terms of QoE level and continuity criteria in defuzzification method.

The system behaviour of the fuzzy engine is investigated by using surface diagrams to study the effect of each two parameters on the system output.

Table B- 1 Studying the effect of different defuzzification methods on QoE

Input parameters						**Fuzzy output**				
Features	**Responsiveness**	**Flexibility**	**Security**	**Rapport**	**Reliability**	**Fuzzy Results (Centroid)**	**Fuzzy Results (Bisector)**	**Fuzzy Results (SOM)**	**Fuzzy Results (LOM)**	**Fuzzy Results (MOM)**
70	52	11	88	100	48	2.525	2.55	2.15	0	2.525
46	40	71	68	76	98	1.535	1.55	1.10	0	1.525
93	95	90	80	90	95	4.464	4.45	4	0	4.5
80	90	84	100	24	60	3.525	3.5	3	0	3.525
89	50	92	22	90	76	2.525	2.5	2.12	0	2.525
26	86	58	90	27	80	2.929	2.85	2	0	2.525
39	57	39	46	28	30	0.536	0.55	0	0	0.475
99	90	89	100	99	90	4.52	4.5	4.2	0	4.6
10	8	30	20	45	12	0.504	0.5	0	0	0.4
75	74	73	72	77	75	2.525	2.5	2.1	0	2.525
93	30	90	95	99	88	3.525	3.55	3.2	0	3.525

40	98	89	90	94	92	4.52	4.500	4.2	0	4.6
90	99	92	22	91	99	3.525	3.500	3.2	0	3.525
95	90	100	89	34	96	4.52	4.500	4.2	0	4.6
88	100	15	99	95	90	4.514	4.500	4.2	0	4.6
97	95	88	100	90	8	4.52	4.500	4.2	0	4.6
73	73	75	70	94	20	2.525	2.500	2.15	0	2.525
22	98	95	90	74	89	4.508	4.500	4.15	0	4.575
18	50	73	95	100	91	2.525	2.55	2.2	0	2.525
89	92	93	21	75	74	3.525	3.5	3.15	0	3.525

B.2 Effect of Bisector Method

Bisector is the vertical line that will divide the region into two sub-regions of equal area. It is sometimes, but not always coincident with the centroid line (MathWorks, 2017). Figure D-2 to Figure D-60 show surface diagrams of the system behaviour for each of the two different parameters. It can be seen that the results obtained in the bisector method is comparable to the centroid method (see section 5.4.3). however, the results obtained in the centroid method were better.

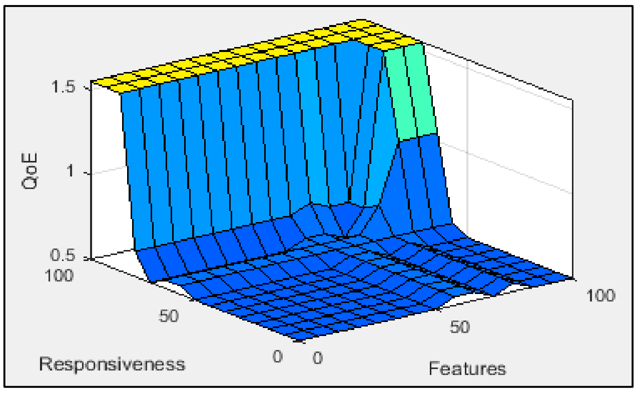


Figure B- 2 Effect of Responsiveness and Features (bisector method)

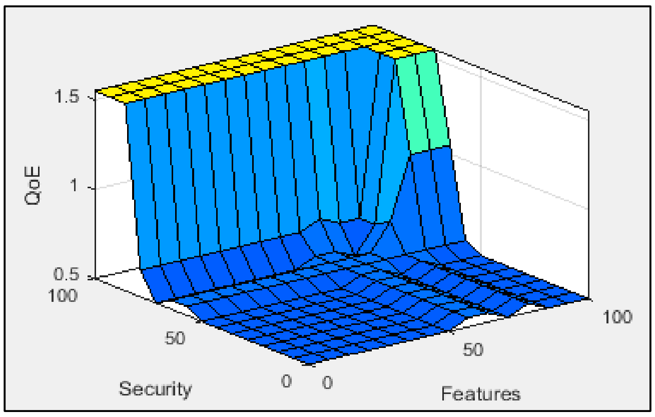


Figure B- 3 Effect of Security and Features (bisector method)

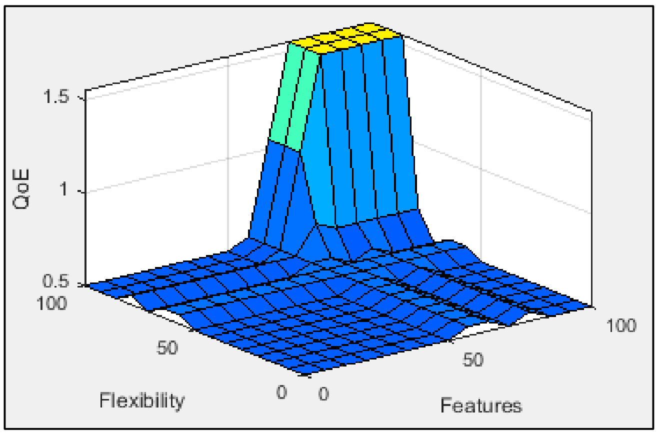


Figure B- 4 Effect of Flexibility and Features (bisector method)

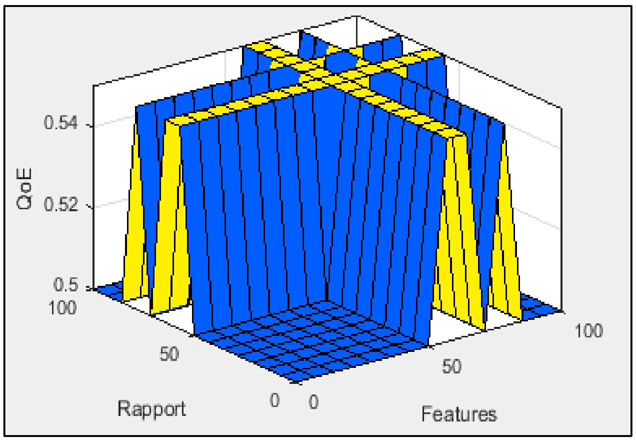


Figure B- 5 Effect of Rapport and Features (bisector method)

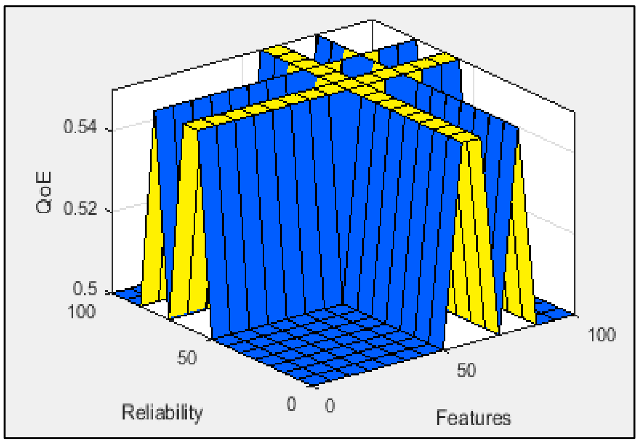


Figure B- 6 Effect of Reliability and Features (bisector method)

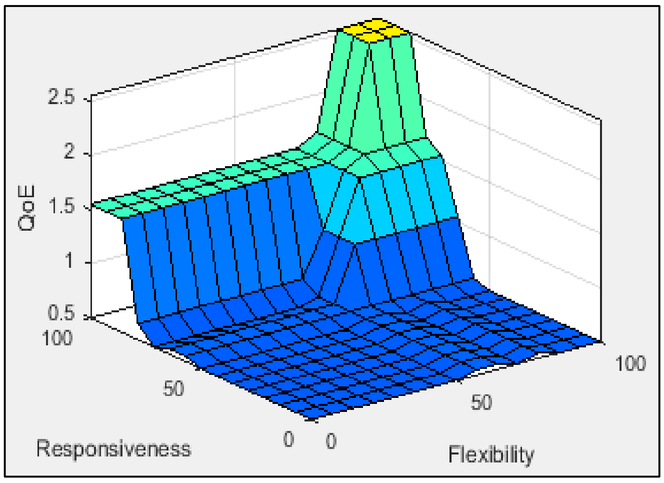


Figure B- 7 Effect of Responsiveness and Flexibility (bisector method)

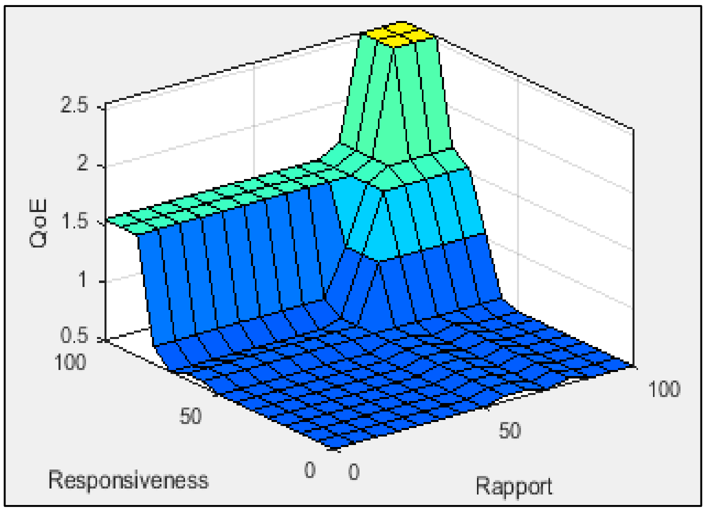


Figure B- 8 Effect of Responsiveness and Rapport (bisector method)

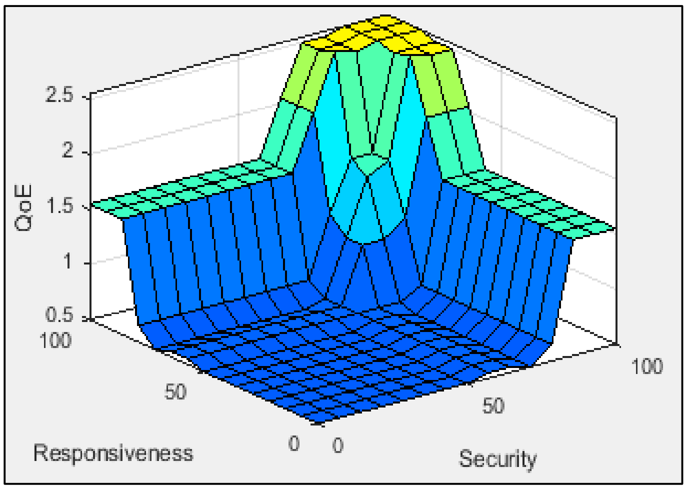


Figure B- 9 Effect of Responsiveness and Security (bisector method)

