GUISET DIRECTED COMPARATIVE STUDY OF FREE AND OPEN SOURCE WEB SERVICES PLATFORMS

IJEOMA NOELLA MBA

2013

GUISET DIRECTED COMPARATIVE STUDY OF FREE AND OPEN SOURCE WEB SERVICES PLATFORM

Ijeoma Noella Mba
(200814292)

A dissertation submitted in fulfillment of the requirements for the degree

Of

Master of Science in Computer Science

Supervisor: Prof M.O. Adigun

Department of Computer Science,

Faculty of Science and Agriculture

University of Zululand

DECLARATION

I, Mba Noella Ijeoma declare that this dissertation represents my work and it has not been submitted in any form for another degree or diploma at any University or any other Institution of tertiary education. All information taken from published and unpublished works of others has been well acknowledged in the text and a list of the respective references is given.

_____

Mba Noella ijeoma

DEDICATION

This work is dedicated to Almighty God who in His infinite mercy sustained and strengthened me throughout the course of this work.

To my family for the belief they have in me, the courage, support, love and care shown to me through the most trying times

ACKNOWLEDGEMENT

To God be the glory for the great things He has done towards me throughout the course of this research work.

My deepest gratitude and appreciation goes to my supervisor Prof. M.O. Adigun for his excellent guidance, care, vast reserve of patience and knowledge and providing me with an excellent atmosphere for doing research.

My sincere gratitude goes to Prof. S.S. Xulu for all his fatherly advice, guidance and support.

A large number of people within the Department thoughtfully commented on the drafts of this research work. I, therefore, recognize and thank the following for their particular contributions: E. Jembere, B. Mutanga, P. Mudali, P. Tariwei, E. Olatunji and J. Iyilade.

My deep sense of gratitude goes to all the staff of Department of Computer Science for their contribution to this work.

On a daily basis, I have been blessed to work with a pleasant and jovial group of fellow students. To all my fellow students at the Center of Excellence, thank you very much.

I am grateful to many families who took me in as their own and made me feel at home in South Africa, especially the Chukwu family, the Kolawole family and the Orisakwe family.

I express my thanks to the following persons who have in one way or the other impacted in my life during the course of this research work: Dr. O. Ekabua, Dr. E. Etobe, Linda Nyamen, Francis Akpagu, Ike Paul, Ngozi Uzuegbu, Dr. Joseph Ikwegbue, Precious Nwachukwu and Francis Olaniyi.

To a special friend and sister, Stella Ochiogu, whose friendship, kindness, understanding, and good judgment have sustained, enlightened, and entertained me over the years of our friendship. Thanks for always being there for me.

I am indebted to my mother, Mrs Josephine Mba, for her care, unconditional love, constant inspiration and guidance which has kept me focused and motivated. Words cannot express my gratitude to you. Thank you, mom.

Finally, I would like to thank my husband and soul mate, U. K. Ezeji. He was always there cheering me up and stood by me through the good and bad times. Nkem, you are the best thing that has ever happened to me. You have consistently helped me keep perspective on what is important in life and shown me how to deal with reality.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ADB	Axis Data Binding
API	Application Programming Interface
AXIOM	Axis Object model
CDDL	Common Development and Distribution License
CXF	Codehaus XFire
DOM	Document Object Model
ESB	Enterprise Service Bus
FOSS	Free and Open Source Software
FTP	File Transfer Protocol
GPL	General Public License
GUI	Graphical User Interface
GUISET	Grid-based Infrastructure for SMME Enabling Technology
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
IT	Information Technology
JBI	Java Business integration
JMS	Java Message Service
JSON	Java Script Object Notation
MTOM	Message Transmission Optimization Mechanism
P2P	Point to Point
RAM	Random Access Memory
REST	Representational State Transfer
SAAJ	SOAP with Attachments API for Java
SaaP	Software-as-a Product
SaaS	Software-as-a Service
SAX	Simple API for XML
SCA	Service Component Architecture
SMME	Small Micro and Medium Enterprises
SMTP	Small Message Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
StAX	Streaming API for XML
URI	Uniform Resource Identifier
WSDL	Web Services Description Language
WSRF	Web Services Resource Framework
WSS	Web Service Security Standard
XML	eXtensible Markup Language
XOP	XML Optimized packaging

ABSTRACT

The aim of GUISET (Grid-based Utility Infrastructure for SMME Enabling Technology) architecture is to provide and deliver e-services to Small Medium and Micro Enterprises (SMMEs), who are most of the time resource-constrained, by taking advantage of transitions and technological developments toward Service Oriented Architecture (SOA) within the software industry. Web Services (WS) is one of such technology that has received widespread adoption within SOA for implementing services. However, the acceptance of Web Services has been facilitated by implementation of core WS-Standards into enterprise application development platforms and tools within both the proprietary and Open Source Communities.

Although, for the target group of GUISET, namely the SMMEs, the Open Source platforms seem a natural choice as proprietary tools are often expensive and are out of their reach. However, even within the open source communities there are many platforms with different characteristics and features. A major challenge is how to choose the appropriate platform that is suitable for GUISET services scenario. One question that is crucial to addressing this challenge (especially within the context of GUISET) is: can a comparative study of existing WS development platforms be used in recommending one for GUISET services? This is the central goal of this research. Our aim was to test the performance of a few leading open source WS platforms with the aim of determining their suitability for developing GUISET services under an e-commerce scenario and thereby use this to recommend an open source WS platform for GUISET services.

In addressing and answering the question, three web services platforms were selected based on some specified criteria for evaluation, namely Axis2, CXF and Metro. The evaluation was performed theoretically and experimentally. Theoretically, the evaluation presents some quality attributes of the selected platform and experimentally, the evaluation presents their runtimes based on marshalling, unmarshalling and round trip times. The result shows that Metro performs better than either Axis2 or CXF. Metro is therefore the recommended platform but highlights on the potential of Axis2 and CXF were given in cases where they might be beneficial.

CHAPTER ONE
INTRODUCTION

1.1 Overview

Service Oriented Architectures (SOA) are gaining popularity as businesses tend to be more responsive and flexible in their communication and meeting customer's need through new and existing technologies (Hodgman, 2007). SOA enables information sources and software functionality to be delivered as individual business service units, which are distributed over a network and also combined to create business application which helps in solving complex problems. SOA is implemented with the Web service (WS) idea. WS as well as grid services are attracting significant industry interest as a low cost and flexible technology alternative for delivery of on-demand business processes. As these technologies becoming well accepted, so is the platform for its development and deployment and Web Service Platform is one of such tools.

The construction of SOAP (Simple Object Access Protocol) processors such as clients, servers and gateways through the use of a framework is known as Web Service Engines. According to (Bayer, 2010), "A SOAP engine is considered a server or a library that transforms SOAP messages into invocations and vice versa" thus it is considered a necessary requirement for web service invocation (Markus *et al*, 2009). For any Web Service Provider toolkit, the core layer is said to be the SOAP engine layer (Asif *et al*, 2007). Web Service engines and tools

provide a way for efficient compression, and representation of data and also efficient processing of XML.

A technology that has received wide adoption within the Service Oriented Computing (SOC) community for implementing Service Oriented Architecture is Web Services (Legner & Heutschi, 2007). Web Services provide standards-based interfaces for service description, discovery, and message definitions to invoke the services. Despite the fact that Web services and service oriented architecture have gained ground with the assurance of solving problems in traditional software, the key factors that compel software professionals to embrace Web services and related technologies are interoperability, modularity and composability; there is then the need for tools that meet the key requirements for development and deployment.

The Popularity of Web Services technology has enhanced the enterprise adoption of the Web services core standards. This enterprise adoption progression is made possible by what Web Service Engine has to offer based on the support from both the open source community and product vendors (Markus *et al*, 2009). Owing to the source code and the entire build system of an open source web service engine that is made available, the SOAP engine can be modified based on the requirements of a particular software development: allowing the removal of unused parts and reduction of the footprint of the software.

A SOAP engine is equipped with some interesting functionalities such as the ability of adding Web Services support to a software product. A SOAP engine packaged together with the

software product may allow its details to be hidden from the client and then the installation guideline can be obtained through the software product guide. It all depends on which is most convenient to the specific application that it is meant for, e.g. e-commerce.

SOAP messages are simple Extensible Markup Language (XML) documents (Box *et al*, 2000). A SOAP engine often has to do more than process SOAP messages. For example, Web Services Description Language (WSDL) documents have to be created, requests have to be sent out and the lifecycle of service objects has to be managed properly. In all, new standards and requirements also apply.

This research work particularly focused on evaluating the performance of different Web services engines focusing on deploying e-commerce on-demand web services requirements of the GUISET (Grid-based Utility Infrastructure for SMMEs Enabling Technologies) architecture as it concerns SMMEs (Small Micro and Medium Enterprises). It starts by inspiring the need for testing the performance of Web Services Platforms and continues by giving a brief background on the move from Software-as-a-product (SaaP) to Software-as-a-service (SaaS) with the benefits of its adoption by the Service Oriented Computing Community and then establishes the problem this dissertation is tackling. The goal and objectives set out to address the said problem and also the importance of the research were established. The methodology that was employed in addressing the problem is discussed next and finally the organization of the rest of the thesis is introduced.

1.2 Statement of the Problem

SOA primarily is based on reusability and interoperability. The idea of SOA is that any business utility that can be reused is a principle candidate to be exposed as a service. The service idea is particularly, relevant to the need to empower SMMEs as active providers of software services. This idea formed the bedrock of our research center's focus known as Grid-based Utility Infrastructure for SMME-Enabling Technologies (GUISET). A major requirement in our GUISET infrastructure is a well researched adoption of a suitable (Web Service) platform that would facilitate rapid development and deployment of service components.

Web Services is currently the accepted industry standard for implementing SOA. Implementation of Web services can be carried out on both proprietary and open source platforms. The open source option seems the way out for SMMEs, which are constrained by limited or no access to expensive software tools. But a critical question they face is: Which open source platform has the right functionality and performance for GUISET Web Service development in some specified scenario?

Existing approaches to software testing have focused mostly in the area of Object-oriented and Component-based development tools and platforms. The evaluation of platforms for SOA has remained in its infancy partly because research in the area is still emerging. It was, therefore, the aim of this study to carry out a comparative study of some leading open-source platforms for developing Web services.

1.3 Research Question

The key research questions addressed by this work are:

I. How can a comparative study of existing web services development platforms be used in recommending platforms for GUISET services?

II. Which open source platforms have the right functionality for GUISET Web Service development in some specified scenario?

1.4 Rationale of the Study

The SMMEs have been identified in various studies as crucial to the economic empowerment and development of most developing countries of the world. However, many of them are still lagging behind in access to digital opportunities. They lack the required infrastructure and tools to participate in the global marketplace. Various governments and non-governmental organizations have, therefore, rolled out a lot of initiatives to address this challenge faced by SMMEs.

It is in view of complementing initiatives addressing the challenges that our research center has crafted an architecture named GUISET (Grid-based Infrastructure for SMME-Enabling Technologies). The main focus of GUISET is to provide an e-infrastructure for on-demand service delivery to the SMMEs. The GUISET idea would meet the above mentioned challenge for the following reasons:

i) First, it is based on utility computing concept where the consumer of the service does not need to own the infrastructure that provides the service but pay-per-use.

ii) Second, it follows a service-oriented computing approach, which means that the cost of application development can be drastically reduced through reuse of existing service components available on the network. The import of this is that SMMEs are not only consumers of services, but can act as active providers of services and thereby shift the traditional dominance of big corporations as providers in software market.

Various research endeavors within our research center are geared towards realizing the GUISET idea. However, a major issue that was yet to be addressed by our research was the choice of the right open source platform that meets some criteria for developing services. This research aimed to address this issue by conducting a comparative analysis of existing open-source platforms for developing Web Services.

The evaluation of Software is an entire field in the Information Systems profession that includes usability testing, functionality testing, performance testing and benchmarking. While, there are various studies carried out to test performance of tools for Object-Oriented and Component based application development environments, very few studies have been targeted at performance of Service oriented tools and thus the need for this study. It was envisaged that at the successful completion of this study, it would help Web service developers to decide on the relative merits in using specific open source platforms for specific scenarios.

1.5 Research Goal and Objectives

The expected outcome of the research as was conducted and the objectives used towards the goal is as described below.

1.5.1 Research Goal

The goal of this research was to evaluate the performance of a few selected open source web service development platforms with view to recommend one for developing a GUISET services.

1.5.2 Research Objectives

The above stated goal was realized as the equivalent of the following set of objectives:

I. To form selection criteria for web services engines to be investigated.

II. To provide a comprehensive taxonomy of features of selected Open-source platforms for Web Service development.

III. To setup an experiment that would compare the performance of selected platforms.

IV. To use the results to recommend a suitable platform for GUISET service development.

1.6 Research Methodology

The methodology that was employed in this study was both theoretical and analytic. It involved the following research activities; Literature survey, Choice of Testing Tools and Formulation of Test Cases, and Experiments to test the Web Service platforms.

1.6.1 Literature Survey

A comprehensive survey of existing body of work relevant to this study was carried out as it concerns SOA, Web services, SOAP engines, and software testing. Particularly, investigation of features of some leading open source platforms for Web services was performed and was used to come up with a set of required features in GUISET. Existing tools for software testing were then assessed and their applicability to our test case was identified.

1.6.2 Choice of Testing Tools and Formulation of Test Cases

Based on knowledge gained from literature survey, criteria for selection of test cases were formed, a list of test cases for the performance analysis was drawn up and design criteria specified for the performance evaluation. The Testing tool selected for performance testing was SOAPUI, which is a tool for testing SOA and the following performance metrics were considered; Marshalling Time, Unmarshalling Time and Round Trip Time.

1.6.3. Testing Experiment

This phase of the work was carried out in two different approaches. First is the Qualitative analysis where a theoretical qualitative comparison of each selected tool was presented with regards to their architectures and underlying models. Second, is the Quantitative analysis where performances testing of the selected SOAP engines with their native binding frameworks were carried out to test the speed at which they can process XML documents based on the metrics identified.

1.7 Organization of Dissertation

The remainder of this dissertation is organized as follows: Chapter two presents an in depth state of the art review of theories and concepts of SOA and platforms for its deployment. Chapter three presents some of the relevant related works in software evaluation of open source tools and classifying them based on their limitation to this project. The criteria for selection of each of the Web Service engine under investigation were described and qualitative analyses of each of the selected platforms were performed with merits and demerits were explained in Chapter four. Chapter five describes the quantitative analysis, where the test environment, performance evaluation and analysis of results of the performance test as was performed were highlighted and analyzed. In conclusion, chapter six gives a summary of the entire research work performed and proper recommendation of the evaluated SOAP engines and possible future work.

CHAPTER TWO

BACKGROUND CONCEPTS

2.1 Introduction

There is growing recognition of the role of Service oriented computing as it regards SOA and Web services in the realization of reusability of assets, interoperability and flexibility among platforms and other technologies as well as reduction in both operational and developmental cost. A Web service is a middleware technology for the purposes of integrating enterprise applications over the Internet (Ng *et al*, 2004). The integration is achieved by Web services support for loosely coupled SOA and messaging, which further allows heterogeneous systems and applications to be programed in different languages and technologies to interoperate with one another. A good number of applications in areas such as scientific computing, e-commerce grid computing, utility computing and finance have been exposed as Web services. Owing to this, many platforms exist for Web services implementation. The question that might arise is what performance level is shown by most Web service implementations.

As was specified in the preceding chapter, the goal of this research is to evaluate a few leading open source web services platform. Therefore, this chapter of background concepts serves as a foundation for the theories which this study was built upon and it is based on literature within the area of SOA and its implementation - Web Services and underlying architecture with a special focus on SOAP and WSDL and most importantly, the GUISET architecture. This literature is supposed to give enough knowledge within the subject area especially as it concerns open source tools and platforms for developing web services and to enable the understanding of

performance of software testing on the specific tools (Web service engines and Binding frameworks) which this research work is meant to cover.

The Chapter is divided into five sections. The first section provides a general overview of our Centre's broad research project, GUISET and the specific requirement of GUISET which this research work is meant to address, while section two discusses SOA, Web Services as its implementation with an extension on Web services standards and an insight on Grid services. Section three describes development environment for Web Services with focus on open source platforms (its requirements and taxonomy of features). We further overview of Web services engines, Data Binding Frameworks which are the basis of software testing that this research work addressed. Section four then discusses software testing, as it concerns this research work. The concluding section is followed by a summary of the chapter.

2.2 Service Oriented Architecture

Service Oriented Architecture (SOA) has gained a lot of attention over the last years as it was viewed as an architectural paradigm and discipline that builds infrastructures which enables service consumers and service providers to work together across heterogeneous domains of technology and ownership (Nickull *et al* 2007). Services in SOA are the main catalyst of electronic data communication but still they have need of further mechanisms to be able to function.

Service Oriented Architecture (SOA) as a paradigm is used for organizing and utilizing distributed capabilities that may be under the control of different ownership domains and

implemented using various technology stacks (MacKenzie *et al*, 2006). In general, organizations can create or provide services that can be used in both solving and supporting solutions for the shortfalls encountered in the course of running their businesses. It is then expected that a service need can be met by services offered by service.

An apparent significance of SOA is that it provides a good structure for matching both the services provided and the services consumed and combining them to tackle the needs of other services by pulling other services (Nickull *et al*, 2007). Therefore it can be said that SOA is a "view" of architecture that focuses in on services as the action boundaries between the needs and capabilities in a manner conducive to service discovery and repurposing.

According to (Myung-Hee et-al, 2009), the main idea of SOA is to provide loose-coupled components between software components in a view of service implementation and to realize business services in a view of enterprise goal (Myung-Hee et al, 2009). More than anything, SOA is an architecture that means it does not necessarily describe the technology involved in computing, but defines a framework which allows programs and services to interact. To actually function, a SOA-based system needs the technology parts, i.e. a set of commonly agreed-upon computing standards. SOA are basically standardized, which could be termed industrial standard and implementation standard. Web Services and relating technologies are still in development but acts as SOA's implementation standard.

2.3 GUISET

GUISET which is an acronym for Grid Based Utility infrastructure for SMME- Enabling Technology is a research project based on the concept of Services (Service Oriented

Architecture, web Services and Grid Computing), Utility Computing and e-commerce. According to (Kabanda *et al*, 2006), GUISET is motivated by the emerging Service Oriented Architecture and ongoing technological convergence between grid services and web services which is creating trends towards Information technology (IT) service provisioning as utilities and the provision of such a technology that will be affordable for Small Medium and Micro Enterprises. The GUISET research project envisaged a future where small businesses (SMME) will act as service providers and provide services at a cost to its clients based on quality of service requirements.

The main idea of GUISET is to provide an e-infrastructure, which would enable SMMEs to pool their resources and expertise together for the sharing and collaboration among themselves and their partners (Adigun *et al*, 2006). The proposed GUISET architecture categorically stated the need for a concrete application building and deployment infrastructure.

However, for GUISET to effectively function in meeting the business needs of the target SMMEs, it must be built /deployed upon a development environment that would fulfill the needs of SMMEs. In fulfilling the needs of SMMEs and eliminating the barriers to successful implementation of IT in small businesses, the GUISET architecture was formed.

2.3.1 The GUISET Architecture

GUISET architecture is a typical Service Oriented Architecture with three layers that uses resources from a pool to grant service requests. These layers are multimodal interfaces layer, Middleware layer and Grid infrastructure layer.

The multi modal interfaces layer of GUISET comprises, integration of application interfaces which are meant as SMME enablers. Here, services would be exposed as Interfaces of different applications, therefore, rendering information to the user. **The middleware Layer** is the utility broker which enables for dynamic service selection and resource sharing. The middleware layer was seen as an integral part of this research work since the SOAP Implementation is a middleware technology.

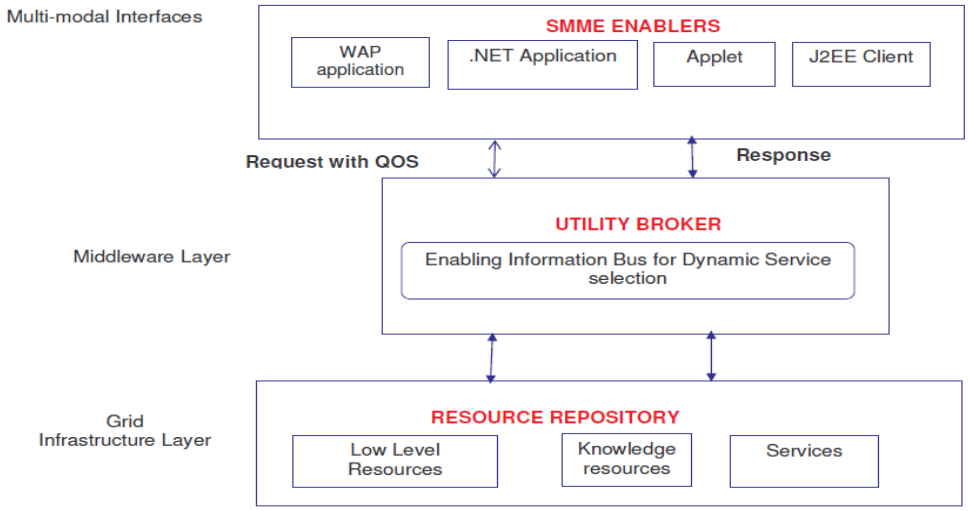


Figure 2.1: The GUISET Architecture (Adigun et al, 2006)

The grid infrastructure layer, which is the resource repository, is made up of low level resource, knowledge resource and services. For the context of this research work, services are more of paramount importance and in a Service Oriented Architecture, all the concept of the architecture is viewed as services. GUISET as a Service Oriented Architecture is assumed to have the functionalities applicable to Service Oriented Architectures like the delivery of Software as a service and the publish-find-and-bind principle of Web services

2.3.2 GUISET Services

In the context of GUISET as architecture, there would be a set of functions that GUISET is meant to accomplish as a Service Oriented Architecture with Web Services as its implementation standard. As a Service-Oriented Computing Architecture, GUISET utilize services as fundamental elements for developing application, therefore, the services are basically derived from existing e-commerce on demand services and they are as follows; From the business point of view as was obtained from existing e-commerce on demand platforms: Cataloging, Product Categorization, Software as a service, Smart Search Capabilities, Storefronts Order processing, Order Integration, Credit Card Processing, Personalization etc. (Demandware Whitepaper, 2008), but from technology point of view as it is based on web services, the services are envisaged as follows: Service Provider, Service Broker and Service requester.

In the case of being a Service provider, GUISET will act as a platform that holds the implementation of the service, as a broker, will act as both public – where services are universally accessible, and private – where only specified sets of service requesters are able to access the service and also as a service requester, will then act as an application that looks for and invokes a service.

