# Exposing Resources as Web services: a Performance Oriented Approach

by

Ravishankar Kanagasundaram, B. Eng.

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

**Master of Applied Science**

**in**

**Electrical and Computer Engineering**

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada K1S 5B6

The undersigned recommend to
the Faculty of Graduate and Postdoctoral Affairs
the acceptance of the thesis

**Exposing Resources as Web services: a Performance Oriented Approach**

submitted by

Ravishankar Kanagasundaram, B. Eng.

in partial fulfillment of the requirements for the degree of

Master of Applied Science in

Electrical and Computer Engineering

_____

Shikharesh Majumdar

Thesis Supervisor, Department of Systems and Computer Engineering

_____

Howard Schwartz

Chair, Department of Systems and Computer Engineering

Carleton University

September 2012

## Abstract

Exposing resources as Web services provides inter-operability and enables various clients that are implemented by using diverse technologies to access different resources, such as computing and database resources. Sharing resources through Web services presents the client with access to a shared resource regardless of the client's programming language or (operating system). This research focuses on exposing computing and database resources as Web services so that various clients can access each resource through a uniform interface. Two different Web service technologies: the RESTful Web service and the SOAP-based Web service are used for exposing computing and database resources as Web services. Based on prototyping and measurement, a performance analysis of the two technologies, the RESTful Web Service and the SOAP-based Web service used for exposing resources as Web services is reported. A novel Hybrid Web service that combines the advantages of both RESTful and SOAP-based Web services is proposed and analyzed.

## Acknowledgements

First of all, I would like to express my sincere thanks to my thesis supervisor, Professor Shikharesh Majumdar, for all his guidance, and continuous support, encouragement and help throughout the completion of this thesis.

I would like to thank Cistel Technology and the Ontario Centers of Excellence for providing the financial support for this research.

I would also like to thank Orlando and Norman for their help on getting me started with Web service tools. Thanks are due to SriPrasanna, Manomohan, Denis, Michael and Junior for proofreading my thesis document.

Finally, I want to thank my wife Tharsica, my parents, my sister and her family, relatives and friends for their support and encouragement throughout my Master's study.

## Table of Contents

## List of Tables

## List of Figures

## List of Symbols and Acronyms

| | |
|---|---|
| **λ** | Arrival rate |
| ***p*** | Probability of invoking the RESTful Web Service |
| **API** | Application Programming Interface |
| **CHWS** | Computing Resource exposed as a Hybrid Web Service |
| **CHWSC** | Computing Resource exposed as a Hybrid Web Service Client |
| **CPU** | Central Processing Unit |
| **CWS** | Computing resource exposed as a Web Service |
| **CWSR** | Computing resource exposed as a Web Service RESTful |
| **CWSRC** | Computing resource exposed as a Web Service RESTful Client |
| **CWSS** | Computing resource exposed as a Web Service SOAP-based |
| **CWSSC** | Computing resource exposed as a Web Service SOAP-based Client |
| **DBHWS** | Database resource exposed as a Hybrid Web Service |
| **DBHWSC** | Database resource exposed as a Hybrid Web Service Client |
| **DBHWSR** | Database resource exposed as a Hybrid Web Service RESTful |
| **DBHWSS** | Database resource exposed as a Hybrid Web Service SOAP-based |
| **DBMS** | Database Management Systems |
| **DBWS** | Database resource exposed as a Web Service |
| **DBWSR** | Database resource exposed as a Web Service RESTful |
| **DBWSRC** | Database resource exposed as a Web Service RESTful Client |
| **DBWSS** | Database resource exposed as a Web Service SOAP-based |
| **DBWSSC** | Database resource exposed as a Web Service SOAP-based Client |
| **FIFO** | First In First Out |

| | |
|---|---|
| **GHz** | Gigahertz |
| **HPC** | High Performance Computing |
| **HTML** | Hyper Text Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **IDE** | Integrated Development Environment |
| **JAX-RS** | Java API for RESTful Web Services |
| **JAX-WS** | Java API for SOAP-based Web Service |
| **JDBC** | Java DataBase Connectivity |
| **JSP** | Java Server Page |
| **NEWT** | National Energy Research Scientific Computing Web Service Toolkit |
| **OS** | Operating System |
| **RAM** | Random Access Memory |
| **REST** | Representational State Transfer |
| **RIDDL** | RESTful Interface Definition and Declaration Language |
| **RPC** | Remote Procedure Call |
| **RSS** | Really Simple Syndication |
| **SOAP** | Simple Object Access Protocol |
| **SPECTS** | Symposium on Performance Evaluation of Computer and Telecommunication Systems |
| **SQL** | Structured Query Language |
| **StudentDB** | Student Database |
| **UDDI** | Universal Description, Discovery, and Integration |
| **URI** | Uniform Resource Identifier |

| | |
|---|---|
| **WADL** | Web Application Description Language |
| **WS** | Web Service |
| **WSART** | Web Service Average Response Time |
| **WSDL** | Web Service Description Language |
| **WSES** | Web Service Evaluation System |
| **WSRT** | Web Service Response Time |
| **WWW** | World Wide Web |
| **XML** | Extensible Markup Language |

# Chapter 1 : Introduction

A Web service is a software system that enables interactions between a client and the server application that may communicate over the Internet [37]. The Web service system supports interoperable communication over a Web. Web services are built based on Web service standards by using which Web service providers and Web service clients agree on a common Web service interface. Web services are language and platform independent: clients developed by using various languages and running on top of different platforms can communicate with the same Web service.

According to [9] Web services have the following benefits. Web services are said to provide loose coupling. A Web service application may include different services and that are each service is independent of each other. Modifying one of the services will not affect the other services. Each Web service is built based on Web service standard and therefore Web services are easy to integrate into a system.

Resources can be exposed as Web services. The advantages of exposing resources as Web services provide inter-operability and enable various clients to access the shared resources. A shared resource allows the right to use by multiple and various types of clients. Client A's characteristic may differ from client B's characteristics. It is therefore important to have user friendly uniform interfaces for each shared resource. One of the ways of providing inter-operability for sharing resources is through Web services. The Web service presents the client with a single pre-defined interface for accessing a shared resource regardless of the client's programming language or

(operating system). Resources can be of various types including computing and database resources.

There are other challenges in exposing a resource as a Web service including: the requirement of a large bandwidth for communicating between a remote client and the shared resource [14]; the requirement of programming techniques to handle multiple concurrent requests for the same resources. The bandwidth problem can be alleviated by providing a high speed network infrastructure for remote clients. The concurrent requests to resource problem are overcome by using appropriate distributed system technologies including cloud and grid technologies that provide resource reservation and resources allocation. This thesis focuses on exposing computing and database resources as Web services.

## 1.1 Exposing a Computing Resource as a Web Service

Figure 1.1 shows an example of exposing a computing resource as a Web service. External client uses a Web browser (e.g. FireFox, Internet Explorer or Google Chrome) to invoke the computing resource exposed as a Web service. The computing system could be running different operating systems such as Windows and Linux. An external client, for example, can execute any Windows "cmd" from the computing resource Web service. For listing the content of the directory, the external client types "dir" then submits the button. The Web service executes the external client's request and sends the results back to the external client through the Web browser. Various other operating system commands can be executed on the computing resource in a similar manner by invoking the Web service. Chapter 3 has a detailed description of the design of the system exposing a computing resource as a Web service.

There are many advantages of exposing a computing resource as Web service such as:

- An external client can be remote or local to the Web service
- The external client can invoke the Web service from any platform
- The external client does not need to have all the applications and software on the external client's platform; instead the client invokes the Web service and uses or runs the program available on the computing resource exposed as a Web service.

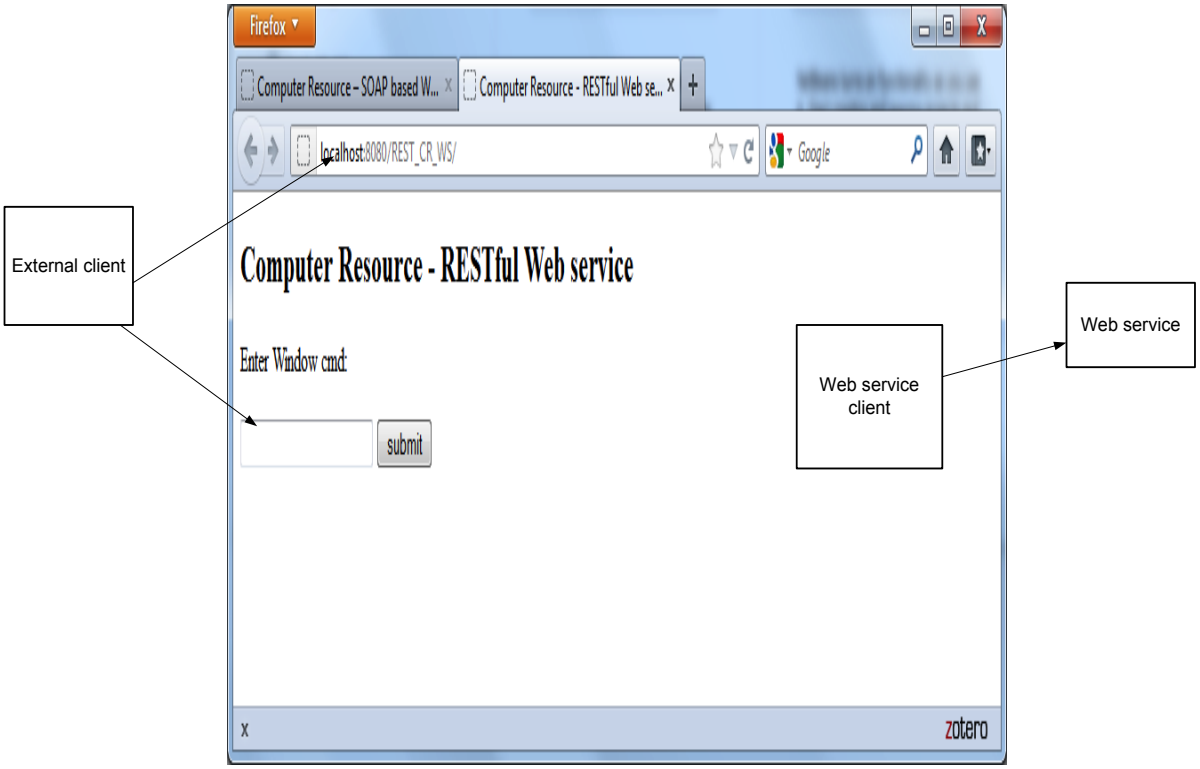


**Figure 1.1: Exposing computing resource as a Web service**

## 1.2 Exposing a Database Resources as a Web Service

Figure 1.2 shows an example of exposing a database resource as a Web service. As described in Section 1.1 the external client uses a Web browser to invoke the

database resource exposed as a Web service. In this example, a student database resource is exposed as a Web service and the external client can insert a student record or read, update or delete the student record. When the external client wants to add a new student record, the external client types the student information into the Web browser and then submits the request by clicking the "submit" button. Once the external client submits the request on the database resource Web service, the Web service executes the external client's request and sends the results back to the external client through the Web browser. Note that external clients can be running on a local or a remote system and use diverse platform. A detailed description of the design of a system exposing a database resource as Web service is presented in Chapter 3.

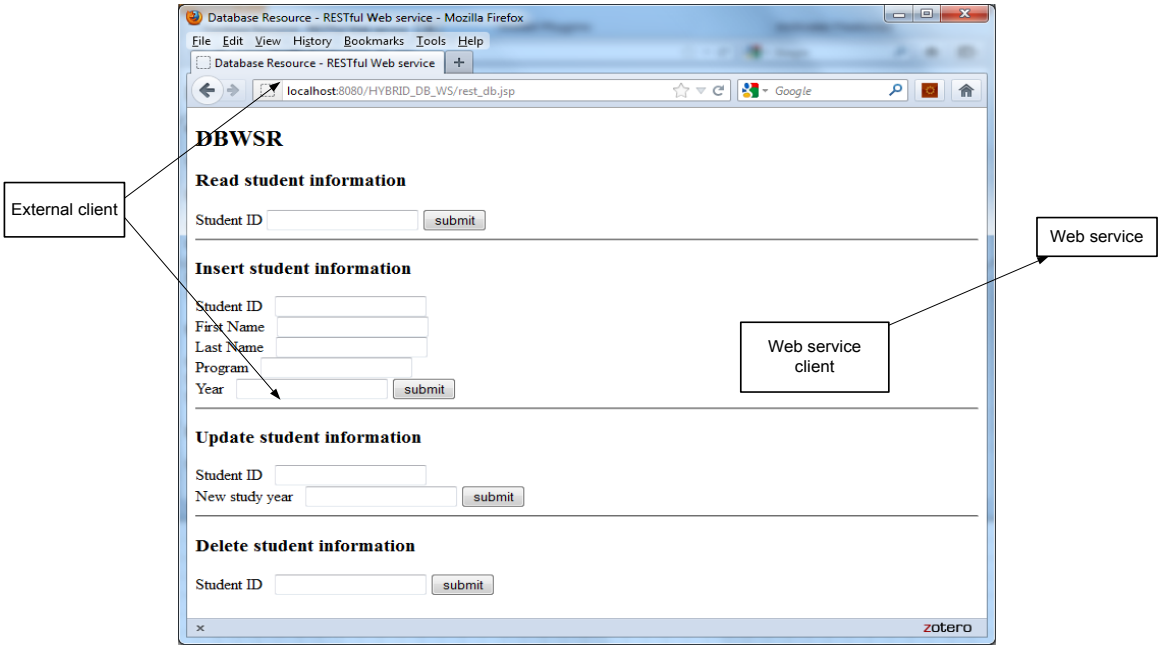


**Figure 1.2: Exposing database resource as a Web service**

## 1.3 Motivations for the Thesis

Carleton University and Cistel Technology are developing a resource management middleware for unifying resources for a distributed environment for bridge infrastructure management. This research project is supported by the Ontario Centers of Excellence. The system needs to make several types of resources (e.g., computer and database) and data analysis tools available to a user on demand. Since diverse users (clients) can be using different programming languages and platforms providing inter-operability is crucial in the context of such distributed systems that unify diverse resources for a user community. One way of providing inter-operability is by exposing resources and tools as Web services. This thesis investigates performance aware approaches for exposing resources as Web services.

Resources can be exposed as Web services using different Web services technologies such as RESTful Web service or SOAP-based Web service. The performance of a Web service is important in the contents of exposing resources as Web services. The RESTful Web service is a lightweight Web service and is expected to have a better performance in comparison of a SOAP-based Web service, which uses the extra XML markup required by SOAP messages.

Web services are often invoked by clients over the Internet. During the communication between the client and the Web service, messages get exchanged over the Internet. The messages can be categorised into two groups: sensitive information messages and non sensitive information messages. For example, updating the student information or bank transaction gives rise to sensitive information messages; whereas listing attractive places in Ottawa typically give rise to non-sensitive information

messages. The message containing the sensitive information needs to be transmitted using a secure channel. This sensitive message requires CPU intensive tasks for encrypting and decrypting message. A SOAP-based Web service supports a number of standards on security, reliable messaging and atomic transactions that are currently not available with RESTful Web service. Thus, a SOAP-based Web service often becomes necessary in the context of such message carrying sensitive information.

This research focuses on analyzing the performance impact of these two Web services technologies and is motivated by the following questions:

- What is the difference in performance achieved by SOAP-based Web service and RESTful Web service technologies for various operations performed on resources exposed as Web services?
- What are the system and workload parameters for which a RESTful Web service technology provides a significant performance benefit over SOAP-based Web service?
- How can we combine the advantages of both the RESTful (lightweight) and SOAP (support for additional standards) based technologies in an environment where carrying (both sensitive information and non-sensitive information) messages are used?

## 1.4 Goals of the Thesis

One of the goals of this thesis is to evaluate the performance for resources exposed as Web services. The RESTful Web service is expected to have performance advantages over and SOAP-based Web service, while SOAP-based Web service provides support for

Web service standards such as security, reliable messaging and atomic transactions that are currently not available with RESTful Web service. Prototype systems based on both the technologies are built and are used with various workloads. A rigorous performance analysis of both the RESTful Web service and the SOAP-based Web service is made.

The impact of various workload parameters on the relative performance of both types of Web services is presented in the thesis. Another important goal of this thesis is to devise a technique for combining the RESTful Web service and the SOAP-based Web service technologies and effectively utilize the advantages from both types of Web services.

## 1.5 Contributions of the Thesis

The main contributions of the thesis include:

- Techniques for exposing and invoking of computing resources and database resources as Web services are described.
    - Results of a thorough performance analysis of the two Web service technologies through prototyping and measurement are reported.
- Various insights into systems behaviour and performance for various combinations of systems and workload parameters are presented.
- A novel technique called Hybrid Web service that combines the RESTful Web service and the SOAP-based Web service technologies is proposed and a description of a prototype implementation is presented.
- The performance advantage of Hybrid Web service is demonstrated through measurements made on the prototype.

A paper based on the research results is published in the proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS) 2012 [10].

## 1.6 Outline of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 gives background information on Web services and discusses related work. Chapter 3 explains techniques for exposing and invoking of computing resources and database resources as Web services in more detail. Chapter 4 describes the design and implementation a novel Hybrid Web service technique. Chapter 5 presents and discusses the results of the performance evaluation for the Web service technologies investigated in the thesis. Finally, Chapter 6 concludes the thesis and provides possible directions for future research.

# Chapter 2 : Background and Related Work

## 2.1 Web Services

Web service is a software system which provides service over the Internet, regardless of operating systems or programming languages [20]. Web service gains more attention for as a distributed approach of services or application integration over the Web [43]. A Web service is capable of performing various ranges of functions such as providing services to a simple request or providing services to complicated business processes. According to [8] Web services are noted to have the following characteristics:

1. Web services are self contained.
2. Web services can be published, discovered and invoked over the Internet.

### 2.1.1 Web Service Components

According to [8] a Web service comprises the following components:

1. Web service registry which acts as a broker for Web services.
2. Web service provider that publishes services on the Web service registry.
3. Web service user or client who discovers the Web services in the registry and invokes the Web services provided by the Web service provider.

The next section describes the relationship among these components.

### 2.1.2 Web Service Architecture

Figure 2.1 shows the architecture for a Web service based system. A Web service provider needs to create a Web service and then publishes the Web service information on the Web service registry. Web service client uses the Web service registry to find out

the Web service information. Finally, the Web service client invokes the Web service available at the Web service provider [9].

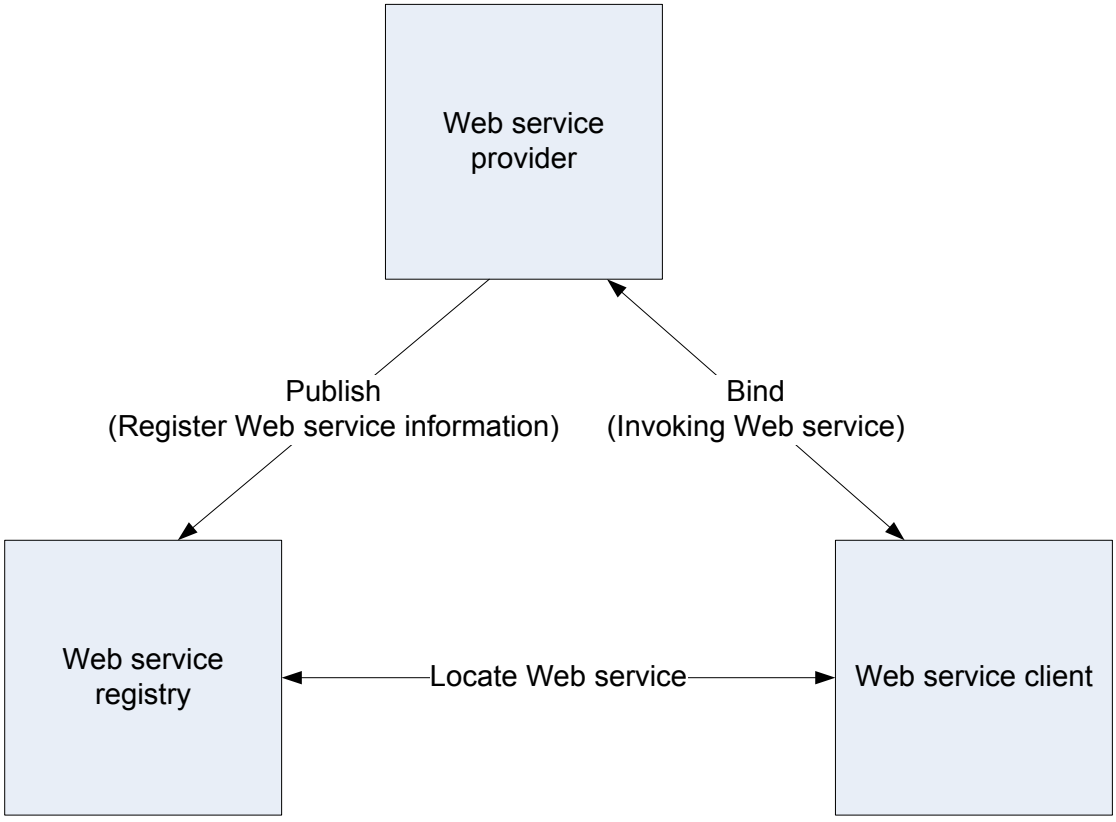


**Figure 2.1: Architecture for Web service (based on [9])**

### 2.1.3 Web Services and Web 2.0

Exposing resources as Web services are relevant to Web 2.0 technologies. The Web 2.0 characteristics such as interactive operations, rich user experience and lightweight user interface are well suited for exposing resources as Web services.

Web 2.0 is the next generation of World Wide Web (WWW) and Web 2.0 term was introduced by Tim O'Reilly during the Media conference in 2004 [20]. Since then the WWW has gone through many innovating changes including [20]:

- changes in application design such as Web layout and components look and feel
- changes in development tools and technologies, such as using Java scripts, Flash technologies, Really Simple Syndication (RSS) and Mashups
- changes in interaction between service providers and service consumer, such as introduction of social networks (Facebook) and support for Web based collaborative authoring (Wiki and Google document).

The following sub sections describe Web 2.0 characteristics, technologies.

### 2.1.4 The Web 2.0 Characteristics

Web 2.0 is not just a new version of the previous Web 1.0. According to [20] Web 2.0 has seven core principles:

1. The organizations are shifted to a new way of making revenue based on the service provided to the customer instead of selling the products in traditional applications. For example, Amazon Web service is charging its client based on client usage [3].
2. Control over unique, hard-to-recreate data sources that get richer as more people use them. Peer to Peer movement and decentralization is one example is using BitTorrent where clients also become a server. Therefore, the services get better and better as more people use them.
3. Trusting users as co-developers: This method of development is used in many open source products. The users contribute through feedback and comments to the products; this adds value to the products. Thus, the company makes better products through peer collaboration.

4. Harnessing collective intelligence: The Web site provides collaborative services. For example, Amazon Web service which sells products online also includes customers' contributions for product review, comments, online profiles and blogging.
5. The term "long tail" is explained as "the collective power of various small sites that makes up the bulk of the Web's content" in [43]. Originations make revenue based on online advertisement. For example Google AdSence which enables advertisement placement on virtually any Web page based on the user search categories.
6. Software above the level of a single device: Web application software needs to work regardless of client devices and client platforms. Thus, Web applications are able to deliver the same quality of performance on different devices and platforms.
7. Lightweight user interfaces, development models, and business models: The Web 2.0 application's interfaces are lightweight (do not have heavy graphics). The developments models of Web 2.0 is continued to evolve with new features that include simultaneous develop, test and release. Furthermore, the feedback from the user is considered in the development model [20].

### 2.1.5 The Web 2.0 Technologies: Services and Tools

Web 2.0 consists of several new Web technologies. The following sub sections describe them.

#### 2.1.5.1 RSS, Wikis and Mashups

RSS is a Web feed format used for syndicating the content of Web sites. RSS consists of summarized information and the information of the metadata. RSS informs the clients or subscribers about the updates of Web sites or blogs they show interest in.

According to [42] Wiki is a simple Web based application which allows users to create or update content using any Web browser. A Wiki is capable of collaborative authoring for groups of people working on a centralized document. Google document also has the similar characteristics as a Wiki. The advantages of using a Wiki are:

1. Asynchronous contribution by a group of people (regardless of their geographical locations).
2. Higher communication efficiency and productivity (Emails need not be exchanged and all updates are visible to all in the group) [18].

Mashups in the context of Web are Web sites, which include information and services from various sources on the Web. One of the examples would be Google map that is used by other Web sites. Developing applications by using a mashup is simpler and quicker than developing a Web application form scratch.

## 2.2 SOAP-based Web Services

A SOAP-based Web service is a traditional Web service that uses Web service Description Language (WSDL). The WSDL is used to describe network services in an XML format and how to access them [9]. The SOAP standard comprises two components:

1. The SOAP envelope is the root element of a SOAP message and the SOAP envelope defines a framework for expressing what is in the message and who should receive the message [38].
2. The SOAP protocol binding - SOAP messages use different of underlying protocols and the SOAP protocol binding framework defines general rules for protocol binding specifications. For example, it declares the features provides by a binding and describes the handlings of potential failures within the binding [38].

### 2.2.1 SOAP-based Web Service Components

SOAP-based Web service is made with following components:

1. SOAP-based Web Service provider
2. Client who invokes SOAP-based Web service
3. Universal Description, Discovery and Integration (UDDI) based registry
4. WSDL
5. SOAP Messages and XML

The following section describes the relationship among these components.

### 2.2.2 SOAP-based Web Service Architecture

Figure 2.2 shows the various important operations performed on for a SOAP-based Web service.

1. SOAP-based Web service registers its Web service information on UDDI-based registry.

2. SOAP-based Web Service's client searches the UDDI registry to locate the Web service.
3. SOAP-based Web Service's client generates a SOAP request according to the WSDL document. The WSDL provides information on how to use a Web service, descriptions of Web service methods and the binding information [38]. The SOAP-based Web service provider generates the WSDL document. Netbeans Web application project automatically generates WSDL document for SOAP-based Web service [22].
4. SOAP-based Web Service's client sends a SOAP request to SOAP-based Web service.
5. SOAP-based Web service executes SOAP-based Web Service's client request.
6. SOAP-based Web service generates a SOAP response accordingly. The SOAP header contains the destination as SOAP-based Web service's client and the SOAP body contains the executed results from previous step.
7. SOAP-based Web service sends SOAP response to SOAP-based Web service's client.

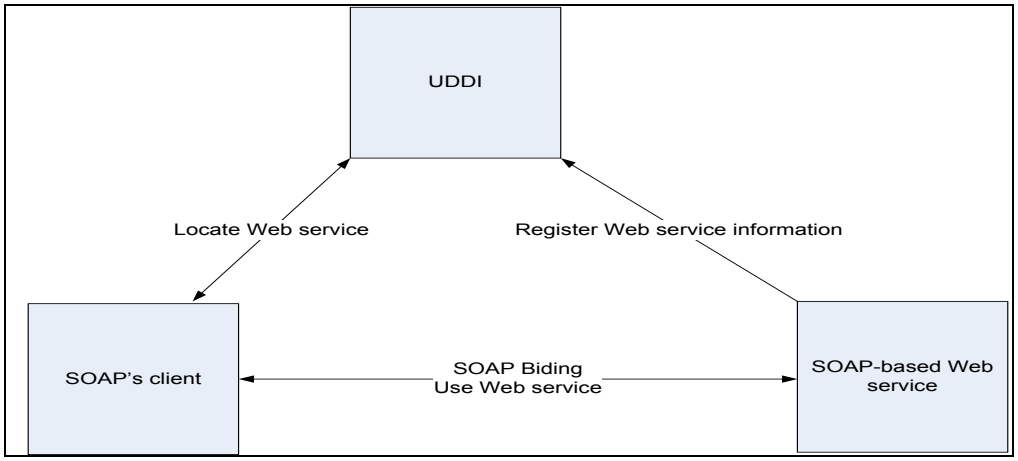


**Figure 2.2: Architecture for SOAP-based Web service (based on [8])**

## 2.3 RESTful Web Services

REST stands for Representational State Transfer [23]. With RESTful Web services, a resource is identified by a Uniform Resource Identifier (URI). The resources are manipulated by using four HTTP methods: GET, PUT, POST and DELETE. Thus, the main advantages of the RESTful Web service technology include:

1. Addressability - The resource is accessed by using its URI and there is no need to have a separate resource discovery and location mechanism such as a UDDI based registry in a conventional SOAP-based Web service system.

2. Statelessness - Each client request is independent and unrelated to the previous request and contains all the required information for the servers to process the request.

3. Simple and uniform interface - Only four HTTP methods GET, PUT, DELETE and POST are required for manipulating the resources.

4. Multi-support for access - The contents of a resource can be accessed in several formats such as plain text, XML, and HTML [20].

RESTful Web services are used in popular Web sites such as Google, Yahoo, and Amazon [14].

### 2.3.1 RESTful Web Service Components

A RESTful Web service includes the following components:

1. RESTful Web service provider
2. RESTful Web service client

3. Resource
4. URI
5. Uniform interface
6. HTTP

The next section describes the relationship among these RESTful Web service components.

### 2.3.2 RESTful Web service Architecture

The architecture for a RESTful Web service system is shown in Figure 2.3. According to [7] a RESTful resource has two types of states one for the Web services and one for the client:

1. The state on the Web service called resource state that holds information about the resource such as how recourses are organized, and how to access the resources.
2. The state on the client side called application state that holds the information about the URI that the client uses to invoke the Web service.

A RESTful Web service client invokes the RESTful Web service by using the HTTP communication protocol and uses any of the methods: GET, PUT, POST and DELETE to manipulate the resource in the RESTful Web service.

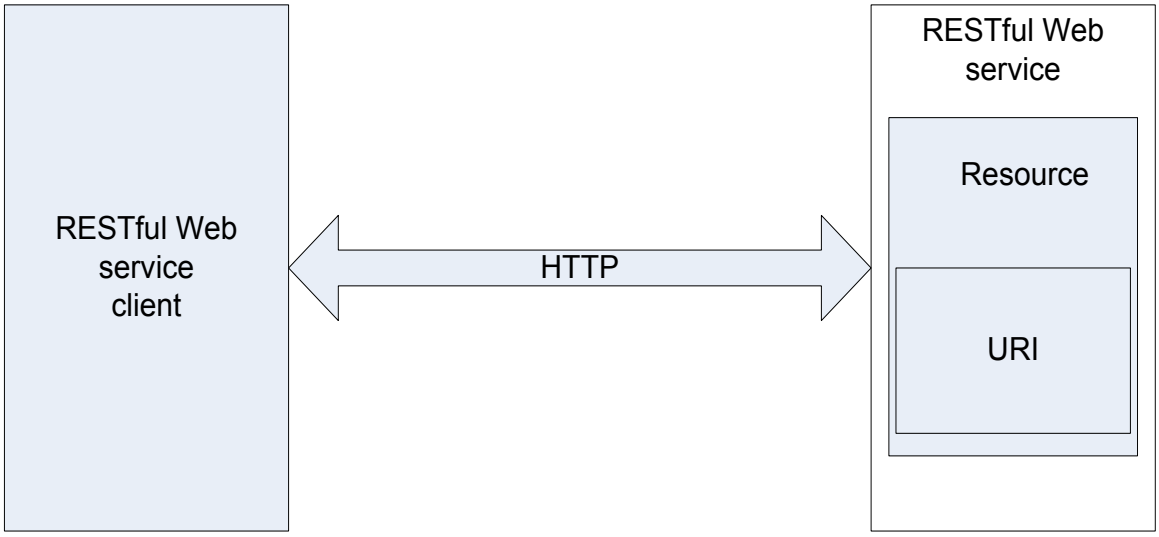


**Figure 2.3: Architecture for RESTful Web service**

## 2.4 Related Research

The RESTful approach for exposing a resource as a Web service is lightweight and is easier to develop and consume. On the other hand, a SOAP-based approach leads to more overheads and the learning curve for the SOAP-based Web service is steep. Both RESTful and SOAP-based Web services are active research areas. A representative set of related work is presented.

