

## ACKNOWLEDGEMENTS

It gives me immense pleasure to thank everyone who has played a major role in the process of me doing my Ph.D. There are a number of people without whom this thesis might not have been written, and to whom I am greatly indebted.

I would like to express my deepest gratitude to the vice chancellor, Registrar and Dean Research and Consultancy Division, BITS Pilani for giving me an opportunity to do this research and enhance my professional carrier.

I would like to thank **Prof. Dr. R.N. Saha**, Director, BITS Pilani, Dubai Campus. I thank him for the constant encouragement, facilities and support that he has provided to carry out my research work.

I would like to acknowledge former-Director Prof. M. Ramachandran, for his encouragement to register for the Ph.D Programme and former-Director Prof. Dr. R.K. Mittal for his constant support to pursue the same. Sincere thanks to Dr.Neeru Sood, Associate Dean Academic Research and consultancy, BITS Pilani, Dubai campus and Dr. K.K. Singh, General Manager, BITS Pilani FZ LLC, for their constant motivation and update on research guidelines.

I would like to gratefully and sincerely thank my supervisor, Prof. Dr. S. Vadivel, BITS Pilani, Dubai Campus for his guidance, patience, technical inputs and encouragement which I felt are the most essential to my Ph.D. study.

I would like to thank the members of the Doctoral Advisory Committee BITS Pilani, Dubai Campus Dr. B.Vijay Kumar, Associate Professor, HOD, CS, BITS Pilani, Dubai Campus, Dr. V. Santhosh Kumar, Assistant Professor, BITS Pilani, Dubai Campus for their valuable suggestions during seminars and presentations.

Further, I would like to thank Dr. Sujala D Shetty, the Doctoral Research Committee Convener and Dr.Madiajagan, Assistant Professor and DRC member in BITS Pilani, Dubai campus for their constant support and advisory role throughout my research work. I thank all my department colleagues for their insightful opinions and valuable suggestions during my seminars.

I would like to thank the IT Support Team and Mr. Sheshadri, Librarian, BITS Pilani, Dubai campus for extending excellent technical facilities and providing excellent library resources on time which really helped me to complete my thesis. I would like to thank Mr. Asif Masood Library Assistant, BITS Pilani, Dubai Campus for helping me in taking print outs of my thesis.

I acknowledge all my colleagues, department technical staff for their support to complete my research work.

I am very thankful to my family for providing constant support these years to complete my research work.

Above all, I would like to thank the Almighty for giving me an opportunity to dream, to work and to reach to this level.

**SUSILA.S**

**2008PHXF030U**

## **ABSTRACT**

Web Services are internet enabled software components. Integration of Enterprise applications can be provided by composing Web Services,. Web Services are being used extensively by many enterprises like IBM, Microsoft and oracle every day. Web Services are becoming a major technique for building loosely coupled distributed systems. Service-oriented architecture (SOA) has been widely employed in e-business, e-government, automotive systems, multimedia services, process control, finance, and many other domains. As many Web Services are available in the internet for similar functionality, which service will be the best for the client requirement, is an elusive task for Web Services operators. So Web Services Selection (WSS) is an indispensable process for Web Services composition.

Quality-of-Service (QoS) is usually employed for describing the non-functional characteristics of Web Services and employed as an important differentiating point of different Web Services. With the prevalence of Web Services on the Internet, Web Services QoS management is becoming more and more crucial. The important issue recognized in the web service selection is the inability to successfully identify a web service that can meet the user's specific nonfunctional requirements in real time.

In this thesis, an extended SOA is implemented for Web Services selection based on quality metrics. The QoS data provided by the service provider through the modified WSDL, is collected by an agent web service by searching many of the Web Services published in the UDDI in that particular functional domain. The web service users can get the address of the agent Web Service for weather forecast which will guide the client for further selection process in that functional domain.

There is lack of web enabled service which can classify the web services into different classes based on their quality metrics. Design and implementation of a QoS classifier Web Service has been carried out using modified ID3 algorithm which uses entropy based discretization technique. This classifier Web Service takes QoS parameters as input

and applies an entropy based discretization algorithm and yields the classification of Web Services into different classes namely excellent, good, average and poor based on QoS values to the web based client. The decision tree rules arrived by the Classifier algorithm is given as input to the visualization Web Service to have tree view of the decision rules. To invoke services one after another a BPEL engine is used. First, the QoS ARFF file is sent to the data mining service, which then gives the output in the dot format. Then, the visualization service is invoked which gives the tree output.

There is a need to have a third party service which can give promised QoS values for the web services in the same functional group. In this thesis, an agent Web Service has been designed to measure important QoS parameter namely throughput in real time and to create a repository for the same. The agent in turn uses SOAPUI to perform a performance testing and the results from SOAPUI are then stored in the agent. When the user looks for a Web Service in a particular functional domain, the agent presents the client with the available QoS parameters, which the client can use to make a selection.

A Hospital Management Application has also been built using Web Services that can implement functions like online appointment booking, preparing a prescription online, online room reservation, storing patient's medical history etc. Web Services are developed as REST Web Service and SOAP Web Service and tested in SOAPUI for their performance. The QoS attributes of REST and SOAP Web Services are analyzed and compared. The scalability test showed that REST Web Services perform better compared to the SOAP Web Services. The application is designed and developed in Eclipse and is deployed to the Google App Engine to make it available online.

# TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS .....</b>	<b>i</b>
<b>ABSTRACT .....</b>	<b>iii</b>
<b>LIST OF TABLES .....</b>	<b>x</b>
<b>LIST OF FIGURES .....</b>	<b>xi</b>
<b>LIST OF ACRONYMS .....</b>	<b>xiv</b>
<b>CHAPTER 1 .....</b>	<b>1</b>
<b>Introduction.....</b>	<b>1</b>
1.1 Motivation .....	3
1.2 Research Gap.....	4
1.3 Objective and Scope of the thesis .....	6
1.4 Methodology .....	8
1.5 Limitations .....	9
1.6 Thesis Organization.....	9
<b>CHAPTER 2 .....</b>	<b>13</b>
<b>Literature Survey.....</b>	<b>13</b>
<b>Background of Web Services.....</b>	<b>13</b>
2.1 Web Services Standards .....	14
2.1.1 WSDL .....	16
2.1.2 UDDI.....	17
2.1.3 SOAP .....	18
2.2 Benefits of Web Services .....	20
2.3 QoS elements - ontology, classification.....	22

<b>CHAPTER 3.....</b>	<b>29</b>
<b>Implementation of Agent for WSS using Data mining technique. ....</b>	<b>29</b>
3.1 Agent Based Architecture. ....	29
3.2 Describing Web Services with QoS .....	31
3.3 WSDL extension with QoS .....	31
3.4 Trustworthiness of claimed QoS .....	32
3.5 QoS attributes that are considered for WSS.....	34
3.6 Feedback from Web Services client.....	38
3.7 Summary .....	39
<b>CHAPTER 4.....</b>	<b>40</b>
<b>WSS using Entropy Discretization Method. ....</b>	<b>40</b>
4.1 About the Dataset .....	41
4.2 Data Normalization .....	42
4.3 WEB SERVICES RELEVANCY FUNCTION (WsRF).....	42
4.4 Service Classification .....	44
4.5 ENTROPY DISCRETIZATION METHOD .....	45
4.5.1 Introduction.....	45
4.5 .2 ENTROPY .....	46
4.5 3 Selecting the Best Split .....	48
4.5.4 Applying entropy-based Discretization on Quality of Web Services Dataset. ....	49
4.6 Summary .....	63
<b>CHAPTER 5.....</b>	<b>64</b>
<b>Implementation of Agent for WSS using Entropy Discretization</b>	
<b>method.....</b>	<b>64</b>
5.1 Decision tree construction.....	65
5.1.1 Decision tree rule induction algorithm using entropy based discretization.....	67
5.1.2 QoS Classification Web Services .....	69

5.2 Summary .....	72
<b>CHAPTER 6.....</b>	<b>73</b>
<b>Implementation of Agent for WSS through Web Services Composition</b>	
<b>73</b>	
6.1 Composition of Web Services.....	73
6.1.1 BPEL4WS.....	74
6.2 WEKA.....	75
6.3 Graphviz .....	80
6.4 Implementation of J48 Classifier Web Service.....	82
6.5 Implementation of Visualization Web Service .....	85
6.6 Composition Of J48 Classifier Web Service and Visualization Web Service .....	92
6.6 Summary .....	94
<b>CHAPTER 7.....</b>	<b>96</b>
<b>Implementation of Agent for Measurement of QoS parameter using</b>	
<b>SOAPUI .....</b>	<b>96</b>
7.1 SOAPUI .....	97
7.2 Features of SOAPUI.....	97
7.3 Agent Implementation using SOAPUI.....	98
7.3.1 Weather Forecast Service- Example Service .....	99
7.3.2 Creating Project For Sample API .....	99
7.3.3 Generating Load TestSuite .....	101
7.3.4 Creating an Average Load Test Case for API .....	101
7.4 Functioning Model of Agent Architecture .....	103
7.4.1 DATABASE CREATION IN MYSQL USING PHPScript.....	103
7.4.2 EXTRACTING QoS ATTRIBUTES FROM MySQL DATABASE.....	104
7.5 Summary .....	105
<b>CHAPTER 8.....</b>	<b>107</b>
<b>Analysis of QoS attributes for REST and SOAP Web Services.....</b>	<b>107</b>

8.1 REST Web Services .....	107
8.1.1 Characteristics of REST.....	108
8.1.2 Example of REST Web Services .....	108
8.2 Design and Implementation Of Database driven Hospital Management System using Web Services .....	109
8.2.1 Tools used .....	110
8.2.2 Methods.....	110
8.2.3 Simple Hospital Management Database in MySQL.....	111
8.2.4 ER Diagram of the database model .....	112
8.2.5 Implementation Of Hospital Management Sytem Using REST Web Service .....	113
8.2.6 OUTPUT of the REST Web Services.....	114
8.2.7 LOAD test for the REST Web Services .....	118
8.2.8 Implementation Of Database driven Hospital Management Using SOAP Web Services .....	122
8.2.9 OUTPUT of database driven SOAP Web Services .....	124
8.2.10 LOAD test for the SOAP Web Services .....	126
8.2.11 Performance Comparison of SOAP and REST Web Services .....	127
8.2.12 Summary .....	129
8.3 Performance measurement of Hospital Management Application using REST Web Service Deployed On Google SQL Cloud .....	130
8.3.1 Database created for the Hospital Management Web Services .....	130
8.3.2 SQL .....	130
8.3.3 Connecting to the database in Google SQL Cloud .....	131
8.3.4 Home page for the Web Services.....	132
8.3.5 Performance of the Web Services.....	134
8.5 Summary .....	139
<b>CHAPTER 9 .....</b>	<b>140</b>
<b>Conclusion, Contributions and Future work.....</b>	<b>140</b>
9.1 Conclusion.....	140
9.2 Contributions.....	142
9.3 Future work .....	142
<b>REFERENCES.....</b>	<b>144</b>
<b>APPENDICES.....</b>	<b>157</b>
<b>APPENDIX – A.....</b>	<b>157</b>



System configurations .....	157
Software used .....	158
<b>APPENDIX – B-1.....</b>	<b>159</b>
Functions and their actions for the Decision Tree Construction for Continuous attributes .....	159
<b>APPENDIX – B-2.....</b>	<b>160</b>
WSDL File .....	160
<b>APPENDIX – B-3.....</b>	<b>161</b>
Functions and their actions for the Web Service Composition.....	161
<b>Appendix B-4.....</b>	<b>169</b>
Code Details of Hospital Management Web Service.....	169
<b>APPENDIX - C.....</b>	<b>174</b>
Data set on QoS.....	174
<b>LIST OF PUBLICATIONS .....</b>	<b>185</b>
International Journals .....	185
International Conferences .....	186
<b>BRIEF BIOGRAPHY OF THE CANDIDATE .....</b>	<b>187</b>
<b>BRIEF BIOGRAPHY OF THE SUPERVISOR .....</b>	<b>188</b>

## LIST OF TABLES

Table 4.1 Non-functional attribute values with units .....	41
Table 4.2 Input Dataset.....	45
Table 4.3 Sorted Dataset.....	49
Table 8.1 Observation Table for LOAD test .....	119
Table 8.2 Observation table for LOAD test .....	120
Table 8.3 Observation Table for LOAD test.....	126
Table 8.4 tps of REST and SOAP .....	128
Table 8.5 bps of REST and SOAP.....	128
Table 8.6 bps of REST and SOAP.....	129
Table 8.7 restrial1.appspot.com.....	135
Table 8.8 RESTtrial1.appspot.com/doctors.jsp .....	135
Table 8.9 RESTtrial1.appspot.com/patients.jsp .....	136
Table 8.10 resttrial1.appspot.com/rooms.jsp.....	136
Table 8.11 resttrial1.appspot.com/reserve.jsp .....	137
Table 8.12 resttrial1.appspot.com/login.jsp.....	137
Table 8.13 restrial1.appspot.com/patient_history.jsp.....	138

## **LIST OF FIGURES**

Fig 2.1 Protocol stack of Web Services.....	15
Fig 2.2 Web Services Interactions.....	16
Fig 2.3 WSDL structure[11].....	17
Fig 3.1 Extended SOA for WSS.....	30
Fig 3.2 Part of WSDL file.....	35
Fig 3.3 Table maintained by the agent.....	36
Fig 3.4 Classification output.....	37
Fig 3.5 Cluster output.....	37
Fig 3.6 client feedback form.....	38
Fig 4.1 Distribution of range of points.....	45
Fig 4.2 Partial Tree Constructed 1.....	51
Fig 4.3 Partial Tree Constructed 2.....	51
Fig 4.4 Partial Tree Constructed 3.....	53
Fig 4.5 Partial Tree Constructed 4.....	54
Fig 4.6 Partial Tree Constructed 5.....	56
Fig 4.7 Partial Tree Constructed 6.....	56
Fig 4.8 Partial Tree Constructed 7.....	58
Fig 4.9 Partial Tree Constructed 8.....	59
Fig 4.10 Partial Tree Constructed 9.....	60
Fig 4.11: Partial Tree Constructed 10.....	61
Fig 4.12: Partial Tree Constructed 11.....	62

Fig 5.1 Agent based architecture for Web Services selection. ....	<b>67</b>
Fig 5.3 Decision tree traversal and classification using Decision tree rule induction algorithm.....	<b>69</b>
Fig 5.4 IDE's tester page for the QoS classification Web Services. ....	<b>70</b>
Fig 5.5 IDE's Method invocation trace for the QoS classification Web Services. ....	<b>70</b>
Fig 6.1 Graphviz Output .....	<b>81</b>
Fig 6.2 J48 Visualization service output .....	<b>92</b>
Fig 6.3 BPEL process .....	<b>93</b>
Fig 6.4 Composite application of the BPEL module.....	<b>94</b>
Fig 7.1 Weather Forecasting Agent Implementation using SOAPUI .....	<b>99</b>
Fig 7.2 Snapshot of New Project Dialogue box .....	<b>100</b>
Fig 6.3 Snapshot of Load Test Suite.....	<b>100</b>
Fig 7.4 Snapshot of Generate Test Suite .....	<b>101</b>
Fig 7.5 Snapshot of Run Test results .....	<b>102</b>
Fig 7.6 Load test Statistics Log File .....	<b>103</b>
Fig 7.7 Automatically generated SQL query .....	<b>104</b>
Fig 7.8 SQL table generated in PHPMyAdmin.....	<b>104</b>
Fig 7.9 Snap shot of database obtained. ....	<b>105</b>
Fig 7.10 Snapshot of application front end.....	<b>105</b>
Fig8.5ER Diagram of the database model created in MySQL Workbench	<b>113</b>
Fig 8.6 Part of code showing the path definition and the GET method.....	<b>114</b>
Fig 8.7 Output of the REST Web Services- index.jsp file .....	<b>115</b>

Fig8.8WADL of the Web Services.....	<b>116</b>
Fig 8.9 Page displayed when URL http:// localhost:8080/WebApplication19/resources/staffs is accessed .....	<b>117</b>
Fig8.11Graph for LOAD test 2.....	<b>121</b>
Fig 8.12 Graph for LOAD test 3.....	<b>121</b>
Fig 8.13 Graph for LOAD test 4.....	<b>122</b>
Fig8.14Web Operations of a SOAP Web Services .....	<b>123</b>
Fig8.15 Output of the SOAPWeb Services when it is tested .....	<b>124</b>
Fig 8.16 Output on entering the address id in the web operation.....	<b>124</b>
Fig 8.17 Output of the addressDetails web operation when address id “2” is inputted .....	<b>125</b>
Fig 8.18 Graph for LOAD test 2.....	<b>126</b>
Fig8.19 Graph for LOAD test 3.....	<b>127</b>
Fig 8.20 Homepage for the Database driven Web Services hosted on the Google Cloud.....	<b>132</b>

## LIST OF ACRONYMS

- *API* Application programming interface
- *ARFF* Attribute-Relation File Format
- *BPEL* Business Process Execution Language
- *BPEL4WS* Business Process Execution Language For Web Services
- *GAE* Google App Engine
- *NF* Non- Functional
- *QoS* Quality Of Service
- *SOA* Service Oriented Architecture
- *SOAP* Simple Object Access Protocol
- *SOAPUI*- Simple Object Access Protocol- User Interface
- *SQL* Structured Query Language
- *UBR* UDDI Business Registry
- *UDDI* Universal Description, Discovery and Integration
- *URI* uniform resource identifier
- *URL* Uniform Resource Locator.
- *WADL* Web Application Description Language
- *WEKA* Waikato Environment for Knowledge Analysis
- *WSDL* Web Services Description Language

- *WSRF Web Service relevancy function*
- *WSS Web Service Selection*

# CHAPTER 1

## Introduction

Research on applying data mining technology to the Web, or Web mining, is given a boost when “information overload” becomes a reality in the Web community. Over recent years, the Web has evolved beyond just being a source of data to also being a source of applications that provide services to users for a variety of requests (e.g. Google for content search, Expedia for travel requests, Interflora for flower delivering, and Amazon for e-commerce related search and purchasing activities). Building blocks for Web-based description, discovery and invocation have been evolving in order to enable increasing degrees of automated processing. This has resulted in the development of the Web Services architecture and its related set of standards and methods. From a consumer’s point of view, knowing the Quality of Service (QoS) of the Web Services plays a crucial role in choosing a particular Web Service over its alternatives. QoS describes the capabilities of a product or service to meet the requirement of consumers. It serves as a benchmark to differentiate the services and the service providers. Quality is the totality of features and characteristics of a service that bears on its ability to satisfy stated or implied needs. Therefore, knowing Web Services ranking based on its QoS becomes vital for both the Web Services provider and the Web Services consumer.

Users who request either for simple or for composite Web Services face the problem of identifying “what is out there on the Web” which is similar to the search problem faced by the users looking for available text content. Just as the users looking for page ranking need Web mining, the users looking for services need service mining. Similarly, methods to search for available services need efficient and knowledgeable means of identifying relevant options for a satisfactory response to the service request.



Service mining is discovery and analysis of Web Services registered with the registry by using the service knowledge. In the Web Services model, the service registry has access to the knowledge required for service mining. Service discovery mining aims to discover services that meet the specified requirements in terms of the service profiles, grounding, and QoS constraints.

It is all about service knowledge. Services are semantically more complicated than data. Different aspects of services are needed to be considered. Therefore, an important part of the service mining is analyzing QoS attributes and properties that describe the services.

With the swelling use of Web Services in standardization of basic content integration, support of complex service-oriented architectures, provision of seamless integration of business processes and applications etc. has led to a boost in numbers of both - Web Services consumers and providers. Thus, QoS becomes a very important aspect in distinguishing the success of a Web Services provider.

The attributes of interest in a Web Service can come under two broad categories - functional attributes and non-functional attributes. In the case of functional attributes, they are described as what the Web Services is about with respect to domain specific details. Functional properties include the input, output, Conditional output, precondition, access condition and effect of service. These can be well understood only by the users who have interest in the specific business domain, whereas Non Functional (NF) attributes are described as how the Web Services are in terms of quality. Here Web Services selection is done based on the NF attributes of the Web Services. The Agent based architecture is proposed for the selection of Web Services based on their different QoS attributes, and the agent is implemented using different techniques. Composition of Web Services is done to achieve Web Services Selection based on desired QoS. A database driven Web Service is implemented using SOAP and REST protocol. An attempt is made to do QoS comparison of Web Services implemented using SOAP and REST Web Services for the same application.

Hospital management System has been developed using SOAP and REST Web Services and they are compared for their QoS. The same application is deployed in Google cloud and their performance in the cloud is also analyzed.

## **1.1 Motivation**

"Web Services are loosely-coupled, self-contained, Web-accessible programs that can be published, discovered (or located), composed, invoked, and executed. Web Services provide a standardized means for diverse, distributed software applications to be published on the Web and to interoperate seamlessly".[1] Corporations are progressively providing services or programs within and between organizations either on corporate intranets or on the cloud[1]. With the increase in number of Web Services providing similar functionalities, more emphasis is placed on how to find the best Web Service that fits the client's requirements.

Let's see an example. A travel Agency is looking for a Web Service to obtain the real time plane ticket booking for its business management system. By searching some UDDI registries, such as those provided by IBM and Microsoft, the company could find hundreds of Web Services for any particular functional domain. Then, if the company tries to contact the service providers, there might be many of them that are not available right away due to several reasons. In this scenario, if a trust worthy service is available that can help the company identify which service provider has the best QoS attributes, the company can use that information to its benefit instead of wasting time in identifying which Web Service works the best.

In addition, allowing current consumers to rate the quality of services they receive can provide consumers with valuable information on how to rank services and how to select Web Services. The traditional means of Web Services discovery is to navigate directly to a known Web Service address. The problem with traditional methods is that the

performance metrics of most of the available Web Services are generally not known to any user.

This makes having searching techniques to discover the Web Services of interest essential. With organizations implementing SOA for daily transactions in business to business and business to customer processes, a huge amount of effort is being put into making the discovery of services automatic and more accurate. Recently, Quality of Service (QoS) has been considered as an important parameter and thus methods have been proposed to improve the accuracy of service discovery considering the QoS of Web Services as an important aspect.

## **1.2 Research Gap**

Web Services can be discovered from UDDI, which is an XML-based registry, enabling companies to publish and discover Web Services on the Internet. By crawling Web Services information in, there are about 21,358 addresses of WSDL (Web Services Description Language) files, which provide XML-based descriptions of Web Services interfaces.

Seekda.com [2] reports that there are totally 28,529 public Web Services in the Internet. From that it can be understood that there are so many Web Services available for the same functionality. Industrial practice witnesses a growing interest in the ad-hoc model for service composition in the areas of supply chain management, accounting, finances, eScience as well as in multimedia applications. There is always a need for composition of web services. As the number of web services available for any functional domain is increasing more and more, the composition problem becomes a decision problem on the selection of component services from a set of alternative services that provide the same functionality but differ in quality of service parameters[3].

Web Services selection is the process of selecting suitable service according to the client's needs. Web Services selection is an indispensable process for Web Services

composition to select best Web Services according to a client's requirement. Which service will be the best suited for the client's requirement is an elusive task for Web Services operators. Service registries are becoming very large, preventing users from discovering desired service. Sometimes users may not be aware of services that can be most beneficial to them.

The present research work addresses the following issues as research gap,

- Main issue identified in this research area of QoS based web service selection is the inability to successfully identify a web service which can meet the user's specific nonfunctional requirements in real time.
- For analyzing the QoS values of web services, already available repositories are used in plenty of methods whereas in this thesis modified WSDL is suggested, from where the extraction of quality attribute values will be done by the agent web service.
- Further, there is lack of literacy for web service client communicating with the software for identifying the web service that satisfies his QoS requirements. Previous work did not take the advantage of developing agent for web service selection itself as web service.
- In previous research work, the efficient classification algorithms work fine with discrete and nominal value attribute values where as the proposed thesis could classify the continuous values of QoS metric into different groups using entropy discretization method.
- There is a lack of common framework for measurement of unknown values of quality attribute values of web services. There is a need to have third party service which can give promised QoS values for the web services in the same functional group.

- There is a deficiency in the experimental QoS values for SOAP and REST services in the real time scenario.

### **1.3 Objective and Scope of the thesis**

With the corporate world moving towards Service Oriented Architecture (SOA), Web Services has become very vital today. In most service-oriented architectures, business to business systems, Web Services play a major role in conducting daily transactions and information exchange. A Web Service is a public interface of an application which can be invoked remotely to perform a business function or a set of functions.

Knowledge of Web Services includes service profile information, such as service provider's contact information, service operation input, service operation output and service grounding information such as the protocol used to interact with the service, and service usage data such as patterns associated with the set of the service. With current Web Services implementations, knowledge of service profiles and of service grounding is described and presented in service description documents.

However, the discovery of these two types of knowledge is limited, because all the relevant aspects of services such as their non-functional properties are not presented in an obvious manner. This limits the use of profile and grounding information in discovering services. The third type of knowledge, i.e. knowledge about service constraints, is presented in service description documents, but there are major limitations with the nature of constraints that can be represented and in how to process those constraints for service discovery. Due to the above problem, the results of service discovery often fail to satisfy the needs of the service request, and the extent of the service discovery failure increases with more complicated, composite service requests.

The solution for the problem of discovering Web Services is to mine the services for the process of selection, which is known as Service mining. Web mining is the use of data

mining techniques to automatically discover and extract information from Web document and services. The differences between Service mining and Web mining, however, are that for the former, a catalog of the services already exists in the service registry. It is not necessary to build an index structure by the discovery agent. Since, service information is well categorized and organized in the catalog, a basic search on the catalog by the service registry is more efficient than exploring the untamed Web space using a Web robot, such as WebCrawler or AltaVista. After interesting services are identified, direct accesses to the service description documents in the providers' Web sites are simple for the user. Service description documents are semi-structured XML documents, none or very little data cleaning and preprocessing in Web content mining is required here. The second difference is that service mining operates on services and aims to discover services amongst the ones registered with the registry based on QoS. So, this is not simply document mining or text mining.

For carrying out Web Service mining, Web Service is mined for its QoS properties. Selecting an appropriate approach for service selection is a very important task. However, selection procedure can be very complex and challenging as many non-functional properties like availability, accessibility, integrity, performance, reliability, regulatory, security have to be kept in mind. The main issues that need to be addressed at time of service selection are-

- a) Finding a group of services that serves the client's needs in terms of pure service methods and parameters (functional attributes)
- b) Evaluating the performance of the services
- c) Comparing the performance of various services to enable the user to make the best possible choice based on QoS.

The first and fundamental objective of the thesis is to ease the process of selecting a Web Service when there are plenty of Web Services registered for the same purpose in the UDDI registry. To select Web Service in that particular functional domain, there is going to be a Web Service which aids the process of selection of Web Services. So, this Web

Service, which aids the process of selection, is named as Agent Web Service. This Web Service has to know the details of all QoS attributes of all Web Services. Comparative Performance analysis on QoS of REST Web Service and SOAP Web Service is done with the help of database driven application. And performance analysis is carried out after deploying the application in the cloud environment.

The objectives of this thesis are:

- Study the research works to better understand the needs and the work already done for Innovative research for Web Service Selection for the same functional domain.
- Define Quality of Service for Web Services.
- Propose extended service oriented architecture for service mining based on QoS.
- Propose architecture to evaluate Web Services for their QoS Attributes.
- To measure and compare QoS attribute values for REST and SOAP Web Services.
- To measure the response time of REST Web Service deployed in cloud, by performing stress testing.

## **1.4 Methodology**

Web Service providers need to register with UDDI registry with extended WSDL file where in the QoS details of the Web Services are also provided along with other details. The client can also register his feedback after consuming the service. The QoS Agent Web Service collects the QoS attribute values of Web Service from their respective

WSDL documents in a database table. The feedback of the client who consumes the Web Service is also added to the database, maintained by the QoS Agent Web Service. A new entropy based data mining algorithm is proposed for the classification of Web Services under different classes, namely Excellent, Good, Average and Poor by the agent. SOAPUI a software tool is used in measurement of QoS attributes of Web Services for creating a new repository of Web Services with QoS. A Database driven SOAP and REST Web Services are created for the same application and their QoS attributes are compared.

## **1.5 Limitations**

Handling the dynamic nature of Web Services, such as sudden disappearance of certain Web Services, or consumer's change of business process or requirements for Web Services is still a challenging problem. To achieve the goal of dynamic Web Service selection which will enable the consumers to discover Web Services satisfying their requirements automatically at run time instead of at design time, QoS enhanced Web Service selection must be automated dynamically. Hence, the limitations of the thesis are to identify the trustable source of QoS data and deal with dynamic Web Service selection.

## **1.6 Thesis Organization**

The rest of this thesis is organized as follows:

### Chapter 2

It deals with the literature review of the research topic and gives overview of Web Services and its standards.

### Chapter 3

Agent based method for Web Services selection is discussed in this chapter. Here, the WSDL file is extended to have QoS attribute tags described by the service provider. After



using the Web Services, the client is asked to specify his experiences about the QoS metrics via his interface. Data mining algorithms are applied on the data that are collected from Extended WSDL files and feedback that are taken from the Web Services users. By applying data mining, the agent can discover some interesting QoS patterns for the future users of the Web Service.

#### Chapter 4

This chapter deals with manual way of ranking and selection of Web Services on the basis of Entropy-Based Discretization with the help of QoS constraints values and classifying them under corresponding service class. Using ranking (service classifier), client can easily choose the relevant Web Services. The algorithm is explained in detail in this chapter.

#### Chapter 5

The ranking and selection of Web Services on the basis of Entropy-Based Discretization with the help of QoS values is explained properly. In this chapter, the same process is implemented using JAVA programming, and having discovery agent Web Services perform the job of Web Services selection for the consumer reduces the complexity for the end user in selecting the Web Service based on their QoS values.

#### Chapter 6

In this chapter, the implementation of real time data mining for service discovery is done using composition of Web Services. Since the classification rules can be better understood with tree visuals, Web Service Composition using BPEL engine is used. The data mining algorithms for classification and visualization will be exported as Web Services and then the SOAP response of classification algorithm will be used to generate visualization outputs as SOAP response in the forms of graphs and trees.

## Chapter 7

Most of the service providers do not supply vital performance related information about their Web Services. In general, the most significant QoS attribute of the Web Service is throughput. Due to that, load test is conducted on Web Services using SOAPUI software to create a repository of Web Services that incorporate QoS attributes. This chapter's primary goal, is to create a repository of QoS value for the Web Services in any particular functional domain. It can assist the user in selection of an appropriate Web Services based on throughput requirements.

## Chapter 8

In this eighth chapter, an attempt is made to compare the QoS of a database driven application implemented using REST and SOAP Web Services which so far is done only using Multimedia messages. Here, a Hospital Management Application is built using Web Services that can implement functions like online appointment booking, preparing a prescription online, online room reservation, storing patient's medical history etc. The application is designed and developed using MySQL workbench, Net beans, Eclipse, Google SQL cloud and Google App Engine. Two different methods are considered for developing a hospital management application. One method implements this application as REST and another as SOAP Web Services hosted within the server running on local host. The application designed and developed is deployed to the Google App Engine to make it available online and to facilitate online transactions via the internet. The scalability testing of the cloud enabled hospital management system has also been done using a custom developed software tool.

## Chapter 9

In this chapter, the conclusion, contribution and future scope of the thesis is discussed.

The Appendices provide supplementary details on system configuration. The code details that are used to implement the algorithms are also given. The reference and the list of publication relating to this thesis is given at the end.

## CHAPTER 2

### Literature Survey

#### Background of Web Services

“Web Services are encapsulated, loosely coupled contracted functions offered via standard protocols”[4] where:

- “Encapsulated” defines the implementation of the assignment is absolutely not distinguished from the outsider.
- “Loosely coupled” defines varying the implementation of one operation does not require modification of the called operation.
- “Contracted” defines there are explicitly available definitions of the operation’s activities, how to relate to the operation along with its input and output constraints.[4]

A Web Service is a method of communication between two electronic devices over a network. The W3C defines "Web Services" as "a software system designed to support interoperable machine-to-machine interaction over a network[4]. It has an interface described in a machine-processable format (specifically Web Services Description Language WSDL). “Other systems interact with the Web Services in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.” [4]

Web Services constitute a distributed computer architecture made up of many different computers trying to communicate over the network to form one system. They consist of a set of standards that allow developers to implement distributed applications - using radically different tools provided by many different vendors - to create applications that use a combination of software modules called from systems in disparate departments or from other companies.

A Web Service contains some number of classes, interfaces, enumerations and structures that provide black box functionality to remote clients. Web Services typically define business objects that execute a unit of work (e.g., perform a calculation, read a data source, etc.) for the consumer and wait for the next request. Web Services consumer does not necessarily need to be a browser-based client. Console-based and Windows Forms-based clients can also consume Web Services. In each case, the client indirectly interacts with the Web Services through an intervening proxy. The proxy looks and feels like the real remote type and exposes the same set of methods. Under the hood, the proxy code really forwards the request to the Web Services using standard HTTP or optionally SOAP messages.

## **2.1 Web Services Standards**

One of the key attributes of Internet standards is that they focus on protocols and not on implementations. The Internet is composed of heterogeneous technologies that successfully interoperate through shared protocols [5]. This prevents individual vendors from imposing a standard on the Internet. Open Source software development plays a crucial role in preserving the interoperability of vendor implementations of standards.

Web Services standards provide an open standards based communication framework that operates within the purview of W3C guidelines. Web Services provide a platform/technology independent communication methodology while SOA is an overall IT strategy framework that aims to provide business agility. The protocol stack of the Web Services is described in figure 2.1.

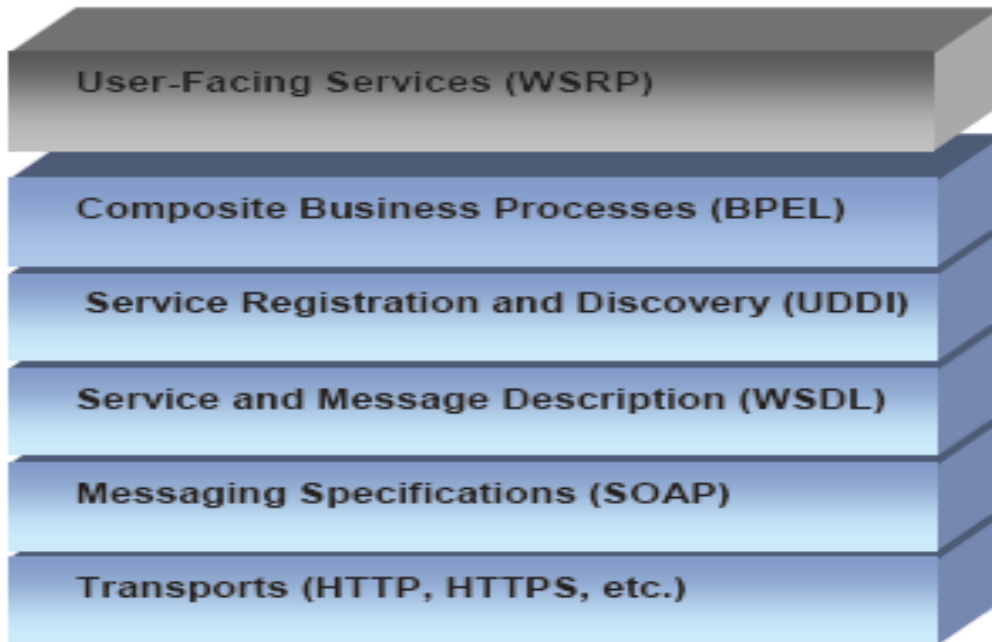


Fig 2.1 Protocol stack of Web Services[6]

**SOAP**- xml –based protocol to exchange structured information among services

**WSDL**-interface between service provider and service consumer and describes abstraction functionality of a service.

**UDDI**-service registry- Defines a set of API to support publication and discovery of Web Services.

**BPEL**-xml based language for formal description of business and business interaction protocols

**WSRP**-gives definition of Web Services interface for accessing and interacting with interactive presentation oriented Web Services.

The following standards play key roles in Web Services: Universal Description, Discovery and Integration (UDDI), Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP).The relationship between these standards is described in figure 2.2 and are explained further in detail.

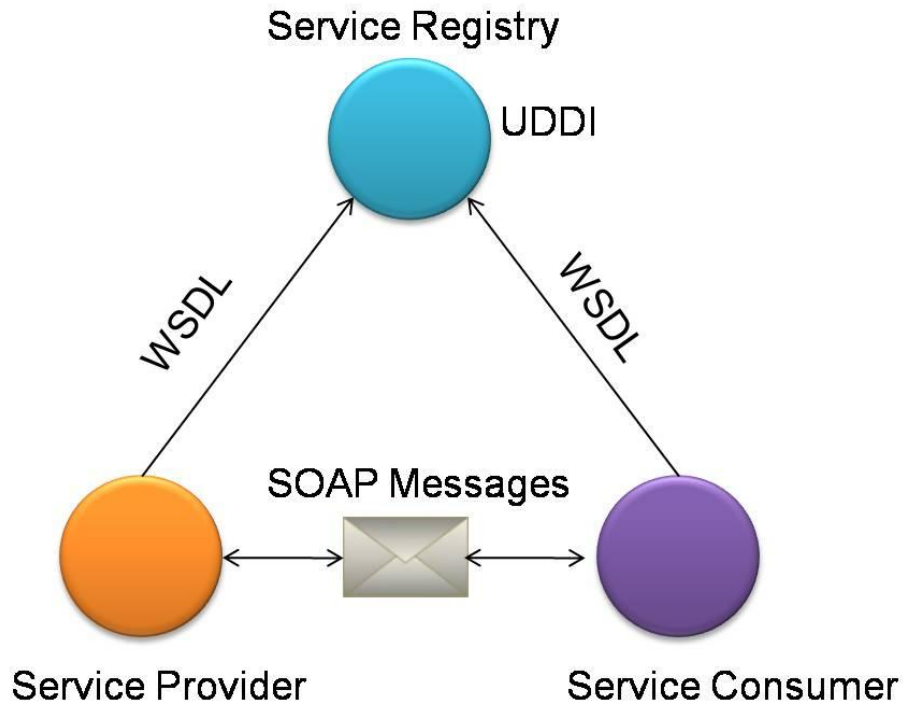


Fig 2.2 Web Services Interactions

### 2.1.1 WSDL

The Web Services Description Language (WSDL) is an XML-based language that provides a model for describing Web Services.

The WSDL defines services as collections of network endpoints, or ports. The WSDL specification provides an XML format for documents for this purpose [7,8,9]. The abstract definitions of ports and messages are separated from their concrete use or instance, allowing the reuse of these definitions. A port is defined by associating a network address with a reusable binding, and a collection of ports defines a service. Messages are abstract descriptions of the data being exchanged, and port types are abstract collections of supported operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding, where the operations and messages are then bound to a concrete network protocol and message

format. In this way, WSDL describes the public interface to the Web Services as shown in figure 2.3.

