Performance measurements of Web services

Sarita Damaraju
West Virginia University

Performance Measurements of Web Services

Sarita Damaraju

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2006

ABSTRACT

Performance Measurements of Web Services

Sarita Damaraju

Web services are rapidly evolving application-integration technologies that allow applications in heterogeneous environments to communicate with each other. In this thesis we perform a measurements-based study of an e-commerce application that uses web services to execute business operations. We use the TPC-W specification to generate a session-based workload. The component level response times and the hardware resource usage on the different machines are measured. The component level response times are extracted from the application server logs. From the results it is seen that as the workload increases the response times of the web services components increase. From the hardware resource usage it is clear that web service components require more processing time due to the processing of XML data required in each web service call. The method used in this thesis allows us to study the impact that different components can have on the overall performance of an application.

ACKNOWLEDGEMENT

I would like to express my gratitude to my advisor Dr. Katerina Goseva – Popstojanova for her constant support and guidance throughout my thesis. I am also grateful to her for introducing me to new and interesting technologies in software development. I also thank my other committee members, Dr. Vasudevan Jagannathan and Dr. Donald Adjeroh for their support. I thank my colleague Venu Datla for his guidance throughout this thesis. Finally I would like to thank my family and friends for their constant encouragement, help and support.

TABLE OF CONTENTS

Chapter 1: Introduction.. 1

Chapter 2: Related Work and Our Contributions 3

Chapter 3: Related Technologies 7

Chapter 4: Prototype description... 10

4.1. Software architecture 10

4.2. Implementation details 11

4.3. Deployment details... 12

Chapter 5: Workload Description... 14

Chapter 6: Measurement Methodology 21

Chapter 7: Experimental Results…... 23

Chapter 8: Conclusion.. 30

References.. 32

Appendix 1 35

List of Figures

Fig 1: UML Deployment Diagram of the Online Bookstore 13

Fig 2: Customer Behavior Model Graph for the Online Bookstore application 16

Fig 3: Response time for Search Results Interaction 24

Fig 4: Response time for Shopping Cart Interaction 24

Fig 5: Response time for Verify Credit Interaction 25

Fig 6: Response time for Confirm Order Interaction 26

Fig 7: Response time for User Login Interaction 26

Fig 8: CPU Utilization in Browsing & Ordering Profiles at Application Server 2 28

Fig 9: Database activity in Browsing & Ordering Profiles at Application Server 2 29

List of Tables

Table 1: Transition Probabilities for the Browsing Profile 17

Table 2: Transition Probabilities for the Ordering Profile 17

Table 3: Web Interaction Mixes for Browsing and Ordering Profiles 18

Table 4: Software Components involved in Web Interactions 21

Chapter1: Introduction

The Internet and Web Technologies have changed the way organizations do business. Nowadays, businesses are willing to put their core business processes on the Internet as a collection of web services [1]. A web service can be defined as “an interface that describes a collection of operations that are network-accessible through standardized XML messaging" [2]. They are quickly becoming a significant technology in the evolution of the Web and distributed computing.

Web services can be viewed as a rapidly evolving application-integration technology that allows applications to interact or communicate with each other irrespective of the platform they are running on or the programming language they have been written in. In short, web services are both platform as well as programming language independent. One of the critical factors for the success of web services is that they leverage well-established technologies such as HTTP and XML. They allow developers to create loosely coupled applications based on XML messaging protocols very easily by leveraging the data independence of XML to solve enterprise integration problems, both inside and outside the firewall. Web service interfaces also act like wrappers that can map to any type of software program, middleware system, database management system, or packaged application. They provide a layer of abstraction necessary for bringing together highly distributed and heterogeneous systems.