2.4 Web Service as an SOA Implementation

Recently, Web Services technology, especially those technologies based on WS- Standards played a key role in the realization of SOA thus acting as its implementation standard (Mahmud, 2005). Although “web service” is a well-known term, it has no universally accepted definition. The definition of Web Services has aroused arguments within the W3C Web Services Architecture Working Group (Muracevic *et al*, 2009). According to W3C, “A Web service is a

software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards" (Booth *et al*, 2004). The puzzlement about Web services arises because there is confusion with the enabling technology not the concept. Web services in general is viewed as the subsequent stage in the progression of the Internet, beyond static web pages and transactional Web sites, Web Services are about application talking to other applications over the Internet via XML-based middleware standards.

In spite of the intricacy of describing web services, there is a general acceptance that the web service concept is an SOA with some limitations to it: Interfaces have to be based on Internet protocols such as HTTP (Hyper Text Transfer Protocol), FTP (File Transfer Protocol), and SMTP (Small Message Transfer Protocol). Except for binary data attachment, messages are required to be in XML.

The goal of the Web Services effort is to achieve interoperability between heterogeneous systems and applications by using Web standards. Therefore, Web Services use a loosely coupled integration model to achieve flexible integration of heterogeneous systems in a variety of domains including business-to-consumer, business-to-business and enterprise application integration (Tingbin *et al*, 2007). Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description, Discovery, and Integration (UDDI) are the

specifications that initially define Web Services. WSDL and SOAP are seen as the two most important protocols due to its direct relationship with messaging.

WSDL introduces and provides the language needed in specifying information required for communication with a Web Service, this includes the service interface, location and the various communication protocols it supports. SOAP is an XML messaging protocol and communication protocol used for basic service interoperability and information exchange in both heterogeneous and distributed environment.

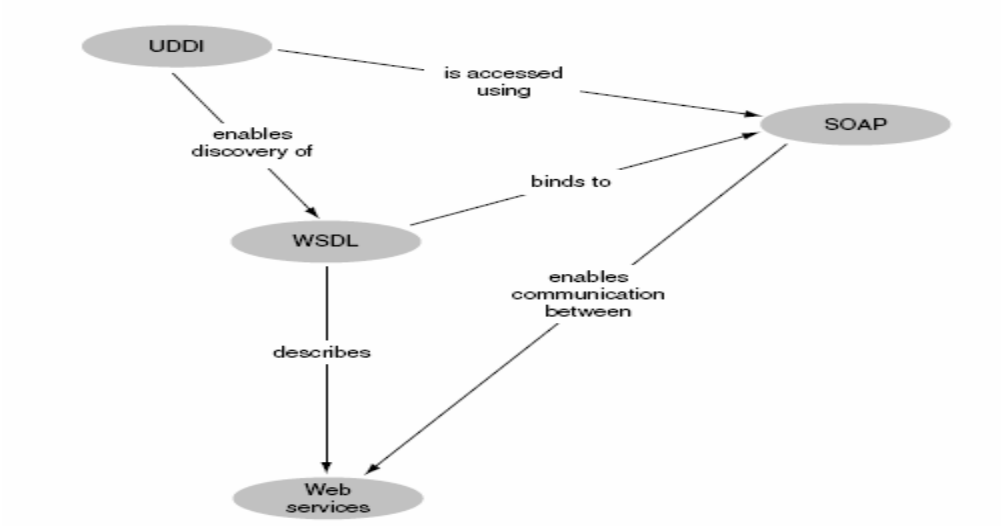


Figure 2.2: Service roles and interactions

In addition, the fundamental messaging infrastructure used by other components and services of Web Service in a typical Web Service architecture is provided by SOAP. SOAP is a lightweight, stateless, communication protocol that provides an extensible XML framework for message exchange in a distributed environment. However, it is not tied to any programing language and it can also be used over a number of transport protocols (Govindaraju, 2004). The structure of a SOAP message is an envelope containing a header and a body. The message payload is carried

by the body and the header carries the other optional components. SOAP, by design, is independent of network protocols, but most current SOAP implementations use the HTTP binding due to its wide availability and ability to go through firewalls.

UDDI provides the infrastructure required to publish and discover services in a systematic way. Together, the specifications allow applications to find each other and interact following a loosely coupled, platform independent model.

There is also another technological different web services in existence: Rest Web service. Rest Web services are presumed to be more web-oriented, loose coupled, efficient, intuitive and simple. On the other hand, this approach is less formal and does not provide user with standard description language, discovery service or complex security model, which is delivered by the SOAP features. That is why SOAP approach of Web Services is more likely used by big enterprises, which are not likely to risk and is more conservative environment. REST aims to take advantage of all HTTP and Web features and it consists of a URI, HTTP AND XML. REST as defined by Rodrinquez, is “a set of architectural principles by which Web services that focused on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages can be designed” (Rodrinquez, 2008).

W3C and OASIS Open acts as the two standardizing bodies which works on Web Services core standards and their enhancements. Most of these specifications that is relevant for this research work will be addressed later in this chapter. In all, Web Services has also emerged as

the architecture of choice for grid standards such as the Web Services Resource framework (WSRF) (Globus Alliance, IBM and HP, 2005). Web service based specifications are now widely used to represent, discover and communicate grid services.

2.4.1 Grid Services

Before going on to discuss Grid Services, Grid computing on itself should be described. Grid computing is perceived as a form of distributed computing in which the use of disparate resources such as compute nodes, storage, applications and data, often spread across different physical locations and administrative domains, is optimized through virtualization and collective management (Srinivasan *et al*, 2005). According to (Ying *et al*, 2004), Grid computing is shifting towards a Service Oriented Architecture based on web services framework owing to its ability for resource and power sharing, and high degree of usability and interoperability. Thus Grid Services are seen as Web Services with improved characteristics and services (Li *et al*, 2005). From the understanding of Web Services as the technology of choice for Internet-based applications with loosely coupled clients and servers makes them the natural choice for building the next generation of grid-based applications. However, Web Services do have certain limitations. In fact, plain Web Services (as currently specified by the W3C) would not be helpful for building a grid application thus the extension to grid services.

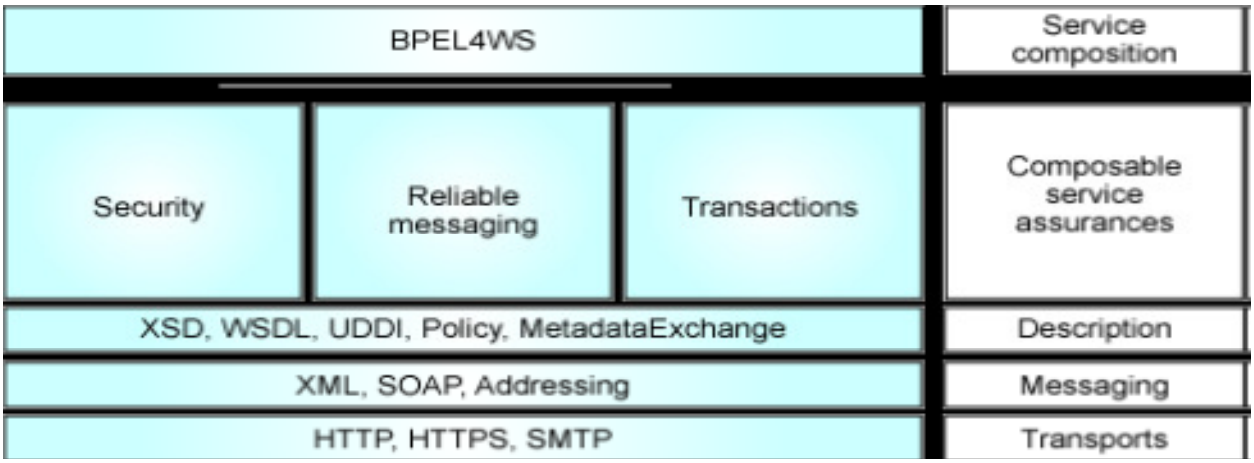


Figure 2.3: Web Services Protocol Architecture (Ferguson et al, 2003)

A Grid service is defined simply as a Web service that follows a set of conventions which could be interfaces and behaviors that describes the way a client interacts with a Grid Service (Foster *et al*, 2002). For instance, Grid services can be defined in terms of standard WSDL (Web Services Definition Language) with minor extensions, and exploit standard Web service binding technologies such as SOAP (Simple Object Access Protocol) and WS-Security (Web Services Security) (Grimshaw & Tuecke, 2003).

2.4.2 Web Services Standard

Web Services are now one of the key technologies in today's software industry. As a result, there is a rapid development process and the stack of interrelated standards that characterize the Web Services infrastructure is maturing continuously. The growing collection of WS-* standards, supervised by the Web Services governing bodies defines the Web Service protocol stack, as shown in the web services protocol architecture depicted as Figure 2.3 of this chapter. The figure is a graphical representation of the different categories of standards and specifications and how they are contained within the context of a web services framework.

Here, we will be looking at the standards that are specified in the most basic layers: messaging and description and discovery.

The messaging standards are intended to give a framework in order to exchange information in a distributed environment. These standards have to be reliable so that the messages are sent only once, and only the intended receiver receives them. This is one of the primary areas wherein a lot of research work is being done, because everything depends on the messaging ability.

2.5 Tools and platforms for Implementing Web Services

There are a lot of tools in both the Market-place and in the Open Source communities for developers to use in building and deploying Web services and Service Oriented Architecture (SOA) applications. As stated by the Soap Ware organization as of 2006 that they are more than seventy Web services toolkits for a variety of platforms and programing languages like C, C++, Dephi and Java. These toolkits differ in their languages, implementation model, API, binding style and even performance (Shirasuna *et al*, 2002).

Tools and Platforms for implementing Web services are seen as general-purpose technology platforms which functions beyond its basics. Gartner believes that the significant influence exhibited by most web service platforms causes Web services technologies and standards to be good indicators of their effect on enterprises' service-oriented architecture (SOA) implementations and software architectures in general (Smith *et al*, 2005).

A Web services platform presumably is an integrated entity that implements Web services specifications to enable the development, execution and management of SOA applications and

content based on Web services technologies. The individual parts of a Web services platform cannot serve as a comprehensive platform unless they have underlying integrative processes and technology. Web services Platform provides three core subsystems which are Invocation, Serialization and Deployment and the former two are basically messaging (Hassen, 2002).

2.5.1 Web Services Engine

Web Service Engines is a framework for the construction of clients, servers, gateways, libraries and others known as SOAP processors. A Web service engine that is based on the SOAP protocol is known as a SOAP engine. A SOAP engine is a SOAP processor that transforms SOAP messages into invocations and vice versa (Axis User Guide, 2005). The core layer of any Web Service Provider toolkit is the SOAP engine layer (Bayer, 2010). However, the main function of the SOAP engine is manipulating SOAP to a certain extent. Since this work is basically concerned with the open source SOAP engines, the description of some its feature is given hereunder.

The Open source SOAP engine makes its source code available. The source code that is made available makes it possible to modify the SOAP engine according to the needs of the particular software development: it is feasible to navigate, edit or remove unused parts and reduce the footprint of the software.

One of the reasons that have caused Web Services technology to be popular in reaching enterprise adoption of the core standards is made possible by Web Service Engine offerings from both open source community and product vendors.

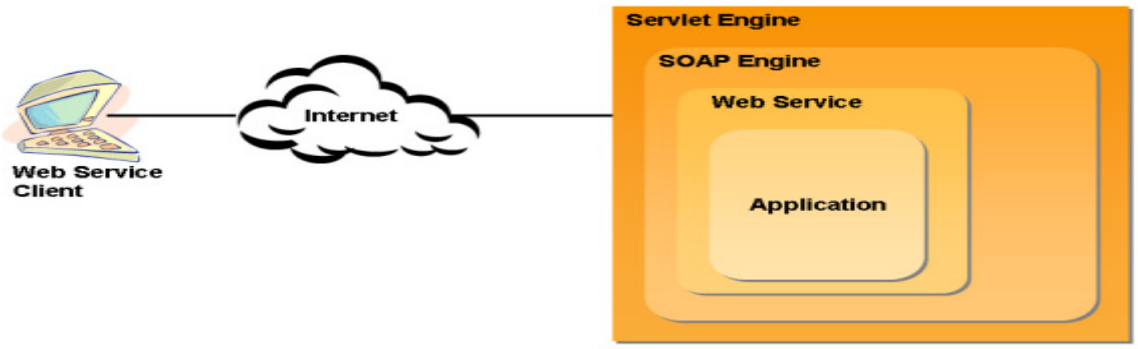


Figure 2.4: Using a SOAP engine to expose functions (Bayer, 2010)

Web Service engines and tools provide a way for data compression, data representation efficiency and efficient processing of XML. Web Services support can be added to a software product using SOAP engine. Figure 2.4 depicts how A SOAP engine is used in exposing functions.

SOAP messages are simple XML documents. SOAP messages are processed by A SOAP engine. But the SOAP is required to do more than that. For instance, the creation of WSDL documents, the dispatching of requests and the management of the lifecycle of service objects. The functions can even get more complicated especially when new standards and requirements need to be applied.

The use of an open source SOAP engine is always a good option for both users and software vendors. The benefits might lay in the large user communities of some open source projects who are a valuable source of support, documentation and bug-fixing.

2.6 Software testing

According to Srivastava, “Software testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results” (Srivastava, 2009).

Software testing is usually performed at different level in the phases of a development process. The object of the test can consist of either the entire program or part of the program. This could be a functionally or structurally related or a single module of the program. Before a test is conducted, there has to be laid out objectives as to reason for the testing.

Service Oriented Architecture (SOA) now form the fabric of IT infrastructure, and has then resulted in active and aggressive testing of Web Services to be seen as essential (Mallal, 2010). Web Services and their platforms are now liable to comprehensive Functional, Regression, stress, load, Performance, Interoperability and Vulnerability Testing that form the Pillars of SOA Testing. Comprehensive testing gives the assurance needed to ensure the robustness, scalability, interoperability and security of SOA and its underlying technologies. This research work will focus on Performance testing of SOAP Engines and Data Binding Framework which would be evaluated against various performance criteria of scalability, throughput and speed.

2.7 Chapter Summary

The chapter has presented a background on this research work as it concerns GUISET as a project, architecture and service and, therefore, indicating the gap that need to be filled which the choice of a web service development upon which GUISET as a service can be deployed. The concepts and technology such as SOA and their standards, web services and grid services

whereupon GUISET is based were discussed. It was also highlighted that there exists a whole lot of platforms both for free and commercial purposes, however to make a choice amongst them is a difficult issue. The chapter has shed light on what this research is to accomplish. Quite a number of features of a web service platform were presented. And lastly, software testing and modeling was emphasized. In the next chapter which is chapter three, state of the art related work of performance evaluation of platforms associated to web services was presented.

CHAPTER THREE

LITERATURE REVIEW

3.1 Introduction

Services are the key element of service oriented computing. Applications can be brought together out of numerous independent, loosely coupled and distributed services, in the service oriented computing model, (Bichler and Lin, 2006). The alternative to implementation of services from a technical point of view is the use of Web services. Web services are based on different XML-based languages for exchange of data and description of interfaces as used in SOAP and WSDL. SOAP provides the fundamental messaging infrastructure used by other components and services of Web service, supporting XML document exchange. The SOAP offers advantages which include simplicity, portability and extensibility; however, efficiency in some instances of performance is sacrificed. Subsequently, there is the need to test whether the web service middleware perform well in production environment. Another question that arises is; what are the performance bottlenecks applicable to Web service, which can then be used for performance evaluation of SOAP engines.

Previous research endeavors has been directed at comparing different web services engines in different instances and scenarios. One thing is that their performance differs and the outcome of their efforts does not necessarily meet the performance objective as limited by this research thus leaving a gap to be filled as it regards GUISET architecture and underlying services. This research work as was previously indicated in both chapter one and two is aimed at evaluating the performance of web service engines, with their data binding framework in a specified

scenario by performing what qualitative and quantitative evaluation. This chapter highlights the performance bottlenecks associated with web service and related work as it concerns this research work. First of all, the Performance constraints which were identified to be the binding framework used, the Parsing model and the software architecture were discussed. Secondly, the related work on performance evaluation of SOAP implementation and how it connects or disconnects from this research work is also discussed. The chapter then concludes with a summary of the reviewed literature as it relates to the goals and objectives of this research work.

3.2 Web Services Engines Performance Constrains

Performance has been identified (Machado *et al*, 2006) as one of the most imperative criterion in the selection of the most appropriate Web Services Toolkit. In addition, it is also said to be an aspect of Quality of service.

Web Service Platform limitations and delimitations are based more on XML messaging which happens specifically when XML data is sent over the wire, and the process of conversion of data takes place; it undergoes the process of Serialization and deserialization. The diagram in Figure 3.1 denotes the stages of sending and receiving SOAP messages in a web service environment which in turn involves serialization and deserialization.

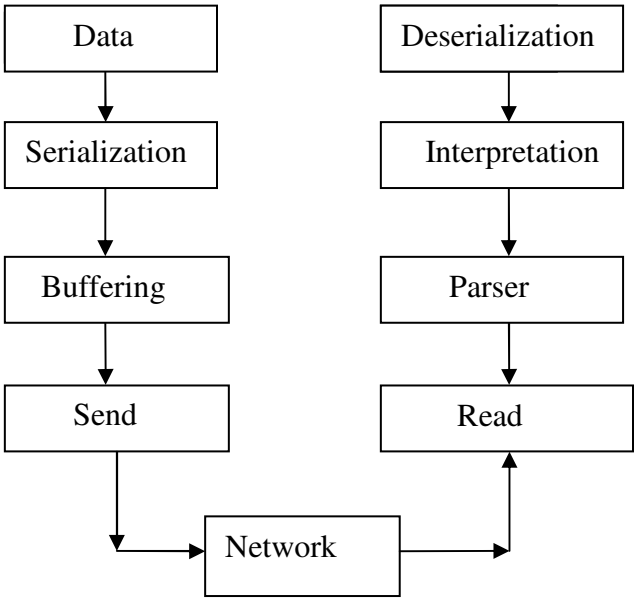


Figure 3.1: Stages to send and receive SOAP message (Chiu *et al*, 2002)

3.2.1 Sending SOAP Messages

According to Chiu *et al* (2002), the sending of a SOAP message is separated into several stages, as represented in Figure 3.1. Although these stages most times are not directly represented in the code, it still provides functional background for debate.

1. **Traverse data structures representing message**: The traversal of data structures representing a message begins when a SOAP message is imparted to correspond to the structure of the sSOAP XML. In serialization, this traversal is a minor process since the traversal of each leaf element can be done with simple operations like member offset calculations, array indexing, or pointer following.

2. **Conversion of machine representation of data**: The conversion of the strings or numbers is done in this stage. These strings and numbers form part of the actual data in their machine representation form. These strings and numbers may either require some processing to convert or may be simple and fast in performing conversion.

3**. Write data to buffer**: The form in which the data is stored can have an effect in the number of memory operations required. Therefore, at this stage, the storage of the data with its corresponding XML tags takes place.

4. **Initiate network transmission**: At this Stage, the contents of the memory buffer are transmitted through the operating system. This may require either one or more system calls. If the latter, buffering would use more memory especially when there is a large message. Alternatively, a vectored send can be used which would concatenate multiple memory buffers in one send call.

The network transmission is usually over TCP/IP. Since TCP/IP requires one packet exchange before transmission can begin, establishing a separate connection for each message adds a round-trip delay to each message.

3.2.2 Receiving SOAP Messages

The receiving of a SOAP message is assumed to be the opposite of sending a SOAP message, but the issues are rather dissimilar. Theoretically, the procedure involved in receiving a SOAP message is divided into four stages:

1. **Read from network into memory buffer**: From the operating system, a SOAP message is read into a memory buffer. A system is required; therefore, the number of system calls is reduced based on the amount of data read. The number of cache misses can be increased or decreased based on whether the amount of data is large or small to fit into the cache.

2. **Parse XML**: At this stage, the syntactic constructs of an XML is identified as parsing takes place. This parsing normally involves a state machine of some kind. There are some accepted models for XML processing, the Document Object Model (DOM), the Simple API for XML (SAX) and the Streaming API for XML (StAX).

3. **Handle elements**: The content of each tag gets interpreted as soon as the tags are identified. The tags would be presented by the parser in the form of text. Then, the delegation of the element interpreted is assigned to the application. This means that the tags must be examined after the parser has already made one pass through it.

4. **Conversion to machine representation**: Eventually, the conversion of text to the machine representation would take place.

3.2.3 Serialization

Serialization is the process of transforming an instance of a language class into an XML element. The inverse process, transforming an XML element into an instance of a language class, is called deserialization. Serialization and deserialization is arguably the most important component of any platform for Web Services. Furthermore, recent studies (Chiu *et al*, 2002, Devaram *et al*, 2003 and Govindaraju *et al*, 2004) have identified Serialization and deserialization as one of the factors that can affect Web services performance.

There exist three respective APIs in J2EE for parsing XML documents and they are SAX (Simple API for XML), DOM (Document Object Model) and StAX (the Streaming API for XML), each of which have relative strengths and weakness.

The DOM model is an API with a tree structured implemented by building an object representation of the XML document in memory. The SAX model has been identified to be

suited for applications that are interested only in a few specific elements of the document. Another technique that is optimized for cases where the XML elements need to be accessed in succession when the elements has been parsed before and need not be visited again is the StaX model.

One of these APIs used by binding frameworks is used in transforming an XML element into an instance of a language class and vice versa.

Therefore, all Web Services platform use either a parsing mechanism or binding framework for serialization and deserialization of XML data. Consequently based on studies, the performance of a Web Service Engine (Web Service platform) as a whole is affected by the efficiency of

- Binding Framework used
- Parsing Model (Chiu *et al*, 2002, Davis & Parashar, 2002, Elfwing *et al*, 2002, Head *et al*, 2005, Govindaraju *et al*, 2004 and Kolhoff and Steel, 2003)
- The Architecture (Suzumura et al, 2008)

3.3 Data Binding Frameworks

The technique that provides a simple and consistent way for applications to present and interact with data is regarded as data binding. According to (Balani and Hathi, 2009), data binding is the key for all web service development and it means mapping between Java objects and message formats which have been exposed by the service's implementation, for instance XML or JSON (Java Script Object Notation). SOAP-based web services would use XML as the data format, while RESTful services have a choice of using XML or JSON as the data format. In the context of this research, data binding is basically XML data binding. XML Data binding

provides a simple and direct way to use XML in Web service development platforms application. With XML data binding, the actual structure of XML documents can be ignored by the application; thereby working directly with the data content of those documents (Sosnoski, 2003). Alternatively XML binding refers to the mapping of XML documents to and/or from its machine representation thereby aiding a simple and spontaneous access to data in XML documents. This translates to easy access of data in XML documents.