### 2.4.1 Exposing Resources as Web Services

In [14] authors describe techniques of exposing resources as Web services. A computing resource is exposed as a Web service using two different Web service technologies: RESTful Web service and SOAP-based Web service. Authors presents the performance comparison of computing resources as Web service one based on RESTful Web service and other one based on SOAP-based Web service. The performance results demonstrate that computing resource exposed with RESTful Web service has better performance over the computing resource exposed with SOAP-based Web service

technology. This thesis also investigated exposing computing and database resources as Web services using various types of Web services. A thorough performance analysis of the two Web service technologies (SOAP-based Web service and RESTful Web service) through prototyping and measurement are reported in this thesis.

Scientific computing continues to move toward the Web and making the High Performance Computing (HPC) available through the Web is useful for scientist. Authors introduce a new Web Toolkit called the National Energy Research Scientific Computing Web service Toolkit (NEWT) to access the HPC resources through a Web browser environment [5]. By using NEWT the scientists and programmers develop Web applications for HPC. The RESTful Web service technology is used to build NEWT. Authors provide examples for client side applications that access HPC's resources directly through the Web browser.

Inter-operability is required in the context of distributed computing environments and one way of achieving inter-operability is using a Web service. The Internet has grown into a complex system which serves millions of people. The Internet is constantly changed with evolving new requirements of network security, reliable, end to end connection and mobility [35]. The performance of the Web services depends on the performance of the network, and it is an important issue for the Web service client and provider. There are more existing researches on service performance at the network level but not at the users' experience level. Authors proposed a new Web Service Evaluation System (WSES) which measures the end to end Web Service performance [35].

### 2.4.2 SOAP-based Web Service Vs RESTful Web Service

In [16] authors compare the traditional SOAP-based Web services with RESTful Web services for distributed data, pointing out the limitations of traditional (SOAP-based) Web services. Traditional Web services are characterized by a higher complexity, and a lower performance [16]. Each of these is briefly discussed.

1. Complexity - It is time consuming to serialize/deserialize native programming language based data into/from SOAP messages, and only a programmer can understand the complexities of the WS-* protocol.

2. Interoperability - Each service is to be defined with a specific service interface in a WSDL. If there are changes in the Web service side then the client has to follow the changes.

3. Performance - SOAP-based Web services are known to be heavy weight and significantly increase the communication network usage (both during resource discovery and message interchange) as well as the processing overheads. However, in a RESTful Web service, resources are identified through universal standard URIs and can be invoked directly.

A detailed comparison of RESTful Web service with "Big" (WS-*) Web services that are SOAP-based is provided by the authors in [33]. Although RESTful Web services are observed to be suitable for basic ad hoc integration scenarios the WS-* Web service are more flexible and can address advanced quality of service and security aspects which are commonly occurring requirements in the field of enterprise computing. RESTful Web services are typically can be more scalable and is often the preferred choice for Internet scale applications whereas SOAP-based Web services are often used for

deploying company's legacy systems which initially have not been built to be Web friendly and need to be integrated with other services and systems [12].

The advantages of migration from SOAP-based Web services to RESTful Web services for improving performance are discussed in [36]. SOAP messages contain a large chunk of XML data that can cause serious network latency [36]. Simplicity and scalability are the advantages of RESTful Web service over SOAP-based Web service.

The authors in [17] provide a comparison between the RESTful Web service and SOAP-based Web service. A detailed decision criterion which help the Web service providers and Web service consumer select which Web service technology to use are presented. In comparison with this thesis the authors in [17] only present business cases and list reasons for selecting either RESTful Web service or SOAP-based Web service. They did not address performance aware techniques for exposing computing and database resources as SOAP-based Web service or RESTful Web service that this thesis focuses on.

According to [7] the architectures of current SOAP-based Web services originated from Remote Procedure Call's (RPC) architectural style. An RPC-style architecture is not suited for Web scale applications because of the complexity and performance limitations of RPC. On the other hand, RESTful Web service fully utilizes Web features. Moreover, a RESTful Web service has the advantage of simplicity. Therefore, the RESTful Web service architecture is providing a new alternative to RPC architecture in Web services. In [7] authors analysed and compared the both RPC architecture and RESTful Web service architecture in detail in terms of scalability, coupling and security. Based on authors studied, the architecture of RESTful Web service out performs the

RPC's architecture in scalability, coupling and performance. In this research, the performances of two Web services technologies RESTful Web service and SOAP-based Web service are compared.

Web Service Description Language (WSDL) is very powerful and received much attention by the scientific community for its characteristics such as providing a platform of the composition of Web services [15]. RESTful Web service technology has become more persuasive among Web services technologies, but there is no good interface description language for a RESTful Web service. In [15] authors propose a flexible and extensible service description language for RESTful Web service called RIDDL (RESTful Interface Definition and Declaration Language). The Web Application Description Language (WADL) uses an XML format, which provides a machine process description [39]. The RESTful Web service uses WADL and WADL has the following disadvantages:

- End point service identifies a specific location for accessing a service using a specific protocol and data format [40]. End point service is part of the description and cannot be used when moving the service from one location to another.
- Can not add new functionality to an existing service. If a new functionality is added then an existing application does not function.

RIDDL overcomes these problems by adding new operations or changing operations on an existing Web service [15].

According to [11] RESTful Web service uses HTTP methods provide a better performance than the SOAP-based Web service for network service. The SOAP-based

Web service is observed to be a better solution for enterprise application integration cases.

### 2.4.3 SOAP-based Web Service Performance Issues and Solutions

In [21] authors describe the design and implementation of new approaches to SOAP message exchange optimizations. The optimizations are achieved by separating data contents from syntax and exploiting stored message exchanges. This novel optimization approach improves the efficiency of message exchanging by avoiding text conversions and conventional serializing and parsing activities.

In [6] authors investigate the use of SOAP/XML Web and grid service for scientific computing. The performance gain of SOAP/XML Web service can be achieved by: 1. Using (SOAP and WSDL) optimized XML data representation. 2. Reducing message passing latency with message chunking and compression.

SOAP-based Web service may be considered a poor choice for high performance Web services due to XML parsing [13]. The performance bottleneck of Web services are XML parsing and Java reflection at runtime. In [13] authors propose a new approach to improve Web service performance: the server maintains a SOAP processor for each Web service. The SOAP processor has a SOAP parser and deserializer that can only recognize the SOAP messages specific to a given Web service. One of the advantages of the SOAP processor is the performance gain where the SOAP processor helps the SOAP engine to accelerate message processing. The proposed SOAP processor used in the SOAP engine produced approximately a three times performance gain [13].

### 2.4.4 Mobile Environment

A comparison between RESTful frameworks and SOAP-based frameworks for mobile hosts is presented in [2]. Authors find that RESTful Web services are more attractive for mobile environments because of the following reasons:

1. RESTful Web services do not require performance demanding parsers.

2. RESTful Web services support caching which can lead to savings in the limited network bandwidth available in a mobile environment.

3. Mobile devices have limited resources and RESTful Web services do not consume large amounts of CPU and memory resources.

Since the introduction of smart phone, the telecommunication and Information Technologies are working together to provide convenient accessible services to the consumer [1]. Compared to the desktop computers, the mobile devices are limited in terms of processing power and storage capacity. These two factors could affect the performance of the mobile applications [1]. Authors compare the performance of the Mobile Web services for an HTTP payload. Two Web services technologies: RESTful Web service and SOAP-based Web service are used for evaluating the Mobile Web service performance. Authors conclude that RESTful Web service outperforms SOAP-based Web service for the following reasons: first, RESTful Web service has architecture advantage such as the mapping of REST URI to HTTP methods. Second, a RESTful Web service produces less HTTP payload than SOAP-based Web service's HTTP payload.

A number of papers existing in the literatures were discussed in this chapter. The main differences between the research presented in this thesis and the existing research are:

1. This research presents a performance aware approach for exposing computing and database resources using both SOAP-based and RESTful Web services technologies.
2. A thorough performance comparison between the two Web services technologies for various resource types using various combinations of systems and workload parameters is performed.
3. This research proposes a technique for combining the advantages of both the RESTful Web service and SOAP-based Web service technologies. To the best of our knowledge this is the first work on such a hybrid Web service technique.

# Chapter 3 : Exposing Resources as a Web Service

## 3.1 Exposing a Computing Resource as a Web Service

Exposing a computing resource as a Web service enables diverse clients implemented using different programming languages and running on top different operating systems to invoke the Web service and use the resource. Both local and remote clients can be handled. A client submits a request containing the name of the program to execute and associated arguments on the Web service. Once the execution is completed, the client receives the results from the Web service. All the applications invoked by the client are hosted on the computer being exposed as a Web service. There are several advantages of exposing the computing resource as a Web service:

1. Regardless of the client's platform a client can use the computing resource.
2. The client does not need to have all the applications and software on the client's platform; instead, the client invokes the Web service and uses or executes the program available on the remote computing resource exposed as a Web service.

Figure 3.1 shows the interaction among the components of a computer resource exposed as a Web services. The client is presented with a Web browser, and then the client chooses the selection of the Operating System (OS) to use. Based on the selection, the client is redirected to the corresponding operating type Web browser to either Windows OS or Linux OS. Then, the client enters the command in the Web browser and submits the request. The client requests are executed on the selected computing resource exposed as a Web service and the executed results returned to the client.

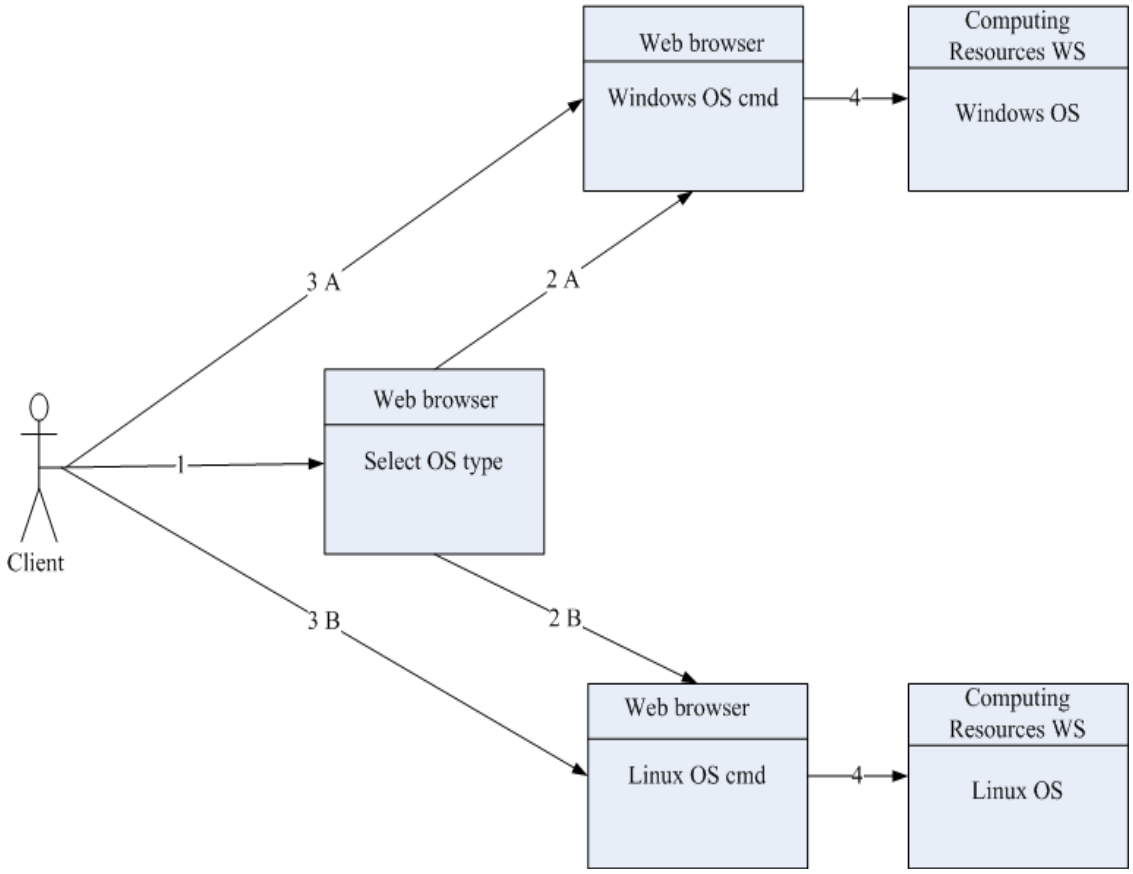


**Figure 3.1: Computer resources exposed as Web services**

### 3.1.1 Computing Resource Exposed as a Web service Type

The computing resource is exposed as two types of Web services. Computing resource exposed as a Web service RESTful (CWSR) is the version based on the RESTful Web service technology and computing resource exposed as a Web service SOAP-based (CWSS) is the version based on the SOAP-based Web service technology. Both CWSR and CWSS are developed in Java using the NetBeans IDE [22]. The NetBeans package includes the Glass Fish server; therefore CWSR and CWSS are deployed on the Glass Fish Server [24].

Figure 3.2 shows the class diagram for computing resource exposed as a Web service. Computing resource exposed as a Web Service (CWS) uses a helper class called Reader

and two Java Application Interface classes called Runtime [25] and Process [26].

With Runtime class, an external command can be executed on a Windows operating system or a Linux operating system. The external command is executed by using the exec method from Runtime class.

The exec method executes the external command in a new process and returns a Process object. This "executecmd" is capable of handling all Windows and Linux commands.

Then CWS class uses the methods getOutput ( ) and getError ( ) from Reader class to retrieve the output results or an error message from the process object generated when the exec method executes the external command.

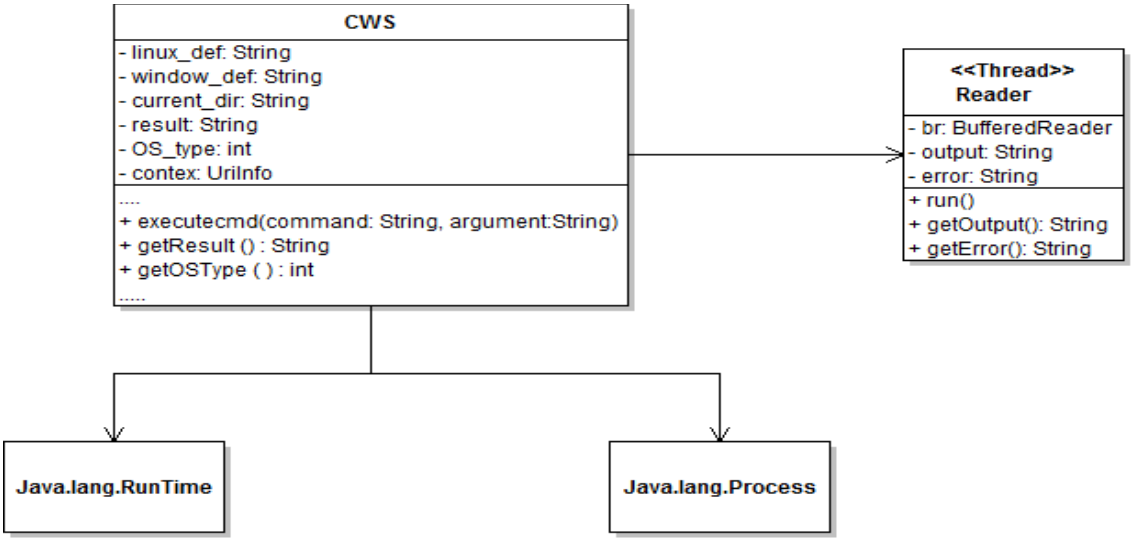


**Figure 3.2 : Class diagram for CWS**

### 3.1.2 CWSR

CWSR is developed with using JAX-RS and it is a Java API for RESTful Web service [27]. RESTful Web service uses WADL, which is a description language, used for

developing HTTP-based Web applications [39]. The RESTful Web service is based on the notion of a resource that can be accessed with HTTP methods PUT, GET, UPDATE and DELETE. CWSRResource is used by CWSR.

#### 3.1.2.1 JAX-RS Annotations

The annotations bind the HTTP operations to methods in CWSR's Java class [34]. Figure 3.3 shows the JAX-RS annotations that are used in CWSR for developing the RESTful Web service.

The following list describes each of the JAX-RS annotations: (item numbers correspond to the numbers marked in Figure 3.3)

1. @Path - Identifies the URI path to the resource class or method which will serve the request [For example @Path("REST_CR_WS")].
2. @GET - Gets a resource that will process the HTTP GET request
3. @Produce - Used to specify the MIME media type of representation used for formatting the results produced by CWSR [27].

```
@Path("REST_CR_WS") ---------- (1) Resource path
public class REST_CR_WSResource {
.....
   @Path("/{command}/{argument}") ---------- (1) Method path
   @GET ----------(2)
   @Produces("text/plain") ----------(3)
   public String executeCommand(@PathParam("command")  String command, @PathParam("argument")
String argument) {
.....
```

**Figure 3.3: JAX-RS annotations used in CWSR**

@Path (“REST_CR_WS”) identifies the URI for CWSR. The “executeCommand” method performs the HTTP GET method on CWSR. @Path (“{command}/{argument}”) provides the parameters to be used in the “executeCommand” method in the CWSR. @Produces(“text/plain”) returns the “executedCommand” result in text format.

### 3.1.3 Invoking CWSR

CWSR can be invoked in several ways. For example, it can be invoked with the Web browser by entering the CWSR’s method path with parameters or it can be invoked by a program. Figure 3.4 shows the CWSR-Client invoking the CWSR with a Web browser.

CWSRC enters “http://localhost:8080/REST_CR_WS/resources/REST_CR_WS/dir” into the Web browser address bar and then the result shows the content of the directory.

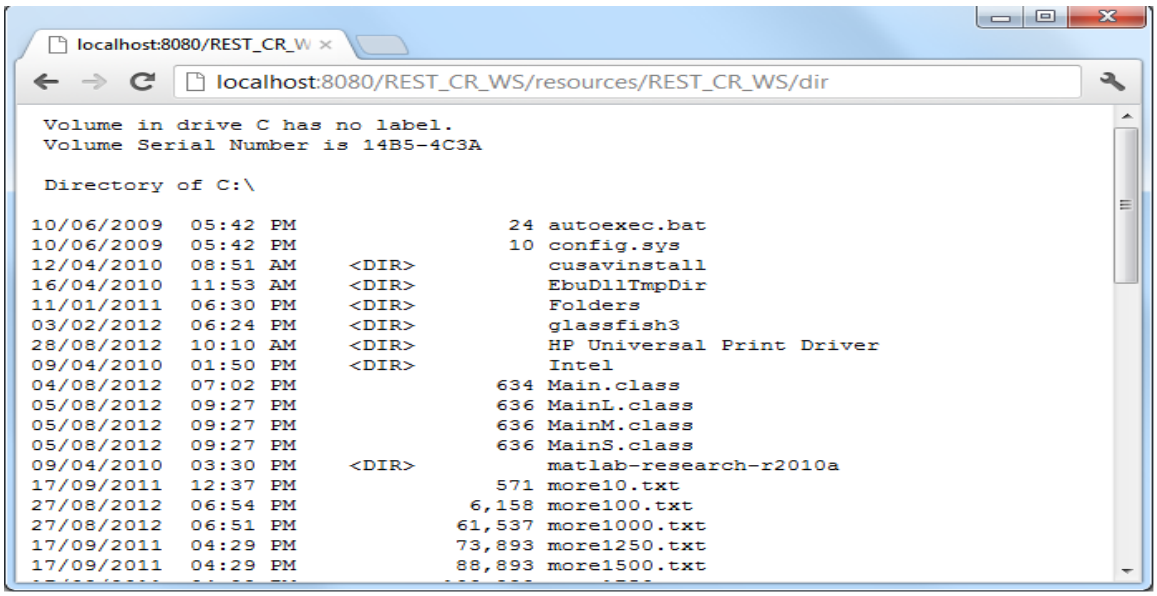


**Figure 3.4: CWSRC invoking CWSR**

Figure 3.5 presents a sequence diagram showing how CWSR is invoked from CWSRC. CWSRC invokes the CWSR with a Web browser and then the CWSRC request is forwarded onto CWSR; where CWSR executes the "executecmd" method in a new process and creates two objects to handle reading the output and the error message from the process.

The wait() method of class "process" is invoked to block the current thread of execution until the execution of the requested command from "executedcmd" method is completed. Finally, the output of the "executecmd" method is retrieved from the output object and sent to CWSRC. If an error occurs during the execution of "executecmd" method, an error message is retrieved from the error object and sent to CWSRC.

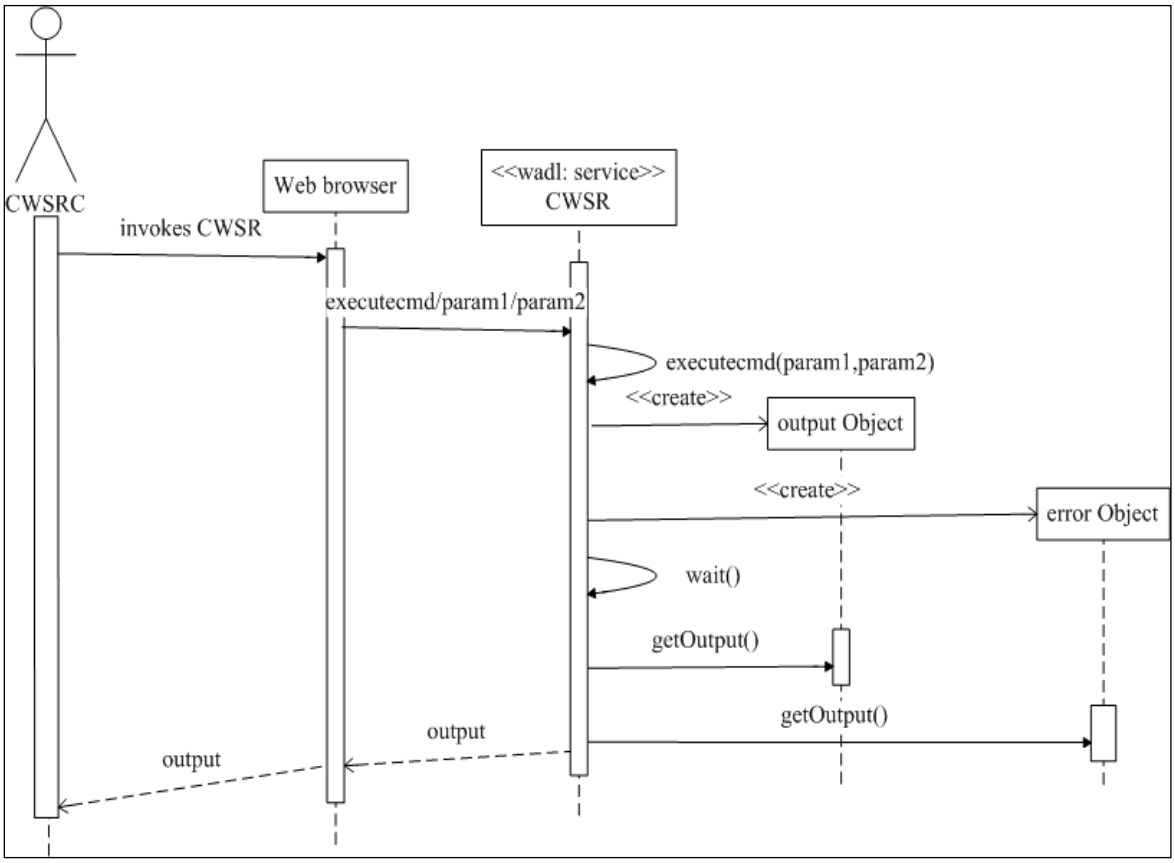


**Figure 3.5: Sequence diagram showing the invocation of CWSR**

### 3.1.4 CWSS

CWSS uses JAX-WS annotation, which is a Java API for SOAP-based Web services [28]. The SOAP-based Web service uses WSDL as discussed in detail in Section 2.2.

CWSS was developed by Lim [14] and Melendez. This component is reused in this work. All the remaining system components CWSR, DBWSS, DBWSS, CHWS and DBHWS were developed as part of this thesis.

#### 3.1.4.1 JAX-WS Annotations

By using annotations from JAX-WS; a computing resource is exposed as a SOAP-based Web service. JAX-WS makes it easy for developing SOAP-based Web service using Java technology. Figure 3.6 shows the JAX-WS annotations used in CWSS.

```
@WebService() ----------------------------------------------------------------(1)
public class SOAP_CR {
.....
@WebMethod(operationName = "executeCommand") ---------------------(2)
public String executeCommand(@WebParam(name="command")String --(3)
command,@WebParam(name="argument")String argument) {
.....
}
.....
}
```

**Figure 3.6: JAX-WS annotations used in CWSS**

Annotations that are used in the CWSS are briefly described below (corresponding numbers marked in the Figure 3.6):

1. @WebService - Java class exposed as a Web service

2. @WebMethod - The method used for Web service operation [For example similar to the method “executecmd” in CWSR].
3. @WebParam - Used for mapping the Web service’s input parameters to a Java method parameter [28].

### 3.1.5 CWSS and CWSS-Client components

Figure 3.7 shows the communication diagram for CWSS-Client (CWSSC) that invokes the CWSS. The communication diagram comprises five entities: Web browser, JavaServer Page(JSP), SOAP request, SOAP response and CWSS.

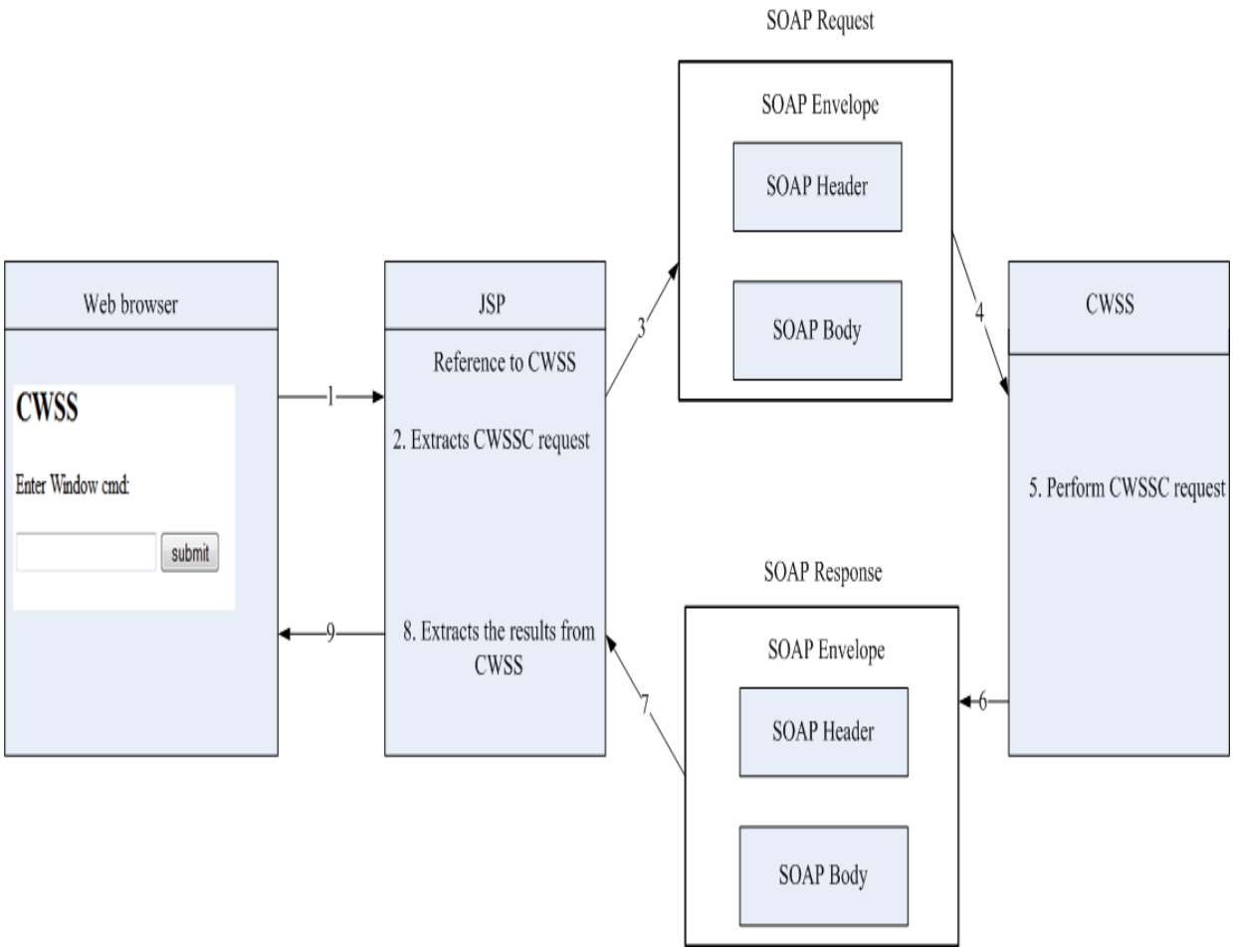


**Figure 3.7: CWSSC invoking the CWSS**

Following subsection describes the entities in the communication diagram.

#### 3.1.5.1 Web Browser

CWSSC interacts with CWSS or gets the CWSS service through the Web browser.

#### 3.1.5.2 JavaServer Page (JSP)

Using JSP technology, a Web developer or Web designer is able to develop and maintain dynamic Web pages [29] and JSP Web applications are platform independent. Once CWSSC submits the request, the JSP then extracts the information submitted by CWSSC (see Figure 3.7). In order to invoke CWSS, a reference to CWSS is needed in JSP. This reference to CWSS is created in the JSP component. All the public Web methods from CWSS are accessible through the JSP component; "executecmd" is an example of such a Web method to execute an operating system command. The "executecmd" is invoked with parameters that are submitted by CWSSC.

#### 3.1.5.3 SOAP Messaging

SOAP messages are used during the communication between JSP and CWSS components. JSP component generates the SOAP request that contains CWSSC's Web service request. After executing the command, CWSS creates the SOAP response with the results of command execution for CWSSC.

#### 3.1.5.4 CWSS

CWSS is the computing resource exposed as a SOAP-based Web service. The executecmd method in CWSS extracts the parameters from the SOAP request sent from JSP and executes the CWSSC's request. When CWSS completes CWSSC's

request, CWSS sends back the result or (an error message) to JSP through the SOAP response message.

The following sequence of interactions occurs among the entities in Figure 3.7:

1. CWSSC enters the Windows operating system command and clicks on the submit button, and the CWSSC request is forward to the JSP.
2. JSP extracts the CWSSC request and call CWSS.
3. JSP sends the SOAP request to CWSS.
4. CWSS receives incoming SOAP request.
5. CWSS performs the CWSSC request.
6. CWSS sends the SOAP response containing the results of executing the command (or an error message in case the command execution is not successful).
7. JSP receives the incoming SOAP response from CWSS.
8. JSP extracts the results/error from the SOAP response.
9. Finally, the result/error is displayed on the Web browser.