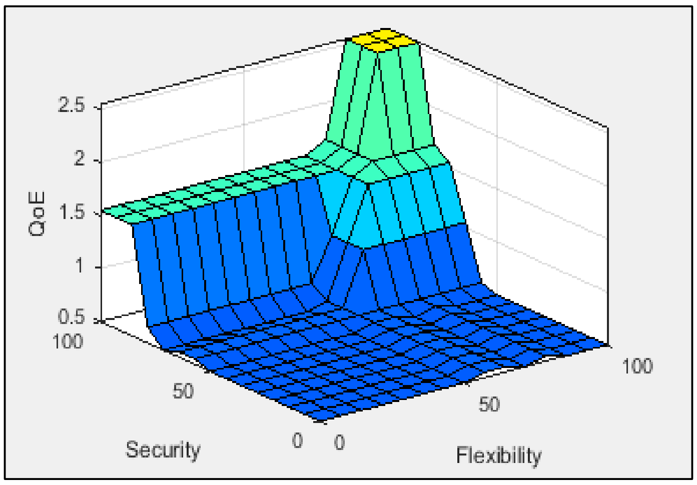


Figure B- 10 Effect of Security and Flexibility (bisector method)

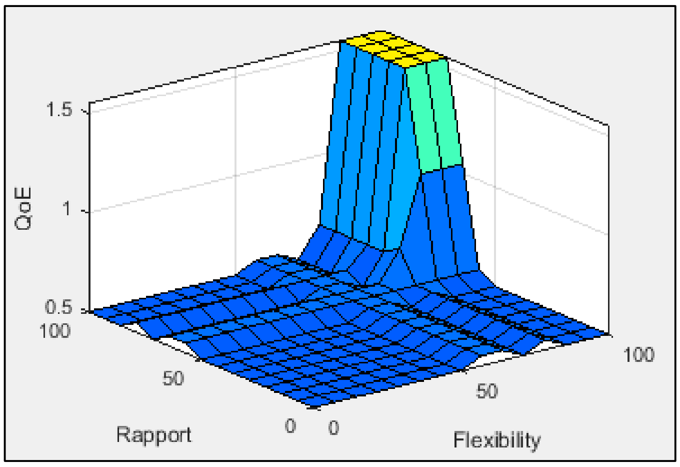


Figure B- 11 Effect of Rapport and Flexibility (bisector method)

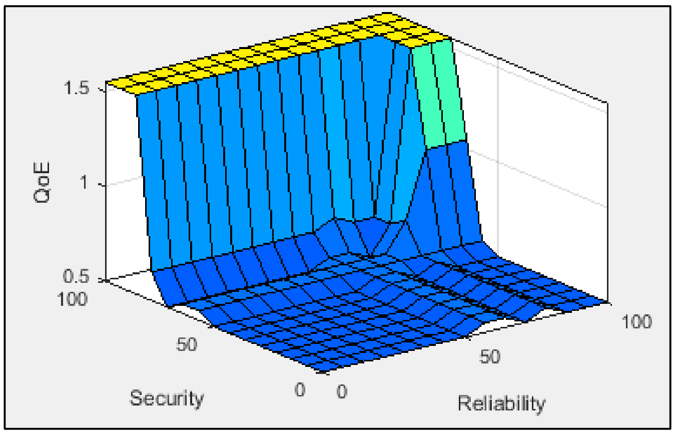


Figure B- 12 Effect of Security and Reliability (bisector method)

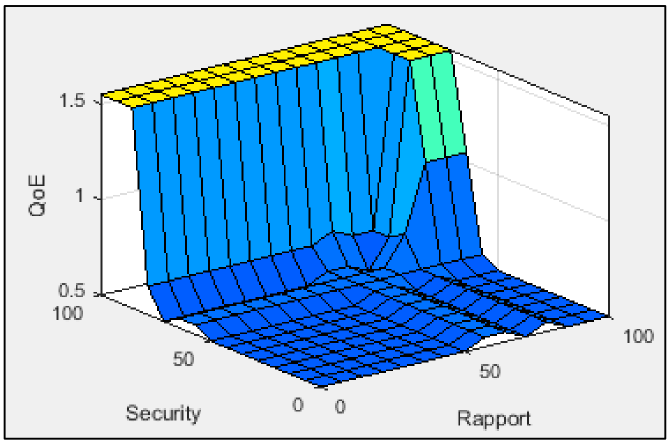


Figure B- 13 Effect of Security and Rapport (bisector method)

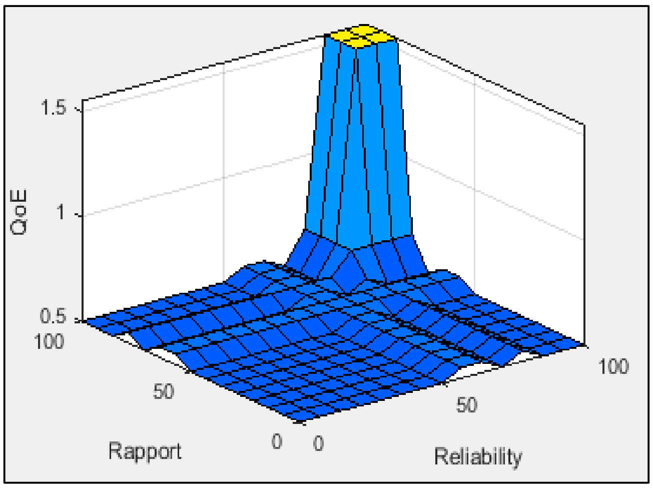


Figure B- 14 Effect of Rapport and Reliability (bisector method)

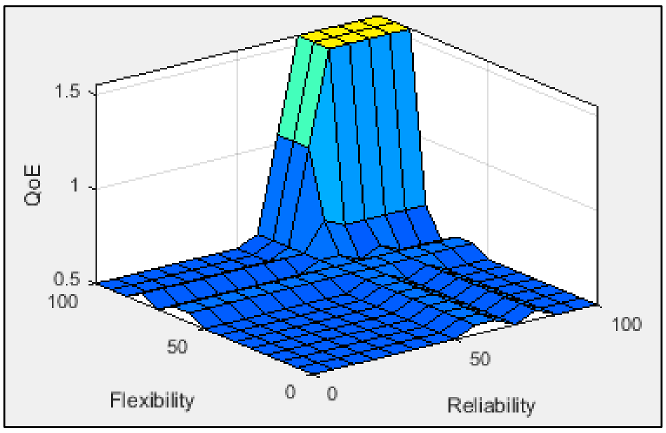


Figure B- 15 Effect of Flexibility and Reliability (bisector method)

B.3 Effect of MOM Method

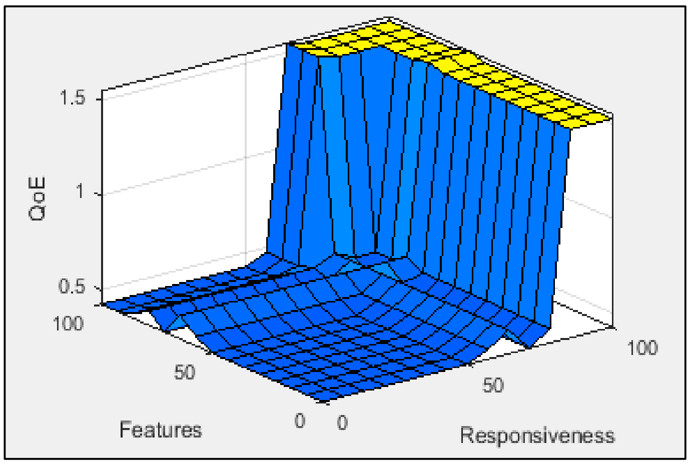


Figure B- 16 Effect of Features and Responsiveness (MOM method)

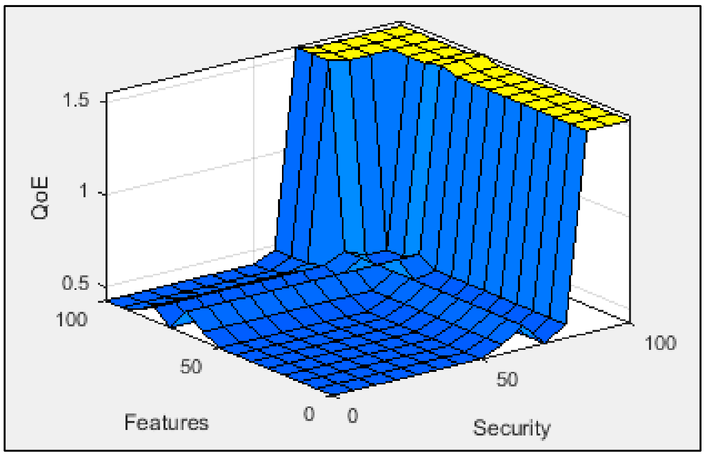


Figure B- 17 Effect of Features and Security (MOM method)

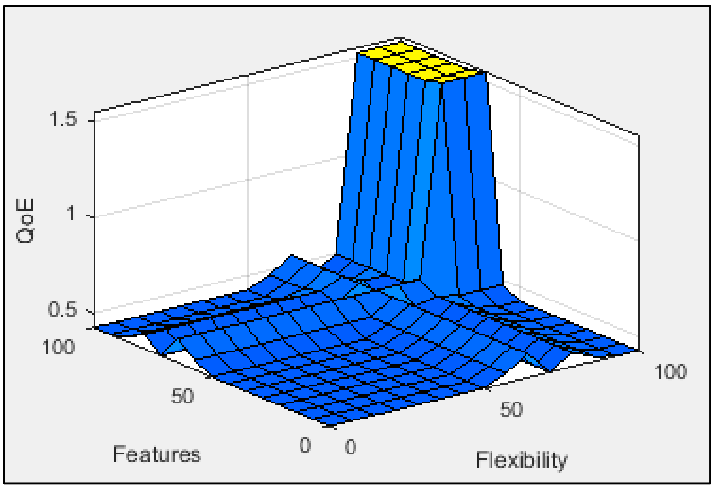


Figure B- 18 Effect of Features and Flexibility (MOM method)