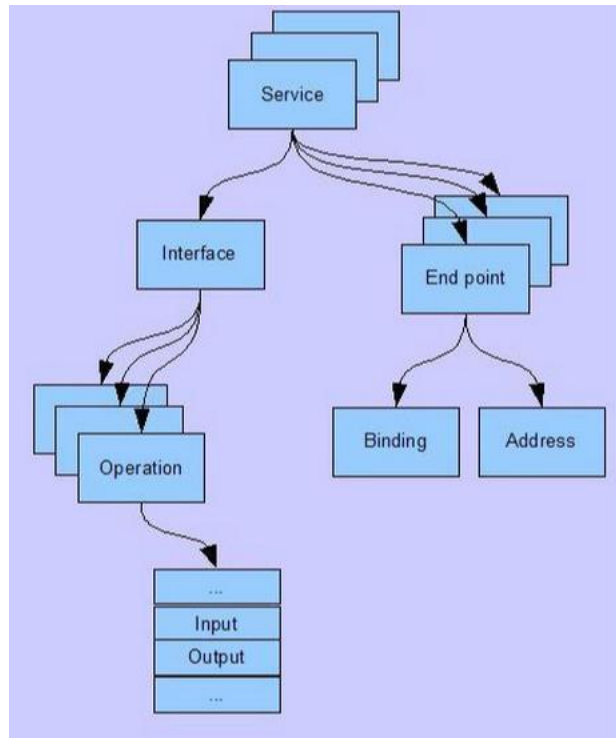


Fig 2.3 WSDL structure[9]

### 2.1.2 UDDI

Universal Description, Discovery and Integration (UDDI) is a directory service where businesses can register and search for Web Services. UDDI is a platform-independent framework for describing services, discovering businesses, and integrating business services by using the Internet. UDDI uses World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF), Internet standards such as XML, HTTP, and DNS protocols. UDDI uses WSDL to describe interfaces to Web Services. For example, if the industry published an UDDI standard for flight rate checking and reservation, airlines could register their services into an UDDI directory. Travel agencies could then search the UDDI directory to find the airline's reservation interface.



When the interface is found, the travel agency can communicate with the service immediately because it uses a well-defined reservation interface.

### **2.1.3 SOAP**

SOAP is a protocol used to communicate between different applications and to exchange information in a Web Services. A SOAP document is written in XML and is language and OS independent.

A SOAP message contains the following elements

1. Envelope
2. Header
3. Body
4. Fault

#### **Envelope**

Envelop is the most important part of a SOAP message. This part distinguishes the SOAP document from other normal XML documents, i.e. the envelope part identifies the XML document as a SOAP message. [10,11]

#### **Header**

The header part of a SOAP message is optional and can be included when required.

#### **Body**

The central part of a SOAP message is the BODY. It contains the actual function of the request i.e. it contains the message that is to be executed to get the desired output.

#### **Fault**

The fault element in a SOAP message is an optional element which contains errors of the SOAP message.

## The HTTP Protocol for SOAP

HTTP communicates over TCP/IP. An HTTP client connects to an HTTP server using TCP. After establishing a connection, the client can send an HTTP request message to the server. WSDL document for this Web Service is given in APPENDIX B-2

### SOAP Request

```
"POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
<m:GetStockPrice>
<m:StockName>IBM</m:StockName>
</m:GetStockPrice>
</soap:Body>

</soap:Envelope>"
```

## SOAP Response

```
"HTTP/1.1                200                OK
Content-Type:  application/soap+xml; charset=utf-8
Content-Length:                nnn

<?xml                    version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-
envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">

<soap:Body  xmlns:m="http://www.example.org/stock">
                <m:GetStockPriceResponse>
                        <m:Price>34.5</m:Price>
                </m:GetStockPriceResponse>

</soap:Body>

</soap:Envelope>"
```

## 2.2 Benefits of Web Services

**Exposing the function on to network:** A Web Service is a unit of managed code that can be remotely invoked using HTTP, i.e., it can be activated using HTTP requests [12]. Hence, Web Services allows you to expose the functionality of your existing code over the network. Once it is exposed on the network, other application can use the functionality of your program.

**Connecting Different Applications:** Web Services allows different applications to talk to each other and share data and services among themselves. Other applications can also use the services of the Web Services. For example, VB or .NET application can talk to java Web Services and vice versa. Thus, Web Services is used to make the application platform and technology independent.

**Standardized Protocol:** Web Services uses standardized industry standard protocol for the communication. All the four layers (Service Transport, XML Messaging, Service Description and Service Discovery layers) use the well defined protocol in the Web Services protocol stack. This standardization of protocol stack gives the business many advantages like wide range of choices, reduction in the cost due to competition and increase in the quality.

**Low Cost of communication:** Web Services uses SOAP over HTTP protocol for the communication, so the existing low cost internet can be used for implementing Web Services. This solution is much less costly compared to proprietary solutions like EDI/B2B.

**Support for Other communication means:** Besides SOAP over HTTP, Web Services can also be implemented on other reliable transport mechanisms. Hence, it gives flexibility to use the communication means of requirement and choice. For example, Web Services can also be implemented using ftp protocol (Web Services over FTP).

**Loosely Coupled Applications:** Web Services are self-describing software modules which encapsulates discrete functionality. Web Services are accessible via standard Internet communication protocols like XML and SOAP. These Web Services can be developed in any technology (like c++, Java, .NET, PHP, Perl etc.) and any application or Web Services can access these services. So, the Web Services are loosely coupled applications and can be used by applications developed in any technology. For example, Web Services developed using Java technologies can be used in VB or .NET applications.

**Web Services sharing:** These days due to complexity of the business, organizations are using different technologies like EAI, EDI, B2B, Portals etc. for distributing computing. Web Services supports all these technologies, thus helping the business to use existing investments in other technologies [9].

**Reusable Components:** Any independent function or operation can be made available to the REST of the world in the form of a Web Services. Exposing the frequently used operations as Web Services can increase the reusability. For example, there is a Web Service for currency conversion which can be used by any application on the web [10].

**Software Integration:** Since the Web Services is based on XML and is not specific to any language or platform, any software implemented on any platform can interact with each other. This is also an excellent means of integrating with already existing software.

### **2.3 Literature Survey on QoS elements - ontology, classification**

In the paper ‘A taxonomy and classification of web service QoS elements’ quality metrics are grouped into different QoS categories, namely performance, correctness, security, reputation, standards compliance and monetary. The performance QoS category of web service represents the parameters affecting the execution of a service. How fast the service request is accomplished is determined by this category. The major QoS elements under this category are availability, accessibility, throughput, response time, servability, capacity, scalability, and reliability. Correctness is defined as the probability of the data provided by the consumer being incorrect and incomplete. In such a situation, it is crucial that the service doesn't fail, and is handled effectively. It is expected to generate error report or process request with the same input data. The scenario of handling an erroneous input correctly by a service is essential to the selection process. The robustness and accuracy of quality metrics are the two main elements in this category. Security is one of the important attributes for message content integrity. Reputation refers to rating, trust, and reputation of a service. Standards compliance is a measure of extent to which the service complies with standards. Monetary elements are price, taxes, validity, etc. The ontologies of these QoS elements are explained diagrammatically.[11]

- 1. Availability** is the likelihood of a service to reply to client requests and is considered as an incrementing element. Classically, Availability is contradictory to capacity and analogous to reliability a bit. There are two subclasses for availability. Uptime, is the time period during which the service is functioning constantly without breakdown. MTTR, is the mean time to repair, denoting the mean time for reinstating an unsuccessful service.
- 2. Reliability** is the probability that a service can be used effectively. It comprises of Fault Rate: the rate of invocation failure; MTBF: mean time between failures; Consistency: the failure rate's lack of variability; Recoverability: how good is the recovery of a failed service; Failover: whether the failed resources are utilized by a service, and how promptly; Disaster Resilience: how good the ordinary and manual disasters are resisted by the service.
- 3. Capacity** is the limit on the number of requests a service can deal with. When a service is operated beyond its capacity, its availability and reliability are negatively affected.
- 4. Economic** captures the economic conditions of using the service and Cost is a key economic attribute.
- 5. Interoperability** evaluates the interoperation performance of a service, whether the service is compliant with a general standard or the specific version of standards.
- 6. Performance** expresses the functional performance from users' perspective. The main aspects are Throughput: the rate of successful service request completion) and Response Time: the duration from the request to getting a response from the service.

7. **Security** represents the safety measures of a service, comprising Audit Ability: auditable capacity is maintained by the service; Authentication: means whether the unidentified requesters are accepted by the service; Encryption: represents the category and strong point of encryption technology; Non Repudiation: means whether the service could be denied if already used by the requester.
8. **Trustworthiness** mainly depends on users experiences of using it and evaluates the credibility of user reports.
9. **Scalability** defines whether the service capacity can improve as the consumer's requirement.
10. **Integrity** is an evaluation of capability of a service to resist illegitimate right to use and protect totality of its information.
11. **Stability** is the rate of modification of the constraints of a service.
12. **Robustness** measures resilience to imperfect input and incorrect Web Services composition. This ontology specifies domain independent quality semantic and should be completed by a domain-specific lower ontology, which can both meet the users' satisfaction and increase the providers' gains.

The Web Services selection problem has been studied mainly in business process and e-science area. In service-oriented architecture (SOA) environment, end-users have to achieve two phases to invoke Web Services. Firstly, fulfill the objective service description, which usually presents as a WSDL document, and query it in the UDDI registry. A set of functionally equivalent services which does the same function but differs from nonfunctional characters, is expected to result from this query. Secondly, select the service which enables the best QoS guaranteed from the set and invoke that. The process of selecting the best QoS guaranteed Web Services from the other Web

Services of same functionality is known as service mining. Service mining is a solution to service overload. The following papers analyze the process of service mining using different techniques.

In the paper "Importance Levels of QoS Elements in Selection of Information Delivery Web Services" by Godse and Mulik, they have identified twenty eight QoS elements which can be considered for any web service. But, out of these quality metrics, which are all relatively more important, QoS metrics for the web service selection is identified by the process of survey among practitioners. In this paper, they have tabulated the responses received and the table shows the importance level on a scale of 1 to 5 by taking weighted average of response for each QoS metric which helps in the process of web service selection.[13]

## **2.4 Literature Survey on QoS based discovery**

Al-Masri, E.; Mahmoud, Qusay H., in the paper "Discovering the best Web Services: A neural network-based solution", discussed about discovering Web Services of interest based on the quality attributes using back propagation method [14]. To achieve this task, they have used the publicly available QoS for Web Services (QWS) Dataset as inputs for the neural network. The feature set is mainly dependent on several factors such as response time, throughput, availability, compliance, among others. The use of neural networks provides a way to optimize the selection of the best available Web Services. The proposed solution provides an effective discovery mechanism for finding the high quality Web Services based on non-functional properties, but it is observed that the back propagation neural network takes a long time during the training mode due to the large data size which could become an issue when implementing such system in real-time manner. In addition, the ability of the system configuration to quickly adapt to current



data being fed into it might become infeasible since the proposed method defined the number of hidden nodes prior in the training mode. So the above method is suitable for off line discovery of Web Services based on QoS.

Xiaopeng Deng; Chunxiao Xing in his paper, "QoS-oriented optimization model for Web Services selection from a Web Services group", a lightweight QoS model which includes four quality attributes namely functionality satisfaction degree, performance satisfaction degree, cost satisfaction degree and trust satisfaction is built up [15]. Then, the set of the QoS vectors are generated from the the QoS parameters of the Web services in the group. Then the QoS vectors are clustered by hierarchical merging clustering approach. But the problem with this approach is, it aims at only four QoS properties, it could have been multi-dimension QoS, as there are many attributes available as QoS parameters for a Web Services, which are all concerns for the Web Services client.

Badr, Y.; Abraham, A.; Biennier, F.; Grosan, C. discussed about a simple Web Services selection scheme based on user's requirement of the various non-functional properties and interaction with the system. The proposed framework utilizes user preferences as an additional input to the selection engine and the system ranks the available services based on the requirement [16]. The disadvantage with this method is that instead of specifying the real value for the non-functional properties, the user has to specify the importance of the different attributes as well.

## **2.5 Literature Survey on QoS based composition techniques**

In this "Policy-based Web Services composition," paper by Soon Ae Chun; Atluri, V.; Adam, N.R., the inter-organizational process is modeled as a composite Web Services, a workflow called service flow [17]. Each participating organization has a set of services to offer, called component (or atomic) services. A composite service is made up of these component services from each organization. However, service composition first requires to select the services that are compatible. They consider three levels of service

compatibility in order to make sensible compositions - syntactic, semantic and policy level compatibilities. Their approach to Web Services composition is based on building the knowledge of policy rules using OWL, DAML-S, RuleML and RDF standards. In this they have not discussed about how to handle conflicting rules of different participating organization that are distributed and heterogeneous. Also, how to derive at policy rules from the organization diverse documents and how to maintain the updates of the organizations rules are not discussed. The ontology for user policy specification needs to match for correct identification of applicable rules. The ontology interoperability between organizations must also be developed.

## **2.6 Literature Survey on QoS of REST and SOAP Web Services**

“Latencies of Service Invocation and Processing of the REST and SOAP Web Services Interfaces.”, studies the latencies experienced by the Web Services client invoking a proprietary multimedia messaging service with both REST and SOAP Web Services [18]. This study has been made utilizing XML, JSON, MTOM/XOP and Google Protostuff message and message content encodings. By using the same underlying service logic and implementation, they were able to clearly measure and make distinctions between these two service access styles in detail to analyze the service invocation complexities and performance penalties of each solution in detail. The measured details included service request building and invocation, request parsing, request and response transmission, and execution of the service logic. However, the effects of the measured details need to be tested on a more realistic networking environment. Further, the above study does not pay attention to databases in the backend which is not the case present in most of the applications that use Web Services. This comparison requires a challenging and careful synchronization mechanism of the time stamping instrumentation clocks, if using two separate networked computers.

In real time scenario, for example when the web service client is looking for weather forecast web service, there are many services available that satisfies the functional

requirements. but the services may vary in terms of their relative quality metrics. In order to identify the one that meets the functional as well as QoS requirements of the web service client, one has to go for web service selection procedure. And currently there are many solutions to provide the web service client with their QoS values but in this thesis data mining algorithm is applied on these collected QoS data and interesting patterns are given as the outcome of the mining process to the web service client. In the coming chapter an extended service oriented architecture is designed for web service selection based on QoS values.

## **CHAPTER 3**

### **Implementation of Agent for WSS using Data mining technique.**

Web Services play a major role in developing business applications in this internet era. As there are many Web Services available for a particular function, finding appropriate Web Services based on our criteria is becoming a tedious process. Currently, finding the exact Web Services with the keyword based search method of UDDI registry doesn't return the Web Services of user interest at all times. An agent based method for Web Services selection, from the information that is given in the modified WSDL file by the Web Services provider is proposed in this chapter. For analyzing the QoS of web services, the already available repository of quality metrics is used in most of the previous research works. Here, the WSDL file is modified to have embedded QoS metrics in it, from where the agent can easily extract the data and maintain database of QoS metrics of all web services in that functional domain. Data mining is done on those data that are collected from WSDL files and also the feedback taken from the Web Services users, by the agent to discover some interesting patterns for further users of the Web Services. The rest of this chapter is organized as follows - firstly, how Web Services are described using WSDL with QoS extension. Further, it explains about how the agent applies data mining into the QoS data collected and produces output for the client who is interested in the particular functional domain.

#### **3.1 Agent Based Architecture.**

Service Oriented Architecture (SOA) service descriptions [19] are saved in a repository that behaves in a similar manner to a telephone directory. The required applications can be searched in the UDDI directory. However, keyword based searching mechanisms for finding web-services does not satisfy the rapidly growing need for QoS (Quality of Service) based searching [20,21]. Many Web Services often provide the same

functionalities to the user, but may differ in their non-functional properties. Considering both the functional and non-functional type of properties during the discovery process, enables us in selecting a reliable service.

Nowadays, the Service providers as well as the clients pay a lot of importance to the QoS values of web-services [22]. According to a client’s view point, a quality based service discovery involves a multi-criteria decision making mechanism which needs the knowledge of both the working of the service and its QoS parameters. However, many of the service requesters are not qualified to test the services for their QoS characteristics themselves and simply accept the information given by the service provider. Based on the above need, a WS-QoS Agent based architecture is proposed to facilitate the discovery of appropriate Web Services with the specified QoS values.

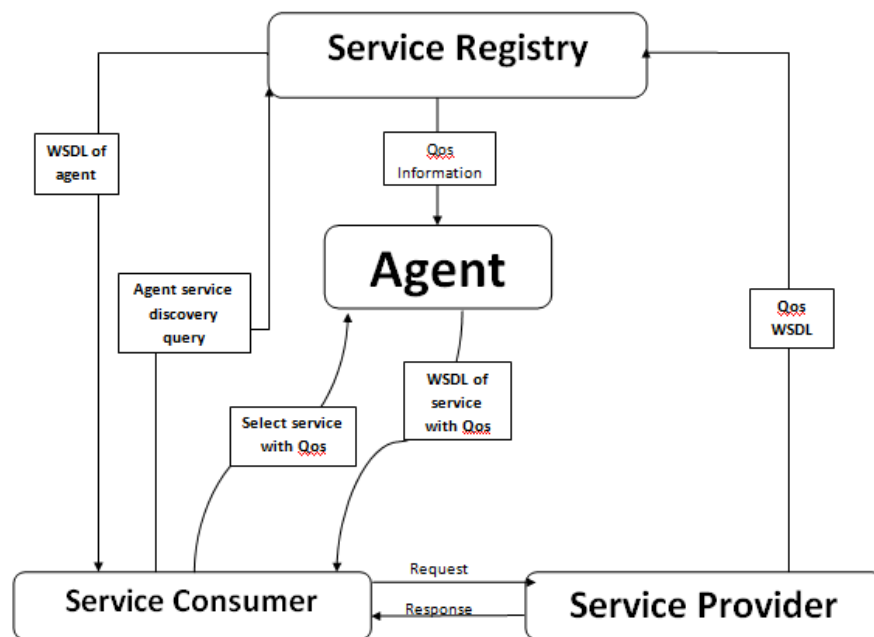


Fig 3.1 Extended SOA for WSS

In the above figure 3.1, there are blocks which are very familiar, namely service registry, service consumer and service provider. One new block involved is Agent, which helps the service consumers to select best Web Services in terms of QoS attributes. Accordingly, the Service Consumer will search to get Agent Service for the functional domain he is sending Agent Service discovery query for as mentioned in the above figure

3.1. After he gets the WSDL of agent service, client contacts the agent service to select service with QoS.

The agent service will reply to the client with the best available web service in terms of QoS. This extended SOA is implemented using SOAP Web Service in the following chapters.

### **3.2 Describing Web Services with QoS**

Functionality in Web Services is defined in WSDL, an XML standard which specifies the operations that the Web Services can perform. The information provided in the WSDL covers all the data needed to invoke the Web Services itself but lacks specifying the Quality of Service. On the other hand, those services may be registered in a UDDI. A UDDI is a library of Web Services which are used to discover them; subsequently the client can select the one which suits him the most in terms of functionality. Nevertheless, discovering Web Services are founded on using keyword-based search techniques, which may not return suitable results to clients' requirements since they might not reach the desired QoS[23].

In order to fulfill these problems there are several proposals to extend WSDL Standards with QoS information. Notice that this information might not represent the real Quality values for the set of attributes which may vary on run time, but the one claimed by the provider instead. Therefore, the provider's trustworthiness is another important part to take into account. Web Service QoS certifier plays major role in checking the trustworthiness of the provider.

### **3.3 WSDL extension with QoS**

WSDL (Web Services Description Language) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure oriented information.

In a WSDL document, the functionality of a service is specified as it defines the methods of the service and how they are invoked. Nevertheless, it lacks support for specifying nonfunctional properties such as QoS. In order to resolve this problem, several proposals of extending WSDL with Quality of Service information has been presented. [22]

Several works needs WSDL extension with QoS as a basis for their contributions. Most of these contributions use a simple WSDL extension. For instance, YunHee Kang proposes the use of annotations[22], which is currently supported by WSDL standard, to describe WSDL with QoS information. An annotation is similar to a comment, and is used in WSDL mainly to embed the documentation.

As semantic Web Services are increasing on popularity, a new kind of annotations has emerged: SAWSDL (Semantic Annotation for WSDL and XML Schema). SAWSDL is an extension of WSDL that provides semantic annotations through the use of ontologies [5]. This WSDL extension is a W3C Recommendation since 2007.

Using SAWSDL, a service provider can advertise his Web Services linking its components with an ontology written in OWL. However, because of the novelty of SAWSDL, most of approaches in QoS extensions are still focused in WSDL Using this structure, WSQDL expresses a way to define a QoS attribute for a Web Services as a complex Type in the WSDL of the Web Service.

### **3.4 Trustworthiness of claimed QoS**

Once the Quality of Service is defined by the service provider, there's an imperative need to check that the QoS claimed is consistent with the real Quality of the Service. This is especially significant for those dynamic quality attributes whose values may vary on run time.

In order to achieve this point, monitoring or testing is needed in order to compare the obtained values with the claimed ones. One of the first and most significant works in this

area is the addition of a new actor in the Web Services discovery [24]. This new actor is known as the Web Services QoS Certifier and is responsible to validate that the quality of Service advertised by the provider is the actual Quality of Service.

In this approach, QoS information is stored and managed by the agent as an intermediary between the client and the service provider. The Agent is responsible to manage and provide to the client the QoS for Web Services. For that purpose, the agent might use an extended WSDL or store the QoS in an internal component of the framework. It may also retrieve the list of Web Services from the UDDI or implement its own Web Services registry. This kind of approach is most used in the literature of Web Services discovery.

Most of the agents' proposals include, apart from a Web Services registry with QoS information, a monitor to obtain those dynamic quality attributes. That is, the agent would be responsible by monitoring the registered services and store those dynamic attributes of the Web Services, whereas static quality attributes such as price should be stored declaratively. Another component of the agent is the module that implements the ranking algorithm for Web Services selection taking into account the QoS of the different Web Services. Obviously, the service user should perform the discovery over the agent instead of using the UDDI directly.

Actually, Agents are working for only one particular functional domain Web Services. For example, agent Web Services can be made for web content search engines, Travel assistance, flower delivering, and e-commerce related search and purchasing activities. In this, Weather Forecast Services are taken as an example and appropriate agent is designed for that. So, there are ten Web Services created for weather forecasting service, where in their WSDL file is extended to have QoS attribute values.



### 3.5 QoS attributes that are considered for WSS

**Availability:** It is the degree of readiness of the service for immediate consumption.

**Response Time:** Time elapsed from the Submission of a request to the time the response is received.

**Price:** This represents price that a consumer of a Web Services must pay in order to use the Web Services.

**Security:** This is ability of the Web Services to provide security mechanisms like encryption, authentication, and access control.

**Latency:** It is the time delay involved in start servicing the request of the client.

**Reliability** is the probability that a service can be used effectively.

**Scalability:** number of throughput at given interval.

**Successability:** the extent to which Web services yield successful results over request messages. Successability can be calculated as the number of successful response messages over the number of request messages.

All these seven QoS attributes taken as quality metric for the Web Services. The values for the attributes can be from 1 to 5. Five specifying the most preferred characteristic e.g. if the response time is less, it is given value 5 and if the response time is too high it is given value 1 for a particular Web Service. These values that are specified in the WSDL file by the Web Services provider are extracted by the agent. There are going to be 4 classes in our classification of Web Services namely platinum, gold, silver and bronze. The WSDL file from ws1 is shown in figure 3.2. If all the attributes are having values 5, it will come under platinum class. If all the values are 4 it is gold, if the values are between 3 and 2 it will come silver and the value 1 in all attribute make the Web Services to come in class bronze. Part of Extended WSDL is shown in the figure 3.2. As

mentioned in the figure below, the availability 5 means the service is too good in terms of availability, same in the case of security, latency, response time, price and accessibility.

```
<Qos>
<Availability>5</Availability>
<Latency>5</Latency>
<ResponseTime> 5</ResponseTime>
<Security>5</Security>
<Price>5</Price>
<Accessibility>5</Accessibility>
<Reliability>5</Reliability>
<Authentication>no</Authentication>
</Qos>
```

Fig 3.2 Part of WSDL file.

The Agent Web Services extract these details from the WSDL link of all Web Services available for the particular functional domain, in our case weather forecast, and maintain a table. All the WSDL files provided by the Web Services provider ws1, ws2...ws10 have values for the QoS attributes that are selected. This quality metric information is extracted from the WSDL file by the agent shown in figure3.3. Web Services client, after using the service, logs his comments and feedbacks to this agent.

Service Name	Availability	Latency	Response Time	Security
Weather1Service	5	5	5	5
Weather2Service	5	4	5	5
Weather3Service	3	4	4	3
Weather4Service	2	3	4	3
Weather5Service	3	4	4	3
Weather6Service	3	1	2	3
Weather7Service	3	2	2	1
Weather8Service	3	2	1	1
Weather9Service	3	2	1	2
Weather10Service	3	2	2	2

Fig 3.3 Table maintained by the agent

WEKA is applied on this data, and the preprocess tool in WEKA produces output which can be used by the classifier. Figure 3.4 shows the J48 classification obtained in this way.

With the help of this pattern, any new instance which is to be classified can be put into the class it belongs to. In WEKA, simple K means clustering is selected to produce clusters shown in figure 3.5. From right clicking on the instance, which instance or Web Services is coming under what class can be found as shown in figure 3.6. Hence, whenever client wants to call a Web Service for a particular domain, he can call the agent for that and the agent will infer about quality of the all the Web Services available for weather forecast.

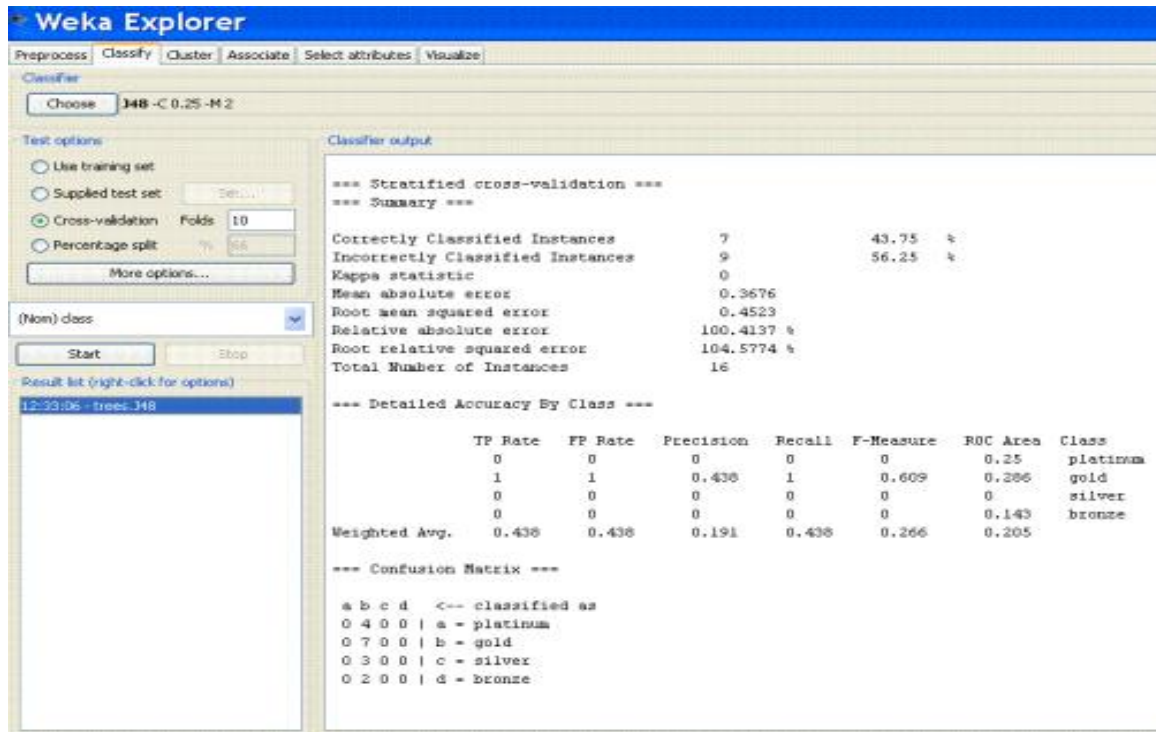


Fig 3.4 Classification output

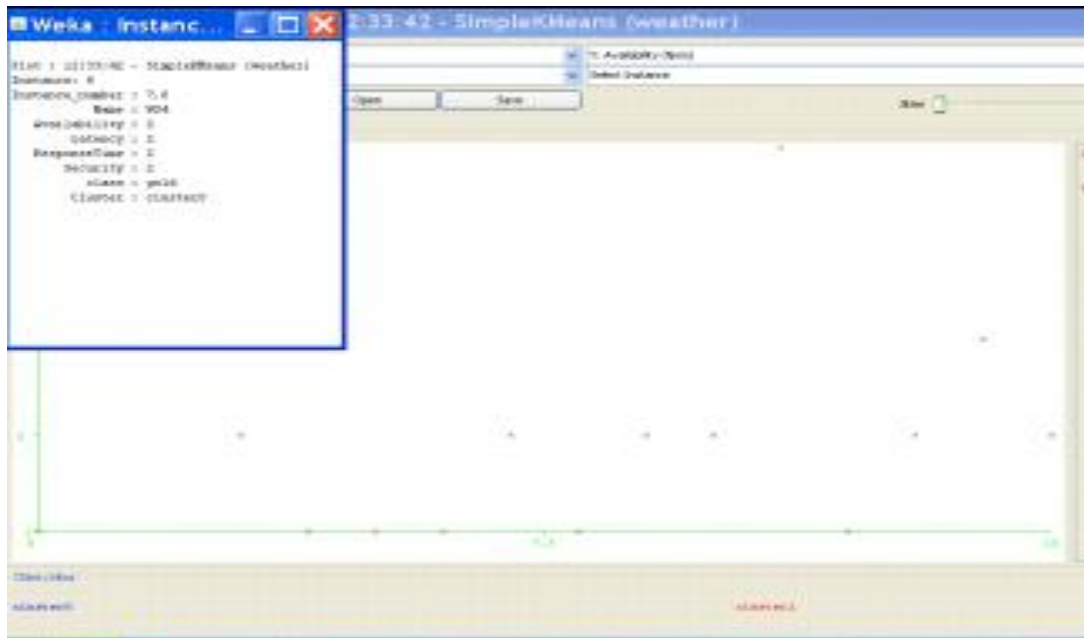


Fig 3.5 Cluster output

### 3.6 Feedback from Web Services client

The QoS attributes that are discussed here like security, speed, latency, response time that are provided by the service provider, can be validated if the client accepts to give feedback on them. So, option is given for the client to enter his views about attributes like response time, latency, throughput and speed. He is guided on how to evaluate the Web Services by giving values 1 to 5. The feedback that the user wants to give about these attributes is collected in a user log access file. Then data mining can be done on these, to extract the Web Service information that satisfies the QoS demands of the future user.

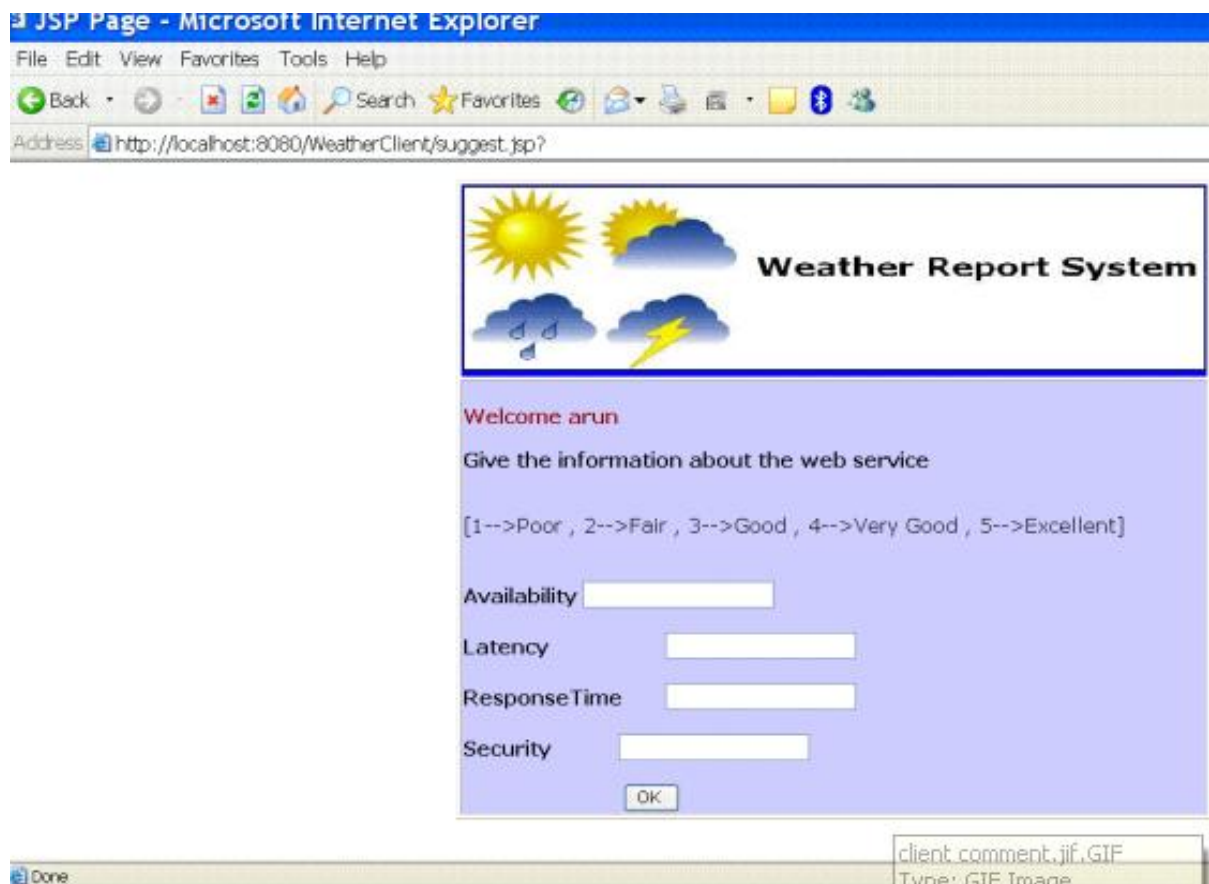


Fig 3.6 client feedback form

The database maintained by the agent is updated with the ones from the client's feedback forms.

### **3.7 Summary**

In this chapter the service mining of Web Services is done using the QoS data provided by the service provider whose validity cannot be confirmed. Eyhab Al-Masri counted and statistically analyzed the attribute in real time and arrived at a dataset [24,25] which is having actual values for almost all attributes of importance and provided a great path for the researchers who are into Web Services selection based on QoS attributes Further in this thesis, improved data mining algorithm is applied on these data sets for the classification of Web Services based on QoS attributes.

## CHAPTER 4

### **WSS using Entropy Discretization Method.**

In conventional technique, service requester is not capable to explore the Web Services in accordance to the non-functional attributes. Non-functional attributes [26, 27] such as response time, availability, throughput, etc are referred by Quality of Service (QoS) of Web Services. QoS based service selection includes, the Web Service selection based on requester's preferences on QoS constraints in that particular functional domain of Web Services, in comparison to the usual selection procedure. Web Services selection process using QoS provides a selection of service on requested quality of service. The limitations in the previous procedure is services are invoked by the functional configurations, in the conventional procedure of Web Services selection [28, 29, and 30]. Therefore, the requester could not make out the characteristics of the provided Web Services on web. In addition, UDDI does not offer standards of QoS for publishing and querying Web Services.

There are various methods proposed earlier for Web Services selection based on QoS such as based on, QoS constraints, QoS ontology, Fuzzy approach, QoS agent and various other methods [31, 32]. In the most of the previous work in this area, only few attributes of quality metrics is considered and no practical solution is arrived to handle multiple QoS constraints which will naturally be a very large dataset. In this proposed method, Web Services are classified under four classes like poor, average, good and excellent, according to the values of their non-functional attributes. In order to classify the Web Services, Entropy Discretization algorithm is used since other algorithms require only discrete or nominal values. This involves dividing the possible values into sub ranges called buckets or bins. Using this, a tree can be built that is used to classify any new Web Services according to its non-functional attributes to one of the four classes. All the available Web Services that match the functional request could be listed to the service requester with the class that they belong to. These services could also be listed according to the provided values of quality of service attributes and matching them to the available services and then notifying requester the classification of the service alongside [33, 34, and 35].

## 4.1 About the Dataset

The updated QWS (QoS of Web Services) Dataset Version 2.0 includes a set of 5,507 Web Services and their QWS measurements that were conducted in March 2008[3,10]. Each row in this dataset represents a Web Services and its corresponding nine QWS measurements (separated by commas). The first nine elements are QWS metrics that were measured using multiple Web Services benchmark tools over a six-day period. The QWS values represent averages of the measurements collected during that period. The last two parameters represent the service name and reference to the WSDL document. The input data set used for the decision tree induction and QoS classification is the “QWS Dataset” [10]. The QWS dataset consists of data from over 5000 Web Services out of which the public dataset consists of a random 365 Web Services which have been chosen and nine QWS(Quality of Web Services) attributes have been measured. Each Web Services was tested for over duration of over ten-minutes for three successive days. The most important objective of this dataset is to put forward a foundation for Web Services researchers. Each instance in the dataset matches up to an accessible Web Services execution available on the open Web nowadays. The non-functional attribute values with units are shown in Table 4.1

Table 4.1 Non-functional attribute values with units

ID	Parameter Name	Description	Units
1	Response Time	Time taken to send a request and receive a response	ms
2	Availability	Number of successful invocations/total invocations	%
3	Throughput	Total Number of invocations for a given period of time	invokes/second
4	Successability	Number of response / number of request messages	%
5	Reliability	Ratio of the number of error messages to total messages	%
6	Compliance	The extent to which a WSDL document follows <a href="#">WSDL specification</a>	%
7	Best Practices	The extent to which a Web service follows <a href="#">WS-I Basic Profile</a>	%
8	Latency	Time taken for the server to process a given request	ms
9	<a href="#">Documentation</a>	Measure of documentation (i.e. description tags) in WSDL	%
10	<a href="#">WsRF</a>	Web Service Relevancy Function: a rank for Web Service Quality	%
11	<a href="#">Service Classification</a>	Levels representing service offering qualities (1 through 4)	Classifier
12	Service Name	Name of the Web service	None
13	WSDL Address	Location of the Web Service Definition Language (WSDL) file on the Web	None



## 4.2 Data Normalization

Mostly, all of the quality of service constraints varies from one another in direction as well as in value range of the utility increments. There is no comparison between them. Therefore, calculation of the weighted average of quality of service constraints is not useful. Constraint values must be transformed such that they reflect the true value in a standard range and also providing the same incrementing direction. All attributes used here are normalized values. Let's say that raw value of constraint,  $Q$ , is denoted by  $q$ , threshold value is denoted by  $q_{th}$  and  $q_{min}$  denotes the minimum [36].