Individual web services are self-contained, network-accessible applications. An individual web service is limited in its capability because it provides only a specific functionality. Because of their strong potential as a business solution to enterprise application integration, more and more businesses will make their core processes available as web services and will also be ready to adopt web services for their own business needs. As new web services are developed all the time, multiple services exist that provide similar functionality. Selecting an appropriate service with the right capability is very important. In such a scenario, Quality of Service becomes a significant factor in differentiating between various service providers. The Quality of Service offered by an individual web service can decide the usage and popularity of that service. Quality

of service refers to the non-functional attributes of a web service such as performance, reliability, availability and security.
Currently, UDDI – Universal Discovery, Description and Integration based look-ups for web services are based only on the functional aspects of web services. UDDI defines a way to publish and discover information about web services [26]. The UDDI business registry is used to store information about web services. The business registry is actually an XML file that describes a business and its web services. Service providers register themselves and the services that they provide in a UDDI registry by publishing the service descriptions. Issues such a web service performance, availability, reliability and security have not yet been fully addressed and therefore the discovery of web services based on non-functional attributes is still at a very nascent stage. Until a business can be assured of the quality of service of a web service, it is highly unlikely that it would integrate the service with its own applications.

Performance is an important quality of service attribute. This thesis studies the performance aspect of an e-commerce application that uses Web services to execute business operations. By focusing on the software architectural view of the e-commerce prototype we analyze the performance aspects of Web services components under controlled workload conditions. We also measure the impact of the application execution on the hardware resources of the system.

The thesis is organized as follows. Chapter 2 describes the related work on the performance evaluation of web services and our contribution. Chapter 3 discusses the technologies and standards related to Web services. The description of the prototype, including the software architecture, implementation, and deployment details, is given in chapter 4. Chapter 5 describes the workload used in our experiments and Chapter 6 describes the measurement methodology used. Chapter 7 presents the experimental results. Finally, the concluding remarks are given in chapter 8.

Chapter 2: Related Work on Web Service Performance and Our Contributions

The Internet and the World Wide Web are evolving into a huge network of service providers and service consumers. Web services technologies are allowing businesses to provide web-based services to millions and millions of users on the Internet. Since these services are more often than not provided by third parties and are invoked dynamically on the Internet, the quality of service that they offer can vary greatly. There are many Quality of Service parameters that are of crucial importance to web services. Some of these parameters are reliability, availability, performance and security.

Performance modeling and evaluation techniques are very important in the design and implementation of software systems. There are three important techniques for the evaluation and analysis of software performance: direct measurement, simulation modeling, and analytical modeling. Simulation modeling and analytical modeling techniques require construction of a model or an abstract representation of the real system. An analytical model of the system involves the use of mathematical tools such as queuing network models and Petri nets, while simulation models are specialized computer programs. While using analytical modeling techniques or simulations it is important to select a model that captures the salient features of the system without obscuring them with irrelevant details. Performance measurements involve direct observation of the system under test. No one technique is best and most of the times a combination of any of the three techniques are used to evaluate the performance of a system.

A lot of research has been done on evaluating the quality of service of web services**.** In [9] that authors proposed a framework called WebQ for web services based workflows. They developed a set of algorithms to compute QoS parameters and implement them using a rule-based system. They also developed a QoS model for web service selection, binding and execution. In [10] the author says that 48% of the production UDDI registry (tModels tested only) has links that are unusable. He says that this is a major shortcoming of the current UDDI registries since they allow for web services discovery based only on

functional requirements. In this paper the author proposed a framework of a regulated UDDI registry model that can co-exist with the currently de-regulated UDDI registries.
He adds a new role to this model : the Web service QoS Certifier. In this new model, a web service has to first communicate with the QoS certifier about its claims of quality of service before it can actually register with the repository. The certifier verifies these claims and then issues a certification if they are true. The web service can register with the UDDI only after it recieves the certification from the QoS certifier. In [11] the authors consider five generic quality criteria for elementary web services: (1) execution price, 2) execution duration, (3) reputation, (4) reliability, and (5) availability. They then present a general model to evaluate the QoS of both elementary and composite web services. Based on this model, a global service selection approach that uses linear programming techniques to compute optimal execution plans for composite services have been described. A similar proposal has been made in [19]. The authors proposed the concept of a Web Service Broker that scans UDDI registries for up-to-date information about web services and their quality of service parameters. They also defined an XML Schema for web service-QoS definitions as a generic mechanism to specify quality of service parameters.