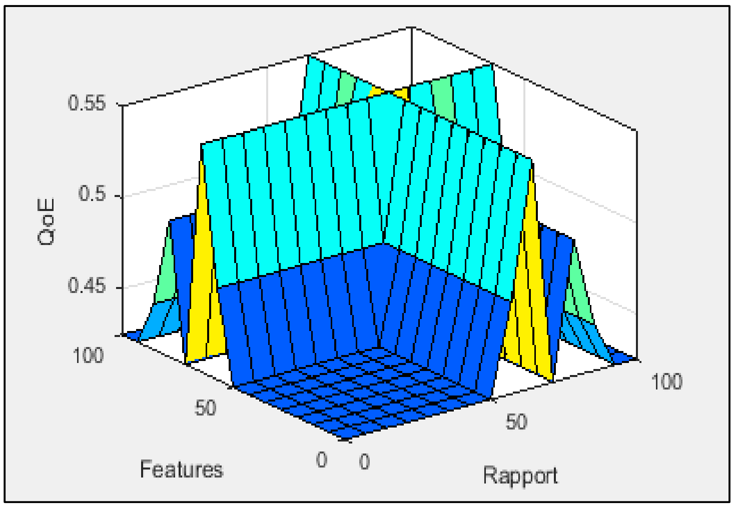


Figure B- 19 Effect of Features and Rapport (MOM method)

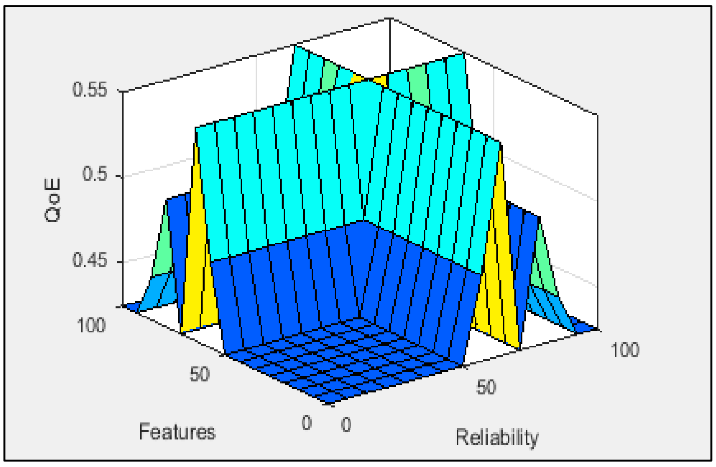


Figure B- 20 Effect of Features and Reliability (MOM method)

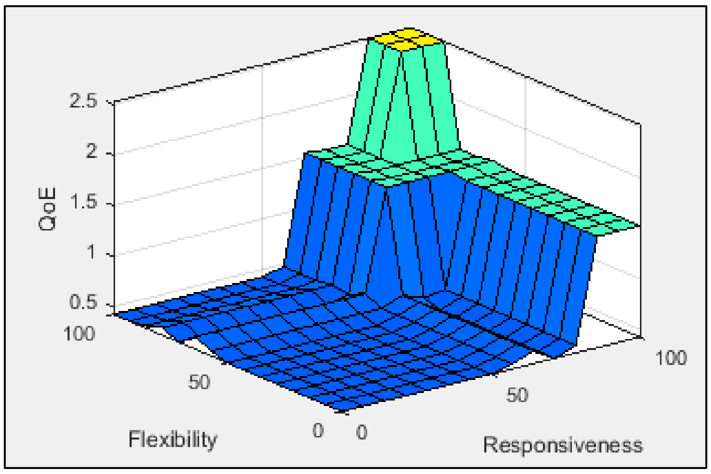


Figure B- 21 Effect of Flexibility and Responsiveness (MOM method)

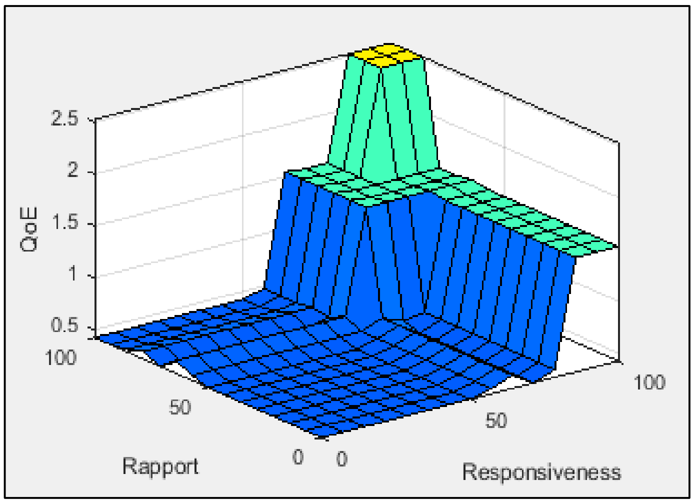


Figure B- 22 Effect of Rapport and Responsiveness (MOM method)

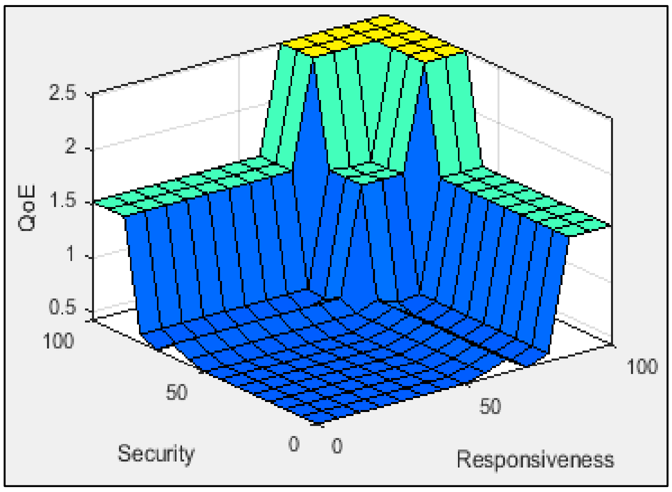


Figure B- 23 Effect of Security and Responsiveness (MOM method)

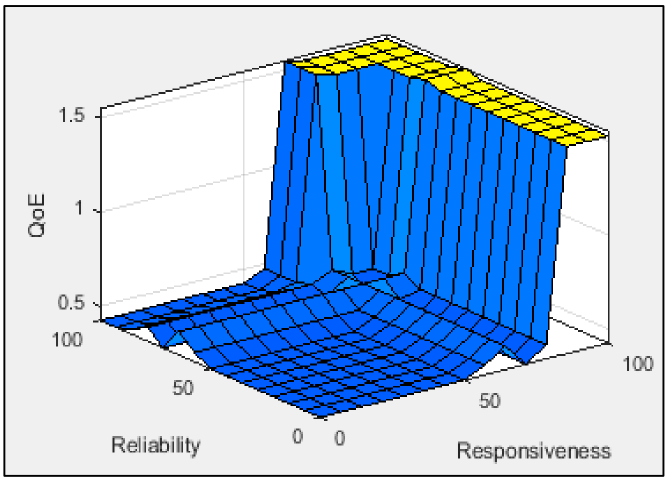


Figure B- 24 Effect of Reliability and Responsiveness (MOM method)

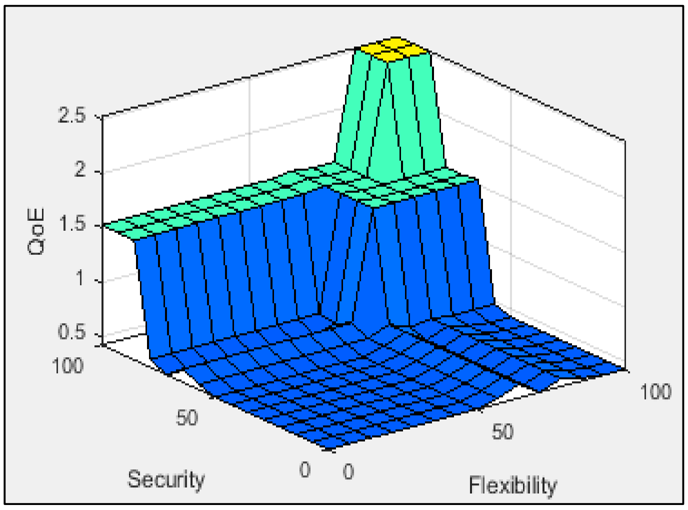


Figure B- 25 Effect of Security and Flexibility (MOM method)

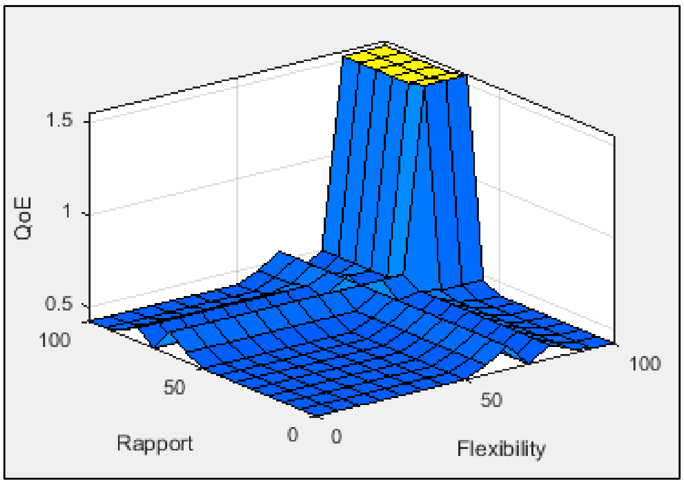


Figure B- 26 Effect of Rapport and Flexibility (MOM method)

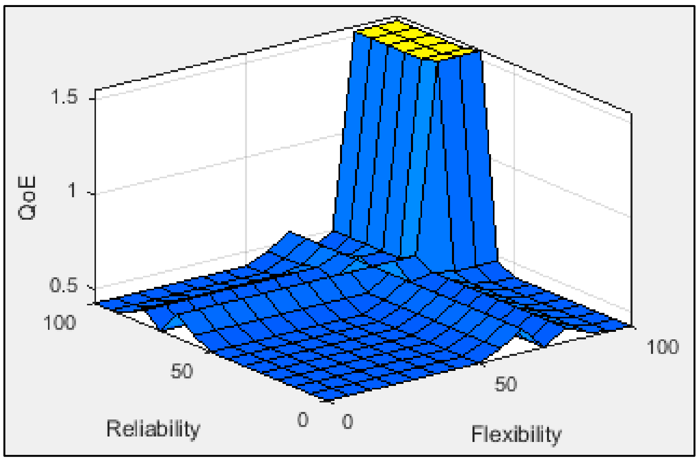


Figure B- 27 Effect of Reliability and Flexibility (MOM method)