3.4 Parsing Model

From the previous section, it was acknowledged that the Parsing model has a great impact in the performance of a web service platform. In that vein, descriptions of the three well known approaches for processing XML are given below:

3.4.1 Document Object Model

Document Object Model (DOM) is a high-level parsing API whose key advantage is ease of use. DOM presents applications by way of an in-memory tree-like structure. This tree structure allows random-access. This means that DOM parses the entire document and creates them as Objects before any part of the document can undergo any other process or action. As soon as the document is in memory, navigation is done freely and parsing randomly, thus providing maximum flexibility. However, the cost of this flexibility and simplicity results in performance penalties which could be a large memory footprint and significant processor requirements which increases with an increase in document size. The entire representation of the document has to be held in memory as objects for the time the parsing process is active. This might not be applicable in the case of working with small documents, because memory and processor requirements can escalate quickly with document size.

3.4.2 SAX (Simple API for XML)

SAX also known as Streaming Push Parser is a low-level event based API. The main benefit of SAX lies in its efficiency. With SAX, XML document is parsed then events get generated and are then passed to the application using callbacks to handlers that implement the SAX handler APIs (Java Web Services Performance Team, 2005). SAX requires the entire document to be parsed. SAX, therefore, can be referred to as a programing model involving an XML parser that sends XML data to the client as the parser encounters elements in an XML. Irrespective of whether or not the client is ready to use an XML data at a particular point in time, the parser sends the data at that time.

3.4.3 StAX (Streaming API for XML)

StAX is an API for reading and writing XML data which is often referred to as pull parsing. Streaming pull parsing is one way to processing XML that has proved to be of memory-efficient, simple, and convenient while taking charge of both the parsing and writing process.

Most parsers fall into two main groups: the tree based or event based. StAX is more or less closely aligned with the latter; which is event based. The main idea of StAX is to bridge the gap between both the tree based and event based. This parsing model combines the efficiency of SAX and the ease of use of DOM even with higher level XML constructs.

StAX allows skipping the document ahead to areas of interest, and getting only the part of the document and arbitrarily stopping or suspending processing at any time. Contrary to SAX, the lower-level events are pushed by the parser to the application. With StAX, multiple XML

sources are allowed to be processed concomitantly. For instance, at the time a document is either importing or including another document, the imported document can be processed at the same time the original document is also been processed by the application. This example is applicable when documents like XML Schemas or WSDL documents are read by the application.

3.5 The Software Architecture

In recent times, the role of architecture has been widely recognized to determine the quality of a software system as a whole(Clement & Northrup, 1996, Garlan & Shaw, 1993, and Perry and Wwolf, 1992). The argument has been on the effect of decision making on the quality of software at each development phase. Therefore, architectural choices made at each phase of development process have the most effect on virtually all quality attributes such as modifiability, reusability, reliability as well as performance (Williams & Smith, 1998).

3.6 Related Work

This section of related work provides descriptive analysis of some significant researches in the field of performance evaluation of Web services products and platforms. The key area of interest includes, but not limited to open source tools (SOAP Engines and Data Binding Frameworks). Performance evaluation of Web services and their development platform is entirely a new area with a little previous work. These evaluations are considered specifically with regard to their relevance to the aims and objectives of this dissertation.

3.6.1 Evaluation of Web Application Technology

An evaluation of Free and open source e-commerce web application technology with regard to SMMEs was carried out to help SMME looking for a means on Internet-enabling business (Msimanga et al, 2004). The evaluation was presented by building and deploying a sample e-commerce application on four J2EE web application containers and four freely available database servers which were selected and evaluated based on functionality to perform required task, ease of use like complexity of installation of the software, user friendliness of interface, amount of documentation and support available. The evaluation was used to determine the performance of each server under different loads. In addition to that, the software used during each stage of the development and deployment process was evaluated. Their experiment focused on open source tools for e-commerce application development and not on testing the performance of Web Service tools. Even though the work by Masinga *et al* (2004), are not based on performance of web service tools or web service engines specifically but the similarities lies in two attributes which are:

I. The approach of using some quality attribute of the technology evaluated as a part of the evaluation criteria which formed the basis of this research work's Criteria for inclusion of any tool for investigation,

II. The use of free and open source tools in building web application that meets the need of SMMEs.

3.6.2 Comparison of XML Binding Frameworks in the Context of Service-Oriented Architecture

The comparison is aimed at objectively performing an evaluation of some popular XML Binding frameworks thereby ascertaining the stance of XML binding framework in the context of Service Oriented Architecture (SOA) platforms (Padmanabhuni *et al*, 2005). In their analysis, Padmanabhuni et-al stated the role of XML binding in SOA and proceeded to state that most XML Binding Frameworks are focused on Code generation from XML Schema. Therefore, they started with a schema grammar for documents to be processed and then used the binding framework to generate the target source code. They performed a quantitative evaluation on selection XML Binding Frameworks based on Marshalling Time, Demarshalling Time and Memory Load as parameters. The test was run with three samples of XML differentiated by its segmentation factor. The Binding frameworks execution was based on the sample XML placed serially in some number of times. The analysis was performed to give practitioners the privilege of making informed choices on the appropriate XML binding platform that suits their needs. The work concluded that a suitable XML binding framework can be chosen depending on the specific requirement of an application.

3.6.3 Comparison of Web Services Engines

3.6.3.1 Generation-Based Comparison

Performance analysis of three different Web Services Engines/platforms combined with different data binding frameworks were evaluated by Kunti et al (2007). The Web Service Platforms were based on first, second and third generation Web Service Engines. The performance analysis studies was limited to binding performance, round trip performance,

scalability speed and throughput which are directly affected by factors such as the binding framework used and the parsing model of the SOAP engine. In testing the performance of various Binding Frameworks and SOAP engine, a simple standardized test application was designed and the test application consists of web service that took an array of objects as a parameter and returned same to client with a defined XML schema. To quantitatively measure the performance of both the Binding Framework and the SOAP engine, the system was recorded at the critical points along the request-response flow with millisecond accuracy and these statistics were calculated based on these parameters: Total Round Trip, Total Marshalling Time and Total Unmarshalling Time.

The work of Kunti et al had some significant influence on this work in the form of the parameters used for performance evaluation, but one important pitfall is the evaluation of generational platforms of which the result shows that the third-generation SOAP engine is better than the first and the second generations. However, evaluation based on generational platforms would not give a fair result since later generations normally are meant to be better than and to improve on the previous ones.

3.6.3.2 Programing Languages-Based Comparison

In evaluation performed at Tokyo Research Laboratory, IBM Research Center, the performance comparison of Web Service Engines in three different programing Languages; which are PHP, Java and C was presented (Suzumura *et al*, 2008). The Comparison was performed in two approaches, Qualitative and Quantitative which is similar to the work in this dissertation. In

qualitative Comparison, a comparative study of web service support was provided by using PHP, Axis 2 C and Axis2 Java from variety of perspectives as the software architecture, The XML processing model, Web Services Standard Support, the programing model and deployment model for server providers, has support for Restful service, and other functionalities. In quantitative comparison:

I. they performed four different variation of experimental test which are the Comparison of Web Servers without the SOAP Engines to obtain the initial overhead for each configuration both in static and dynamic state without a SOAP engine

II. Comparison with a Web Service example where the web service returns the response to a query in the request SOAP Engine. The size of the request message is calculated and the response is also calculated based on how many bytes long.

III. Comparison with a Benchmark Suite. A benchmark Suite for SOAP processing was used to echo SOAP messages containing an array of structured elements. In doing this evaluation, they came up with the profiling for each SOAP engine which enabled them to comment on throughput, and the Memory usage of each SOAP engine which was termed the memory footprint.

The reported work basically created awareness that PHP is a more viable option for publishing SOAP/WS based web services unlike in this work which is concentrating on web service frameworks built with java programing language.

3.6.2.3 Scientific Computing-Based Comparison

This study reported by Govindaraju et al (2004), compared the performance of some widely used SOAP toolkits for various workloads commonly used in scientific computing like varying array sizes of doubles, integers and strings. According to the study, it was stated that performance study for scientific data structures would aid in the design and development of new toolkits and also guide users in choosing appropriate toolkits for their current application requirement (Govindaraju et al,2004). The approach taken by this performance evaluation was comparing and contrasting the performance of SOAP toolkits based on their performance characteristics and other design features that can be used in optimizing SOAP implementations. The identification of SOAP features that has effect on the performance was made imperative. Their objective of identifying the features of SOAP that affects Web Service performance leads to literature review on features like Role of HTTP, Parsing XML, XML generation, etc. The insights on various Soap features that affect performance is useful although most of the bottlenecks mentioned like parsing XML by the use of either SAX, DOM or Stax is being taken care of with regards to this research work since all the tools used for the evaluation are Stax based parsers.

3.6.2.4 Comparison Based on Stimulated and Actual hosted Environments

The performance testing approach based on testing both the simulated environment and the actual hosted environment was done by comparing performance metrics like response time and throughput for web services. It was argued that the testing approach will help developers in early life cycle of web services and also in tuning the applications before putting it out for

proper use (Tripathi *et al*, 2010). The performance measurements suggested that from modeling perspective web service can be stimulated first and tested for various performance metrics, which would also give results close to the original one since simulation and actual hosted environment show similar results. Due to this fact, one can actually go on with testing simulated web services environment with the assurance that the result would be similar when the real web service is published using the tested tools.

3.6.3.5 Comparison Based on Network Latency

The experimental evaluation performed in this related work is the latency performance of SOAP implementations that operates over HTTP. The experiment set up the client and the server to run on separate hosts to enable reports on network delay. Consequently, when the client and server are running on separate hosts, SOAP automatically performs poorly (Davis *et al*, 2002). In this work the performance interest is in the processing of XML documents and, therefore, network latency is eliminated by allowing the server and the client to run on same system.

3.6.3.6 Comparison Based on Commercial SOAP Implementation with non-SOAP Technologies

The experiment looks at the performance of contemporary commercial Web Service implementations compared to non-Soap technologies like binary/TCP (Ng *et al*, 2004). The results show that SOAP implementations have the ability to deliver good performance when handling short messages.

3.7 Summary of Related Work

Table 3.1: Summary of Related Work

Authour and Year	Performance testing	Qualitative Approach	Quantitative Approach	Use of Bench mark Suite	Performance parameter	Marshalling	Unmarshalling	Round Trip Time	Transaction per Second	Other relationship	other dissimilarities
Msimanga et al 2004	Yes	Yes	Yes	Yes	Yes	N/A	N/A	N/A	N/A	1.)Based on SMME 2.)Quality attributes formed part of selection criteria	Based on Web Application Technology
Padmanabhuni et al,year	Yes	N/A	Yes	Yes	Yes	Yes	Yes	N/A	N/A	XML Binding Framework	Use of Memory Load as a parameter
Kunti et al, 2007	Yes	N/A	Yes	Yes	N/A	Yes	Yes	Yes	Yes	Web Service Engines	Based on generational platforms
Suzumura et al, 2008	Yes	Yes	Yes	Yes	N/A	N/A	N/A	N/A	Yes	1.)Web Service Engines 2.) Qulitative approach	Based on Web services engines in different languages
Govindaraju et al,2004	Yes	N/A	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Web Service Engines	Based on scientific data structures
Tripathi et al, 2010	Yes	Yes	Yes	Yes	Yes	N/A	N/A	N/A	Yes	Soap tools	Based on Simulated and hosted environments

3.8 Chapter Summary

In this chapter, theories and research that influences the goal and objectives of this dissertation have been discussed. The main objective of this research as stated in chapter one was to evaluate the performance of a few leading open source web service platforms with a view to recommend one as the best for developing GUISET services. Therefore, the discussion was mainly on performance of web service platform and their associated bottleneck. From the discussion, it was pointed out that the constrains that inhibit good performance of web services platform is its serialization and deserialization which has a lot to do with the data binding framework, Parsing model and the architecture of the system. The chapter has also presented a state-of-the-art relevant related work with a summary of their relationship with this research work. Subsequently, chapter four as the chapter of comparative study framework will give an outline on the web services platforms under investigation and also describe the test environment.

CHAPTER FOUR

QUALITATIVE COMPARATIVE STUDY

4.1 Introduction

In order to achieve the goal of this research, which was to evaluate a few leading open source platforms for web service development, there was the need to choose from the numerous existing web services platforms available for evaluation. Consequently, with the aid of the Literature Review presented in Chapter Three, decisive factors were formulated to enable the selection of evaluation platforms. Subsequently, three platforms were chosen. In addition to this, there was also a need to perform qualitative comparisons of the three chosen web services platforms.

The remainder of this chapter is arranged as follows: Section 4.2 gives a description of the selection criteria with highlights on each of the selected platforms. The selected Web Services Platform is discussed in Section 4.3. Section 4.4 discusses the Qualitative aspect of the comparative evaluation of Web Services Engine. In the Qualitative evaluation, the Software Architecture, XML Parser and Processing Model, Programing Model and Deployment Model and other functionalities were compared. Section 4.5 is then used to compile the Qualitative Comparative Framework and Section 4.6 gives the Chapter Summary.

4.2 The criteria for inclusion of Web Service Engine for Investigation

The efficiency of a Web Service Engine based on SOAP that can be used in deploying e-commerce on demand service for SMMEs can be increased by identifying some criteria that are relevant to this study. However, choosing a framework for the comparative investigation

performed by this work is dependent upon several factors. These factors were set up based on some models on the concept and subject of software evaluations.

Therefore, the criterion for inclusion of any particular platform for comparison is as a result of the following factors: Community, Stability and Development Activity, Documentation Support, Licensing and Free and Open Source.

4.2.1 Community

The amount of Community involvement can be said to be a good indicator of interest in open source software as was indicated by Golden (2005). Thus, the community is one of the most important aspect of an open source software project. The user community of an open source project is made up of those that either use or participate in the software. Participation comes in different forms like filing bug reports, fixing bugs and extending the software. The community of any open source software comprises both the user community and development community. When community participation is large and active, then the software is accepted. According to Duijnhouwer and Widdows (2005), people get attracted to a software project if the software is good enough to use. Therefore, all successful open source projects have a common phenomenon, which is a successful, wide-ranging community that supply all that is required to sustain and maintain the wide adoption of the software.

In this research work, the community activity was identified based on the on the visibility of the community and the activity level. Community visibility, in this case, is observed through the project's website(s) and its friendliness which is based upon the presence of the user community and mailing list on the website. The activity level is identified by the amount of

posts with regards to variance on dates, number of posts, number of users and response and also the friendliness of users towards one another.

Axis2 website contains a tab named “getting involved” which one can select to join or subscribe to the mailing list. There is also the presence of an “Issue Tracking” link which directs to a page where issues like bugs, feature requests and other issues on the project can be submitted for discussion.

From the research conducted**,** the **Glue** Web Service Engine does not seem to have a website dedicated to the project.

The **CXF** website contains a mailing list, issue reporting and FAQs (Frequently Asked Questions) links. The activity level based on the post and responses are high from observation. The friendliness of the interface on the CXF website is good as well.

The **Metro** website offers a friendly community and the site and the site also offers of the “mailing list”, user forum the issue tracker links.

SpheonJSOAP offers a website with mailing lists, forums, a tracker and support links. The community activity on the forum was last updated in 2008 and which points to poor community involvement with the project. The mailing list when navigated, does not contain any other information on how to either subscribe to or join the mailing list.

XINS offers a mailing list where contributions and comments can be made. A forum can be accessed through the Sourceforge website. This forum is further divided into Open Discussion, Help and Announcement. Sections on the Open Discussion contain no post. The latest activity in the Help Section was July 2009 and the announcement in 2011.

4.2.2 Documentation Support

Documentation in this context refers, to the availability of information resources like user guide, books, release notes, installation guides, tutorials and active community and support groups. An information system that is not available when needed is almost as bad as none at all. The documentations could be scattered over a number of locations or the core documentation might often be short with little or no information. In addition, Information that is valuable and useful could be hidden away in the source code and only available to experienced programmers.

The open source community comes in handy to offer help and support. The archived mails from the mailing list offer support through the information available and responses to questions.

In detailing the concept of the availability of technical information and the extent to which a non-technical person can understand the said available documentation, as well as the level of support present for each of the SOAP engines under review, are presented as follows:

Documentation in **Axis2** is well detailed and it comes in the form of both User Guide and Tutorials. The support groups and communities provide helpful answers to questions and problems.

The documentation provided by **CXF** is good and explanatory information can be understood even by first-time developers.

Glue has limited or no documentation, because it is also available in a commercial option which would in most cases require purchase before any kind of documentation can be seen by the user. In addition, the free version of Glue is unsupported.

Metro is documented in detail. There are many tutorials and samples available. The support team is a strong one.

SpheonJSOAP has inadequate documentation. A User Guide document is the one available and it contains little information on the usage of the tool.

XINS has a limited documentation in the form of a User Guide of 10 pages.

4.2.3 Stability and Development Activity

There are many evolving open source software, most of which started out as an experiment by either a single programmer or a group of programmers. The software may in the long run be abandoned or discontinued due to less or no development activity, community activity and or usage. Software projects are never referred to as being static since there is always a need to improve or add to the existing features (Wheeler, 2005). A software product that is continually developed and that has stayed around for a longer period of time indicates good stability and has improved chances of survival. Even though, age cannot be said to be a sure bet of a

product's survival. A software project always stabilizes with time through continued and constant development activity.

Continued Activity has its interest in the maintenance and enhancement of the tool in the sense that the tools should be able to be migrated, up dated and re-written to conform to new standards or specifications that are ever-evolving.

Axis2 started as Apache SOAP and moved on to Axis 1.x which was designed as a follow-on. It was then developed from the well-known Axis1.x series and is a complete rewrite of Axis1.x from the ground up. Axis2 uses a modular architecture that allows easily extensibility.

CXF was extensively retooled from the X-fire and Celtix projects. Before the fusion, Xfire has been integrated with other open source projects and even closed source projects. The CXF project now belongs to the Apache Foundation.

Glue was created by The Mind Electric. GLUE as it was then named was the company's flagship product. It was released in two flavors: the free unsupported "Standard" version and the company supported but commercial licensed "Professional" edition.

The product was later acquired by webMethods as part of the merger with The Mind Electric. Glue was rebranded with the webMethods prefix to become webMethods Glue as a result of the merger.

The web service stack named **Metro** was as a result of the bundling of two components/ projects which are the JAX-WS RI, the Reference implementation of the JAX-WS specification

and WSIT, a Java implementation of some of the WS-*. Metro interoperability with the .NET framework has been enhanced.

SpheonJSOAP was released in December of 2001 and since then has remained unchanged with a promise of updates on the tool.

XINS development has been up to date trying to meet up with all the other available SAOP engines. XINS 3.0 alpha 2 edition was released in March 26, 2011.

4.2.4 Licensing Requirement

The license under which a particular software is offered is one of its distinctive features (Chavan, 2005). It is, therefore, of essence that the license of the web services platforms under investigation are considered due to the fact that the code might be modified and be used for distribution. The Open Source Initiative (OSI) and The Free Software Foundation (FSF) made available a record of approved licenses. With this record of licenses, the considered project can be evaluated against the requirement needed from the platform. One license that is mostly used is the GNU General Public License (GNU GPL) (Weber, 2004).

Since, according to Wheeler (2005), a software project that uses a well-known public license has the potential of having better resources. Thus, the software project is scrutinized for OSI approved.

Axis uses the Apache license 2.0(Apache-2.0) and this license falls under the most popular and widely used of licenses with strong communities.

CXF is also licensed under Apache license 2.0(Apache-2.0).

Glue is licensed as professional.

Metro is licensed under the Common Development and Distribution License 1.0 (CDDL-1.0), and GNU General Public License version 2.0 (GPL-2.0).

Spheon JSOAP is licensed under the BSD 2-Clause "Simplified" or "FreeBSD" License (BSD-2-Clause).

4.2.5 Impact of FOSS on the Investigation

Free and Open Source Software (FOSS) presents a feasible choice for developing Web services although they are not as easy to use as commercial products. However, with free and open source tools, it is presumed that the tools are readily available with a fully functional enterprise solution that does not cost money. Furthermore, with open source, there is no worry of the effect of what happens to, or the decisions of software vendors. In this vein, Axis2, CXF, Metro, XINS and SPHEON JSOAP are all free and open source except for Glue which is a commercial product.

4.2.5 Checklist and Summary of Selection Criteria

There are various contending web service platforms such as Axis, CXF, Glue, Metro, XINS and JSOAP and others not included in this write up due to non relevance to the study, but based on

the criteria's on each of the SOAP engines, these three were finally chosen for evaluation; Axi2, CXF and Metro. The summary is as indicated below:

a) **Community**: Axis2, CXF and Metro communities are presumed to be active based on the observations on their respective websites and community forums. The topics number several thousands and the replies on most topics range from 4 to 46 replies. Meanwhile XINS and Spleon JSOAP present little community activity on their respective forums.

b) **Documentation Support**: Documentation for Axis2, CXF and Metro are very detailed. Their respective documentations are rationally complete and they also have a number of manuals and tutorials from both users and developers. Sphleon JSOAP has only one Document in the form of a user manual which is not detailed. No documentation was found for Glue except for articles which are meant for the commercial edition with no support for the open source version.

c) Stability and Development Activity

d) **Software License:** Axis, CXF, Metro and Spleon JSOAP all have licenses that are approved from the OSI except for XINS which is licensed as a professional product.

e) **Free and open Source/ Current version:** Axis, CXF, Metro and Spleon JSOAP are all free and open source in their current versions as of the time the research was conducted. Glue is presented as both open source and commercial, but with nothing pointing to support for the free and open source version.

Table 4.1 depicts a breakdown of the selection.

Table 4.1: Tabular Presentation of How the Tools Compare

Criteria	**Axis2**	**CXF**	**Glue**	**Metro**	**Spheon JSOAp**	**XINS**
Community	√	√	X	√	X	X
Document Support	√	√	X	√	X	X
Stability and Development Acitivity	√	√	√	√	X	X
Licensing requirement	√	√	X	√	√	√
Impact of FOSS on the Investigation	**√**	**√**	**X**	**√**	**√**	**√**

From Table 4.1, the selection of Axis2, CXF and metro for evaluation is depicted by the fact that they have all or most of the required features.

4.3 Overview of Selected Web Services Stack under Investigation

4.3.1 Axi2

Axis2 (Deepal, 2008) is an open source project that is a core engine for Web services whose main design criteria is flexibility and extensibility. It is a total re-design and rewrite of the well known Apache Axis SOAP stack. The rebuilding of Apache Axis2 is claimed to enhance performance awareness and flexibility to support an assortment of Web Services standards as well as asynchronous web services. It also has support for REST. The server platform of Axis2 provides service monitoring functions and allows easy management of services and

deployment. Axis2 implementations are available both in Java and C. Axis2 has the potential of adding Web services interfaces to Web applications, and can also function as a stand-alone server application.