## 3.2 Exposing a Database Resource as a Web Service

By exposing a database resource as a Web service, the remote or local client can query, retrieve and update data stored in the database. We have created a student database to be exposed as a Web service. This is a simple relational database which has the following attributes: student id, last name, first name, year of study and program. The focus of this thesis is on the performance of the techniques used for exposing a resource as a Web service. The technique for exposing the database as a Web service described in this

research can be extended to other databases as well. The following database queries are supported by the student database:

- Insert a student record
- Read a student record
- Update a student record
- Delete a student record

### 3.2.1 Database Resource Exposed as a Web Service Types

In this research both the RESTful Web service and SOAP-based Web service technologies are used for exposing the database resource as a Web service. Database resource exposed as a Web service RESTful (DBWSR) is the version based on the RESTful Web service technology and Database resource exposed as a Web service SOAP-based (DBWSS) is the version based on the SOAP-based Web service technology. Both the DBWSR and DBWSS are developed in Java using the NetBeans IDE and are deployed on the Glass Fish Server.

### 3.2.2 Database Resource Exposed as a Web Service (DBWS)

The student database is created using the MySQL Server [19]. The following MySQL queries: insert, get, update and delete are used with DBWS. The student database is accessed through Java DataBase Connectivity (JDBC) [30]. JDBC is the JavaSoft specification of a standard API which allows Java programs to access database management systems (DBMS). The JDBC API contains a set of interfaces and classes written in the Java which allow the programmer to develop applications that connect to databases, send queries written in the structured query language (SQL), and process the

results [31].

### 3.2.3 DBWSR

Figure 3.8 shows the JAX-RX annotations used in DBWSR for inserting a student record into the student database.

```
@Path("REST_DB_WS")
public class REST_DB_WS {
.....
   @PUT
   @Path("/putstudent/{studentnumber}/{fname}/{lname}/{program}/{classyear}")
   @Consumes(MediaType.TEXT_PLAIN)
   public Response putStudent(@PathParam("studentnumber") int number, @PathParam("fname")
   String fname, @PathParam("lname") String lname, @PathParam("program") String program,
  @PathParam("classyear") int cyear) {

.....
```

**Figure 3.8: JAX-RS annotations for the insert database operation on DBWSR**

As in the case of exposing computing resource as RESTful Web service, (see Section 3.1.2.1) the JAX-RS annotations were used in the DBWSR for developing the RESTful Web service.

@Path ("REST_DB_WS") identifies the URI for DBWSR. The "putStudent" method performs the HTTP PUT method on DBWSR. @Path ("/putstudent..") identifies the URI for the putStudent method in DBWSR. @Path ("../{student...}/{...}.....") are the parameters are used in "putStudent" method. @Consumes(MediaType.TEXT_PLAIN) expects student information in text format for performing the insert database operation on DBWSR.

DBWSR has similar JAX-RS annotations for other database operations.

Figure 3.9 shows database resource exposed as a RESTful Web service and the interactions among its components.

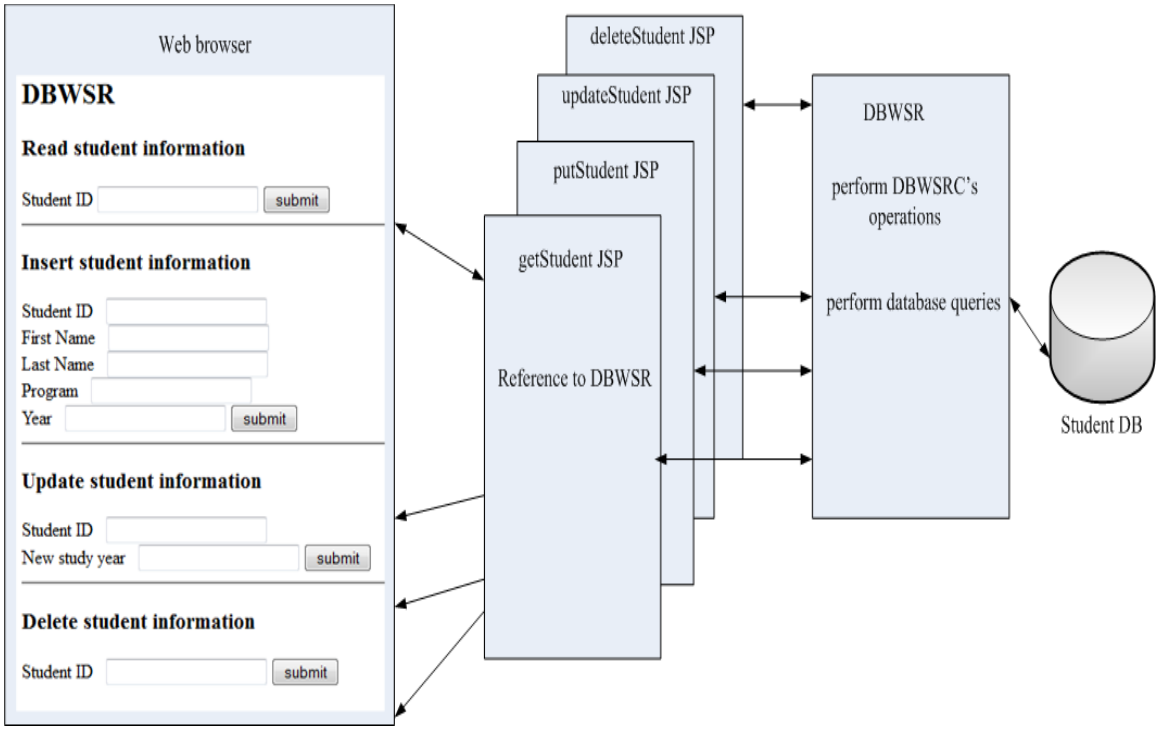


**Figure 3.9: Database resource exposed as DBWSR**

DBWSR can be invoked through a Web browser or from a program. DBWSR-Client (DBWSRC) uses Web browser to perform one of the operation on the student database. Based on the operations performed, one of the following JSPs invokes DBWSR: getStudent JSP, putStudent JSP, updateStudent JSP, and deleteStudent JSP. If the getStudent JSP is invoked for example, the getStudent JSP extracts information submitted by DBWSRC and invokes the getStudent method on DBWSR. In case of the other operations invoked by DBWSRC, the corresponding method putStudent, updateStudent or deleteStudent is invoked on DBWSR. The reference to DBWSR is

created in all JSP components. All the public Web methods from DBWSR are accessible by all JSP components.

DBWSR extracts information from the incoming request from JSPs. Then DBWSR executes the corresponding database operations. The techniques to connect to the student database, and how to execute the database operations were described in Section 3.2.2. DBWSR sends the results of performing the requested operation or an error message (in case of the requested operation was not successful) back to the corresponding JSP. The corresponding JSP displays the results or the error message to DBWSRC.

### 3.2.4 Sequence Diagram for Invoking the Insert Database Operation on DBWSR

Figure 3.10 shows the sequence diagram for invoking the insert database operation on DBWSR. The following sequence of events takes place for the insert database operation on DBWSR:

1. DBHWSC enters the student information and submit the request for adding student record into student database.
2. Student information is forwarded to the DBWSR Servelet.
3. DBWSR Servlet invokes the putStudent service with the submitted student information.
4. DBWSRResource executes the putStudent method using the following sequence of operations:

a) DBWSRResource connects to studentDB.

b) DBWSRResource generates a SQL statement to put the student information into studentDB.

c) DBWSRResource sends the generated SQL statement for execution on the studentDB.

d) DBWSRResource receives confirmation message (or an error message in case the operation was not successful) from studentDB.

5. DBWSR Servlet receives the response from DBWSRResource
6. DBWSR Servlet sends the results / error message to on the Web browser for display.

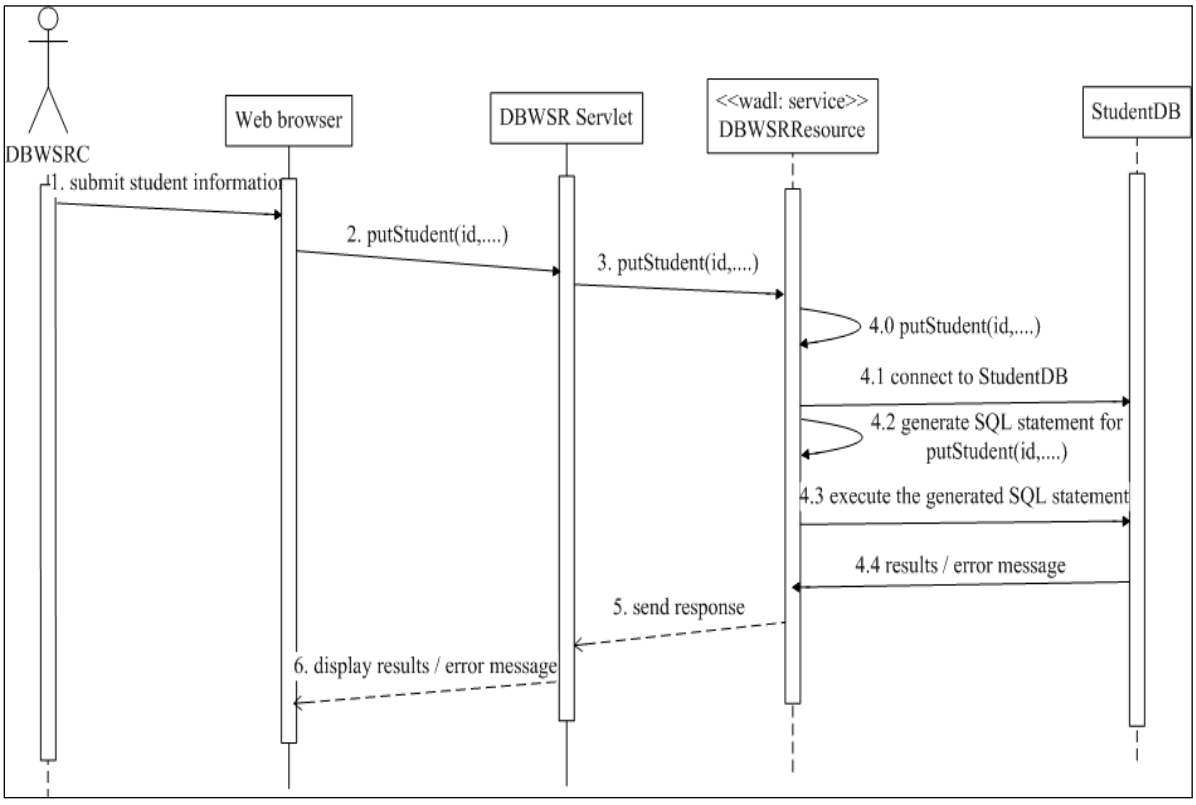


**Figure 3.10: Sequence diagram for invoking the insert database operation on DBWSR**

### 3.2.5 DBWSS

DBWSS uses JAX-WS that described in Section 3.1.4.1. The same connection technique used for DBWSR (discussed in detail in Section 3.2.2) has been used to connect the student database to DBWSS. Figure 3.11 shows the JAX-WS annotations used in DBWSS for inserting student record into the student database.

```
@WebService(serviceName = "SOAP_DB_WS")
public class SOAP_DB_WS {

.............
@WebMethod(operationName = "insert")
putStudent(@WebParam(name = "fname") String fname, @WebParam(name = "lname") String
lname, @WebParam(name = "number") int number, @WebParam(name = "program") String
program, @WebParam(name = "cyear") int cyear)

.....................
}
```

**Figure 3.11: JAX-WS annotations for the insert database operation on DBWSS**

As in the case of exposing computing resource as a SOAP-based Web service, (see Section 3.1.4.1) the JAX-WS annotations were used in the DBWSS for developing the SOAP-based Web service. The annotations @WebService, @WebMethod and @WebParam map Java code to the WSDL files.

Read, update, and delete database operations in DBWSS use annotations shown in the Figure 3.11.

Figure 3.12 shows the database resource exposed as a SOAP-based Web service the interactions among its components.

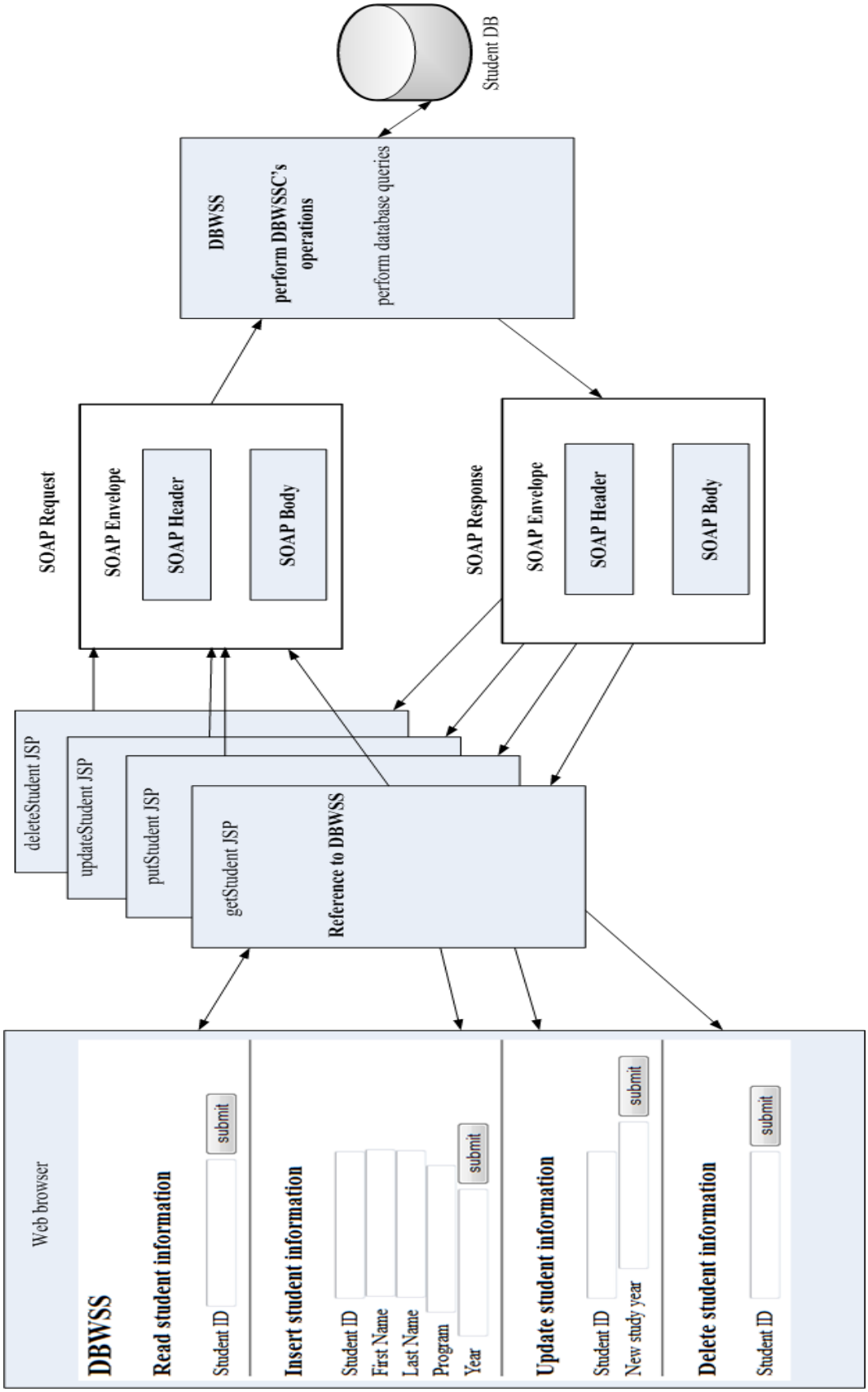


**Figure 3.12: Database resource exposed as DBWSS**

Web browser and StudentDB in Figure 3.12 are identical to those in Figure 3.9. The SOAP request and SOAP response components are different from those described in Figure 3.9. Though, the sub component that performs database queries within the DBWSS is identical to that of DBWSR shown in Figure 3.9. The reference to DBWSS is created in all JSP components; and all the public Web methods from DBWSS are accessible by all the JSP components.

DBWSS-Client (DBWSSC) invokes one of the student database operations. The corresponding JSP extracts the information submitted by DBWSSC. From the extracted information the JSP generates the SOAP request and sends the SOAP request to DBWSS. DBWSS extracts the information from the incoming SOAP request from JSPs. Then DBWSS invokes the corresponding database operation. Section 3.2.2 describes techniques used for connecting the student database and performing the database operations on student database. After the request is completed, DBWSS generates the SOAP message with the results of the operation (or an error message in case the operation was not successful) and sends it back to the corresponding JSP. JSP displays the results or the error message to DBWSSC through DBWSS.

### 3.2.6 Sequence Diagram for Invoking the Insert Database Operation on DBWSS

Figure 3.13 shows the sequence diagram for invoking the insert database operation on DBWSS.

This sequence diagram is very similar to the sequence diagram for invoking the insert database operation on DBWSR. The differences are indicated in bold (see Figure 3.13). The main difference is that DBWSS uses a DBWSS servlet. The DBWSS servlet uses the SOAP request and SOAP response to communicate with DBWSS. The DBWSS

servlet in turn invokes DBWSS that communicates with StudentDB.

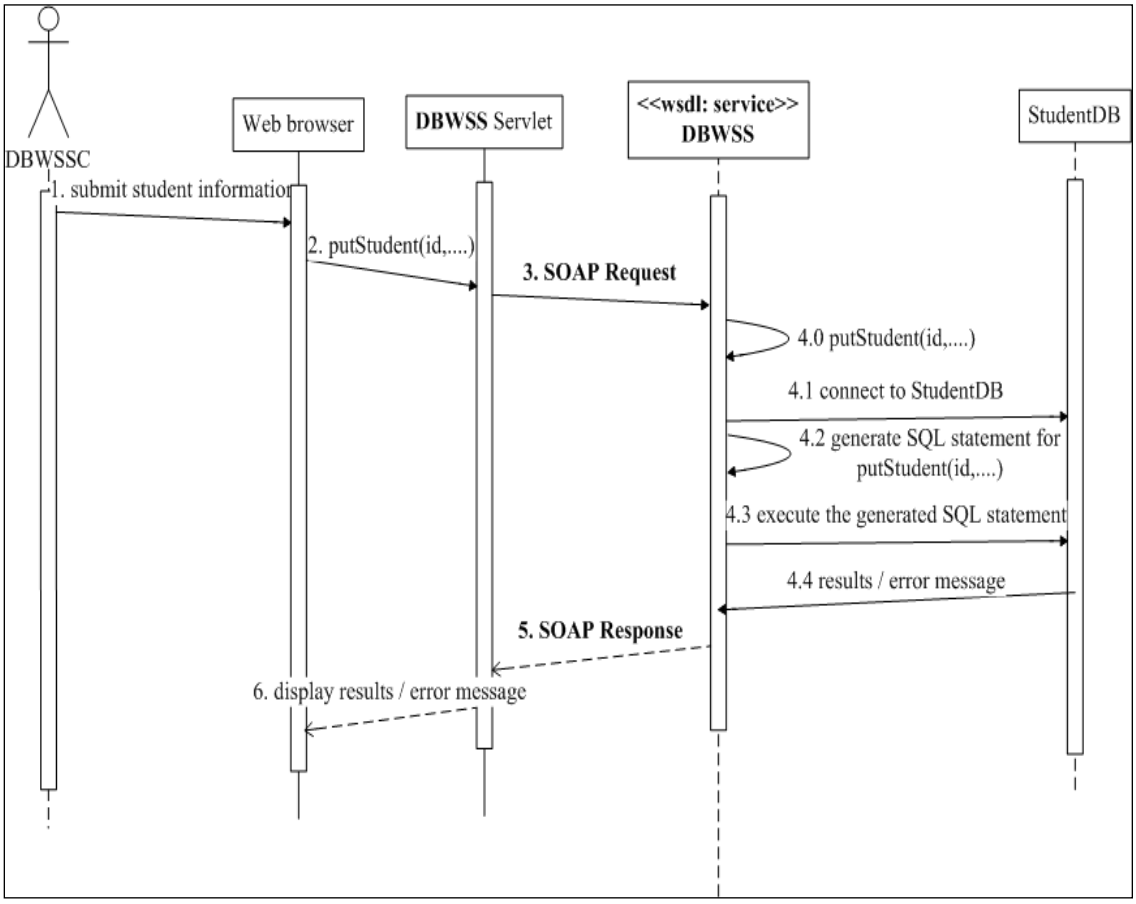


**Figure 3.13: Sequence diagram for invoking the insert database operation on DBWSS**

# Chapter 4 : Hybrid Web Service

Both RESTful and SOAP-based Web services were discussed in the earlier sections. Each Web service has distinct features which are suitable for different types of applications. The SOAP-based Web services support a number of standards that are critically important in the context of a variety of different distributed applications. Although lightweight, the RESTful Web service does not provide support for any such standard.

This research proposes a new Web service technique called Hybrid Web service that combines the advantages of both the RESTful and SOAP-based Web services. The advantages of the RESTful Web service that are incorporated into the Hybrid Web service include:

1. its lightweight nature of using simple message exchange over HTTP [16].
2. its simple requirement of using only four set of HTTP operations on the resources class.

The Hybrid Web service also includes the advantages of SOAP-based Web service including its support for several standards: the WS-* that includes the standards for WS-Security for security, WS-Reliable Messaging for error handling and WS-Atomic Transactions for transaction atomicity.

Although security and atomicity of transactions may be critically important in the context of some operations performed by a client, the other operations may not have such

security or transaction atomicity requirements. Thus, using the SOAP-based Web services for all the operations can slow the system down significantly. The Hybrid Web service exploits this diversity in client operations and tries to achieve high performance as well as satisfy the other non-performance requirements (such as security) of the application by selectively using the two types of Web services technology. The SOAP-based Web service technology is used only when the operational demand for the support of the other standards that are not available with RESTful whereas the lightweight RESTful Web services technology is used in all the other cases.

## 4.1 Computing Resource Exposed as a Hybrid Web Service (CHWS)

Figure 4.1 shows the components of exposing computing resource as a Hybrid Web service.

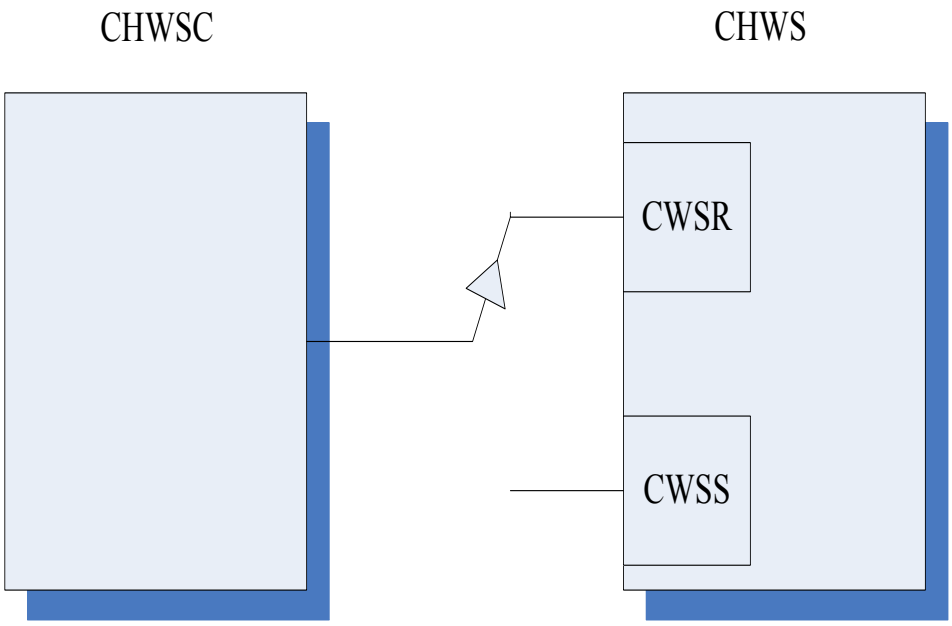


**Figure 4.1: Exposing Computing resource as a Hybrid Web service**

The client using CHWS is called CHWS-Client (CHWSC). The CHWS consists of two components: CWSR - RESTful Web service component and CWSS - SOAP-based Web service component each of which is invoked selectively. The CHWS is implemented in Java using the NetBeans IDE. The same implementation techniques used for CWSR (see Section 3.1.2) and CWSS (see Section 3.1.4) are used for implementing the respective RESTful and SOAP-based components in the Hybrid Web service.

### 4.1.1 CHWS Selection Implementation

Figure 4.2 shows the class diagram for CHWS. CHWS is an interface class which has an invoke method with the command parameter. CWSR and CWSS implement the invoke method from interface CHWS. CHWSC then invokes the appropriate class's implementation of the invoke method.

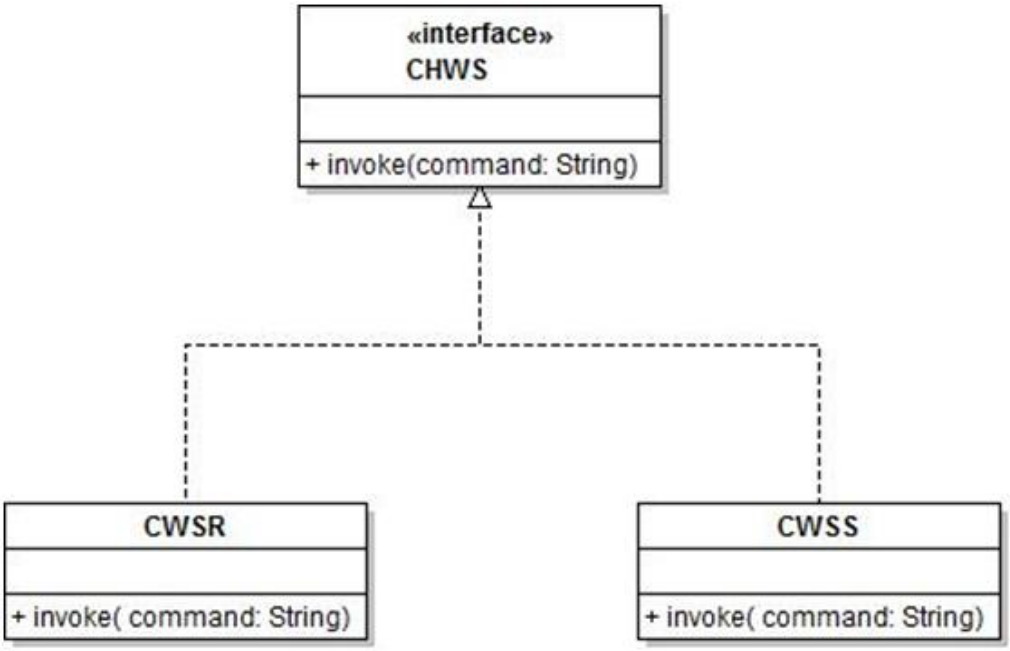


**Figure 4.2: Class diagram for CHWS**

### 4.1.2 Consuming the Computing Resource as a Hybrid Web Service

CHWS can be invoked by a program or by a Web browser. Figure 4.3 shows how CHWSC invokes the CHWS through a Web browser. The client explicitly requests for the CWSR (RESTful) or the CWSS (SOAP-based) component. Following sequence of events take place on the system where a computer resource is exposed as a Hybrid Web service.

1. Using the CHWS Web browser, CHWSC selects either CWSR or CWSS.
2. CHWSC is redirected to the CWSR Web browser or CWSS browser based on the choice made in step 1.
3. The remaining sequence of events for CHWSC on CHWSS is identical to the sequence of events described in Section 3.1.5.
4. The following sequence of events occurs in the system when CHWSC selects CWSR.
    a) CHWSC enters the Windows command and clicks on the submit button.
    b) The CHWSC request is forwarded to the CWSR JSP.
    c) CWSR JSP extracts the CHWSC request
    d) CWSR JSP invokes CWSR.
    e) CWSRResource performs the CHWSC request.
    f) CWSRResource sends the executed results (or error message in case the command execution is not successful) back to CWSR JSP.
    g) The result/error displays on the Web browser.

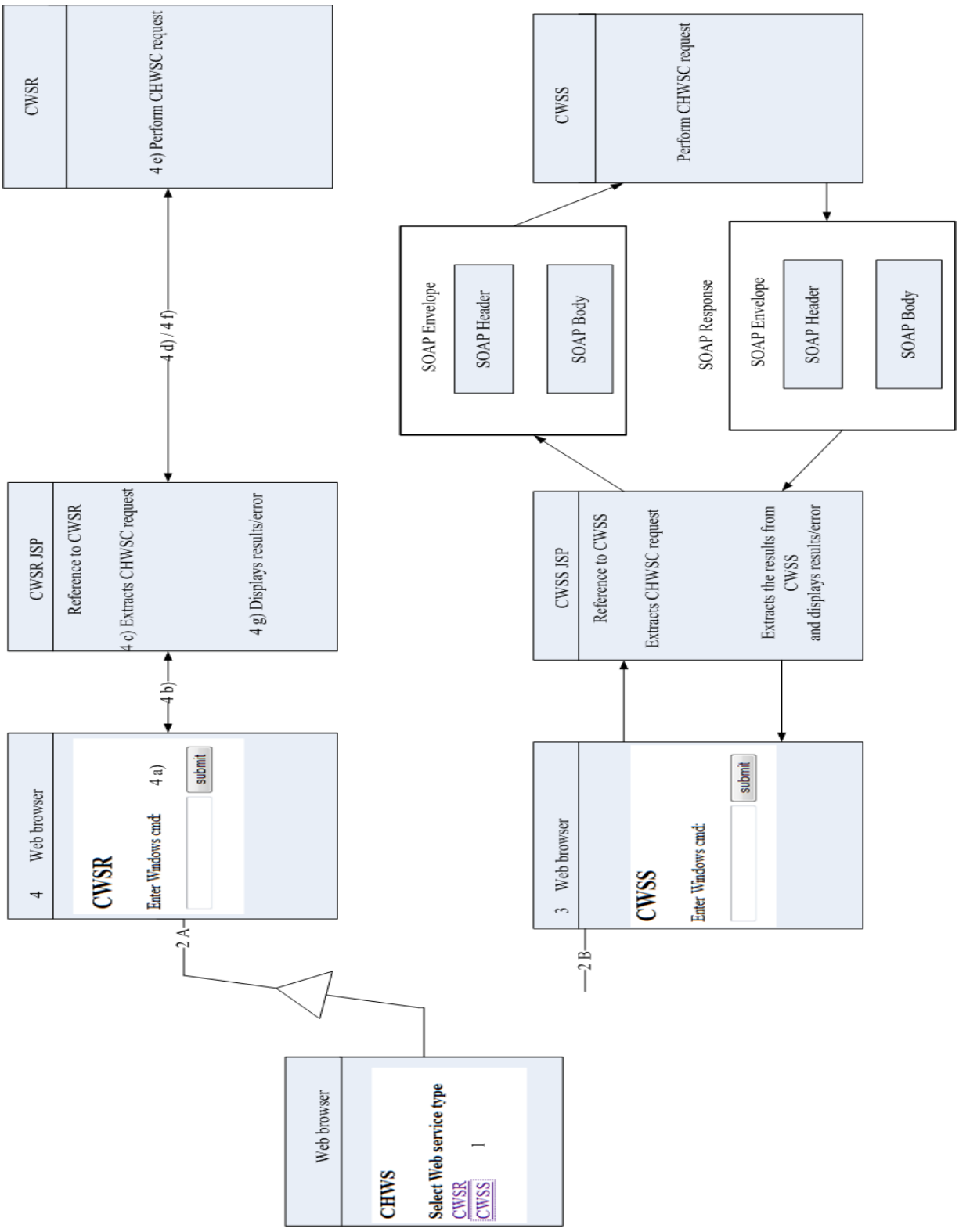


**Figure 4.3 : Computer resource exposed as a Hybrid Web service**

## 4.2 Database Resource Exposed as a Hybrid Web Service (DBHWS)

A client can invoke the two versions of the Hybrid Web services. In the first version called DBHWS - Type I, the client explicitly requests for the RESTful or the SOAP-based component. In the second version, the client requests for the service that in turn invokes the RESTful Web service or the SOAP-based Web service depending on the operation performed by the client. For example, reading a student record operation can use the RESTful Web service whereas updating a student record that leads to a change in the database uses the SOAP-based Web service because of its support for atomic transactions that is important in the context of the update operation. The second version of DBHWS is called DBHWS - Type II. The types of operations requiring the SOAP-based technology are stored in a configuration repository that is consulted when a request arrives at the Hybrid Web service and an appropriate Web service (RESTful or SOAP-based) is chosen by comparing the requested operation with those stored in the configuration repository.