Data normalization of a constraint is calculated according to equation (1) if the effectiveness of it increases with the value of the constraint,  $q$ . Or else, equation (2) is applied.

$$Q' = \frac{(q - q_{min})}{(q_{max} - q_{min})}, \text{ if } q_{max} - q_{min} \neq 0 \quad (1)$$

$$Q = \frac{(q_{th} - q)}{(q_{th} - q_{min})}, \text{ if } q_{th} - q_{min} \neq 0 \quad (2)$$

## 4.3 WEB SERVICES RELEVANCY FUNCTION (WsRF)

WsRF is brought into play to evaluate the quality standing of a Web Services based on quality metrics. QWS parameters help decide which of the accessible Web Services is most excellent appropriate for a client search. Because of their importance, QWS attributes is selected as shown in Table 4.1(1 through 9).

A Web Service with the highest calculated value for WsRF is the most desirable and relevant for the client based on his/her preferences. Due to the fact that QWS parameters vary in units and magnitude, QWS metrics must be normalized to be able to compute the WsRF and perform QWS-based ranking. Normalization provides a more uniform distribution of QWS

measurements that have different units. In addition, normalization allows for weights or thresholds to be associated with QWS parameters and provides clients with effective ways to fine-tune and control QWS search criteria.

In order to allow for different circumstances, there is an apparent need to weight each factor relative to the importance or magnitude that it endows upon ranking Web Services based on QWS parameters. Each weight in this array represents the degree of importance or weight factor associated with specific QWS property values. The values of these weights are fractions in the sense that they range from 0 to 1. In addition, all weights must add up to 1.0. Each weight is proportional to the importance of a QWS parameter to the overall Web Services relevancy ranking. The larger the weight of a specific parameter, the more important that parameter is to the client and vice versa. Here to emphasis equal importance to equal weightage is given to all the attributes.

The importance level corresponding to each QWS varies since these properties vary in unit. Due to the fact that each QWS property chosen by clients has an associated unit that is different from other properties (i.e. response time in milliseconds, and throughput in invokes/second), it is mandatory to clarify that each weight represents a different degree of significance which must be optimized.

On the whole, final rank value is calculated using weighted sum of the quality of service constraints which were normalized, according to equation (3) in which V represents the rank value of the Web Service.

$$V = \sum_{i=0}^n W_i \times Q_i \tag{1}$$

## 4.4 Service Classification

The service classification characterizes the services into different classes based on their QoS attributes as shown below.

1. Excellent (High quality)
2. Good
3. Average
4. Poor (Low quality)

On the whole, Web Services that belong to a particular functional service group can be classified in the above said classes using their computed WsRF values. Using WsRF values found for every Web Services, classification format is applied to relate each Web Services to a particular service group. The classification can be useful to distinguish between ranges of services that offer the similar functionality. The implementation of this algorithm is done in the kernel Microsoft Windows XP, Professional Version 2002 Service Pack 3, v.5938 in Computer Intel(R) Core (TM) 2 CPU , 4400 @ 2.00 GHz, 2.00 GHz, 0.99 GB of RAM

The part of the dataset is shown in table 4.2.

Table 4.2 Input Dataset

Response Time	Availability	Throughput	Successability	Reliability	Classification
255.08	12	8.1	13	53	POOR
64.96	18	4.3	18	60	POOR
68.91	19	4.4	20	60	POOR
451	23	1.8	24	42	AVERAGE
136.94	26	3.1	26	67	POOR
542.87	26	4.4	26	53	POOR
382.71	27	5.7	28	73	POOR
501.79	28	4.8	28	73	POOR
316.07	32	1	32	60	POOR
214.62	32	2.3	32	53	AVERAGE
689.42	34	1.1	34	67	POOR
266.83	36	0.9	37	60	POOR
261	36	0.9	37	60	POOR
667.11	38	2.1	38	73	POOR
83.33	39	1.7	40	80	POOR
142	39	6.3	40	73	POOR

## 4.5 ENTROPY DISCRETIZATION METHOD

### 4.5.1 Introduction

Entropy is a measure often used in data mining algorithms to measure the disorder of a set of data. For a range of real values in which every point is associated with one of the two class labels, the distribution of the labels can have three main basic shapes as shown in figure 4.1.

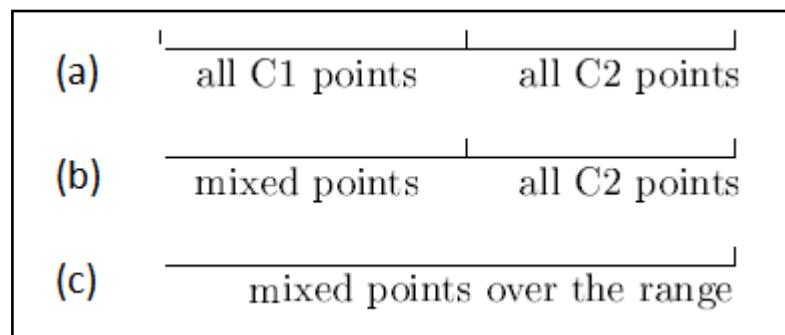


Fig 4.1 Distribution of range of points

- a) Big intervals in each containing the same class of points like C1 and C2
- b) Big intervals but not all of them containing the same class of points
- c) Class points randomly mixed over the range.

Using the middle point between the two classes, the entropy method partitions the range in case of Figure 4.1(a) into two intervals. The entropy of such a partition is 0. For the case of Figure 4.1(b), the method partitions the range in such a way that the right interval contains as many as C2 points as possible and contains as few C1 points as possible. This is to minimize the entropy of this feature. For the case of Figure 4.1(c), the method ignores the feature as mixed points over a range do not provide rules for reliable classification. That a range is partitioned into at least two intervals is called discretization. In general, ideally discriminating features (as shown in figure 4.1(a)), sub-optimal features (as shown in figure 4.1(b)), and those features with random class distributions can be effectively identified by the entropy-based discretization method.

#### 4.5 .2 ENTROPY

Initially, the range of a continuous variable, from a database sample, is divided into intervals which contain at least one case each. This is done after sorting on the variable values. At most, there would be  $m$  intervals ( $O(m)$ ) for  $m$  cases. This converts the continuous variable into a discrete one, with  $O(m)$  values [37].

Entropy, or information, is maximized when the frequency probability distribution has the maximum number of values. Since there is a discrete partition for every distinct value in the continuous distribution in the database, there is no information or entropy loss from the database sample.

The entropy of a discrete random variable  $X$  is defined as

$$\text{Entropy (t)} = -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t) \quad [1]$$

Information Gain, which calculates the reduction in entropy (*Gain in information*) that would result on splitting the data on an attribute, A.

$$\text{Gain (S,A)} = \text{Entropy (S)} - \sum_{v \in A} \frac{|S_v|}{|S|} \text{Entropy (S}_v)$$

Where v is a value of A, |S<sub>v</sub>| is the subset of instances of S where A takes the value v, and |S| is the number of instances

Using Entropy-based Discretization, classification of Web Services could be done easily. Using this classification the requester could choose the most suitable Web Services according to the functional requirements and his QoS preferences. Using Entropy-based Discretization a tree is obtained whose nodes belong to the QoS attributes. Tracing these nodes, leaf nodes can be reached which represent the classification of the Web Services into four classes.

In order to present the most suitable service to the service requester, the Quality of Service attributes are used, as they completely define the quality of a Web Service. In this section, a method is proposed to use Entropy-based Discretization in order classify Web Services into four classes, i.e. Poor, Good, Average and Excellent. Using this method a classifier tree is obtained. Using this tree, any new Web Services whose QoS submitted by client can be classified into one of the four classes mentioned earlier by tracing the tree according to their QoS attribute values.

There may be several features that are immaterial to the classification in data mining. Taking such features into consideration during classification leads to an increase in the dimensionality of the problem. It subsequently raises several computational difficulties and possibly introduces noise effect on the classification accuracy [38,39]. So, how to select important features among the many available for classification is a problem that has been attracting tremendous research effort previously and currently. Here there are only five attributes namely throughput, response time, successability, reliability and availability are considered.

This classifier consisting of a committee of *cascading* decision trees. Each tree is constructed by using one of the top-ranked features as its root node. In this example availability is taken as root node. The learning phase of this classifier is to construct a certain number of trees[90].

The following steps are used to construct the tree.

Suppose  $n$  is the number of features describe a given data. To construct  $K$  ( $K \leq n$ ) number of trees, the following steps are used

Step 1: Use gain ratios to rank all the features into an ordered list with the best feature at the first position.

Step 2:  $i=1$

Step 3: Use the  $i^{\text{th}}$  feature as root node to construct the  $i^{\text{th}}$  tree.

Step 4: Increase  $i$  by 1 and goto Step 3, until  $i = K$

### **4.5 3 Selecting the Best Split**

The idea is to first rank all individual features according to their entropy value, then use the average entropy value as a splitter to cut off all the lower ranked features. The steps are as follows:

1. Rank all features into an ascending order according to their entropy values,
2. Remove those features that are ignored (not discretized) by the entropy-discretization method,
3. Calculate the average of the entropy values of the remaining features, and
4. Select those features that have smaller entropy values than the average for classification

#### 4.5.4 Applying entropy-based Discretization on Quality of Web Services Dataset

In all services, count the number of services that are under each category like poor, good average and excellent, then,

Number of services for each classification.

Poor (P) = 35

Average (A) = 159

Good (G) = 96

Excellent (C) = 10

Table 4.3 Sorted Dataset

Sl. No	Response Time	Availability	Throughput	Successability	Reliability	Classification
1	255.08	12	8.1	13	53	POOR
2	64.96	18	4.3	18	60	POOR
3	68.91	19	4.4	20	60	POOR
4	451	23	1.8	24	42	AVERAGE
5	136.94	26	3.1	26	67	POOR
6	542.87	26	4.4	26	53	POOR
7	382.71	27	5.7	28	73	POOR
8	501.79	28	4.8	28	73	POOR
9	316.07	32	1	32	60	POOR

Entropy (E) = 1.53665

Sorting the table (part of the table is given in table 4.3) in ascending order according to Availability column,

Options for splitting points (T): 55 and 79 (for availability attribute)



First, choosing T as 55:

For S1 ( $A_v \leq 55$ ), it comes like,

$$P = 24$$

$$A = 3$$

$$E(S1) = 0.50325$$

For S2 ( $A_v > 55$ ), it becomes,

$$P = 11$$

$$A = 156$$

$$G = 96$$

$$C = 10$$

$$E(S2) = 1.35301$$

$$E(S|55) = 1.27653$$

$$\text{Gain}(55) = 0.26012$$

Now choosing T as 79,

For S1 ( $A_v \leq 79$ ),  $P = 32$

$$A = 58$$

$$E(S1) = 0.93894$$

For S2 ( $A_v > 79$ ),

$$P = 11$$

$$A = 156$$

$$G = 96$$

$$C = 10$$

$$E(S2) = 1.32086$$

$$E(S|79) = 1.20628$$

$$\text{Gain}(79) = 0.33037$$

Since Gain (79) > Gain (55), then choose splitting point as 79. Obtained incomplete tree (figure 4.2):

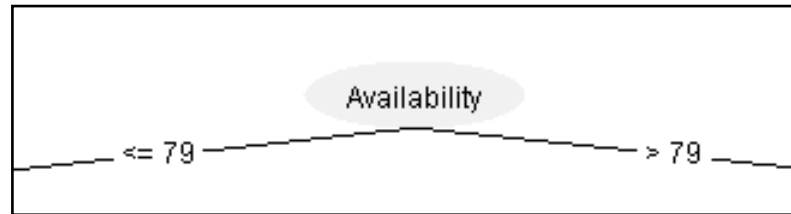


Fig 4.2 Partial Tree Constructed 1

For Availability  $\leq 79$ , from table:

For Availability  $\leq 52$ , Classification = Poor (27 / 3)

Now the incomplete tree looks as shown in figure 4.3

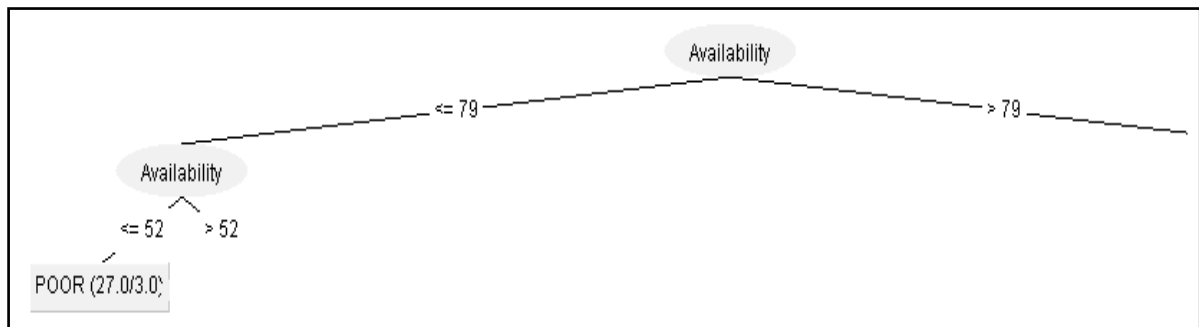


Fig 4.3 Partial Tree Constructed 2

In the remaining table for Availability > 52, then

$$P = 29$$

$$A = 34$$

$$E = 0.99545$$

Now, arrange the remaining table in ascending order, according to Response Time (RT).

Options for T: 825 and 1730

Choosing  $T = 825$ :

For  $S1$  ( $RT \leq 825$ ),

$$P = 2$$

$$A = 51$$

$$E(S1) = 0.23181$$

For  $S2$  ( $RT > 825$ ), then

$$P = 6$$

$$A = 51$$

$$E(S2) = 0.97095$$

$$E(S|825) = 0.34913$$

$$\text{Gain}(825) = 0.64632$$

Now choosing  $T = 1730$ ,

For  $S1$  ( $RT \leq 1730$ ),

$$P = 5$$

$$A = 55$$

$$E(S1) = 0.41382$$

For  $S2$  ( $RT > 1730$ ),

$$P = 3$$

$$E(S2) = 0$$

$$E(S|1730) = 0.39411$$

$$\text{Gain}(1730) = 0.60134$$

Since  $\text{Gain}(825) > \text{Gain}(1730)$ , then choose splitting point as  $RT = 825$ . Obtained incomplete tree (figure 4.4):

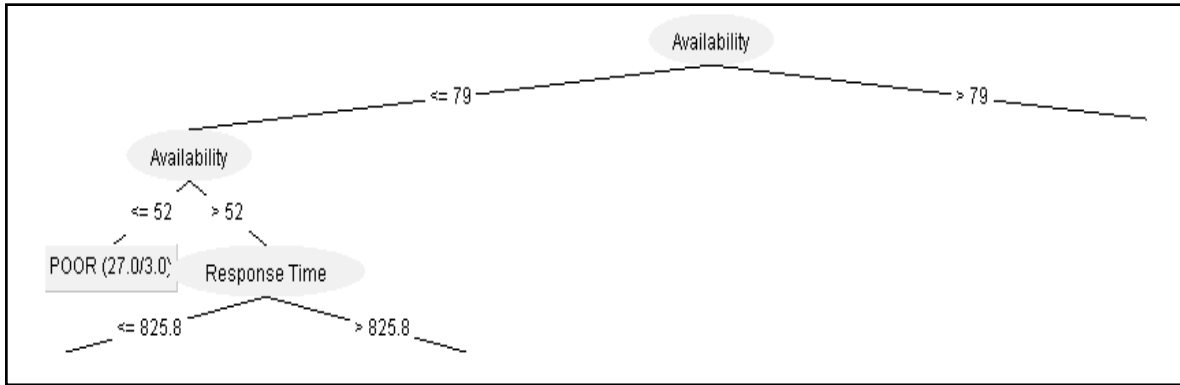


Fig 4.4 Partial Tree Constructed 3

For  $RT \leq 825$ , from the remaining table

For Reliability (Rel)  $\leq 80$  : Classification = Average (48)

For Rel  $> 80$ , from remaining table then

For Av  $\leq 64$  : Classification = Poor (2)

For Av  $> 64$  : Classification = Average (3)

For  $RT > 825$ , from the remaining table

For Rel  $> 78$  : Class = Poor (2)

For Rel  $\leq 78$ , from the remaining table

For  $RT \leq 1730$  : Classification = Average (5 / 1)

For  $RT > 1730$  : Classification = Poor (3)

Obtained incomplete tree (figure 4.5):

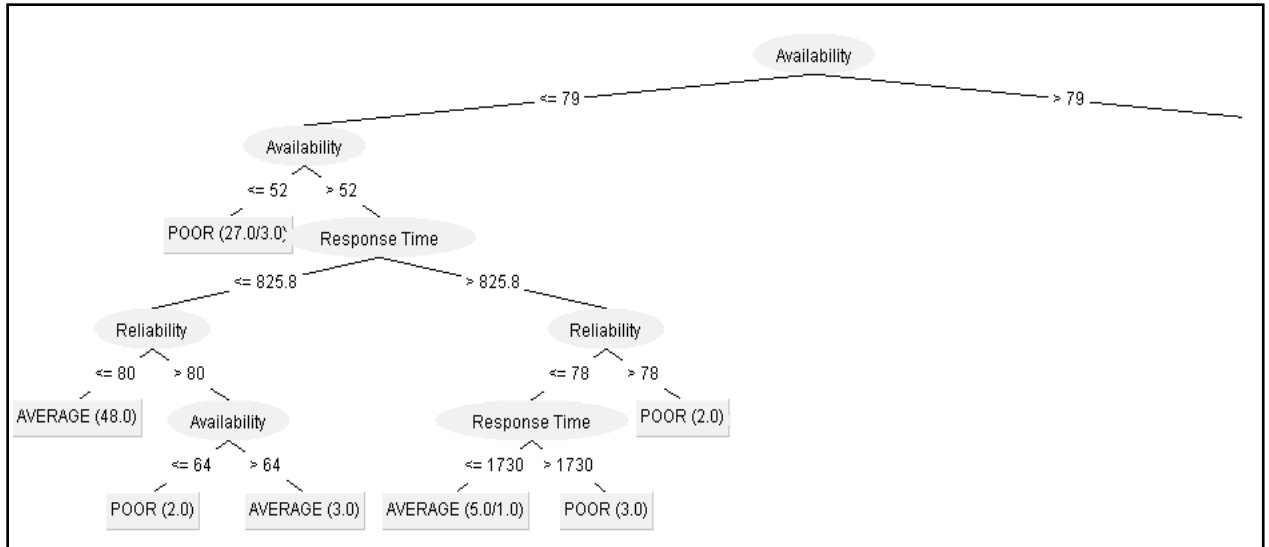


Fig 4.5 Partial Tree Constructed 4

For  $Av > 79$ , from the remaining table

$P = 11$

$A = 156$

$G = 96$

$C = 10$

$E = 1.32086$

Now, arrange the remaining table in ascending order according to Throughput ( $T_p$ ).

Options for T: 8.2 and 21.4

Choosing  $T = 8.2$ :

For  $S_1 (T_p \leq 8.2)$ ,

$P = 3$

$A = 68$

$G = 35$

$E (S_1) = 1.08427$

For  $S_2 (T_p > 8.2)$ ,

$A = 33$

$G = 61$

$$C = 10$$

$$E(S_2) = 1.3018$$

$$E(S|8.2) = 1.192$$

$$\text{Gain}(8.2) = 0.12886$$

Now choosing  $T = 21.4$ ,

For  $S_1$  ( $T_p \leq 21.4$ ),

$$P = 3$$

$$A = 100$$

$$G = 90$$

$$E(S_1) = 1.09811$$

For  $S_2$  ( $T_p > 21.4$ ),

$$A = 1$$

$$G = 6$$

$$C = 10$$

$$E(S_2) = 1.22104$$

$$E(S|21.4) = 1.10807$$

$$\text{Gain}(21.4) = 0.21279$$

Since  $\text{Gain}(8.2) < \text{Gain}(21.4)$ , then choose splitting point as  $T_p = 21.4$ . Obtained incomplete tree (figure 4.6):

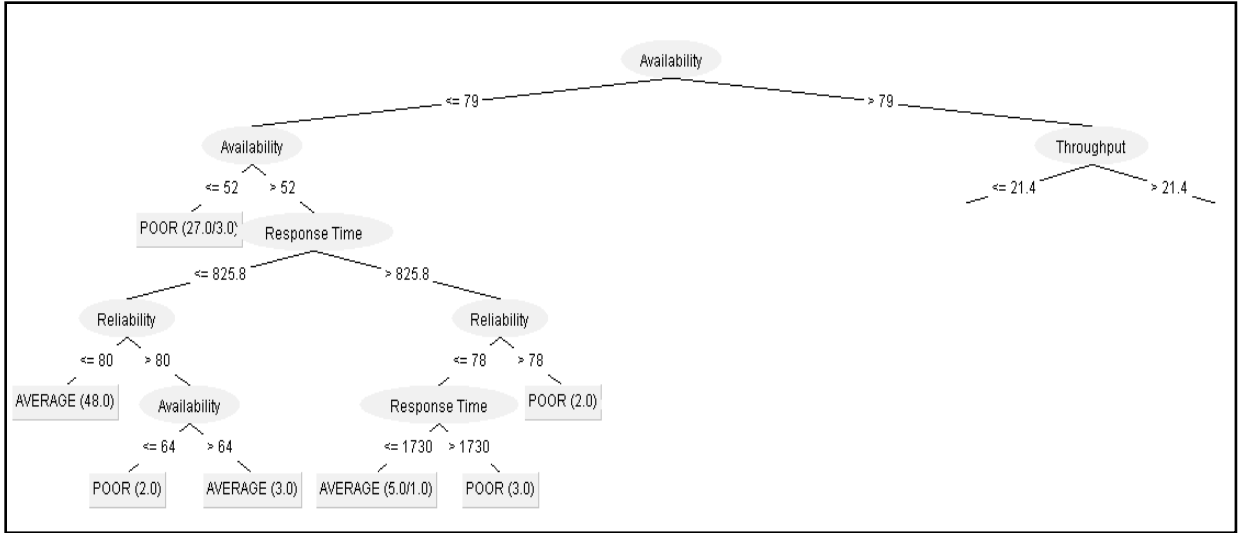


Fig 4.6 Partial Tree Constructed 5

For  $T_p > 21.4$ , from the remaining table

For Successability ( $S_u \leq 95$ ) : Classification = Good (6 / 1)

For  $S_u > 95$  : Classification = Excellent (11 / 1)

Obtained incomplete tree (figure 4.7):

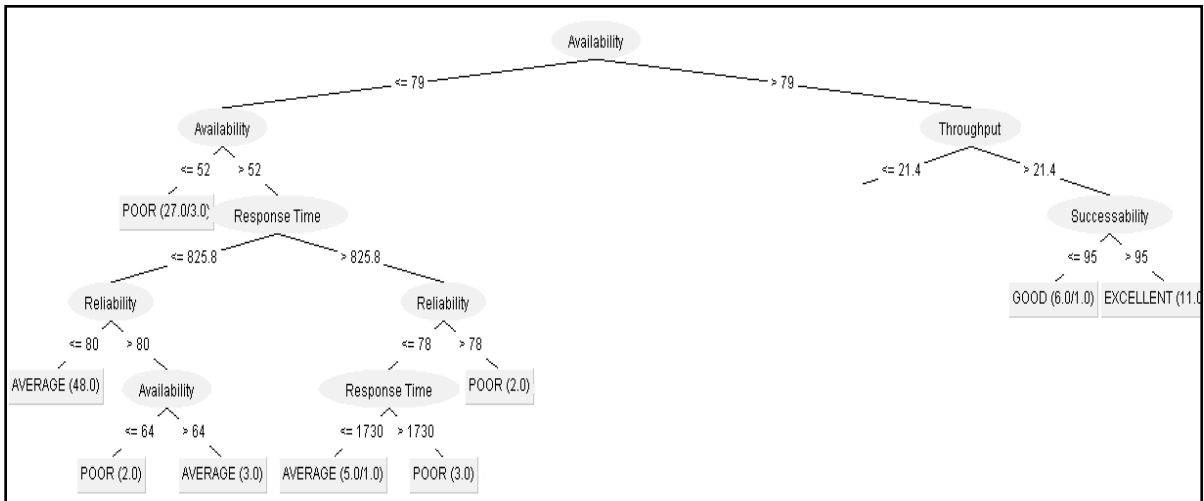


Fig 4.7 Partial Tree Constructed 6

For  $T_p \leq 21.4$ , from the remaining table

$$P = 3$$
$$A = 100$$
$$G = 90$$
$$E = 1.09811$$

Now, arrange the remaining table in ascending order according to RT.

Options for T: 725 and 2835

Choosing T = 725:

For S1 ( $RT \leq 725$ ),

$$A = 87$$

$$G = 90$$

$$E(S1) = 0.9998$$

For S2 ( $RT > 725$ ),

$$P = 3$$

$$A = 13$$

$$E(S2) = 0.69621$$

$$E(S|725) = 0.97463$$

$$\text{Gain}(725) = 0.12348$$

Now choosing T = 2835,

For S1 ( $RT \leq 2835$ ),

$$A = 99$$

$$G = 90$$

$$E(S1) = 0.99836$$

For S2 ( $RT > 2835$ ),

$$P = 3$$

$$A = 1$$

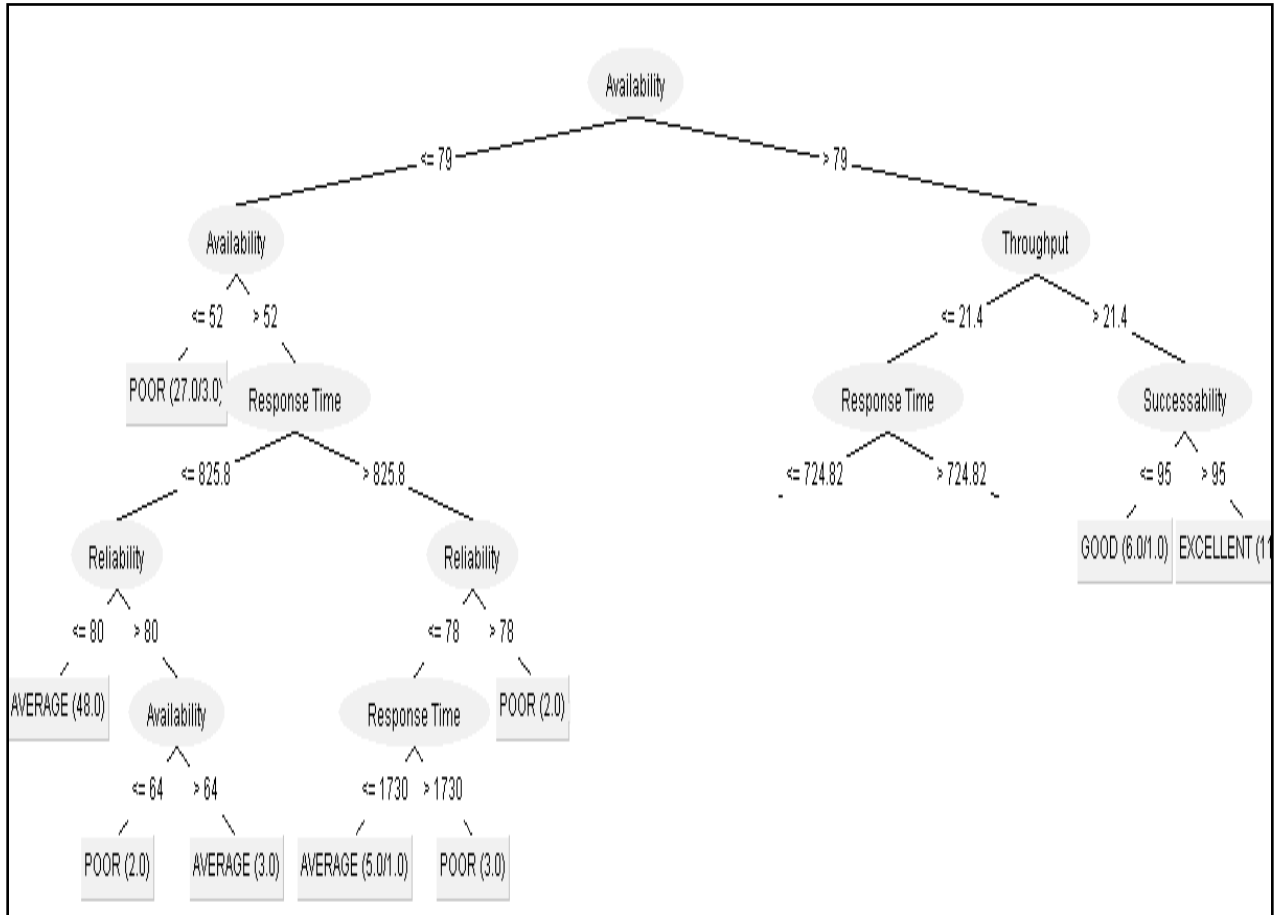
$$E(S2) = 0.81128$$



$$E(S|2835) = 0.99448$$

$$\text{Gain}(2835) = 0.10363$$

Since  $\text{Gain}(725) > \text{Gain}(2835)$ , then choose splitting point as  $\text{RT} = 725$ . Obtained incomplete tree (figure 4.8):



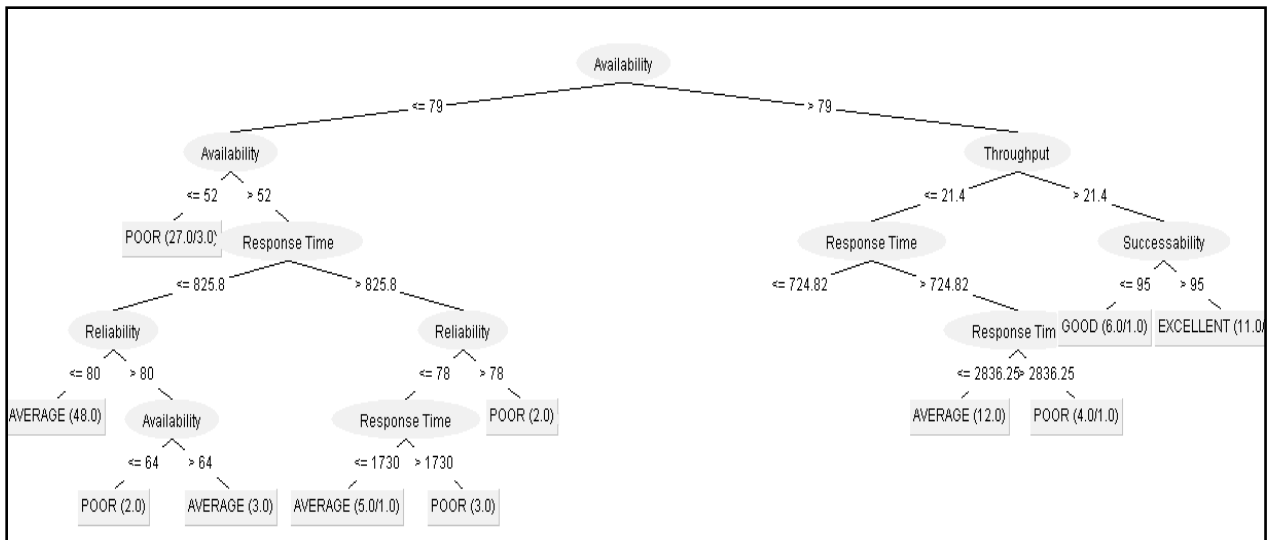


Fig 4.9 Partial Tree Constructed 8

For  $RT \leq 725$ , from the remaining table

$$A = 87$$

$$G = 90$$

$$E = 0.9998$$

Now, arrange the remaining table in ascending order according to Rel.

Options for T: 60 and 67

Choosing T = 60:

For S1 (Rel  $\leq 60$ ),

$$A = 6$$

$$G = 42$$

$$E(S1) = 0.54356$$

For S2 (Rel  $> 60$ ),

$$A = 81$$

$$G = 48$$

$$E(S2) = 0.95226$$

$$E(S|60) = 0.84143$$

Gain (60) = 0.15837

Now choosing T = 67,

For S1 (Rel ≤ 67),

A = 20

G = 58

E (S1) = 0.82128

For S2 (Rel > 67),

A = 67

G = 32

E (S2) = 0.52666

E (S|67) = 0.86971

Gain (67) = 0.13009

Since Gain (60) > Gain (67), then choose splitting point as Rel = 60. Obtained incomplete tree (figure 4.10):

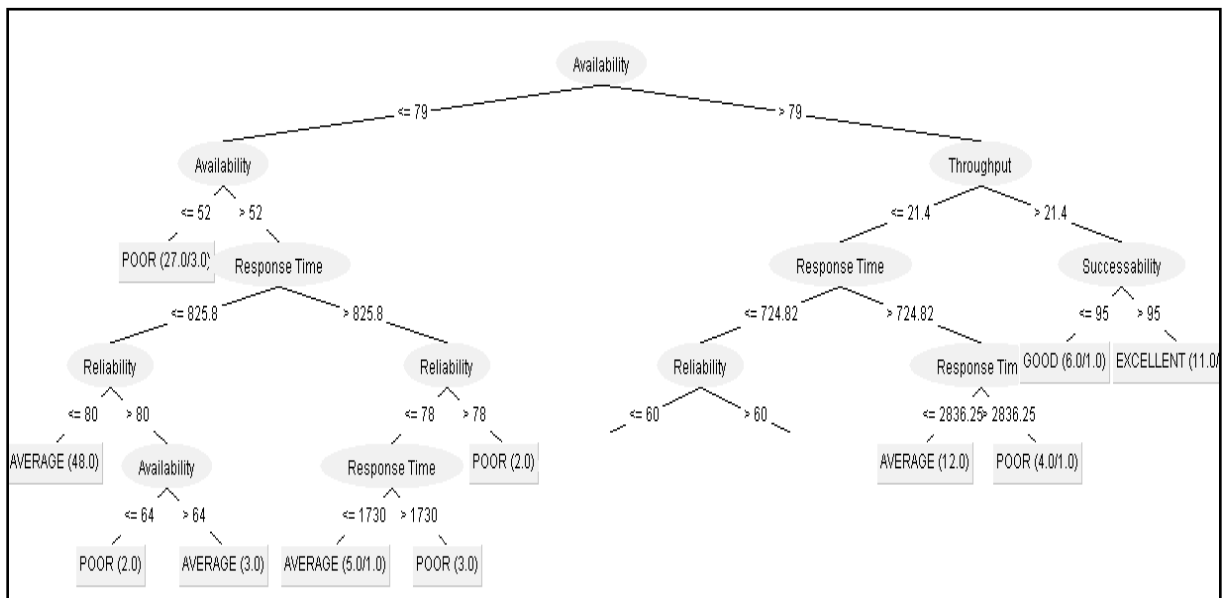


Fig 4.10 Partial Tree Constructed 9

For  $Rel \leq 60$ , from the remaining table

For  $Tp > 3.1$  : Classification = Good (25)

For  $Tp \leq 3.1$ , from the remaining table

For  $Av > 86$  : Classification = Good (11)

For  $Av \leq 86$ , from the remaining table

For  $Rel > 53$  : Classification = Average (4)

For  $Rel \leq 53$ , from the remaining table

For  $RT > 375$  : Classification = Average (2)

For  $RT \leq 375$  : Classification = Good (6)

Obtained incomplete tree (figure 4.11):

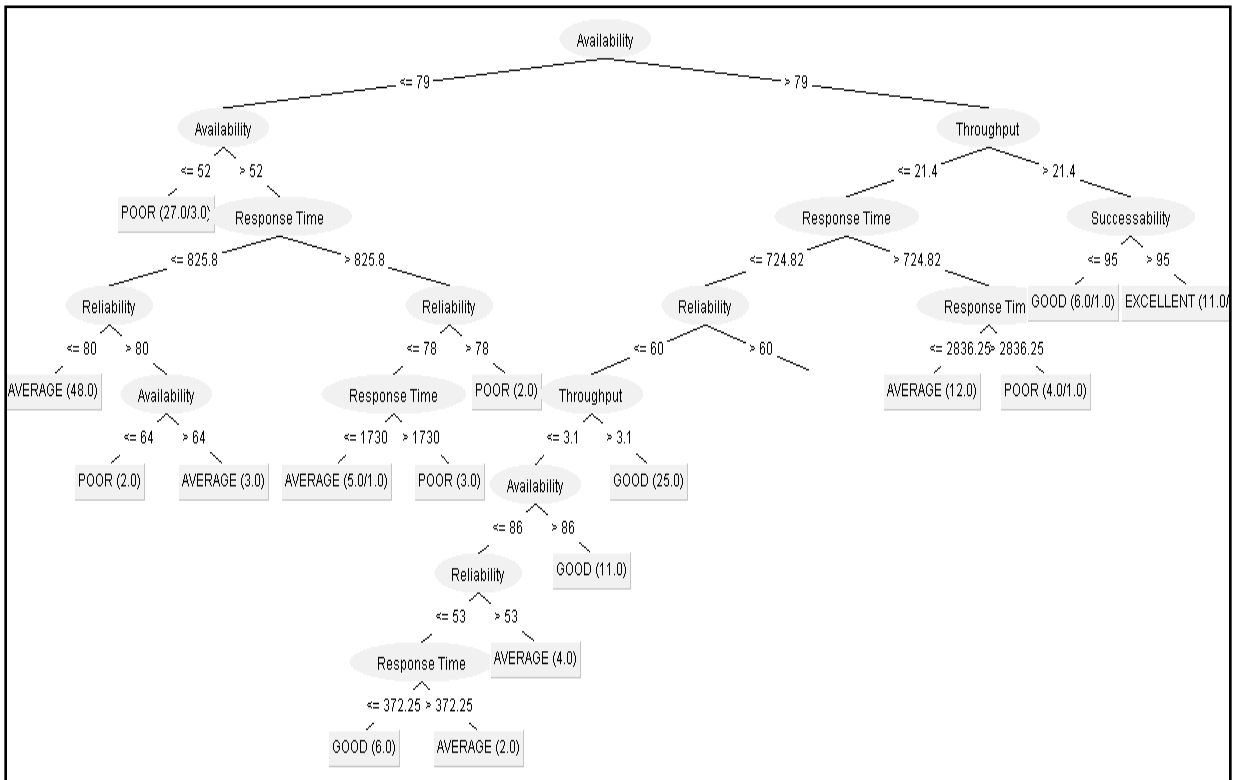


Fig 4.11: Partial Tree Constructed 10

For  $Rel > 60$ , from the remaining table

For  $Tp \leq 8.1$  : Classification = Average (55)

For  $Tp > 8.1$ , from the remaining table

For  $RT \leq 127$  : Classification = Good (24/1)

For  $RT > 127$ , from the remaining table

For  $Rel \leq 67$  : Classification = Good (7)

For  $Rel > 67$ , from the remaining table

For  $Tp \leq 13.3$  : Classification = Average (11/2)

For  $Tp > 13.3$  : Classification = Good (15/1)

Now all the instances of the table have been classified and hence the final and complete tree is obtained as in (figure 4.12):

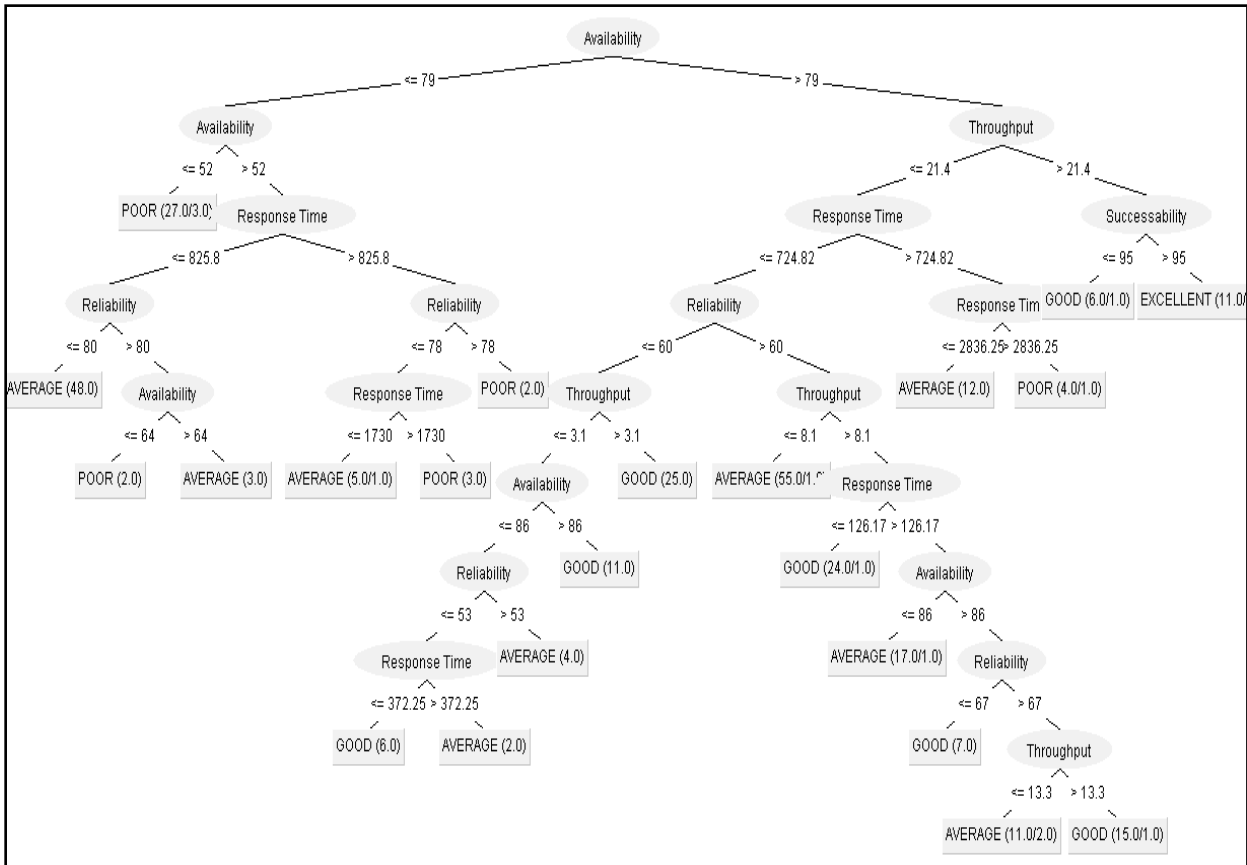


Fig 4.12: Partial Tree Constructed 11

With the help of this tree any new service can be classified according to its non function attributes of quality of service into four classes. And using these classifications service

requester can choose the most appropriate Web Services matching to the requirements of the same.