4.3.2 CXF

CXF (Naveen *et al*, 2009) is an open source services framework which is used in building and developing services with frontend programing APIs, like JAX-WS and JAX-RS and also provides an easy to use, standard-based programing model for developing web services. Apache CXF is the product of a combination of two projects, Celtix and XFire, hence the name **CXF.** Celtix was an open source Java-based Enterprise Service Bus (ESB) project and a product of ObjectWeb consortia that delivers open source middleware solutions. The project was sponsored by IONA. On the other hand, XFire, a Java-based SOAP framework, is an open source project from Codehaus. Both Celtix and XFire, while in their initial versions, had many things in common and therefore the developers of both projects decided to bring out the best of both worlds and, planned a better 2.0 version of Celtix and XFire. The communities of both these projects entered incubation at the Apache Software foundation to develop version 2.0. It took about 20 months at the Apache incubator before CXF finally rolled out. CXF is now formally known as Apache CXF which concentrates on delivering an open source web service framework. The framework which had its first release as v 2.0 is now evolved as v2.2 at the time of this write-ups, with bug fixes, and the addition of new features.

Apache CXF builds on a highly-configurable architecture to support a range of transports, data bindings, and extension technologies.

4.3.3 Metro

Sun/Oracle Metro (Sosnoski, 2009) is the open source stack based on the JAXB 2.x reference implementations and JAX-WS 2.x Java standards. Therefore, Metro is referred to as JAXWS Reference Implementation. The current code base supports JAX-WS 2.x Web Services Standards and JAX 2.x data binding but it also uses additional components to provide features beyond the basic support as defined by JAX-WS. Unlike the other toolkits, the only XML binding mechanism supported by Metro is JAXB, although others could be added.

Metro is said not to provide the kind of flexibility exhibited by some other web services platforms like Axis2 and CXF it has remained a platform of choice due to its close ties with official standards developers and its .NET interoperability.

The major issue with Metro is that it is not licensed under the Apache License, v2. Instead, it is distributed via a dual CDDL and GPL v2 license. The CDDL license (which is the only remotely compatible license to the Apache License, v2) is more stringent in the requirements it places on derivative or utilizing software. However, the requirements are not overwhelming and only relate to changes made to the original CDDL licensed codebase. The Metro toolkit provides convenient tooling in the form of Netbeans plugins. This allows for configuration of most security properties using a GUI and hence proves user friendly.

4.4 Comparative Qualitative Evaluation

To accomplish the purpose of this research, it necessary to understand some typical characteristics of the frameworks and evaluate them in the context of the features that in one way or the other affects the efficiency of the SOAP Engine. The Comparative Evaluation as has

been reported in the preceding chapters is conducted in two ways: the qualitative and quantitative. Comparative Evaluation is referred to as research in which an assessment and the findings of the evaluation process are set and obtained through a comparative framework (Vartiainen, 2002).

In view of performing qualitative evaluation of the selected platforms, the word quality is extensively looked at to properly denote its meaning to the concept of this work. Therefore as was defined by Crosby (1996), it is a conformance to requirement and its measurement is defined in terms of measurement done against a set of defined requirements and determined by the level of conformance. In order to effectively report on this aspect of evaluation, the requirements of the Web service platform that this work is considering are: the software architecture, XML Parser and Processing Model, Programing Model and Deployment Model, Asynchrony Support, the standard support and REST Web services Support.

4.4.1. Software Architecture

Any given architecture is as a result of what that architecture ought to attain. This means that for the architecture to be successful, its evaluation should be based on the expected requirements to be met by the architecture. In the case of SOAP engines, its architecture is meant to deliver SOAP messages which form the basic unit of web service interaction. The delivery of a SOAP Message is defined by some components known as SOAP Nodes. The SOAP Nodes are the SOAP sender and the SOAP receiver. A SOAP Sender usually sends a SOAP message and SOAP Receiver receives it. This indicates that all Web service interaction is done via SOAP.

In web services interaction, the middleware component takes responsibility for the intricacies in SOAP messaging and the users stick with what they are used to for instance the programing language they are accustomed to.

The following are the identified requirements that SOAP engine architecture ought to have:

1. Provision of a framework for processing SOAP messages. This framework should be built to be extensible to enable the users to extend the SOAP processing per service or per operation basis.
2. The ability of deploying Web service with or without WSDL.
3. Provision for a Client API for invoking Web services. This API should be able to support both the Synchronous and Asynchronous programing models.
4. The ability of configuring a SOAP engine and its components through deployment.
5. The ability of sending and receiving SOAP messages with different transports.

A) Axis 2

The Axis2 architecture principle is to maintain uniformity. The principles are as listed below:

- The logic in Axis2 architecture is separated from the states. The code that does the processing is executed freely by parallel threads, since the code does not have a state inside Axis2.
- The information model keeps all information, which then allows the system to be suspended and resumed.

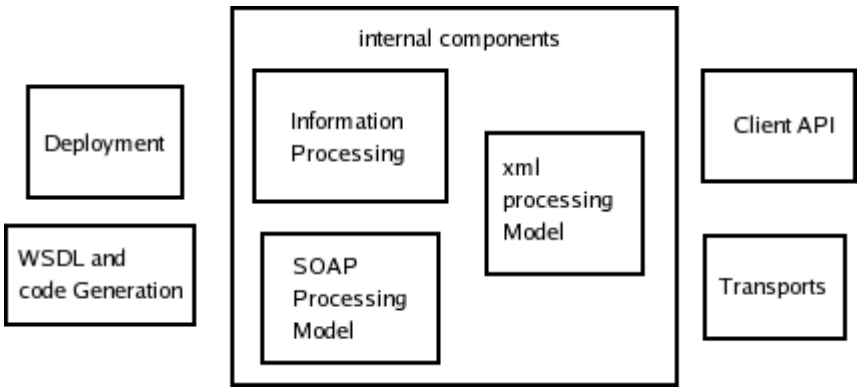


Figure 4.1: Axis Architecture (Apache Axis2 Architecture Guide, 2011)

The architecture of Axis2 is modular in nature and it is as illustrated in Figure 4.1. Axis2 Framework is built up of core modules which are the Information Model, the XML Processing Model, the SOAP Processing Model, the Client model, the Client API and Transports. These modules put together form the core architecture of Axis2. There are also some other modules which are known as Non-core/other modules (Code Generation, Data Binding etc) and they are layered on top of the core modules.

Axis2 allows invocation of Web services using Java representations, while the SOAP messaging for is handled behind the curtain. Before any massage is handed over to the core engine, the massage must be transformed into a SOAP message. Since an incoming message can either be in the form of a SOAP message or a non-SOAP message (REST JSON or JMX). Afterwards the conversion to a SOAP message will take place at the transport level.

Irrespective of these functionalities, Axis2 architecture is claimed to be built with performance especially on memory and speed as a key consideration. Axis2 Core Architecture is built on

WSDL, SOAP and WS-Addressing specifications. Some other specifications like JAX-RPC, SAAJ and WS-Policy are layered on top of the Core Architecture.

B) CXF

The overall CXF architecture is represented in Figure 4.3 and is primarily made up of the following components:

1. Bus: The bus is considered the major component of the CXF architecture. The endpoints are defined in a common context by the bus. This means that it wires all the runtime infrastructure components and provides a common application context.
2. Messaging & Interceptors: Most functionality is built on this layer of the architecture in view of the fact that this layer provides the low level message and pipeline layer.
3. Front ends: The programing model to create service is provided by frontends. This Frontend modeling as a CXF concept allows web services to be created using different frontend APIs. Web services can be created by using simple factory beans and JAX-WS implementation. In addition, the model allows creation of dynamic web service clients. The primary frontend supported by CXF is JAX-WS.
4. Services: Services are hosted WSDL-like model known as service model. This service model does the description the service.
5. Bindings: Bindings in CXF provides the functionality for interpretation of the protocols for example SOAP, REST and CORBA.
6. Transports: Destinations and Conduits make up the transport abstraction that CXF uses to achieve transport

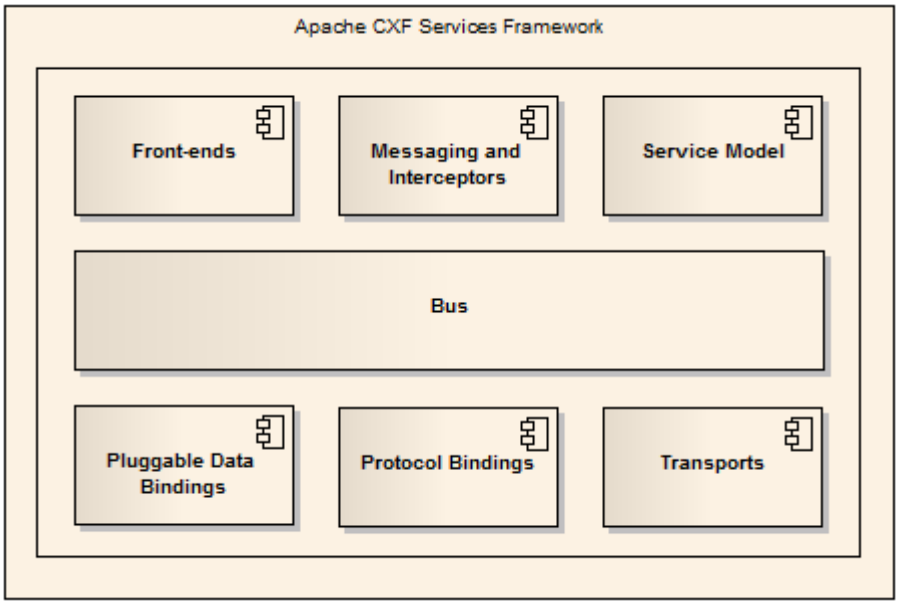


Figure 4.2: CXF Architecture (Apache CXF Software Architecture Guide)

C) Metro

Metro has a layered and pluggable architecture consisting of the application code layer, the annotated classes layer which can also be termed the Strongly typed layer and the Messaging Layer. This architecture claims to be portable. The upper layer uses annotations extensively which enables ease of use, good integration with tools and fewer generated classes. The lower layer is API-based thereby more traditional. This lower layer is used for advanced scenarios and claims to be extensible. The extensibility enables the support for "pay as you go" model.

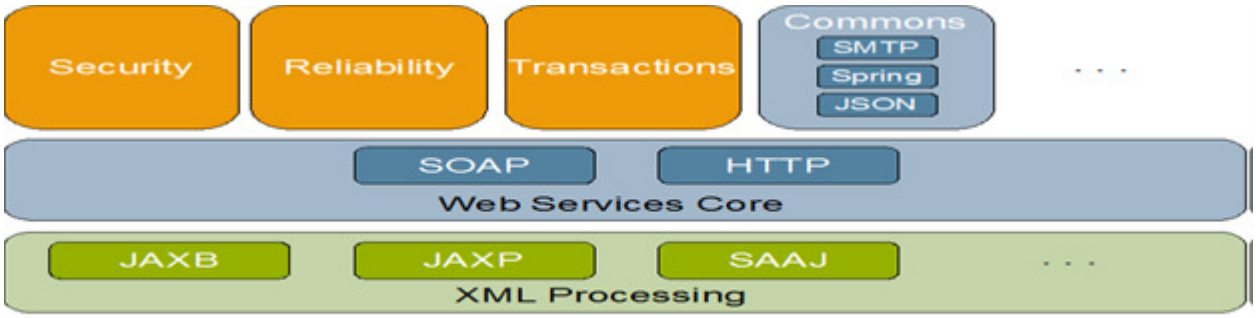


Figure 4.3: Metro Architecture (Arun Gupta, 2007)

The Metro architecture is represented in Figure 4.3. The core of Metro implements the JAX-WS API and serves as the foundation where all the higher-level features plugs in. In Metro, the request and response processing is done by the use handlers. It does support a variety of WS technologies coupled with the handlers and these technologies are integrated into the Metro Engine itself rather than a separate component.

4.4.2 XML Parser and Processing Model

Handling of the SOAP Message is the one of the chief and intricate task of Web Service Engine. The efficiency of handling a SOAP message is the single most important factor that decides the performance.

A) Axis2

The XML parser as a core module of AXIS2 is called AXIOM or AXis Object Model. AXIOM was developed as part of Apache Axis2. However, it is a pure standalone XML Info-set model with its own features which can be used on its own without Axis2. AXIOM provides a simple API for SOAP and XML info-set. This has the functionality of hiding the complexities of efficient XML

processing within its implementation. AXIOM (AXIs Object Model) forms the basis of the XML representation for every SOAP-based message in Axis2.

AXIOM is StAX-based and is object model complaint. In addition, it also has supports for on-demand building of the object tree. The on-demand object tree building entails support for "pull through" model where tree building can be turned off and then access to the underlying pull event stream will then be direct. There is also support built in for XML Optimized Packaging (XOP) and MTOM. The combination of these supports brings on the carrying of binary data by XML in an efficient and transparent way. Therefore, it is presumed to be an easy to use API with high performant architecture.

B) CXF

CXF use Interceptors as the fundamental processing unit. InterceptorChain is created and invoked as soon as a service is invoked. Each interceptor gets a chance to either read or transform a message, process headers and validate the message. The CXF clients and servers uses interceptors. For instance, at the time, a CXF server is been invoked by the CXF client, the client will have the outgoing interceptor chain and the server will have the incoming interceptor chain. When the client receives the response sent back from the server, the server at this point will have the outgoing message chain and the client will have the incoming message chain. In addition, when error occurs, a separate outbound error handling chain will be created by the server and an inbound error handling chain will be created by the client.

C) Metro

In Metro, handlers are used as part of the request and response processing. The library uses Streaming API for XML (StAX) for XML processing, thus providing a good performance in terms of security header processing than when DOM is used.

4.4.3 Programing Model and Deployment Model

The aspect of Web Service Engines compared in this section is in respect to programing style and ease of deployment for implementing and publishing web services which invariably means support for hot deployment. When services are deployed while, the system is up and running, the process is referred to as a hot deployment. In a real world environment, the availability of the system is important irrespective of whether or not services are being created. Business viability could be affected when system processing is slow, even for a moment. In which case, new service may be required to be added to the system. If this addition can achieved without shutting down the servers, it is considered a great achievement.

A) Axis

In the Axis2 deployment model, services are deployed, transports are configured, and SOAP processing model per system, service, or operation basis extended by the user. The Axis2 deployment mechanism enables the bundling together of all files (class, library, resources, and configuration) an archive file, and dropping it in a specified location in the file system by the developer.

Hot deployment and hot update is one of the new features of Axis2 even though it is not a new technical concept for Web Services platforms. Therefore, Axis2 now come complete with the features of hot deployment.

The issue of hot deployment is addressed in Axis2 by providing a Web Service hot deployment ability, in which the system need not be shut down when deploying a new web service. Therefore, it is achieved by dropping the needed web service archive into the directory for services in the repository. This would then lead to the automatic deployment and availability of the service by the deployment model.

B) CXF

CXF is said to have support for a variety of Frontend programing models. Due to this support, CXF implements the JAX-WS APIs. This JAX-WS support involves extensions to the standard enhancing ease of, automatically generating request and response bean classes code, and no requirement for WSDL for simple cases.

There are "simple frontend" available in CXF that allows the creation of clients and endpoints without annotations. CXF have support for code first development starting from Java and contract first development with WSDL.

CXF offers a flexible deployment model where services can be developed and unit tested in a standalone environment, and promoted for deployment in an application server environment. Web services developed with CXF can be deployed with light weight containers like Tomcat and

also J2EE-based containers such as Websphere, Weblogic, JBoss, Geronimo, and JOnAS. It can also be deployed in the two tier client/server environment. CXF provides integration with a Service Component Architecture (SCA) container like Tuscany. It also supports Java Business Integration (JBI) integration with a web service deployed as a service engine in JBI containers.

C) Metro

Web Services deployment is simple. Metro has support for two types of deployment models for publishing web services. They are JSR-109 deployment model and JAX-WS RI specific deployment model. In JSR-109 deployment model, the specification of a web service is done as a servlet class in web.xml while all other specification of additional configuration is in webservices.xml. The servlet framework is used to publish the web service in a JSR-109 deployment supported container. In the case of configuring a web.xml with Java EE5, the use of annotation can be optional. This simplicity works more on JSR-109 supported Java EE Containers. In JAX-WS RI deployment model, configuration of web service specification is done in sun-jaxws.xml and the RI servlet classed is specified in web.xml. Therefore, it would be said that web service deployment on plain servlet container uses the JAX-WS RI deployment model. There is a need for minimal configuration of web.xml and sun-jaxws.xml before the deployment of JAX-WS web services on a Servlet Container.

4.4.4 Web Service Standards Support

WS-Standards support is used as a decisive factor for selection of a SOAP engine. This is so because using a web service Standard like Web Service Security Standard (WSS) as an example: services can receive massages that are secured with signatures described in the WSS.

A) Axis2

Axis2 has support for a variety of WS standards which are as follows; JAX-WS - Java API for XML-Based Web Services (JAX-WS) 2.0, Web Services Metadata for the Java Platform, JAX-RS - The Java API for RESTful Web Services, SAAJ - SOAP with Attachments API for Java (SAAJ), Basic support: WS-I Basic Profile 1.1, Quality of Service: WS-Reliable Messaging, Metadata: WS-Policy, WSDL 1.1 - Web Service Definition Language, Communication Security: WS-Security, WS-SecurityPolicy, WS-SecureConversation, WS-Trust (partial support), Messaging Support: WS-Addressing, SOAP 1.1, SOAP 1.2, Message Transmission Optimization Mechanism (MTOM).

B) CXF

CXF has support for a range of web service specifications which includes WS-Addressing, WS-Policy, WS-ReliableMessaging and WS-Security, WS-SecureConversation, WS-Security, WS-SecurityPolicy, SOAP, WSDL and WS-Trust. CXF also implements the JAX-WS APIs which gives it the advantage of building web services with ease. The support for JAX-WS covers many other areas like the generation of WSDL to and from Java classes and the provision of API which enables the creation of simple messaging receiving server endpoints and dispatching API which allows the sending raw XML messages to server endpoints

C) Metro

Metro supports these Web service Standards: JAX-WS 2.0/2.1/2.2, WS-I Basic Profile 1.2 and 2.0, WS-I Attachments Profile 1.0, WS-I Simple SOAP Binding Profile 1.0, WS-Addressing 1.0 - Core, SOAP Binding, WSDL Binding.

4.4.5 Support for Multiple Data Binding

A Web Service platform works in combination with a Data Binding framework. As has been described a data binding implements mapping of XML elements and Java objects between each other. Of all the functions of a data binding, the most vital is the provision of data conversion. There are other functionalities that can be achieved with data binding which are the provision of support sdl2java code generation and the production of XML schema.

A) Axis

Data binding in Axis2 is by design not included in the core of its architecture. Therefore, the code generation allows the plug in of different type of data binding frameworks. This plug in is made possible by an extension mechanism as created in the design. In this extension mechanism, firstly, the extension is called by the code generator engine and then executed by the core emitter. The data binding supported by Axis includes but is not limited to: Axis2 Data Binding, XML Beans, JIBX, JAXMe and JAX Bri.

B) CXF

CXF provides data binding components that transparently handle the mapping. Currently supported data bindings include JAXB 2.x (default), Aegis, Apache XMLBeans, and JiBX

C) Metro

Metro does data binding through JAXB 2.0 only.

4.4.6 Asynchrony support

Asynchronous communications play an important role when it comes to Web services. Asynchronous or non-blocking Web Service invocation is a major requirement in Web Services. This nature of communication is common in enterprise platform integration connecting disparate systems.

A) Axis

Axis2 pioneers a convenient client API for invoking services. This client API is made up of two classes called Service Client and Operation Client. The Service Client API is used regularly when the requirement is for sending and receiving XML. On the contrary, the operation client is used in advanced cases when working with SOAP header or any other advanced task. With Service Client, either the SOAP body or the payload can be access. Although SOAP headers can be added, there is no way to retrieve the SOAP header by using the Service Client. Then, Operation Client will be used for the function.

B) CXF

CXF provides support for JMS transport for its services, and enables them to exchange messages asynchronously. JMS is a Java standard that provides a platform to develop applications that can communicate asynchronously with external systems. In JMS, the messages are exchanged using two popular communication models, Point-to-Point (P2P) and Publisher-Subscriber (Pub-Sub).

In the P2P model the messages are exchanged through the concept of queues. Each message has only one consumer. P2P is used to process messages synchronously and asynchronously. In the Pub-Sub model, the messages are exchanged through the concept of topics. A consumer subscribes to a topic in order to receive the message. A message in this model can only be exchanged asynchronously. Queue and Topic are called as destinations.

C) Metro

With Metro, a client application can use the JAXWS which enables Asynchronous Mapping binding declaration so that web service import can generate asynchronous polling and callback operations along with the normal synchronous method when it compiles a WSDL file. Asynchronous invocations require special consideration. The first form is the one that invokes Asynchronous method also known as the polling method. The response returns to the user immediately and may be polled for completion. In the meantime, the client program can do other work.

4.4.7 Support for Rest web services

Web services have become a standard way to achieve interoperability between systems. There are two main approaches for developing web services; one is by using the Simple Object Access Protocol (SOAP) and the other is by using the Representational State Transfer (REST) architecture style. Services built using the REST architecture style which has been termed RESTful services transports a simple XML encapsulated data over HTTP just like a web page request to the web server. This simplifies the development of web services without imposing overheads caused by the SOAP-based development approach. RESTFul web services are

particularly useful when it is only necessary to transmit and receive simple XML messages. The REST architecture is represented as a resource and identified by a Uniform Resource Indicator (URI). From literatures, we understood that all three investigated platforms have good support for rest web services.

4.5 Qualitative Comparative Framework

The three platforms evaluated seem to compete well. The qualitative comparison performed on some of the features of the selected platforms is summarized as follows:

Software Architecture: The Axis2 Architecture presents a modular, flexible and configurable architecture. Its modular nature allows additional functions in the form of modules for supporting different web services standards to be installed. CXF Presents a flexible and embeddable component based architecture. CXF is comprised of a backbone component which the bus. This bus component describes a common ground for all the endpoints by wiring runtime infrastructure components and providing a common application context.

XML Parser and Processing Model: The XML processing model for Axi2, CXF and Metro are based on StAX API (Streaming API for XML). Axis2 uses its own customized StAX API known as AXIOM.