While the above mentioned papers discuss the quality of service of web services, very few researchers have focused on the performance aspect of web services. Web services are distributed in nature and performance is a very important quality parameter of any distributed system. Performance analysis is also the key to understanding scalability issues in e-commerce sites [12].

In [13] the authors studied the impact of introducing web services into an already tightly coupled application. They compared the overall throughput, response times and latency of two versions of the same application. One implementation uses the Java Messaging Service while the other version is implemented using web services. They take measurements by instrumenting the application at different points during execution. The JMS version of the application has better performance than the version that was implemented using web services. In [14] the author presented a reengineering scenario where a legacy application is migrated to web services. The case study used was an auction site that was originally implemented using J2EE but is now migrated to web services. The major metrics used in this study were latency and scalability. It was observed that by migrating to web services the scalability of a traditional EJB application is reduced.

In [15] the authors proposed a web services-based clinical decision support infrastructure. They used SPE – Software Performance Engineering techniques and Layered Queuing Network Models to analyze the performance implications of such an infrastructure. The performance analysis presented in this paper can be generalized to other web services-based systems.

A simulation tool was used in [16] to analyze the performance of web services. Using the JSIM simulation tool the authors created "what-if" scenarios to visualize the web processes in action before integrating them with other application components. In [17] the authors evaluated the performance of web services when used with resource-constrained clients such as handheld mobile devices.
The performance impact of web services on Internet servers was studied in [20]. The authors compare two applications that perform the same task – an electronic book inventory system. One application uses Active Server Pages technology while the other application uses web service. The throughput and response times were measured and it was seen that the ASP version of the application had a higher throughput and lower response time than the web services version. The web services impose additional overhead on the servers since they have to parse XML data. The workload was generated using two different versions of the S-client workload generator. One version emulated a specified number of clients while the other version overloaded the server.

Contributions

The work presented in this thesis focuses on the performance aspect of web services and studies their contribution to the overall server-side response time when integrated with other EJB components. A three-tier e-commerce application representing an online bookstore was developed for this purpose.

The goal of this thesis is to study the performance of web services when they are integrated with other middle-tier components like Enterprise Java Beans to create an online e-commerce application. Since this is as measurements-based study it was necessary to use a synthetic workload to simulate the activities of online users browsing an e-commerce site. Since the traffic in e-commerce sites is typically session-based, the TPC-W specification was used. The TPC-W specification is geared towards business-to-customer e-commerce interactions. While the TPC-W specification also uses an online bookstore as a prototype for its experiments, it should be noted that the prototype developed in this thesis uses web services components to perform some of its e-business functions. The credit-card verification service and the book search service have been implemented as web services in this case.

In this thesis we study the performance of the web services components and the EJB components and the effect they have on the hardware resources of the system. Measurements at the software architecture level are taken by instrumenting the application to log the component level response times in the application. The application server logs were then parsed using the AWK [7] scripting language to extract the component level response times. Hardware resource usage is measured using the Windows 2000 Performance Monitoring tool. Similar work is presented in [18] and [28]. The case-study used in [18] and [28] represents an online travel agency while the case-study used in this thesis represents an online bookstore.

The TPC-W specification is used to generate a synthetic workload. Two different workload profiles are generated: the Browsing profile and the Ordering profile. The component response times and the hardware resource usage for both workload profiles are then compared. The results are then used to help identify the hardware and software bottlenecks in the system and to understand the overhead introduced by the web service components into the system.

Chapter3: Technologies that enable the implementation of Web Services

The following technologies have been standardized to implement web services. Each technology targets one specific layer in the web services architecture stack. These technologies are: XML (Extensible Markup Language), WSDL (Web Services Description Language), SOAP (Simple Object Access Protocol), and UDDI (Universal Description, Discovery and Integration). Other technologies may also be used.