The algorithm is tested for many real data set against its training data set, where the result of the algorithm which puts the web services in one of the four classes namely EXCELLENT, GOOD, AVERAGE and POOR is checked manually by calculating the WSRF value for the web services based on its quality metrics. The class arrived using the WSRF value and algorithm is same all the time. This proves the tested data or unsupervised data works fine with the trained data set.

## **4.6 Summary**

In the initial stages of service-oriented computation, in order to obtain a suitable Web Services one has to search into UDDI Business Registries (UBRs). Since the numbers of Web Services available were very less and countable in hundreds, there was no requirement for any advanced web-service search engine. But now, numbers of Web Services in the registry are increasing rapidly and hence, access points to the registries, i.e. WSDL, are no more a meager as there are a lot of web registries, containing Web Services, are available on the internet.

Using Entropy-based Discretization, classification of Web Services could be done easily. Using this classification the requester could choose the most suitable Web Services according to his/her functional requirements and QoS preferences. Using Entropy-based Discretization a tree is obtained, in which the nodes belong to the QoS attributes. Tracing these nodes, could reach leaf nodes, which represent the classification of the Web Services into one of the four classes. In this chapter classification algorithm for Web Service Selection with specific QoS is explained in detail and in the next chapter the implementation of the same is discussed.

## **CHAPTER 5**

### **Implementation of Agent for WSS using Entropy Discretization method.**

In order to obtain a better reach for any Web Service amongst consumers involves making the consumer's task of discovering that Web Service easier and letting the consumer know about how good the Web Services is [41, 42]. Data mining can help us accomplish both the above mentioned goals. Data mining can enable us to simplify Web Services discovery by employing data mining techniques such as text mining on the UDDI registry to help the consumer find the service he needs, by mining on the list of different services available in the UDDI registry[42,43]. The data mining classification algorithm is applied in the QoS database which can help, to determine how good a Web Services is.

Classification is the process of determining to which group/class a new object belongs. Given a set of objects (training set), where each object contains a set of characteristics and a class. Classification finds a model based on the training set such that the class of previously unseen objects (test set) could be determined (as accurately as possible) using the values of their characteristic attributes. The use of classification to determine the quality of Web Services have been explained further and implemented in this chapter.

In order to classify Web Services a decision tree classifier is constructed using Entropy based discretization. Decision tree classifier is a classification model in data mining which represents the class to object relationship in the form of a tree and Entropy based discretization is a data transformation technique which is used when certain algorithms require only discrete or nominal values. This involves dividing the possible range of values into sub ranges called buckets or bins.

## 5.1 Decision tree construction

The entropy based discretization is used to construct the decision tree using a variant of the ID3 algorithm [44, 45]. ID3 algorithm looks through the attributes of the training set and extorts the attribute that best splits the given examples. If the attribute perfectly classifies the training sets then the ID3 algorithm stops; else it recursively executes on the split subsets to get their "best" splitting attribute. The algorithm exploits a greedy search technique. The "best" splitting attribute is determined by means of entropy and information gain [46,47].

The variant of ID3 that has been used is modified to handle continuous attributes in the test and training sets. The pseudo code of the algorithm "Decision Tree construction for Continuous Attributes" is as follows,

```
BuildDecTree (Training_Set ) returns node_ptr  
IF all entries of the Training_Set all of the same class  
THEN return a pointer to a LEAF node with class = one with the most  
examples  
ELSE  
choose split_point S based on information gain.  
IF splitting fails  
THEN return pointer to LEAF node as above  
ELSE  
split_examples( S, Training_Set, +examples, -examples)  
LET +b = BuildDecTree( +examples )  
AND -b = BuildDecTree( -examples )  
return a pointer to a BRANCH  
with criteria = S  
+branch = +b  
-branch = -b  
END {if choosing fails}  
END {if examples of same class}
```



Once the decision tree has been constructed using the algorithm, the decision tree could be used to classify new Web Services based on their QoS parameters namely response time, availability, throughput, success ability and reliability. The function details used for this algorithm implementation is given in APPENDIX B -1

The time complexity of this algorithm is  $O(mn^2)$ , where  $m$  is the number of records and  $n$  is the number of attributes. This is because, there are  $m$  records and at a time and for computing information gain, it has to consider each of the  $n$  attributes. So in a particular level, the complexity is  $O(mn)$ . In the worst case, there will be a split corresponding to each of the  $n$  attributes. So altogether it becomes like  $O(mn^2)$  in the worst case. But here as the numeric data are split based on statistical mean the number of levels in the worst case is  $\log_2 m$ . So the time complexity becomes  $O(mn \log_2 m)$ .

The QoS information is stored in the UDDI registry and is updated by service providers in case of amendments. The service discovery agent obtains the QoS information from the UDDI registry, stores them in its database and classifies the Web Services based on its QoS information.

Therefore, a service requestor who is looking for a service can send a Web Service discovery request to the agent with his QoS requirements and specification. The agent will mine the QoS attribute of all available Web Services belonging to that particular functional domain by applying modified ID3 algorithm and suggest the available services which meet the requestor's requirements as shown in figure 5.1.

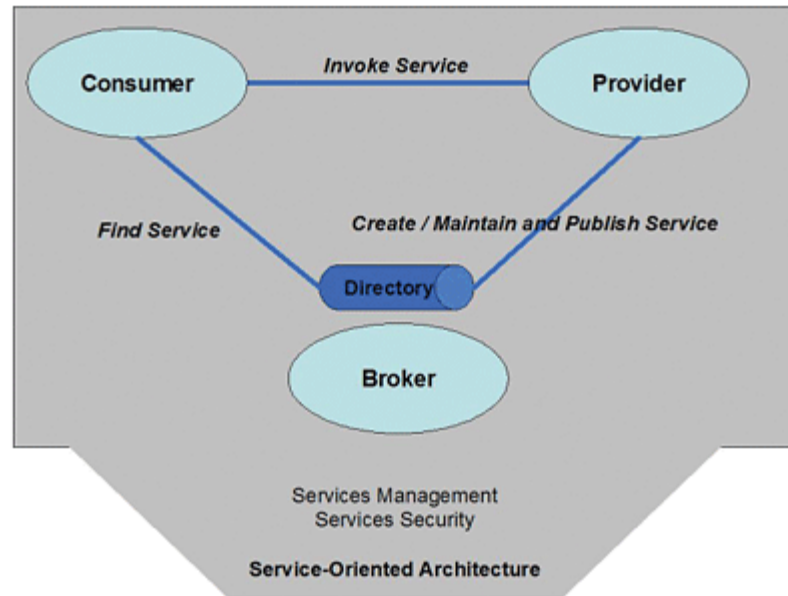


Fig 5.1 Agent based architecture for Web Services selection[49].

The same dataset which is used in the previous chapter is taken for consideration in this case also. The main functions that are used for this coding and partial code is given in APPENDIX – B-1 and the database is given in APPENDIX – C.

### 5.1.1 Decision tree rule induction algorithm using entropy based discretization.

#### Design:

The design uses a variation of ID3 algorithm to induce the decision tree which enable decision tree classification for continuous datasets. ID3 algorithm has been used because of its quick build time and accuracy comparable to that of other algorithms like C4.5 for numeric datasets. ID3 inherently uses entropy based discretization for creating pure bins out of the training dataset.

#### Implementation:

The Java based implementation of this algorithm has been carried out and results of the same have been recorded. The result consists of a decision tree which is represented in the form of rules. This java program is also capable of accepting inputs of QoS parameter test values

from the user, traversing the decision tree and giving out the classification of the test Web Services. The resultant tree will be used as input for classification Web Service whose design is discussed in the next paragraph. Output of the Decision tree rule induction algorithm is shown in figure 5.2 and Decision tree traversal and classification using Decision tree rule induction algorithm is shown in figure 5.3

```

1  import java.io.BufferedReader;
2  import java.io.File;
3  import java.io.FileInputStream;
4  import java.io.InputStreamReader;
5  import java.util.Scanner;
6  import java.util.StringTokenizer;
7  import java.util.Vector;
8
9  public class DecTree {
10
11
12     int numAttributes;           // The number of attributes including the output attribute
13     String []attributeNames;    // The names of all attributes. It is an array of dimension numAttributes.
14

```

---

```

run:
if( ResponseTime <= "173") {
    if( Throughput <= "3.8") {
        if( Throughput <= "8.1") {
            Classification = "POOR";
        } else {
            if( Throughput <= "4.3") {
                Classification = "POOR";
            } else {
                if( Throughput <= "4.4") {
                    Classification = "POOR";
                } else {
                    if( Reliability <= "53") {
                        Classification = "AVERAGE";
                    } else {
                        if( ResponseTime <= "451") {
                            Classification = "AVERAGE";
                        } else {
                            if( ResponseTime <= "464.62") {

```

Fig 5.2 Screenshot showing output of the Decision tree rule induction algorithm

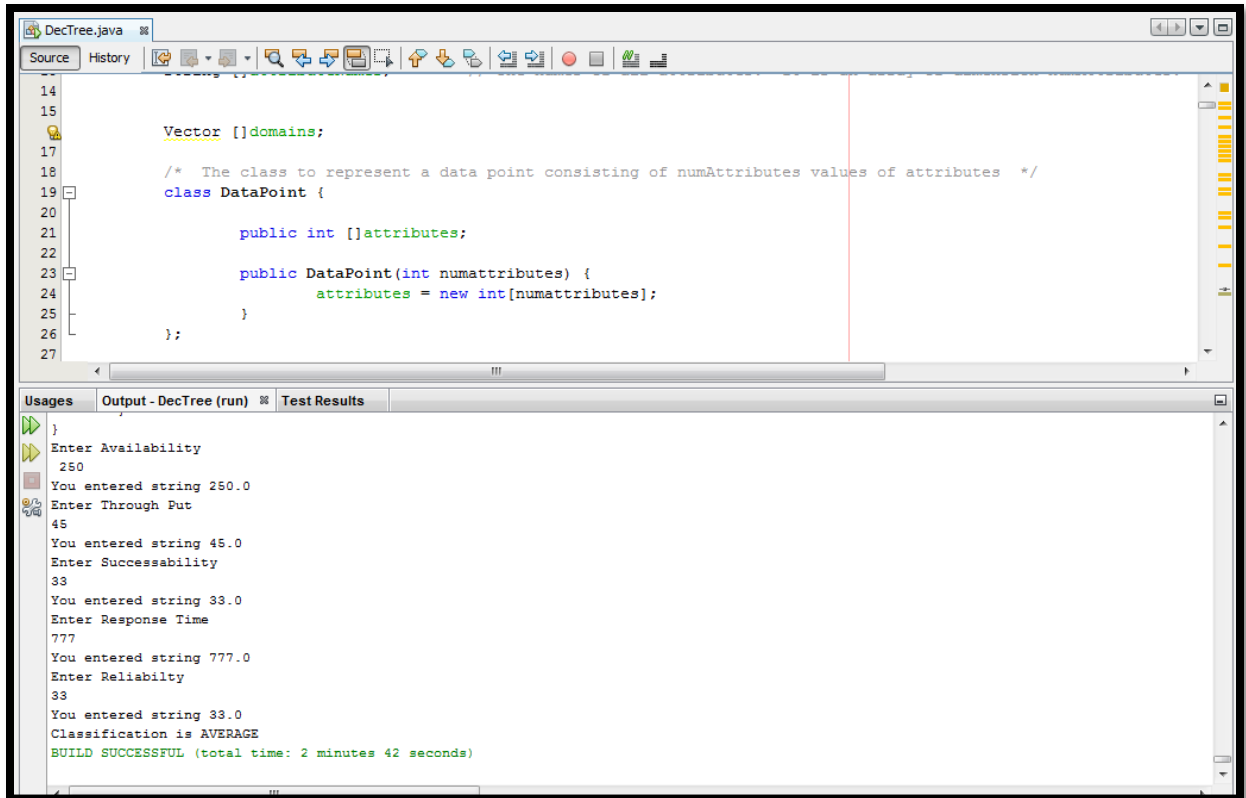


Fig 5.3 Decision tree traversal and classification using Decision tree rule induction algorithm.

## 5.1.2 QoS Classification Web Services

### Design:

The rules generated using the Decision tree rule induction algorithm are consolidated and made into SOAP based Web Services, so that any web client can choose a Web Service with particular QoS.

### Implementation:

The Web Services use Java API for XML Web Services (JAX-WS) which is a significant part of the Java EE 5 and EE 6 platforms. The Web Services is deployed on the glassfish server. On successful deployment, the Web Services is then tested using the IDE's tester page which runs on the local browser. As values for these attributes are entered, as in figure

5.4, the tracing for QoS classification is done, and final class of the Web Service is displayed as in figure 5.5

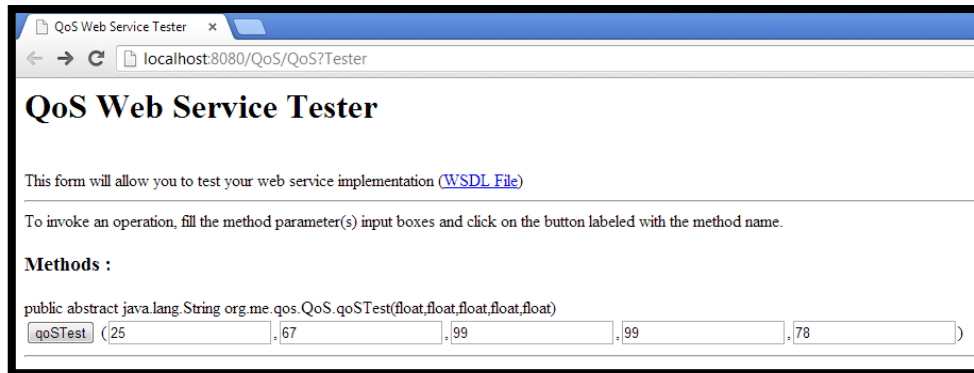


Fig 5.4 IDE's tester page for the QoS classification Web Services.



Fig 5.5 IDE's Method invocation trace for the QoS classification Web Services.

Clippings of the SOAP request, and SOAP response for the QoS classification Web Services is given below,

### SOAP Request

---

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:QoSTest xmlns:ns2="http://qos.me.org/">
      <Availability>25.0</Availability>
      <ResponseTime>67.0</ResponseTime>
      <Throughput>99.0</Throughput>
      <Reliability>99.0</Reliability>
      <Successability>78.0</Successability>
    </ns2:QoSTest>
  </S:Body>
```

### SOAP Response

---

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:QoSTestResponse
xmlns:ns2="http://qos.me.org/">
      <return>AVERAGE</return>
    </ns2:QoSTestResponse>
  </S:Body>
</S:Envelope>
```

The WSDL file for this Web Service is given in APPENDIX B-2.

## 5.2 Summary

Web Services are becoming increasingly used and a large number of consumers are building their business solutions using Web Services technology[50,51,52]. The need for QoS specifications for Web Services have arisen due to

1. Consumer's prospect for superior Web Services performance.
2. Services provider's obligation to provide high quality service so as to improve the usability and utility of their services which in turn decides their standing in the market.

This chapter dealt with the design and implementation of a QoS classifier Web Service. This classifier Web Service takes QoS parameters as input and applies a entropy based discretization algorithm and yields the classification of Web Services into different classes based on QoS attributes to the web based client.

The resultant output is a decision tree rules. In the next chapter, the output of the data mining classification algorithm is composed with visualization Web Service get more relevant information in the form of visual tree. In the tree structure, the web service client will get to know, how different attributes play role in deciding the class of the web service.

## **CHAPTER 6**

### **Implementation of Agent for WSS through Web Services Composition**

Data mining helps to extract the useful information from the data, which can be useful to make practical interpretations for the decision making. In the previous chapter the agent web service could classify the web services into different groups based on their QoS values. The results of the classification web service can be better understood in the form of trees, so the visualization software is exported as Web Services. The SOAP response of the classification web service will be given as SOAP request to the visualization web service to generate final output in forms of trees. In this chapter, the processes of data mining namely classification and visualization are made into Web Services and composed. The composition of Web Services is done through BPEL engine is discussed in detail.

#### **6.1 Composition of Web Services**

A novel way of mining QoS attribute of Web Services have been done using the principles of composition of Web Services. A Web Service is used to implement J48 classification algorithm and will output a decision tree rules[55,56,57]. Another Web Service is used which will generate the decision tree output from the decision tree rules. The decision tree output will enable a client to understand the effect of various QoS parameter visually on deciding the class of the Web Service. Since both the Web Services are to be invoked one after the other, BPEL Engine will be the natural choice for hosting the Web Services that are to be composed.

Web Services and SOA (Service Oriented Architecture), viewed in a process-oriented perspective, need a language in order to define how services can be composed into business



processes [68]. Such definitions would allow describing abstract process definitions as well as executable processes.

Recently many languages have emerged and proposed in the literature for composition and execution of Web Services, including: WSCL, XLALNG, WSFL, BPMN, WSCI, BPML and BPEL4WS[72].

### 6.1.1 BPEL4WS

The Business Process Execution Language for Web Services (BPEL4WS), which is also referred to as BPEL, is currently a de facto standard for building, specifying and executing business processes for Web Services composition and orchestration.

BPEL composes Web Services to get a specific result. The composition result is named a process, involved services are called partners, and message exchange is referred to as an activity. In other words, a process contains a set of activities and it invokes external partner services using a WSDL interface [61,66].

A BPEL process defines the order in which involved Web Services are composed, either in sequence or in parallel. BPEL allows describing conditional activities. An invocation of a Web Services can for example rely on the result of another Web Service's invocation. With BPEL, it is possible to create loops, declare variables, copy and assign values as well as to use fault handlers. Complex business processes can be built algorithmically by using all these constructs. It can be helpful to describe business processes graphically through UML (Unified Modeling Language) activity diagrams [74].

BPEL supports two different ways of describing business processes that support orchestration and choreography [64]:

- **Orchestration:** Executable processes allow for specifying the details of business processes. They follow the orchestration paradigm and can be executed by an orchestration engine.

- **Choreography:** Abstract business protocols allow specification of the public message exchange between parties only. They do not include the internal details of process flows and are not executable. They follow the choreography paradigm [69].

The role of BPEL is to define a new Web Services by composing a set of existing services through a process-integration type mechanism with control language constructs[70].

A process consists of a set of activities. It interacts with external partner services through a WSDL interface [61]. To define a BPEL process,

1. A BPEL source file (.bpel) which describes the execution order, activities and conditional behaviors are needed.
2. A process interface (.WSDL) that defines the ports, the namespace, partner link types, operations, and messages which are required to determine process activities, and WSDL files[80] are needed in order to create a valid, executable BPEL definition.

The BPEL process element is the root element of BPEL process definition. It has a name attribute and it is used to specify the definition related namespaces and using BPEL offers many interesting benefits[74].

## 6.2 WEKA

WEKA comes with an API. Using this API, most of WEKA's tasks can be exported to be used in other programs and software[63,73]. The classes for the data mining task can be added to the library. These classes are then used to train the dataset and give the results accordingly.

WEKA uses flat file format for reading the database. ARFF (Attribute-Relation File Format) is an ASCII format that stores the set of attributes of data.

A simple ARFF file is shown here

```
@RELATION car
```

```
@ATTRIBUTE engine-capacity NUMERIC
@ATTRIBUTE horse-power NUMERIC
@ATTRIBUTE millage NUMERIC
@ATTRIBUTE top-speed NUMERIC
@ATTRIBUTE class {car-small, car-medium, car-high-end}
```

```
@DATA
4000,1100,4,364, car-high
2000,700,10,220,car-medium
1100,300,16,180, car-small
```

First the attributes, engine-capacity, horse-power, millage and top-speed are declared and then the classes are specified. Then add the data in the order attributes are declared. The attributes can be of various type:

- Numeric
- String
- <nominal-specs>
- date [<date-format>]

WEKA loads the data from the ARFF file as instances; the class for WEKA instances is present in:

```
WEKA.core.instances
```

The input data file is first read using the java's FileReader and then the a new class for the instances is declared and the read content from the filereader is added to it.

```
“data = new FileReader(file);
instances = new Instances(data);
// the last attribute is made the index
instances.setClassIndex(instances.numAttributes()-1);“
```

Now these instances can be used to train classifiers or create clusters or some other data mining.

Since our notion is to develop agent Web Services for Web Services selection through distributed data mining, apply the classification algorithm of WEKA software on the data set that is already used. the QoS dataset is available as CSV (Comma Separated File) file, after this classification process of data mining is done through the Web Services another Web

Services for visualization purpose is called known as Graphviz software, where the results of the classification Web Services is given in tree form.

WEKA API has classes for J48 tree. First a classifier using the J48 class is created and the instance data. Then the training sets and training results are shown. A small snippet for creating a classifier and showing the result of the J48 tree classification using the WEKA api is shown below:

```
“import WEKA.classifiers.trees.J48;
import WEKA.core.instances;

//get the data from the file
file_content = new FileReader(file);
data_instance = new Instances(data_content);
//the last attribute is make the index
data_instance.setClassIndex(data_instance.numAttributes()-1);

//create a new classisfier using the J48 class
J48 classifier = new J48();
//build the classifier from data_instances
classifier.buildClassifier(data_instances);
//print the classifier results
System.out.println(“classifier.toString());”
```

The output of the above code for a file containing QoS attributes is as follows

“-----

```
Response Time <= 306.8
| Throughput <= 9
| | Documentation <= 32
| | | Response Time <= 125
| | | | Availability <= 88: AVERAGE (5.0)
| | | | Availability > 88: GOOD (3.0)
| | | Response Time > 125: AVERAGE (46.0/1.0)
| | Documentation > 32
| | | Throughput <= 3.8
| | | | Successability <= 98
| | | | | Compliance <= 89
| | | | | | Documentation <= 87: AVERAGE (10.0)
| | | | | | Documentation > 87
| | | | | | | Compliance <= 78: GOOD (2.0)
| | | | | | | Compliance > 78: AVERAGE (4.0/1.0)
| | | | | | | Compliance > 89
| | | | | | | Successability <= 96
| | | | | | | Latency <= 7.6: GOOD (7.0)
| | | | | | | Latency > 7.6
| | | | | | | | Throughput <= 1.2: AVERAGE (2.0)
| | | | | | | | Throughput > 1.2: GOOD (4.0/1.0)
| | | | | | | Successability > 96: AVERAGE (3.0)
| | | | | | Successability > 98: GOOD (6.0)
| | | | Throughput > 3.8: GOOD (15.0/1.0)
| | Throughput > 9
| | | Latency <= 63.33
| | | | Documentation <= 11
| | | | | Throughput <= 15.3
| | | | | | Compliance <= 89
```

| | | | | Throughput <= 13.3: AVERAGE (10.0/1.0)  
 | | | | | Throughput > 13.3: GOOD (4.0/1.0)  
 | | | | | Compliance > 89: GOOD (3.0)  
 | | | | | Throughput > 15.3  
 | | | | | Throughput <= 25.6: GOOD (23.0)  
 | | | | | Throughput > 25.6  
 | | | | | Response Time <= 136.94: GOOD (3.0)  
 | | | | | Response Time > 136.94: EXCELLENT (3.0)  
 | | | | | Documentation > 11  
 | | | | | Reliability <= 73  
 | | | | | Latency <= 3.63  
 | | | | | Reliability <= 67  
 | | | | | Best Practices <= 79: EXCELLENT (2.0)  
 | | | | | Best Practices > 79: GOOD (2.0)  
 | | | | | Reliability > 67: EXCELLENT (8.0/3.0)  
 | | | | | Latency > 3.63: GOOD (27.0/2.0)  
 | | | | | Reliability > 73: EXCELLENT (2.0)  
 | | | | | Latency > 63.33: AVERAGE (7.0/1.0)  
 Response Time > 306.8  
 | | | | | Response Time <= 1041  
 | | | | | Latency <= 50  
 | | | | | Documentation <= 33  
 | | | | | Throughput <= 18.6: AVERAGE (27.0/2.0)  
 | | | | | Throughput > 18.6: GOOD (3.0/1.0)  
 | | | | | Documentation > 33  
 | | | | | Successability <= 68: AVERAGE (3.0)  
 | | | | | Successability > 68  
 | | | | | Documentation <= 42  
 | | | | | Throughput <= 9.6: AVERAGE (4.0)  
 | | | | | Throughput > 9.6: GOOD (3.0)  
 | | | | | Documentation > 42: GOOD (7.0)

```
| | Latency > 50
| | | Throughput <= 1.9: POOR (8.0/1.0)
| | | Throughput > 1.9
| | | | Response Time <= 411.83: AVERAGE (8.0)
| | | | Response Time > 411.83
| | | | | Documentation <= 40: POOR (9.0/1.0)
| | | | | Documentation > 40: AVERAGE (5.0/1.0)
| Response Time > 1041: POOR (22.0/1.0)
```

Number of Leaves: 35

Size of the tree: 69

This API is going to be used to export these classes as Web Services. The data file will be uploaded to the server which contains the mining code. The SOAP response will have the data mining result.

The output of this SOAP web service which gives the result of classification process is given as input to another web service to view the result of classification in the tree format with the help of BPEL execution engine. Graphviz the web service used for visualization process is explained in the following paragraph.

### **6.3 Graphviz**

Graphviz or Graphic Visualization Software is an open source tool for drawing graphs specified in dot language. This free software was initiated by AT&T Labs Research. It is released under the Eclipse Public License.

DOT is a plain text graph description language used by graphviz to plot directed or undirected graphs. It has a very simple syntax and easy to parse. The graphviz DOT files can

easily be generated using computer programs because of the simplicity of the syntax. The file extensions of dot files is .dot.

A sample of a dot file can be seen here.

```
digraph example {  
    // The label attribute can be used to change the label of a node  
    N0 [label="Foo"];  
    // Here, the node shape is changed.  
    N1 [shape=box color=blue];  
    // These edges both have different line properties  
    N0 -> N1 -> N2 [color=blue];  
    N1 -> N3 [style=dotted];  
}
```

Here the nodes and the connections between the nodes are explained. Each node can have various attributes like labels, shape, color etc. Each connection can also have attributes, like style, color, label etc.

The output of the above code is:

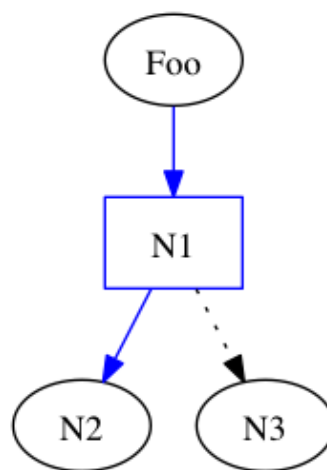


Fig 6.1 Graphviz Output



The output of the J48 tree will be available in dot format and is passed to the input of the visualization service and get the image of the tree as a png and save it. This JFrame image is shown in figure 6.1.

## 6.4 Implementation of J48 Classifier Web Service

In data mining classification is the problem of identifying the sub-population to which new observation belong. Thus the requirement is that new individual items are placed into groups based on quantitative information on one or more measurements based on the training set in which previously decided groupings are already established. J48 classifier is used to generate a decision tree for classification. [65] This service gives the J48 classification decision rules for the given QoS data. The SOAP request and SOAP response messages are as follows. [62]

Output

### SOAP Request

---

```
“<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
<ns2:execute xmlns:ns2="http://source/">
<file>/Users/susila/Work/WEKA/data/QoS.arff</file>
</ns2:execute>
</S:Body>
</S:Envelope> “
```

## SOAP Response

---

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlSOAP.org/SOAP/envelope/">
<S:Body>
<ns2:executeResponse xmlns:ns2="http://source/">
<return>-----
```

```
Response Time <= 306.8
| Throughput <= 9
| | Documentation <= 32
| | | Response Time <= 125
| | | | Availability <= 88: AVERAGE (5.0)
| | | | Availability > 88: GOOD (3.0)
| | | Response Time > 125: AVERAGE (46.0/1.0)
| | Documentation > 32
| | | Throughput <= 3.8
| | | | Successability <= 98
| | | | | Compliance <= 89
| | | | | | Documentation <= 87: AVERAGE (10.0)
| | | | | | Documentation > 87
| | | | | | Compliance <= 78: GOOD (2.0)
| | | | | | Compliance > 78: AVERAGE (4.0/1.0)
| | | | | Compliance > 89
| | | | | | Successability <= 96
| | | | | | | Latency <= 7.6: GOOD (7.0)
| | | | | | | Latency > 7.6
| | | | | | | Throughput <= 1.2: AVERAGE (2.0)
| | | | | | | Throughput > 1.2: GOOD (4.0/1.0)
| | | | | | Successability > 96: AVERAGE (3.0)
| | | | Successability > 98: GOOD (6.0)
```

| | | Throughput > 3.8: GOOD (15.0/1.0)  
 | Throughput > 9  
 | | Latency <= 63.33  
 | | | Documentation <= 11  
 | | | | Throughput <= 15.3  
 | | | | | Compliance <= 89  
 | | | | | Throughput <= 13.3: AVERAGE (10.0/1.0)  
 | | | | | Throughput > 13.3: GOOD (4.0/1.0)  
 | | | | | Compliance > 89: GOOD (3.0)  
 | | | | Throughput > 15.3  
 | | | | | Throughput <= 25.6: GOOD (23.0)  
 | | | | | Throughput > 25.6  
 | | | | | Response Time <= 136.94: GOOD (3.0)  
 | | | | | Response Time > 136.94: EXCELLENT (3.0)  
 | | | Documentation > 11  
 | | | | Reliability <= 73  
 | | | | | Latency <= 3.63  
 | | | | | Reliability <= 67  
 | | | | | | Best Practices <= 79: EXCELLENT (2.0)  
 | | | | | | Best Practices > 79: GOOD (2.0)  
 | | | | | Reliability > 67: EXCELLENT (8.0/3.0)  
 | | | | | Latency > 3.63: GOOD (27.0/2.0)  
 | | | | Reliability > 73: EXCELLENT (2.0)  
 | | Latency > 63.33: AVERAGE (7.0/1.0)  
 Response Time > 306.8  
 | Response Time <= 1041  
 | | Latency <= 50  
 | | | Documentation <= 33  
 | | | | Throughput <= 18.6: AVERAGE (27.0/2.0)  
 | | | | Throughput > 18.6: GOOD (3.0/1.0)  
 | | | Documentation > 33

```

| | | | Successability <= 68: AVERAGE (3.0)
| | | | Successability > 68
| | | | | Documentation <= 42
| | | | | | Throughput <= 9.6: AVERAGE (4.0)
| | | | | | Throughput > 9.6: GOOD (3.0)
| | | | | | Documentation > 42: GOOD (7.0)
| | | Latency > 50
| | | | Throughput <= 1.9: POOR (8.0/1.0)
| | | | Throughput > 1.9
| | | | | Response Time <= 411.83: AVERAGE (8.0)
| | | | | Response Time > 411.83
| | | | | | Documentation <= 40: POOR (9.0/1.0)
| | | | | | Documentation > 40: AVERAGE (5.0/1.0)
| | Response Time > 1041: POOR (22.0/1.0)

```

Number of Leaves: 35

Size of the tree: 69

</return>

</ns2:executeResponse>

</S:Body>

</S:Envelope> “

## 6.5 Implementation of Visualization Web Service

Visualization Web Service is designed, which will take dot string input file and will produce tree image as output using grapviz. Since the client can have tree view, it enables the client to see the impact of QoS parameter in deciding the class of the Web Service in the form of tree.

The Web Service code is as:

```
<code>

package source;

import java.awt.BorderLayout;
import java.awt.Color;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.swing.JFrame;
import WEKA.gui.treevisualizer.PlaceNode2;
import WEKA.gui.treevisualizer.TreeVisualizer;
```

```
@WebService()
```

```
public class J48Visualize {
```

```
    @WebMethod(operationName = "operation")
```

```
    public String operation(@WebParam(name = "cls_data")
```

```
String cls_data) {
```

```
    TreeVisualizer tv = new TreeVisualizer(null, cls_data, new PlaceNode2());
```

```
    JFrame jf= new JFrame("WEKA Classifier Tree");
```

```
    jf.setAlwaysOnTop(true);
```

```
    jf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```
    jf.setSize(800,600);
```

```
    jf.getContentPane().setLayout(new BorderLayout());
```

```

        jf.getContentPane().add(tv, BorderLayout.CENTER);
        jf.setVisible(true);
        tv.setBackground(Color.white);
        tv.fitToScreen();
        return cls_data;
    }

}
</code>

```

The J48 classifier is going to classify the data and give us an output in a string format. This Web Services will call another method on the classify class which will convert the output of the J48 tree into a dot file. This output can then be used in graphviz to generate a graph.

The Web Service is described here:

```

<code>
package source;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import WEKA.classifiers.trees.J48;
import WEKA.core.Instances;

@WebService()

```

```

public class J48data {

    @WebMethod(operationName = "mine")
    public String mine(@WebParam(name = "file")
String file) throws IOException, Exception {
        Instances data;
        data = new Instances(new BufferedReader(new FileReader(file)));
        data.setClassIndex(data.numAttributes() - 1);
        J48 cls = new J48();
        cls.buildClassifier(data);
        return cls.graph();

    }

}
</code>

```

This webservice works as the earlier described earlier classification webservice. It just returns the J48 tree in a dot format.

Invocation of this webservice

SOAP Request

```

<?xml version="1.0" encoding="UTF-8"?>
    <S:Envelope xmlns:S="http://schemas.xmlSOAP.org/SOAP/envelop"><S:
Header/>
    <S:Body>
        <ns2:mine xmlns:ns2="http://source/"><file>/Users/susila/Work/WEKA-
3-6-4/data/QoS.arff</file></ns2:mine>
    </S:Body>
</S:Envelope>

```

## SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body>
<ns2:mimeResponse xmlns:ns2="http://source/"><return>digraph J48Tree
    {
N0 [label="Response Time" ]
N0->N1 [label="<= 388.5"]
N1 [label="Throughput " ]
N0->N2 [label="> 388"]
N2 [label="Response Time" ]
N2->N3 [label="<= 104.1"]
N3 [label="Latency" ]
N2->N4 [label="> 104.1"]
N4 [label="POOR" shape=box style=filled ]
N3->N5 [label="<=67"]
N5 [label="Documentation" ]
N3->N6 [label="> 67"]
N6 [label="Documentation" ]
N5->N7 [label="<=33"]
N7 [label="AVERAGE " shape=box style=filled ]
N5->N8 [label="<33"]
N8 [label="Successability" ]
N8->N9 [label="<=98"]
N8 [label="Successability" ]
N8->N10 [label=">98"]
N10 [label="GOOD " shape=box style=filled ]
N1->N11 [label="<= 31"]
N8 [label="Documentation" ]
N1->N12 [label=">31"]
N12 [label="Latency" ]
```



```

N12->N13 [label="<= 63.3"]
N13 [label="Documentation" ]
N13->N15 [label=">11"]
N15 [label="Realiability" ]
N15->N16 [label=">73"]
N16 [label="EXCELLENT " shape=box style=filled ] }
</return>
</ns2:mimeResponse>
</S:Body>
</S:Envelope>

```

The SOAP response is as desired , the dot format of the tree.