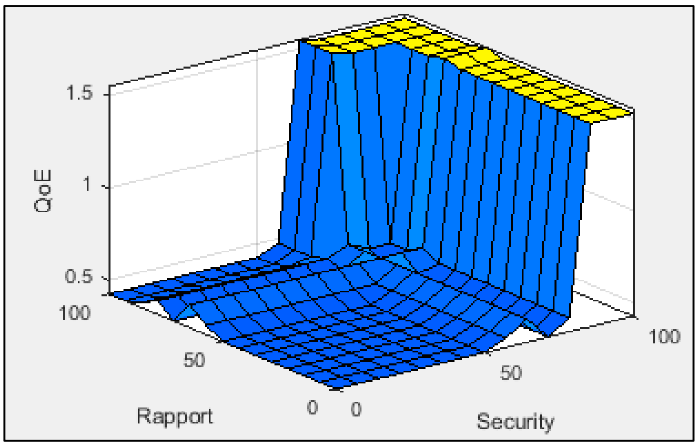


Figure B- 28 Effect of Rapport and Security (MOM method)

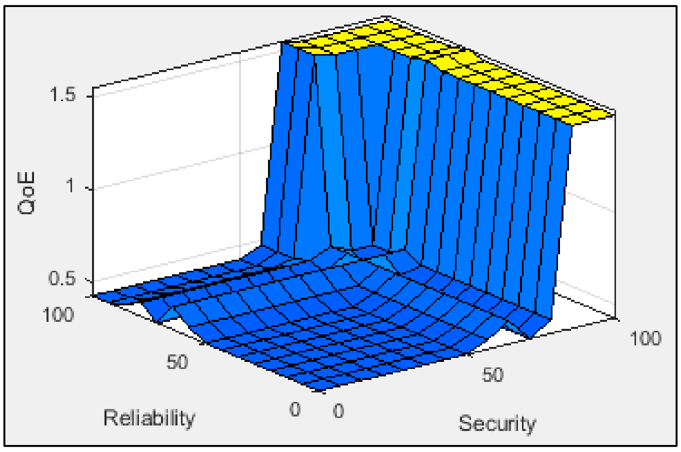


Figure B- 29 Effect of Reliability and Security (MOM method)

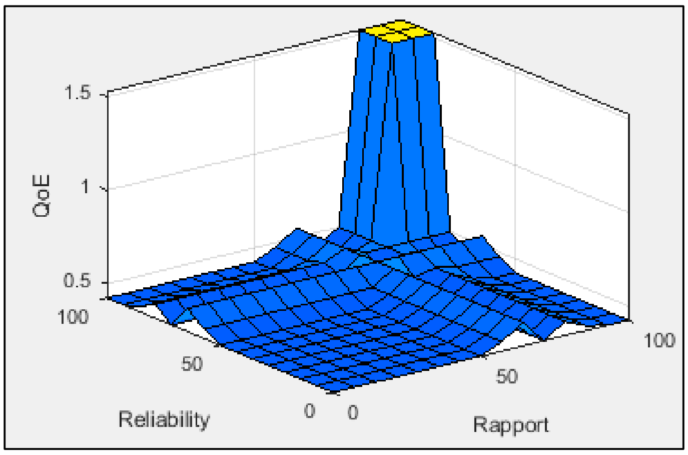


Figure B- 30 Effect of Reliability and Rapport (MOM method)

B4. Effect of LOM Method

The LOM method caused the QoE value to drop to the zero level.

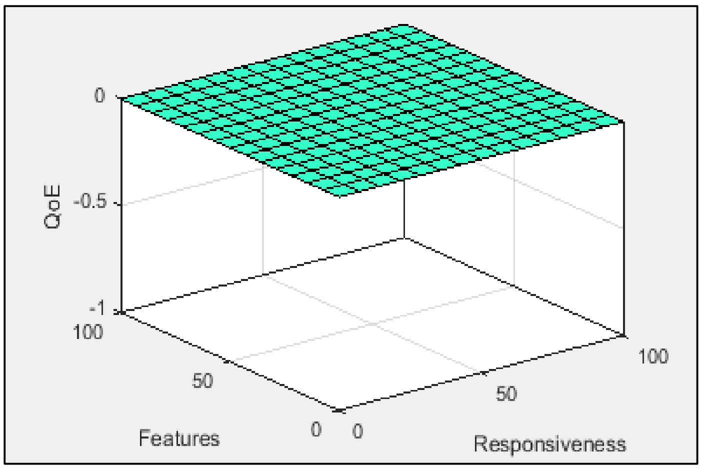


Figure B- 31 Effect of Features and Responsiveness (LOM method)

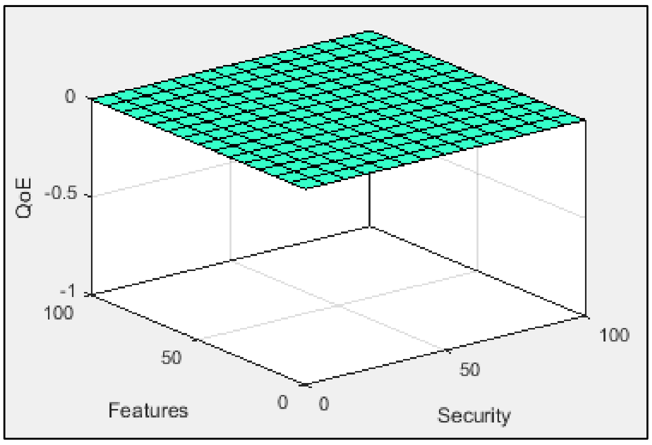


Figure B- 32 Effect of Features and Security (LOM method)

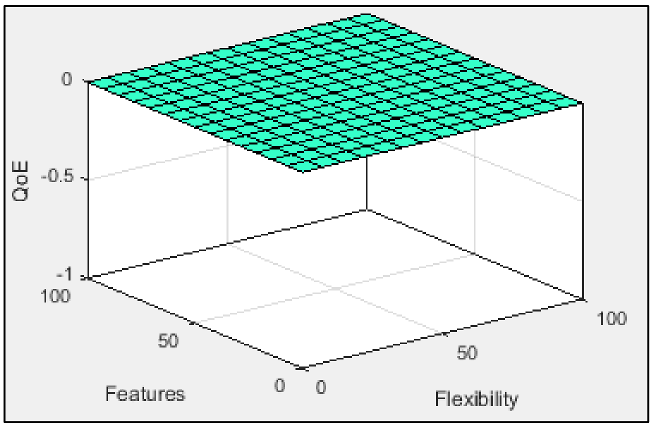


Figure B- 33 Effect of Features and Flexibility (LOM method)

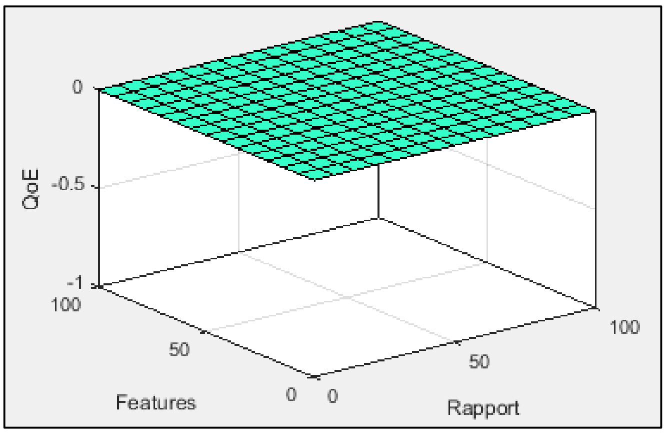


Figure B- 34 Effect of Features and Rapport (LOM method)

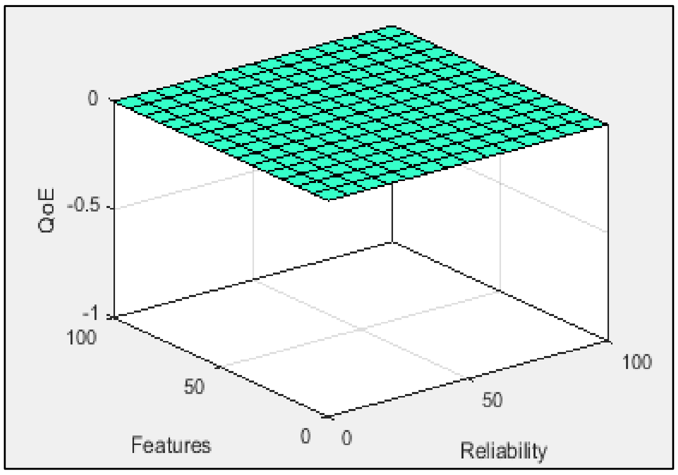


Figure B- 35 Effect of Features and Reliability (LOM method)

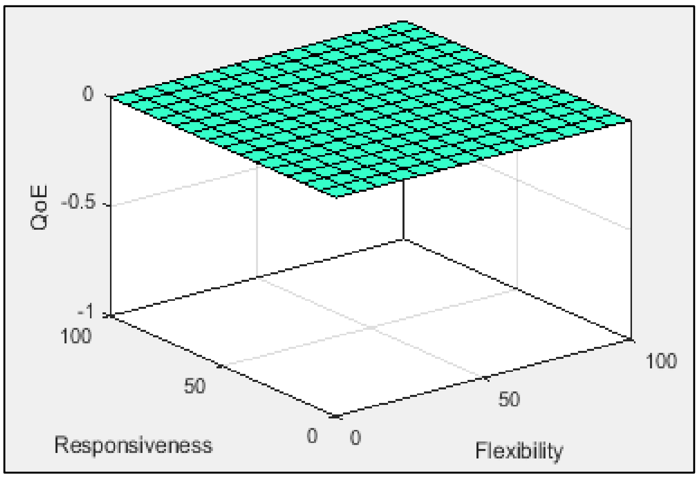


Figure B- 36 Effect of Responsiveness and Flexibility (LOM method)

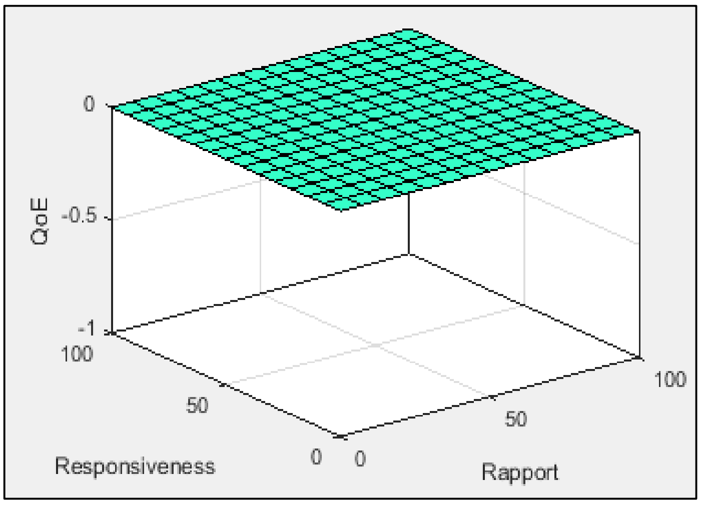


Figure B- 37 Effect of Responsiveness and Rapport (LOM method)