Figure 4.4 presents an example of DBHWS - Type I.

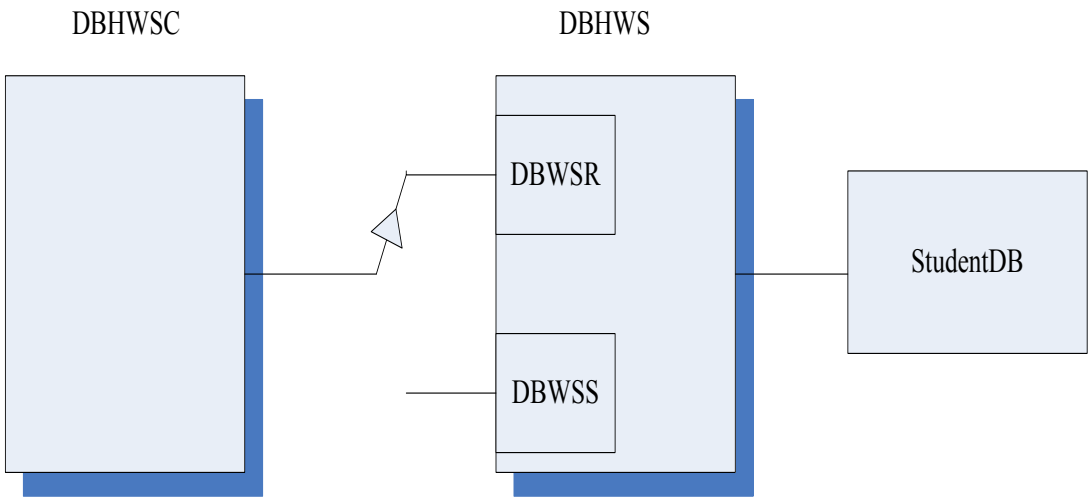


**Figure 4.4: Exposing database resource as a Hybrid Web service - Type I**

The DBHWS consists of two components: DBWSR - the RESTful Web service component and DBWSS - the SOAP-based Web service component each of which is invoked selectively. The same implementation techniques used for DBWSR (see Section 3.2.3) and DBWSS (see Section 3.2.5) were used for implementing the respective RESTful and SOAP-based components in the Hybrid Web service.

### 4.2.1 DBHWS - Type I Implementation

The technique used for the selection of DBHWS is similar to that used in the case of CHWS selection implementation. Figure 4.5 shows the class diagram for DBHWS where DBHWSC explicitly requests for the DBWSR or the DBWSS component. DBHWS is an interface class that has an invoke method with two parameters: database operation type (op_type) and, student information (student). DBWSR and DBWSS implement the invoke method from interface DBHWS. DBHWSC invokes the appropriate class's implementation of the invoke method.

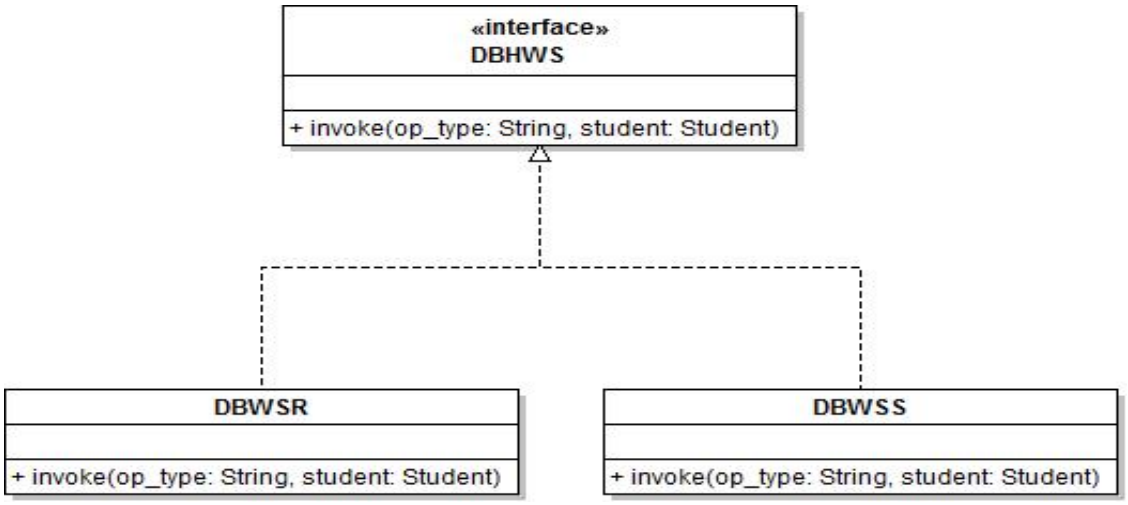


**Figure 4.5: Class diagram for DBHWS - Type I**

Figure 4.6 shows the database resource exposed as Hybrid Web service and the among its components interactions.

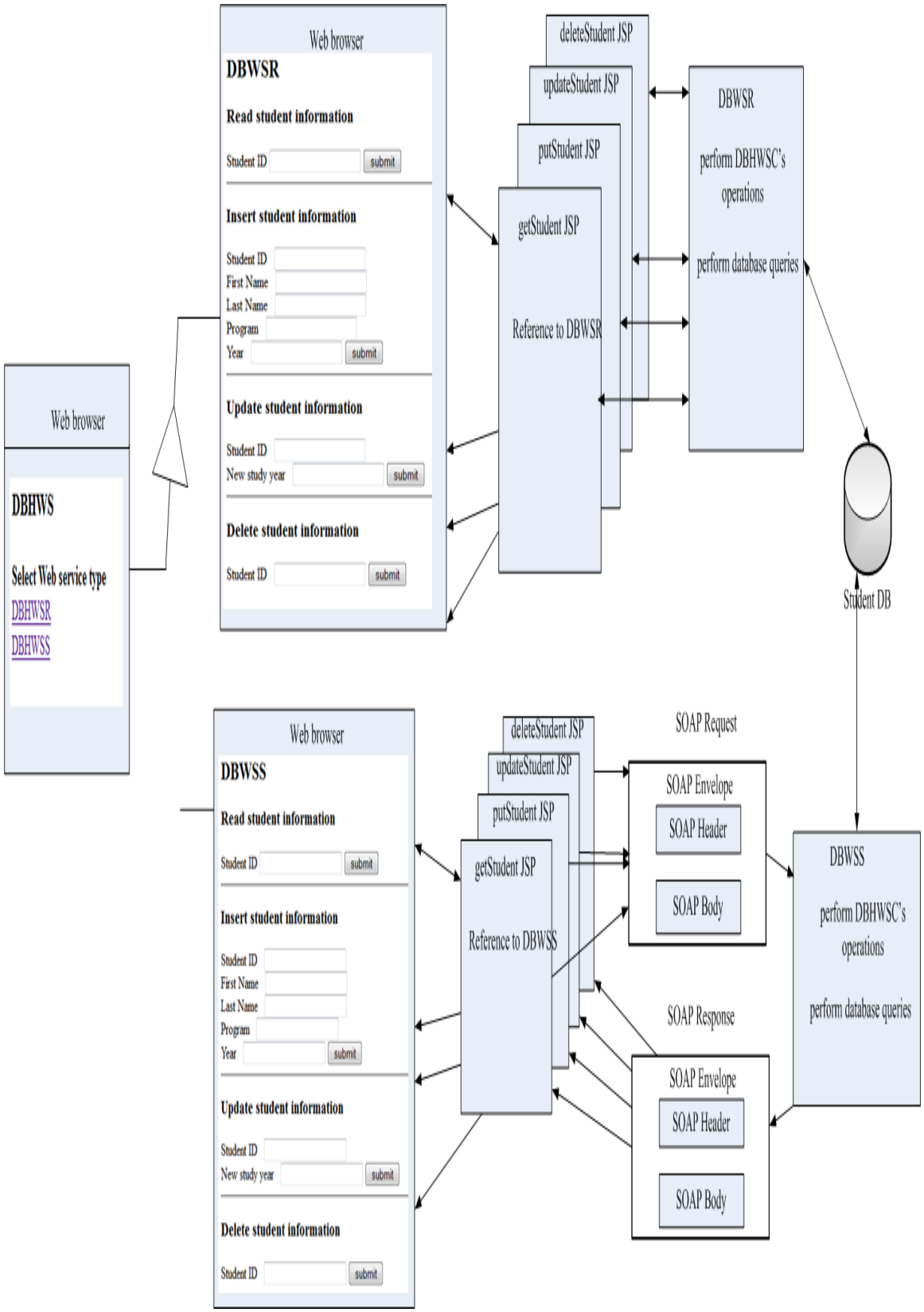


**Figure 4.6: Database resource exposed as a Hybrid Web service - Type I**

DBHWS can be invoked by a program or through a Web browser. For the Web browser, DBHWSC is presented with a selection of Web services. DBHWSC selects a Web service type and performs various database operations.

DBHWS - Type I has two major components: DBWSR and DBWSS. The interaction within DBWSR and DBWSS are identical to the interactions within the systems describesd in Section 3.2.3 and Section 3.2.5 respectively. Web browser is used by DBHWSC for selecting the DBHWSR or DBHWSS. The StudnetDB is accessed by both DBWSR and DBWSS. However only one of them has excusive access to the StudentDB at a time.

### 4.2.2 DBHWS - Type I Sequence Diagram

Figure 4.7 shows a sequence diagram when DBHWSC selects DBHWSR for performing the insert database operation on DBHWS -Type I.

The following events occur when DBHWSC select the DBHWSR from the Web browser:

1. DBHWSC selects DBHWSR and DBHWSC is redirected to DBWSR Web browser.

2. DBHWSC enters the required student information and submits the request.

3. DBHWSC Servlet extracts DBHWSC's request.

4. DBHWSC Servlet invokes the DBWSRResource.

The remaining events 5 to 7 are identical to the events 4 to 6 described in Section 3.2.4 for Figure 3.10.

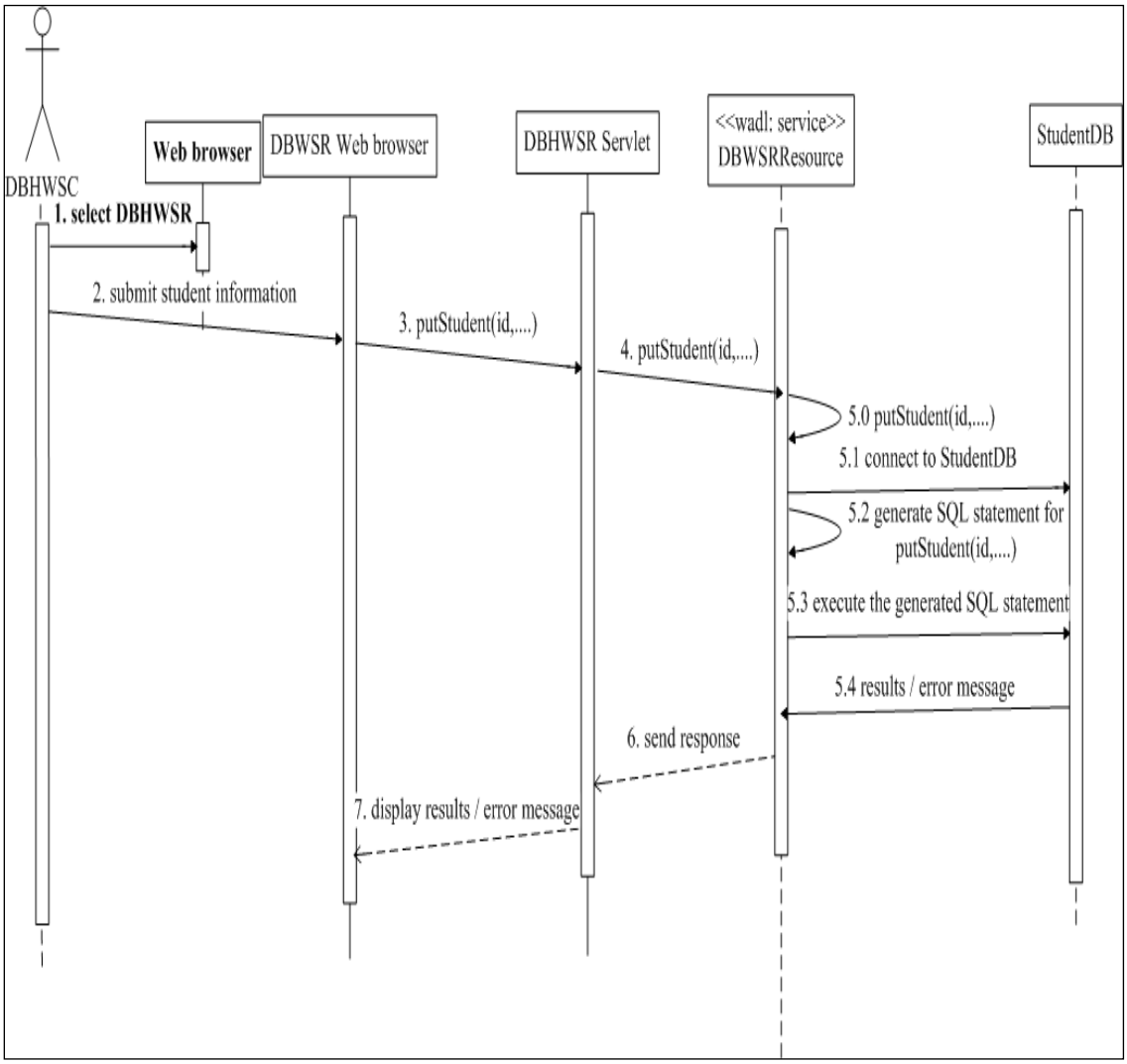


**Figure 4.7: Sequence diagram for the insert database operation on DBHWS - Type I**

### 4.2.3 DBHWS - Type II

In Figure 4.8, DBHWSC request one of the student database operations described in Section 3.2. Based on DBHWSC requests either one of the DBWSR or DBWSS

component is invoked. The read database operation invokes DBWSR whereas insert, update and delete database operations invoke DBWSS.

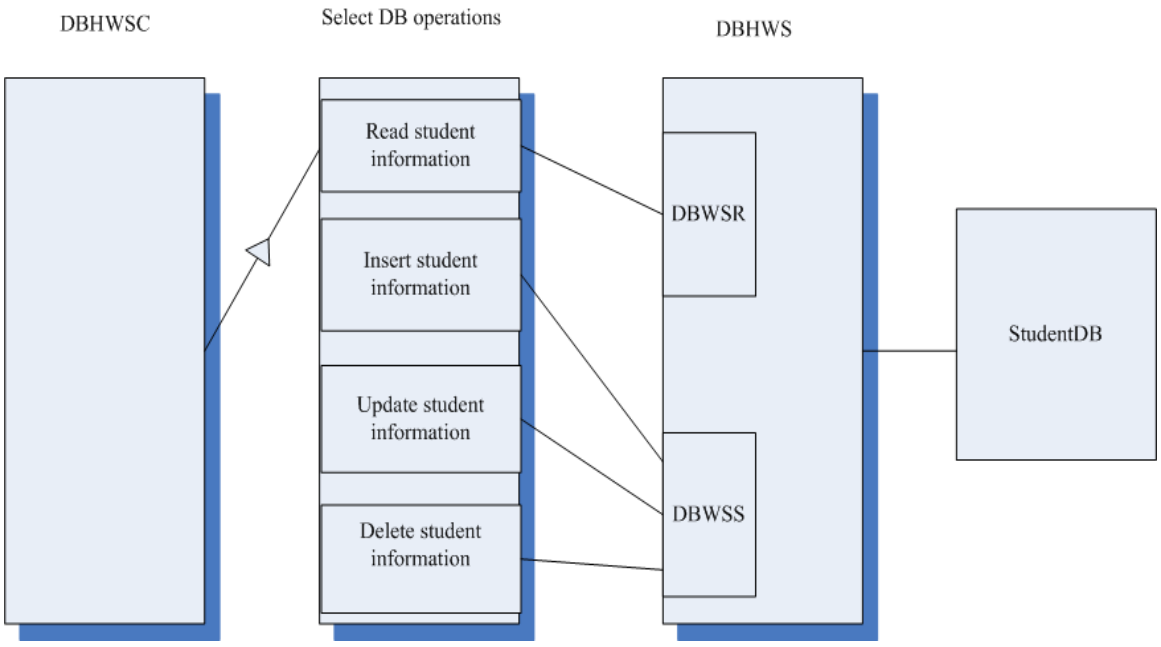


**Figure 4.8: Exposing Database resource as a Hybrid Web service – Type II**

Once again, database resource exposed as a Hybrid Web service is implemented in Java using the NetBeans IDE. The same implementation techniques used for DBWSR (see Section 3.2.3) and DBWSS (see Section 3.2.5) were used for implementing the respective RESTful and SOASP-based components in the Hybrid Web services. The studentDB is accessed by both the RESTful and the SOAP-based Web services and either of them has an exclusive access to the student database at a time.

### 4.2.4 DBHWS - Type II Implementation

Figure 4.9 shows the class diagram for DBHWS - Type II. DBWSR or the DBWSS component is invoked based on the database operation performed by DBHWSC. The configuration repository defines the rules to be used for selecting DBWSR or DBWSS based on the operation. As described in Section 4.2.1 DBHWS is an interface

class; and DBWSR and DBWSS implement the invoke method from interface DBHWS. The invoke method is used for selecting the DBWSR or the DBWSS. DBHWSC uses the DBHWS interface class. Based on the database operation requested by the client and the information stored in the configuration repository DBWSR or DBWSS is invoked. DBHWSC remains the same as described in Section 4.2.3.

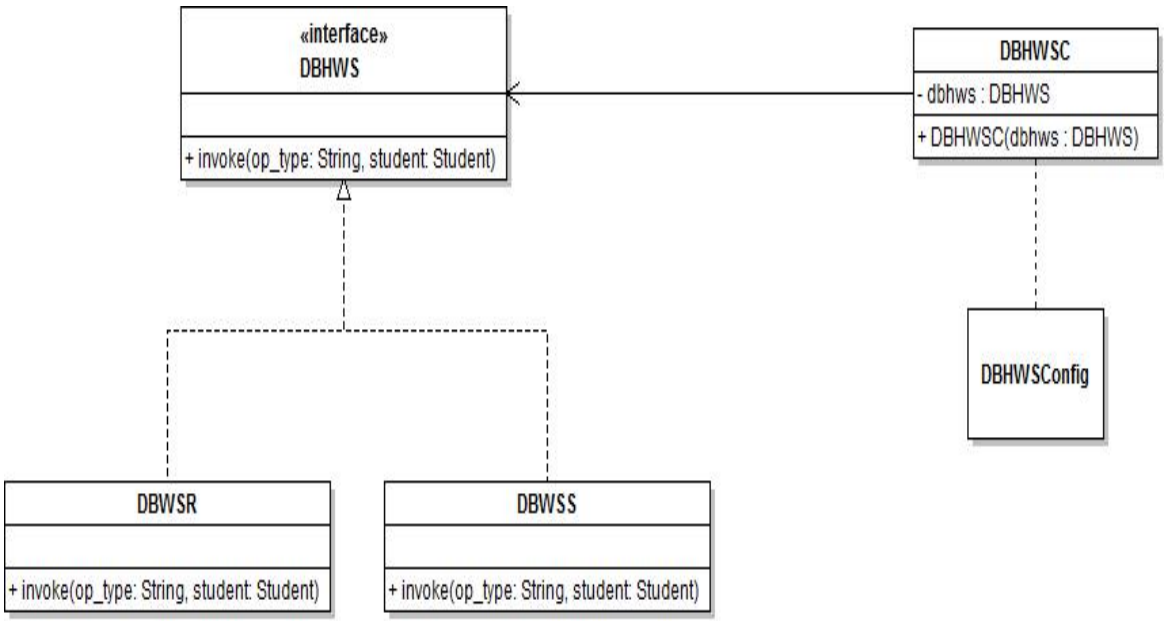


**Figure 4.9: Class diagram for DBHWS -Type II**

Figure 4.10 shows DBHWS - Type II and the interactions among its components. This system can also be invoked by a program or through a Web browser. In the case a Web browser is used, DBHWSC is presented with a selection of database operations. DBHWSC selects one of the operations and provides the required student information. The request is then submitted to the DBHWS Web browser.

DBHWS JSP has references to DBWSR and DBWSS. DBHWS JSP invokes DBWSR or DBWSS based on the database operation performed by DBHWSC.

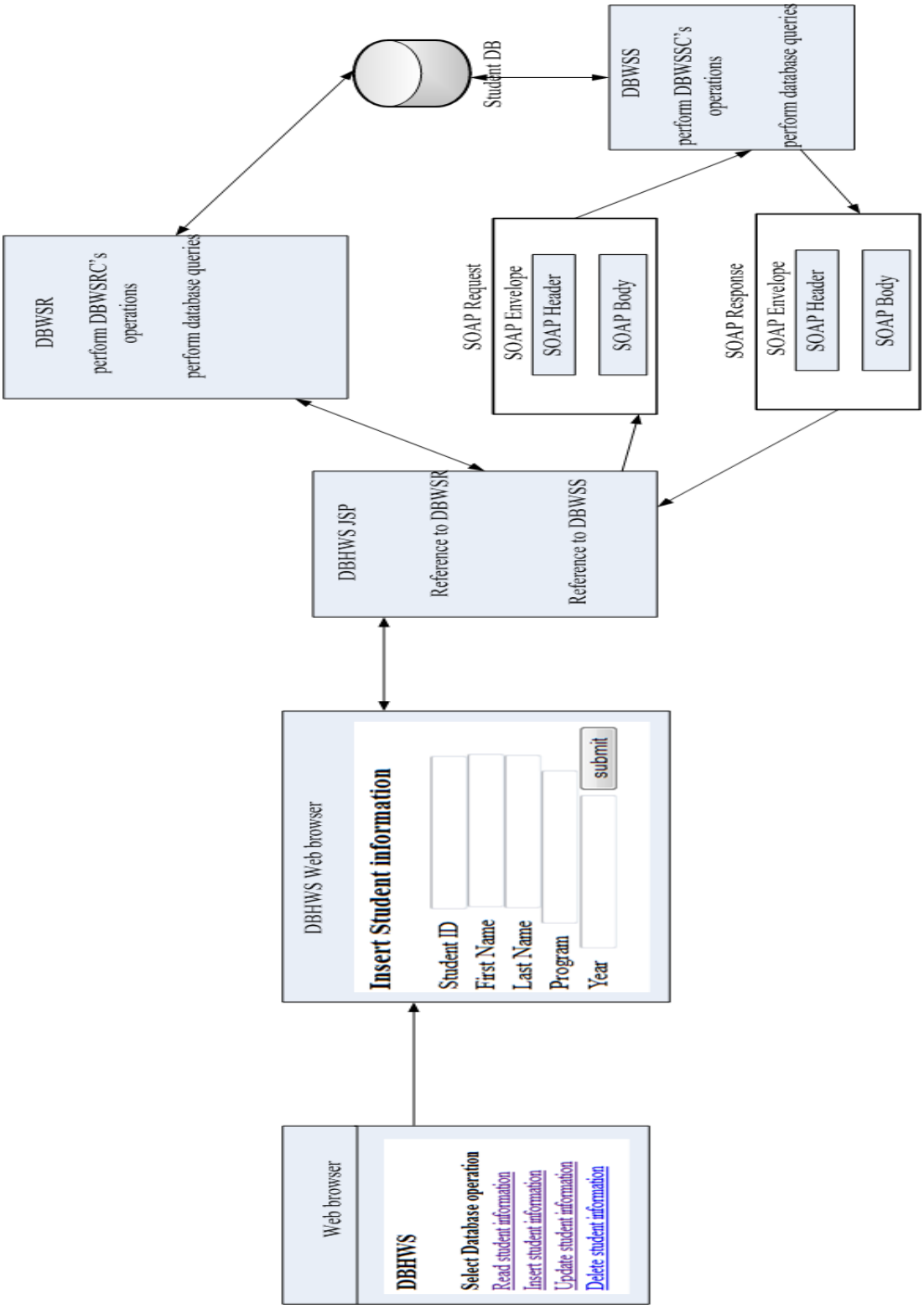


**Figure 4.10: Database resource exposed as Hybrid Web service - Type II**

The interactions between DBHWS JSP and DBWSR as well as between DBHWS JSP and DBWSS are similar to that describesd in Section 3.2.3 and Section 3.2.5 respectively.

### 4.2.5 DBHWS - Type II Sequence Diagram

Figure 4.11 shows a sequence diagram for DBHWSC performs the insert database operation on DBHWS - Type II.

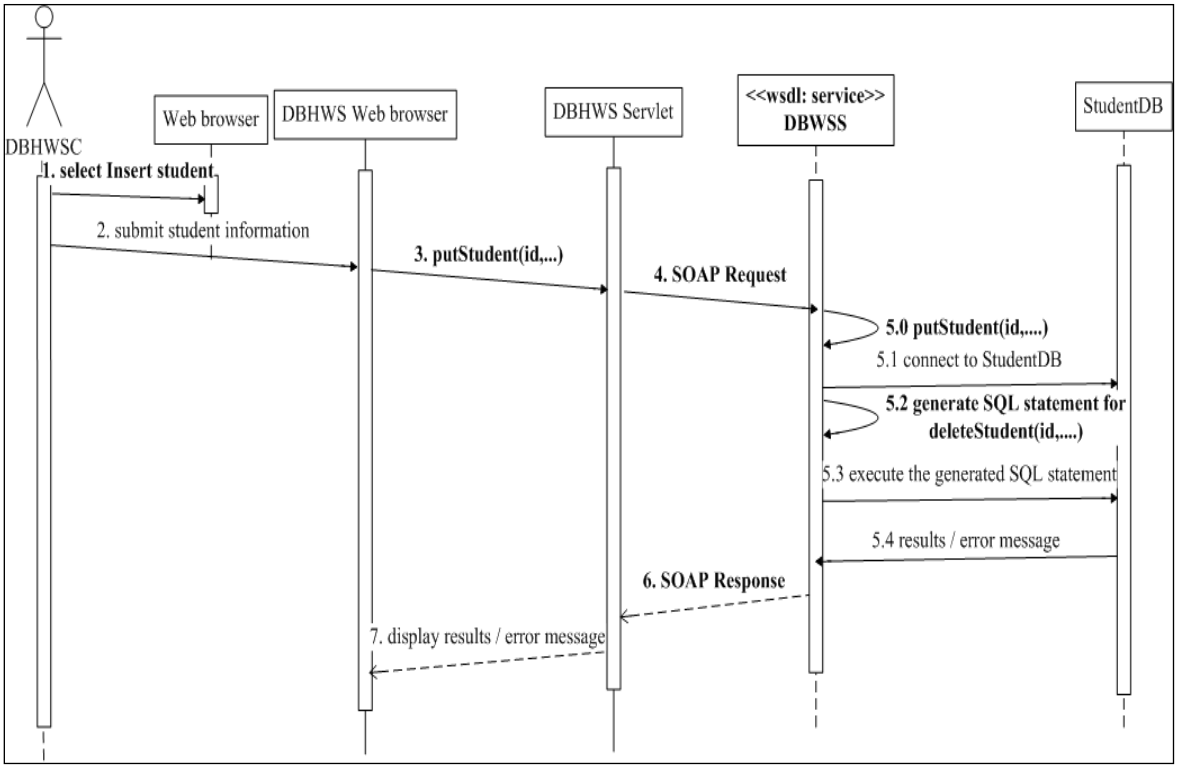


**Figure 4.11: Sequence diagram for the insert database operation on DBHWS - Type II**

The differences between Figure 4.7 and Figure 4.11 are highlighted in the Figure 4.11.

1. DBHWSC performs the insert database operation on DBHWS - Type II.
2. *DBHWSC enters the required student information and submits the request (identical to event 2 Figure 4.7).*

3. *DBHWS Servlet extracts DBHWSC's request (identical to event 3 Figure 4.7).*
4. DBHWS JSP sends the SOAP request to DBWSS.

The remaining events 5 to 7 are identical to the events 4 to 6 described in Section 3.2.6 for Figure 3.13.

# Chapter 5 : Performance Analysis

This chapter presents and discusses the results of the performance analysis conducted on prototypes of systems that expose resources as Web services. The performance analysis is divided into three parts:

1. Analysis of systems exposing a computing resource as a Web service.
2. Analysis of systems exposing a database resource as a Web service.
3. Analysis of systems exposing the computer and the database resources as a Hybrid Web service.

## 5.1 Experimental Setup

The performance analysis of the systems exposing computing resource and database resources as Web services and Hybrid Web services was accomplished by conducting experiments on various interactions between client and Web services. The NetBeans IDE is used to develop the clients and Web services; both clients and Web services are running on the same machine. The system that was used for the experiments is described in Section 5.1.2. A thorough validation was conducted to ensure that clients and Web services are running on the different cores. For each Web service request, the client establishes a new connection with the Web service. Following sequence of operations take place with each Web service request by the client.

1. Establish a new connection with the Web service.
2. Invoke the Web service operation.
3. Receive result.
4. Record the performance metrics.
5. Close the connection with the Web service.

For systems subject to a Poisson stream of request arrivals, each experiment is run long enough (1500 request arrivals) such that the system reaches a steady state. For all the systems investigated, an experiment is repeated 60 times such that a confidence interval of ±5% is achieved at a confidence level of 95%.

Section 5.1.1 describes the performance metrics, Section 5.1.2 describes the systems specifications and Section 5.1.3 describes the parameters used in the experiments.

### 5.1.1 Performance Metrics

The following two performance metrics are used in this research:

1. Web service Response Time (WSRT)
2. Web service Average Response Time (WSART): WSART is the average WSRT computed over all the runs of a given experiment.

The WSRT is calculated as follows: Before the client invokes the Web service, the client takes a timestamp (request start time) by using *nanoTime ( )* method provided by Java's System class [32]. Then the client takes another timestamp (request finish time) after the requested Web service results are received by the client. The WSRT is the time differences between the requests the second and the first time stamps. That is, WSRT includes all associated overheads on the client side and the Web service side: such as the request set up time at the client side, overheads associated with invoking the service and result set up (incurred prior to sending the result to client) at the Web service side. WSART is computed by taking the mean of all the WSRT. The WSART is used for analysing the performance of various Web services.

In this research, we used the clients CWSRC and CWSSC (described in Section 3.1.3 and Section 3.1.5 respectively); DBWSRC and DBWSSC (described in Section 3.2.4 and Section 3.2.6 respectively); CHWSC and DBHWSC (described in Section 4.1 and Section 4.2 respectively) to compare the WSARTs achieved with the RESTful Web services with that of the SOAP-based Web services.

### 5.1.2 Systems Specifications

The client and the corresponding Web services were running on two different cores in the same system. The system comprised of an Intel Core i7 CPU 860, 2.80 GHz (4 cores) and 4GB of RAM running under the Windows 7 operating system. Similar relative performance results for the two Web service technologies are expected from different systems.

A number of key experiments were repeated on a two machine system using a Linksys router (Model No. WRT54GS V7) connecting the two machines. The Web service was run on the machine described in the previous paragraph where as the client was run on a system comprising of an Intel Core 2 Duo CPU, 2.26 GHz and 2GB of RAM running under the Mac OS X Snow Leopard operating system. The differences in performance between the two machines and the single machine systems were observed to be less than 2%.