XML: Extensible Markup Language

Extensible Markup Language or XML is a universal text-based standard for representing and exchanging documents on the Internet or on Intranets. XML is a meta-language that can be used to create other specialized markup languages. XML allows users to create their own customized tags and document structures, thereby enabling the definition, transmission, validation, and interpretation of documents and data between different applications and between organizations. Since XML data is stored in plain text format, it provides a software- and hardware-independent way of sharing data across different platforms, operating systems and languages.

An XML Parser is required to read and manipulate XML documents. The parser extracts the actual data out of the XML document and creates new data structures from them. Parsers also check whether documents conform to the XML standard and have a correct structure. This is an essential for the automatic processing of XML documents. The correct document structure can be defined by using XML Schemas. XML Parsers are available in different languages.

XML is also a core technology in the development of Web services.

WSDL: Web Services Description Language

Web Services Description Language or WSDL is an XML document used to describe web services. The WSDL document describes the following critical pieces of information about a given web service: all the publicly available functions of the web service, the data types used in the message requests and responses, the address where the service can be accessed and the protocol binding information. Using the information in the WSDL document a client can locate and invoke the functions of a web service.

WSDL document can also contain other elements, like extension elements and a service element that makes it possible to group together the definitions of several web services in one single WSDL document.

SOAP: Simple Object Access Protocol

Simple Object Access Protocol or SOAP is the industry standard protocol for XML-based messaging which provides users with a transmission framework for inter-application or inter-service communication via HTTP. The SOAP specification establishes a standard message format that consists of an XML document capable of hosting RPC and document-centric data.

A SOAP message is actually an HTTP POST message with an XML envelope as a payload. It is a lightweight protocol based on XML and can be used to exchange structured messages between different applications that are distributed over a network.

A SOAP message consists of the following parts:

- An envelope that identifies the document as a SOAP message,
- An optional Header element that contains header information,
- A required Body element which contains a set of encoding rules and a convention to represent RPCs and responses,
- An optional Fault element used to provide information about errors encountered while processing the message.

The SOAP envelope is used for describing what the message actually contains. SOAP also defines data encoding rules. The set of encoding rules are used to express data types defined by applications. SOAP data encodings specify rules for object serialization; that is, mechanisms for marshaling and unmarshalling data streams across the net.

SOAP can be used in combination with a variety of network protocols like HTTP, SMTP, FTP and RMI over IIOP.

UDDI: Universal Description, Discovery and Integration

UDDI defines a way to publish and discover information about web services [7]. The UDDI business registry is used to store information about web services. The business registry is actually an XML file that describes a business and its web services. Service providers register themselves and the services that they provide in a UDDI registry by publishing the service descriptions. The UDDI business registry classifies the services under different categories. Service requestors then search a registry for the type of service that they are looking for. Once they find a service, they use the service descriptions to contact the provider of that service and bind with or invoke that service.

A UDDI business registration consists of the following components:

- Business Description or White Pages – Basic contact information about a company such as address, contact information, urls, and other known identifiers; This information allows clients to discover a web service based on business identification.
- Business Service or Yellow Pages - Industrial categorizations based on standard taxonomies; this information allows clients to identify web services based upon its category.
- Binding Template or Green Pages – This gives technical information about services exposed by the business such as service entry point and other implementation details.
- tModel – A tModel describes the web service interfaces and provides pointers to specifications that describe their implementation.

Chapter 4: Prototype Description and Experimental Setup

The e-commerce prototype developed represents an online bookstore. The application allows customers to shop for books online based on author name or book title, add books to a shopping cart and purchase books in a secure manner. This chapter describes the architecture, implementation details and the deployment details of the prototype developed.