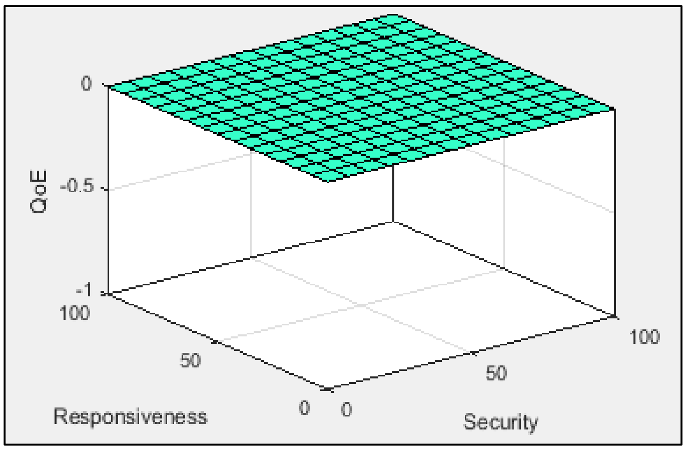


Figure B- 38 Effect of Responsiveness and Security (LOM method)

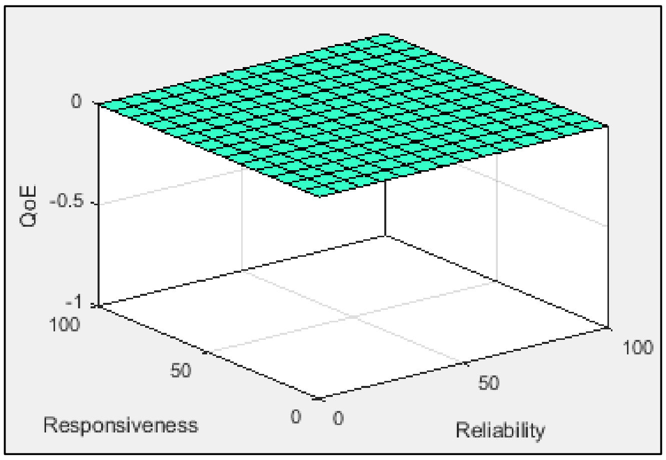


Figure B- 39 Effect of Responsiveness and Reliability (LOM method)

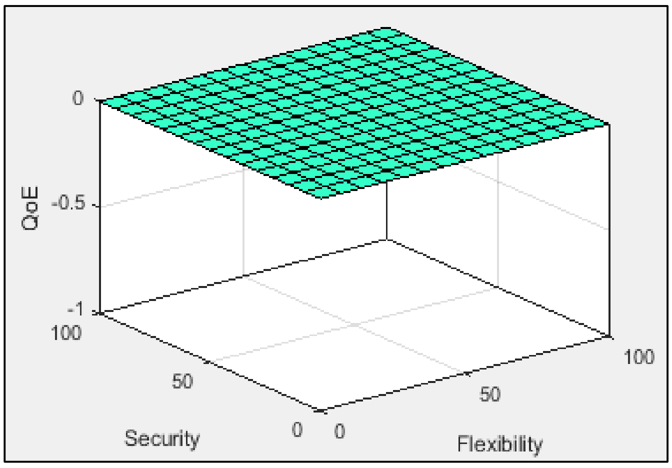


Figure B- 40 Effect of Security and Flexibility (LOM method)

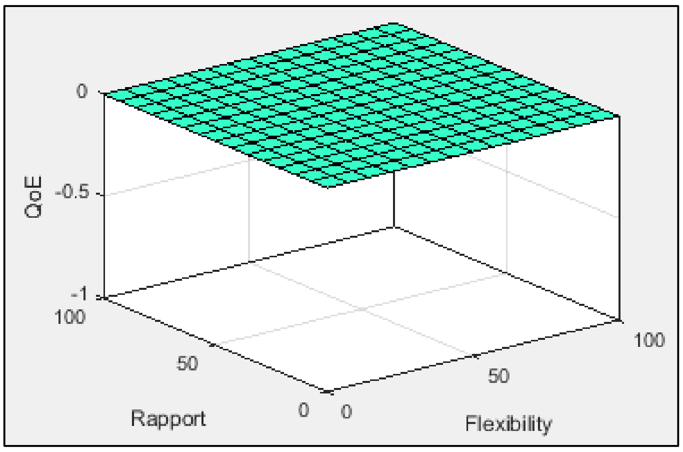


Figure B- 41 Effect of Rapport and Flexibility (LOM method)

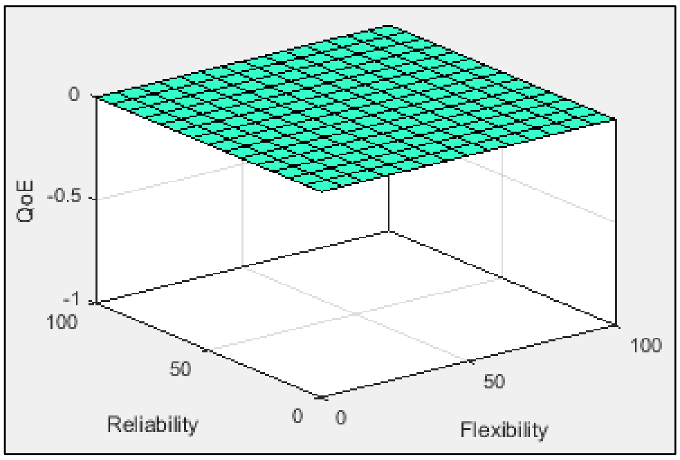


Figure B- 42 Effect of Reliability and Flexibility (LOM method)

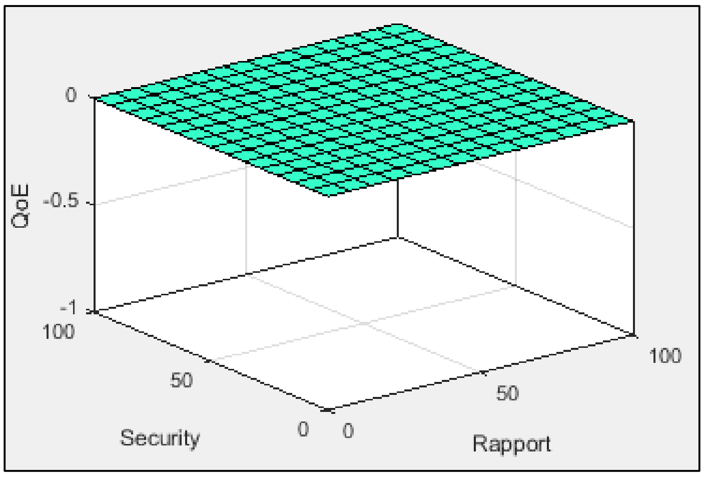


Figure B- 43 Effect of Security and Rapport (LOM method)

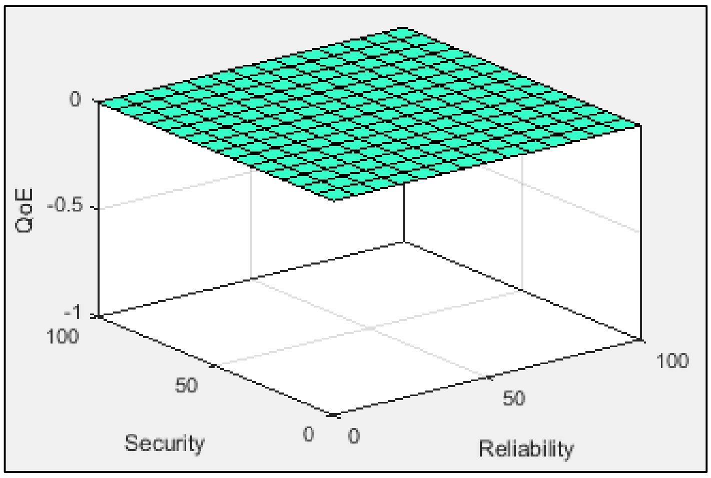


Figure B- 44 Effect of Security and Reliability (LOM method)

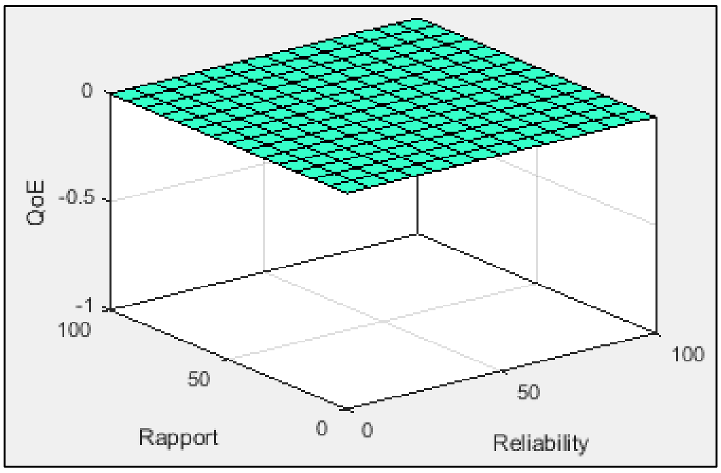


Figure B- 45 Effect of Rapport and Reliability (LOM method)

B5. Effect of SOM Method

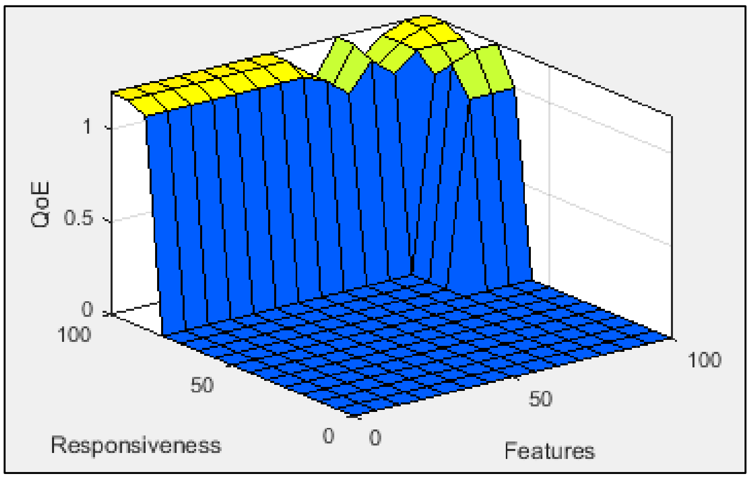


Figure B- 46 Effect of Responsiveness and Features (SOM method)