### 5.1.3 Parameters Used in the Experiments

Table 5.1 shows the summary of the Workload parameters used in the experiments. The first column has the names of the workload parameters. The corresponding values and default values to parameters are shown in the second and the

third column respectively. A factor at a time approach is used in the performance analysis. One of the workload parameter is varied while others were held at the default values (see Table 5.1). Based on the experiment, λ was varied from a low value at which negligible queuing occurs to a value that captures the impact of queuing at higher system loads.

**Table 5.1: Summary of the parameters used in the experiments**

| Workload Parameter Name | Values | Default Value |
|---|---|---|
| CWSR requests | dir, java-version, help, path, ver, vol, read file, execute program | dir |
| CWSS requests | dir, java-version, help, path, ver, vol, read file, execute program | dir |
| DBWSR requests | read, insert, update, delete | read |
| DBWSS requests | read, insert, update, delete | read |
| CHWS request | dir, java-version, help, path, ver, vol, execute program | dir |
| DBHWS request | read, insert, update, delete | read |
| Size of Web request (number of record in DB) | 1, 2, 3, 4, 5 | 1 |
| Arrival rate, λ (requests/s) | Dependent on the experiment | N/A |

Two types of resources (computing and database) were exposed as Web services using the RESTful Web service, the SOAP-based Web service and the Hybrid Web service. As shown in Table 5.1 CWSR, CWSS and CHWS correspond to exposing Windows based computing resources as Web services with RESTful, SOAP-based Hybrid Web services respectively. The listed Windows commands in Table 5.1 were used in the performance evaluations. The Windows command "dir" was chosen as default value because it was also used in a similar research in the literature (refer to [14]).

Similarly, DBWSR, DBWSS and DBHWS are exposing student database resources as Web service. Read, insert, update and delete database queries were used in the performance evaluations. The read database query is used as default value because it will not modify the information in the database.

## 5.2 Performance Evaluation for Exposing a Computing Resource as a Web Service

For CWSR and CWSS, the clients CWSRC (for RESTful Web service) and CWSSC (for SOAP-based Web service) use the Windows commands: "dir", "java –version (version is one of the parameter that goes with the java command)", "help", "path", "ver" ,"vol" during the different experiments that were used to compare the performance of CWSR (RESTful Web service) with CWSS (SOAP-based Web service).

### 5.2.1 Performance of Different Web Service Request

Figure 5.1 and Figure 5.2 show WSART for different Windows command executions with CWSR and CWSS.

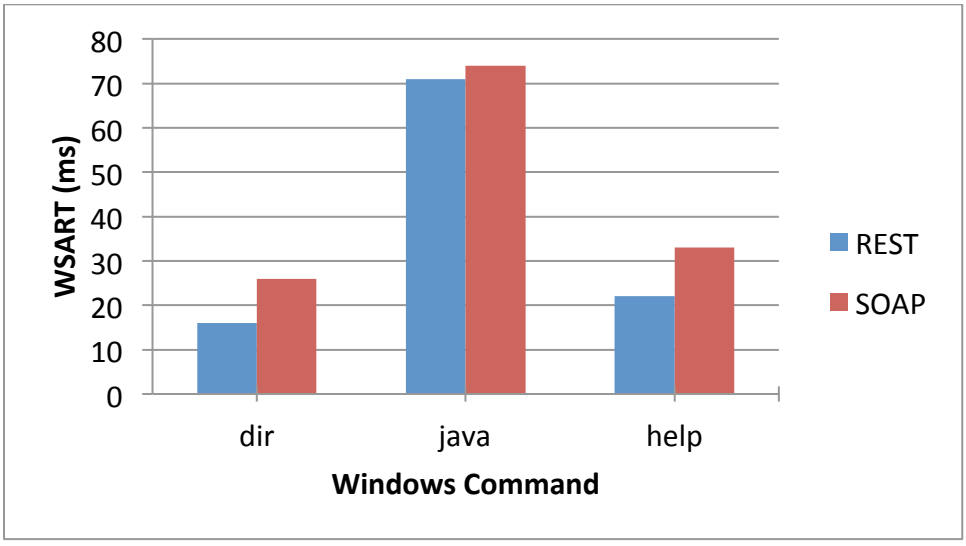


**Figure 5.1: Performance for Windows commands for CWSR and CWSS**

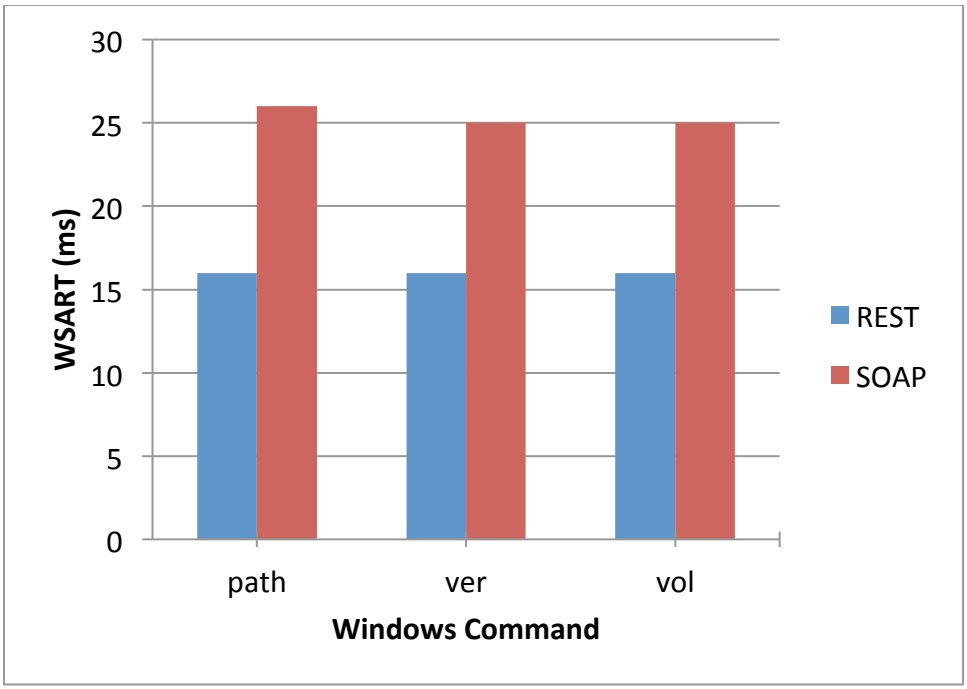


**Figure 5.2: Performance for Windows commands for CWSR and CWSS**

Figures clearly show that WSART achieved with each command for the RESTful Web service is lower than that of for the SOAP-based Web service. The WSART for RESTful Web service is 38% lower than that of SOAP-based Web service for Windows command "dir" because the RESTful Web service does not use the SOAP messaging layer. The result message produced by the SOAP Web service consists of results and the extra XML markup required by SOAP. These additional overheads are responsible for the inferior performance of the SOAP-based Web service. Although there is a substantial difference in performance between the two types of Web services for dir, help, path, ver and vol; both types of Web services results in comparable WSARTs for the java command. This is because of the execution time for the operating system command is much higher in comparison to the overhead incurred by any of the Web service technology. As a result WSART is dominated by the execution time for the command and the overheads from the Web service technology play only a minor role. Therefore, a

comparable WSART can be achieved for both the technologies.

### 5.2.2 Effect of Web Service Request Size

This section presents and analyses the effect of Web service request size on the computing resource exposed as a Web service. Different sizes of text files are used in this experiment while clients CWSRC and CWSSC performed read operation on (CWSR and CWSS) those text files. In this experiment that input files size is varied. The experiments have been divided into two groups: clients perform read operation on file set1 in one and clients perform read operation on file set2 on the other. Figure 5.3 shows the WSART for reading file set1 (6 Kb - 60 Kb) and Figure 5.4 shows the WSART for reading file set2 that includes larger files (up to 150 Kb).

As expected the RESTful Web service gives rise to a lower WSART than SOAP-based Web service for reading the files in file set1 and file set2. (see Figure 5.3 and Figure 5.4) The WSART for reading the smallest file with RESTful Web service is 32% lower than that achieved with the SOAP based Web service (see Figure 5.3), while in the case of reading the largest file (150 Kb) the WSART for the RESTful Web service is only 2% lower than that for the SOAP based Web service (see Figure 5.4).

As explained in the previous section, SOAP-based Web service has higher WSART because of the result message produced by SOAP Web service consists of results and the extra XML markup required by SOAP messages. These additional overheads are responsible for the inferior performance of the SOAP-based Web service. However, the effect of this additional XML markup data transfer becomes a less dominant factor for reading larger files because a greater volume of data needs to be transferred through both

Web Services: RESTful and SOAP-based. As a result a smaller performance gain is accomplished with the RESTful Web service. Thus, the performance of the SOAP-based Web service is expected to be closer to that of the RESTful Web service for higher file sizes.

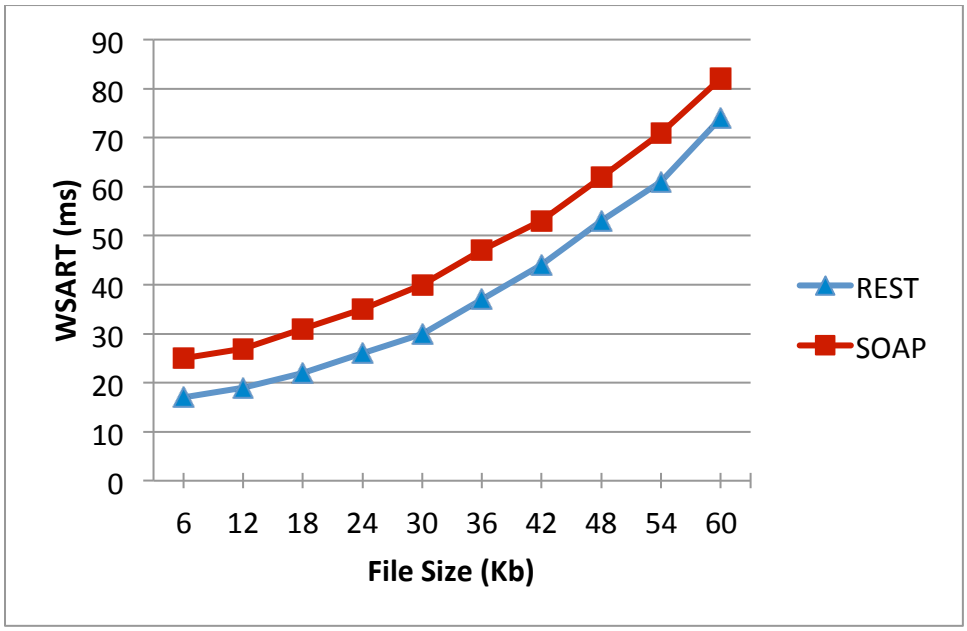


**Figure 5.3: Performance of CWSR and CWSS for reading file set1**

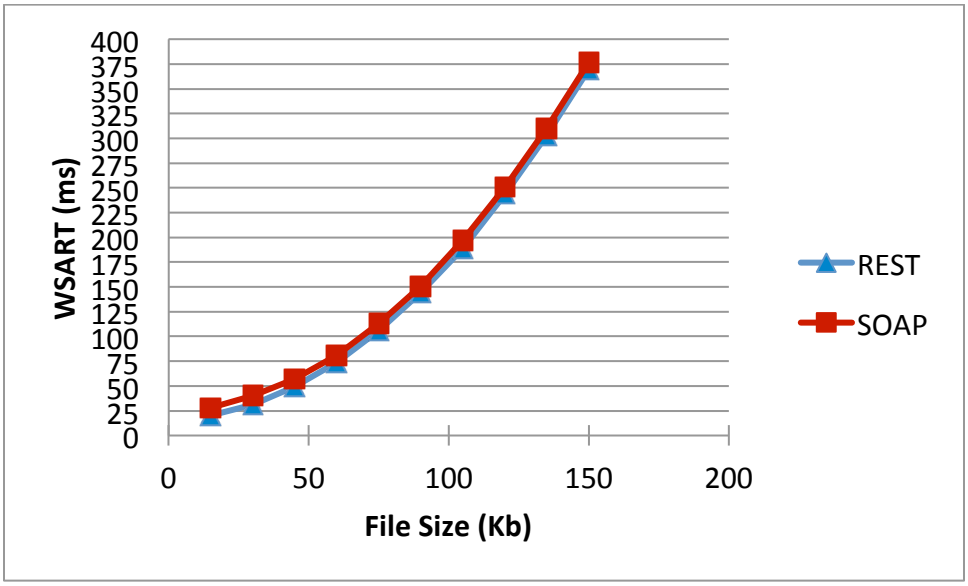


**Figure 5.4: Performance of CWSR and CWSS for reading file set2**

### 5.2.3 Performance Evaluation for Executing Programs on the Computing Resource

The performance evaluation for executing a program on the computing resource exposed as a Web service is presented in this section. The service time is simulated by executing a for loop for a specific interval of time. Experiments are performed with both fixed service times as well exponentially distributed service times. Three different mean service times are used: 25 ms, 50 ms and 100 ms. Figure 5.5 corresponds to the case with fixed service times and Figure 5.6 to the case where the service times are exponentially distributed.

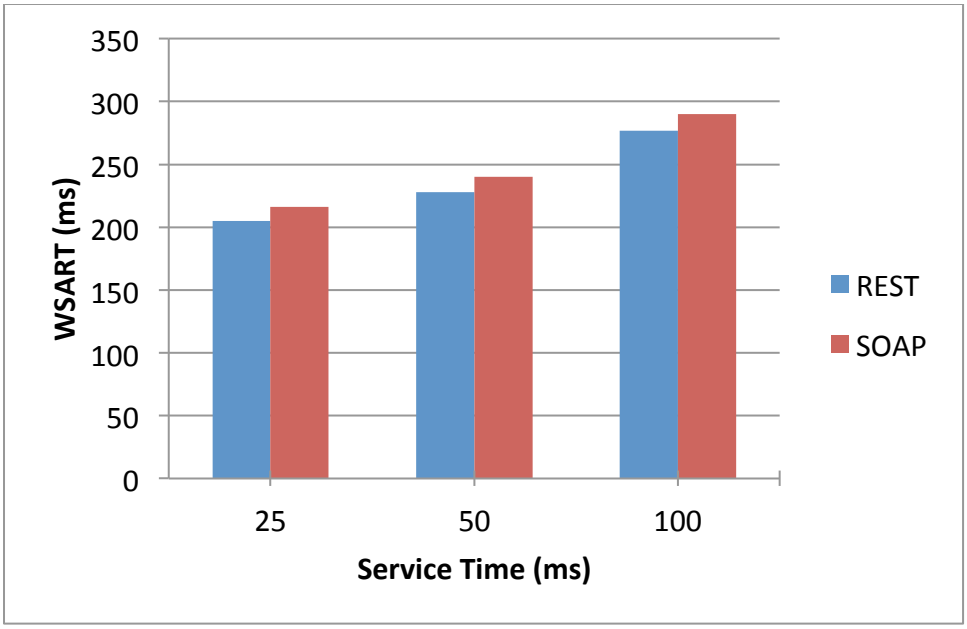


**Figure 5.5: Performance for executing a program on CWSR and CWSS for fixed service times**

The RESTful Web service marginally outperformed the SOAP based Web service in all the cases. When the service time was doubled (from 25 ms to 50 ms) the WSART for both RESTful Web service and SOAP based Web services are increased by 11% (see

Figure 5.5). Figure 5.5 shows that the difference reduces to 5% when the service time is increased from 50 ms to 100 ms. This indicates that the performance difference between the two types of Web services is expected to be smaller for services with higher CPU execution times. Similar performance results are observed for both fixed service times as well exponentially distributed service times (see Figure 5.5 and Figure 5.6).

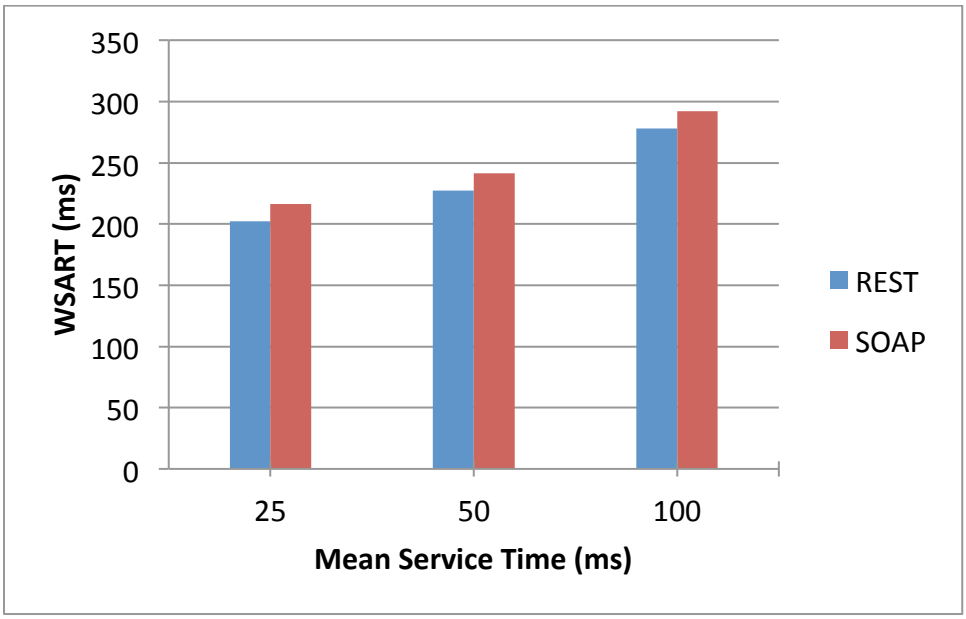


**Figure 5.6: Performance for executing a program on CWSR and CWSS with exponentially distributed service time**

### 5.2.4 Effect of Request Arrival Rates

This section presents an analysis of the effect of request arrival rates on performance. The systems were subjected to a Poisson stream of requests with an arrival rate of λ in (requests/second). Poisson arrival processes have being used to simulate the arrival processes of a real system [4]. If a Web service is not able to serve the request because the Web service is busy with serving the previous request, the new request will be added to a First In First Out (FIFO) queue. The arrival rate is varied from 0.001

requests/sec where negligible queuing of requests occurs to a value at which significant queuing occurs on the system. Three sets of experiments are conducted to observe the performance of CWSR and CWSS while varying the request arrival rates. In the first set, CWSRC and CWSSC invoked Windows commands "dir" and "java – version" (on CWSR and CWSS) for varying request arrival rates. In the second set of experiments are performed with both fixed service times as well exponentially distributed service times (refer to Section 5.2.3) on CWSR and CWSS for changing request arrival rates.

Figure 5.7 shows WSART for Windows command "dir" execution with varying arrival request rates. Figure 5.8 shows WSART for Windows command "java - version" for changing arrival request rates. As λ increases contention on the system increases and WSART for any given type of Web service is observed to increase.

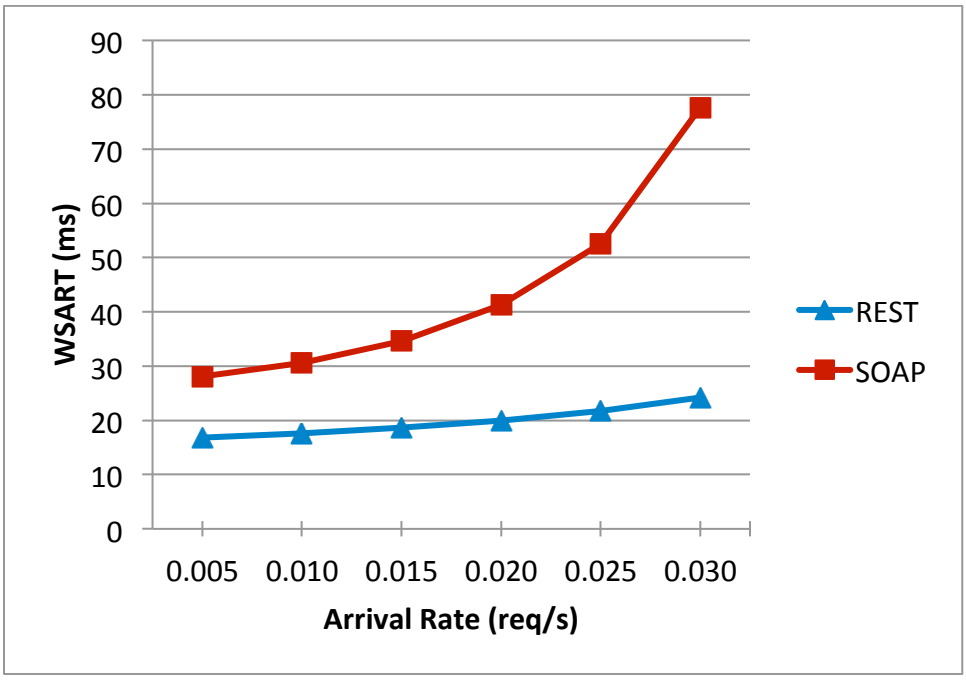


**Figure 5.7: Performance of CWSR and CWSS for Windows command "dir" with different request arrival ratets**

The RESTful Web service has 40% and 69% lower WSART than the SOAP-based service for $\lambda = 0.01$ and $\lambda = 0.03$ respectively (see Figure 5.7). For both Windows commands, the SOAP-based Web service has a higher WSRT than the RESTful Web service. For increasing $\lambda$, the queuing time is higher for SOAP based Web service than the RESTful Web service. Due to these two factors, the RESTful Web service gives rise to a lower WSART than SOAP-based Web service for executing the Windows commands on CWSR and CWSS.

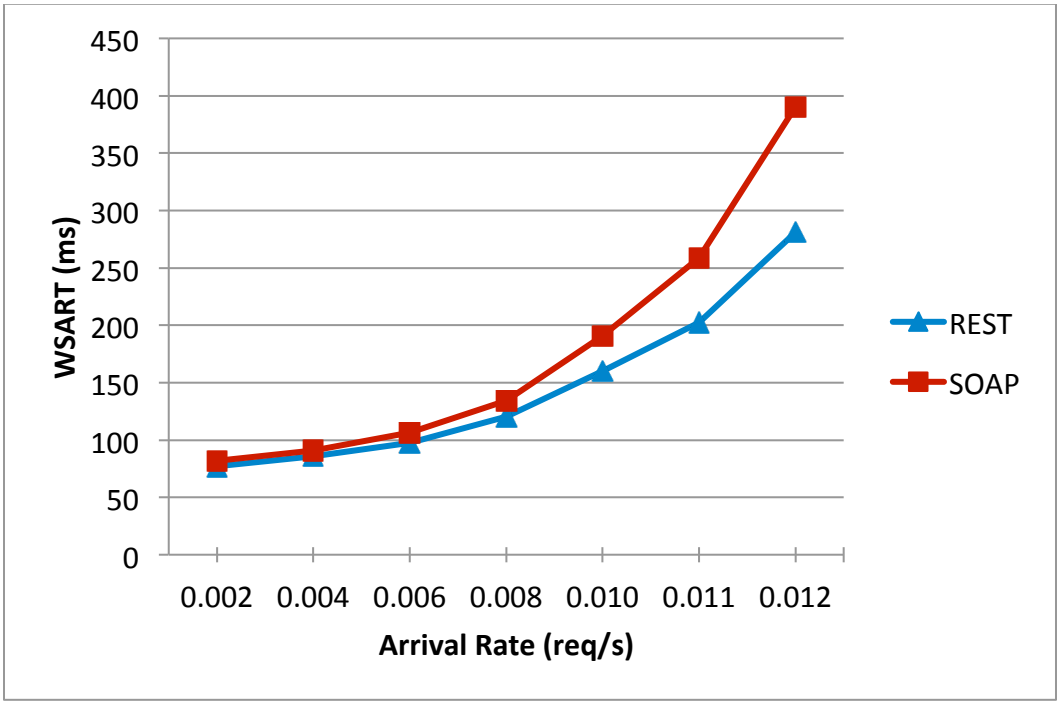


**Figure 5.8: Performance of CWSR and CWSS for Windows command "java -version" with different request arrival rates**

Figure 5.9 to Figure 5.11 show the WSART for fixed service times of 25 ms, 50 ms and 100 ms respectively on CWSR and CWSS for different request arrival rates. Figure 5.12 shows the WSART for an exponentially distributed service time with a mean of 25 ms on CWSR and CWSS for different request arrival rates. As observed in the

Section 5.2.3, there is only a marginal difference in WSART for RESTful Web service and SOAP-based Web service for higher service times and smaller arrival rates. At higher arrival rates, however, the RESTful Web service performs better than the SOAP-based Web service: the performance difference tends to increase as λ increases. When the (mean) service time is 25 ms a higher difference in performance (see Figure 5.9 and Figure 5.12) between the two types of Web services is observed. For example at a λ of 0.005 the performance difference is 54% (see Figure 5.9) and 56% (see Figure 5.12). This indicates that a higher variation in service times produced by the exponentially distribution leads to a higher performance difference.

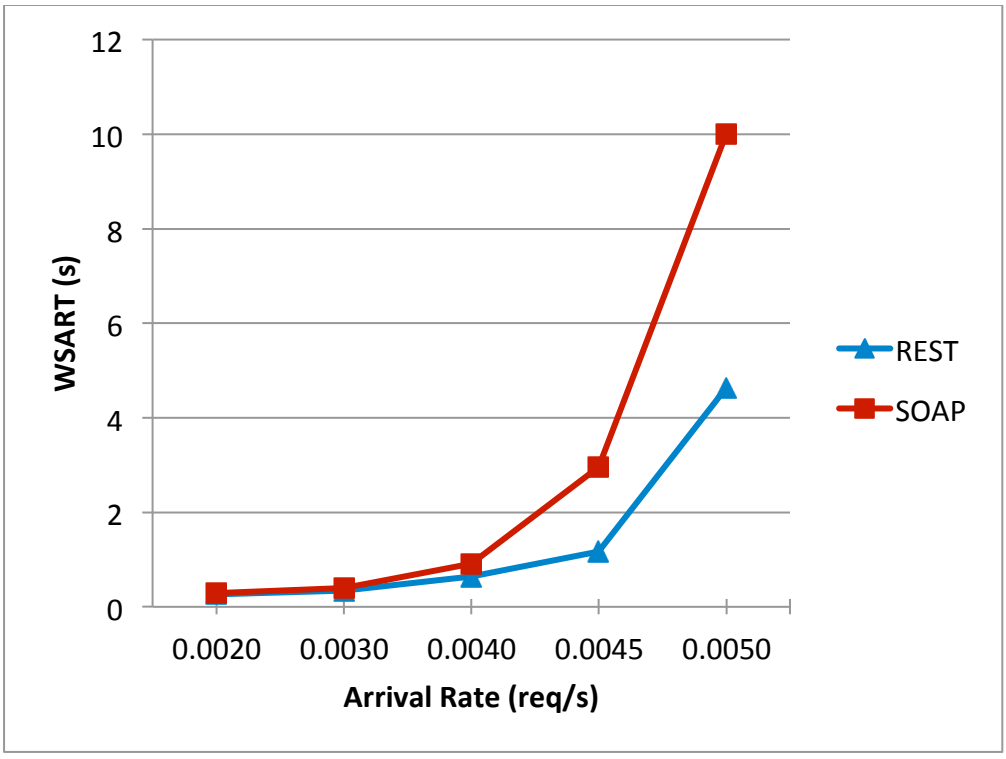


**Figure 5.9: Performance for executing a program on CWSR and CWSS with different request arrival rates (fixed service time of 25 ms)**

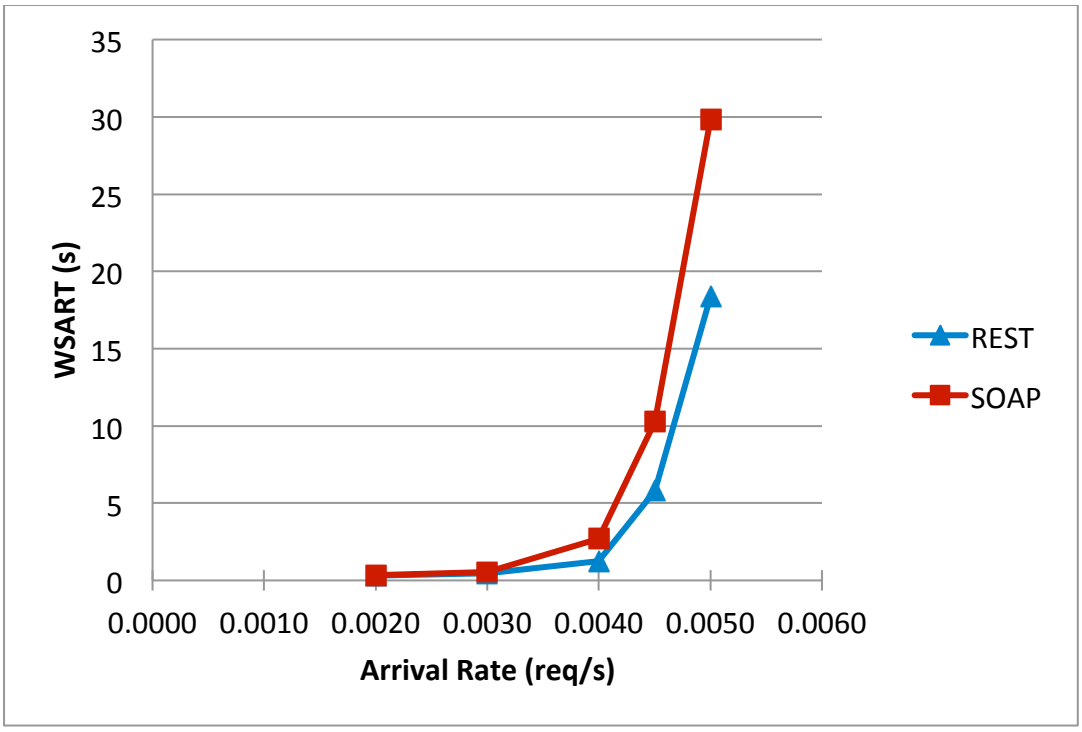


**Figure 5.10: Performance for executing a program on CWSR and CWSS with different request arrival rates (fixed service time of 50 ms)**

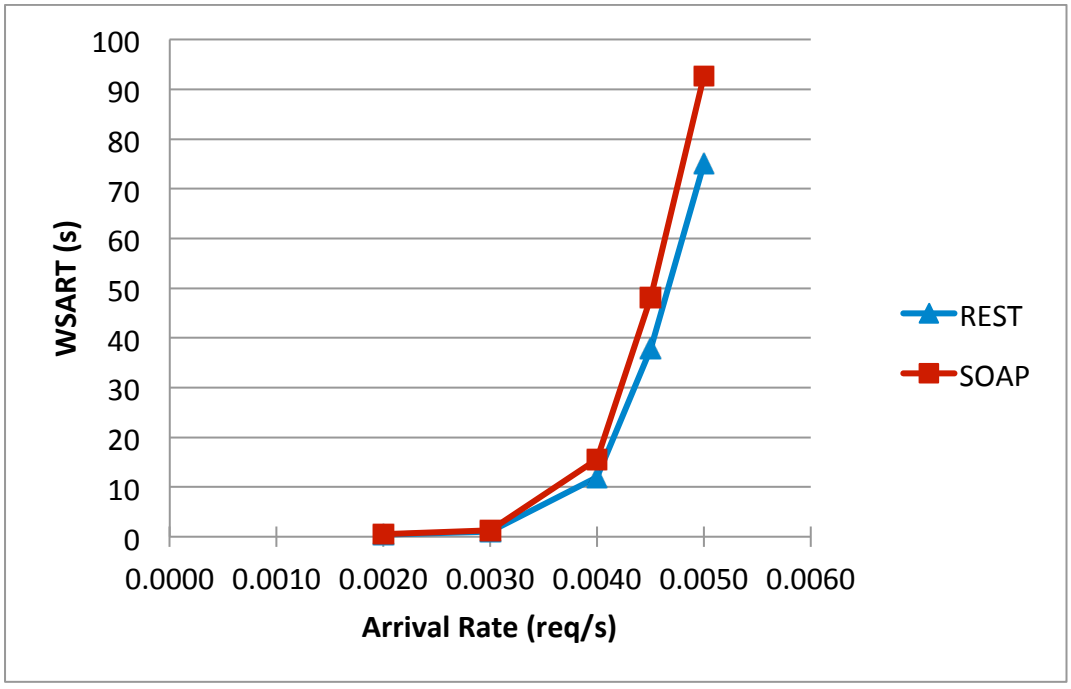


**Figure 5.11: Performance for executing a program on CWSR and CWSS with different request arrival rates (fixed service time of 100 ms)**

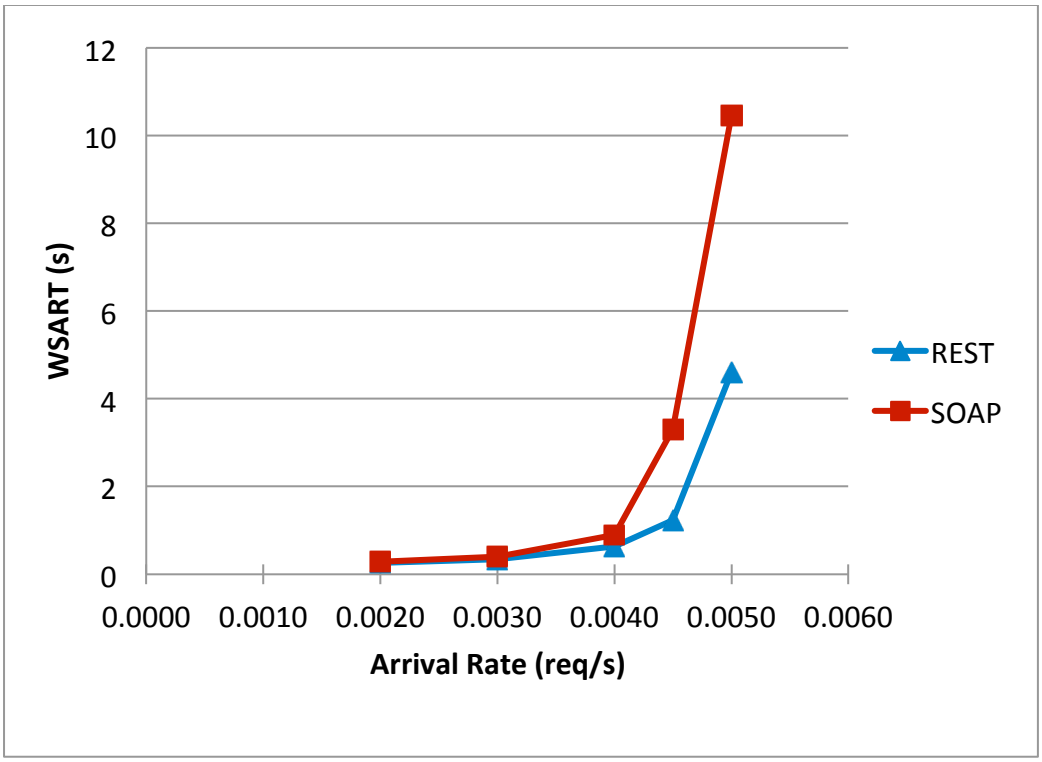


**Figure 5.12: Performance for executing a program on CWSR and CWSS with different request arrival rates (exponentially distributed service time with a mean of 25 ms)**

## 5.3 Performance Evaluation for Exposing a Database Resource as a Web Service

We used both DBWSR and DBWSS (described in Sections 3.2.3 and 3.2.5) to analyze the performance of the database resources exposed as Web services. The clients DBWSRC and DBWSSC invoked DBWSR and DBWSS with the following database operations or queries:

- Insert student record
- Read student record
- Update student record
- Delete student record

### 5.3.1 Performance of Web Service Requests

This section presents the performance comparisons of DBWSR DBWSS for different database queries. Figure 5.13 shows the WSART for the read database operation on DBWSR and DBWSS which Figure 5.14, Figure 5.15 and Figure 5.16 show the WSART for insert, delete and update database operations respectively on DBWSR and DBWSSS.