This response now is input to the visualization service, The desired Tree is got in an image format.

Calling the visualization service:

### SOAP Request

---

```

<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlSOAP.org/SOAP/envelope/"><S:Header/>
<S:Body>
<ns2:operation xmlns:ns2="http://source/"><cls_data>digraph J48Tree {
N0 [label="Response Time" ] N0->N1 [label="<= 388.5"]
N1 [label="Throughput " ]
N0->N2 [label="> 388"]
N2 [label="Response Time" ]
N2->N3 [label="<= 104.1"]
N3 [label="Latency" ]
N2->N4 [label="> 104.1"]

```

```

N4 [label="POOR" shape=box style=filled ]
N3->N5 [label="<=67"]
N5 [label="Documentation" ]
N3->N6 [label="> 67"]
N6 [label="Documentation" ]
N5->N7 [label="<=33"]
N7 [label="AVERAGE " shape=box style=filled ]
N5->N8 [label="<33"]
N8 [label="Successability" ]
N8->N9 [label="<=98"]
N8 [label="Successability" ]
N8->N10 [label=">98"]
N10 [label="GOOD " shape=box style=filled ]
N1->N11 [label="<= 31"]
N8 [label="Documentation" ]
N1->N12 [label=">31"]
N12 [label="Latency" ]
N12->N13 [label="<= 63.3"]
N13 [label="Documentation" ]
N13->N15 [label=">11"]
N15 [label="Realiability" ]
N15->N16 [label=">73"]
N16 [label="EXCELLENT " shape=box style=filled ] }
</cls_data></ns2:operation>
</S:Body>
</S:Envelope>

```

The output is the desired tree which is shown in figure 6.2

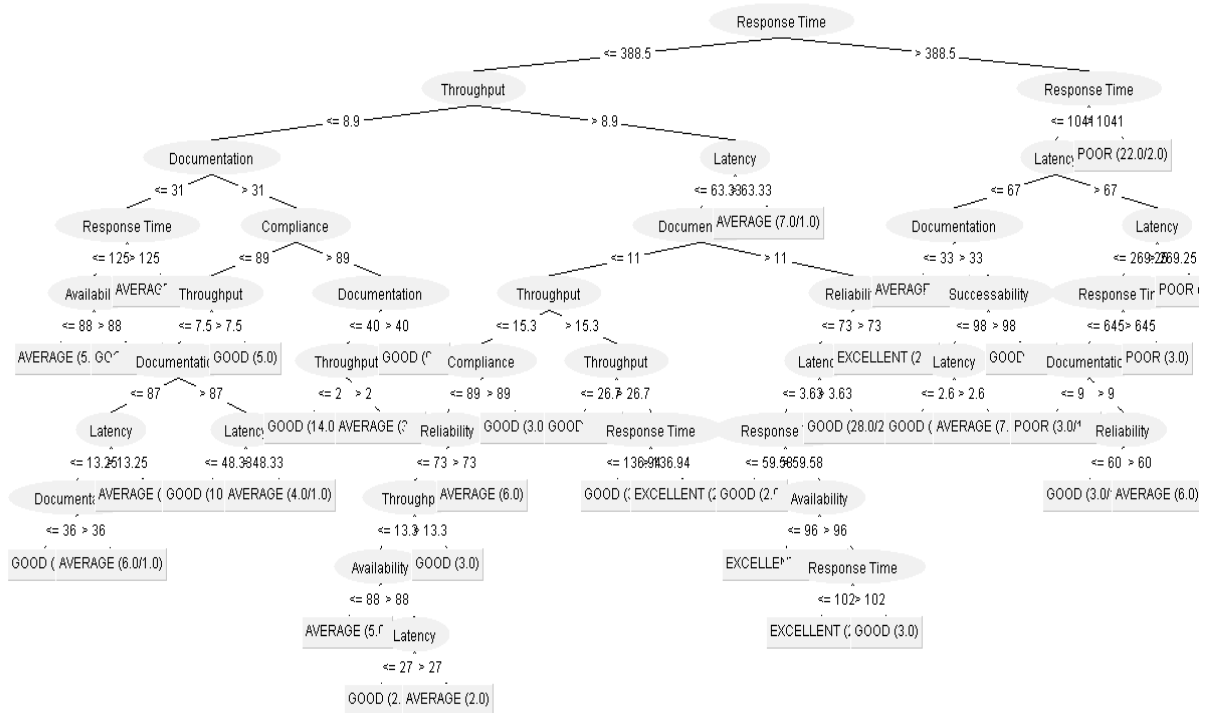


Fig 6.2 J48 Visualization service output

## 6.6 Composition Of J48 Classifier Web Service and Visualization Web Service

The output of J48 classification Web Service should be the input of the visualization service. BPEL visual editor available in Netbeans 6.7.1 is used to make the Web Services talk to each other. The process flow is shown in figure 6.3.

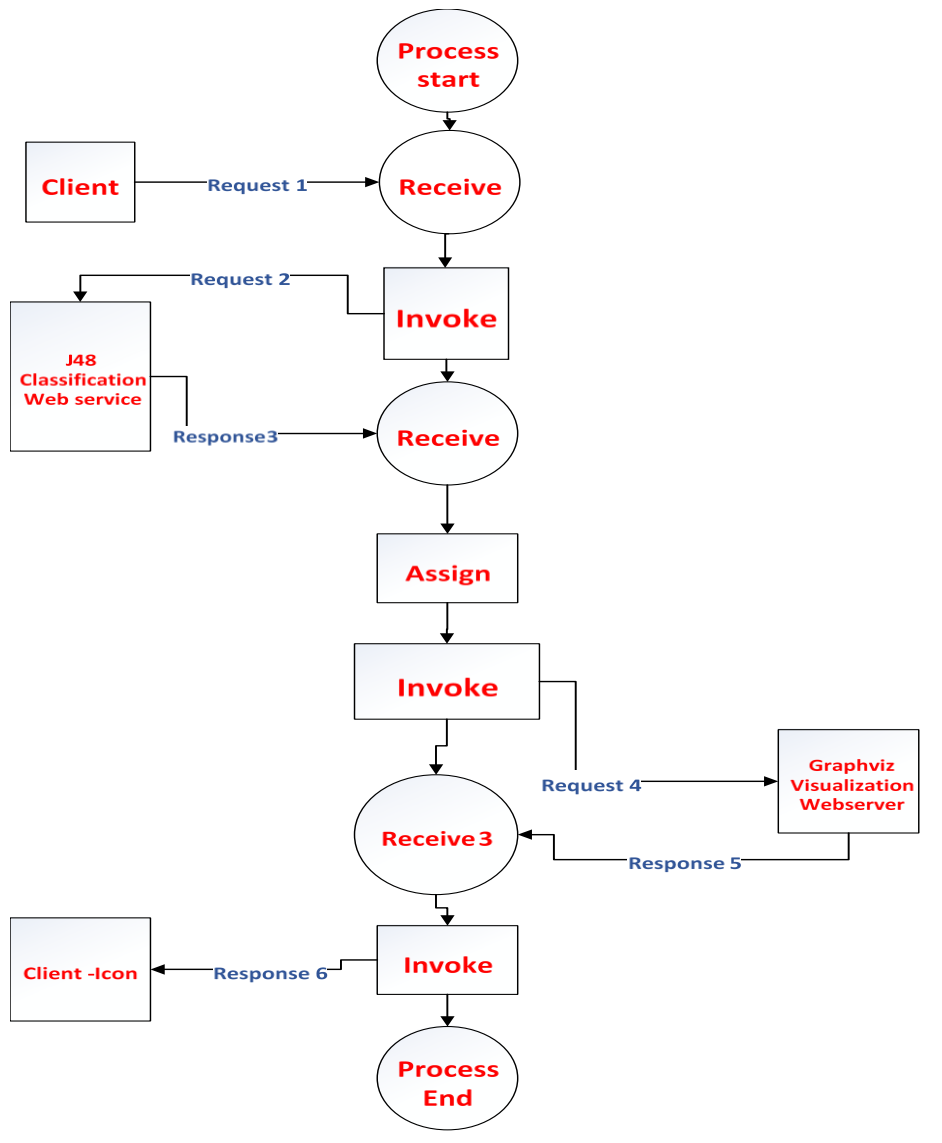


Fig 6.3 BPEL process

The “J48Data” is the data mining service and the “J48Visualize” is the visualization service.

The partner links are described using the WSDL file of both of these services.

First, the data mining service is discovered and called by uploading a file to it.

The output of the “reply” is then “assigned” to the input of the J48Visualization service.

The J48Visualization service is then invoked using the output of the mining service.

The Variable used:

- mineIn: the input to the mining Web Services
- mineOut: the return of the mining Web Services
- operationIn: the input to the visualization service

The BPEL file of the process is shown in figure 6.4.

The composite application of the BPEL module will look like this.

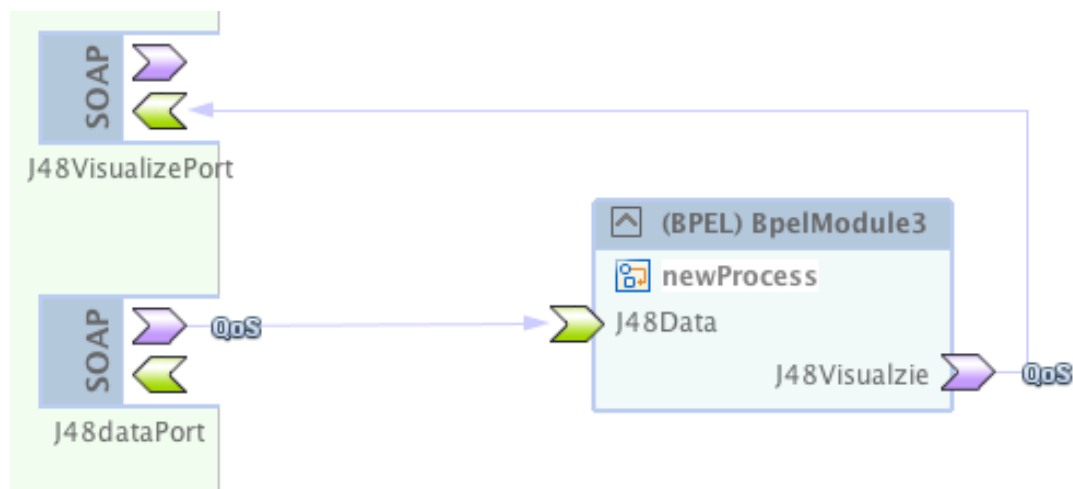


Fig 6.4 Composite application of the BPEL module

The SOAP request is sent from the J48data port which invokes the J48Data service. The output of that service then invokes the J48Visualize service. The code details of this composition Web Service is given in APPENDIX B-3

## 6.6 Summary

In this chapter a novel way of WSS using QoS attributes based on composition of Web Services have been illustrated. J48 classification algorithm is exported as Web Service whose request contain QoS attribute in ARFF. The resultant output is a decision tree rules. The

output of the data mining algorithm is available to us in both machine readable xml format and normal formatted String. The machine readable format of the output can be further processed to mine more data or get more relevant information in the form of visual tree by passing it to another Web Services. The output of the J48 tree data mining service is refined to give the output in dot format. Then pass the output of this service to the visualization service, which in turn will give us the visualized, graphed output of the tree.

A BPEL process is defined to compose above Web Services. First the QoS ARFF file is sent to the data mining service, which then gives the output in the dot format. Then the visualization service is invoked and in the output of the data mining service is assigned as the input for the visualization service. The visualization service then draws the graph and gives tree output.

So far this thesis discussed about selection of Web Service using QoS attribute of SOAP enabled Web Services. there is yet another Web Service known as REST Web Service which is also getting wide popularity in distributed enterprise applications nowadays[75]. If a technique is available to measure the performance metrics of REST and SOAP Web Services under identical operational environment, then that will be of immense help to the clients for choosing the appropriate service for his application in SOA. In the next chapter, an attempt is made to measure the QoS value of both REST and SOAP Web Service for a database driven application.

## **CHAPTER 7**

### **Implementation of Agent for Measurement of QoS parameter using SOAPUI**

Web Services are the most emerging distributed applications which can be published and invoked using standardized protocols over the public or private registries. Due to rapid development of the Web, there are numerous functionally similar Web Services available. To select one among the so many functionally similar Web Services, the non-functional criteria such as Quality of Service (QoS) will be considered [65,66]. However, most of the clients are not experienced enough to acquire the best selection of Web Services based on its described QoS properties.

One of the QoS attributes of Web Services which is considered to be most crucial for Web Services consumers is throughput of the Web Services. In order to create a new repository of quality metric for all the Web Services available in particular functional domain, an agent web service is designed. This will aid the Web Service users to choose the Web Service with high quality of parameter. Contacting this agent web service will assist with analyzed throughput of all Web Services in that functional domain [67]. This agent web service makes use of SOAPUI [95] software for checking the throughput attribute of the Web Services. Throughput is defined as the measure of maximum requests that can be handled in a given unit of time. To analyse the throughput of Web Services, SOAPUI software is used which is explained in the following paragraphs. In order to enable to create a new repository for all the Web Services hosted in any particular functional domain, a novel approach for the measurement of throughput is suggested in this chapter.

## 7.1 SOAPUI

SOAPUI is a free tool for SOAP testing. It has a Graphical User Interface that is very easy and simple to use and it includes advanced features like creation and immediate execution of practical mechanization, security, regressions and load testing. It provides all the following features.

- All inclusive performance testing.
- Easy to use GUI.
- Efficient service simulation.
- Comprehensive reporting features.
- Advanced functionality.

## 7.2 Features of SOAPUI

**Service Simulation:** Mock Services enable us to generate and reproduce strong and multilayered tests of Web Services before they are implemented. The users can now test and use the services without constructing them.

**Functional Testing:** It provides automated functional, security regression and performance testing and enables the creator to certify their applications.

**Analytics:** Helps in making extensive testing reports that can be transferred to any system and made to order.

**Load Testing:** Provides the most sophisticated testing mechanisms for efficient and extensive testing of Web Services.

**Performance Testing:** Performance testing involves an extensive array of tasks in Web Services testing and is an area of great perplexity.



Load testing (or Performance Testing) is an essential part of the creating cycle of any service and ensure that the applications are capable enough of handling extensive loads in agreed environment resources. Performance Testing measures the number of requests that can be handled by a service. Performance testing is usually done by two methods- load testing and stress testing. Load testing involves confirming the agreed response within average limits and Stress testing involves pushing the application beyond the limit to confirm its behavior.

### **7.3 Agent Implementation using SOAPUI**

Client wants to use Weather Forecast Web Service with the best available throughput. Now Agent checks the UDDI registry to collect all the Weather Forecast Web Services. Performance testing on these Web Services is done using SOAPUI tool, the log file generated by the software is converted to mysql database and information is presented to the client via his interface which aids the client to select the best Web Service in terms of throughput in the Weather Forecast domain.

SOAPUI runs the load test with as many threads as the system can handle. Numerous loads can be easily run side by side to verify more advanced situations which may arise in real time scenario Weather Forecasting Agent Implementation using SOAPUI is shown in figure 7.1.

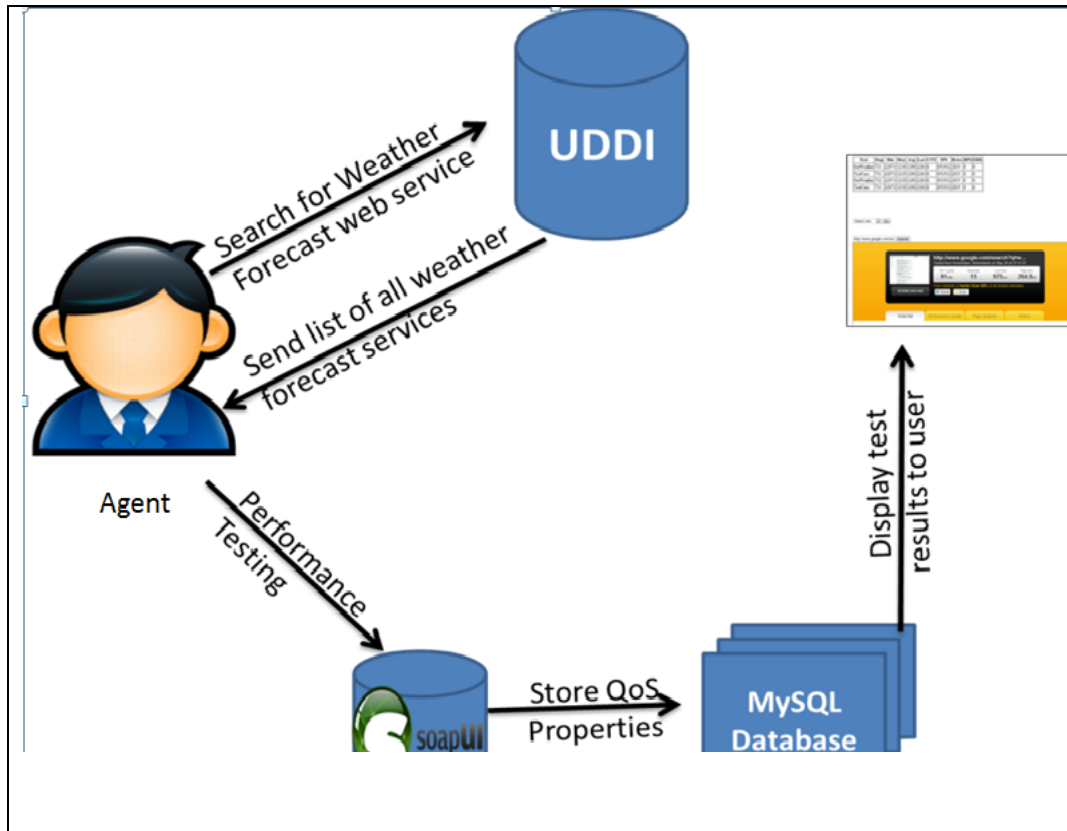


Fig 7.1 Weather Forecasting Agent Implementation using SOAPUI

### 7.3.1 Weather Forecast Service- Example Service

The Weather API's mentioned below takes the City name or the Zip code as the constraints and then sends the weather forecast as the response

Weather Forecast API : <http://www.websvcx.net/globaIweather.asmx?WSDL>

### 7.3.2 Creating Project For Sample API

**Step 1:** Start SOAPUI

**Step 2:** Create a new project by selecting File → New Project. The window for new Project Dialogue box looks as shown in figure 7.2.

**Step 3:** API name and WSDL / Service name has to be entered in the window that opens.

**API Name:** Weather API-1

**WSDL Link:**<http://www.webservices.net/globalweather.asmx?WSDL>

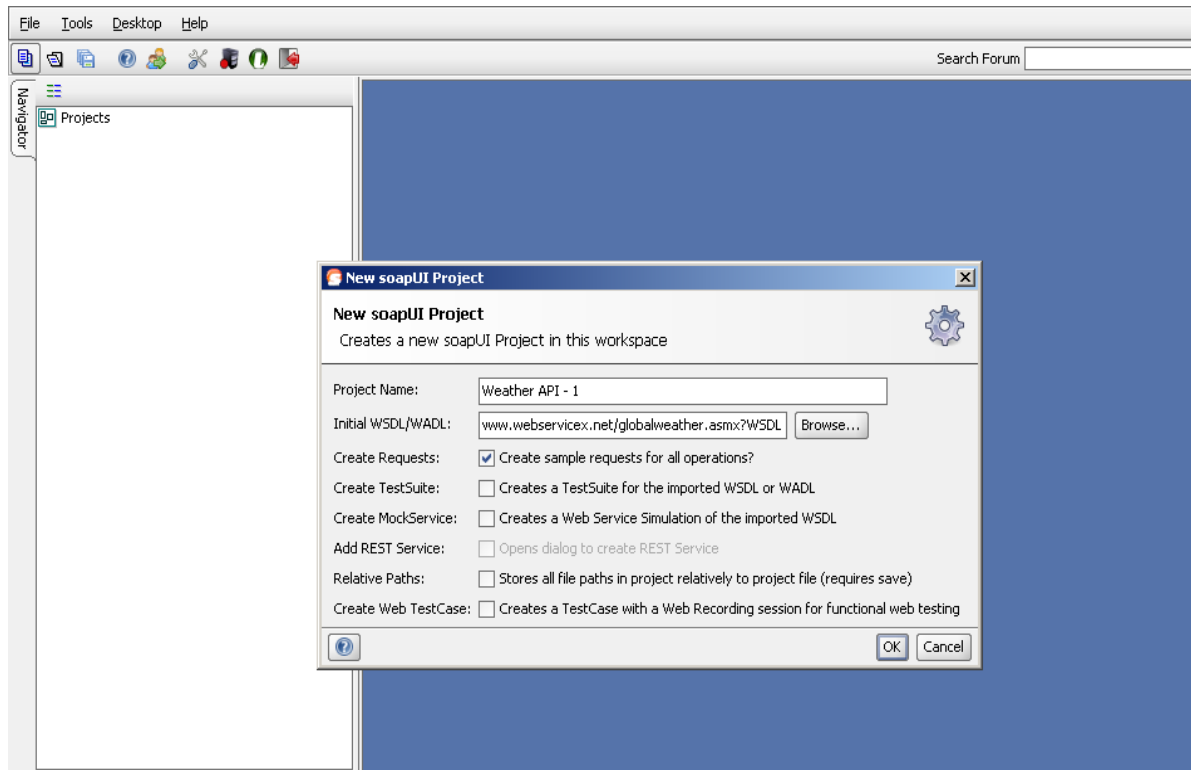


Fig 7.2 Snapshot of New Project Dialogue box

The API is authenticated to check if the path is valid. The Snapshot of Load Test Suite is shown in figure 7.3

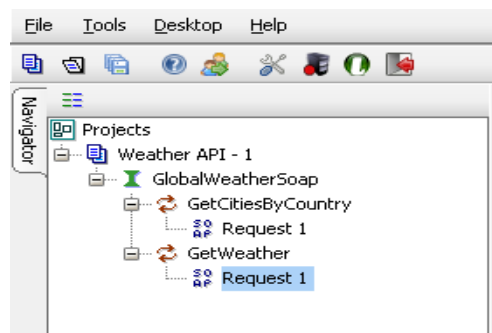


Fig 7.3 Snapshot of Load Test Suite

### 7.3.3 Generating Load TestSuite

Right click on *GlobalWeatherSOAP* and then Generate TestSuite. Then select GetWeather API and then check the **Generate LoadTest**, check box as shown in figure 7.4.

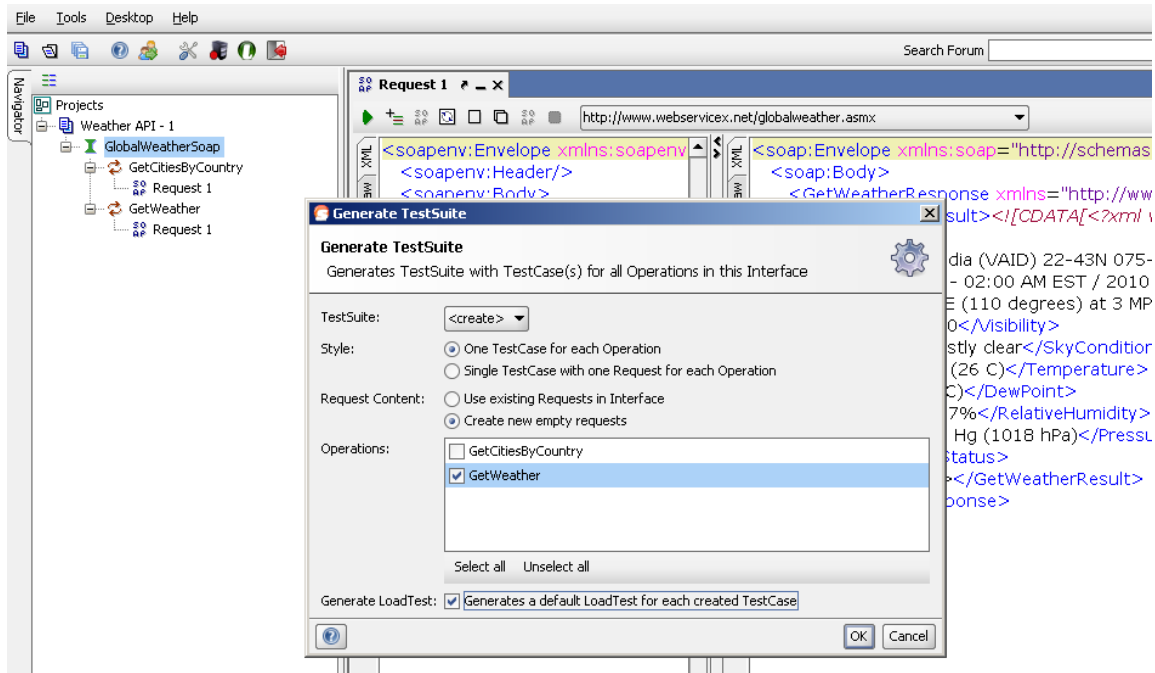


Fig 7.4 Snapshot of Generate Test Suite

### 7.3.4 Creating an Average Load Test Case for API

Now let us generate a load test under average load

Enter the Virtual Users / Threads- **15**

Configure the test to run for **300 Seconds**

Hence, Request per second=  $15/300 = 0.05$  requests/second

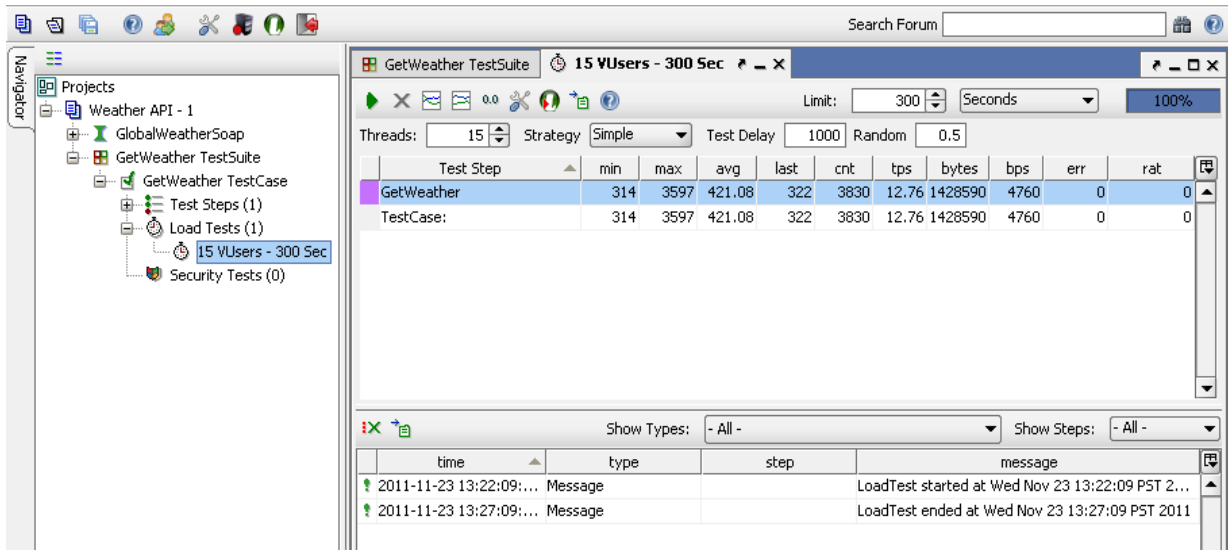


Fig 7.5 Snapshot of Run Test results

Similarly SOAPUI is used to conduct load test for other two weather API. Snapshot of Run Test results is shown in figure 7.5

**Weather API - 2:** <http://www.websvicex.net/usweather.asmx?WSDL>

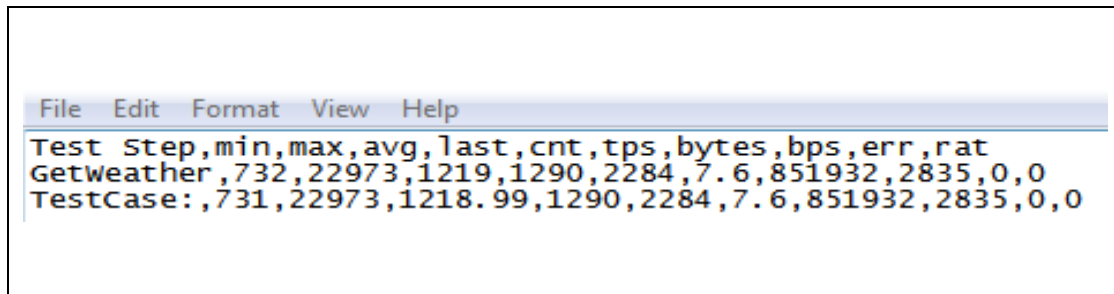
**Weather API-3:** <http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL>

The Load test statistics can be exported to a CSV or Text file format. The CSV format is preferred since the PHP code can easily identify the individual records in a .CSV file and copy them in a MySQL database.

## 7.4 Functioning Model of Agent Architecture

### 7.4.1 DATABASE CREATION IN MYSQL USING PHPScript

SOAPUI creates a .csv(Comma separated values) file which contains the performance statistics for the load test conducted on a Web Services. The file is in the format shown below in the figure 7.6.

A screenshot of a text editor window with a menu bar containing 'File', 'Edit', 'Format', 'View', and 'Help'. The text area contains three lines of CSV data: a header row with column names, a row for 'Getweather', and a row for 'TestCase:'.

```
File Edit Format View Help
Test Step,min,max,avg,last,cnt,tps,bytes,bps,err,rat
Getweather,732,22973,1219,1290,2284,7.6,851932,2835,0,0
TestCase:,731,22973,1218.99,1290,2284,7.6,851932,2835,0,0
```

Fig 7.6 Load test Statistics Log File

A way needed to be found to enable the data in the log file to be stored in a database so that performance statistics of many Web Services could be logged and then presented to the user on request. A MySQL database is easy to generate and gives the flexibility of writing complicated queries to manipulate the data.

A php script was written to extract the data in the log file and put in a MySQL database created using phpMyAdmin and WAMP server.

PHP script offers easy extraction from .CSV files. First the contents of the *statistics.csv* file are copied the array *\$rows*. PHP automatically recognises that the first line of the csv file is the column heading and the subsequent rows are the record values.

The array is then traversed row by row and an sql query is generated by appending the \$q string with these QoS values. Every time the php code is run, the following SQL commands are generated automatically as shown in figure 7.7

```
INSERT INTO max_server(test_step,min,max,avg,last,cnt,tps,bytes,bps,err,rat) VALUES('GetWeather','732','22973','1219','1290','2284','7.6','851932','2835','0','0')
INSERT INTO max_server(test_step,min,max,avg,last,cnt,tps,bytes,bps,err,rat) VALUES('TestCase','731','22973','1218.99','1290','2284','7.6','851932','2835','0','0')
```

Fig 7.7 Automatically generated SQL query

The following table is generated in the PHPMYAdmin after the execution of the sql queries. The table can be edited in PHPMYAdmin itself. SQL table generated in PHPMYAdmin is shown in figure 7.8

	id	test_step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
<input type="checkbox"/> Edit Inline Edit Copy Delete	1	GetWeather	731	22973	1219	1290	2284	8	851932	2835	0	0
<input type="checkbox"/> Edit Inline Edit Copy Delete	2	TestCase:	731	22973	1219	1290	2284	8	851932	2835	0	0
<input type="checkbox"/> Edit Inline Edit Copy Delete	3	GetWeather	732	22973	1219	1290	2284	8	851932	2835	0	0
<input type="checkbox"/> Edit Inline Edit Copy Delete	4	TestCase:	731	22973	1219	1290	2284	8	851932	2835	0	0

Fig 7.8 SQL table generated in PHPMYAdmin

#### 7.4.2 EXTRACTING QoS ATTRIBUTES FROM MySQL DATABASE

In this implementation of the Agent architecture, the user would be given a detailed description of the QoS parameters of the various Web Services in the functional domain of his interest. The user choose the appropriate Web Service based on his QoS requirement

The values of the parameters would be extracted from the database that was created above and would be displayed to the user. The frontend of the application for this purpose used

PHP and HTML for this purpose. The final view of the frontend of the application is shown below in figure 7.9.

Test	Step	Min	Max	Avg	Last	CNT	TPS	Bytes	BPS	ERR
GetWeather	731	22973	1219	1290	2284	8	851932	2835	0	0
TestCase:	731	22973	1219	1290	2284	8	851932	2835	0	0
GetWeather	732	22973	1219	1290	2284	8	851932	2835	0	0
TestCase:	731	22973	1219	1290	2284	8	851932	2835	0	0

← Table generated using php code

Fig 7.9 Snap shot of database obtained.

The frontend has been developed using PHP and HTML. The table is then populated and displayed in the HTML body by traversing the array. When client contacts the agent to know about which is the best Web Services in terms of throughput, then it could see the output generated by the agent using SOAPUI as in the figure 7.10 below, based on which client could get to know about which is the best Web Services in terms load test.

### Best API

API Name	Threads	Seconds	Request/Sec
Weather API – 1	236	60	4.38
Weather API – 2	164	60	2.73
Weather API – 3	184	60	3.06

Fig 7.10 Snapshot of application front end

## 7.5 Summary

Due to the recent boost in the popularity of web based services, there has been a sudden increase in the number of services that provide similar functionalities. These services can only be distinguished on the basis of their QoS properties. These properties are not considered by many service selection mechanisms.



In this chapter an attempt is made to measure important QoS parameter in real time and to create a repository of the same using an agent approach. The agent in turn uses SOAPUI to perform performance testing, results from SOAPUI are stored in the agent, when the user look for a Web Service in a functional domain the agent present the client with the available QoS parameters using which the client can make selection.

## **CHAPTER 8**

### **Analysis of QoS attributes for REST and SOAP Web Services**

Web Services are internet enabled software components. By composing Web Services integration of Enterprise applications can be achieved. Web Services are being used extensively by many enterprises like IBM, Microsoft and oracle every day. SOAP and REST are very popular types of Web Services. A Hospital Management Application is built using Web Services that can implement functions like online appointment booking, preparing a prescription online, online room reservation, storing patient's medical history etc. The application is designed and developed using MySQL workbench, Netbeans, Eclipse, Google SQL cloud and Google App Engine.

Two different methods are considered for developing a hospital management application. One method implements this application as REST Web Service and the same is hosted as SOAP Web Services within the server running on local host. The application designed and developed using the second method is deployed to the Google App Engine to make it available online and to facilitate online transactions via the internet. The scalability testing of the cloud enabled hospital management system has also been done using a custom developed software tool.

#### **8.1 REST Web Services**

REST, Representational State Transfer, is an architectural style of network design. The resources in a RESTful Web Services are invoked using an URI[85](Unique Resource Identifier) commonly an URL [86] (Uniform Resource Locator). A representation of the requested resource is returned and the client is said to be in one state.

When the client accesses other resources present in one resource, through hyperlinks, it is said to change state. The client changes its state by progressing through the network of web pages. Hence this system is known as Representational State Transfer.

### 8.1.1 Characteristics of REST

#### Characteristics of REST

- Platform Independent
- Language Independent
- Simpler when compared to SOAP
- Implements HTTP methods
- Representation of resources is generated by a URL i.e. every resource in the Web Services is named
- Representations of resources are interconnected thereby enabling state transfer i.e. client progresses through the network of web pages and transfers from one state to the other
- Stateless – When a client sends a request it should provide all the details required to understand and process the request and should not depend on the server's memory for successful completion of the request
- REST uses the HTTP methods GET,POST,PUT,DELETE to perform the CRUD(Create, Read, Update and Delete) operations
  1. GET method is similar to READ
  2. POST method is similar to Update
  3. PUT is similar to Create
  4. DELETE is similar to DELETE

### 8.1.2 Example of REST Web Services

Consider a project CustomerDB which contains a Customer Database with customers and discountCodes tables. Every customer can be identified by a unique ID number. The details of the customers are stored in the customers table. Now to access a resource in the customer table an URI should be specified.

URI is: <http://localhost:8080/CustomerDB/resources/customers/{customerID}>

To access any resource in the discountCodes table an appropriate URL should be specified.

URI: <http://localhost:8080/CustomerDB/resources/customers/{customerID}/discountCodes/>

The URI <http://localhost:8080/CustomerDB/resources/customers/> gives the details of all the customers in the customers table. The client is in one state now. When the client accesses a particular resource in the customers table through the URI <http://localhost:8080/CustomerDB/resources/customers/{customerID}> the client transfers its state. This is called State Transfer.

## **8.2 Design and Implementation Of Database driven Hospital Management System using Web Services**

The performance analysis on REST and SOAP Web Services is carried out with the help of Hospital Management application can be used for better functioning of the Hospital. This application can be developed to enable online booking of appointments, enable doctors to view his/her patient's list, prepare an online prescription and view the patient's medical history. It can also be developed to allow patients to login to their account and view their appointment time, medical history, prescriptions and other details. Hospital Management applications are very useful and are also used widely in hospitals.

In the following sections database driven hospital management Web Services have been developed using different methods. The developed Web Services can implement all/some of the following functions

- Display the details of doctors
- Display the details of patients
- Display the details of rooms
- Staff login
- Patient login
- Online appointment booking

- Preparing online prescription
- Viewing patient's history

### **8.2.1 Tools used**

The following tools were used to create the Hospital Management Web Services

- Netbeans 6.9.1- This has been used for creating the SOAP and REST Web Services
- MySQL Workbench- This has been used to create a relational database for the Hospital Management Web Services
- Eclipse Indigo- This has been used to create a REST Web Services using Entity classes and objects. This has been used to develop a database driven Web Services
- Google SQL Cloud- This has been used to create a traditional relational database for the Hospital Management Web Services
- Google App Engine- The applications developed were deployed to the Google App Engine

### **8.2.2 Methods**

Two methods have been used to develop a Hospital Management Web Services.

**METHOD 1:** Used to create a database driven Web Services

1. Creating a database in MySQL Workbench
2. Using that database in Netbeans to create SOAP and RESTWeb Services

**METHOD 2:** Used to create a database driven Web Services

1. Creating a database in Google SQL cloud
2. Authorizing an application in the Google App Engine to access the database created in the Google SQL cloud
3. Establishing a connection to the database in the code written to implement the Web Services
4. Hosting the application on Google cloud

### 8.2.3 Simple Hospital Management Database in MySQL

In this chapter database driven Web Services have been created using MySQL Workbench and Netbeans. A simple Hospital Management Database was created in MySQL Workbench. Simple SOAP and REST Web Services have been created to display the details of every table in the database created in MySQL Workbench.