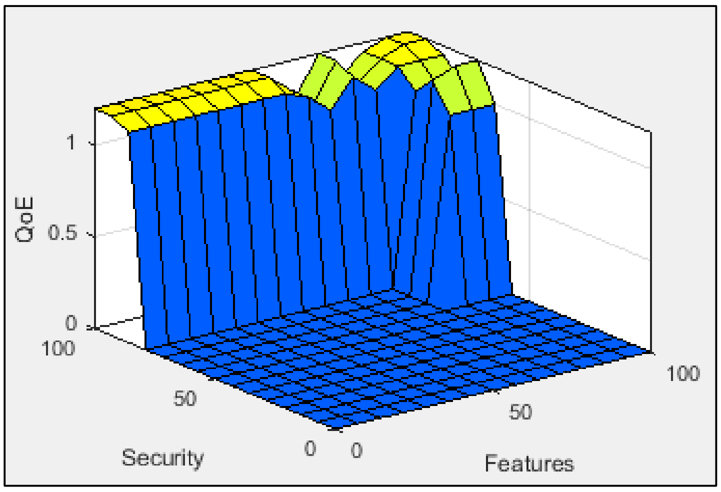


Figure B- 47 Effect of Security and Features (SOM method)

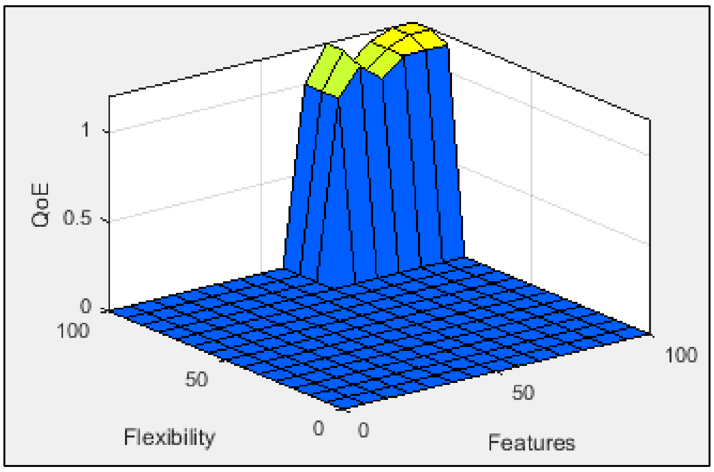


Figure B- 48 Effect of Flexibility and Features (SOM method)

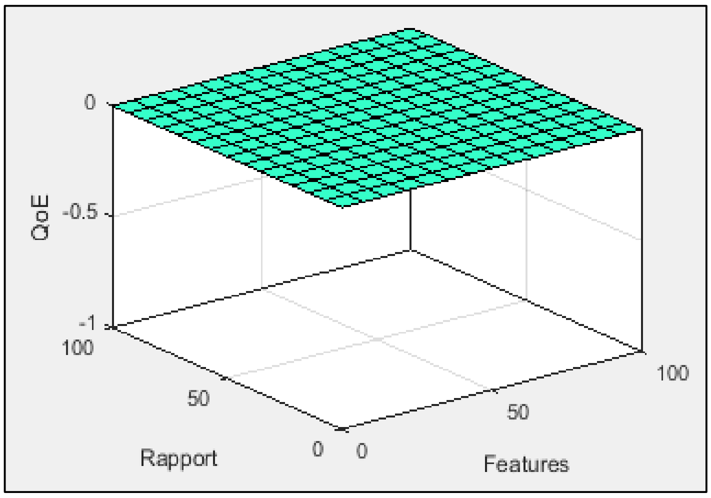


Figure B- 49 Effect of Rapport and Features (SOM method)

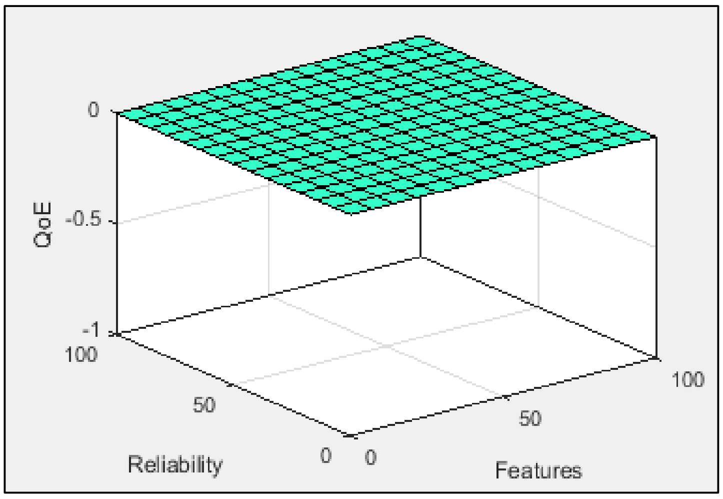


Figure B- 50 Effect of Reliability and Features (SOM method)

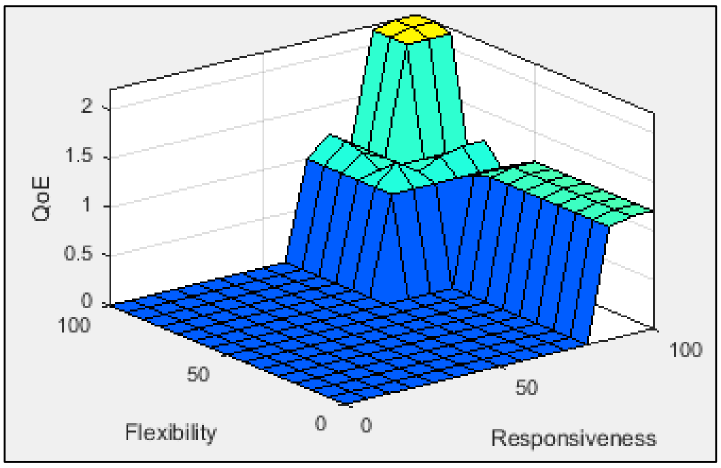


Figure B- 51 Effect of Flexibility and Responsiveness (SOM method)

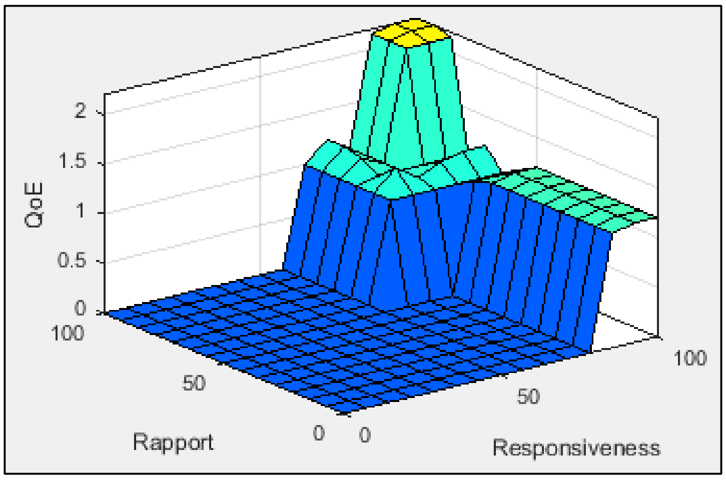


Figure B- 52 Effect of Rapport and Responsiveness (SOM method)

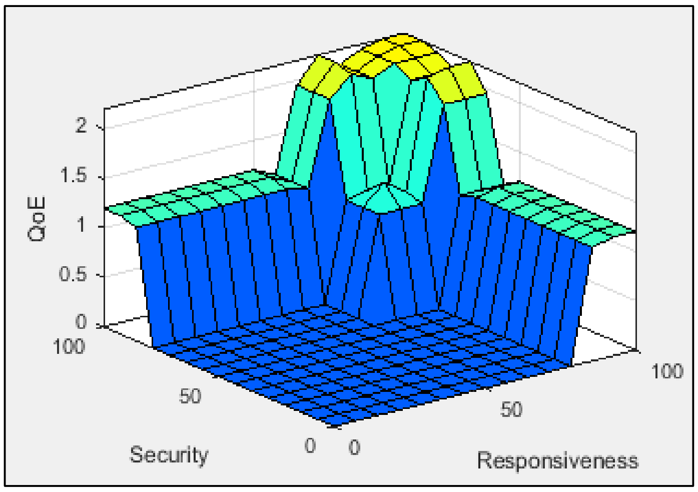


Figure B- 53 Effect of Security and Responsiveness (SOM method)

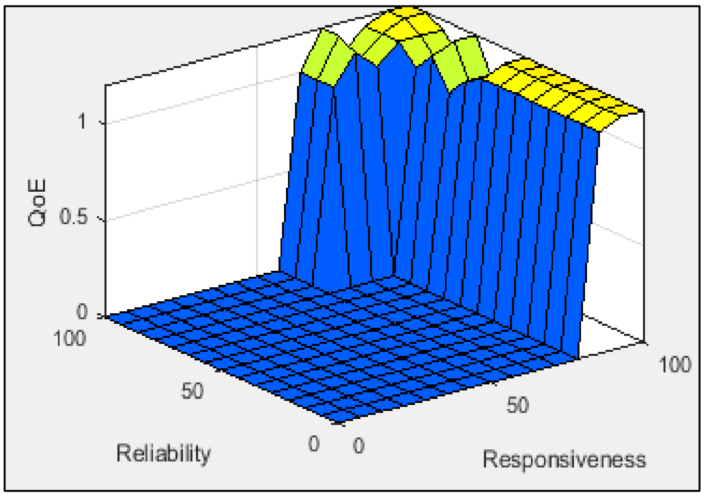


Figure B- 54 Effect of Reliability and Responsiveness (SOM method)