Figure 5.13 to Figure 5.16 show that WSART achieved with each operation on the RESTful Web service is lower than that achieved on the SOAP-based Web service. The performance for read database operation on DBWSR and DBWSS shows that the WSART for the RESTful is 90% lower than the SOAP-based Web service (see Figure 5.13). For the remaining database operations insert, delete and update, the RESTful Web service achieves a WSART that is lower than that achieved with the SOAP-based Web service by 45%, 51%, and 50% respectively (see Figures 5.14 to Figure 5.16).

A RESTful Web service uses four HTTP methods GET, PUT, POST and DELETE. According to [41] the HTTP GET method is safe, idempotent and free from side effects. Also the HTTP GET method is used for information retrievals. The remaining HTTP methods are capable of modifying data on DBWSR. Read, insert, update and delete database operations use the following HTTP methods: GET, PUT, POST and DELETE respectively. The WSART for the read database operation of RESTful Web service is 88% lower than the update database operation in RESTful Web service (highest WSART in RESTful Web service). This is because the update operation makes changes on the database that are not requires for the read operation. The largest performance difference between the two types of Web services is observed for the read operation: the WSART

for the read database operation on the RESTful Web service is only 1/10 of the WSART required by the read database operation on SOAP-based Web service (see Figure 5.13).

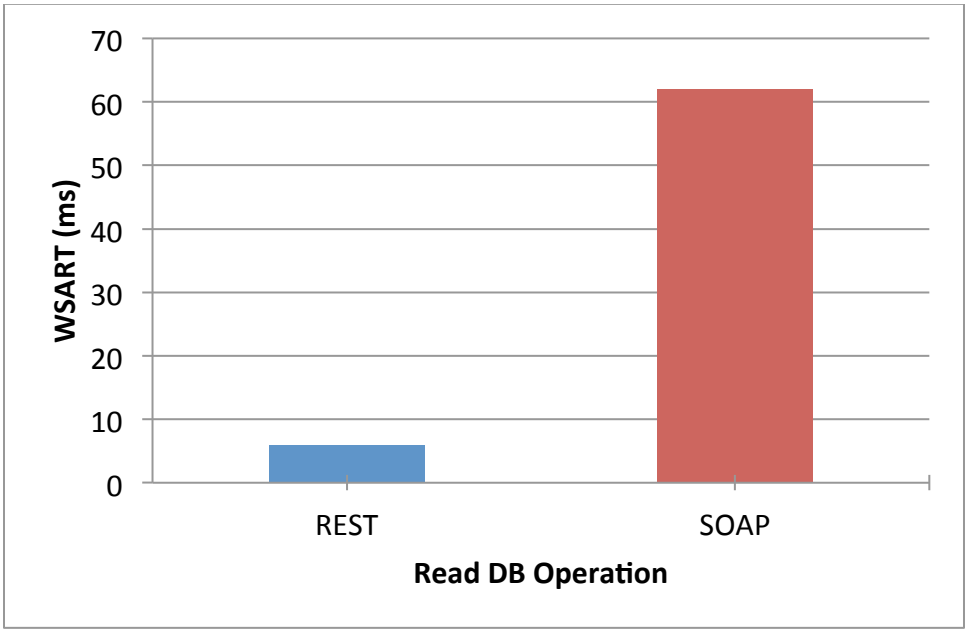


**Figure 5.13: Performance of DBWSR and DBWSS for the read database operation**

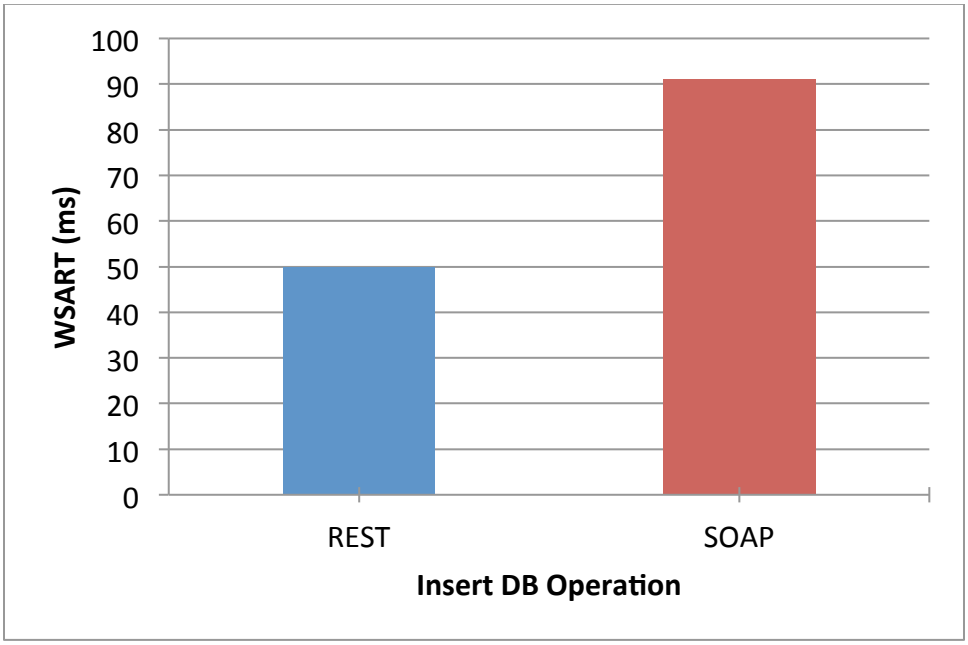


**Figure 5.14: Performance of DBWSR and DBWSS for the insert database operation**

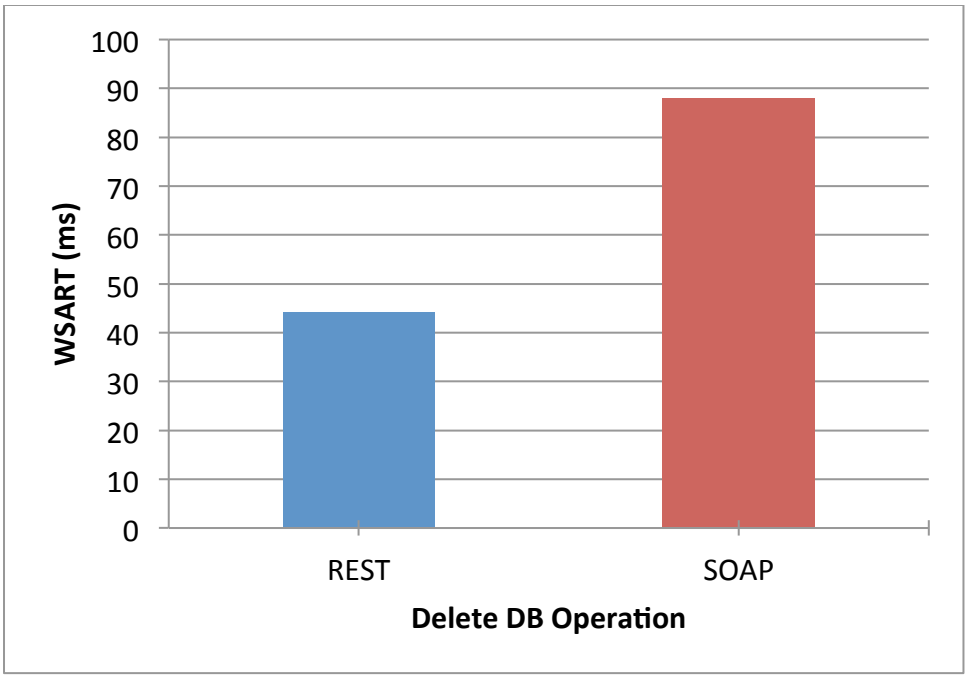


**Figure 5.15: Performance of DBWSR and DBWSS for the delete database operation**

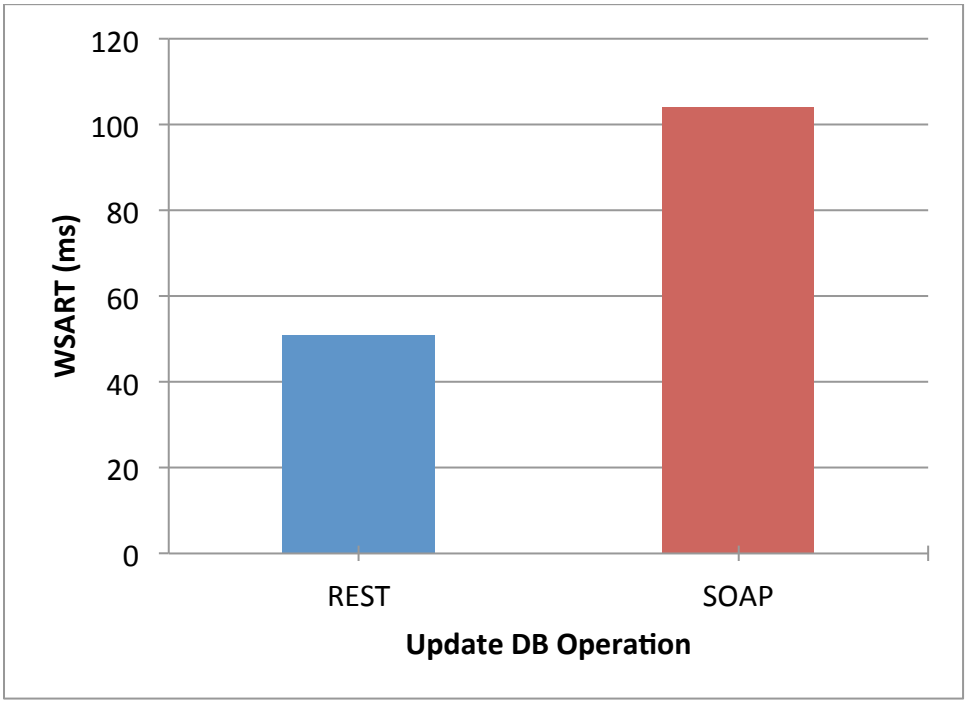


**Figure 5.16: Performance of DBWSR and DBWSS for the update database operation**

### 5.3.2 Effect of Web Service Request Size

The effect of Web service request size on the performance of a database resource exposed as a Web service is discussed in this section. The clients DBWSRC and DBWSSC perform the same database operation on DBWSR and DBWSS respectively with varying number of student records. For each record a separate Web service invocation is made. Figure 5.17 shows the WSART for the read database operation performed on DBWSR and DBWSS with Figure 5.18, Figure 5.19 and Figure 5.20 showing the WSART for insert, delete and update operations performed respectively on DBWSR and DBWSS. The operation is repeated on one record until when the number reach n.

For any given number of records, the WSART for reading student record(s) for the RESTful Web service is much smaller than that that of the SOAP-based Web service (see Figure 5.17). Figure 5.18 to Figure 5.20 display the WSARTs computed for the other database operations. In all the cases, the RESTful Web service has a smaller WSART compared to that of the SOAP-based Web service. Both types of Web services provide the same student information to its client but the size of the message is different. As discussed earlier, for the RESTful Web service the message contains only the student information but in the case of SOAP-based Web service the message contains the student information as well as the XML markup required by SOAP messages. This is the primary reason why the RESTful Web service to gives rises to a lower response time in comparison to the SOAP-based Web service.

As the number of records increases the performance difference between the two types of Web services namely RESTful Web service and SOAP-based Web service is observed to increase (see Figure 5.17 to Figure 5.20).

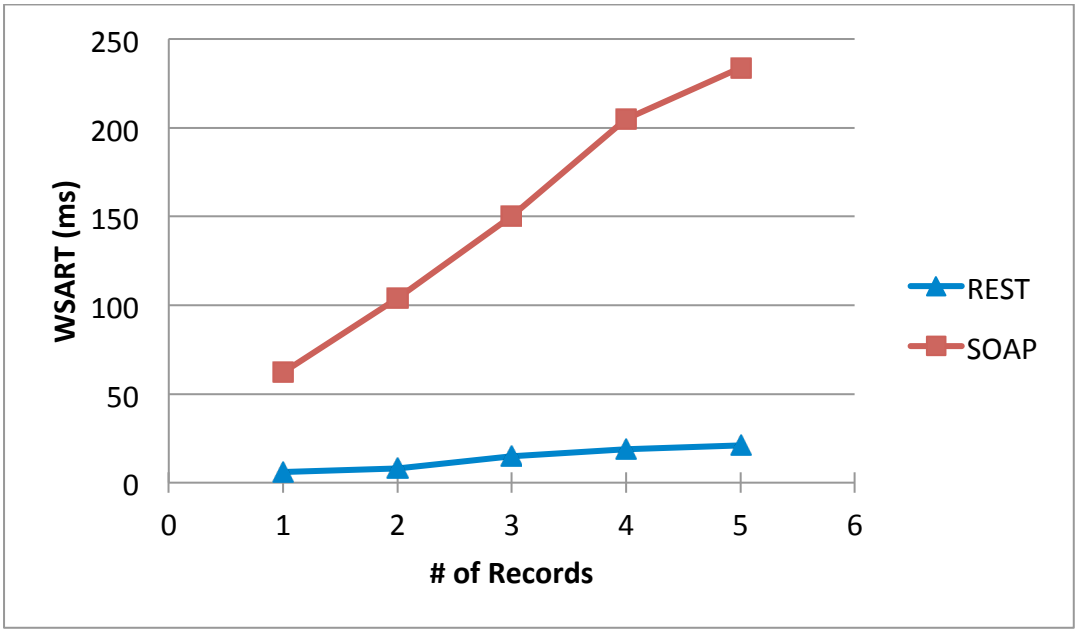


**Figure 5.17: Effect of Web service request size on DBWSR and DBWSS - read student records database operation**

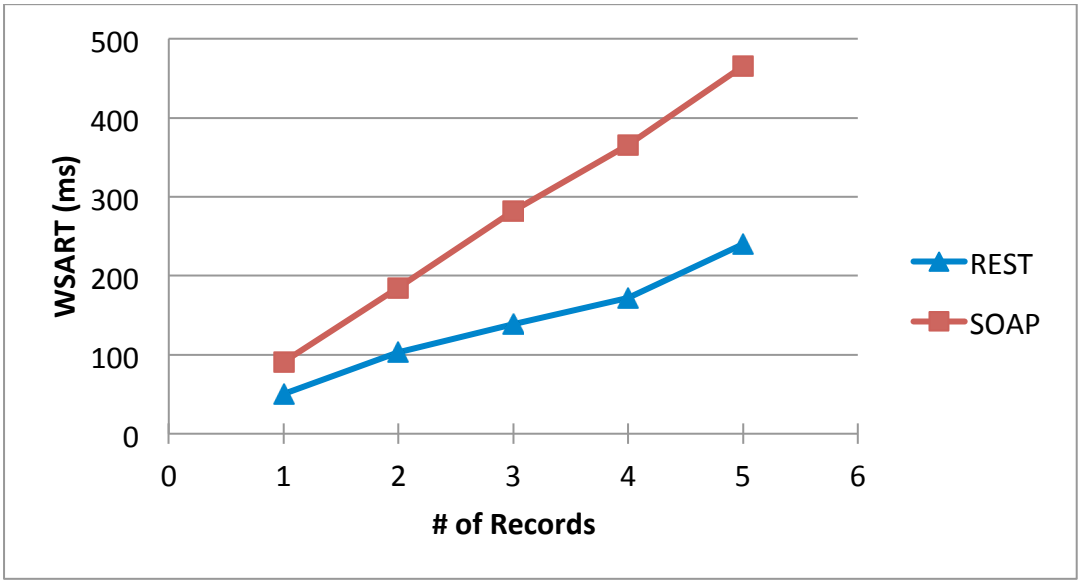


**Figure 5.18: Effect of Web service request size on DBWSR and DBWSS - insert student records database operation**

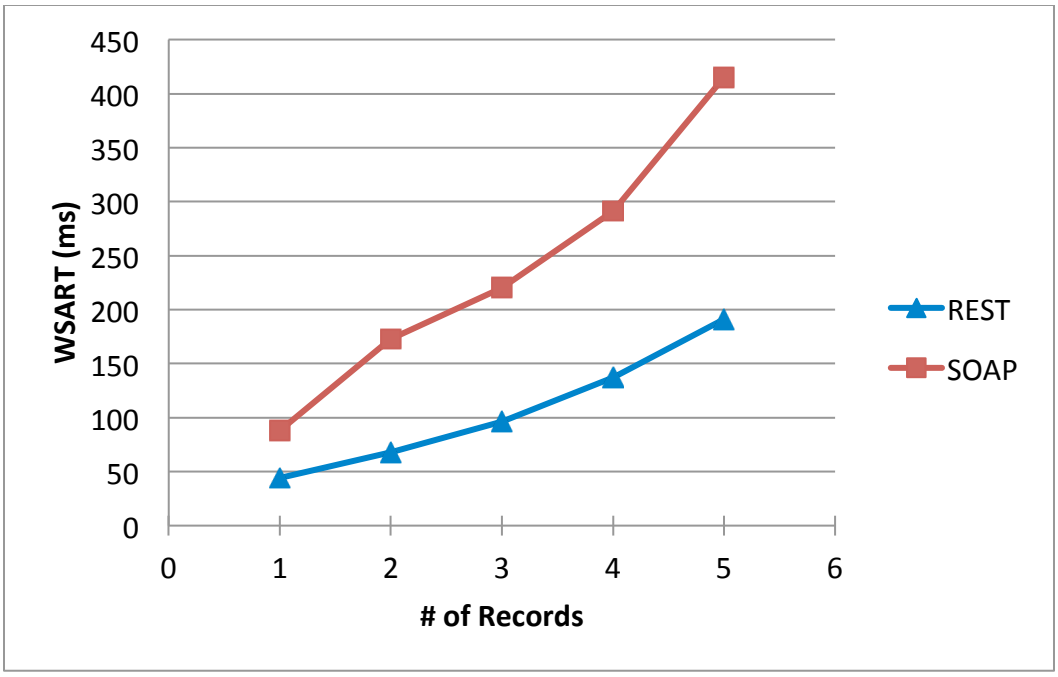


**Figure 5.19: Effect of Web service request size on DBWSR and DBWSS - delete student records database operation**

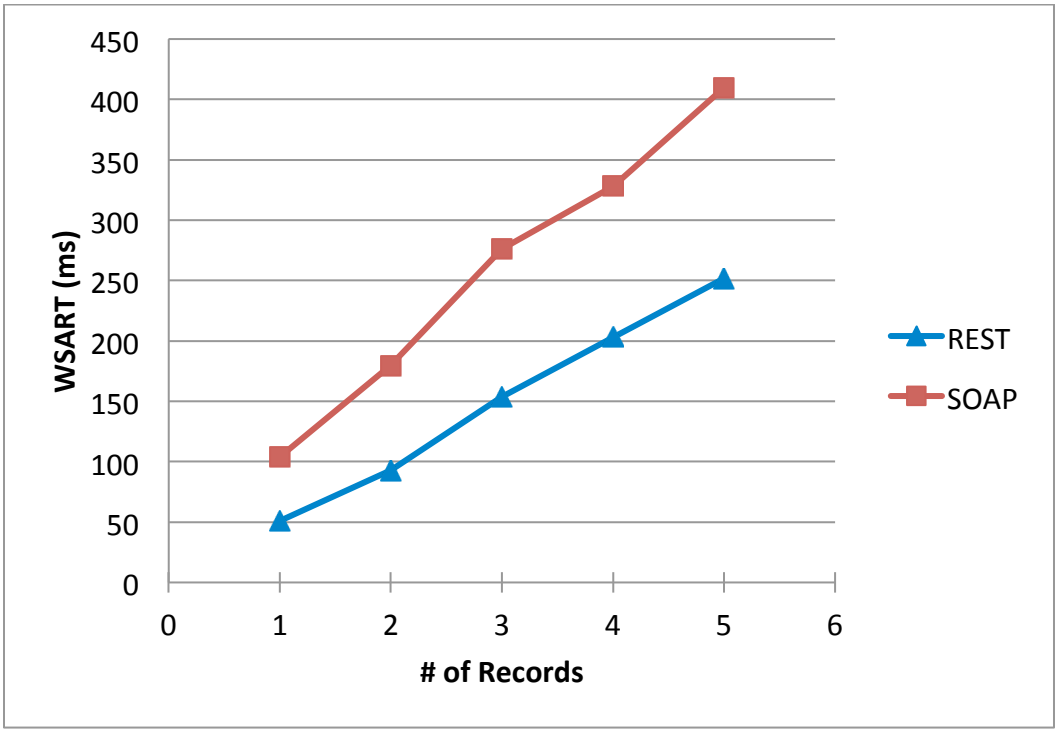


**Figure 5.20: Effect of Web service request size on DBWSR and DBWSS - update student records database operation**

### 5.3.3 Effect of the Arrival Rates of Request

The effect of the arrival rates of request on a database resource exposed as a Web service is presented in this section. The clients DBWSRC and DBWSSC performed different database operations (described in Section 5.3) on DBWSR and DBWSS. The number of student record was held at 1 for all the database queries. Similar technique used in Section 5.2.4 was used for generating the arrival rate of Web service requests.

Figure 5.21 shows the WSART for the update database operation performed on DBWSR and DBWSS for different request arrival rates with Figure 5.22, Figure 5.23 and Figure 5.24 showing the WSART for read, insert and delete operations performed respectively on DBWSR and DBWSS for different request arrival rates. For the update database operation with a request arrival rate of $\lambda = 0.002$ the WSART for RESTful Web service is 46% lower than the SOAP-based Web service. However when $\lambda = 0.009$ the WSART for RESTful Web service is 87% lower than that of the SOAP-based (see Figure 5.21). A similar relative performance was achieved by the two types of Web services with the other database operations (see Figure 5.22 to Figure 5.24). As discussed earlier, for DBWSS the WSART increases with $\lambda$. This is because at the higher queuing delay experienced by the Web requests at higher values of $\lambda$. In all these cases as $\lambda$ increases the queuing delay increases and the performance difference between two types of Web services increases. For a given $\lambda$, the highest WSART is achieved with the update operation, the graph for which is shown in Figure 5.21.