The Database created in MySQL has been named **mgmt**. The tables in the database mgmt are

1. Staff
2. Patients
3. addresses
4. staff\_addresses
5. patient\_addresses
6. patient\_rooms

#### Details of the tables created

1. Staff table

This table contains details of all staff members working in the hospital. The table contains columns staff\_id, gender, staff\_job\_title, staff\_first\_name, staff\_middle\_name, staff\_last\_name, staff\_qualifications, staff\_birth\_date and other\_staff\_details

2. Patients table

This table contains details of all patients. The table has columns patient\_id, gender, date\_of\_birth, patient\_first\_name, patient\_middle\_name, patient\_last\_name, height, weight, home\_phone, work\_phone, cell\_monile\_phone and other\_detail

3. Table addresses

Columns:

address\_id, line\_1\_number\_building, line\_2\_number\_street, line\_3\_area\_locality, city, zip\_postcode, state\_province\_county, country and other\_address\_details

4. Table staff\_addresses

Contains details of staff members. Columns: staff\_id, address\_id, date\_address\_from, date\_address\_to, Staff\_staff\_id, addressed\_address\_id

5. Table patient\_addresses

Columns:

patient\_id, address\_id, join\_date, exit\_date, Patients\_patient\_id, addresses\_address\_id

6. Table patient\_rooms

Columns: patient\_id, room\_id, date\_stay\_from, date\_stay\_to, Patients\_patient\_id [8]

#### **8.2.4 ER Diagram of the database model**

ER Diagram shows the relation between the tables of a database. The primary key of each table is also indicated in the ER diagram. Figure 8.5 shows the ER diagram of the database which has been created in MySQL workbench.

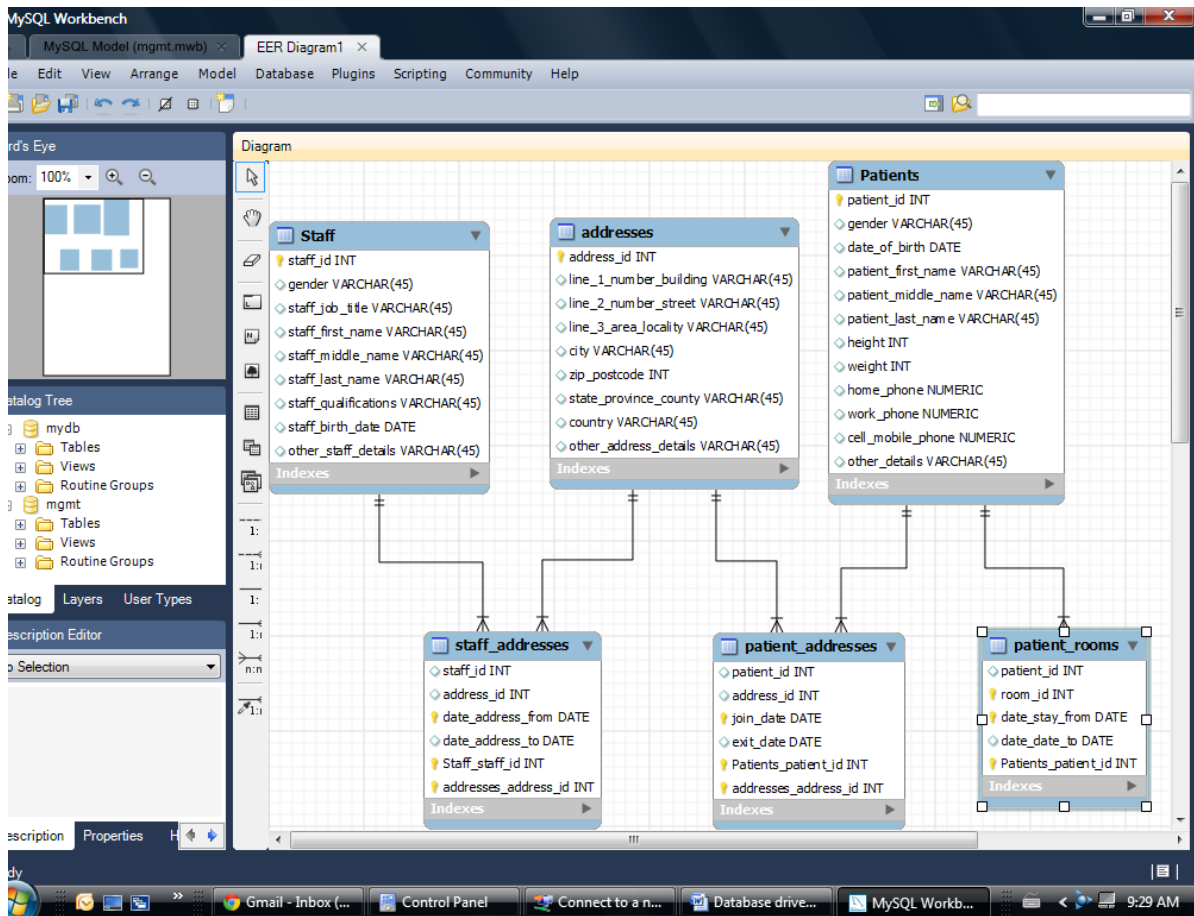


Fig8.5ER Diagram of the database model created in MySQL Workbench

### 8.2.5 Implementation Of Hospital Management System Using REST Web Service

The database created in MySQL workbench is imported to Netbeans[92] and is used to create a REST Web Services. In a REST Web Services all resources are accessed by specifying URI's. Hence every resource has a path defined [93].

For example, the table addresses has the path “/addresses/”.

In order to access the contents of the table addresses the URL:<http://localhost:8080/WebApplication/resources/addresses> should be used. This URI



reads the contents of the table addresses and returns the details in XML format. Figure 8-6 shows a part of the code which defines the path of the resource.

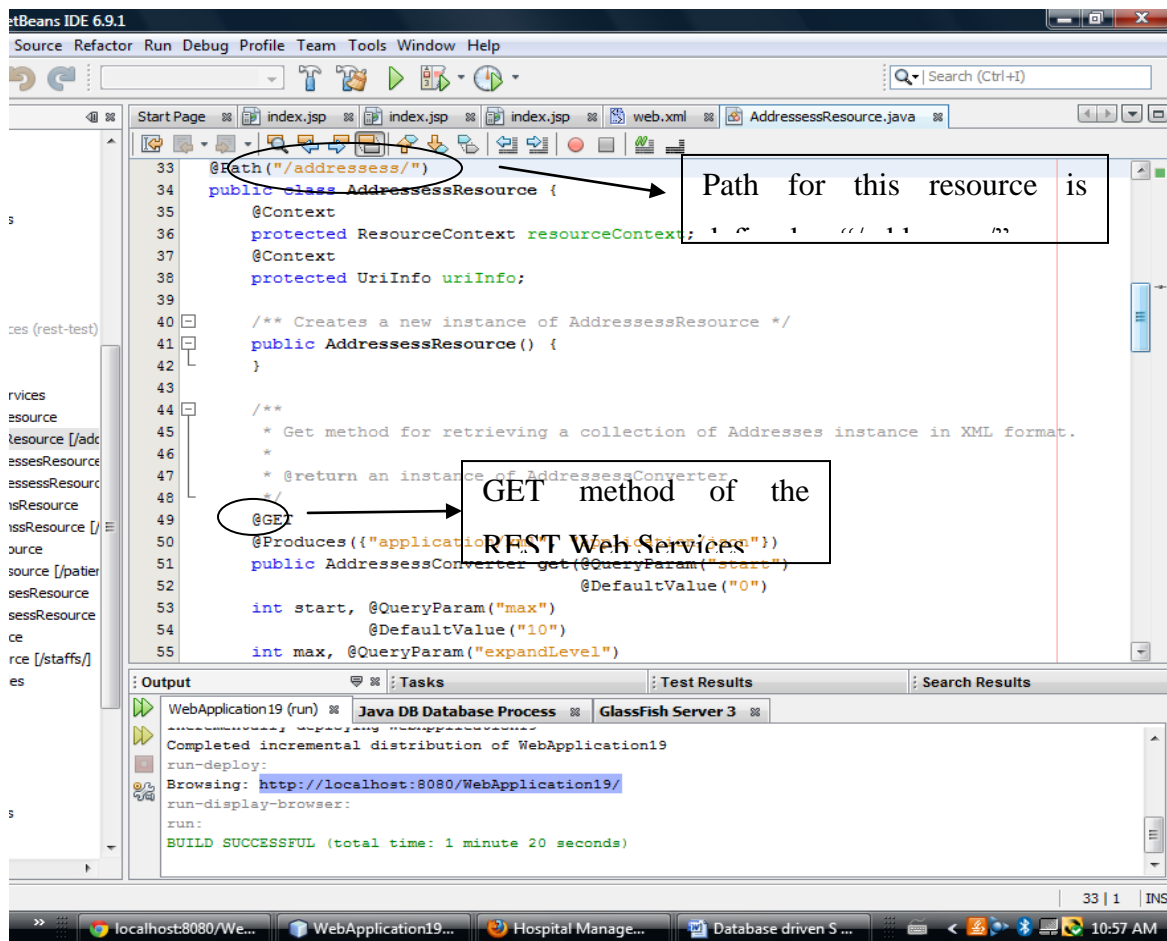


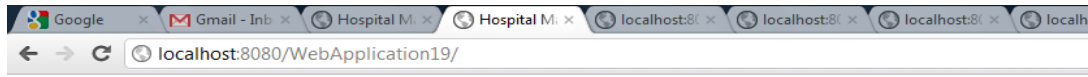
Fig 8.6 Part of code showing the path definition and the GET method

Similarly every other table has an URI and every resource in that table also has an URI.

## 8.2.6 OUTPUT of the REST Web Services

index.jsp

This page displays the list of links which can be used to view the details of patients, staff, addresses and rooms which is shown in figure 8.7.



## Links

To see the wadl doc

[click here](#)

To see the staff list

[click here](#)

To see the patients list

[click here](#)

To see the rooms record

[click here](#)



Fig 8.7 Output of the REST Web Services- index.jsp file

## WADL document

The first link in the index.jsp page gives access to WADL document. The WADL for the current Web Services is shown in figure 8.8.



Fig8.8WADL of the Web Services

## Staff details

The Staff table in the database can be accessed by specifying its path as `/staff`. When the URI is invoked the data present in the Staff table appears in XML format. Style sheet can be used to represent the data in tabular form. Refer to Figure 8.9.

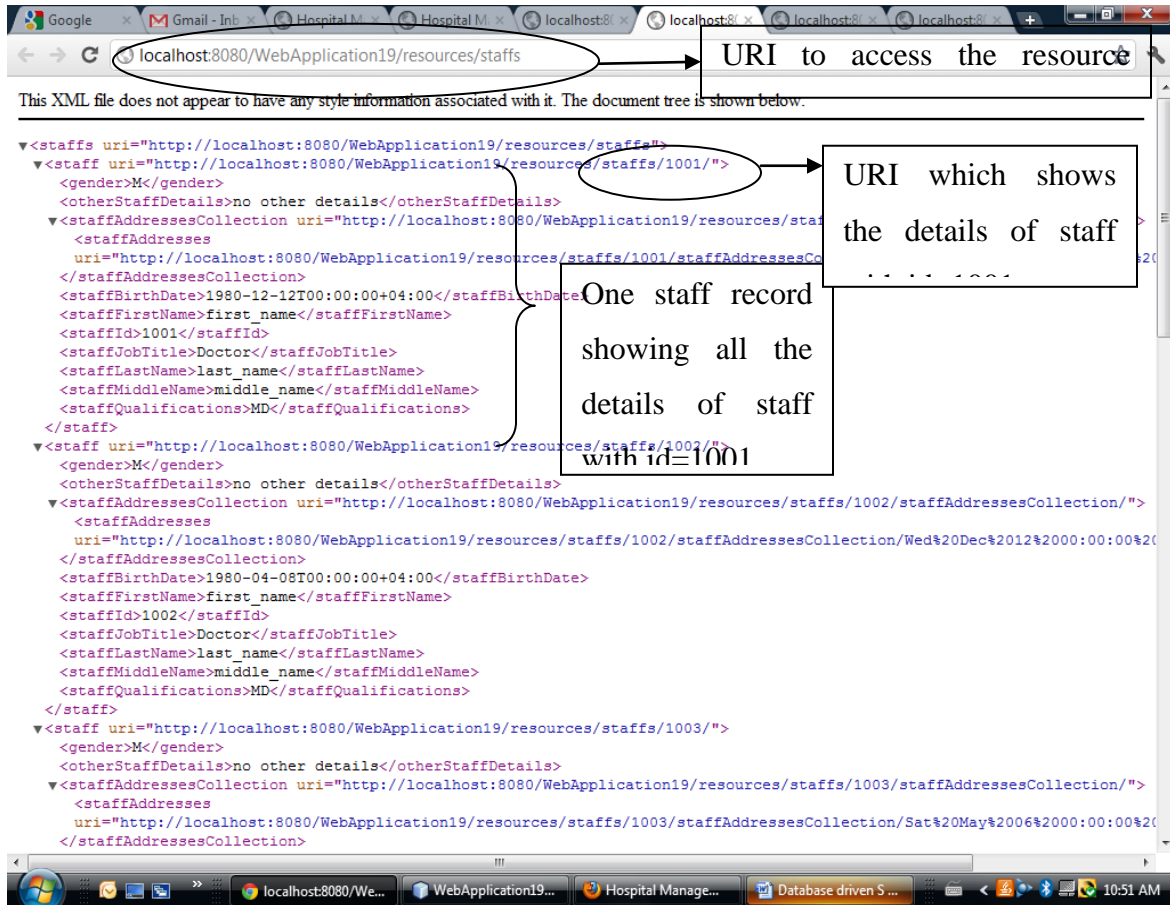


Fig 8.9 Page displayed when URL `http://localhost:8080/WebApplication19/resources/staffs` is accessed

### **Patient details**

The Patients table in the database can be accessed by specifying its path as **/patientss**. When the URI is invoked the data present in the Patients table appears in XML format.

### **Patient Room details**

The patient\_rooms table in the database can be accessed by specifying its path as **/patientRoomss**. When the URI is invoked the data present in the patient\_rooms table appears in XML format.

### **Address details**

The addresses table in the database can be accessed by specifying its path as **/addressess**. When the URI is invoked the data present in the addresses table appears in XML format.

### **Staff Address details**

The staff\_addresses table in the database can be accessed by specifying its path as **/staffAddressess**.

### **Patient Address details**

The patient\_addresses table in the database can be accessed by specifying its path as **/patientAddressess**. When the URI is invoked the data present in the patient\_addresses table appears in XML format.

## **8.2.7 LOAD test for the REST Web Services**

LOAD test is done to measure the performance of a Web Services when multiple users are accessing it. SOAPUI is the testing tool used for this purpose. The Web Services was tested for 5, 25, 50 & 100 threads and the observations were noted. The observations can be seen in Tables 8.1&8.2.

Observation tables

**Resource Method: /addresses/ GET-> get**

**Table 8.1** Observation Table for LOAD test-1

Load Test	Thread Count	Min	Max	Avg	Last	cnt	tps	Bytes	bps
1	5	5	3815	200.01	5	311	5.01	2769455	44662
2	25	5	1292	75.06	17	1623	26.66	14452815	237476
3	50	5	1522	93.02	47	3084	51.03	27463020	454452
4	100	5	1589	91.05	7	5961	97.38	53082705	867251

where,

Min: Shortest time the step has taken

Max: Longest time the step has taken

Avg: Average time the step has taken

Last: last time the step has taken

cnt: number of times the step has been executed

tps: transactions per second

bytes: number of bytes processed

bps: bytes processed per second [4]

The Graph for LOAD Test is shown in figure 8.4

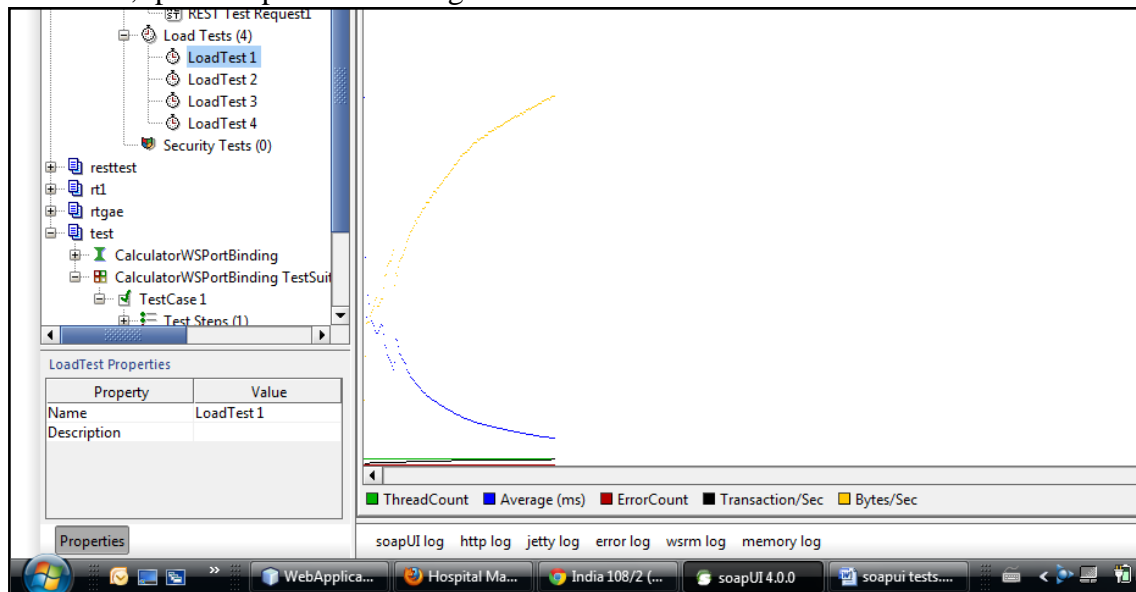
**Resource Method: /staff/ GET-> get**

**Table 8.2** Observation table for LOAD test-2

Load Test	Thread Count	Min	Max	Avg	Last	cnt	tps	Bytes	bps
1	5	1	919	13.39	2	311	5.01	320330	5165
2	25	1	1333	53.08	13	1623	26.66	1671690	27467
3	50	1	1427	61.76	30	3084	51.03	3176520	52564
4	100	1	1481	64.16	2	5961	97.38	6139830	100310

## Graphs

Graphs for the LOAD test observations tabulated in the previous section are generated by SOAPUI. These screen shots of graphs are shown in Figures 8.10, 8.11, 8.12, and 8.13.. The graphs are drawn for the thread counts 5,25,50 and 100 respectively, from the graph it is inferred that the transactions per second (tps) which is shown in black color is increasing according to the thread count. Similarly bytes per second which is in yellow color is decreasing as the thread count increases. From the statics graph with default options, it is very clear that how the average response time, tps and bps values change as the thread count increases.



**8.10** Graph for LOAD test 1

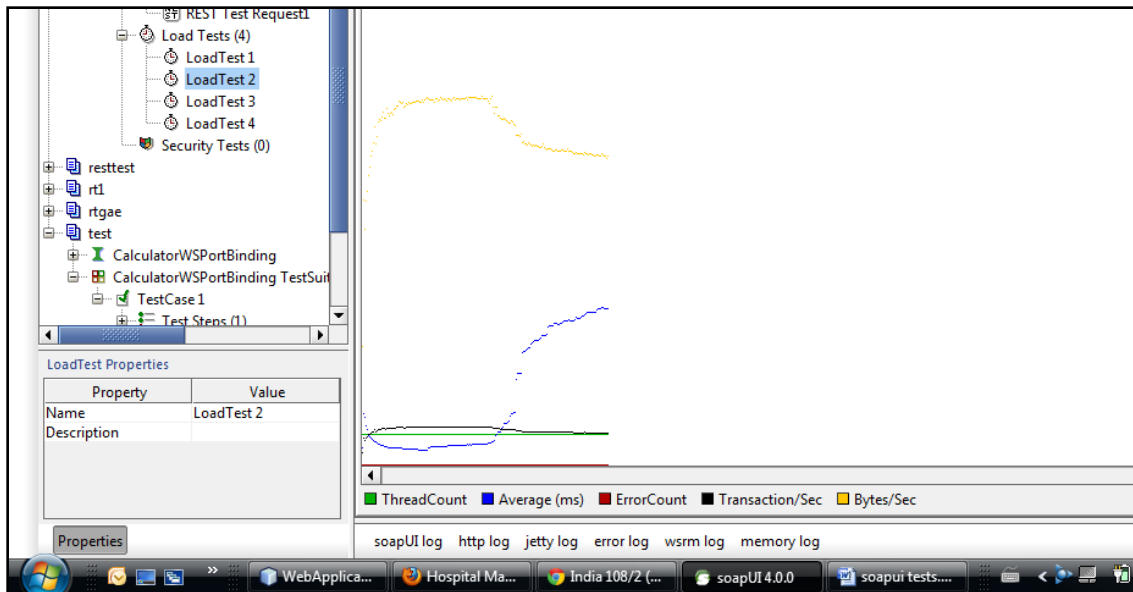


Fig8.11 Graph for LOAD test 2

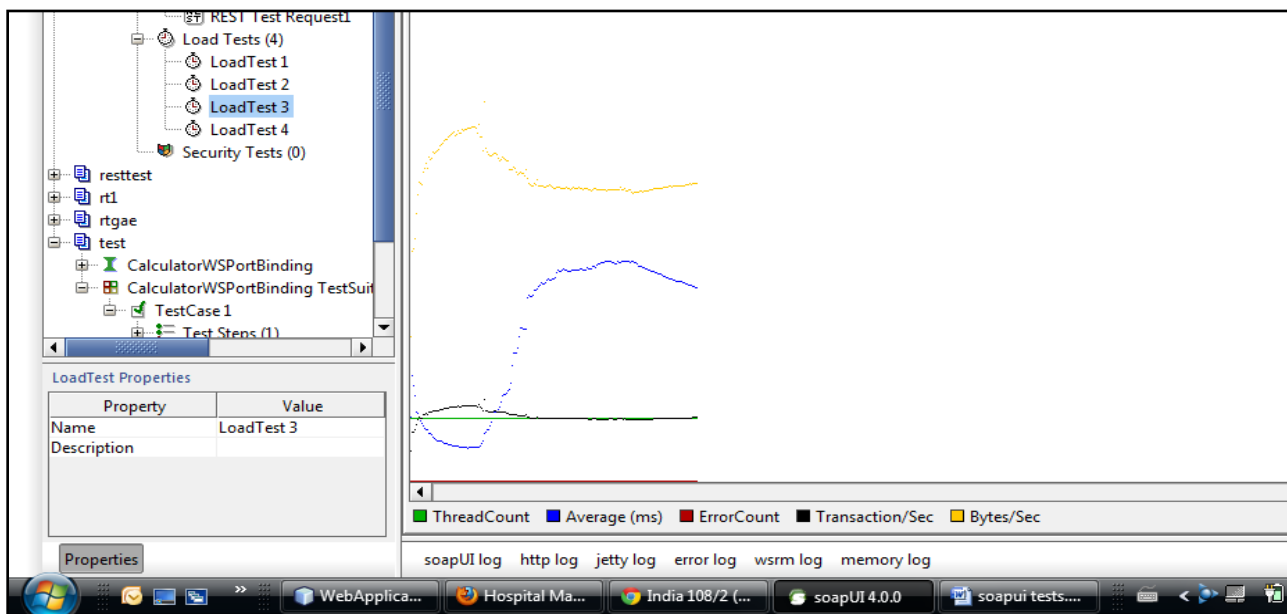


Fig 8.12 Graph for LOAD test 3



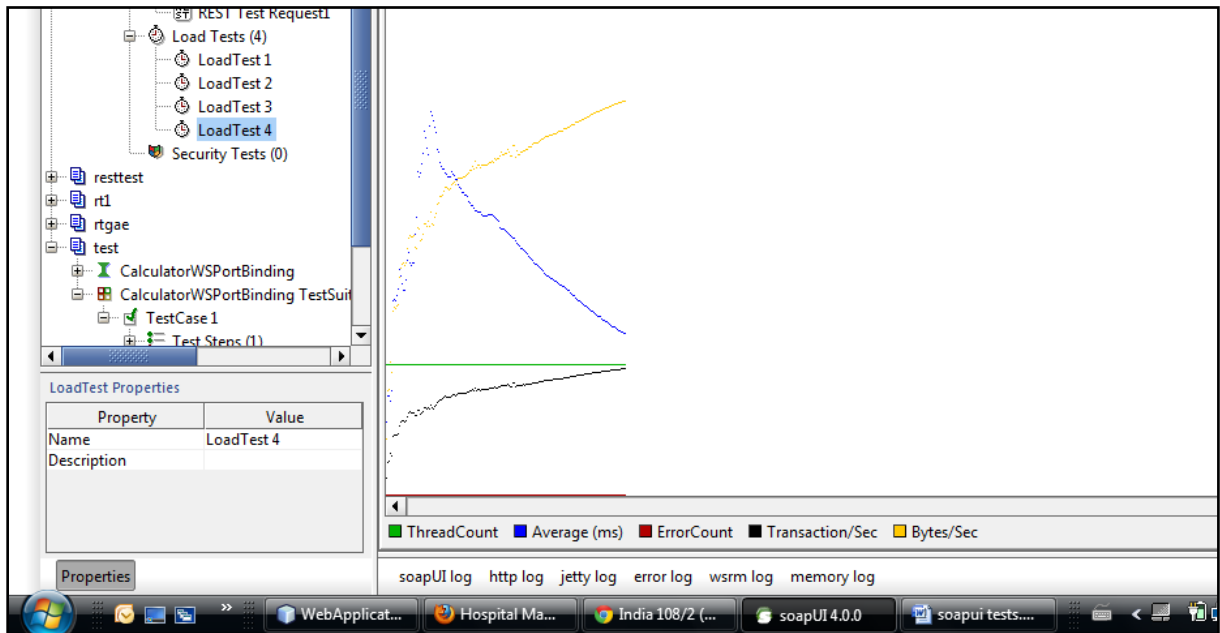


Fig 8.13 Graph for LOAD test 4

## 8.2.8 Implementation Of Database driven Hospital Management Using SOAP Web Services

The database used for the REST Web Services is used to develop a SOAP Web Services[88]. Web operations are defined to display the data in the tables Staff, Patients, patient\_rooms, addresses, patient\_addresses and staff\_addresses. Refer to Figure 8.14

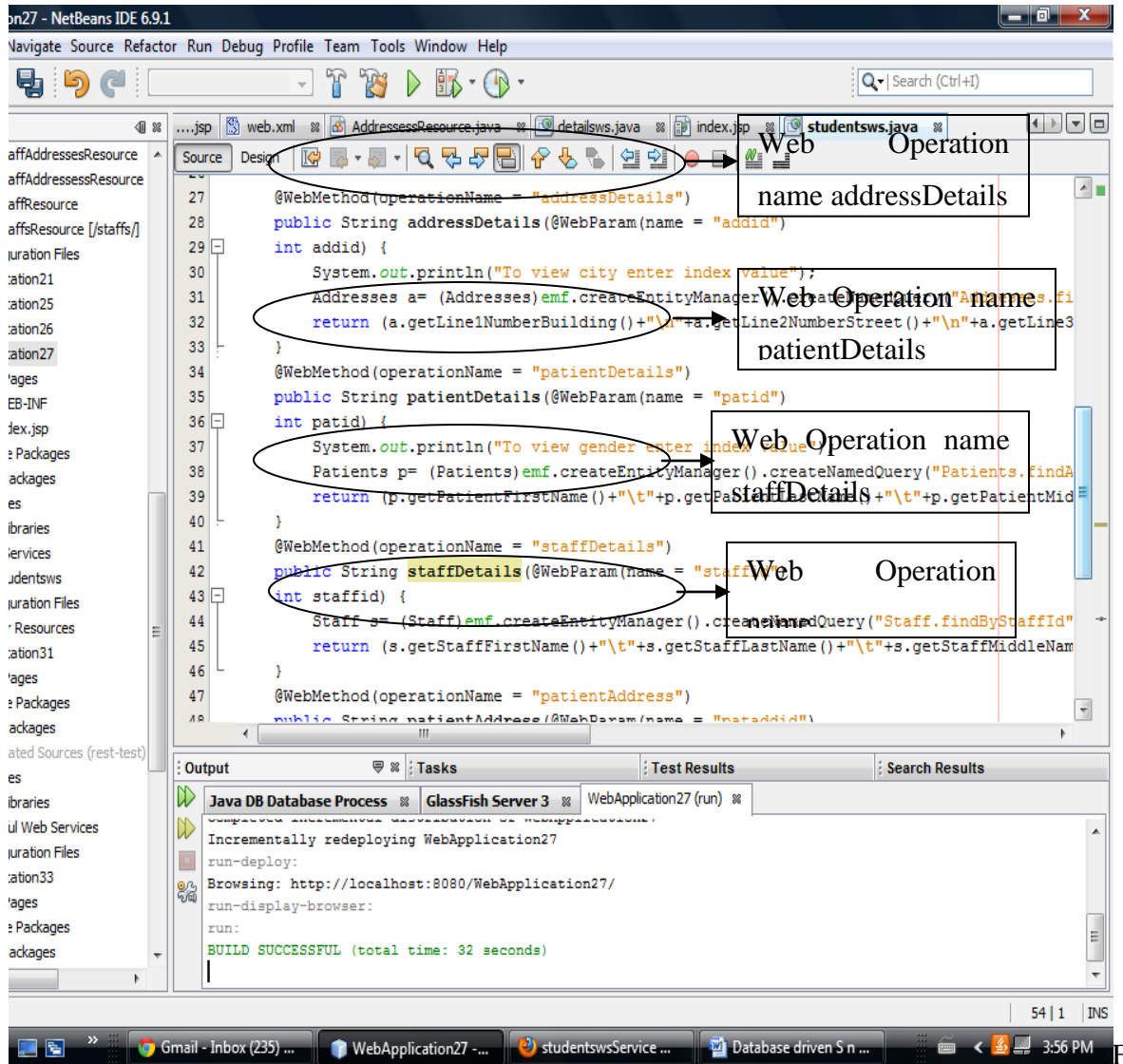


Fig8.14 Web Operations of a SOAP Web Services

## 8.2.9 OUTPUT of database driven SOAP Web Services

Test page: When the SOAP Web Services is tested the following page opens in the browser as shown in figure 8.15.

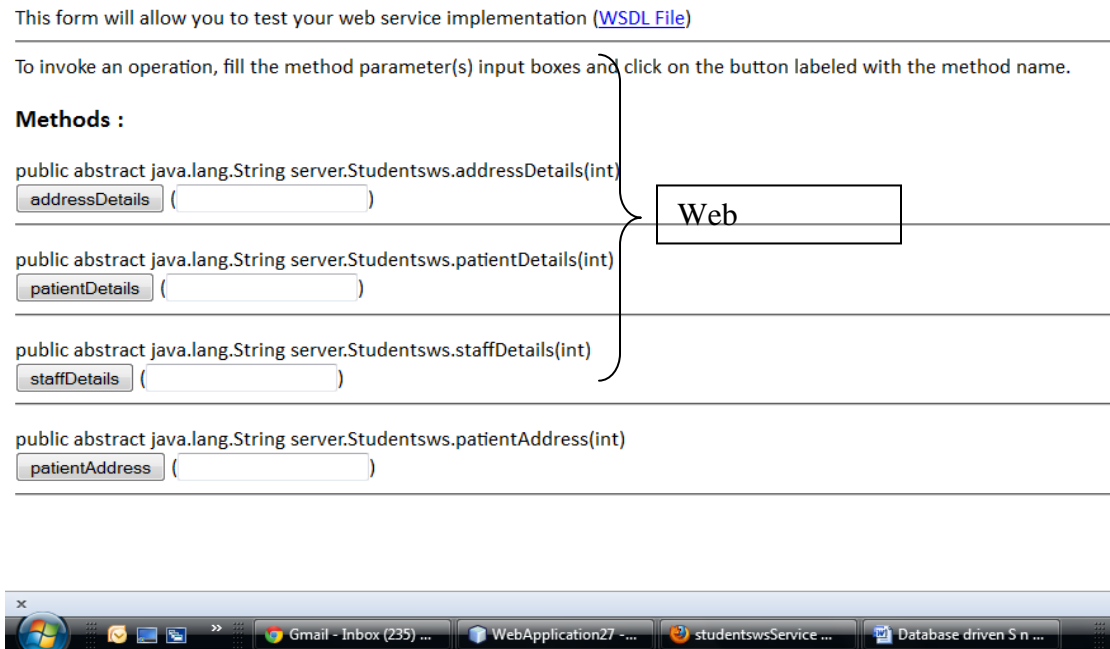


Fig8.15 Output of the SOAPWeb Services when it is tested

Address details

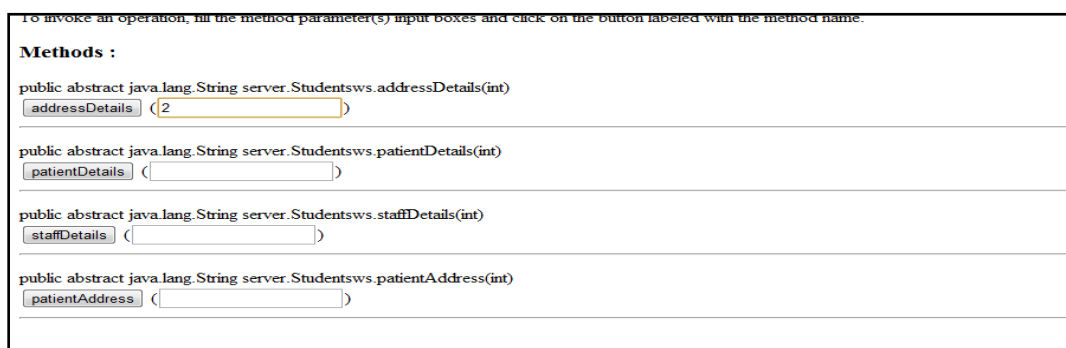


Fig 8.16 Output on entering the address id in the web operation

Output on entering the address id in the web operation can be seen in Fig 8.16

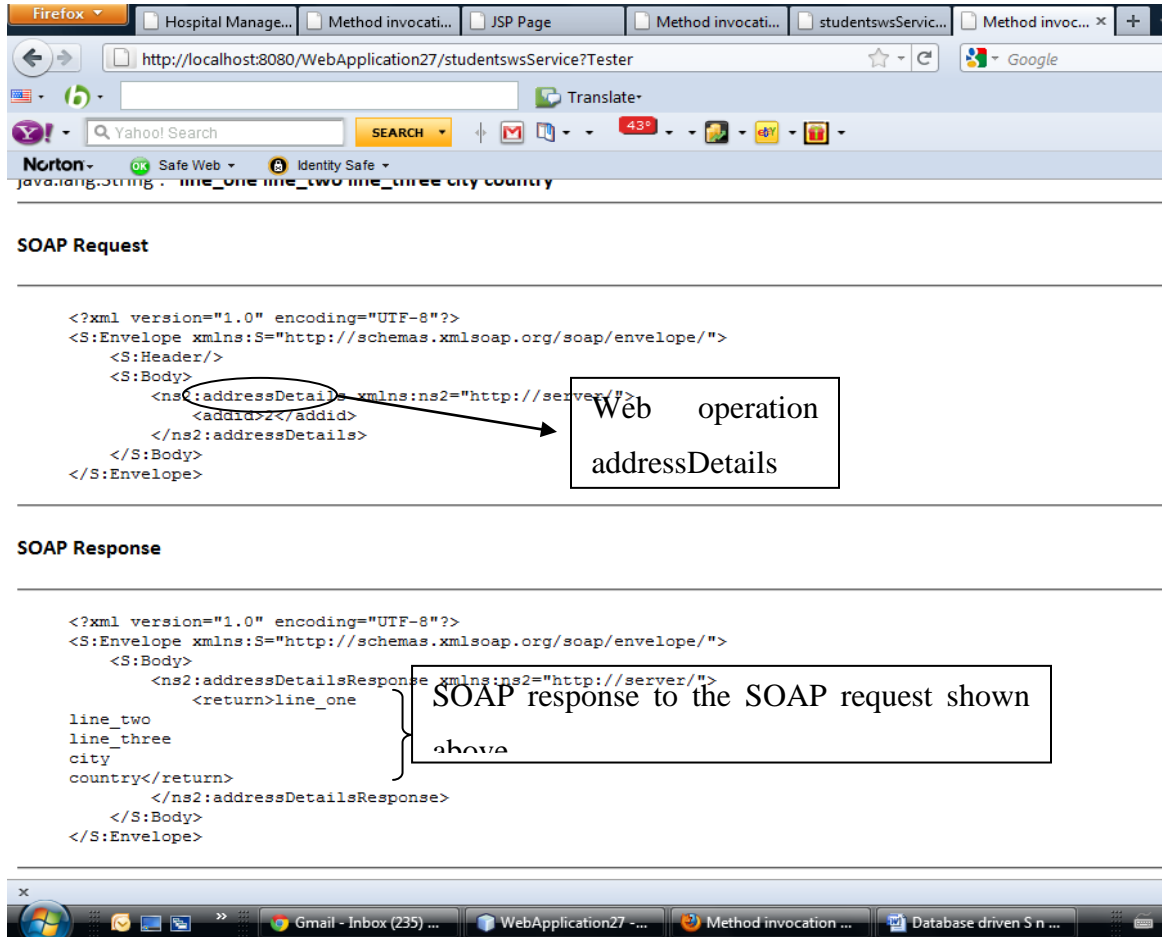


Fig 8.17 Output of the addressDetails web operation when address id "2" is inputted

Response contains the address details of the address id provided as shown in figure 8.17.

### Patient Details

Similarly, when patient id is entered into the patientDetails web operation, the patient's first name, middle name, last name and gender are returned.