Table 4.2: Qualitative Evaluation Framework

Criteria	Axis	CXF	Metro
Software Architecture	It has a modular architecture which is flexible and configurable	It is a component based architecture	It has a layered architecture
XML Parser and Processing Model	XML processing model is based on AXIOM (Axis Object Model) and StAX parser	StaX Parser	Streaming API for XML (StAX) for XML processing
Programing model and Deployment Model	It has many features which one of prominent is the handling of objects as POJO which allow for easy development and publishing of Web Services	It uses annotations as a easy way of applying behaviours to user-defined classes and methods that must be declared in some external source (such as an XML configuration file) or programatically (with API calls).	Supports two deployment models to publish web services, one using JSR109 deployment model and the other JAX-WSRI specific deployment model.
Web Services Standard Support	WS-reliable messaging, WS-Coordination, WS-Atomic Transaction, WS-Security, WS-Addressing	WS Addressing, WS Notification, WS Reliable Messaging, WS Policy, WS Security	WS-Addressing, WS-I Basic Profile , WS-I Simple SOAP Binding Profile WS-I Attachment Profile
REST Support	✓	✓	✓
Asynchrony support	✓	✓	✓
Support For Multiple Data Binding	Supports Pluggable Data Binding. There is option to use, ADB, XMLbeans, JiBX, JaXMe and JAXBRI	There is the option to use Aegis, the xFire data binding, but also XMLBeans, Castor, JiBX, JAXB	Metro's databinding through JAXB 2.0

Programing Model and Deployment Model: Their programing models and Deployment models are unique to each of the platforms. It shows that what is lacking in a particular platform can be available in the other.

Web Services Standard Support: All three platforms support an assortment of different Web Services Standard Support.

REST Support: The three platforms have support for RESTful web services.

Asynchrony Support: the support for asynchrony is present in all three platforms since they are integrated with the JAX-WS 2.0. JAX-WS 2.0 main contribution to web services is to support asynchronous web services.

Support for Multiple Data Binding: Axis2 and CXF have support for different data bindings but Metro only has support for one.

4.6 Chapter Summary

In this chapter, we have presented the selection criteria for inclusion and with the selected tools, we have performed a qualitative evaluation of the Web Service Engines selected and have used it to draw up a comparative evaluation framework.

From the qualitative point of view, one would say that none of the Web Services framework is superior to the other, even though one can be better in some aspect than the other. The Axi2 structure is modular with many features and can be used as a standalone application server for Web services. One of its special features is the support of removable data binding frameworks. Its disadvantages come in its complexity and insufficient JAXWS support. CXF and Metro comes complete with JAXWS support which allow for seamless integration with spring framework. CXF is portable and easy to use. If there is a need for a SOAP engine to be embedded into existing software then the choice would be CXF.

The Metro Web Services stack is said to deliver secure, reliable, transactional interoperability between Java EE and .Net 3.0 in building, deploying and maintaining Composite Applications for Service Oriented Architecture. Metro ease-of-development features enables support for W3C

and WS-I standards such as SOAP and WSDL, asynchronous client and server, and its databinding through JAXB 2.0.

CHAPTER FIVE

QUANTITATIVE COMPARATIVE STUDY: PERFORMANCE ANALYSIS OF SELECTED SOAP ENGINES

5.1 Introduction

In recent times, there are a range of web services toolkits available, which are primarily for the development of SOAP web services which are built to be simple and flexible but with no consideration for performance. Web Services is more or less like remote computing, therefore, the performance of Web Services and their underlying development platforms are affected by factors like latency and network bottlenecks, most of which are caused by the xml or SOAP transmission and processing. XML is naturally in the form of messages; therefore, the likelihood of XML being the main contributing factor to extended response time is high. SOAP on the other hand is the accepted structure for XML processing and this processing is done by the SOAP engine. XML is human-readable and wordy, text-encoded, metadata-encoded as well as maintainable, but low in terms of performance. Due to these performance issues, this work did a selection of a few SOAP engines, critically compared their features using a designed comparative framework in order to identify critical characteristics that might be beneficial to the need of satisfying the deployment of a flexible e-commerce on demand web service.

As mentioned in section 4.2.5, three platforms were included on the basis of meeting specified criteria for inclusion. Therefore, this chapter discusses the results obtained based on the performance evaluation on the three selected platforms. The test results obtained are analyzed theoretically which helps in summarizing the resulting data obtained as results in a meaningful way in order to draw conclusions from the data's subsequent analysis.

For quantitative evaluation, the testing environment was set up to determine performance of the selected platforms based on the time they were each used for processing XML. The Integrated Development Environment used was described to explain some phenomenal characteristics required for Web Services Development.

Section 5.2 highlights the choice of testing tool and development environment. The test environmental set up was explained in Section 5.3. Section 5.4 describes experimental design used for the performance evaluation of the three SOAP engines discussed in Section 3.2. The experimental results were then described in sections 5.5, 5.6, 5.7 and 5.8. Then Section 5.9 discusses the experimental results. The chapter was then summarized in section 5.10.

5.2 Choice of Testing Tool and Development Environment

The selection of SOAPUI as a testing tool was due to some of its features (SOAPUI, online article) which includes, but not limited to:

I. Integrality with different Integrated Development Environments (IDEs) and other tools. This makes it a necessary part of a development process. There are plugins available for most of the IDE like Eclipse, IntelliJ IDEA, Maven, NetBeans, and JBoss.
II. Support for multiple protocols such as SOAP, REST, HTTP, and other web services standards
III. It is open source and has got an open source community and partners around it. This has increased the pace of its advancement.

IV. The graphical interface is easy to use. This makes it easier to work with SOAP and REST-based Web services. The design allows testing experience to both technical and non-technical users to be simple and inclusive.

V. It allows for customization and development of features as soapUI Plugins. This is due to its design which modular and Extension API which enables extensible test framework. Favorite features can be developed with ease through the use of Plugins.

In the case of choosing a development environment that was used in creating Web Services, the available options were Netbeans, Eclipse and IntelliJ. IntelliJ was not used because it was a commercial tool. This then limits the choice to either Netbeans or Eclipse, which are both open source. Netbeans was then chosen as a matter of personal preference due to the fact that both Netbeans and the other counterpart Eclipse were reviewed for specific functionalities overviewd below.

The following functionalities were reviewed ;

I. Support for the selected Web Service Engines

II. Support for standards like J2EE 1.4 and Java EE 5, Java EE 6, including the JAX-WS 2.2, JAX-RS 1.1, JAX-RPC (JSR-101)* and JAXB 2.2 web service standards.

III. Ability to create and deploy a SOAP engine service from either a Java class or a WSDL document.

IV. Ability to work with databases- If there is a need to access the database to enable change

V. Support for the creation and development of web services from Java classes or WSDL files through the use of the Web Services wizards and Visual Designer.

VI. Enablement for SOAPUI plugin. This plugin enables the creation of web service testing projects which include test cases and allow SOAP monitoring.

5.3 Test Environment

This section describes the test environment that was used for the performance measurements. The testing tool used is SOAPUI, which is an open source functional testing tool for SOA and Web Service testing. The test system consisted of an integration of the testing tool, SOAPUI with the development environment, Netbeans. This integration consisted of WSDL files operations with the SOAP calls bindings and the reading of trace data from local files with a driver. Each of the selected SOAP engine is made to implement the operations defined in the WSDL document for evaluation. The test setup was running on windows XP Professional as the host operating system. The tests were carried out on a Desktop workstation with 2.79GHz Pentium 4 processor, 400MHz front-size bus and 1.24 GB of RAM. The Java code was run on Tomcat. The Web service stack versions were Metro with JAXWS as the data binding framework, Axis2 with its native data binding, which is known as Axis Data Binding (ADB) and CXF also using JAXWS as data binding framework.

To obtain optimum results, we assumed that there is no overhead cost and network latency.

5.4 Quantitative Evaluation

In the web service scenario, the consumer and service provider communicate with each other through the exchange of messages. The messages are marshalled at the client end and unmarshalled at the server end. In web service terminology, marshalling is the process of converting Java objects to XML files, which are to be sent over a network. Unmarshalling refers to converting an XML file back to a Java object. These marshalling and unmarshalling takes place when a web service invocation is initiated.

Marshalling**,** even though considered synonymous with serialization in some aspects, they are not the same. When an object is marshaled, a record of the state and codebases is made in that when the marshaled object will be unmarshalled, the original copy can be acquired by the class definition of the object through automatic loading. Therefore, any object that can be serialized can also be marshaled. Serialization process involves the breaking down of object to a form that can be sent across the network. Before an object is sent across the network, it must be serialized. After serialization comes Marshalling of the encrypted object across the network. Therefore, in this research work, the performance metrics are as follows:

Marshalling Time is the time it takes to generate an XML representation for the object in memory as the request is made.

Unmarshalling Time as the reverse of marshalling is the time it takes to build an object in memory from an XML representation.

Round Trip Time is the time it takes to send a message or request from the client to the server and receive a response message back from the server to the client.

This research work takes the approach of measuring the time required to execute a particular sequence of requests when both the client and server are running on a single machine which give room to eliminate the effect of network latencies and overhead from timing results. Each test was performed in a standard procedure to enable quantitative measures of the outcome and to achieve a fair basis for a comparative evaluation. The overall set up on the server side consists of a web server, a SOAP engine (web services engine), and the web service implementation.

In performing these tests that require the given metrics, a simple ecommerce application was designed and deployed using each of the Web service engine. This ecommerce application is made up of a web service that took a range of object as parameter and returns the same to the client.

The schema used is representing a collection of four products with different price tags in the form of a quotation service. The quotation service uses a created database of prices of four different products.

The designed procedure measures the performance of each case with different product value ranging from one to up-to one million products in quantity.

In this work, the performance evaluation uses a set of request sequence. Firstly, the query parameter is adjusted and used to match the information on the database. Then, the invocation of the web service is done by a single client which enables the time taken to unmarshal, marshal and complete a roundtrip to be determined.

The load test stimulated invocation of web service for a period of one minute with a number of users simultaneously and repeatedly. The results are recorded to generate reports and study for Transaction per second and speed of each web service engine.

When the consumer makes a request on the remote service, the data is first marshalled and placed over the network to be sent to the server. The server receives this marshalled data, unmarshalls it, and invokes the service method. The process is repeated in the same manner when the server sends back the response to the client. Marshalling and unmarshalling are the core services that are provided by client and server at runtime. The processes of marshalling and unmarshalling form part of bottlenecks that can affect the efficiency of a Web service engine. As a result, using the data collected during testing, average values were calculated and graphs were plotted to represent marshalling, unmarshalling and round trip as it corresponds to the tools under investigation.

Using the results of the performance metrics during the first phase of performance testing, average values for marshalling, unmarshalling and round trip times were calculated based on the results of the test performed.

5.5 Experimental Results and Analysis

This section describes in detail all experiments and analysis. It also discusses the results that were obtained. The purpose of this analysis is to understand the impact of the cost of processing XML. As discussed in Section 3, the experiments will also establish baseline metrics for future work on tradeoffs in complex environments as this in particular is made to be as simple as possible without giving anything ambiguous.

5.5.1 Experiment I: Response Time Performance

An experiment was conducted to determine the effect of varying the quantity of products requested on the response time. This experiment was set up as follows: requests were made for Product A with varied product quantity, two different products (Product A and B) requests were made with varied product quantity, and in the same vein, three different products, A, B and C requests were made, respectively.

Product A

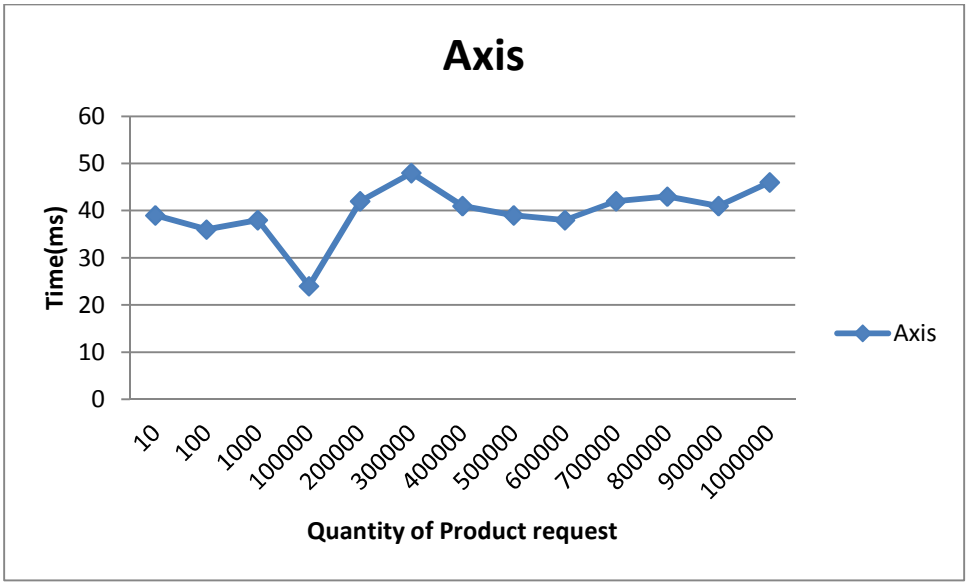


Figure 5.1: Time(ms) vs Quantity of Product Request for Product A on Axis2

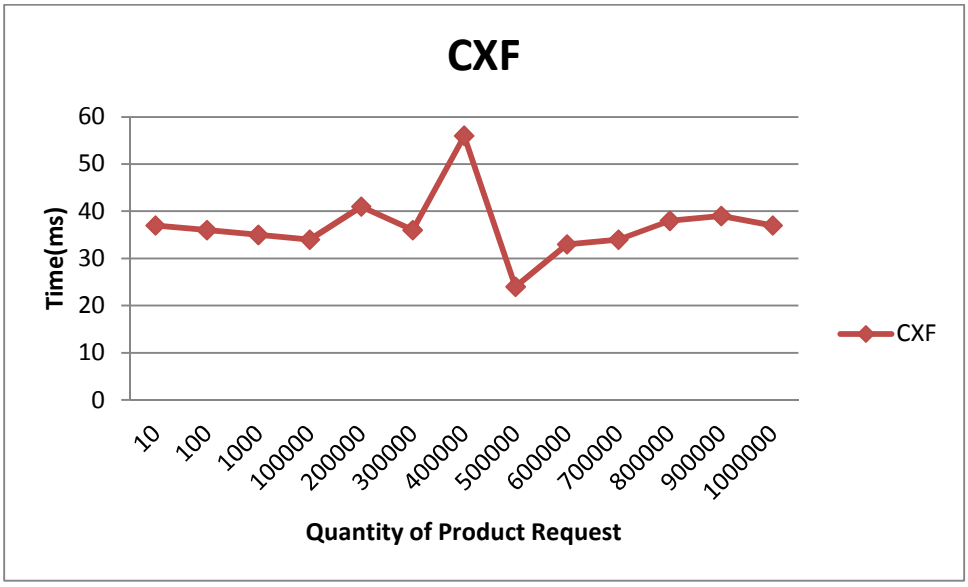


Figure 5.2: Time(ms) vs Quantity of Product Request for Product A on CXF

Figure 5.3: Time(ms) vs Quantity of Product Request for Product A on Metro

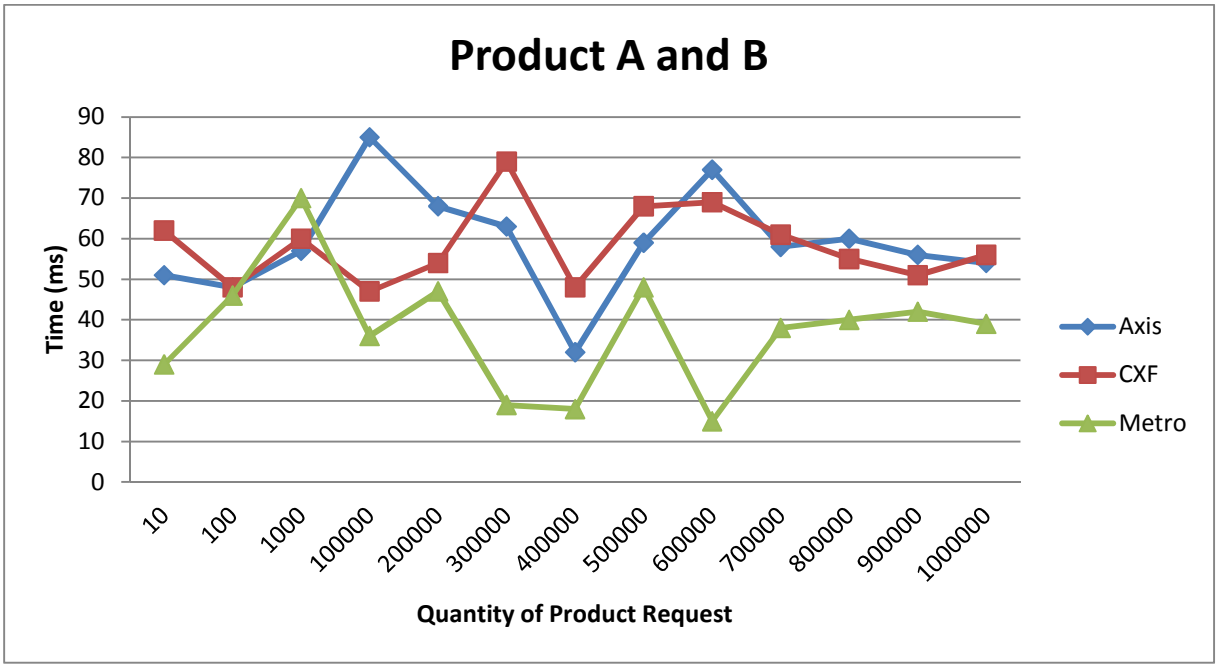


Figure 5.4: Time(ms) vs Quantity of Product Request for Product A and B

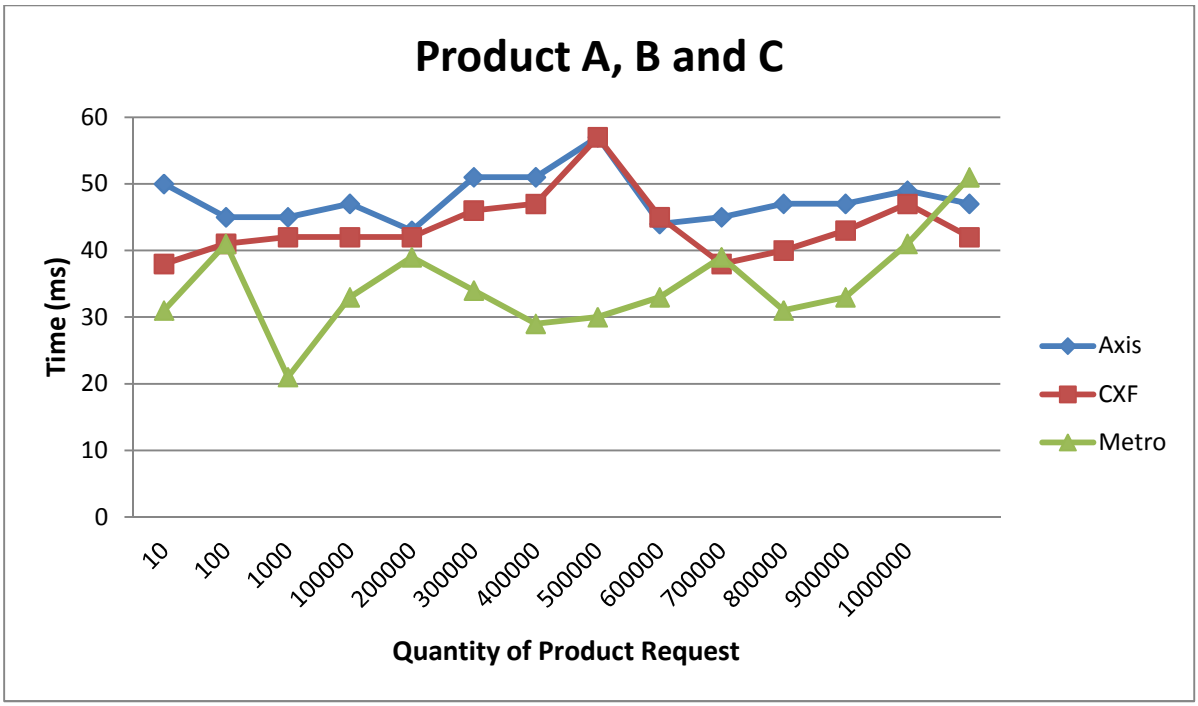


Figure 5.5: Time(ms) vs Quantity of Product Request for Product A, B and C

It was envisaged that the increase in the quantity of product request on SOAP engine would affect or even delay the response time, but the experiment gave a different results. It was observed that the increase in the quantity of Product Request does not in any way affect the response time. From the above given graphs, using Figure 5.1, 5.2 and 5.3 for instance, an increase in the quantity of product requests from 700,000 to 1,000,000 maintained almost a steady response time of 42, 43,41 and 46 for Axis, 34, 38, 39, and 37 for CXF and 38, 35, 34, and 29 for Metro.

Owing to the result from Experiment I, which shows that the increases in quantity of products requested does not have an effect on the response time. Thus, the effect of increasing to the quantity of product request be considered negligible in the remaining experiments. Then, the

varying of product request with regards to Product A, Products A and B, Products A, B and C and Products A, B, C and D will be used.

5.5.2 Experiment II: Marshalling Performance

The aim of this experiment is to measure the total time elapsed between the business logic completing execution and the SOAP engine servlet sending the reply message. This experiment in turns refers to using marshalling as a parameter for measuring performance of Web Service engine. In essence, the aim of the experiment was to ascertain the time it actually takes to complete the process of converting the Java objects to XML as a web service request is made since both the request and response of a call are to be converted into a form suitable for transmitting over a network.

Since XML parsing, de-serialization and serialization are the three most time-consuming stages in the process of handling SOAP messages, they also determine the Web Services performance to a large degree, especially when effective load is increasing. Finding out the effects of XML parsing, serialization and de-serialization are three key factors that would help in improving the performance of Web Services.

Building on the results of the experiments, Figure 5.1 illustrates the performance measures of each web service engine.