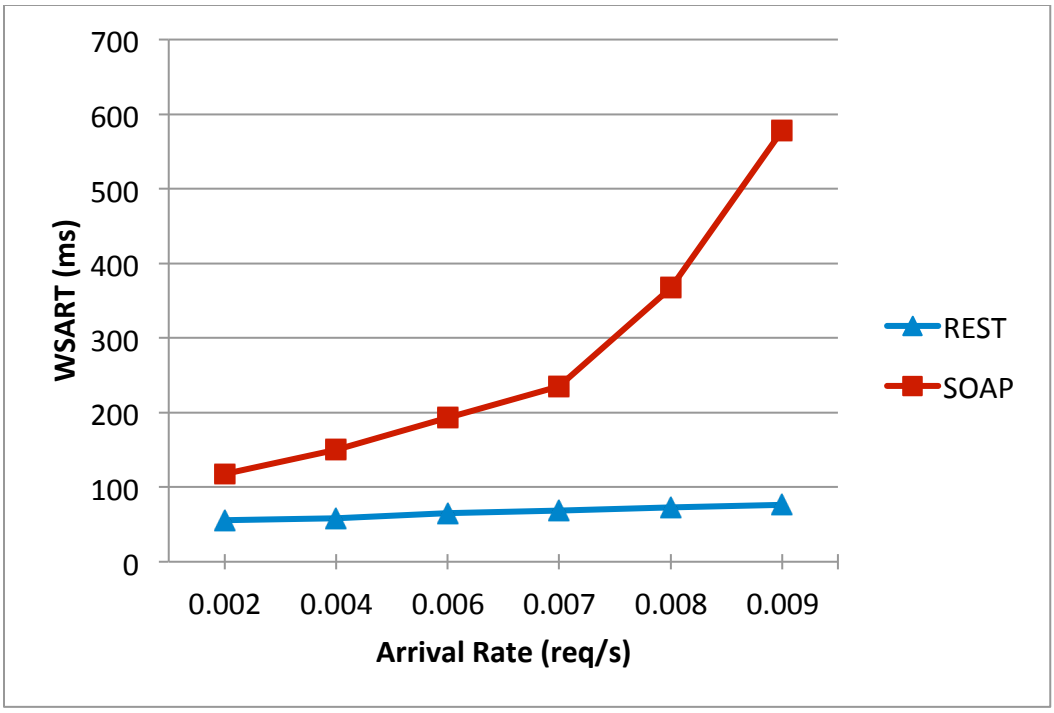


**Figure 5.21: Performance of DBWSR and DBWSS for the update database operation with different request arrival rates**

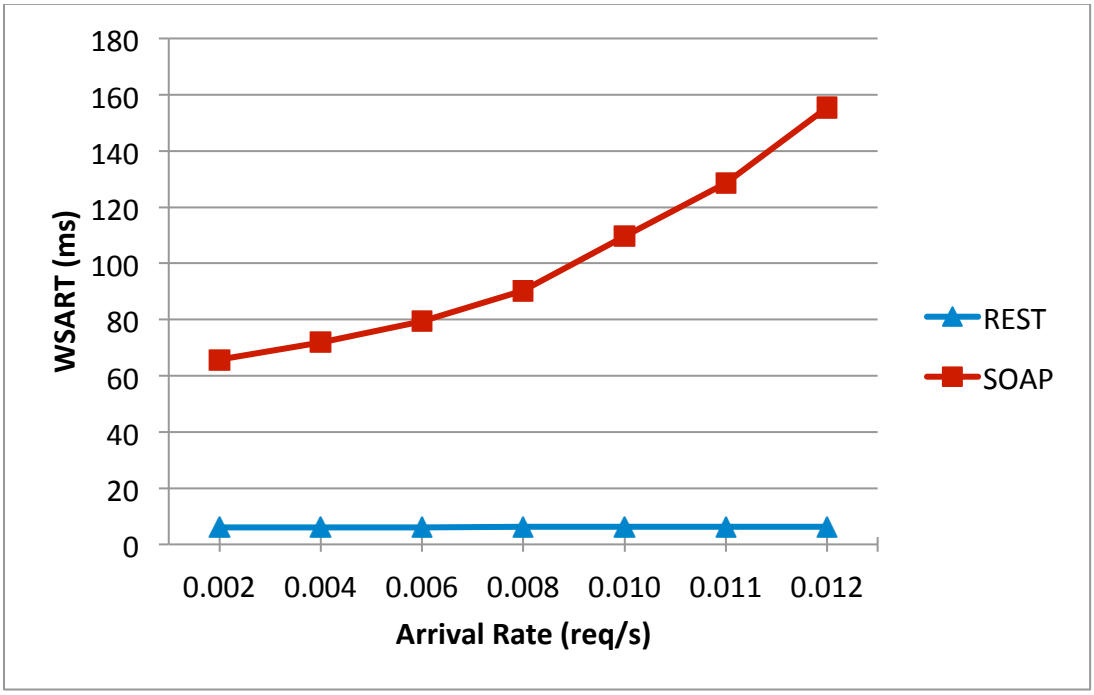


**Figure 5.22: Performance of DBWSR and DBWSS for the read database operation with different request arrival rates**

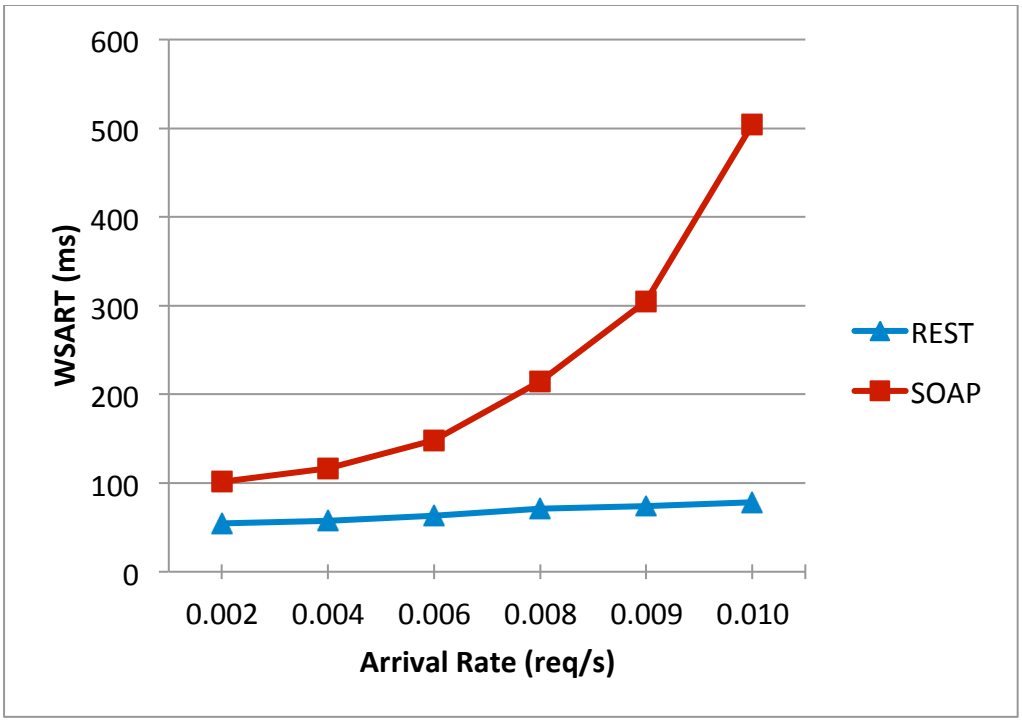


**Figure 5.23: Performance of DBWSR and DBWSS for the insert database operation with different request arrival rates**

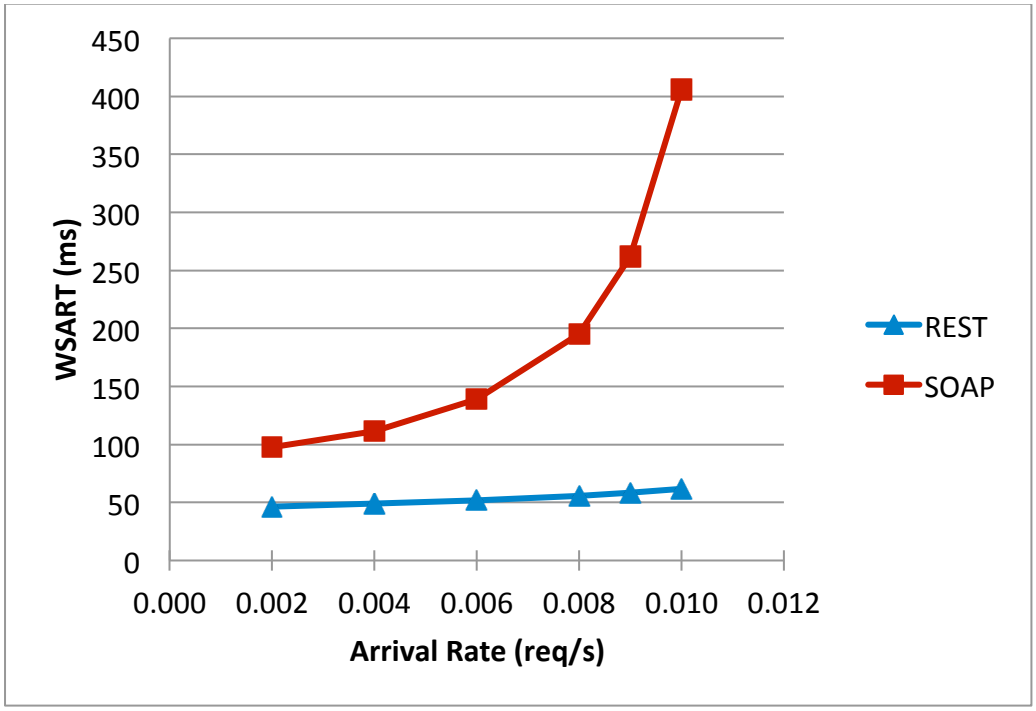


**Figure 5.24: Performance of DBWSR and DBWSS for the delete database operation with different request arrival rates**

## 5.4 Performance Evaluation of a Resource Exposed as a Hybrid Web Service

A new parameter p is introduced to evaluate the performance of the Hybrid Web service. *p* is the probability of invoking the RESTful Web service. $p = 0$ implies that the SOAP-based Web service is always invoked whereas $p = 1$ means that the client always invokes the RESTful Web service.

Two types of performance results are presented. For experiments that used an open stream of request arrivals the values measured on the system were plotted (Figure 5.29 and Figure 5.35 to Figure 5.39). For each of the other experiments used for plotting WSART as a function of *p* (Figure 5.25 to Figure 5.28 and Figure 5.30 to Figure 5.33) two measurements were made: one for the case in which the SOAP-based Web service was invoked and the other for the case in which the RESTful Web service was invoked. These two values were combined using the following equation:

$$WSART = p \times WSART_R + (1-p) \times WSART_S$$

where $WSART_R$ and $WSART_S$ are the Web service average response time measured for the RESTful Web service and the SOAP-based Web service respectively.

### 5.4.1 Performance of Computing Resource Exposed as a Hybrid Web Service

The performance for executing different Windows commands on a computing resource exposed as Hybrid Web service is discussed in this section. CHWSC uses the Window commands that were described earlier (see Section 5.2). CHWSC explicitly selects the RESTful or a SOAP-based technology irrespective of the type of operations is used in all the experiments.

Figure 5.25 shows the WSART as a function of p for executing Windows command "dir" on CHWS and Figure 5.26 shows the WSART as a function of *p* for executing Windows command "java - version" on CHWS. As shown in Figure 5.25 as *p* increases the performance improvements achieved by the Hybrid Web service increases sharply. This is because as *p* increases the lightweight RESTful Web service gets used more often and system performance improves. System performance is observed to increase sharply with *p*: WSART is observed to decrease by 38% as p is increased from 0 to 1. Even when the RESTful Web service is used 20% of the time (p=0.2) there is a substantial improvement in WSART achieved by the Hybrid Web service in comparison to the conventional SOAP-based Web service (see Figure 5.25).

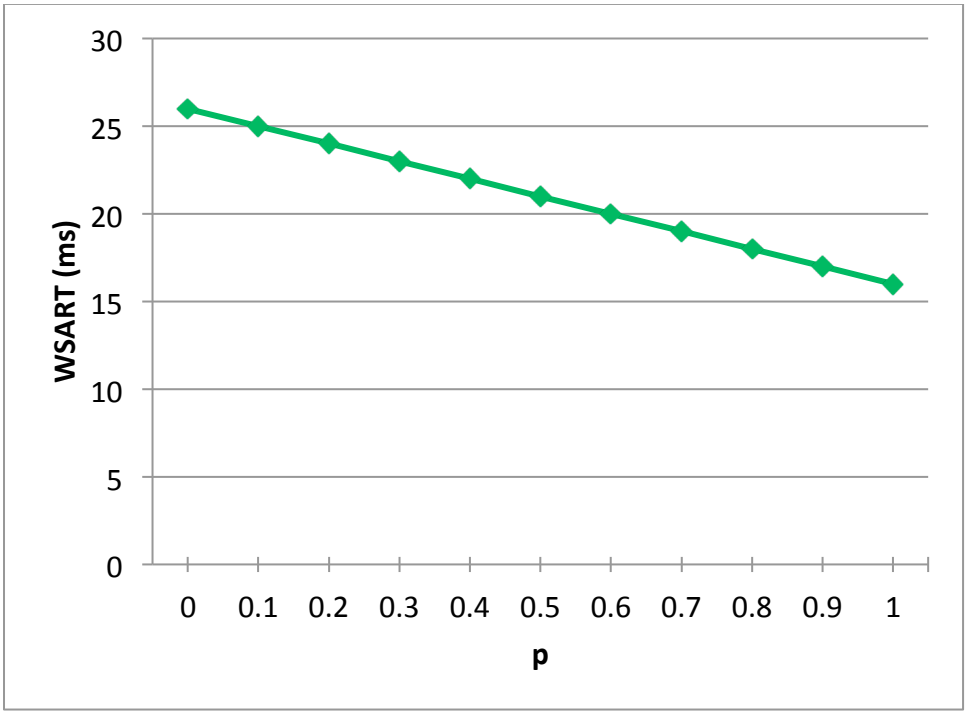


**Figure 5.25: Performance for executing the Windows command "dir" on CHWS**

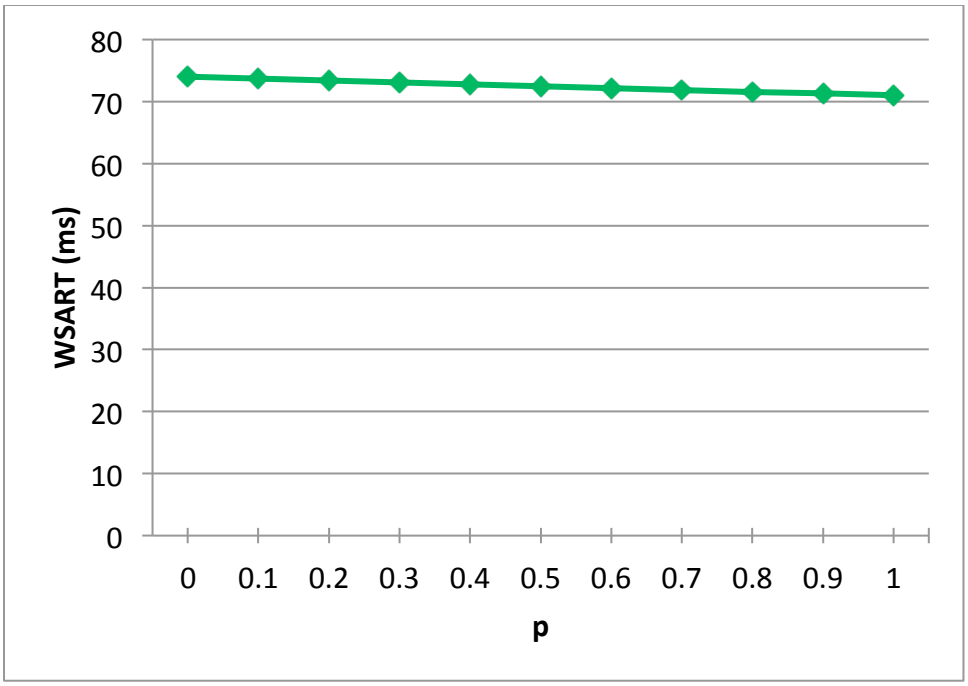


**Figure 5.26: Performance for executing the Windows command "java -version" on CHWS**

### 5.4.2 Performance for Executing Programs on Computing Resource Exposed as a Hybrid Web Service

This section presents an analysis of executing programs on the computing resource exposed as a Hybrid Web service. CHWSC executes programs with various service times on CHWS for various values of *p*. As described earlier CHWSC explicitly selects the RESTful or a SOAP-based technology then executes the programs on CHWS.

Figure 5.27 shows the WSART as a function of *p* for executing programs with fixed service times of 25 ms, 50 ms and 100ms on CHWS and Figure 5.28 shows WSART as a function of *p* for executing programs with the exponentially distributed service time with a mean of 25 ms. The WSART for executing the program (exponentially distributed

service time) with different arrival rate for requests is shown in the Figure 5.29. WSART seems to be dominated by the CPU execution time overhead and the overhead from Web service technology play only a minor role and a comparable performance is achieved for both the technologies. WSART is observed to decrease by 6% for both fixed and exponentially distributed service times a mean of 25 ms as *p* is increased from 0 (SOAP-based only) to 1 (RESTful only) (see Figure 5.27 and Figure 5.28). From Figure 5.29 it can been seen the WSART increases with λ for both Web service technologies due to the higher queuing delay experienced by the Web requests at higher values of λ.

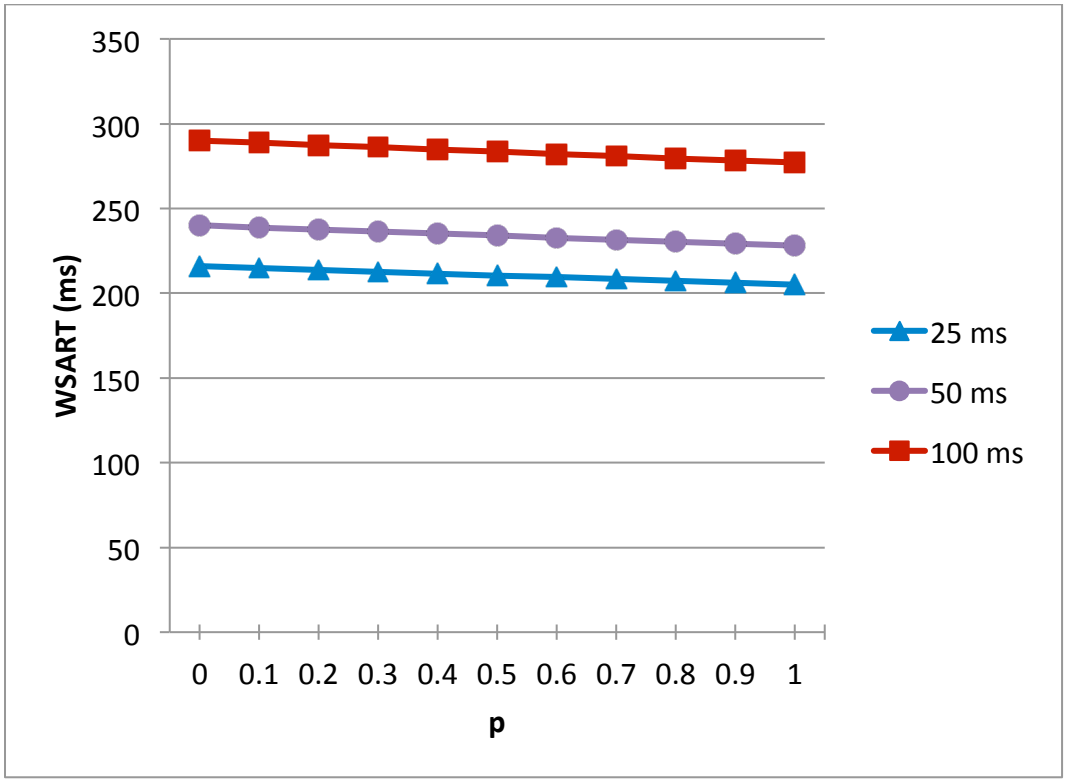


**Figure 5.27: Performance for fixed service times (25ms, 50ms and 100ms service time)**

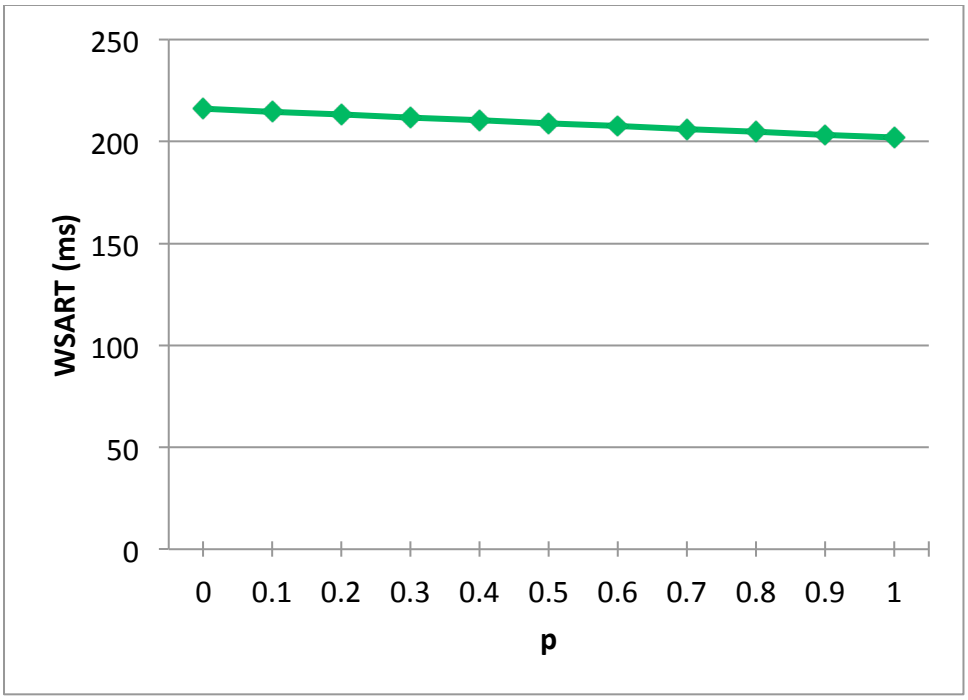


**Figure 5.28: Performance for an exponentially distributed service time (mean 25 ms)**

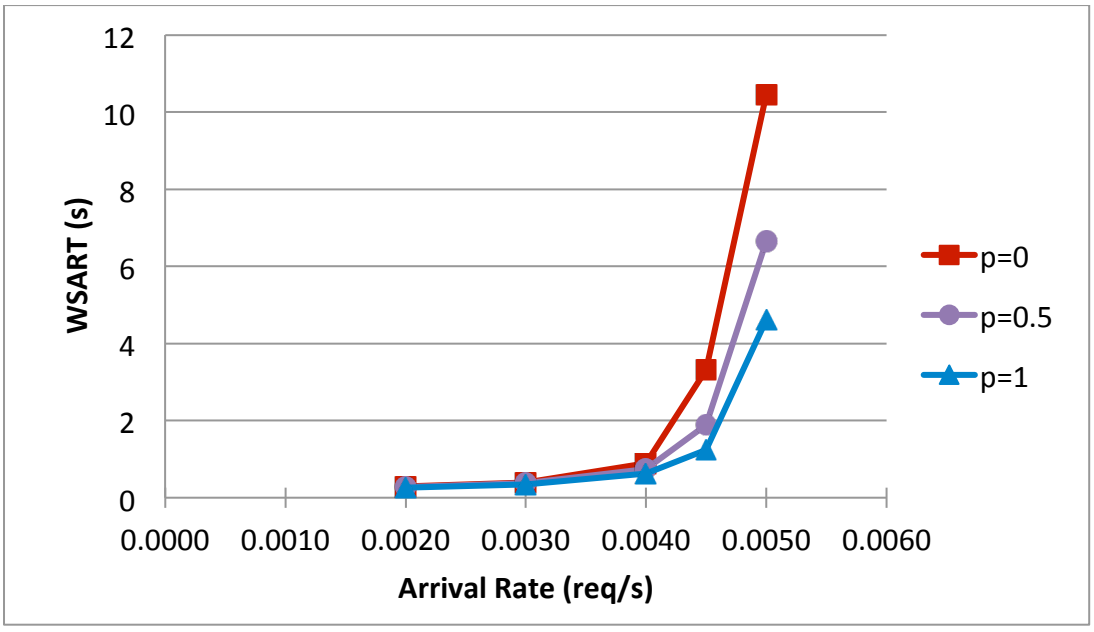


**Figure 5.29: Performance for exponentially distributed service times (mean 25 ms) with different request arrival rates**

### 5.4.3 Performance of Database Resource Exposed as a Hybrid Web Service

This section presents an analysis of the performance of different database operations perform on the database resource exposed as a Hybrid Web service. DBHWSC uses the same set of database operations or queries (described in Section 5.3) to invoke the DBHWS. The number of student record was held at 1 for all the database queries.

DBHWSC invokes the DBHWS in two ways. In the first by requesting an operation (described earlier) where the client explicitly selects the SOAP-based or RESTful technologies irrespective of the type of operations is used in all the experiments (DBHWS - Type I). In the second, DBHWS client requests for the service that in turn invokes the RESTful Web service or the SOAP-based Web service depending on the operation performed by the DBHWSC (DBHWS - Type II). As discussed in Section 4.2.3 the read database operation invokes the RESTful Web service while the insert, update and delete operations invoke the SOAP-based Web service.

Figure 5.30 shows the WSART as a function of $p$ for the read database operation performed on DBHWS - Type I with Figure 5.31, Figure 5.32 and Figure 5.33 showing the WSART as a function of $p$ for insert, update and delete database operations performed respectively on DBHWS - Type I.

As shown in Figures 5.30 to Figure 5.33, as $p$ increases, the RESTful Web service is used more by clients and performance improvements achieved by the Hybrid Web service increases sharply. The WSART for read database operation is decreased by 90% as $p$ is increased from 0 to 1 (see Figure 5.30). Similarly the WSART for insert, update, delete and database operations decreased by 45%, 51% and 50% respectively for

increasing *p* from 0 to 1.

As observed in Figure 5.30 to Figure 5.33, the performance improvement is higher for the read database operation as *p* is changed from 0 from 1. The RESTful Web service's message contains only the student information but for the SOAP-based Web service the message contains the student information as well as the XML markup required by SOAP messages. As observed with the database resource exposed as a Web service for intermediate *p* values (between 0 and 1) the Hybrid Web service produces a substantially higher system performance in comparison to the pure SOAP-based Web service for all the database operations.

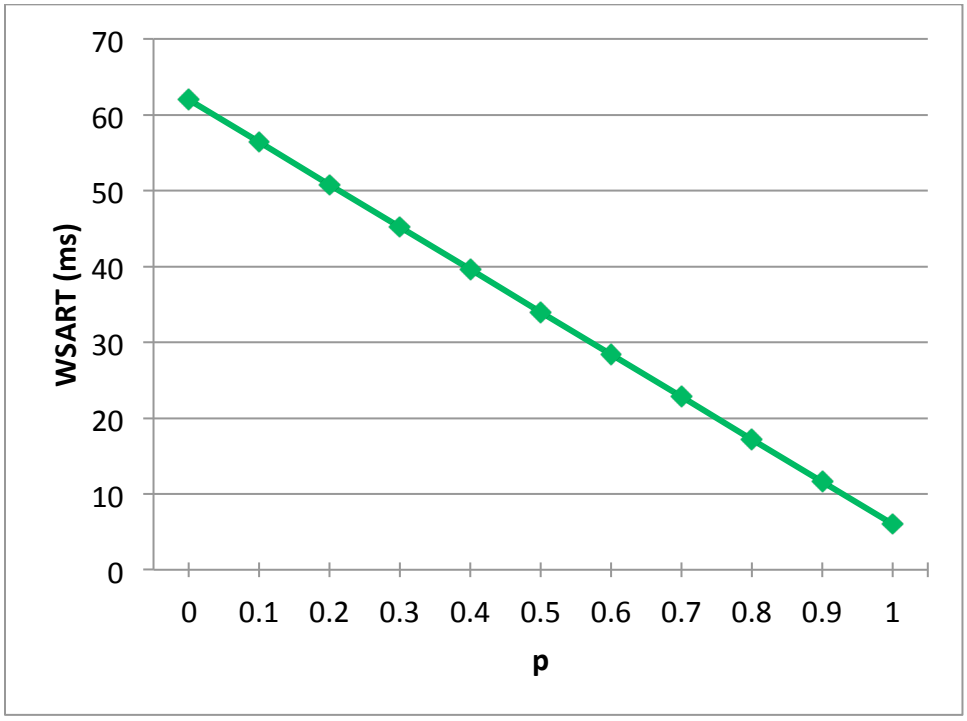


**Figure 5.30: Performance for the read database operation performed on DBHWS - Type I**

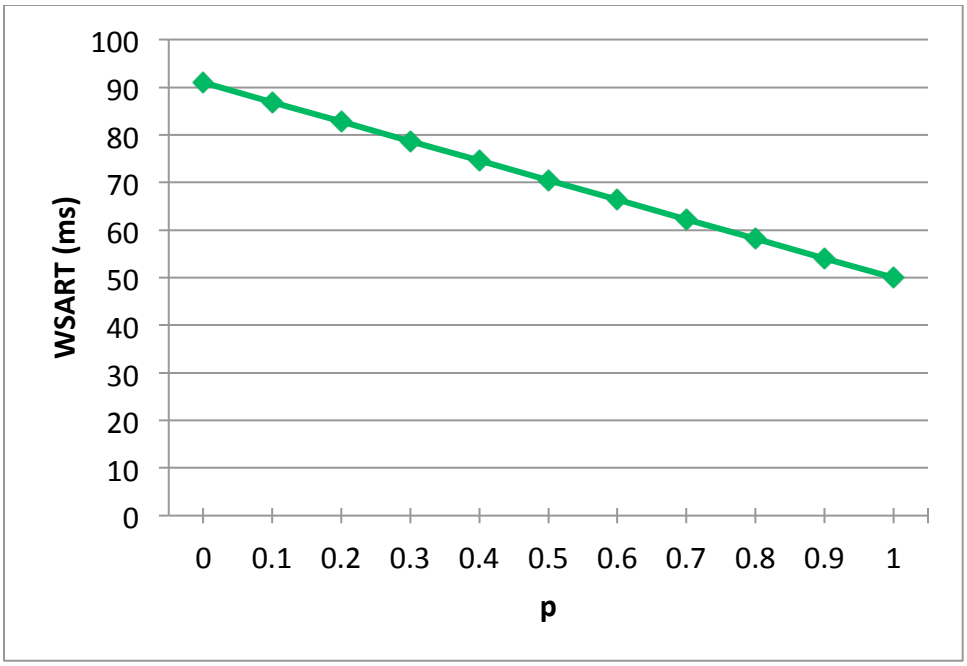


**Figure 5.31: Performance for the insert database operation performed on DBHWS - Type I**

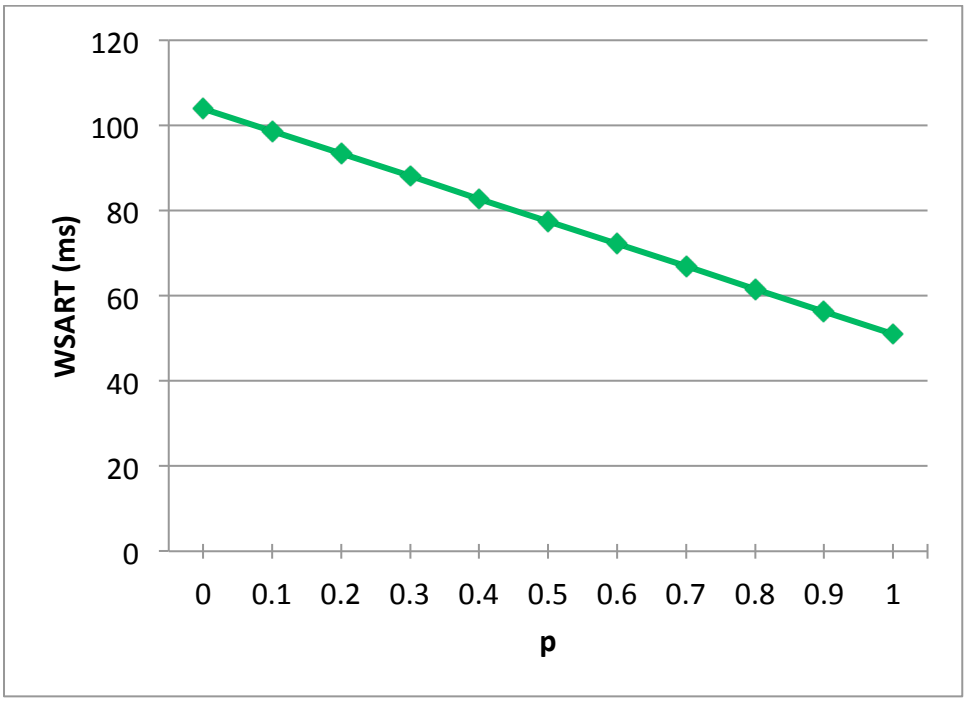


**Figure 5.32: Performance for the update database operation performed on DBHWS - Type I**

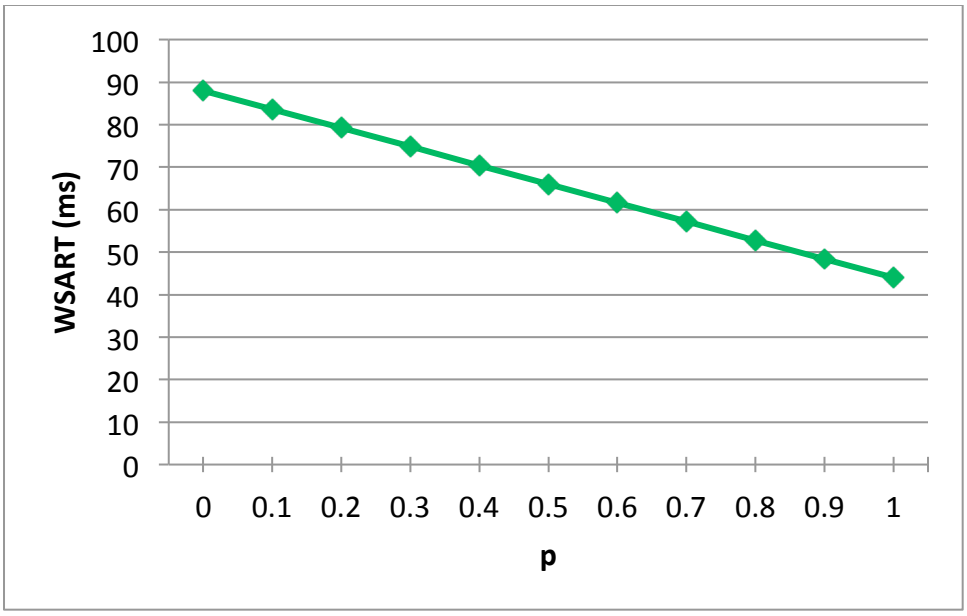


**Figure 5.33: Performance for the delete database operation performed on DBHWS - Type I**

Figure 5.34 shows the WSART for different database operations performed on DBHWS - Type II.

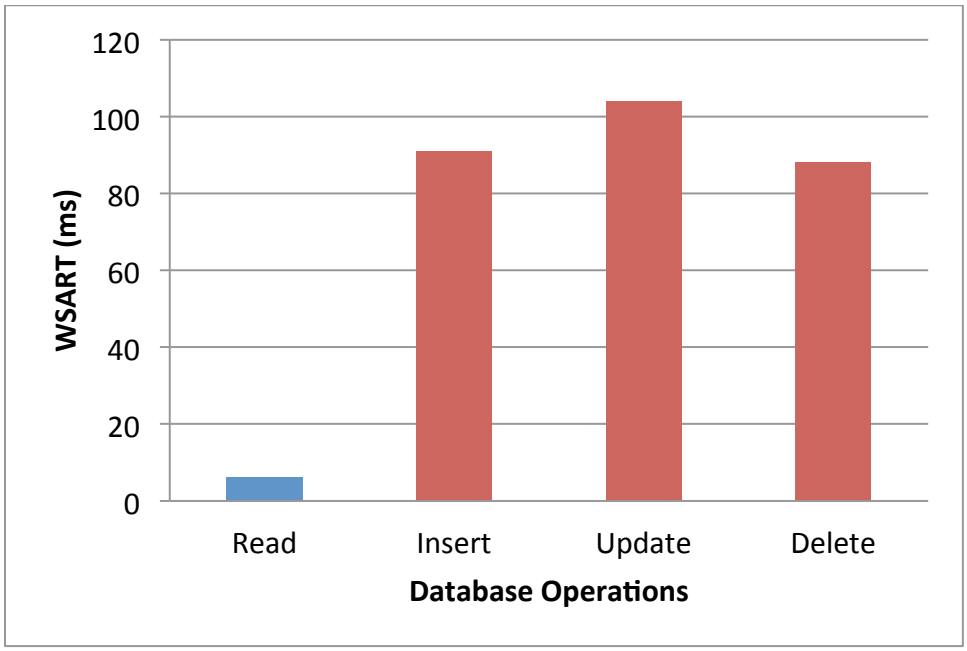


**Figure 5.34: Performance of DBHWS - Type II for different database operations**

As described earlier the read database operation invokes the DBWSR component of DBHWS. The rest of the database operations invoke the DBWSS component of DBHWS. The smallest WSART is observed for the read database operation (see Figure 5.34).