## 8.2.10 LOAD test for the SOAP Web Services

Observation table

Resource: address Details web operation

**Table 8.3** Observation Table for LOAD test

Load Test	Thread Count	Min	Max	Avg	Last	cnt	tps	Bytes	bps
1	5	5	9477	445.07	7977	240	3.7	65760	1015
2	25	5	24247	360.55	24247	759	9.62	207966	2636
3	50	5	26316	504.57	26316	765	11.94	209610	3273
4	100	4	27185	3895.07	3227	563	8.95	154262	2453

The observation and inference for the graphs referred in Fig 8.18 and 8.19 given in table 8.3  
Graphs

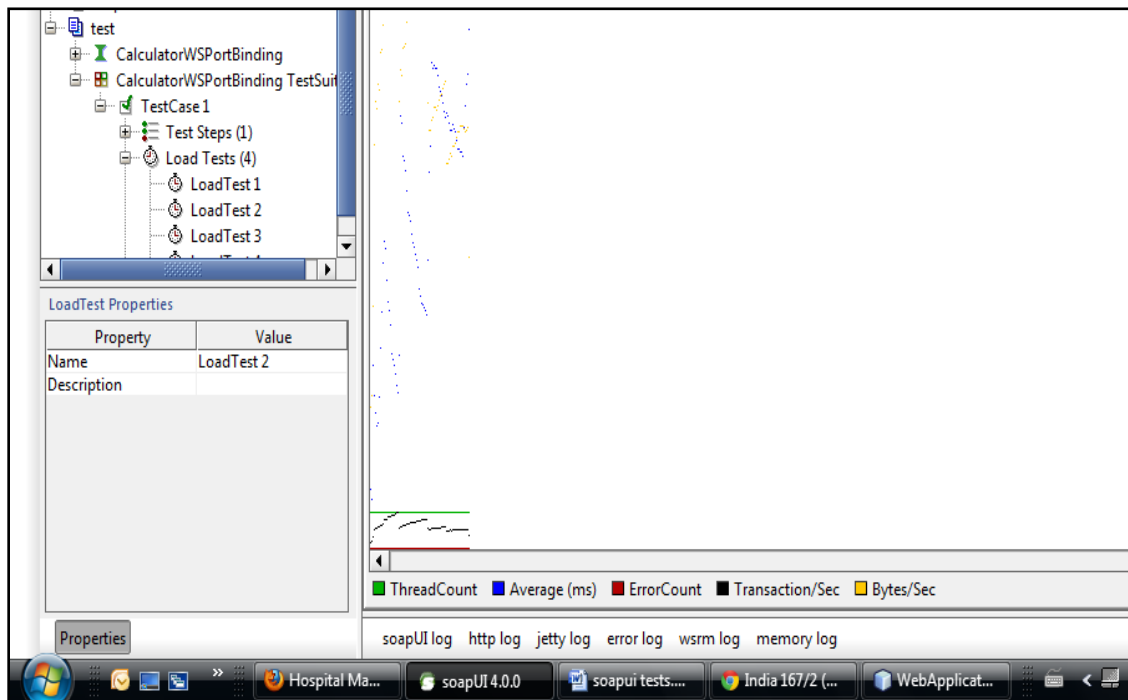


Fig 8.18 Graph for LOAD test 2

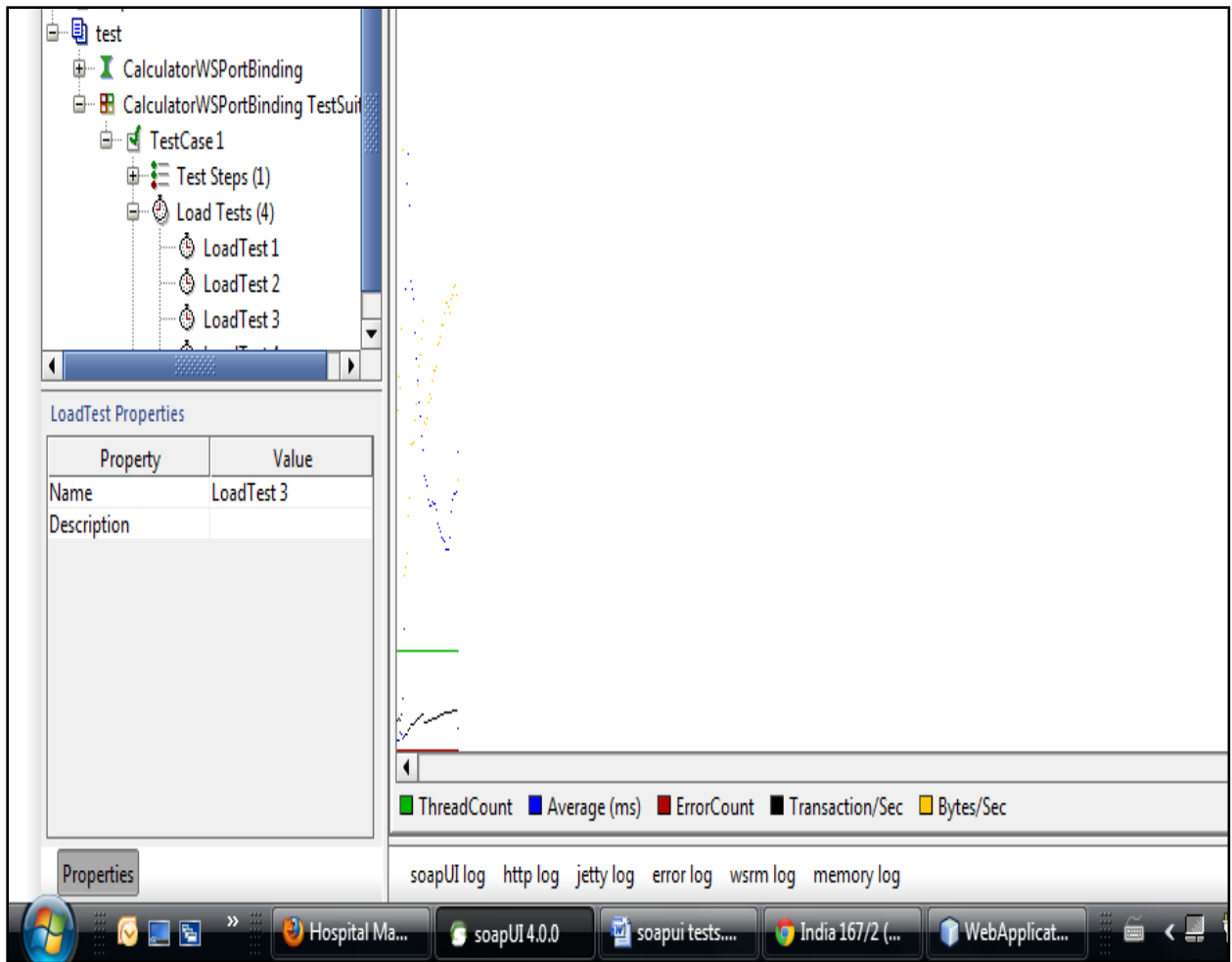


Fig 8.19 Graph for LOAD test 3

### 8.2.11 Performance Comparison of SOAP and REST Web Services

REST and SOAP Web Services are being compared based on the data shown in Tables 8-1 & 8-3.

Transactions per second of REST and SOAP Web Services (tps)

It can be observed from Table 8-4 that a REST Web Services has more transactions per second when compared to a SOAP Web Services

**Table 8.4** tps of REST and SOAP

Thread Count	REST tps	SOAP tps
5	5.01	3.7
25	26.66	9.62
50	51.03	11.94
100	97.38	8.95

Bytes per second of REST and SOAP Web Services (bps)

The number of bytes processed per second is greater for a REST Web Services when compared to SOAP Web Services. Refer to Table 8.5

**Table 8.5** bps of REST and SOAP

Thread Count	REST bps	SOAP bps	Bytes	Bytes
5	44662	1015	2769455	65760
25	237476	2636	14452815	207966
50	454452	3273	27463020	209610
100	867251	2453	53082705	154262

Minimum, Maximum and Average Time a step has taken

It is observed from Table 8.6 that the Maximum time a step has taken in a REST Web Services is much lower when compared to the SOAP Web Services. The Average time taken by a step is also lower for REST Web Services when compared to SOAP Web Services.

**Table 8.6** bps of REST and SOAP

Thread Count	REST Min	SOAP Min	REST Max	SOAP Max	REST Avg	SOAP Avg
5	5	5	3815	9477	200.01	445.07
25	5	5	1292	24247	75.06	360.55
50	5	5	1522	26316	93.02	504.57
100	5	4	1589	27185	91.05	3895.07

From the observations , it is understood that for this particular Database driven Hospital Management Web Services the REST implementation has a better performance than the SOAP implementation. The additional time taken by the SOAP Web Services to process a step or a request might be because of the overhead caused due to XML headers and tags, which are not present in a REST request.

### 8.2.12 Summary

A database driven hospital management application has been created using REST as well as SOAP Web Services. These Web Services have been tested and the output has been verified. LOAD test for these Web Services were done in SOAPUI. The observations of both SOAP and REST Web Services were noted and compared and it was observed that for this particular Web Services the REST implementation yielded better results. The SOAP and REST Web Services created in this chapter are deployed to the local server. This problem can be solved if these Web Services are hosted on a cloud. An attempt is made to measure the QoS of REST Web Service which is deployed in GAE. The performance measurement of REST Web Service under cloud environment will enable the client to decide which cloud to choose for hosting his service.



## 8.3 Performance measurement of Hospital Management Application using REST Web Service Deployed On Google SQL Cloud

Google SQL cloud enables us to create a relational database in SQL. This database is stored on the Google cloud and can be used by establishing an appropriate connection to the instance created. In order to create a database an instance should be created. Applications deployed to the Google app engine[99] can access this database by adding the name of the application in the authorized applications list of the instance created.

### 8.3.1 Database created for the Hospital Management Web Services

The code detail for this application is given briefly in APPENDIX B-4.

The instance created for developing a hospital management database is named **database**.

The name of the database is **mgmt**.

A connection can be established to this database by importing the AppEngineDriver:

```
import com.google.appengine.api.rdbms.AppEngineDriver;
```

Connection to the database can be established by:

```
DriverManager.registerDriver( new AppEngineDriver());  
C= DriverManager.getConnection(“jdbc:google:rdbms://RESTdatabase:database/mgmt”);
```

### 8.3.2 SQL

Structured Query Language is used to create, update and manage relational databases.

The database for the Hospital Management Web Services has been created using SQL in the Google SQL cloud.

This Web Services performs the following functions

1. Displays the doctor's list

2. Displays the patient's list
3. Appointment booking
4. Appointment booking by patients
5. Staff login
6. Doctor can view his patients list, history and can prepare an online prescription for his patient
7. Patient login
8. Patient can view his medical history
9. Display the available rooms and reserve a room

A relational database has been created to develop a Web Services that performs the above functions.

Tables created

1. doctors
2. patients
3. login\_details
4. patient\_login
5. patient\_history
6. rooms

SQL statements used to create tables are given in APPENDIX B-4

### **8.3.3 Connecting to the database in Google SQL Cloud**

After creating a database and its tables, a connection should be established to gain access to the database tables.

This is achieved by the statement:

```
C= DriverManager.getConnection("jdbc:google:rdbms://RESTdatabase:database/mgmt");
```

where **database** is an instance of **rest database** and **mgmt** is the name of the database created.

With this statement a connection is established and code can be written to either display or update data in the database tables.

### 8.3.4 Home page for the Web Services

Homepage for the Database driven Web Services hosted on the Google Cloud is shown in figure 8.20.



Fig 8.20 Homepage for the Database driven Web Services hosted on the **Google Cloud**

## **Google App Engine**

Google App Engine enables application developers to host their applications in the Google cloud.[96] The developer need not worry about the computing infrastructure of the cloud.

Users can deploy up to ten applications to the Google App Engine free of cost by logging into their Gmail accounts. Deploying applications to the cloud enables the user to access the application from anywhere and from any computer thereby removing the pain of building the application again and again.[98]

The Database driven Hospital Management Web Services is designed to display the doctor's list. This is achieved by the doctors.jsp file which connects to the database, named mgmt.

The patients.jsp file is coded to display the list of patients and also to add a new patient into the database. Details of a new patient can be added by using the HTML form,once this form is submitted control is directed to the patientServlet.java file which contains the doPost method required to update the details to the database table patients.

Since this form is also used to book an appointment and reserve a slot, code to check if the desired slot is available or not is written. This is achieved by considering the name of the doctor entered and also the desired slot. These values are compared with the details of other patients already existing. If a particular doctor is already reserved for a particular slot and the same slot is requested by the new patient, then a message "Sorry this slot is already Reserved!!" appears. If the requested slot is free for the doctor then the details of the new patient are added to the patients table. The patient name and password entered in the form are added to the patient login table and is used by patients for logging in.

The two files patients.jsp and patientServlet.java are used to display the details of the patients already existing and also used to add a new patient to the patients table. Patient can reserve the room if the room is not already occupied by the other patient.

These two files enable staff to login and view the details of his/her patients, prepare a prescription and also view the patient's history. If a doctor's name matches with the

corresponding password in the table then login is successful and the doctor will be able to view his patient's details. The name of the patient along with the patient's ID number and appointment time are displayed. The doctor will be able to prepare a prescription for the patient by clicking on the "Prepare Prescription" link. On clicking the link control is directed to patient history page. The doctor will be able to view patient's history by clicking on the View History link which redirects the doctor to the patient login. If the login details of the doctor do not match with the details stored in the login details table a blank page appears and the correct login details should be entered for successful login.

After successful login, the doctor, can view his patient and history and prepare online prescription for his patient.

To implement the online appointment scheduling system, the application displays the categories treated in the hospital. Categories can be like General Physician, Cardiologist, and Dermatologist. The names of all doctors who belong to that category are displayed. The name of the doctor, available time and the Reserved Slot is shown to enable a patient to see what slots are reserved and if the slot is free for that doctor that can be reserved by the patient. This Web Services has been deployed to the GAE and can be accessed from anywhere with internet access. The link is [resttrial1.appspot.com](http://resttrial1.appspot.com)

### **8.3.5 Performance of the Web Services**

A simple Java program has been written to measure the time taken to load a page of this Web Services which is hosted on the Google cloud. This java tool can be used to measure the scalability of any web application.

URL: RESTrial1.appspot.comNumber of Threads: 10

**Table 8.7** restrial1.appspot.com

Thread	Time in ms
1	941
2	1018
3	1090
4	1160
5	1165
6	1166
7	1167
8	1170
9	1175
10	1176

URL: RESTrial1.appspot.com/doctors.jspNumber of Threads: 10

**Table 8.8** RESTrial1.appspot.com/doctors.jsp

Thread	Time in ms
1	471
2	474
3	474
4	524
5	528
6	532
7	545
8	551
9	551
10	637

URL: RESTrial1.appspot.com/patients.jspNumber of Threads: 10

**Table 8.9** RESTrial1.appspot.com/patients.jsp

Thread	Time in ms
1	401
2	428
3	430
4	478
5	482
6	497
7	516
8	531
9	547
10	553

URL: RESTrial1.appspot.com/rooms.jspNumber of Threads: 10

**Table 8.10** resttrial1.appspot.com/rooms.jsp

Thread	Time in ms
1	456
2	484
3	612
4	735
5	736
6	751
7	754
8	754
9	754
10	970

URL: RESTrial1.appspot.com/reserve.jspNumber of Threads: 10

**Table 8.11** resttrial1.appspot.com/reserve.jsp

Thread	Time in ms
1	427
2	436
3	450
4	460
5	492
6	496
7	511
8	540
9	533
10	841

URL: RESTrial1.appspot.com/login.jspNumber of Threads: 10

**Table 8.12** resttrial1.appspot.com/login.jsp

Thread	Time in ms
1	348
2	356
3	368
4	381
5	395
6	402
7	416
8	431
9	440
10	450



URL: RESTtrial1.appspot.com/patient\_history.jsp Number of Threads: 10

**Table 8.13** resttrial1.appspot.com/patient\_history.jsp

Thread	Time in ms
1	344
2	351
3	359
4	369
5	381
6	392
7	397
8	405
9	413
10	422

From all the above table 8.7 to 8.13, it's very apparent that as the number of Web Services thread count increases, the delay in response time is not that high. The analysis of the results carried out for this study clearly show how in all cases in the light of the described scenario, REST is the fastest. The conventional SOAP-based solutions here tend to cause 4-5 fold delays compared with REST Web Services on the request-response processing cycle.

A relational database has been created in Google SQL cloud. An application deployed to the Google App engine has been authorized to access the database instance created in the Google SQL cloud. Database driven Hospital Management Web Services has been created. This Web Services can display the details of all tables in the database, doctor can login and can view his patients list, history and can prepare an online prescription, patient can login and view his history, patient can also book an appointment online and room reservation can also be done. The performance of this Web Services has been tested by writing a simple Java program and the time taken to load a web page of this Web Services has been noted.

## 8.5 Summary

Web Services are internet enabled software components using which provide integration of Enterprise applications. Web Services are being used extensively by many enterprises every day. SOAP and REST are the popular types of Web Services.

A Hospital Management Application has been implemented in the form of SOAP and REST Web Services [95]. The design and implementation of this application has been done in two different methods using MySQL, Netbeans, Eclipse, Google App Engine and Google SQL cloud.

In the first method, database for this application was created in MySQL workbench. The application was implemented as SOAP and REST Web Services in Netbeans. These Web Services were tested in SOAPUI and their performance was compared. The scalability test showed that a REST Web Services performed better when compared to the SOAP Web Services.

The second method implements the Hospital Management application as a database driven Enterprise Application integration (EAI) Web Services by creating the database required in Google SQL cloud. The application is designed and developed in Eclipse and is deployed to the Google App Engine to make it available online [96]. The application designed using this method can be accessed from any place with internet access and at any time. Patients can book an appointment online from any part of the world at any time. A Java tool is developed to measure the scalability of the web application. The Hospital Management application developed using this method can be accessed by using the link <http://RESTtrial1.appspot.com>

Nowadays service providers hosting the service in cloud rather than the web servers as the cloud provides more significant advantages[96]. It will be of great interest to anybody, to know how the REST Web Service performs under a cloud environment. so an attempt is made for the same in this chapter of the thesis.

## CHAPTER 9

### Conclusion, Contributions and Future work

#### 9.1 Conclusion

Web Services are turning out to be progressively utilized and a large number of consumers are building their business solutions using Web Services technology. With the development of Web Services and applications, quality of service (QoS) has become a key issue in the selection of Web Services. The requirement for QoS specifications for Web Services has arisen due to consumer's prospect for superior Web Services performance and services provider's obligation to provide high quality service so as to improve the usability and utility of their services which in turn decides their standing in the market. Current discovery model and standards such as UDDI, WSDL do not give much support to QoS description and evaluation.

In the first chapter, The WSDL file of the provider is modified to accommodate QoS values of the Web Services. The service mining of Web Services is done by agent Web Service using the QoS data, that are provided by the service provider. The agent Web Service can classify the Web Services into different classes based on their QoS data.

In the next few chapters, new algorithm is designed and implemented for classifying Web Services into different classes using their QoS values. This classifier Web Service takes QoS parameters as input and applies an entropy based discretization algorithm and yields the class that the Web Services belong to as output to the client. In the process of selection of Web Services, these are categorized as different groups like excellent, good, average and poor.

A novel way of WSS using QoS attributes based on composition of Web Services have been illustrated. J48 classification algorithm is exported as Web Service whose request contain QoS attribute in ARFF. The resultant output is decision tree rules. Then the output of this service is passed to the visualization service, which in turn gives us the visualized, graphed

output of the tree. A BPEL process is defined to compose above Web Services. The tree view of the classification of Web Services according to the class they correspond to will be the final output of the above composition.

Out of all the QoS properties, the latency and throughput of the Web Services is always of more concern to the Web Services users, hence the working model of the architecture is created to demonstrate the processes involved in classifying the Web Services based on those attributes. An attempt is made to measure throughput in real time and to create a repository of the same using an agent approach. The agent in turn uses SOAPUI to perform a performance testing and the results from SOAPUI are then stored in the agent. When the user looks for a Web Service in a particular functional domain, the agent presents the client with the available QoS parameters, which the client can use to make a selection.

With the advent of REST Web Services there is widespread interest in performing QoS attributes for the same and compare the results with that of SOAP enabled one. In the last chapter, an attempt is made to implement the same. A hospital management application has been developed exclusively using SOAP, as well as exclusively using REST Web Services. A technique is made available to measure the performance metrics of REST and SOAP Web Services under identical operational environment which will be of immense help to the clients for choosing the appropriate service for their application in SOA. Here, an attempt is made to measure the QoS value of both REST and SOAP Web Service for a database driven application. REST Web Service is deployed in GAE to make it available online. It will be of great interest to someone who wants to know how the REST Web Service performs under a cloud environment. The Hospital Management application developed using this method can be accessed by using the link <http://RESTtrial1.appspot.com>. This application is also investigated for the scalability testing of the cloud enabled hospital management system that has also been done using a custom developed software tool.

## 9.2 Contributions

The contribution of this thesis work can be given as follows.

- Proposed extended service oriented architecture for service mining, using Entropy based discretization algorithm, using QoS attribute values of the Web Services in that particular functional domain.
- Proposed architecture to evaluate Web Services for their QoS attribute values.
- The Hospital management system is deployed in the cloud in order to experimentally evaluate and compare the QoS values for SOAP and REST web services.

## 9.3 Future work

There are several research directions in which further investigations could be made regarding this work in the future. Even though the discussed approaches of Web Services classification and services discovery are functional and useful to the end user, these methods do come with some inherent limitations like broker based Web Services selection which introduces a single failure point and service discovery broker might turn out to be a bottleneck. Inconsistencies in the UDDI registry or the QoS parameters not being up to date might result in flawed discoveries.

In this thesis for QoS classification of Web Services, the database which is used is a real time database collected over thousands of Web Services. But at the same time, since the Web Services are highly dynamic, their QoS values may change over time. More investigations are required to study the temporal correlations and periodicity features of the Web Services

QoS values. For the Web Services QoS value prediction, more research on the correlation and combination of different QoS properties has to be conducted.

Meanwhile, in this research the only focus is on QoS model and computation, without taking user individual preferences into consideration. How to dynamically evaluate candidate Web Services based on user preference of QoS attribute is an important problem to be solved. Future work on the project can involve incorporating the load testing process in the architecture instead of using a third party testing tool. Also, the architecture can be made to advice the user on the selection of service by dynamically running performance tests in real time.

As the applications are developed and deployed in cloud, securing cloud enabled Web Services is very crucial, especially when confidential data is being transferred between the client and the service. The impact of security over non secured Web Services can be carried out as part of future work.

## REFERENCES

- [1]. Al-Masri, E.; Mahmoud, Qusay H., in the paper "Discovering the best Web Services: A neural network-based solution," Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on vol., no., pp.4250,4255, 11-14 Oct. 2009 doi: 10.1109/ICSMC.2009.5346817
  
- [2]. Xiaopeng Deng; Chunxiao Xing, "A QoS-oriented Optimization Model for Web Services Group," Computer and Information Science, 2009. ICIS 2009. Eighth IEEE/ACIS International Conference on , vol., no., pp.903,909, 1-3 June 2009 doi: 10.1109/ICIS.2009.91
  
- [3]. Badr, Y.; Abraham, A.; Biennier, F.; Grosan, C., "Enhancing Web Services Selection by User Preferences of Non-functional Features," *Next Generation Web Services Practices, 2008. NWeSP '08. 4th International Conference on* , vol., no., pp.60,65, 20-22 Oct. 2008 doi: 10.1109/NWeSP.2008.39
  
- [4]. Soon Ae Chun; Atluri, V.; Adam, N.R., "Policy-based Web Services composition," *Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications, 2004. Proceedings. 14th International Workshop on* , vol., no., pp.85,92, 28-29 March 2004 doi: 10.1109/RIDE.2004.1281707
  
- [5]. Aihkisalo, T.; Paaso, T., "Latencies of Service Invocation and Processing of the REST and SOAPWeb Services Interfaces," *Services (SERVICES), 2012 IEEE Eighth World Congress on*, vol., no., pp.100,107, 24-29 June 2012 doi: 10.1109/SERVICES.2012.55
  
- [6]. E. Al-Masri, and Q.H. Mahmoud, "Investigating Web Services on the World Wide Web". In Proceedings of the 17th International World Wide Web Conference (WWW2008), pp. 795-804, 2008.
  
- [7]. <http://www.cs.toronto.edu/~shirin/sohrabi-2012-09-thesis.pdf>

- [8]. Efficient QoS-aware Service Composition Mohammad Alrifai, Thomas Risse  
[http://link.springer.com/chapter/10.1007%2F978-3-0346-0104-7\\_5](http://link.springer.com/chapter/10.1007%2F978-3-0346-0104-7_5)
- [9]. Web Services Directory (WSIndex), <http://www.wsindex.org>, Accessed February 2008.
- [10]. Web Services List, <http://www.webservicelist.com>, Accessed February 2008.
- [11]. XMethods, <http://www.xmethods.net>, Accessed February 2008.
- [12]. E. Maximilien and M. Singh. "Conceptual model of Web Services reputation", ACM SIGMOD Record, 31(4), 2002.
- [13]. K. Sivashanmugam, K. Verma, and A. Sheth, "Discovery of Web Services in a federated registry environment," ICWS, pp. 270-278, 2004.
- [14]. Eyhab Al-Masri and Qusay H. Mahmoud, "A Broker for Universal Access to Web Services". 2009 Seventh Annual Communications Networks and Services Research Conference, page 118-123.
- [15]. Hartwig Gunzer, "Introduction to Web Services". Borland Software Corporation, page 4-14. <http://www.itmanage.info/technology/webservice/webservices.pdf>
- [16]. D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004
- [17]. Haiteng Zhang, Zhiqing Shao, Hong Zheng, and Jie Zhai, "Web Service Reputation Evaluation Based on QoS Measurement," The Scientific World Journal, vol. 2014, Article ID 373902, 7 pages, 2014. doi:10.1155/2014/373902



- [18]. Tewari, V.; Dagdee, N.; Singh, I.; Garg, N.; Soni, P.; , "An Improved Discovery Engine for Efficient and Intelligent Discovery of Web Services with Publication Facility," *Services - II, 2009. SERVICES-2 '09. World Conference on* , vol., no., pp.63-70, 21-25 Sept. 2009.
- [19]. "Agent based discovery of Web Services to enhance the quality of Web Services selection", *International Journal of Computer Science and Network Security (IJCSNS)*, Dr.SangH.Lee.Vol.11,No.2,pp159-163.  
[http://paper.ijcsns.org/07\\_book/201102/20110226.pdf](http://paper.ijcsns.org/07_book/201102/20110226.pdf) Feb'2011 paper id (2101109)
- [20]. SungHeun Nam; YunHee Kang; , "XML Schema Design for Web Services Quality Management," *Future Generation Communication and Networking, 2008. FGNCN '08. Second International Conference on* , vol.2, no., pp.99-102, 13-15 Dec. 2008
- [21]. Badr, Y.; Abraham, A.; Biennier, F.; Grosan, C.; , "Enhancing Web Services Selection by User Preferences of Non-functional Features," *Next Generation Web Services Practices, 2008. NWESP '08. 4th International Conference on* , vol., no., pp.60-65, 20-22 Oct. 2008
- [22]. why data mining in CRM, 2011 <https://alsysoncrm.wordpress.com/2011/02/21/why-data-mining-in-crm>
- [23]. *International Journal of Advanced Research in Computer and Communication Engineering* Vol. 2, Issue 4, April 2013 WEKA Approach for Comparative Study of Classification Algorithm Trilok Chand Sharma1 , Manoj Jain2

<http://www.ijarcce.com/upload/2013/april/60-trilok-WEKA%20approach%20for%20comparative.pdf>

- [24]. V. Diamadopoulou, C. Makris, Y. Panagis and E. Sakkopoulos, "Techniques to support Web Services selection and consumption with QoS characteristics". *Journal of Network and Computer Applications*, vol. 31, 2008, pp. 108-130.
- [25]. SungHeun Nam; YunHee Kang; , "XML Schema Design for Web Services Quality Management," *Future Generation Communication and Networking, 2008. FGCN '08. Second International Conference on*, vol.2, no., pp.99-102, 13-15 Dec. 2008
- [26]. Al-Masri, E., and Mahmoud, Q. H., "Discovering the best Web Services", (poster) 16th International Conference on World Wide Web (WWW), 2007, pp. 1257-1258. (for QWS Dataset Version 1.0 or QWS Dataset Version 2.0)
- [27]. Al-Masri, E., and Mahmoud, Q. H., "QoS-based Discovery and Ranking of Web Services", IEEE 16th International Conference on Computer Communications and Networks (ICCCN), 2007, pp. 529-534. (for QWS Dataset Version 1.0 or QWS Dataset Version 2.0)
- [28]. Al-Masri, E., and Mahmoud, Q.H., "Investigating Web Services on the World Wide Web", 17th International Conference on World Wide Web (WWW), Beijing, April 2008, pp. 795-804. (for QWS-WSDLs Dataset Version 1.0)
- [29]. Pieter Adriaans and Dolf Zantinge, *Introduction to Data Mining and Knowledge Discovery, Third Edition* (Potomac, MD: Two Crows Corporation, 1999); *Data Mining* (New York: Addison Wesley, 1996).
- [30]. Jensen, "Data Mining in Networks," ppt; K.A. Taipale, "Data Mining and Domestic Security: Connecting the Dots to Make Sense of Data," Columbia Science and

Technology Law Review 5 (December 2003): 28, available at <http://stlr.org/cite.cgi?volume=5&article=2>.

- [31]. Marc M. Van Hulle, "Data Mining". Luc Dehaspe, Oncolmethylome BVBA, <http://www.slideshare.net/Tommy96/data-mining-4035491>
- [32]. Osmar R. Zaiane, "Principles of Knowledge Discovery in Databases". University of Alberta, 1999. <http://webdocs.cs.ualberta.ca/~zaiane/courses/cmput690/slides/ch1s.pdf>
- [33]. Liang – Jie – Zhang, Jia Zhang, "Services Computing" Springer Berlin Heidelberg, New York. ISBN 978-3-540-38281-2 Springer Berlin Heidelberg New York <http://ir.nmu.org.ua/bitstream/handle/123456789/141883/70775b935870742058bfc4a42026f77a.pdf?sequence=1>
- [34]. Thomas Erl, "Service Oriented Architecture: A Field Guide to Integrating XML and Web Services", Prentice Hall Publications, 2004. ISBN-13: 007-6092025443 ISBN-10: 0131428985
- [35]. "Web Services selection based on QoS attributes using entropy discretization method", International Journal of Computer Applications(IJCA) Published by Foundation of computerScience,NewYork, U.S.A.<http://www.ijcaonline.org/archives/volume30/number2/3611-4119>
- [36]. Raj, R.J.R.; Sasipraba, T., "Web service selection based on QoS Constraints," Trendz in Information Sciences & Computing (TISC), 2010 , vol., no.,pp.156,162,17-19 Dec.2010 doi: 10.1109/TISC.2010.5714629URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5714629&isnumber=5714593>

- [37]. Liu Sha Guo Shaozhong Chen Xin Lan Mingjing Zhengzhou, “ A QoS Based Web Services Selection Model”, International Forum on Information Technology and Applications, 2009.
- [38]. Manish Godse, Umesh Bellur, Rajendra Sonar, “Automating QoS based Service Selection”, 2010 IEEE International Conference on Web Services PP: 530-540.
- [39]. Kyriakos Kritikos and Dimitris Plexousakis, “Requirements for QoS Based Web Services Description and Discovery”, IEEE Transactions on Services Computing, Vol. 2, 2009.
- [40]. Atin Aggarwal ‘Web Services Selection Based on QoS Attributes’ Project report, BITS Pilani, Dubai Campus
- [41]. Sidney Rosario, Albert Benveniste, “Probabilistic QoS and Soft Contracts for Transaction-Based Web Services Orchestrations”, IEEE Transactions on Services Computing, 2008.
- [42]. Qu Li-li, Chen Yan, “QoS Ontology Based Efficient Web Services Selection”, International Conference on Management Science & Engineering, 2009.
- [43]. Serhani M. A., Dssouli R., Hafid A., Sahraoui H., “A QoS broker based architecture for efficient Web Services selection”, IEEE International Conference on Web Services, 2005.
- [44]. Yu T., Lin K. J., “Service selection algorithms for Web Services with end-to-end QoS constraints”, Proceeding of Information Systems and E-Business Management, 2005.
- [45]. Tran Vuong Xuan, Tsuji Hidekazu, “QoS based ranking for Web Services: Fuzzy Approach”, International Conference on Next Generation Web Services Practices, 2008.

- [46]. Lin M., Xie J., Guo H., Wang H., "Solving QoS-Driven Web Services Dynamic Composition as Fuzzy Constraint Satisfaction", IEEE International Conference on E-Technology, E-Commerce and E-Service, 2005.
- [47]. Santhi T., Ananthanarayana V. S., D'Mello Demian Antony, "A QoS broker based architecture for Web Services selection", International Conference on Modelling & Simulation, 2008.
- [48]. Menasce D. A., "QoS Issues in Web Services", IEEE Internet Computing, 2002.
- [49]. Fayyad U., Irani K., "Multi-interval Discretization of continuous-valued attributes for classification learning, International Joint Conference on Artificial Intelligence, 1993.
- [50]. Ellis J. Clarke, Bruce A. Barton, "Entropy and MDL Discretization of Continuous Variables for Bayesian Belief Networks", INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, 2000.
- [51]. Anbazhagan Mani and Arun Nagarajan, "Understanding quality of service for Web Services". IBM software lab, 2002.
- [52]. QWS Dataset, <http://www.uoguelph.ca/~qmahmoud/qws/index.html>
- [53]. Ian H. Witten; Eibe Frank (2005). "[Data Mining: Practical machine learning tools and techniques, 2nd Edition](#)". Morgan Kaufmann, San Francisco.
- [54]. Architecture of the World Wide Web, First Edition, W3C Working Draft, I. Jacobs, 9 December 2003 (See <http://www.w3.org/TR/2003/WD-webarch-20031209/>.)
- [55]. Web Services Description Language (WSDL) 1.1, W3C Note, 15 March 2001 ([http://www.w3.org/TR/WSDL#\\_service](http://www.w3.org/TR/WSDL#_service))

- [56]. Service Oriented Architecture (SOA) and Specialized Messaging Patterns, Adobe Technical White Paper, Duane Nickull, Laurel Reitman, James Ward, Jack Wilber.  
[www.ijarcsse.com/docs/papers/Volume\\_4/7\\_July2014/V4I7-0451.pdf](http://www.ijarcsse.com/docs/papers/Volume_4/7_July2014/V4I7-0451.pdf)
- [57]. Online:[http://www.service-architecture.com/Web\\_Services/articles/web\\_services\\_explained.html](http://www.service-architecture.com/Web_Services/articles/web_services_explained.html)
- [58]. Online:<http://www.ibm.com/developerworks/webservices/library/ws-quality/index.html>
- [59]. Online:<http://www.w3c.or.kr/kr-office/TR/2003/ws-QoS/#QoS-2>
- [60]. Online:<http://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/2.htm>
- [61]. Online:<http://www.hiraeth.com/books/ai96/QBB/id3.html>
- [62]. Al-Masri, E., and Mahmoud, Q. H., "Discovering the best Web Services", (poster) 16th International Conference on World Wide Web (WWW), 2007, pp. 1257-1258.
- [63]. Rajesh Sumra and Arulazi. D., Quality of Service for Web Services-Demystification, Limitations, and Best Practices, March 2003.  
[http://www.developer.com/java/web/article.php/10935\\_2248251\\_2/Quality-of-Service-for-Web-ServicesdashDemystification-Limitations-and-Best-Practices-for-Performance.html](http://www.developer.com/java/web/article.php/10935_2248251_2/Quality-of-Service-for-Web-ServicesdashDemystification-Limitations-and-Best-Practices-for-Performance.html).
- [64]. [http://www.developer.com/services/article.php/10928\\_2027911\\_2/Quality-of-Service-for-Web-ServicesdashDemystification-Limitations-and-Best-Practices.htm](http://www.developer.com/services/article.php/10928_2027911_2/Quality-of-Service-for-Web-ServicesdashDemystification-Limitations-and-Best-Practices.htm)
- [65]. Jia Zhang and Liang-Jie Zhang, Criteria Analysis and Validation of the Reliability of Web Services-oriented Systems, IEEE International Conference on Web Services (ICWS'05)

- [66]. “Web Services selection through QoSWeb Services”, Published papers in the International Journal of Software and Web Sciences (IJSWS), ISSN (Online): 2279-0071, ISSN (Print): 2279-0063 (September-November, 2013, Issue 6, Volume 1). Pp 18-23.<http://iasir.net/IJSWSpapers/IJSWS13-325.pdf>
- [67]. Getting Started with JAX-WS Web Services, [www.netbeans.org](http://www.netbeans.org)
- [68]. Dr. Iiavarasan Egambaram, G. Vadivelou, S. Prasath Sivasubramanian., QoS based Web Services selection.
- [69]. Aditya Julka, ‘Non Functional Property Based Service Selection’ Project report, BITS Pilani, Dubai Campus
- [70]. Bernhard Borges, Kerrie Holley and Ali Arsanjani, IBM, Service-oriented architecture, <http://searchsoa.techtarget.com/news/1006206/Service-oriented-architecture>
- [71]. “A QoS broker based architecture for efficient Web Services selection” by M.Adel Serhani & Rachida Dssouli, IEEE Computer Society paper
- [72]. Susila, S.; Vadivel, S.; Julka, A., "Broker architecture for Web Services selection using SOAPUI," Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on , vol., no., pp.219,222, 8-10 Dec. 2012 doi:10.1109/ICCCTAM.2012.6488102  
URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6488102&isnumber=6488050>
- [73]. “Non-functional property based service selection”by Hong Qing Yu and Stephan Reiff-Marganiec, University of Leicester, Computer Science Department, Leicester, UK.