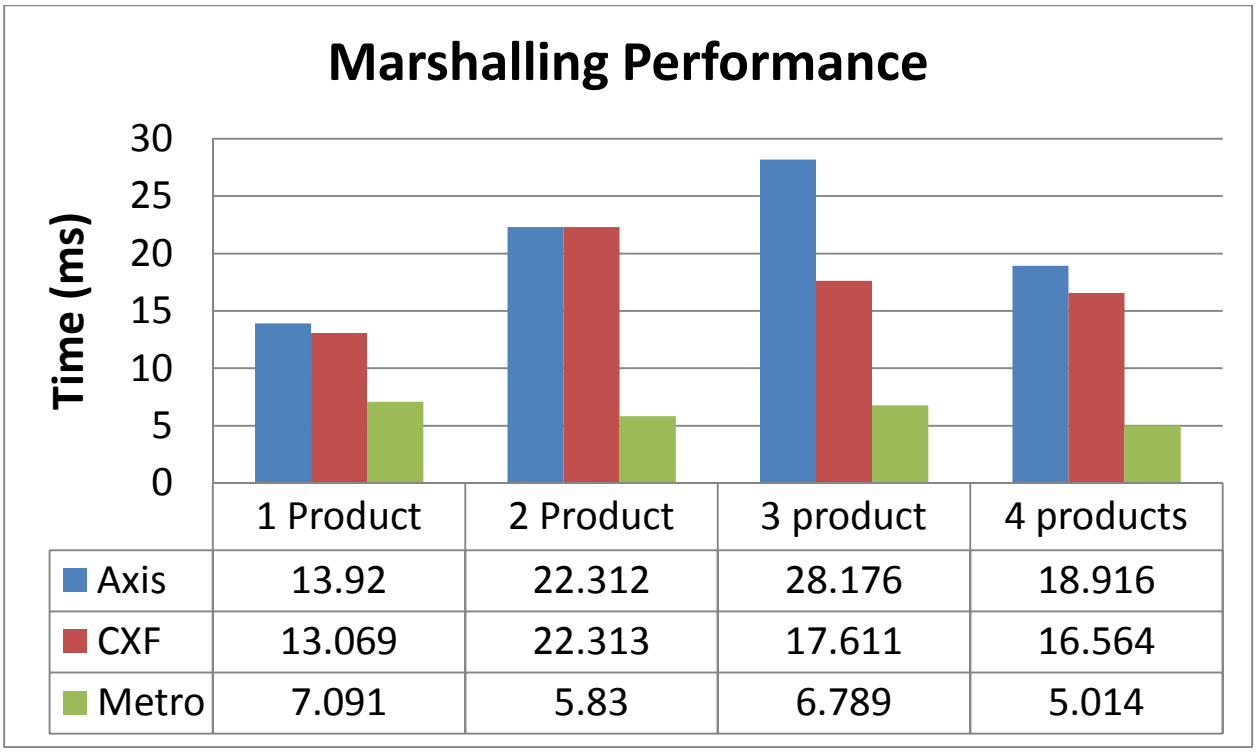


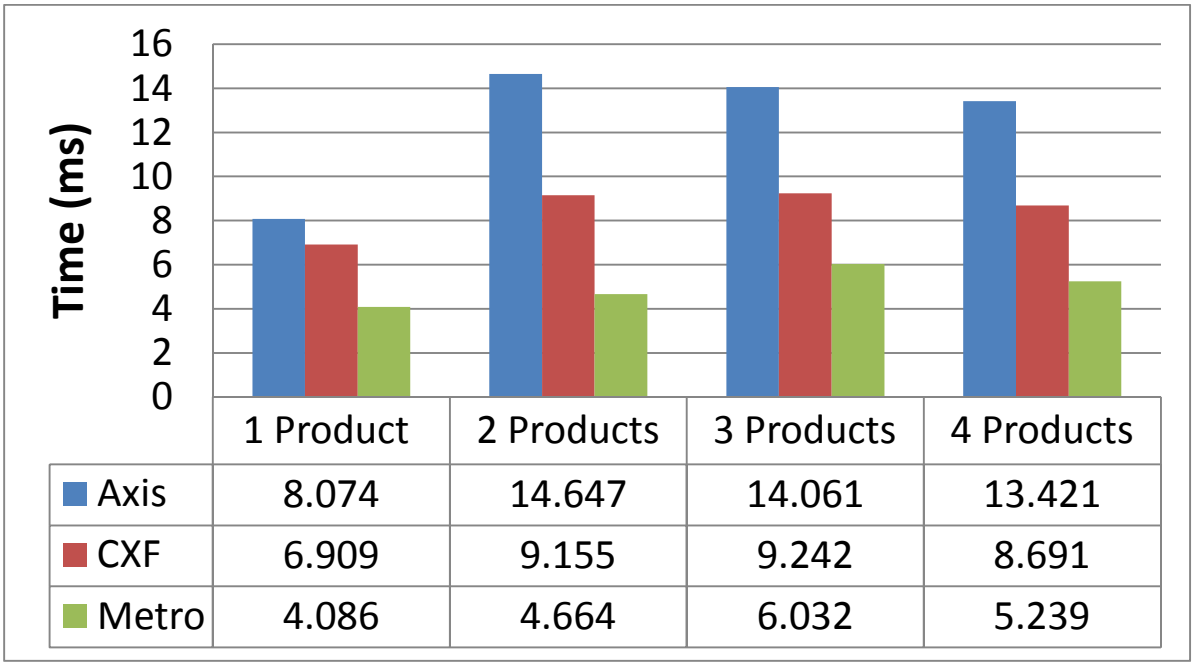
Figure 5.6: Time(ms) vs No of Product Request (Marshalling)

Figure 5.6 shows the marshalling performance. It is interesting to note from the above given graph that Metro fared two to three times faster and better than both Axis and CXF. Axis and CXF match marshalling performance closely among themselves for the time taken to complete marshalling, when one product and two products were requested. Consequently, for the time taken when three different products were requested, CXF fared better than Axis.

Therefore, Metro offers the best marshalling performance while Axis and CXF fared almost the same. This could be due to its usage of JAXB2.x as data binding framework. JAXB binding provides a way to convert from Java technology objects to XML and from XML to Java technology objects.

5.5.3: Experiment III: Unmarshalling performance

The purpose of using unmarshalling time as a parameter is same as the reason for using marshalling as a parameter since they both form part of the web service bottleneck. In web services performance evaluation, the XML parser speed when reading a SOAP message has been identified as a slowdown to performance. Size optimization is not supported in XML because of its verbose nature. SOAP on its own is XML based, therefore has most limitations associated with XML. When parsing SOAP message, which includes all typing information, multiple parsing passes may be required by the XML parser to extract the SOAP envelope and body from the SOAP packet.



	1 Product	2 Products	3 Products	4 Products
Axis	8.074	14.647	14.061	13.421
CXF	6.909	9.155	9.242	8.691
Metro	4.086	4.664	6.032	5.239

Figure 5.7: Time(ms) vs. No of Product Request (Unmarshalling)

Figure 5.7 depicts the unmarshalling performance of the three tested web services platforms.

The figure indicates that Metro outperforms CXF and Axis in all product requests while CXF follows suit by outperforming Axis also in all request. The difference in Axis and CXF when one product was requested is minimal.

5.5.4: Experiment IV: Round Trip Performance

The purpose of the measurement of the time taken from the invocation of service by the client and the receiving of the final object by the client is to indicate its measure of both processing time and transmission time.

In the context of this research work, testing various SOAP engines simultaneously is important to enable their comparison and the selection of the best fit based on the performance needs of targeted applications, thereby, the evaluation of the round-trip time provided by these Web service platforms. Even though the metric might be perceived as simple, it provides a relevant view for comparing the performance that can be provided by each platform.

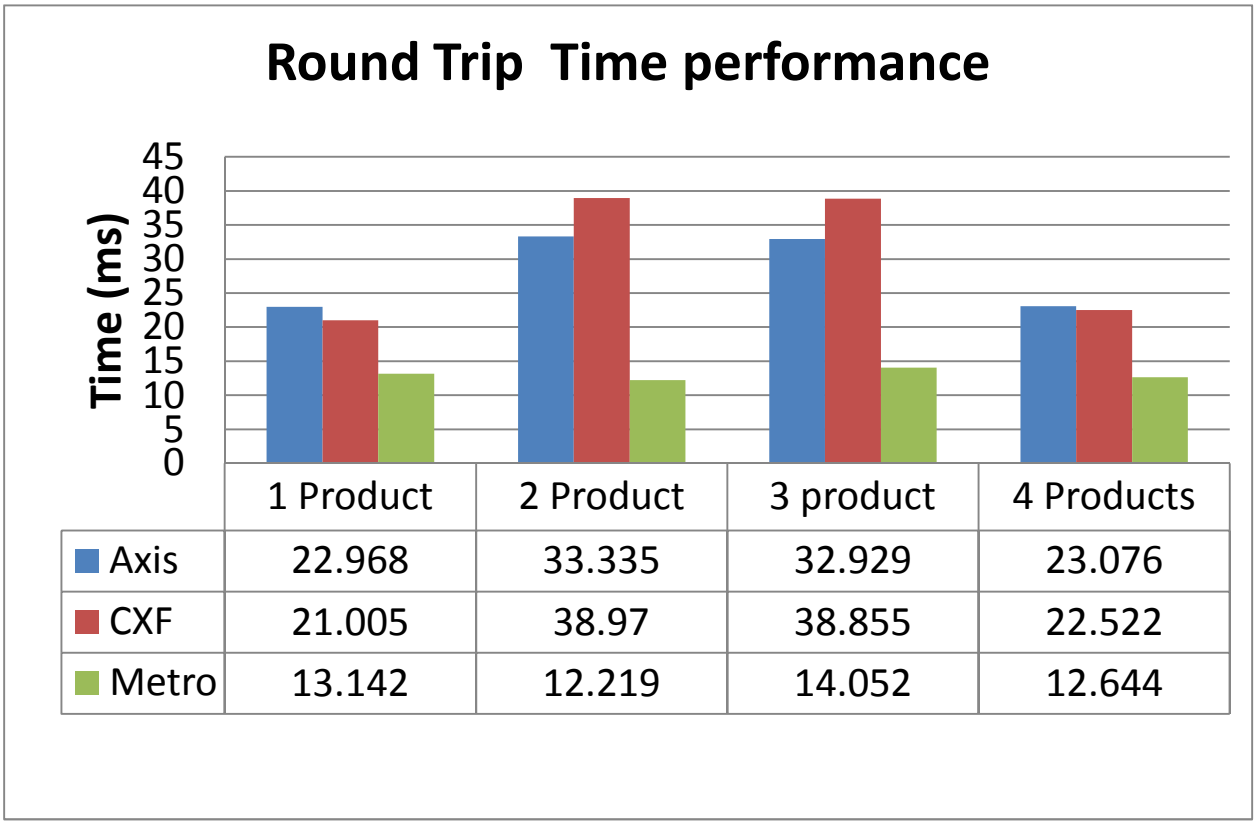


Figure 5.8: Time(ms) vs. No of Product Request (Round Trip)

In Figure 5.8, the chart above clearly shows that metro is the most efficient of the three as it provides the best round trip latency result. Axis and CXF provide the almost the same round-trip timing results. This could be because these three platforms have been developed, optimized, and used in different cases.

5.5.5: Experiment V: Load Testing Performance

Load testing is an integral part of any efficient testing scheme, therefore, in this research we consider it necessary to involve number of transactions a particular platform can carry in a space of 60 seconds.

The Transaction per second (TPS) timing can be described as the web service ability to serve clients under varying levels of load. In this experiment TPS is tested to ascertain the number of transactions that can be achieved in a given time space.

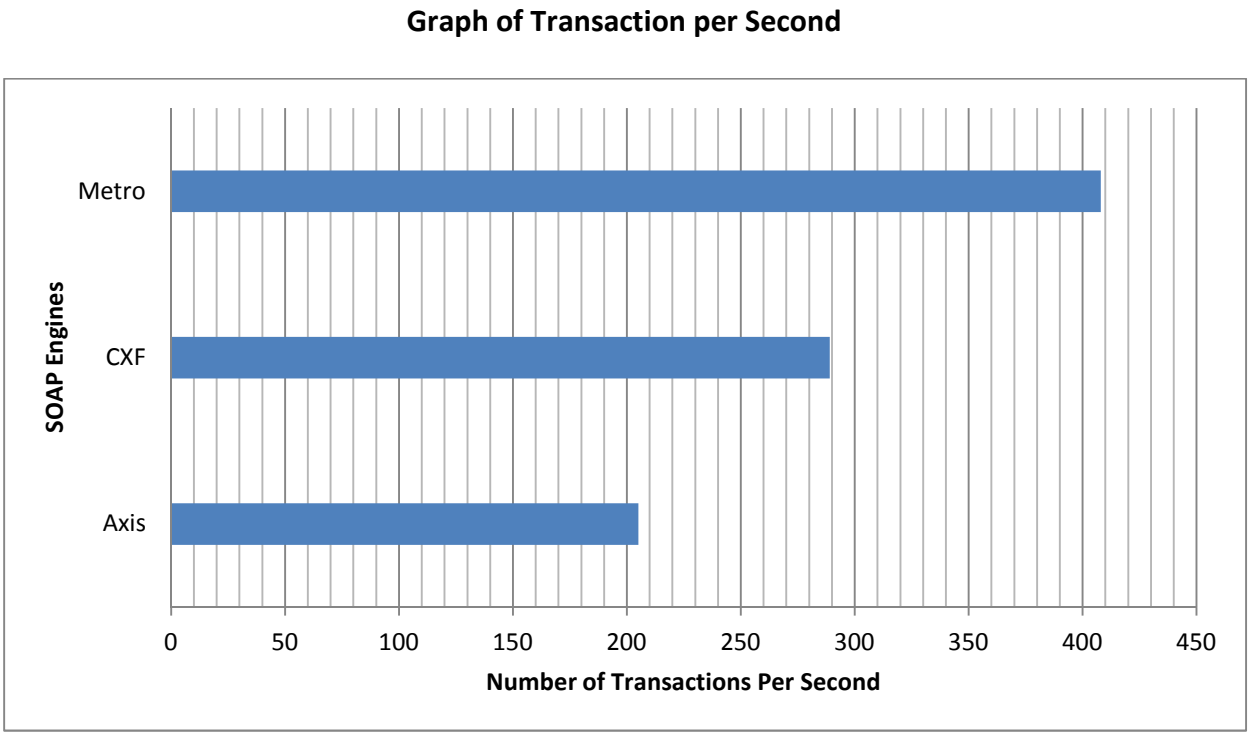


Figure 5.9: Transaction Per Second

The graph in Figure 5.9 shows the performance of Metro, CXF and Axis based on Load testing them for a period of 60 seconds. The x-axis represents the respective SOAP Engines, and the y-axis represents the number of transactions. In this scenario, a transaction is seen as the sending and receiving of an array of items.

5.5.6 Discussion of Result

Web services performance is relative to the amount of payload sent. The relationship is based on the increase in the payload sent resulting in the increase in both the serialization and deserialization processing resource, and also the processing for binding and parsing. The performance of web service platforms is generally considered to be affected by this relationship. The amount of payload was not put into much consideration in this research work.

Since the choice of XML parser has an effect on unmarshaling, and also affect web service performance, and in this case the three SOAP engines under investigation are all using the pull parser model that has been optimized for speed and performance; and also allows for validation to be easily built into the generated marshalling code thus the implication of level of speed on unmarshalling performance due to the parsing model is eliminated. Therefore, other factors could then cause either one to be better than the other. One attribute could be the data-binding framework used for their respective xml binding. In the case of axis 2 which uses its built-in/native data binding known as AXIOM which is developed specifically for axis2.

AXIs Object Model (AXIOM) which is StAX based object model is made to enhance performance for both CPU and Memory intensive applications. Although Axis2 with its AXIOM are meant to be memory-efficient by deferring creation of XML tree when not required.

Even with all the good qualities and the efficiency of AXIOM, the experimental results show that it does a good work, but not as much as the XML parser used in both of CXF and Metro. Metro and CXF timing results in most of the evaluations conducted are comparable. This could be due

to their use of JAXB 2.0 as the data binding framework. Metro, on the other hand, was the best performer of the entire three evaluated web service platforms. This could be due to its enhanced features which are based on the close ties with specifications developers.

5.6 Chapter Summary

This work which looked at performance of three Web service Engines is meant to help in recommending a particular platform for GUISET services development. The use of performance as a decisive factor is because performance of any given software based system is one of the primary attribute that can help in shaping the architecture and subsequently the efficiency of the system. In view of the fact that Web services is the widely preferred architecture for implementing client-server systems.

Based on test timings reported in this chapter, Metro is faster in basic request response processing than either Axis2 or CXF. The recommendation for GUISET service development will be emphasized in the next chapter (Chapter 6).

CHAPTER SIX

SUMMARY, CONCLUSION AND RECOMMENDATION

6.1 Summary

Due to the emergence of web services and the popularity it has achieved, it has become an important technology for communications between disparate applications in both different enterprises and within a single enterprise. The performance of core SOAP engines has, therefore, become crucial and has been taken into consideration in this work**.** Performance as an attribute is often under tension and this tension is amplified in systems that use XML- based web services.

This research report started with the presentation of a gap that needs to be filled as regards exploring Web service platform that could be used in developing and deploying GUISET services. The main purpose of this work was to evaluate a few leading open source platforms for web service development and recommend one for GUISET. This evaluation was to be achieved by reviewing existing SOAP engines and selecting three for the evaluation. The primary basis for the decision between the three different web service platforms investigated is based on individual characteristics in such areas as the underlying architecture, programing and deployment model, messaging model and other features like asynchrony support, web services standard support, response time, and resource usage. For the purposes of the experiments performed during the course of this research work, three different SOAP engines were chosen for investigation after an implementation of the criteria for inclusion.

This chapter summarizes and concludes the work done in this research. The remainder of the chapter is arranged as follows. Section 6.2 presents the conclusions drawn from this research. The recommendations achieved in this research work were presented in section 6.3 and section 6.4 then presents the future work.

6.2 Conclusions

In conclusion, comparative evaluations of three commonly used and available web service platforms (Axis2, CXF and Metro) have been presented both in their features and through experimentations.

Qualitatively, some specific features of the SOAP engines were compared and conclusion drawn. One would say that no particular Web Services framework is in general superior to the others, even though one can be better in some aspect than the other. Axis2 has many good features; its architecture structure is modular, it can be used as an application server for Web services if need be. Its support of removable data binding frameworks is a merit on the platform. Its demerits come in lack of support for JAXWS support. CXF and Metro comes complete with JAXWS support which allow for seamless integration with spring framework. Metro Web Services stack is interoperable, secure, consistent and transactional. Metro builds, deploys, and maintains Composite Applications for Service Oriented Architecture. Metro has unequivocally support for W3C and WS-I standards such as SOAP and WSDL, asynchronous client and server, and its data binding through JAXB 2.0. CXF is said to be easy to use, convenient and has support for JAXWS.

There are also some underlying components that all three SOAP engines share within their different underlying components. CXF web services stacks builds on virtually the same technologies as Axis2 and Metro in some aspects. As has been mentioned in the previous chapters, CXF primarily uses JAX-WS 2.x web service configuration and JAXB 2.x data binding just like metro and like Axis2, it uses the WSS4J WS- Security.

The quantitative performance attributes a high performance rate to Metro than any other of the three Web Services platforms evaluated which is then followed closely by CXF and then Axis.

6.3 Recommendations

As mentioned in the preceding section, Metro performed better than CXF and Axis during experimental evaluation**,** therefore, this research work makes a recommendation that Metro is a better choice to Developing and deploying GUISET services due to its performance. In addition, when coding against standards is required, then Metro is also a good option, because it offers the most comprehensive support for the WS-* standards. The Key merit of Metro is the compatibility it has with Microsofts Windows Communication Foundation.

CXF would be recommended as a tool of choice for GUISET, when and if GUISET requires the SOAP engine to be embedded into an existing software and also, if it requires a seamless integration with Spring framework.

6.4 Limitations and Future Work

For the evaluation done in this research work, the performance metrics considered are only the marshalling, unmarshalling and round trip timing which all inculcates serialization. There are also different metrics which can be involved in the stages of sending and receiving SOAP messages that can be parametized and even optimized.

There is a possibility of an effect on performance due to the data binding framework used. Each of the frameworks evaluated comes with a native data binding framework. Therefore as a recommendation for the future, this research can be extended further by testing these Web Services stacks with different flavors of pluggable data bindings and also the stages of sending and receiving SOAP messages can be optimized.

REFERENCES

Adigun, M.O., Emuoyibofarhe, O. J. and Migiro, S.O, (2006). "Challenges to Access and Opportunity to use SMME enabling Technologies in Africa", A Presentation at 1st All Africa Technology Diffusion Conference, Johannesburg- South Africa, June 12-14, 2006.

Alex, N., Chen, S. and Greenfield, P., (2004). "An evaluation of Contemporary Commercial SOAP Implementations", Proceeding of the 5th Austialasian Workshop on Software and System Architecture, Melbourne, Australia, April 2004. Pp. 64 – 71.

Apache Software Foundation, (2011). "Apache Axis". Available from http://ws.apache.org/axis,2003 Last accessed: 5th October 2011.

Apache Software Foundation, (2011). "Apache Axis2 Architecture Guide" Available from http://axis.apache.org/axis2/java/core/docs/Axis2ArchitectureGuide.html Last accessed: 15th November 2011.

Apache Software Foundation, "Apache CXF Software Architecture Guide" Available from http://cxf.apache.org/docs/cxf-architecture.html Last Accessed: 19th October 2011

Arun Gupta "Metro – The Web Services Stack in Glassfish". Available from http://blogs.sun.com/arungupta Last accessed: 1st November 2011.

Asif, M., Majumdar, S., and Dragnea R., (2007). "Hosting Web Services on Resource Constrained Devices," IEEE International Conference on Web Services (ICWS 2007), pp.583-590.

Balani, N., and Hathi, R., "Apache CXF Web Service Development" Packt Publishing, Birmingham – Mumbai Dec 2009.

Bayer, T., (2010). "How to Support Web Services in Products" Available from http://www.predic8.com/how-to-support-web-services.htm Last accessed: 4th Feb 2010.

Bichler, M. and Lin, K.-J. (2006). Service-Oriented Computing. Computer, 39(3):99–101.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D., (2004). *Web Services Architecture.* W3C Working Group – February 2004 Available from http://www.w3.org/TR/ws-arch/ Last Accessed: Oct. 22,2011]

Box, D., Ehnebuske, D,Kakivaya, G., Layman, A., Mandelsohn N., Nielsen H., Thatte S.,and Winer D. (2009). "Simple Object Access Protocol(SOAP)" Available from http://www.w3.org/TR/soap Last accessed: 10th March 2010.

Chavan, A., (2005). “Seven Criteria for Evaluating Open-Source Content Management Systems” Linux Journal Website, 2005. http://www.linuxjournal.com/node/8301/.

Chiu, K., Govindaraju, M., and Bramley R., (2002). “Investigating the Limits of SOAP Performance for Scientific Computing” Pros of 11th IEEE International Symposium on High Performance Distriuted Computing(HPDC 2002) Edinburgh: Scotland, pp. 246 – 254,2002.