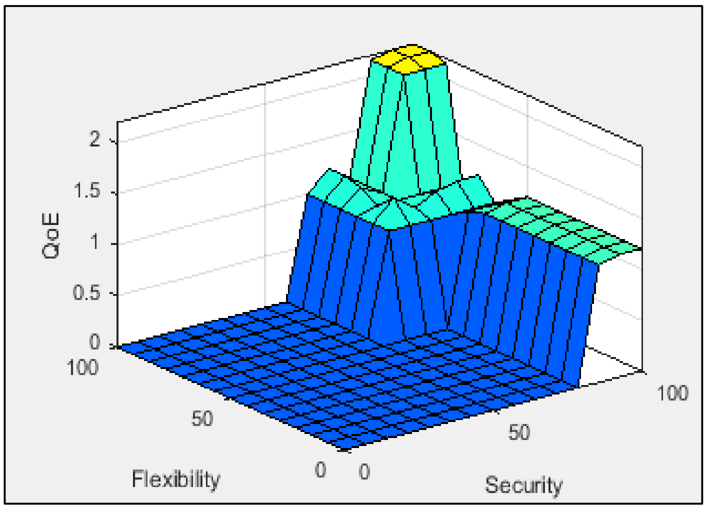


Figure B- 55 Effect of Flexibility and Security (SOM method)

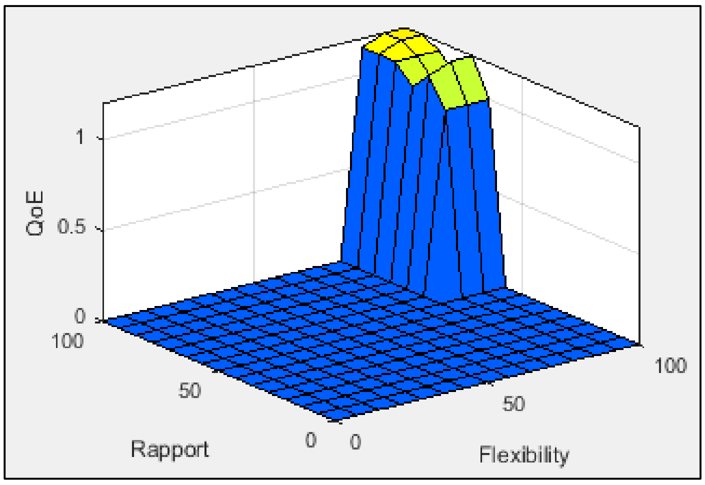


Figure B- 56 Effect of Rapport and Flexibility (SOM method)

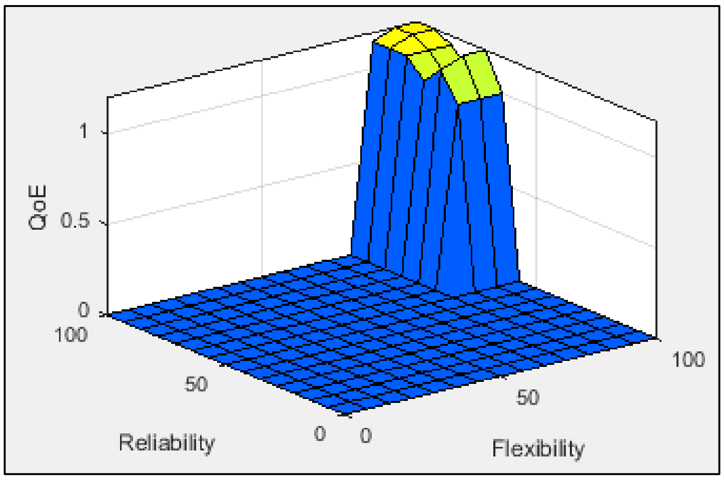


Figure B- 57 Effect of Reliability and Flexibility (SOM method)

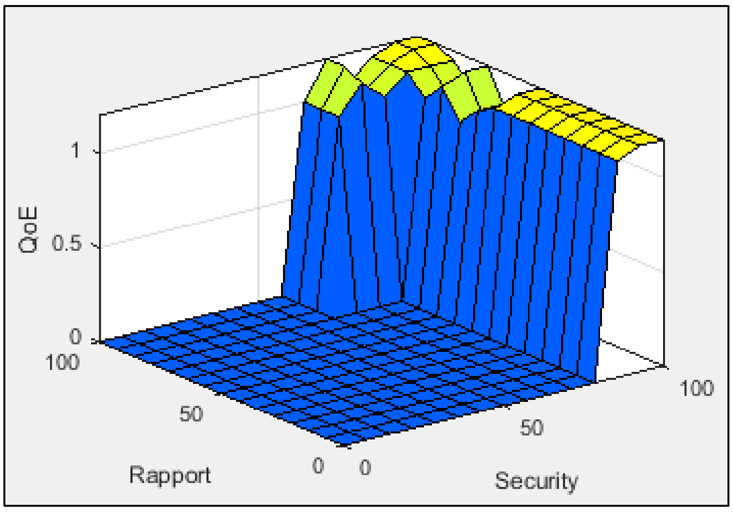


Figure B- 58 Effect of Rapport and Security (SOM method)

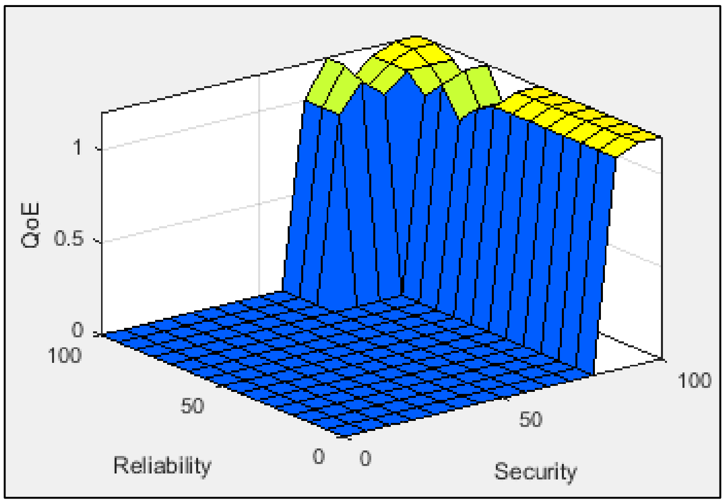


Figure B- 59 Effect of Reliability and Security (SOM method)

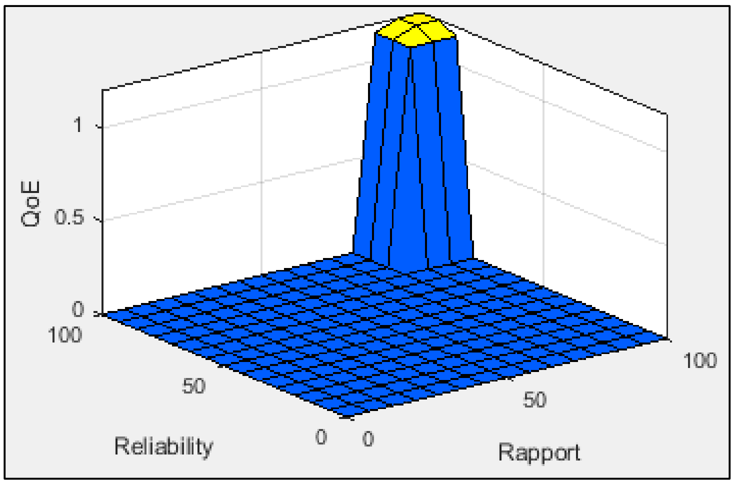


Figure B- 60 Effect of Reliability and Rapport (SOM method)

APPENDIX C

MonSLAR API SPECIFICATION

C.1 Introduction

This appendix presents the main API specifications for MonSLAR. The API is based on REST technology.

C.2 HTTP methods

HTTP methods are used to retrieve the resources of the REST, where these resources and their representations are the required monitored data. The API specification for MonSLAR is shown below:

```
{
  "swagger":"2.0",
  "info":{
    "version":"1.0.0",
    "title":""
  },
  "host":"localhost:8080",
  "basePath":"/MonSLAR/api",
  "tags":[
    {
      "name":"monitor"
    },
    {
      "name":"options"
    }
  ],
  "schemes":[
    "http"
  ],
  "paths":{
    "/monitor/qoe":{
      "head":{
        "tags":[
          "options"
        ],
        "summary":"Returns value of QoE",
        "description":"Returns qoe",
        "operationId":"HeadQOE",
        "produces":[
          "application/json"
        ],
        "parameters":[

        ],
        "responses":{
          "200":{
            "description":"successful operation",
            "schema":{
              "type":"double"
            }
          }
        }
      }
    },
    "/options/measure":{
      "options":{
        "tags":[
          "options"
        ],
        "summary":"Returns measured parameters values of options",
        "description":"Returns measured parameters values of options",
        "operationId":"getMeasures",
        "produces":[
          "application/json"
        ],
        "parameters":[

        ],
        "responses":{
          "200":{
            "description":"successful operation",
            "schema":{
              "type":"string"
            }
          }
        }
      }
    }
,
    "/sla ":{
      "options":{
        "tags":[
          "options"
          ],
          "summary":"Returns SLA updated parameters",
          "description":" Returns SLA updated parameters",
          "operationId":"getSLA",
          "produces":[
            "application/json"
          ],
          "parameters":[

          ],
          "responses":{
            "200":{
              "description":"successful operation",
              "schema":{
                "type":"string"
              }
            }
          }
        }
      }
  }
}
```

Figure C- 1 MonSLAR API specification

APPENDIX D

MonSLAR JAVA CODE

This appendix presents the java code for implementing the main requests used in MonSLAR. Figure D-1 introduces the code of the HEAD method used in the monitoring request-A, Figure D-2 shows the java code of the POST used for sending the measurements in monitoring request-B. The java code for the OPTIONS method used to implement the monitoring request-C is depicted in Figure D-3, whilst Figure D-4 introduces the code for the OPTIONS REST method used for the management request in the proposed middleware.