- [74]. “An Efficient WS-QoS Broker Based Architecture for WebServices Selection”by T.Rajendran, Dr.P.Balasubramanie, Resmi Cherian, 2010 International Journal of Computer Applications (0975 – 8887).
- [75]. Clifton, Christopher (2010). "Encyclopedia Britannica: Definition of Data Mining".
- [76]. Kantardzic, Mehmed (2003). Data Mining: Concepts, Models, Methods, and Algorithms. John Wiley & Sons
- [77]. Fayyad, Usama; Gregory Piatetsky-Shapiro, and Padhraic Smyth (1996). "From Data Mining to Knowledge Discovery in Databases"
- [78]. Pang ning tan, Michael Steinbach, Vipin 'Cluster Analysis: Basic Concepts and Algorithms' Addison Wesley 2006
- [79]. Online: <http://www.w3.org/TR/ws-gloss/>, WebServices Glossary
- [80]. Dushyant Rijhwani, ‘A novel approach to distributed data mining by composing Web Services’ Project report, BITS Pilani, Dubai Campus
- [81]. Online: <http://www.w3.org/TR/WSDL20/> , Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language
- [82]. Online: <http://www.w3.org/TR/SOAP12-part1/>, SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)
- [83]. Shetty S., S. Vadivel, Vaghella S., “WEKA based Data Mining as Web Services”, World Academy of Science, Engineering and Technology 64 2010



- [84]. Ali Shaikh Ali, Omer F. Rana and Ian J. Taylor, “Web Services Composition for Distributed Data Mining”, International Conference on Parallel Processing Workshops, 2005
- [85]. Two Crows Corporation, Introduction to Data Mining and Knowledge Discovery, Third Edition (Potomac, MD: Two Crows Corporation, 1999); Pieter Adriaans and Dolf Zantinge, Data Mining (New York: Addison Wesley, 1996).
- [86]. Online: [http://www.w3schools.com/WSDL/WSDL\\_uddi.asp](http://www.w3schools.com/WSDL/WSDL_uddi.asp), WSDL and UDDI, [accessed:19th May 2011]
- [87]. Online: <http://www.javabeat.net/tips/144-benefits-of-using-web-services.html>, Benefits of using Web Services, [accessed: 19th May 2011]
- [88]. Online: <http://www.w3.org/TR/wscl10/>, W3C. Web Services Conversation Language (WSCL) 1.0. 2002. [Accessed 21st May, 2011].
- [89]. Milanvoic, Nikola et Malek, Mirosław. Current Solutions for Web Services composition. s.l. : IEEE Computer Society, 2009
- [90]. Online:[http://www.oracle.com/technology/pub/articles/matjaz\\_bpel1.htm](http://www.oracle.com/technology/pub/articles/matjaz_bpel1.htm), Matjaz and B. A Hands-on Introduction to BPEL. [Accessed 21st May 2011]
- [91]. Papazoglou, Michael. Web Services: Principles and Technology. s.l. : Prentice, Hall, 2008
- [92]. Fuhrer Patric, et. al Web Services Orchestration and Composition, September 2009
- [93]. Online:<http://www.w3schools.com/uri/default.asp>
- [94]. Microsoft. 2011 Microsoft. <http://msdn.microsoft.com/en-s/library/ms190796.aspx>

- [95]. Upadhyaya, B.; Ying Zou; Hua Xiao; Ng, J.; Lau, A., "Migration of SOAP-based services to RESTful services," *Web Systems Evolution (WSE)*, 2011 13th IEEE International Symposium on , vol., no., pp.105,114, 30-30 Sept. 2011,
- [96]. Online: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6081828&isnumber=6081811>
- [97]. Smartbear Software. 2011. <http://www.SOAPui.org/Load-Testing/load-test-window.html>
- [98]. Online:CloudTutorial. <http://thecloudtutorial.com/cloudtypes.html>
- [99]. Infoworld.1994-2011 Infoworld Inc. <http://www.infoworld.com/d/security-central/gartner-seven-cloud-computing-security-risks-853>
- [100]. Gigaom.2011 GigaOm <http://gigaom.com/2010/01/14/who-exactly-owns-your-data-in-the-cloud/>
- [101]. M Siva Manaswini, 'Database driven Web Services and Cloud Computing' Project report, BITS Pilani, Dubai Campus, 2012
- [102]. Eric Newcomer. Understanding Web Services. Boston: Pearson Education,2002
- [103]. Judith Hurwitz, Robin Bloor, Marica Kaufman, and Dr. Fern Halper. Cloud Computing for Dummies. Indianapolis: Wiley Publishing 2010
- [104]. Michael P. Papazoglou. Web Services: Principles and Technology, Pearson Education, 2008, ISBN 0321155556,. 9780321155559.

- [105]. Kyle Roche and Jeff Douglas. Beginning Java Google App Engine: Apress 2009
- [106]. Web Services 2011 Oracle. <http://docs.oracle.com/javaee/6/tutorial/doc/bnbxw.html>

# APPENDICES

## APPENDIX – A

### **System configurations.**

The implementations of the work in the thesis are done in the programming lab of BITS Pilani Dubai Campus. A detail of the system configuration is as follows.

System:

Microsoft Windows XP

Professional

Version 2002

Service Pack 3, v.5938

Computer:

Intel(R) Core (TM) 2 CPU

4400 @ 2.00 GHz

2.00 GHz, 0.99 GB of RAM.

## **Software used**

WEKA 3.6 an open source data mining software

Graphviz open source tool for drawing graphs.

SOAPUI open source tool for load test of Web Services.

Java Version 1.6

Java Derby database and Java persistence API (JPA)

NetBeans IDE 6.9.1

MySQL Workbench

Eclipse Indigo

Google SQL Cloud

Google App Engine

## APPENDIX – B-1

### Functions and their actions for the Decision Tree Construction for Continuous attributes.

main function	<ol style="list-style-type: none"><li>1. The default constructor is invoked.</li><li>2. Training data is read from the source file.</li><li>3. function createDecisionTree() is called.</li></ol>
function createDecisionTree()	<ol style="list-style-type: none"><li>4. Function decomposeNode is called with root as the argument.</li><li>5. Root node is printed.</li><li>6. The values of availability, through put, response time, Successability and reliability of the test set are read as user input.</li></ol>
function decomposeNode	<ol style="list-style-type: none"><li>1. Entropies of all attributes and attribute values are calculated.</li><li>2. The best attribute and value are located so as to give maximum decrease in entropy.</li><li>3. Dataset is divided into two nodes using the selected attribute and value.</li><li>4. The resulting two nodes are further decomposed recursively using the same function until all leaf nodes.</li></ol>

## APPENDIX – B-2

### WSDL File

This XML file does not appear to have any style information associated with it.  
The document tree is shown below.

```
<!--
  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
  Metro/2.2-b13 (branches/2.2-6964; 2012-01-09T18:04:18+0000) JAXWS-
  RI/2.2.6-promoted-b20 JAXWS/2.2 svn-revision#unknown.
-->
<!--
  Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
  Metro/2.2-b13 (branches/2.2-6964; 2012-01-09T18:04:18+0000) JAXWS-
  RI/2.2.6-promoted-b20 JAXWS/2.2 svn-revision#unknown.
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-
1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-
policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns
:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://
schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://qos.me.org/" xmlns:xsd
="http://www.w3.org/2001/XMLSchema" xmlns:qos="http://schemas.xmlsoap.org/wsd
l/" targetNamespace="http://qos.me.org/" name="QoS">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://qos.me.org/" schemaLocation="http://
localhost:8080/QoS/QoS?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="QoSTest">
    <part name="parameters" element="tns:QoSTest"/>
  </message>
  <message name="QoSTestResponse">
    <part name="parameters" element="tns:QoSTestResponse"/>
  </message>
  <portType name="QoS">
    <operation name="QoSTest">
      <input wsam:Action="http://qos.me.org/QoS/QoSTestRequest" message=
"tns:QoSTest"/>
      <output wsam:Action="http://qos.me.org/QoS/QoSTestResponse" messag
e="tns:QoSTestResponse"/>
    </operation>
  </portType>
  <binding name="QoSPortBinding" type="tns:QoS">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style
="document"/>
    <operation name="QoSTest">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="QoS">
    <port name="QoSPort" binding="tns:QoSPortBinding">
      <soap:address location="http://localhost:8080/QoS/QoS"/>
    </port>
  </service>
</definitions>
```

## APPENDIX – B-3

### Functions and their actions for the Web Service Composition.

#### Clusterring Web Service

```
“<code>

package models;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import weka.clusterers.ClusterEvaluation;
import weka.clusterers.DensityBasedClusterer;
import weka.clusterers.EM;
import weka.core.Instances;

/**
 *
 * @author susila
 */
@WebService()
public class clustering1 {

/**
```



```

* Web Service operation
*/
@WebMethod(operationName = "execute")
public String execute(@WebParam(name = "input") String input) throws IOException,
Exception {
    //TODO write your implementation code here:
    ClusterEvaluation eval;
    Instances data;
    String[] options;
    DensityBasedClusterer cl;
    data = new Instances(new BufferedReader(new FileReader(input)));
    StringBuffer result;
    result = new StringBuffer();
    result.append("\nWEKA - DEMO\n=====\\n\\n");
    options = new String[2];
    options[0] = "-t";
    options[1] = input;
    result.append(ClusterEvaluation.evaluateClusterer(new EM(), options));
    cl = new EM();
    cl.buildClusterer(data);
    eval = new ClusterEvaluation();
    eval.setClusterer(cl);
    eval.evaluateClusterer(new Instances(data));
    result.append("\n--> manual" + "\n\n No. of
clusters:").append(eval.getNumClusters()).append("\n");
    cl = new EM();

```

```

eval = new ClusterEvaluation();
eval.setClusterer(c1);
eval.crossValidateModel(c1, data, 10, data.getRandomNumberGenerator(1));
result.append("\n-->      Density(CV)"      +      "\n\n      No      of
Clusters:").append(eval.getNumClusters()).append("\n\n\n");
return result.toString();
}
}
</code>”

```

### **Classification Web Service**

```

“<code>
package source;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import weka.classifiers.trees.J48;
import weka.core.Instances;
/**
 *
 * @author susila
 */

```

```

@WebService()
public class J48data {
    /**
     * Web Service operation
     */
    @WebMethod(operationName = "mine")
    public String mine(@WebParam(name = "file") String file) throws IOException, Exception
    {
        //TODO write your implementation code here:

        Instances data;

        data = new Instances(new BufferedReader(new FileReader(file)));
        data.setClassIndex(data.numAttributes() - 1);

        StringBuffer result;

        J48 cls = new J48();

        cls.buildClassifier(data);

        result = new StringBuffer();

        result.append(cls.graph());

        return result.toString();

    }
}
</code>”

```

### **Visualization Service**

```
<code>
```

```
package source;
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.swing.JFrame;
import weka.gui.treevisualizer.PlaceNode2;
import weka.gui.treevisualizer.TreeVisualizer;
```

```
@WebService()
```

```
public class J48Visualize {
```

```
    /**
```

```
     * Web Service operation
```

```
     */
```

```
    @WebMethod(operationName = "operation")
```

```
    public String operation(@WebParam(name = "cls_data")
```

```
        String cls_data) {
```

```
        TreeVisualizer tv = new TreeVisualizer(null, cls_data, new PlaceNode2());
```

```
        JFrame jf= new JFrame("Weka Classifier Tree");
```

```
        jf.setAlwaysOnTop(true);
```

```
        jf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

```

        jf.setSize(800,600);

        jf.getContentPane().setLayout(new BorderLayout());
        jf.getContentPane().add(tv, BorderLayout.CENTER);
        jf.setVisible(true);

        tv.setBackground(Color.white);

        tv.fitToScreen();

        return cls_data;

    }

}

```

### **J48 Web Service with dot format output**

<code>

```

package source;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import weka.classifiers.trees.J48;
import weka.core.Instances;

/**
 *
 * @author susila
 */
@WebService()
public class J48data {

    /**

```

```

* Web Service operation
*/
@WebMethod(operationName = "mine")
public String mine(@WebParam(name = "file")
String file) throws IOException, Exception {
    Instances data;
    data = new Instances(new BufferedReader(new FileReader(file)));
    data.setClassIndex(data.numAttributes() - 1);
    J48 cls = new J48();
    cls.buildClassifier(data);
    return cls.graph();
}
}

```

```

<?xml version="1.0" encoding="UTF-8"?>
<process
  name="data_mine"
  targetNamespace="http://enterprise.netbeans.org/bpel/BpelModule3/newProcess"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sxt="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Trace"
  xmlns:sxed="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Editor"
  xmlns:tns="http://enterprise.netbeans.org/bpel/BpelModule3/newProcess">
  <import
    namespace="http://enterprise.netbeans.org/bpel/J48dataServiceWrapper"
    location="Partners/J48data/J48dataServiceWrapper.wsdl"
  >
  <importType="http://schemas.xmlsoap.org/wsdl/">
  <import
    namespace="http://source/"
    location="Partners/J48data/J48dataService.wsdl"
  >
  <importType="http://schemas.xmlsoap.org/wsdl/">
  <import
    namespace="http://enterprise.netbeans.org/bpel/J48VisualizeServiceWrapper"
    location="Partners/J48Visualize/J48VisualizeServiceWrapper.wsdl"
  >
  <importType="http://schemas.xmlsoap.org/wsdl/">
  <import
    namespace="http://source/"
    location="Partners/J48Visualize/J48VisualizeService.wsdl"
  >
  <importType="http://schemas.xmlsoap.org/wsdl/">
  <partnerLinks>
  <partnerLink
    name="J48Visualzie"
    xmlns:tns="http://enterprise.netbeans.org/bpel/J48VisualizeServiceWrapper"
    partnerLinkType="tns:J48VisualizeLinkType" partnerRole="J48VisualizeRole"/>
  <partnerLink
    name="J48Data"
    xmlns:tns="http://enterprise.netbeans.org/bpel/J48dataServiceWrapper"
    partnerLinkType="tns:J48dataLinkType" myRole="J48dataRole"/>
  </partnerLinks>
  </process>

```

```

</partnerLinks>
<variables>
<variable name="MineOut" xmlns:tns="http://source/" messageType="tns:mineResponse"/>
<variable name="OperationOut" xmlns:tns="http://source/"
messageType="tns:operationResponse"/>
<variable name="OperationIn" xmlns:tns="http://source/" messageType="tns:operation"/>
<variable name="MineIn" xmlns:tns="http://source/" messageType="tns:mine"/>
</variables>
<sequence>
<receive name="Receive1" createInstance="yes" partnerLink="J48Data" operation="mine"
xmlns:tns="http://source/" portType="tns:J48data" variable="MineIn"/>
<reply name="Reply1" partnerLink="J48Data" operation="mine" xmlns:tns="http://source/"
portType="tns:J48data" variable="MineOut"/>
<assign name="Assign1">
<copy>
<from>$MineOut.parameters/return</from>
<to>$OperationIn.parameters/cls_data</to>
</copy>
</assign>
<invoke name="Invoke1" partnerLink="J48Visualzie" operation="operation"
xmlns:tns="http://source/" portType="tns:J48Visualize" inputVariable="OperationIn"
outputVariable="OperationOut"/>
</sequence>
</process>
</code>

```

## Appendix B-4

### Code Details of Hospital Management Web Service

#### Database Details:

##### Table doctors

```
CREATE TABLE doctors (doctor_name VARCHAR(255), category VARCHAR(255), qual  
VARCHAR(255), address_line_one VARCHAR(255), entryID INT NOT NULL  
AUTO_INCREMENT, available_time VARCHAR(255), PRIMARY KEY(entryID));
```

##### Table patients

```
CREATE TABLE patients (patient_name VARCHAR(255), purpose VARCHAR(255), issue  
VARCHAR(255),age INT, entryID INT NOT NULL AUTO_INCREMENT, time  
VARCHAR(255), doctor_name VARCHAR(255),PRIMARY KEY(entryID));
```

##### Table login\_details

```
CREATE TABLE login_details (doctor_name VARCHAR(255), password  
VARCHAR(255));
```

##### Table patient\_login

```
CREATE TABLE patient_login (p_n VARCHAR(255), password VARCHAR(255),  
PRIMARY KEY(p_n));
```



### Table patient\_history

```
CREATE TABLE patient_history (patient_name VARCHAR(255), purpose VARCHAR(255),  
problem_found VARCHAR(255), prescription_details1 VARCHAR(255),  
prescription_details2 VARCHAR(255), prescription_details3 VARCHAR(255), tests  
VARCHAR(255), test_result VARCHAR(255));
```

### Table rooms

```
CREATE TABLE rooms (room_num INT, status VARCHAR(255), patient_id INT);
```

<b>File doctors.jsp</b>	<ol style="list-style-type: none"><li>1. A connection to the database, named <b>mgmt</b>, is established</li><li>2. Then an SQL query to select all the data in the doctors table is executed The SQL query is: SELECT doctor_name, category,qual,address_line_one, entryID, available_time FROM doctors</li><li>3. The result of the query is stored in <b>rs</b> and then data is retrieved by using the <b>getString()</b> method</li></ol>
<b>patients.jsp</b>	<ol style="list-style-type: none"><li>1. A connection is established to the database <b>mgmt</b></li><li>2. SQL query that selects all the data in the table name called patients is executed and the result set is stored in the variable <b>rs</b>.</li><li>3. Data is retrieved using <b>getString()</b> and <b>getInt()</b> methods</li><li>4. Details of a new patient can be added by using the HTML form</li><li>5. Once this form is submitted control is directed to the <b>patientServlet.java</b> file which contains the doPost method required to update the details to the database table patients</li><li>6. This is achieved by specifying the path of the form <b>form action="/patients" method="post"</b></li></ol>
<b>patientServlet.java</b>	<ol style="list-style-type: none"><li>1. The details of the form submitted are retrieved by using the <b>getParameter()</b> method</li><li>2. These details are stored in variables</li><li>3. Connection is established to the database <b>mgmt</b> and the data in these variables is stored in the table patients by using the INSERT SQL command</li></ol>

	<ol style="list-style-type: none"> <li>4. Since this form is also used to book an appointment and reserve a slot, code to check if the desired slot is available or not is written</li> <li>5. This is achieved by considering the name of the doctor entered and also the desired slot. These values are compared with the details of other patients already existing</li> <li>6. If a particular doctor is already reserved for a particular slot and the same slot is requested by the new patient, then a message “Sorry this slot is already Reserved!!” appears</li> <li>7. If the requested slot is free for the doctor then the details of the new patient are added to the patients table. The SQL query used is:  <pre>"INSERT INTO patients (patient_name, purpose, issue, age, doctor_name, time) VALUES( ?, ?, ?, ?, ?, ? );"</pre> </li> <li>8. The patient name and password entered in the form are added to the patient_login table and is used by patients for logging in</li> <li>9. After this statement is executed the page is redirected to patients.jsp page</li> </ol>
<b>Files rooms.jsp and roomServlet.java</b>	<ol style="list-style-type: none"> <li>1. A connection is established to the database and the SQL command SELECT is used to read all the data in the table rooms</li> <li>2. The entries of the rooms table are displayed along with a form which enables the reservation of a room</li> <li>3. Once the form is submitted control is redirected to the roomServlet.java file</li> </ol>
<b>roomServlet.java</b>	<ol style="list-style-type: none"> <li>1. The details submitted in the form are updated to the table rooms by the SQL command UPDATE</li> </ol>
<b>Files reserve.jsp and reserveServlet.java</b>	<ol style="list-style-type: none"> <li>1. This page provides a form in which a patient is supposed to enter his details</li> <li>2. Once this form is submitted control is directed to the reserveServlet.java file</li> </ol>
<b>reserveServlet.java</b>	<ol style="list-style-type: none"> <li>1. Connection is made to the database</li> <li>2. The values submitted in the form are stored in variables</li> <li>3. The name of the doctor and the requested slot variables are used to compare with the other appointments in order to avoid appointment clashes</li> <li>4. If the requested slot is reserved a message “Sorry this</li> </ol>

	<p>slot is Reserved” appears</p> <ol style="list-style-type: none"> <li>If there is no clash then details of the appointment are displayed</li> <li>Once the appointment is booked the details of the new patient are updated to the <b>patients</b> table and the patient name and password are stored in the <b>patient_login</b> table</li> </ol>
<b>Files login.jsp and loginServlet.java</b>	<ol style="list-style-type: none"> <li>This page provides a login page for the staff to login</li> <li>Once this form is submitted the control is directed to the loginServlet.java file</li> </ol>
<b>loginServlet.java</b>	<ol style="list-style-type: none"> <li>Connection to the database is made</li> <li>The login details of every doctor is stored in the <b>login_details</b> table</li> <li>The entries in this table are selected by using the SQL command SELECT</li> <li>The doctor’s name and password submitted in the form are compared to every row in the login_details table</li> <li>If a doctor’s name matches with the corresponding password in the table then login is successful and the doctor will be able to view his patient’s details</li> <li>The name of the patient along with the patient’s ID number and appointment time are displayed</li> <li>The doctor will be able to prepare a prescription for the patient by clicking on the “Prepare Prescription” link. On clicking the link control is directed to patient_history.jsp</li> <li>The doctor will be able to view patient’s history by clicking on the View History link which redirects the doctor to the patient login page: pat_details.jsp</li> <li>If the login details of the doctor do not match with the details stored in the login_details table a blank page appears and the correct login details should be entered for successful login</li> </ol>
<b>patient_history.jsp</b>	<ol style="list-style-type: none"> <li>This page provides a form for the doctor to fill in the prescription details for a patient</li> <li>Once this form is submitted the control is directed to the patienthistoryServlet.java</li> </ol>
<b>patienthistoryServlet.java</b>	<ol style="list-style-type: none"> <li>Connection to the database is established</li> <li>The details submitted in the form are inserted into the patient_history table using the SQL command INSERT</li> <li>A success message appears and the page is redirected</li> </ol>

	to patient_history.jsp
<b>pat_details.jsp</b>	<ol style="list-style-type: none"> <li>1. This page is the login page for patients</li> <li>2. Once this form is submitted control is directed to patdetailsServlet.java</li> </ol>
<b>patdetailsServlet.java</b>	<ol style="list-style-type: none"> <li>1. A connection to the database is established</li> <li>2. Data from the patient_login table is selected by using the SQL command SELECT</li> <li>3. The details submitted by the login form are compared with the details in the patient_login table</li> <li>4. If the login is successful then the history of that particular patient is retrieved from the patient_history table</li> </ol>
<b>display.jsp</b>	<ol style="list-style-type: none"> <li>1. It displays the categories treated in the hospital. Categories can be like General Physician, Cardiologist, Dermatologist etc</li> <li>2. Once the category is entered in the form and the form is submitted control is redirected to the displayServlet.java</li> </ol>
<b>displayServlet.java</b>	<ol style="list-style-type: none"> <li>1. Connection to the database is established</li> <li>2. Data from the doctors table is selected using the SQL command SELECT</li> <li>3. The category of every doctor in the table is compared with the entered category and details of the doctor belonging to the entered category are displayed</li> <li>4. The names of all doctors who belong to that category are displayed</li> <li>5. A table showing the name of the doctor, available time and the Reserved Slot is shown to enable a patient to see what slots are reserved</li> <li>6. To reserve a particular slot a link "Click here to Reserve" should be used</li> <li>7. On clicking the link the user is redirected to the reserve.jsp page</li> </ol>

## APPENDIX - C

### Data set on QoS

Response Time	Availability	Throughput	Successability	Reliability	Classification
451	23	1.8	24	42	AVERAGE
255.08	12	8.1	13	53	POOR
64.96	18	4.3	18	60	POOR
68.91	19	4.4	20	60	POOR
451	23	1.8	24	42	AVERAGE
136.94	26	3.1	26	67	POOR
542.87	26	4.4	26	53	POOR
382.71	27	5.7	28	73	POOR
501.79	28	4.8	28	73	POOR
316.07	32	1	32	60	POOR
214.62	32	2.3	32	53	AVERAGE
689.42	34	1.1	34	67	POOR
266.83	36	0.9	37	60	POOR
261	36	0.9	37	60	POOR
667.11	38	2.1	38	73	POOR
83.33	39	1.7	40	80	POOR
142	39	6.3	40	73	POOR
430.5	40	3.3	40	73	POOR
464.62	40	3.6	40	78	POOR
49.43	42	10.6	43	73	AVERAGE
334.71	43	4.4	43	73	POOR

173	46	3.8	47	78	POOR
298.83	46	7.7	46	83	POOR
43	47	3.6	47	80	POOR
1423.5	47	2.2	47	78	POOR
1104.67	47	3.1	47	83	POOR
496.43	48	0.8	48	67	POOR
4480.8	52	1	53	50	POOR
408.21	56	5	58	73	AVERAGE
141.77	56	7.5	56	67	AVERAGE
311.25	56	7	56	83	POOR
1987	56	1.9	57	60	POOR
67.25	56	15.5	56	78	AVERAGE
669.1	56	1.3	57	60	AVERAGE
219	56	10.4	56	60	AVERAGE
460	56	2.6	57	60	AVERAGE
130.75	56	10.8	56	83	POOR
146.08	57	10.2	59	53	AVERAGE
203	57	1.2	59	67	AVERAGE
71.75	59	1.2	60	67	AVERAGE
47.27	61	20.3	62	67	AVERAGE
119	61	2.9	62	53	AVERAGE
235.9	61	4	61	73	AVERAGE
184.74	62	7.9	62	67	AVERAGE
130.33	63	7.8	63	73	AVERAGE
271	63	2.7	64	67	AVERAGE
236.67	63	6.4	63	53	AVERAGE
187.75	64	8.1	65	73	AVERAGE
1021.5	65	3.1	65	73	POOR
335.58	65	8.9	66	73	AVERAGE
372.25	66	4.7	66	80	AVERAGE

173	67	0.5	68	73	AVERAGE
193.6	67	3.4	67	73	AVERAGE
196	69	9.7	70	73	AVERAGE
131.25	70	4.8	70	73	AVERAGE
293.5	70	2.5	70	73	AVERAGE
109.75	70	11.4	70	73	AVERAGE
1069.5	71	3	72	83	POOR
1035	71	8.4	72	73	AVERAGE
307.75	71	2.1	71	73	AVERAGE
297.38	71	1.9	72	73	AVERAGE
2440.33	71	1.1	72	67	POOR
154	71	14.4	72	73	AVERAGE
50	72	13.3	72	73	AVERAGE
580.5	72	4.4	72	67	AVERAGE
115	72	12.9	72	83	AVERAGE
42.5	72	13.1	72	73	AVERAGE
198.5	72	15.1	72	83	AVERAGE
266.92	72	1.4	72	80	AVERAGE
42.5	72	13.2	72	73	AVERAGE
203	72	22.1	72	83	AVERAGE
540.5	72	3.7	72	67	AVERAGE
1529	72	5.3	72	83	POOR
1730	72	2	72	67	AVERAGE
205	73	1.4	74	58	AVERAGE
184.67	73	2.7	74	73	AVERAGE
388.5	73	2	73	58	AVERAGE
166.67	75	1.2	75	67	AVERAGE
1316.17	75	1.2	75	73	AVERAGE
136.71	76	2.8	76	60	AVERAGE
59.58	77	8.8	78	67	AVERAGE

511.25	77	4.7	78	73	AVERAGE
60.05	77	9.1	77	67	AVERAGE
241.5	78	7.7	79	67	AVERAGE
1314.75	78	3.5	79	73	AVERAGE
332.25	78	5.9	78	73	AVERAGE
532.75	78	5.5	78	67	AVERAGE
162.25	78	9.1	78	73	AVERAGE
335.5	78	3.6	79	73	AVERAGE
224	78	3.9	79	73	AVERAGE
2836.25	79	2.4	79	73	POOR
107.57	80	1.7	81	67	AVERAGE
131.57	80	2.3	80	53	GOOD
551.79	80	1.8	81	53	AVERAGE
55.5	81	19.1	82	73	GOOD
1360	83	10.4	84	83	AVERAGE
132	83	14.3	84	73	AVERAGE
408	83	15.2	84	83	AVERAGE
499	83	19.7	84	83	AVERAGE
115	83	22.3	84	83	AVERAGE
107	83	31.3	84	73	GOOD
333	83	14.9	84	83	AVERAGE
115	83	22.8	84	73	GOOD
1041	83	12.8	84	73	AVERAGE
171	83	18.6	84	80	AVERAGE
292	83	16.6	84	58	GOOD
136	83	21.4	84	83	AVERAGE
180	83	13.5	84	73	AVERAGE
283	83	20.3	84	73	GOOD
581	83	15.6	84	50	GOOD
664	83	11.4	84	53	GOOD



134.07	84	12.2	85	60	GOOD
389	84	1	84	67	AVERAGE
316.3	84	2	85	53	GOOD
482	85	16	95	73	AVERAGE
269.83	85	4.5	86	53	GOOD
269.09	85	3.9	85	53	GOOD
179	85	1.1	95	60	AVERAGE
431.33	85	9.4	95	67	AVERAGE
239.22	85	7.7	86	53	GOOD
411.83	85	0.4	86	53	AVERAGE
104.5	85	15.6	95	73	GOOD
724	85	13.1	95	50	GOOD
252.35	85	5.3	86	53	GOOD
219.2	85	0.9	95	58	AVERAGE
207.2	85	1.1	95	73	AVERAGE
170	85	3.1	95	60	AVERAGE
287.22	85	3.5	86	53	GOOD
272.43	85	1.9	86	53	GOOD
146.83	85	2.1	95	83	AVERAGE
599	85	10.3	95	73	AVERAGE
133	86	7.7	95	73	AVERAGE
114	86	16.1	86	73	GOOD
67.5	86	6	86	73	AVERAGE
108	86	0.7	95	73	AVERAGE
320.48	86	1.2	86	53	GOOD
229.75	86	10.1	95	73	AVERAGE
645	86	8	86	73	AVERAGE
515.2	86	9.2	95	73	AVERAGE
207	86	13.5	86	73	AVERAGE
119.33	86	1.2	95	73	AVERAGE

227.6	86	14.5	95	73	AVERAGE
283.74	86	3.3	87	53	GOOD
401.62	86	8.1	95	60	GOOD
114	86	17.9	86	73	GOOD
240.6	86	1.2	95	73	AVERAGE
290.3	86	4.4	86	53	GOOD
324	86	23.1	95	73	GOOD
284.65	86	3.1	86	53	GOOD
1296	86	6.9	86	73	AVERAGE
718	86	3.3	87	67	AVERAGE
149.67	86	11.2	95	73	AVERAGE
417	86	6.5	86	73	AVERAGE
274.52	86	4.3	86	53	GOOD
122	86	14.5	86	67	GOOD
167.5	86	16.1	86	73	AVERAGE
287.22	86	5.5	86	53	GOOD
244	86	26.7	95	83	GOOD
333.52	86	3	86	53	GOOD
409.5	86	1.7	87	58	AVERAGE
107	87	1.9	95	73	AVERAGE
294.5	87	14.5	95	73	AVERAGE
179.29	87	2.6	95	60	GOOD
2561.33	87	1.2	96	67	AVERAGE
120	87	24.2	95	67	GOOD
3484	87	5.8	96	73	POOR
339	87	11.7	95	67	GOOD
87.14	87	20.5	95	67	GOOD
158.8	87	5.8	96	73	AVERAGE
235.4	87	1.4	95	73	AVERAGE
213.2	88	1.6	96	73	AVERAGE

113.25	88	1.7	96	53	GOOD
227	88	21.2	96	73	GOOD
617.67	88	7.1	96	73	AVERAGE
239.33	88	9	96	83	AVERAGE
130.14	88	20.5	96	80	GOOD
305.8	88	13.5	96	58	GOOD
143.33	88	15.7	88	73	GOOD
196	88	9.8	96	73	AVERAGE
248.8	88	1.1	96	67	AVERAGE
276	88	1.6	96	67	AVERAGE
1138.33	88	4.7	88	73	AVERAGE
312	88	3	96	73	AVERAGE
302.75	89	7.1	90	73	AVERAGE
3321.4	89	1.4	96	73	POOR
149.67	89	1.1	96	73	AVERAGE
210	89	15.7	96	73	GOOD
320.4	89	12.3	96	73	AVERAGE
490.5	89	4.1	96	73	AVERAGE
116	89	12.9	89	67	GOOD
148.95	89	10.8	96	67	GOOD
229.79	89	7.4	89	67	AVERAGE
242.4	89	1.1	96	73	AVERAGE
262.5	89	5.3	96	73	AVERAGE
245.8	89	1.2	96	73	AVERAGE
221.48	90	10.9	97	53	GOOD
106.75	90	16.2	96	73	GOOD
300.12	90	7.9	97	67	AVERAGE
205.33	90	3.5	97	60	GOOD
122	90	3.9	97	73	AVERAGE
154	90	23.9	97	73	GOOD

136.71	90	3.8	97	60	GOOD
146.5	90	17.6	97	73	GOOD
127	90	23.1	96	67	EXCELLENT
306.8	90	12.7	97	73	AVERAGE
220.6	90	1.3	96	73	AVERAGE
825.8	90	5.9	91	60	AVERAGE
102	90	18.6	97	73	GOOD
102.62	91	15.3	97	67	GOOD
128.31	91	12.4	97	67	GOOD
672.2	91	7.9	97	73	AVERAGE
215.6	91	15.9	97	73	GOOD
498.5	91	4.8	91	60	GOOD
223	91	3.9	97	73	AVERAGE
163	91	33.2	97	73	EXCELLENT
1226	91	6.3	97	73	AVERAGE
121	91	7.9	97	73	AVERAGE
77.9	91	17.1	97	73	GOOD
463.6	91	2	91	73	AVERAGE
474.91	91	5.8	97	60	GOOD
262.5	92	6.9	97	60	GOOD
4207.5	92	1.1	92	80	POOR
500.71	92	2.2	93	67	AVERAGE
227	92	0.9	97	58	GOOD
175	92	4.3	97	73	AVERAGE
334	92	1.5	97	67	AVERAGE
65.4	92	6.8	92	67	GOOD
330.19	92	8.6	97	67	GOOD
142.5	93	4.4	98	73	AVERAGE
305.4	93	12.2	98	73	GOOD
250.4	93	2	98	50	GOOD

135.67	93	2.4	98	58	GOOD
265.09	94	10.2	94	73	AVERAGE
124.17	94	2.1	98	73	AVERAGE
100	94	36.3	98	73	EXCELLENT
126.67	94	9.5	98	73	AVERAGE
215.5	95	3.6	95	78	AVERAGE
197	95	0.8	98	58	GOOD
180	95	14.6	99	73	GOOD
270.5	95	19.1	98	78	GOOD
58	95	16	98	73	GOOD
114.5	95	14.2	99	67	GOOD
93.37	96	13.5	99	67	GOOD
3610.2	96	1.4	99	67	AVERAGE
123	96	14.6	99	83	AVERAGE
115	96	28	99	73	EXCELLENT
144.5	96	22.3	99	67	EXCELLENT
204.6	96	1.9	99	58	GOOD
854	96	14.7	99	73	AVERAGE
905	96	1.6	99	73	AVERAGE
259	97	1.2	99	58	GOOD
151.33	97	6.9	99	73	AVERAGE
91.8	97	24.7	99	73	EXCELLENT
233.2	97	2.1	99	73	AVERAGE
239.05	97	8.8	99	67	GOOD
585.5	97	1	99	60	GOOD
56	97	9	99	73	GOOD
324.67	97	4.1	98	80	AVERAGE
134.08	97	15.4	99	73	GOOD
123.92	97	15.6	99	73	GOOD
140.5	97	18	99	73	GOOD

126.17	98	12	100	67	GOOD
255	98	1.3	99	67	AVERAGE
301.8	98	1.2	100	73	AVERAGE
383.2	98	2.1	100	73	AVERAGE
352	98	21.5	100	58	EXCELLENT
789.8	98	2.2	95	73	AVERAGE
597.5	98	3.7	100	60	GOOD
243	98	25.6	100	73	EXCELLENT
63.25	98	25.6	100	67	EXCELLENT
154	98	14	99	73	GOOD
111	98	0.8	100	67	AVERAGE
202	98	9.7	99	80	AVERAGE
248.4	98	1.4	99	58	GOOD
443	98	10.1	100	58	GOOD
265.4	99	1.5	100	67	AVERAGE
109.6	99	19.3	100	73	GOOD
148.56	99	2.3	100	73	AVERAGE
724.82	99	4.7	100	60	GOOD
570.5	99	8.3	99	67	GOOD
63.8	99	18.1	100	73	GOOD
470	99	0.5	100	60	GOOD
100	99	19.8	100	73	GOOD
121.67	99	9.6	100	73	GOOD
166.63	99	9.6	100	67	GOOD
219	99	16.4	99	80	GOOD
95.25	99	16.3	100	73	GOOD
322.33	99	9	100	73	AVERAGE
293	99	27.8	100	73	EXCELLENT
1334	100	8.3	100	73	AVERAGE
184	100	12.1	100	80	AVERAGE

300.6	100	1.4	100	67	AVERAGE
2123	100	8.8	100	73	AVERAGE
510	100	13.9	100	73	GOOD
113.75	100	1	100	73	AVERAGE
125	100	17.4	100	67	GOOD
182.33	100	11.6	100	73	GOOD
51.5	100	1.8	100	73	AVERAGE
223.6	100	0.7	100	73	AVERAGE
257.5	100	3.4	100	73	AVERAGE
228.6	100	14.1	100	73	GOOD

## LIST OF PUBLICATIONS

### International Journals

1. Susila, S.; Vadivel “**Agent based** discovery of Web Services to enhance the quality of Web Services selection”, International Journal of Computer Science and Network Security (IJCSNS), Dr. Sang H. Lee. Vol.11, No.2, pp 159-163. ([http://paper.ijcsns.org/07\\_book/201102/20110226.pdf](http://paper.ijcsns.org/07_book/201102/20110226.pdf) Feb’2011 paper id (2101109))
2. Susila, S.; Vadivel “**Web Services selection based on QoS attributes using entropy discretization method**”, International Journal of Computer Applications(IJCA) Published by Foundation of Computer Science, New York, USA  
<http://www.ijcaonline.org/archives/volume30/number2/3611-4119>
3. Susila, S.; Vadivel“Web Services selection through QoS Web Services”, Published papers in the International Journal of Software and Web Sciences (IJSWS), ISSN (Online): 2279-0071, ISSN (Print): 2279-0063 (September-November, 2013, Issue 6, Volume 1). Pp 18-23 <http://iasir.net/IJSWSpapers/IJSWS13-325.pdf>
4. Susila, S.; Vadivel“Qos Measurement Tool For Web Service Selection” International Journal of Web Engineering2014;3(1): 1-8doi:10.5923/j.web.20140301.01  
<http://article.sapub.org/10.5923.j.web.20140301.01.html>



## International Conferences

5. Susila, S.; Vadivel, S.; "A novel approach to add semantics to Web Services", International Conference on Recent Advances in Applied & Biomedical Informatics and Computational Engineering in Systems Applications. Proceedings of the International Conference on Recent Advances in Applied & Biomedical Informatics and Computational Engineering in Systems Applications. Florence, Italy Pages 59-64. Online at <http://www.wseas.us/books/2011/Florence> August 23-25,2011 paper id(1111201)
  
6. Susila, S.; Vadivel, S.; Julka, A., "**Broker architecture for Web Services selection using SOAPUI**," *Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on* , vol., no., pp.219,222, 8-10 Dec. 2012 doi:10.1109/ICCCTAM.2012.6488102  
URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6488102&isnumber=6488050>

## **BRIEF BIOGRAPHY OF THE CANDIDATE**

**Ms. Susila** received the ME degree in computer science from College of engineering, Guindy, ANNA UNIVERSITY, Chennai in 2006. She worked as Application development Specialist in Intellicon Pvt. Ltd, India. Then she worked as Lecturer in engineering colleges that are affiliated to ANNA UINVERSITY in Coimbatore. She is currently working as Senior Lecturer in the Computer Science Dept of BITS, Pilani-Dubai from 2006. She continues her service in the same campus till date. Her current areas of interest are selection of Web Services based on QoS attributes of Web Services and security.

## **BRIEF BIOGRAPHY OF THE SUPERVISOR**

**Dr.S.Vadivel** received the PhD degree in Computer Science and Engineering from I.I.T Madras, India by 1989. After that he worked in Crompton Greaves in Bombay as research executive for 3 years. Then he worked as Assistant Professor in Engineering in Government College at Tamil Nadu, India for 4 years. Then he joined as Research Lead in Think business networks a multinational software company in Tamil Nadu. He has joined BITS, Pilani-Dubai as faculty in CSE by Jan 2003 and currently working as professor in CSE in the same institute. He has 25 publications in various international journal and conferences. His current research interests are in Web Services and security, embedded controllers, data mining, and Architecture of enterprise software applications.