Clements, P.C., and Northrup, L.M., (1996). “ Software Architecture : An Executive Overview,” Technical Report No. CMU/SE!-96-TR-003, Software Engineering Institute, Carnegie mellon University, Pittsburg, PA, February, 1996

Crosby, P.B (1996). “Quality of free; The Art of Making Quality certain. New York/London: McGraw-Hill” ISBN:0070145121

Curbera, F., Nagy W. A. and Weerawarana, S., (2001). “Web services: Why and How”, IBM T.J. Watson Research Center, August, 2001.

Davis, D., and Parashar, M., (2002). “Latency Performance of SOAP Implementations”, Proceedings Of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), Berlin, Germany, pp. 407 – 412, 2002

Deepal Singhe JayaApache Axis2 PackT Publishing, May 2008.

Devaram, K.,and Anresen, D.,(2003). “SOAP Optimization Via parameterized Client-Side Caching”, Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems, Marina Del Rey, pp. 785 – 790, 2003.

Duijnhouwer, F., and Widdows C., (2003). “Capgemini Open Source Maturity Model” http://www.capgemini.com/technology/opensource/solutions.shtml, August 2003. http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_.

Elfwing, R., Paulsson, U., and Lundberg, L., (2002). ”Performance of SOAP in Web Service Environment Compared to CORBA” Proceedings of the Ninth Asis- Pacific Software Engineering Conference (APSE), Queensland, Australia, 2002.

Foster, I., Kesselman, C., and Nick J. M., Steven Tuecke, (2002). “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,” tech. report, Glous Project; http://www.globus.org/research/papers/ogsa.pdf (current June 2002).

Ferguson, F. F., Lovering, B., Storey, T. and Shewchuk, J., (2003). “Reliable, Transacted Web Service: Architecture And Composition”. MSDN technical articles, Sept 2003. Available from www.msdn.microsoft.com Last Accessed: 10th April 2010

Garlan, D., and Shaw, M., (1993). “An Introduction to Software Architecture,” in Advaces in Software Engineering and knowledge Engineering. Volume 2, V. Ambriola and G. Tortora, ed., Singapore, World Scientific Publishing, 1993, pp. 1 – 39.

Golden, B., (2005). “Succeeding with Open Source,” Addison-Wesley Pearson Education, 2005. ISBN 0-321-26853-9.

Govindaraju, M., Slominisk, A., Chiu, K., Liu, P., Van Engelen, R., and Lewis, M.J., (2004). “Toward Characterizing the Performance of SOAP Toolkits”, Proceedings of the 5th IEEE/ACM International Workshop on Grid computing, Pitsburgh, USA, pp. 365 – 372, 2004

Grimshaw, A.S., and Tuecke, S., (2003). "Grid Services Extend Web Services," Web Services Journal, August 2003.

Hassen, M. D., (2002). “SOA using Java web services” Prentice Hall Cape Town 2002.

Head, M., Govindaraju, M., Slominski, A., Liu, P., Abu-Ghazaleh, N., Van Engelen, R., Chiu, K., and Lewis, M. (2005). “A Benchmark Suite for SOAP-based Communication in Grid Web Services”. In *IEEE SC*, page 19, 2005.

Hodgman, R.**,** (2007). **“**SOA Case Study: Leveraging SOA in Business Processes” The additional enabler, policy-based, real-time session management, November 2007.

Hua, L., Wei, J., Niu, C., and Zheng, H., (2006). “*High Performance SOAP Processing Based on Dynamic Template-Driven Mechanism”*. Chinese Journal of Computers.2006, 29（7）: 1145 - 1156.

Hugo, H. and Ellen, B., (2004). “Web Service Glossary”, W3C Working Group Note 11 February 2004.

Hussin, H. and Noor, R. M., (2005). “Innovation Business Through e-commerce: Exploring the Willingness of Malaysian SMEs” The second International Conference on Innovations in IT, September 2005.

Java API for XML Web Services (JAX-WS) Users Guide. Available from http://jax-ws.java.net/nonav/2.2.1/docs/UsersGuide.html Last accessed: 10th December 2011

Kabanda, S.K., Iyilade, J.S., and Adigun, M.O., (2007). “Knowledge Resources in a Grid enabled Infrastructure – The case of deep rural SMMEs in South Africa”, In Proceedings of the 4th International Conference on Intellectual Capital and Knowledge Management (ICICKM, 2007), 15 – 16 Oct 2007, Cape Town, South Africa.

Kohlhoff, C., and Steele, R., (2003). “Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems”, Proceedings of the 12th International World Wide Web Conference (WWW2003), Budapest, Hungary, 2003.

Kunti, K., Muralidhar, R., and Badveeti, N., (2007). “A Performance Comparison of Popular Open Source Web Services Engines” Available from *http://soa.sys-con.com/node/439687?page=0,0* Last Accessed: 22 September 2009

Legner, C., and Heutschi, R., (2007). “SOA Adoption in Practice - Findings From Early SOA Implementations” Proceedings of the 15th European Conference on Information Systems, St Gallen, Switzerland, 2007: p. 1643 – 1654.

Machado, A. C. C., and Ferraz, C. A. G., (2005). “Guidelines for Performance Evaluation of Web Services” WebMedia 2005, December 5 – 7, 2005, Pocos de Cadas, MG, Brazil.

Mallal, R., (2010). “Pillars of SOA Testing”. Digital Software Magazine. Available from http://www.softwaremag.com/sponsor/sponsor-index/crosscheck-networks/thought-leadership/pillars-of-soa-testing Last Accessed: 10th March 2010

MacKenzie, C. M., Laskey K., McCabe, F., Brown, P. F., Metz, R., and Hamilton, B. A., (2006) “Reference Model for Service Oriented Architecture 1.0” Technical Paper by OASIS Standard, 12 October 2006. Availaible from http://docs.oasis-open.org/soa-rm/v1.0/ Last Accessed: 28th October 2009.

Markus, M., Gärtner, J., Dohmann, H., and Freisleben B., (2009). "SOAP4IPC: A Real-Time SOAP Engine for Industrial Automation," Proceeding of Parallel, Distributed and Network-based Processing, pp.220-226.

Msimanga, M. and Chadwick, J., (2004). “An Evaluation of Free and Open Source e-commerce web application technology with regard to SMMEs” Southern African telecommunication Networks and Applications Conference (SATNAC) 2004 Proceedings Vol 2, pp 3-8.

Muracevic, D., Kurtagic, H., (2009). “Geospatial SOA using RESTful Web Services” Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces, Page(s): 199 – 204.

Myung-Hee, L., Cheol-Jung, Y., Ok-Bae, J., (2009). “Embedded System Software Testing Using Mobile Service Based On SOA” International Journal of Advanced Science and Technology

Naveen, B., and Rajeev, H., (2009). “Apache CXF Web Service Development” Packt Publishing, Birmingham – Mumbai.

Ng, A., Chen, S., and Greenfield, P., (2003). "An Evaluation of Contemporary Commercial SOAP Implementations", Proceedings of the 5th Australiasian Workshop on Software and System Architectures, Melbourne, Australia, pp 64 – 71,2003.

Nickull, D., Reitman, l., Ward, J., and Wilber, J. "Service Oriented Architecture (SOA) and Specialized Messaging Patterns" (Technical White Paper)

Padmanabhuni, S., Majumdar, B., Mysore, U., and Sitaram, V., (2005). "XML Binding Frameworks in the Context of Service Oriented Architecture" Available from http://soa.sys-con.com/read/114130.htm last accessed: 10th June 2010

Papagozlou, M., and Georgaplus, D., (2003). "Service Oriented Computing", Communications of the ACM, Volume 46, No. 10

Perry, D. E., and Wwolf, A. L., (1992). "Foundations for the Study of Software Architecture," Software Engineering Notes, vol. 17, no. 4, pp.40-52.

Rodrinquez, A., (2008). "RESTful Web services: The basics" online article, November 2008

Shirasuna, H. N., and Sekiguchi, S., (2002). "Evaluating web services based implementations of GridRPC". In *IEEE International Symposium on High Performance Distributed Computing HPDC-11*, page 237, 2002.

Smith, D. M., Abrams, C., Sholler, Daniel., Plummer D. C., and Cantara, M., (2005). Magic Quadrant for Web Services Platforms, 2005

SOAPWare.Org, "The Leading Directory for SOAP 1.1 Developers", Available from http://www.soapware.org/directory/4/implementations,2005 Last Accessed: 24 September 2011

Sosnoski, D., (2003).Online article on XML and Java technologies: Data binding, Part 1: Code generation approaches -- JAXB and more

Sosnoski, D., (2009). "Java Web services: Introducing Metro" Available from http://www.ibm.com/developerworks/java/library/j-jws9/index.html Last accessed: 10th June 2010.

Srinath, P., Chathura, H., Jaliya, E., (2006). "Axi2, Middleware for next generation web services", ICWS 2006. Available from http://apache.axis.org/axis2/ Last accessed: 6th March 2010.

Srinivasan, L., and Treadwell, J., (2005). "An Overview of Service-Oriented Architecture, Web Services and Grid Computing," HP Software global Business Unit, November 3 2005.

Srivastava, P. R., and Vijay, R. K., (2009). "An Ant Colony optimization approach to test sequence generation for control flow based software testing", Proceeding of 3rd International Conference on Information Systems, Technology and Management. Ghaziabad india, March 2009.

Suzumura, T., Trent, S., Tatsubori, M., Tozawa, A., and Onodera, T., (2008). "Performance Comparison Of Web Services Engines in PHP," Java and C, 2008 IEEE

Tingbin, C., Wang L., Zhang Y., and Sun F., (2007). "Wireless Communications, Networking and Mobile Computing," 2007. WiCom 2007. International Conference on Page(s): 4706 - 4709

Tripathi, S., and Abbas, Q., (2010). Performance Comparison of Web Services under Simulated and Actual Hosted Environments International Journal of Computer Applications (0975 – 8887) Volume 11– No.5, December 2010.

Using NetBeans IDE 5.5. Avalaible from http://netbeans.org/project_downloads/usersguide/Using_N_etBeans55.pdf Last accessed 5th October 2011

Van ENgelen, R. A., (2003). "Pushing the SOAP envelope with Web Services for Scientific Computing", Proceedings of the International Conference on Web Services, Las Vegas, pp. 346 – 352, 2003.

Vartiainen, P., (2002). "On the Principles of Comparative Evaluation**"** *Evaluation* July 2002 *8: 359-371 sage online journals.*

Weber, S., (2004). The Success of Open Source. Harvard University Press 2004.

Williams, L. G., and Smith, C. U., (1998). "Performance Evaluation of Software Architectures," performance Engineering Services and Software Engineering Research.

Wheeler, D., (2005). "How to evaluate Open Source / Free Software (OSS/FS) Programs". URL http://www.dwheeler.com/oss_fs_eval.html. Retrieved on February 17, 2011.

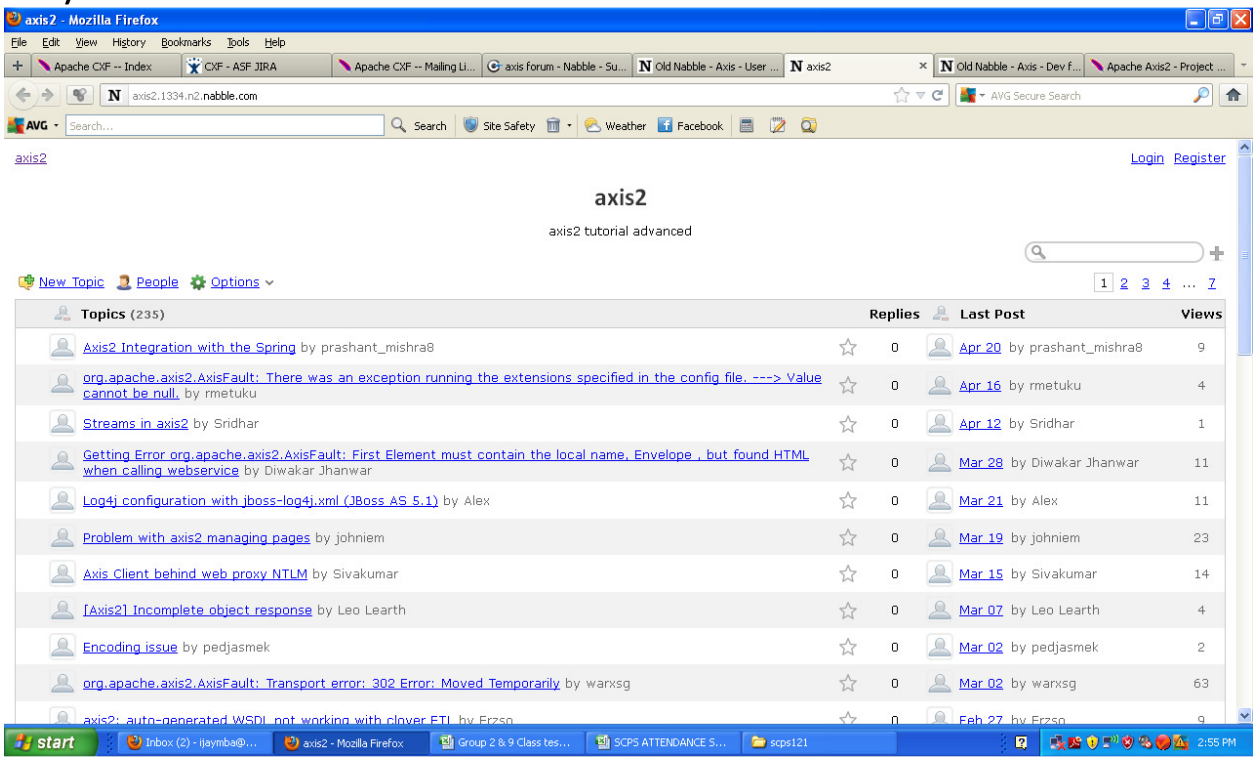
Ying, Y., Huang, Y., and Walker, D. W., (2004). "Using SOAP with Attachments for e-Science," in *UK e-Science All Hands Meeting*, Nottingham, UK, 2004.

About SOAPUI. http://www.soapui.org/About-SoapUI/features.html

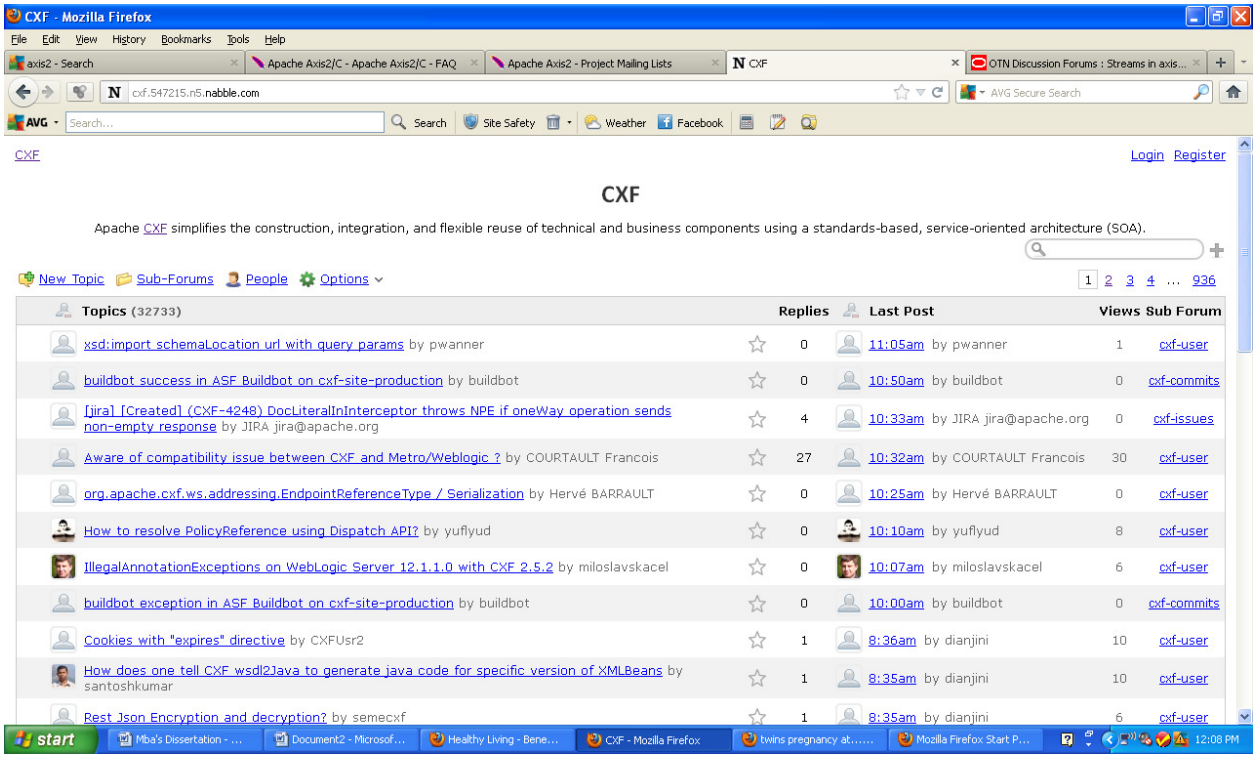
APPENDIX

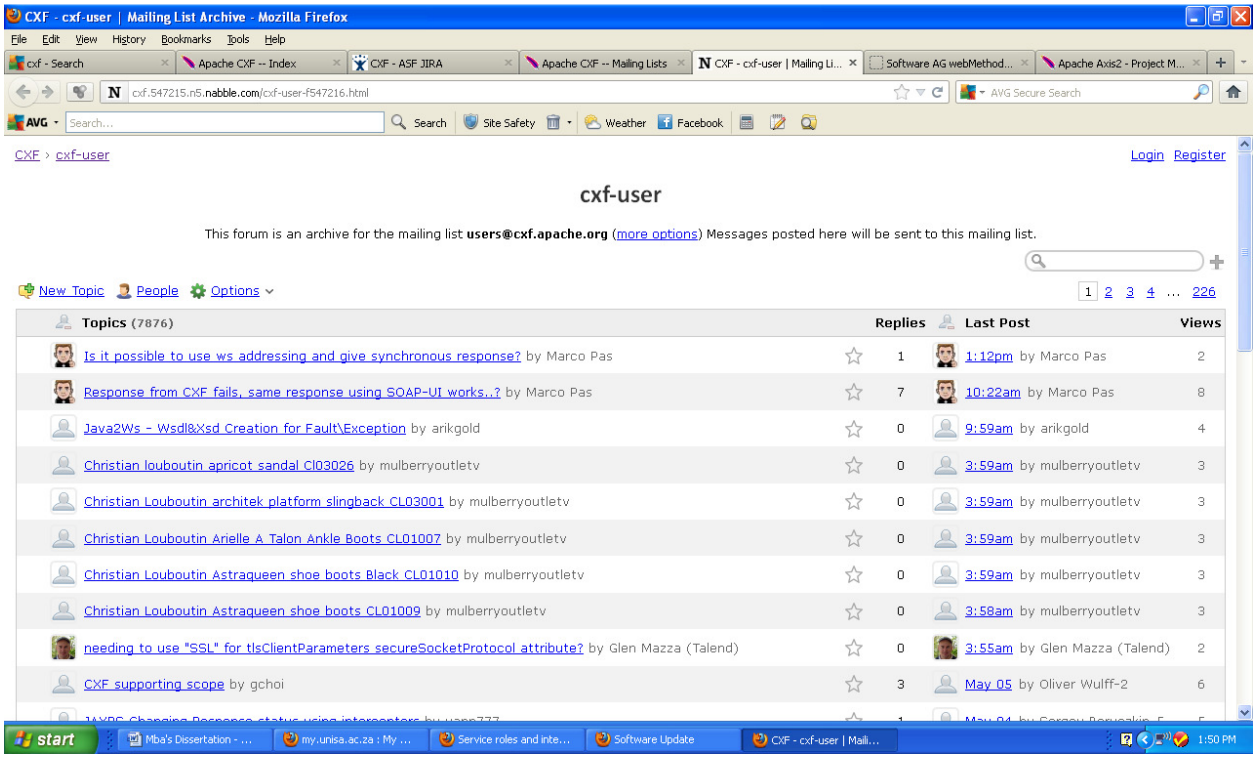
Appendix 1: Community Forums Screenshots