### 5.4.4 Effect of Request Arrival Rate

This section presents the effect of the arrival rate for Web service requests on the performance of the Hybrid Web service. Once again, the number of student record is held at 1 for all the database queries. Three p values 0 (SOAP-based only), 0.5(SOAP-based and RESTful) and 1 (RESTful only) were chosen for this experiments while varying the Web service request arrival rates. Techniques used similar to that outlined in Section 5.2.4 has been used for generating the arrival rates for Web service requests.

Figure 5.35 shows WSART for the read database operation performed on DBHWS - Type I for different arrival rate of requests. Figure 5.36, Figure 5.37 and Figure 5.38 show WSART for insert, update and delete database operations performed respectively on DBHWS - Type I for different arrival rate of requests.

As observed with all database operations for high $\lambda$ the WSART for SOAP-based Web service ($p = 0$) is significantly higher than that achieved for a Hybrid Web service ($p = 0.5$). For example, the read database operation with $\lambda = 0.012$ has a 68% lower WSART for the Hybrid Web service ($p = 0.5$) than that for the pure SOAP-based Web service (see Figure 5.35). At values of $\lambda$ higher than those used in Figure 5.35 to Figure 5.38 a sharper increase in WSART is expected for each system. A higher performance

difference between the Hybrid and the SOAP-based Web service and between the Hybrid and the RESTful Web service is expected.

The Hybrid Web service (with p = 0.5) is observed to behave similarly for other database operations (see Figure 5.36, Figure 5.37 and Figure 5.38). The Hybrid Web service comprises of RESTful Web service and SOAP-based Web service. The WSRT for RESTful Web service is lower than that of the SOAP-based Web service which also leads to a smaller queuing delay for the Hybrid Web service in comparison to the SOAP-based Web service. With more queuing occurring at higher values of λ, the performance advantage of the Hybrid Web service increases with an increase in λ.

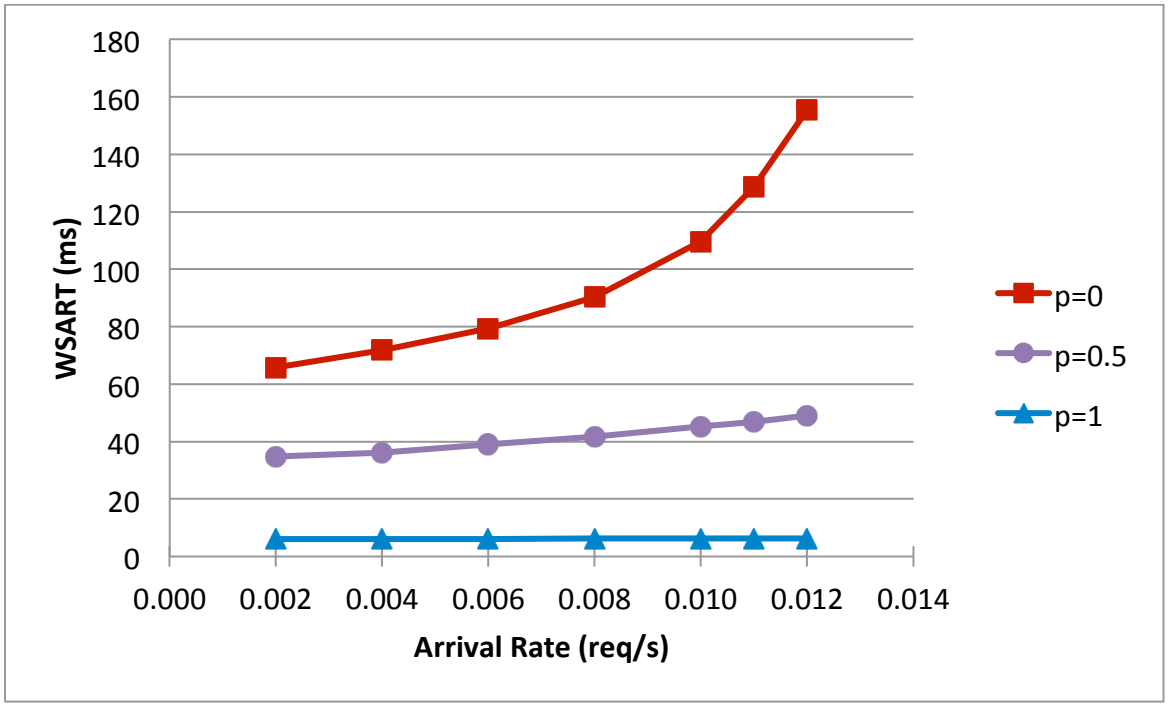


**Figure 5.35: Performance of DBHWS - Type I for the read database operation with different request arrival rates**

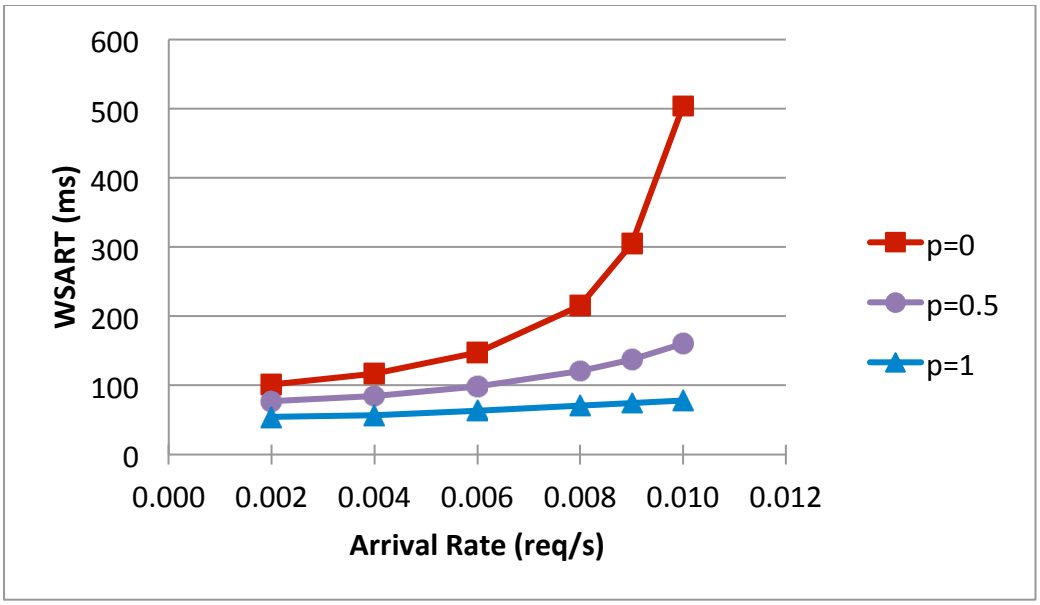


**Figure 5.36: Performance of DBHWS - Type I for the insert database operation with different request arrival rates**

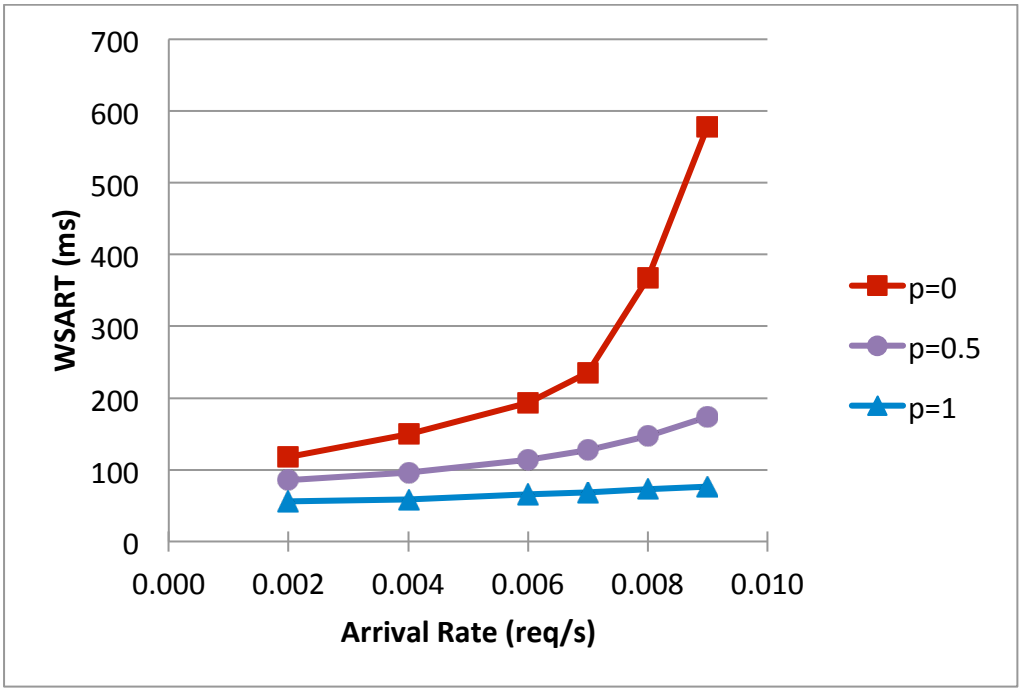


**Figure 5.37: Performance of DBHWS - Type I for the update database operation with different request arrival rates**

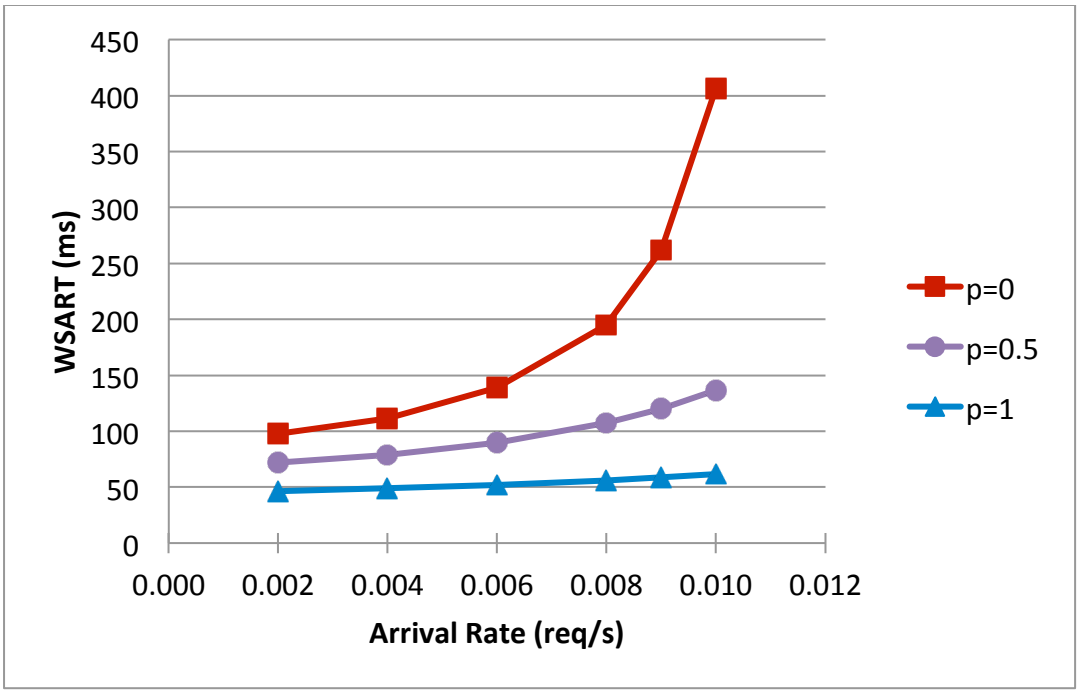


**Figure 5.38: Performance of DBHWS - Type I for the delete database operation with different request arrival rates**

Figure 5.39 shows the WSART for different database operations performed on DBHWS - Type II for different request arrival rates.

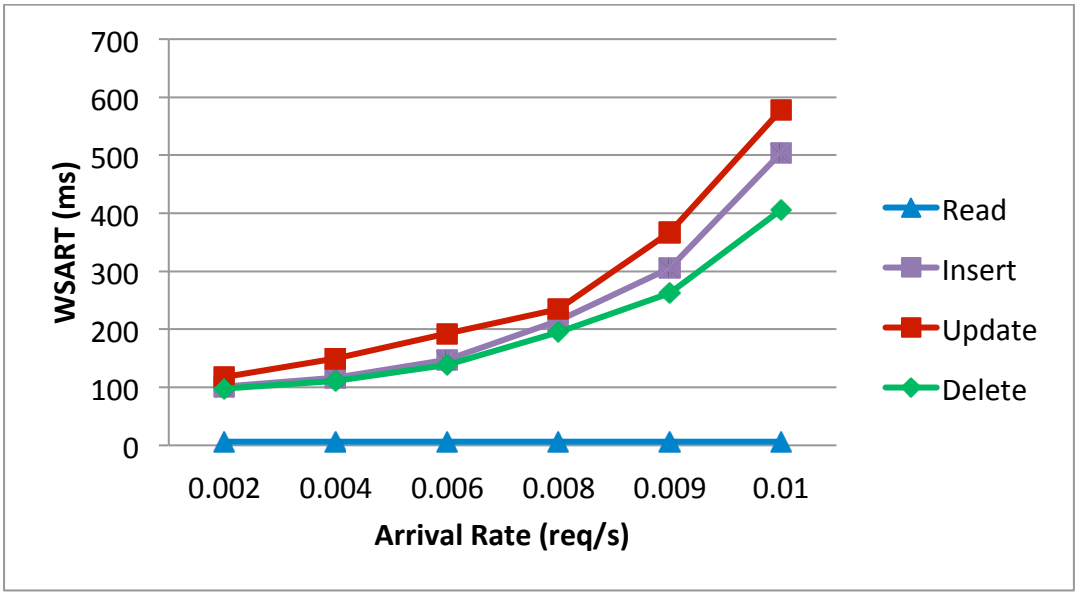


**Figure 5.39: Performance of DBHWS - Type II for different database operations with different request arrival rates**

Note that the RESTful Web service is always used for the read database operation and the SOAP-based Web service for all the other database operations. WSART increases with λ and the highest WSART is achieved for the update database operation on DBHWS - Type II (see Figure 5.39).

## 5.5 Discussions of Experimental Results

This section summaries a set of key observations for the performance results for exposing computing and database resources as Web services. There were three types of systems used for the performance analysis: computing resource exposed as a Web service (both SOAP-based and RESTful), database resource exposed as a Web service (both SOAP-based and RESTful) and computing and database resources exposed as a Hybrid Web service.

It can be observed from the performance results that the RESTful Web service technology outperformed the SOAP-based Web service technology in the first two systems. However, the level of performance difference between two technologies varied from one system to another.

The RESTful Web service performed significantly better than that of the SOAP-based Web service for exposing database resource as Web services. The read database operation achieved maximum performance improvement with RESTful Web service. The WSART was observed to be lower by as much as 90 % than that for the SOAP-based Web service (see Section 5.3.1).

The performance advantages of the RESTful Web services technology over the SOAP-based Web service technology are small for communication bound systems

characterized by a payload comprising large file size. On the other hand, when a payload comprises a small file then the difference between the performances achieved by the RESTful Web services and SOAP-based Web services increases. This behavior is aptly captured in Figure 5.13 that corresponds to a system in which small database records are read. The performance difference between the two Web service technologies is also observed to increase with an increase in arrival rate.

However, in a system exposing a computing resource as a Web service, the performance gain achieved by the RESTful Web service is marginal especially for higher service times. Although the RESTful Web service performs better than the SOAP-based Web services for each operation, both technologies demonstrate a comparable performance for a system that is compute bound.

As discussed in Section 1.3, one of the questions that motivated this research concerns the determination of the system and workload parameters for which a RESTful Web service technology provides a significant performance benefit over SOAP-based Web service. Experiments were performed to determine such parameters. In one experiment the CPU execution on a computer resource exposed as a Web service was varied and the CPU execution time that produced a performance difference of 5% and 2% were determined. Similarly, the length of the file transferred on a computer resource exposed as a Web service was varied and the file sizes that produced a performance difference of 5% and 2% were determined. The results are presented in Table 5.2. Clearly for a data size above 135Kb or a CPU execution time of 550 ms service time on a computer resource exposed as a Web service the SOAP-based Web service technology

achieves a comparable performance with that achieved by the RESTful Web service technology.

**Table 5.2: RESTful Web service performance improvement over SOAP-based Web service**

| | **5% performance improvement** | **2% performance improvement** |
|---|---|---|
| Data size (Read file) | 90 Kb | 135 Kb |
| Service time (Execute program) | 95 ms | 550 ms |

The performance of Hybrid Web service is discussed in Section 5.4. The Hybrid Web service seems to achieve a significant performance advantage over a conventional SOAP-based Web service. The performance advantage of the Hybrid Web service is observed to improve with increase in *p* values.

# Chapter 6 : Conclusions

This chapter summarizes the thesis and presents conclusions. Directions for further research are also discussed.

## 6.1 Summary and Conclusions

This research focuses on exposing computing resources and database resources as Web services. Two different Web service technologies: the RESTful Web service and the SOAP-based Web service were used for exposing computing and database resources as Web services.

This research proposed a new Web service technology called the Hybrid Web service. The novel Hybrid Web service utilizes the advantages from both RESTful and SOAP-based Web services: the lightweight characteristics of the RESTful Web service and the support for various standards such as security, reliable messaging and atomic transactions provided by the SOAP-based Web service.

Techniques for exposing and invoking of computing resources and database resources as both RESTful and SOAP-based Web services were described in Chapter 3. Prototype systems based on both the technologies were built and were subjected to various workloads.

The computer resource and the database resource exposed as Web services (RESTful Web service, SOAP-based Web service and Hybrid Web service) support following operations.

- Computer resource exposed as a Web is capable of handling all Windows and Linux operating systems commands.
- Database resource exposed as a Web service provides supports for inserting a student record, reading, updating and deleting a student record.

Two versions of Hybrid Web services were devised. The first version called DBHWS - Type I where a client explicitly requests for the RESTful or the SOAP-based component. The second version called DBHWS - Type II. In the second version, the client requests for the service that in turn invokes the RESTful Web service or the SOAP-based Web service depending on the operation performed by the client. The types of operations requiring the SOAP-based technology are stored in a configuration repository that is consulted when a request arrives at the Hybrid Web service and an appropriate Web service (RESTful or SOAP-based) is chosen by comparing the requested operation with the operations stored in the configuration repository. The prototype implementation of the Hybrid Web service was discussed in Chapter 4.

Based on prototyping and measurement a rigorous performance analysis of resources exposed as the RESTful Web service, the SOAP-based Web service, and the Hybrid Web service was made. The impact of various workload parameters on the relative performance of RESTful Web service, SOAP-based Web service and Hybrid Web service were presented in Chapter 5.

There were three types of systems used in the performance analysis: computing resource exposed as a Web service (both SOAP-based and RESTful), database resource exposed as a Web service (both SOAP-based and RESTful) as well as computing and

database resources exposed as Hybrid Web services. The insights resulting from the analysis of system performance are summarized.

- It can be observed from performance results that the RESTful Web service technology outperformed the SOAP-based Web service technology in all three systems. However, the levels of performance difference between the two technologies vary for all the three systems and are dependent on workload parameters.
- **Compute bound operation**: The performance gain achieved by the RESTful Web service is marginal when the application is characterized by a large CPU execution times. Although the RESTful Web services perform better than the SOAP-based Web services for each operation both technologies demonstrate a comparable performance for a system that is compute bound (see for example Figures 5.5 and 5.6).
- **Communication bound systems:** The performance advantages of the RESTful Web service technology over the SOAP-based Web service technology are small for communication bound systems characterized by a payload comprising large file sizes. For small CPU execution times and small file sizes a much higher difference in the performance between a SOAP-based and a RESTful Web service is observed. For example, the WSART for the read database operation achieved with the RESTful Web service was observed to be lower by as much as 90 % than that for the SOAP-based Web service when the size of data file transfer was 85 bytes (refer Section 5.3.1).

- **Performance improvement:** Experiments were performed to determine the system and workload parameters for which a RESTful Web service technology provides a significant performance benefit over SOAP-based Web service. For example, in one experiment the CPU execution on a computer resource exposed as a Web service was varied and a CPU execution time of 95 ms and 550 ms were observed to produce a performance difference of 5% and 2% respectively (refer to Table 5.2). Smaller CPU execution times are required for observing higher performance improvements for the RESTful Web service. Similarly, the length of the file transferred on a computer resource exposed as a Web service was varied in another experiment of a file sizes of 90 Kb and 135 Kb were observed to produce a performance difference of 5% and 2% were respectively (refer to Table 5.2). Smaller file sizes are required for observing a higher performance difference.
- **Scalability**: For applications which are characterized by a small data payload and a small CPU execution time the RESTful Web services technology demonstrates a much higher scalability (see Figures 5.17 to 5.20 for example). The rate of increase in WSART with the number of records read was observed to be much smaller for the RESTful Web services compared to that achieved with the SOAP-based Web services.
- In comparison to the SOAP-based Web service, a higher scalability is exhibited by the RESTful Web service when the system load is increased (see Figure 5.21 and Figure 5.22 for example).

- The SOAP-based component of the Hybrid Web service is used only when the operations performed by the client demand the support of other standards such as security that are not supported by the RESTful Web service. The lightweight RESTful technology is activated for the other operations. The experimental results demonstrate the effectiveness of the Hybrid Web service. Even when the RESTful Web service was used only 20% of the time (p=0.2) there was a substantial improvement in WSART achieved by the Hybrid Web service in comparison to the conventional SOAP-based Web service.
- When the system is subjected to various intensities of request arrivals, the performance advantage of the Hybrid Web service is observed to increase with an increase in $\lambda$. The lower WSRT achieved by the RESTful Web service in comparison to that of the SOAP-based Web service leads to a smaller queuing delay for the Hybrid Web service in comparison to the conventional SOAP-based Web service. This performance difference produced by the queuing delay increases at higher values of $\lambda$ as more and more queuing starts occurring on the system.

## 6.2 Future Work

Directions for future research are presented next.

- Devising a technique and tool for analysing a client program and installing mechanisms for use with a Hybrid Web services is worthy of investigation. Such a tool needs to able to parse the client code and use the configuration repository (described in Section 4.2.4) to install an appropriate call to a SOAP-based or a RESTful Web service depending on the operation being performed.

- Exposing a mobile device as a Web service can form an interesting direction for future research. Mobile devices are limited in terms of processing power and memory that add to the challenges of exposing a mobile device as a Web service. Exposing a mobile device as a Web service and investigating its performance are worthy of future research.

## References


[1] F. Aijaz, S. Z. Ali, M. A. Chaudhary, and B. Walke, "Enabling High Performance Mobile Web Services Provisioning," in *Proceedings of the IEEE 70th Vehicular Technology Conference*, Anchorage, Alaska, USA, September 2009, pp. 1–6.

[2] F. AlShahwan and K. Moessner, "Providing SOAP Web Services and RESTful Web Services from Mobile Hosts," in *Proceedings of the 5th International Conference on Internet and Web Applications and Services*, Barcelona, Spain, May 2010, pp. 174–179.

[3] Amazon, "Amazon EC2 Pricing." [Online]. Available: http://aws.amazon.com/ec2/pricing/. [Accessed: 31-July-2010].

[4] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, "Poisson Process," in *Discrete-Event System Simulation*, 5th ed., New Jersey: Pearson - Prentice Hall, 2011, pp. 211–216.

[5] S. Cholia, D. Skinner, and J. Boverhof, "NEWT: A RESTful Service for Building High Performance Computing Web Applications," in *Proceedings of the Gateway Computing Environments Workshop*, New Orleans, Louisiana, USA, November 2010, pp. 1–11.

[6] R. A. V. Engelen, "Pushing the SOAP Envelope with Web Services for Scientific Computing," in *Proceedings of the International Conference on Web Services*, Las Vegas, Nevada, USA, June 2003, pp. 346–352.

[7] X. Feng, J. Shen, and Y. Fan, "REST: An Alternative to RPC for Web Services Architecture," in *Proceedings of the 1st International Conference on Future Information Networks*, Beijing, China, October 2009, pp. 7–10.

[8] IBM, “Web Services Architecture Overview.” [Online]. Available: http://www.ibm.com/developerworks/webservices/library/w-ovr/. [Accessed: 21-June-2012].

[9] B. Ioannis G, “Introduction to Web Services,” 2005. [Online]. Available: http://www.cl.cam.ac.uk/~ib249/teaching/Lecture1.handout.pdf. [Accessed: 23-July-2012].

[10] R. Kanagasundaram, S. Majumdar, M. Zaman, P. Srivastava, and N. Goel, “Exposing Resources as Web Services: A Performance Oriented Approach,” in *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, Genoa, Italy, July 2012, pp. 1–10.

[11] E. Landre and H. Wesenberg, “REST versus SOAP as Architectural Style for Web Services,” in *Proceedings of the 5th International Workshop on SOA & Web Services*, Montreal, Quebec, Canada, October 2007.

[12] M. Lanthaler and C. Gutl, “Towards a RESTful Service Ecosystem,” in *Proceedings of the 4th IEEE International Conference on Digital Ecosystems and Technologies*, Dubai, UAE, April 2010, pp. 209–214.

[13] L. Li, C. Niu, N. Chen, and J. Wei, “High Performance Web Services Based on Service-Specific SOAP Processor,” in *Proceedings of the International Conference on Web Services*, Chicago, Illinois, USA, September 2006, pp. 603–610.

[14] N. Lim, S. Majumdar, and B. Nandy, “Providing Interoperability for Resource Access Using Web Services,” in *Proceedings of the 8th Annual Communication Networks and Services Research Conference*, Montreal, Quebec, Canada, May 2010, pp. 236–243.

[15] J. Mangler, E. Schikuta, and C. Witzany, "Quo Vadis Interface Definition Languages? Towards a Interface Definition Language for RESTful Services," in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, Taipei, Taiwan, January 2009, pp. 1–4.

[16] J. Meng, S. Mei, and Z. Yan, "RESTful Web Services: A Solution for Distributed Data Integration," in *Proceedings of the International Conference on Computational Intelligence and Software Engineering*, Wuhan, China, December 2009, pp. 1–4.

[17] M. Muehlen, J. V. Nickerson, and K. D. Swenson, "Developing Web Services Choreography Standards: The Case of REST Vs. SOAP," *Decision Support Systems*, vol. 40, no. 1, pp. 9–29, July 2005.

[18] S. Murugesan, "Understanding Web 2.0," *IT Professional*, vol. 9, no. 4, pp. 34–41, July–August 2007.

[19] MySQL, "MySQL Downloads." [Online]. Available: http://dev.mysql.com/downloads/. [Accessed: 09-June-2011].

[20] T. O'Reilly, "What is Web 2.0 - O'Reilly Media," *What is Web 2.0 Design Patterns and Business Models for the Next Generation of Software*, 30-September-2005. [Online]. Available: http://oreilly.com/pub/a/web2/archive/what-is-web-20.html?page=1. [Accessed: 25-January-2012].

[21] S. Oh and G. C. Fox, "Optimizing Web Service Messaging Performance in Mobile Computing," *Future Generation Computer Systems*, vol. 23, no. 4, pp. 623–632, May 2007.

[22] Oracle, "NetBeans IDE - Features." [Online]. Available: http://netbeans.org/features/index.html. [Accessed: 21-August-2012].

[23] Oracle, “RESTful Web Services.” [Online]. Available: http://www.oracle.com/technetwork/articles/javase/index-137171.html. [Accessed: 08-March-2012].

[24] Oracle, “Oracle GlassFish Server.” [Online]. Available: http://www.oracle.com/us/products/middleware/application-server/oracle-glassfish-server/index.html. [Accessed: 08-March-2012].

[25] Oracle, “Runtime (Java 2 Platform SE v1.4.2).” [Online]. Available: http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Runtime.html. [Accessed: 12-February-2012].

[26] Oracle, “Process (Java 2 Platform SE v1.4.2).” [Online]. Available: http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Process.html. [Accessed: 12-February-2012].

[27] Oracle, “Building Web Services with JAX-WS - The Java EE 6 Tutorial (JAX-RS).” [Online]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html. [Accessed: 27-November-2011].

[28] Oracle, “Building Web Services with JAX-WS - The Java EE 6 Tutorial.” [Online]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html. [Accessed: 27-November-2011].

[29] Oracle, “JavaServer Pages Overview.” [Online]. Available: http://www.oracle.com/technetwork/java/overview-138580.html. [Accessed: 22-June-2012].

[30] Oracle, “Java SE Technologies - Database.” [Online]. Available: http://www.oracle.com/technetwork/java/javase/jdbc/index.html. [Accessed: 21-August-2012].

[31] Oracle, “12 Database Web Services.” [Online]. Available: http://docs.oracle.com/cd/B14117_01/java.101/b12021/callouts.htm. [Accessed: 01-February-2012].

[32] Oracle, “System (Java 2 Platform SE 5.0).” [Online]. Available: http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/System.html. [Accessed: 25-March-2011].

[33] C. Pautasso, O. Zimmermann, and F. Leymann, “RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision,” in *Proceedings of the 17th International Conference on World Wide Web*, Beijing, China, April 2008, pp. 805–814.

[34] S. Sangeetha, “JAX-RS: Developing RESTful Web Services in Java.” [Online]. Available: http://www.devx.com/Java/Article/42873/1954. [Accessed: 27-November-2011].

[35] F. Sha, K. Yu, L. Zhang, and X. Wu, “A Performance Evaluation Method and It’s Implementation for Web Service,” in *Proceedings of the 3rd IEEE International Conference on Broadband Network and Multimedia Technology*, Beijing, China, October 2010, pp. 218–222.

[36] B. Upadhyaya, Y. Zou, H. Xiao, J. Ng, and A. Lau, “Migration of SOAP-based Services to RESTful Services,” in *Proceedings of the 13th IEEE International*

*Symposium on Web Systems Evolution*, Williamsburg, Virginia, USA, September 2011, pp. 105–114.

[37] W3C, "Web Services Architecture." [Online]. Available: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis. [Accessed: 23-July-2012].

[38] W3C, "SOAP Version 1.2 Part 1: Messaging Framework." [Online]. Available: http://www.w3.org/TR/2003/REC-soap12-part1-20030624/#transpbindframew. [Accessed: 02-August-2012].

[39] W3C, "Web Application Description Language." [Online]. Available: http://www.w3.org/Submission/wadl/. [Accessed: 02-August-2012].

[40] W3C, "Web Services Glossary." [Online]. Available: http://www.w3.org/TR/ws-gloss/. [Accessed: 02-August-2012].

[41] W3C, "HTTP/1.1: Method Definitions." [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html. [Accessed: 22-August-2012].

[42] Wiki, "What is Wiki." [Online]. Available: http://wiki.org/wiki.cgi?WhatIsWiki. [Accessed: 26-February-2012].

[43] T. A. Yang, D. J. Kim, V. Dhalwani, and T. K. Vu, "The 8C Framework as a Reference Model for Collaborative Value Webs in the Context of Web 2.0," in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, Waikoloa, Hawaii, USA, January 2008, pp. 319–328.