D.1 HEAD Method (Monitoring Request-A)

```
/*
Code shows the HEAD method to return the value of QoE
*/

@HEAD
        @Path("/qoe")
                public Response getHeader(@Context HttpHeaders headers, @Context
                HttpServletRequest request) throws Exception{

// Manage client details
                Integer currentUserId = (Integer) request.getSession().getAttribute(
                                "currentUserId");
                if (currentUserId != null) {
                        File file = new File ("QoEstimator.fcl");
                        FuzzyQoE fq = new FuzzyQoE();

// Retrieve the data from Fuzzy Logic

                        fq.RunFuzzy();
                        Double value = new DBConnect().getQoE(new Integer(currentUserId));
                        long result = new ReadFromJSON().GetValue();

// Return the QoE value in the header of the HEAD response

                return Response.ok().header("QoE", value).header("Violation", result).build();
                } else {

// Return error message

                        return Response.status(Response.Status.FORBIDDEN).build();
                        }
                }
```

Figure D- 1 depicts the java code of the HEAD method (monitoring request-A)

D.2 POST Method (Monitoring Request-B)

```
/*
Code shows the POST method to send the measurements from the client side
*/

@POST
// Create a new data base
        @Path("/post")
        @Consumes(MediaType.MULTIPART_FORM_DATA)
        public Response uploadFile(@FormDataParam("file") InputStream incomingData)
        {
                StringBuilder JSONBuilder = new StringBuilder();
                try {
                BufferedReader in = new BufferedReader(new InputStreamReader( incomingData));
                        String line = null;
                        while ((line = in.readLine()) != null) {
                                JSONBuilder.append(line);
                        }
                }
                catch (Exception e)
                {
                        System.out.println("Error Parsing: - ");
                }

                try {
                        FileWriter file = new FileWriter("c:\\MonitoredMetrics.json");
                        file.write(JSONBuilder.toString());
                        file.flush();
                        file.close();

                } catch (IOException e) {
                        e.printStackTrace();
                }

        return Response.status(200).entity("measurements has been uploaded
        successfully").build();
                }
        }
```

Figure D- 2 depicts the java code of the POST method (monitoring request-B)

D.3 OPTIONS Method (Monitoring Request-C)

```
/*
Code shows the OPTIONS method to return the values of the measured parameters
*/

@OPTIONS
        @Path("/measure")
        public Response getHeader1(@Context HttpServletRequest request)
                        throws JsonGenerationException, JsonMappingException, IOException {

// Manage client details

                Integer currentUserId = (Integer) request.getSession().getAttribute(
                                "currentUserId");
                if (currentUserId != null) {
                        List<MeasuredParameters> measuredParametersList = new

// Retrieve the values from the repository for the specific user

                        ReadMeasuredParameters()
                                        .getAllMeasuredParameters(new Integer(currentUserId));
                        ResponseBuilder responseBuilder = Response.ok();
                        responseBuilder = responseBuilder.header("result",
                                        new
                ObjectMapper().writeValueAsString(measuredParametersList));


// Embed the data in the header of OPTIONS response

                        return responseBuilder.build();
                }

// Return a forbidden http status error if data is not available

                else {
                        return Response.status(Response.Status.FORBIDDEN).build();
                }
        }
```

Figure D- 3 Depicts the java code of the OPTIONS method (monitoring request-C)

D.4 OPTIONS Method (Management Request)

```
/*
Code shows using the OPTIONS method to retrieve the SLA parameters values
*/

@OPTIONS
        @Path("/sla")
        public Response getHeader1(@Context HttpServletRequest request)
                        throws JsonGenerationException, JsonMappingException, IOException {

// Manage client details

                Integer currentUserId = (Integer) request.getSession().getAttribute(
                                "currentUserId");
                if (currentUserId != null) {
                        List<SLAParameters> SLAParametersList = new ReadSLAParameters()
                                        .getSLAParameters(new Integer(currentUserId));
                        ResponseBuilder responseBuilder = Response.ok();
                        responseBuilder = responseBuilder.header("result",
                                        new
                ObjectMapper().writeValueAsString(measuredParametersList));


// Embed the SLA parameters in the header of OPTIONS response

                        return responseBuilder.build();
                } else {

// Return a forbidden http status error if data is not available

                        return Response.status(Response.Status.FORBIDDEN).build();
                }
        }
```

Figure D- 4 Depicts the java code of the OPTIONS method (management request)

APPENDIX E

MonSLAR PERFORMANCE

Additional measurements have been presented to study the behaviour of the proposed middleware. This considered evaluating the overhead caused by the proposed middleware in terms of the time that it adds to the overall response time of the system.

In order to get the required performance measurements, a number of request-response pairs have been exchanged. This allowed the researcher to study the behaviour of the proposed middleware and acquire the overall response time caused by the monitoring dashboard, which includes the HEAD requests, in addition to the time required for collecting the monitored data from the database repository.

E.1 Experiment Objectives

The objective of the experiment is to evaluate the performance of MonSLAR in terms of the response time overhead.

E.2 Experiment Setup

The simulation environment is shown in **Error! Reference source not found.**. To i nvestigate the overhead caused by the proposed middleware, two scenarios were used. The first scenario 'With MonSLAR' was used to measure the response time overhead as a result of implementing MonSLAR, where the measured response time represents the use of the SaaS application in addition to the use of MonSLAR to monitor the user's satisfaction; while the second scenario 'Without MonSLAR' is about measuring the response time overhead caused by the SaaS application without MonSLAR.

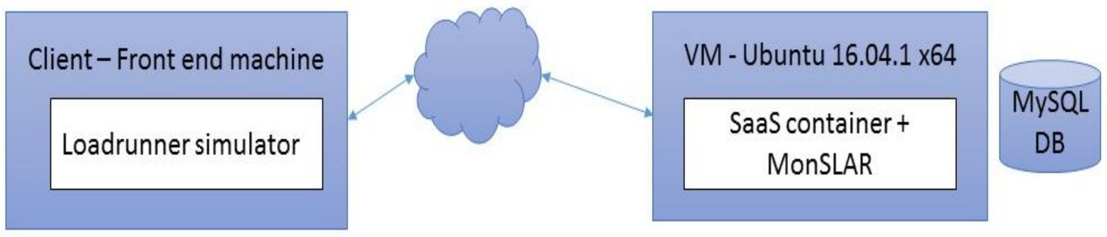


Figure E- 1 Experiment testbed architecture

For each of the aforementioned scenarios, the effect of changing the number of users on the response time was studied. The evaluation process considered evaluating the monitoring system with a different number of users (50, 100, 150, and 200). The measurements are collected for the four different numbers of users, to study the behaviour of the system in each of these cases. The results for these scenarios are discussed in the results section.

E.3 Experiment Results

The results for the measured overhead are presented in Figure E-2, which at the same time provide an indication of the ability of MonSLAR to manage the different number of users. Considering this set of user numbers helped in studying the behaviour of the monitoring system with increasing the number of users, as this set was considered in SLAM (Moustafa et al., 2015). Although the comparison with the available monitoring frameworks is not conclusive because of the different monitored applications and environments, it shows that additional response time overhead caused by the monitoring process is comparable to the overhead caused by the available monitoring frameworks.

Figure E-2 shows the average response time for four different group of users. It can be seen in Figure E-2 that the amount of the overhead for each case with and without using the monitoring system are comparable.

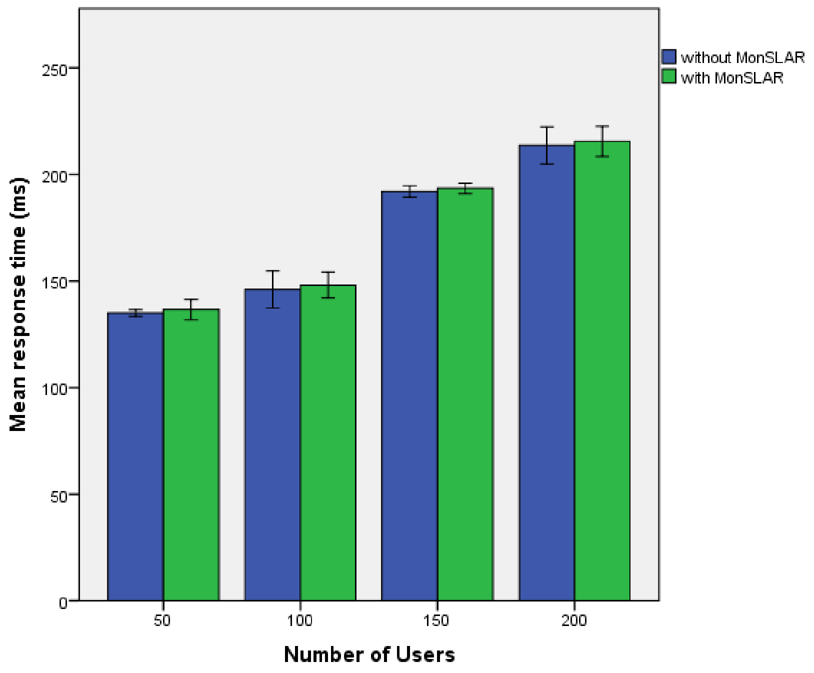


Figure E- 2 Response time overhead (With MonSLAR vs Without MonSLAR)

APPENDIX F

USER STUDY

This appendix presents the survey used to investigate the effect of SaaS-Qual parameters on QoE value. The survey involved a questionnaire to ask the participants about user satisfaction with respect to SaaS cloud computing.

F.1 Introduction

Thank you for agreeing to participate in this study that is being conducted at the University of Salford to help understanding the users' experience about cloud computing services based on parameters defined in the Service Level Agreement (SLA) which is the contract signed between the user and the provider that states the level of services delivered to the user.

Your feedback is very important. The results derived from the questionnaire will be used to evaluate the QoE value which has been estimated as part of the research.

F.2 Deciding the users' satisfaction

Suppose that you are receiving a service and you signed a contract with the provider, this contract states six different parameters. These parameters are: 1- Responsiveness: (The service provider's ability to ensure the availability and performance of the delivered application (e.g., disaster recovery planning) as well as the responsiveness of support staff (e.g., 24-7 hotline support availability));2- Reliability: (The provider's ability to perform the promised services timely, dependably, and accurately), 3- Flexibility: (The customers' ability to change the contract with the provider (e.g., cancellation period, payment model)), 4-Rapport: (The provider's ability to provide knowledgeable and support (e.g., joint problem solving)), 5- Features: (Means "application meet the business requirements of a customer" (e.g., user interface, reporting)); 6- Security: (usage of encryption, or antivirus technology).

In the cases below, please tick the answer that indicates how satisfied are you with the delivered services for the given cases, taking into your consideration: Bad, the user is not satisfied with the service (the service level is too low); medium, the user is not sure and can't decide whether the received service is good or bad (not clear); good, means the user is satisfied with the received service (this level of the SLA parameter is acceptable).

Table F- 1 User Study Cases

Case	Responsiveness	Reliability	Flexibility	Security	Features	Rapport	User Satisfaction				
							Strongly dissatisfied	Dissatisfied	Neutral	Satisfied	Strongly satisfied
1	Bad	Good	Good	Good	Good	Good					
2	Good	Good	Good	Good	Bad	Good					
3	Good	Good	Good	Bad	Good	Good					
4	Good	Good	Good	Good	Good	Bad					
5	Good	Good	Bad	Good	Good	Good					
6	Good	Bad	Good	Good	Good	Good					
7	Medium	Bad	Medium	Medium	Medium	Good					
8	Good	Good	Good	Good	Bad	Medium					
9	Bad	Good	Medium	Good	Bad	Good					
10	Good	Medium	Good	Bad	Good	Medium					