a) AXIS2

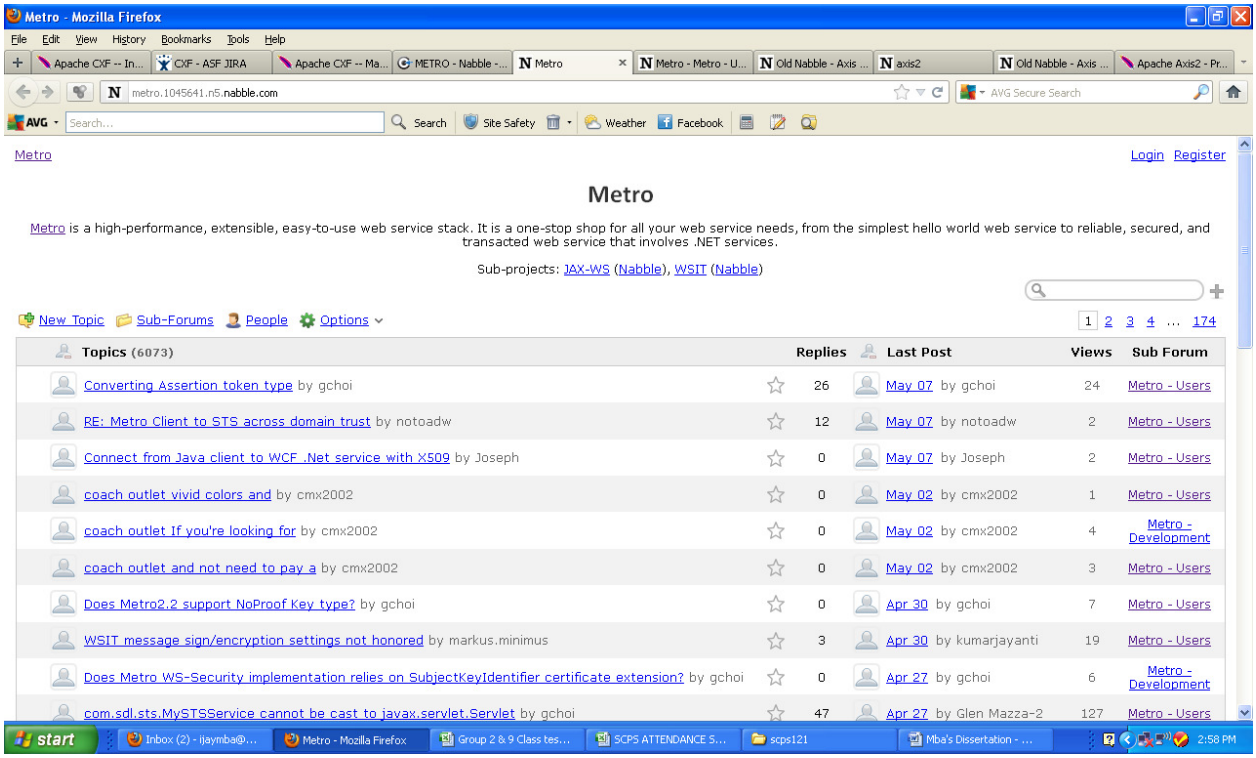


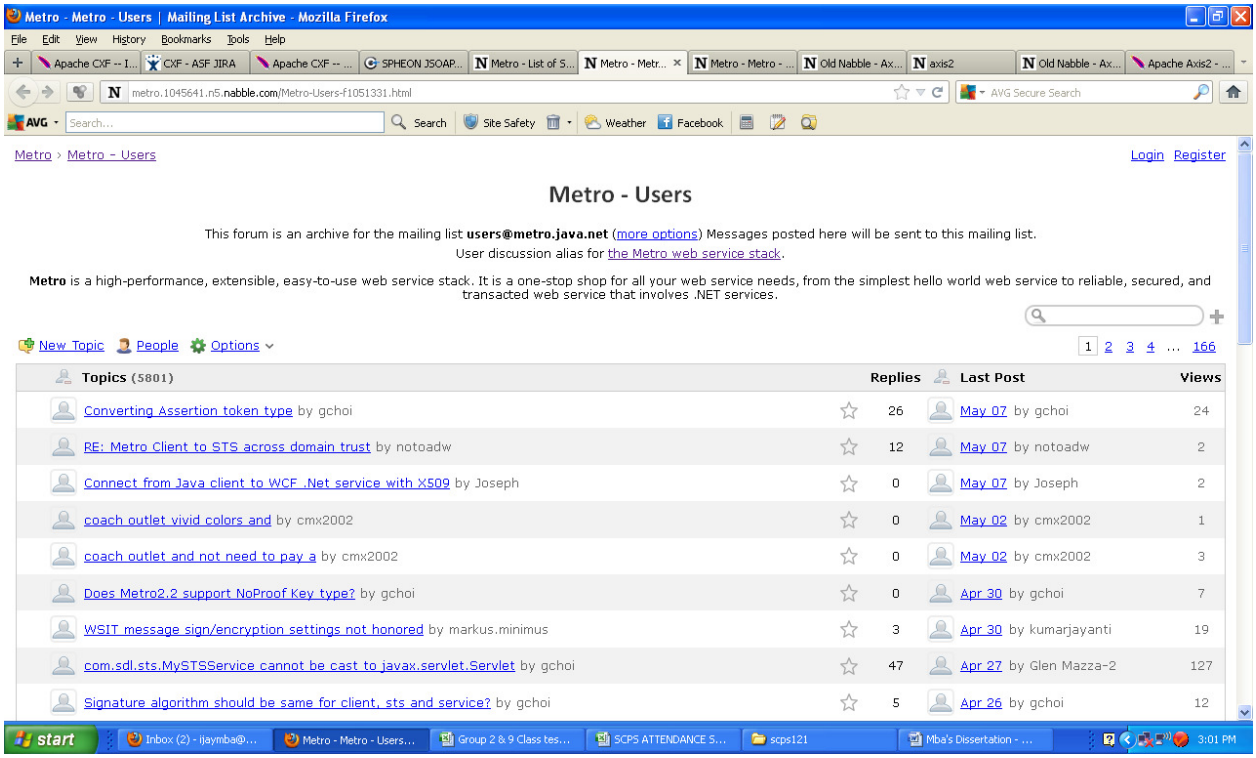
b) CXF


CXF - Mozilla Firefox
CXF
Login Register
CXF
Apache CXF simplifies the construction, integration, and flexible reuse of technical and business components using a standards-based, service-oriented architecture (SOA).
New Topic Sub-Forums People Options
1 2 3 4 ... 936
Topics (32733) Replies Last Post Views Sub Forum
xsd:import schemaLocation url with query params by pwanner 0 11:05am by pwanner 1 cxf-user
buildbot success in ASF Buildbot on cxf-site-production by buildbot 0 10:50am by buildbot 0 cxf-commits
[jira] [Created] (CXF-4248) DocLiteralInInterceptor throws NPE if oneWay operation sends non-empty response by JIRA jira@apache.org 4 10:33am by JIRA jira@apache.org 0 cxf-issues
Aware of compatibility issue between CXF and Metro/Weblogic ? by COURTAULT Francois 27 10:32am by COURTAULT Francois 30 cxf-user
org.apache.cxf.ws.addressing.EndpointReferenceType / Serialization by Hervé BARRAULT 0 10:25am by Hervé BARRAULT 0 cxf-user
How to resolve PolicyReference using Dispatch API? by yuflyud 0 10:10am by yuflyud 8 cxf-user
IllegalAnnotationExceptions on WebLogic Server 12.1.1.0 with CXF 2.5.2 by miloslavskacel 0 10:07am by miloslavskacel 6 cxf-user
buildbot exception in ASF Buildbot on cxf-site-production by buildbot 0 10:00am by buildbot 0 cxf-commits
Cookies with "expires" directive by CXFUsr2 1 8:36am by dianjini 10 cxf-user
How does one tell CXF wsdl2Java to generate java code for specific version of XMLBeans by santoshkumar 1 8:35am by dianjini 10 cxf-user
Rest Json Encryption and decryption? by semecxf 1 8:35am by dianjini 6 cxf-user

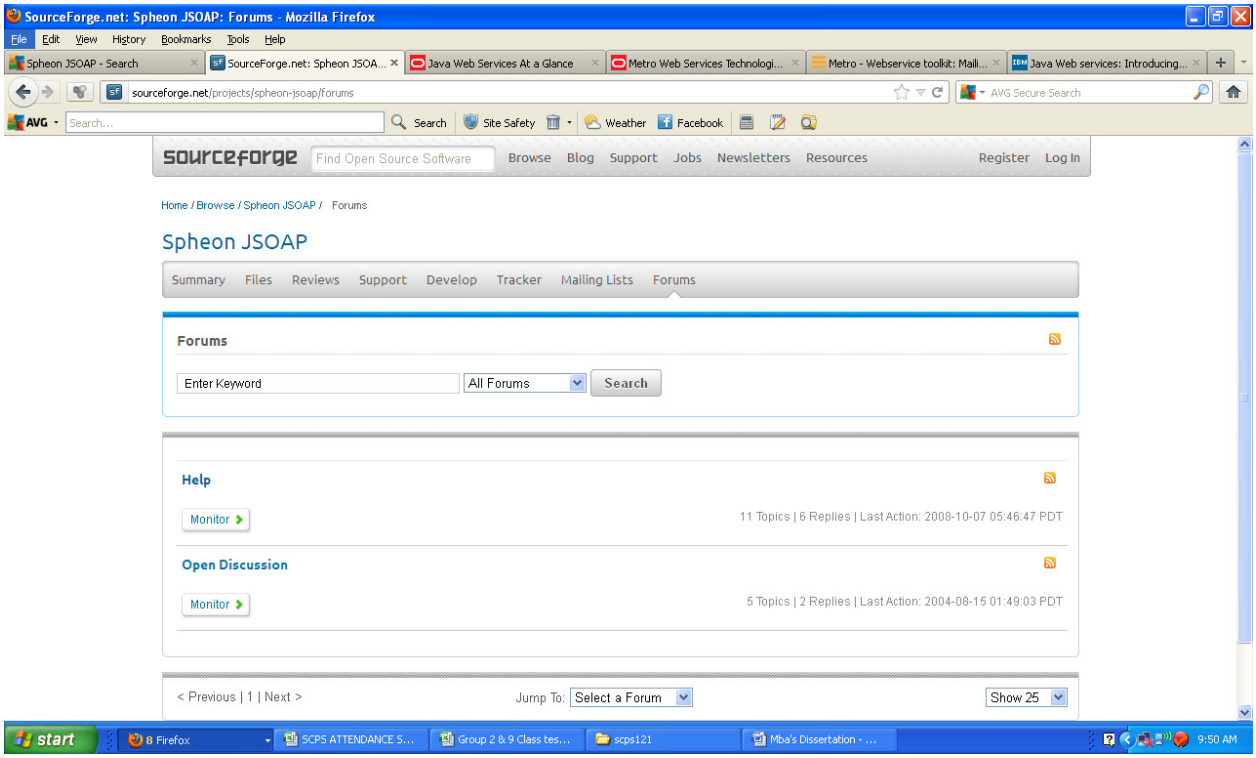

CXF - cxf-user | Mailing List Archive - Mozilla Firefox
CXF › cxf-user
Login Register
cxf-user
This forum is an archive for the mailing list users@cxf.apache.org (more options) Messages posted here will be sent to this mailing list.
New Topic People Options
1 2 3 4 ... 226
Topics (7876) Replies Last Post Views
Is it possible to use ws addressing and give synchronous response? by Marco Pas 1 1:12pm by Marco Pas 2
Response from CXF fails, same response using SOAP-UI works..? by Marco Pas 7 10:22am by Marco Pas 8
Java2Ws - Wsdl&Xsd Creation for Fault\Exception by arikgold 0 9:59am by arikgold 4
Christian louboutin apricot sandal Cl03026 by mulberryoutletv 0 3:59am by mulberryoutletv 3
Christian Louboutin architek platform slingback CL03001 by mulberryoutletv 0 3:59am by mulberryoutletv 3
Christian Louboutin Arielle A Talon Ankle Boots CL01007 by mulberryoutletv 0 3:59am by mulberryoutletv 3
Christian Louboutin Astraqueen shoe boots Black CL01010 by mulberryoutletv 0 3:59am by mulberryoutletv 3
Christian Louboutin Astraqueen shoe boots CL01009 by mulberryoutletv 0 3:58am by mulberryoutletv 3
needing to use "SSL" for tlsClientParameters secureSocketProtocol attribute? by Glen Mazza (Talend) 0 3:55am by Glen Mazza (Talend) 2
CXF supporting scope by gchoi 3 May 05 by Oliver Wulff-2 6

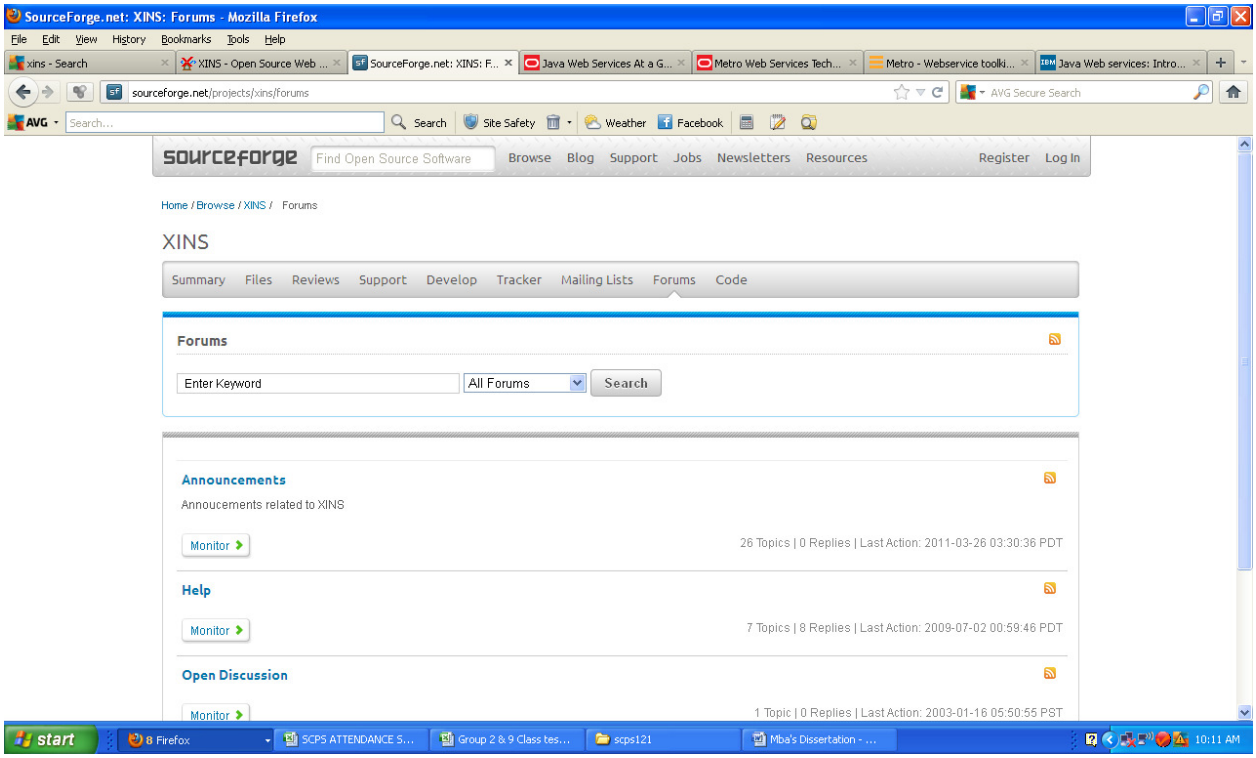
c) METRO


Metro - Mozilla Firefox
Metro
Login Register
Metro
Metro is a high-performance, extensible, easy-to-use web service stack. It is a one-stop shop for all your web service needs, from the simplest hello world web service to reliable, secured, and transacted web service that involves .NET services.
Sub-projects: JAX-WS (Nabble), WSIT (Nabble)
New Topic Sub-Forums People Options
1 2 3 4 ... 174
Topics (6073) Replies Last Post Views Sub Forum
Converting Assertion token type by gchoi 26 May 07 by gchoi 24 Metro - Users
RE: Metro Client to STS across domain trust by notoadw 12 May 07 by notoadw 2 Metro - Users
Connect from Java client to WCF .Net service with X509 by Joseph 0 May 07 by Joseph 2 Metro - Users
coach outlet vivid colors and by cmx2002 0 May 02 by cmx2002 1 Metro - Users
coach outlet If you're looking for by cmx2002 0 May 02 by cmx2002 4 Metro - Development
coach outlet and not need to pay a by cmx2002 0 May 02 by cmx2002 3 Metro - Users
Does Metro2.2 support NoProof Key type? by gchoi 0 Apr 30 by gchoi 7 Metro - Users
WSIT message sign/encryption settings not honored by markus.minimus 3 Apr 30 by kumarjayanti 19 Metro - Users
Does Metro WS-Security implementation relies on SubjectKeyIdentifier certificate extension? by gchoi 0 Apr 27 by gchoi 6 Metro - Development
com.sdl.sts.MySTSService cannot be cast to javax.servlet.Servlet by gchoi 47 Apr 27 by Glen Mazza-2 127 Metro - Users


Metro - Metro - Users | Mailing List Archive - Mozilla Firefox
Metro › Metro - Users
Login Register
Metro - Users
This forum is an archive for the mailing list users@metro.java.net (more options) Messages posted here will be sent to this mailing list.
User discussion alias for the Metro web service stack.
Metro is a high-performance, extensible, easy-to-use web service stack. It is a one-stop shop for all your web service needs, from the simplest hello world web service to reliable, secured, and transacted web service that involves .NET services.
New Topic People Options
1 2 3 4 ... 166
Topics (5801) Replies Last Post Views
Converting Assertion token type by gchoi 26 May 07 by gchoi 24
RE: Metro Client to STS across domain trust by notoadw 12 May 07 by notoadw 2
Connect from Java client to WCF .Net service with X509 by Joseph 0 May 07 by Joseph 2
coach outlet vivid colors and by cmx2002 0 May 02 by cmx2002 1
coach outlet and not need to pay a by cmx2002 0 May 02 by cmx2002 3
Does Metro2.2 support NoProof Key type? by gchoi 0 Apr 30 by gchoi 7
WSIT message sign/encryption settings not honored by markus.minimus 3 Apr 30 by kumarjayanti 19
com.sdl.sts.MySTSService cannot be cast to javax.servlet.Servlet by gchoi 47 Apr 27 by Glen Mazza-2 127
Signature algorithm should be same for client, sts and service? by gchoi 5 Apr 26 by gchoi 12

d) Spheon JSOAP





Appendix 2: Web Service Test Implementation

```
package org.uz.earth.ijmba.stockapp;

import javax.xml.bind.annotation.XmlElement;

/**
 *
 * @author ijmba
 */
public class OrderItem {

    private String productId = "";
    private int quantity = 1;

    public OrderItem() {
    }

    public OrderItem(String productId, int quantity) {
        this.productId = productId;
        this.quantity = quantity;
    }

    /**
     * @return the product
     */
    @XmlElement(name = "Product_Id", nillable = false, required = true, type
= String.class)
    public String getProductId() {
        return productId;
    }

    /**
     * @param product the product to set
     */
    public void setProductId(String productId) {
        this.productId = productId;
    }

    /**
     * @return the quantity
     */
    @XmlElement(name = "Quantity", nillable = false, required = true, type =
Integer.class)
    public int getQuantity() {
        return quantity;
    }

    /**
     * @param quantity the quantity to set
     */
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}
```

```
package org.uz.earth.ijmba.stockapp;

import javax.xml.bind.annotation.XmlElement;

/**
 *
 * @author ijmba
 */
public class Invoice {

    private double totalPrice;

    public Invoice() {
    }

    public Invoice(double totalPrice) {
        this.totalPrice = totalPrice;
    }

    /**
     * @return the totalPrice
     */
    @XmlElement(name = "Total_Price", nillable = false, required = true, type
= Double.class)
    public double getTotalPrice() {
        return totalPrice;
    }

    /**
     * @param totalPrice the totalPrice to set
     */
    public void setTotalPrice(double totalPrice) {
        this.totalPrice = totalPrice;
    }
}
```

```
package org.uz.earth.ijmba.stockapp.axis;

import java.text.MessageFormat;
import java.util.HashMap;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.jws.WebService;
import javax.servlet.http.HttpServletRequest;
import org.apache.axis2.transport.http.HTTPConstants;
import org.uz.earth.ijmba.stockapp.Invoice;
import org.uz.earth.ijmba.stockapp.OrderItem;
import org.uz.earth.ijmba.stockapp.StockOrder;

/**
 *
 * @author ijmba
 */
@WebService(endpointInterface = "org.uz.earth.ijmba.stockapp.StockOrder")
public class StockOrderImpl implements StockOrder {

    private final static HashMap<String, Double> stockTable = new
HashMap<String, Double>();
    private final static Logger logger =
Logger.getLogger(StockOrderImpl.class.getName());

    static {
        stockTable.put("prod1", 20.00);
        stockTable.put("prod2", 10.00);
        stockTable.put("prod3", 40.00);
        stockTable.put("prod4", 5.00);
    }

    @Override
    public Invoice orderStock(OrderItem[] orderItems) {
        long startOperation = System.nanoTime();
        HttpServletRequest request = findRequest();
        if (request != null) {
            logger.log(Level.INFO, MessageFormat.format("Processing Operation
({0})...", "orderStock"));
        }

        double subTotal = performStockOrder(orderItems);
        Invoice invoice = new Invoice(subTotal);
        logger.log(Level.INFO, MessageFormat.format("Operation ({0}) Ended",
"orderStock"));

        if (request != null) {
            long startRequest = (Long) request.getAttribute("startRequest");
            long endOperation = System.nanoTime();
            request.setAttribute("endOperation", endOperation);
            logger.log(Level.INFO, MessageFormat.format("UNMARSHAL_TIME:
{0}ns", (startOperation - startRequest)));
        }
        return invoice;
    }
```

```
    private double performStockOrder(OrderItem[] orderItems) {
```

```
  double subTotal = 0d, price = 0d;
        for (OrderItem orderItem : orderItems) {
            price = stockTable.get(orderItem.getProductId()) == null ? 0d :
stockTable.get(orderItem.getProductId());
            subTotal += price * orderItem.getQuantity();
        }
        return subTotal;
    }

    private HttpServletRequest findRequest() {
        org.apache.axis2.context.MessageContext context =
org.apache.axis2.context.MessageContext.getCurrentMessageContext();
        return context == null ? null : (HttpServletRequest)
context.getProperty(HTTPConstants.MC_HTTP_SERVLETREQUEST);
    }
}

package org.uz.earth.ijmba.stockapp.axis;

import java.text.MessageFormat;
import java.util.HashMap;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.jws.WebService;
import javax.servlet.http.HttpServletRequest;
import org.apache.axis2.transport.http.HTTPConstants;
import org.uz.earth.ijmba.stockapp.Invoice;
import org.uz.earth.ijmba.stockapp.OrderItem;
import org.uz.earth.ijmba.stockapp.StockOrder;

/**
 *
 * @author ijmba
 */
@WebService(endpointInterface = "org.uz.earth.ijmba.stockapp.StockOrder")
public class StockOrderImpl implements StockOrder {

    private final static HashMap<String, Double> stockTable = new
HashMap<String, Double>();
    private final static Logger logger =
Logger.getLogger(StockOrderImpl.class.getName());

    static {
        stockTable.put("prod1", 20.00);
        stockTable.put("prod2", 10.00);
        stockTable.put("prod3", 40.00);
        stockTable.put("prod4", 5.00);
    }

    @Override
    public Invoice orderStock(OrderItem[] orderItems) {
```

```
        long startOperation = System.nanoTime();
        HttpServletRequest request = findRequest();
        if (request != null) {
            logger.log(Level.INFO, MessageFormat.format("Processing Operation
({0})...", "orderStock"));
        }

        double subTotal = performStockOrder(orderItems);
        Invoice invoice = new Invoice(subTotal);
        logger.log(Level.INFO, MessageFormat.format("Operation ({0}) Ended",
"orderStock"));

        if (request != null) {
            long startRequest = (Long) request.getAttribute("startRequest");
            long endOperation = System.nanoTime();
            request.setAttribute("endOperation", endOperation);
            logger.log(Level.INFO, MessageFormat.format("UNMARSHAL_TIME:
{0}ns", (startOperation - startRequest)));
        }
        return invoice;
    }

    private double performStockOrder(OrderItem[] orderItems) {
        double subTotal = 0d, price = 0d;
        for (OrderItem orderItem : orderItems) {
            price = stockTable.get(orderItem.getProductId()) == null ? 0d :
stockTable.get(orderItem.getProductId());
            subTotal += price * orderItem.getQuantity();
        }
        return subTotal;
    }

    private HttpServletRequest findRequest() {
        org.apache.axis2.context.MessageContext context =
org.apache.axis2.context.MessageContext.getCurrentMessageContext();
        return context == null ? null : (HttpServletRequest)
context.getProperty(HTTPConstants.MC_HTTP_SERVLETREQUEST);
    }
}
```