4.1. Prototype Description

Successful e-commerce companies like Amazon and eBay are able to scale their operations faster than their competitors in order to meet growth in traffic and transactions. These companies typically use multi-tier distributed systems that leverage web technologies and industry-standard hardware and software components. As a result, they are able to build highly scalable and flexible systems that are ready to meet continuously increasing demand as well as new competitors.

Most e-commerce systems use three-tier architecture, which consists of the following three layers or tiers:

- Presentation layer which is responsible for presenting the application to the end-users,
- Business logic layer which implements the business logic of the application,
- Database layer, which is responsible for managing the persistence of the application information.

The prototype developed in this thesis uses 3-tier architecture.

The presentation layer consists of a collection of web pages that accept user input and deliver content to the users. The web pages used are: Home page, Search page, Shopping Cart page, Login page, Credit Card Details page and Confirm Order page.

The business logic layer consists of components that implement the business logic of this application. The components used in this layer are:

BookStore Web Service: The Bookstore web service accepts author last name or book title as search criteria and returns a list of books that match the given criteria. In an earlier implementation of this prototype a publicly hosted web service was used [27]. This web service was not available almost 90% of the time. Also, the publicly hosted service returned only the price of a given book and was considered limited in its functionality. Hence, the BookStore web service was implemented and hosted locally.

CreditCard Web Service: The CreditCard web service is a locally hosted web service that validates a customer's credit card information.

Customer EJB: The Customer-EJB is an Enterprise Java Bean Component validates the customer's username and password during login and then stores customer information for the remainder of the customer session.

Order EJB: The Order-EJB is an Enterprise Java Bean component that is used to persist the customer's order details into the database.

The data layer of the application uses a relational database management system to store data. All data in the database is stored in the form of tables. Each component of the business logic layer – BookStore Web Service, CreditCard Web Service, Customer EJB and Orders EJB manipulates a corresponding database table in response to a user request.

4.2: Implementation and Deployment Details of Prototype

The primary goal of this thesis is to study the contribution of the web services and EJB components to the overall server-side response time of a given interaction. It must be noted that the main reason the web services are hosted locally is because in earlier implementations of the prototype, the publicly hosted web service was not available most of the time. By hosting the web services and EJB components of the application locally, any network delay associated with the service calls to these components is avoided. This gives us a better idea of the response times of the individual components.

Another goal of this thesis is to see how the web service and EJB components scale with increasing workload. Again, it seemed better to host the web services locally because most publicly hosted web services try to limit the number of calls to a service in a given time period in order to avoid overloading the server and Denial-Of-Service attacks.

Implementation Details

The online bookstore application has been developed using a host of J2EE technologies. The Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multi-tier enterprise applications [21].

The presentation layer is implemented as a set of JSP pages. Java Server Pages is a J2EE technology that is used to create dynamic web pages. The JSP pages are hosted on a Tomcat version 5.0 web server.

The components in the business logic layer are developed using Enterprise Java Beans and Web Services technologies. The BookStore Web Service and the CreditCard Web Service are web services that have been developed using JAX-RPC. JAX-RPC: Java API for XML based RPC is an API that is now integrated with the J2EE 1.4 specification and can be used to develop web services. The Customer EJB and Orders EJB are EJB components. Enterprise Java Beans technology is the standard server side component architecture for the J2EE platform [22].

The data layer uses Oracle 9i Release 2 as the backend database management system.

Deployment Details

The application is deployed across a network of 4 computers. The Web Server, JSP pages and the EJB components run on one machine with a 3 GHz Pentium 4 Processor and 1 GB RAM. The Web Services are hosted on another machine with a 3 GHz Pentium 4 Processor and 1 GB RAM.

The database server runs on a separate 3GHz Pentium 4 dual processor machine with 1GB RAM. All the three machines run Windows 2000 operating system.

The workload generator runs on a 2.8 GHz Pentium 4 M Processor machine with 704 MB RAM. This machine runs Windows XP operating system.

All the machines are inter-connected through an Ethernet LAN with a speed of 1 Gbps.

The UML deployment diagram of the online bookstore is shown in Fig.1.

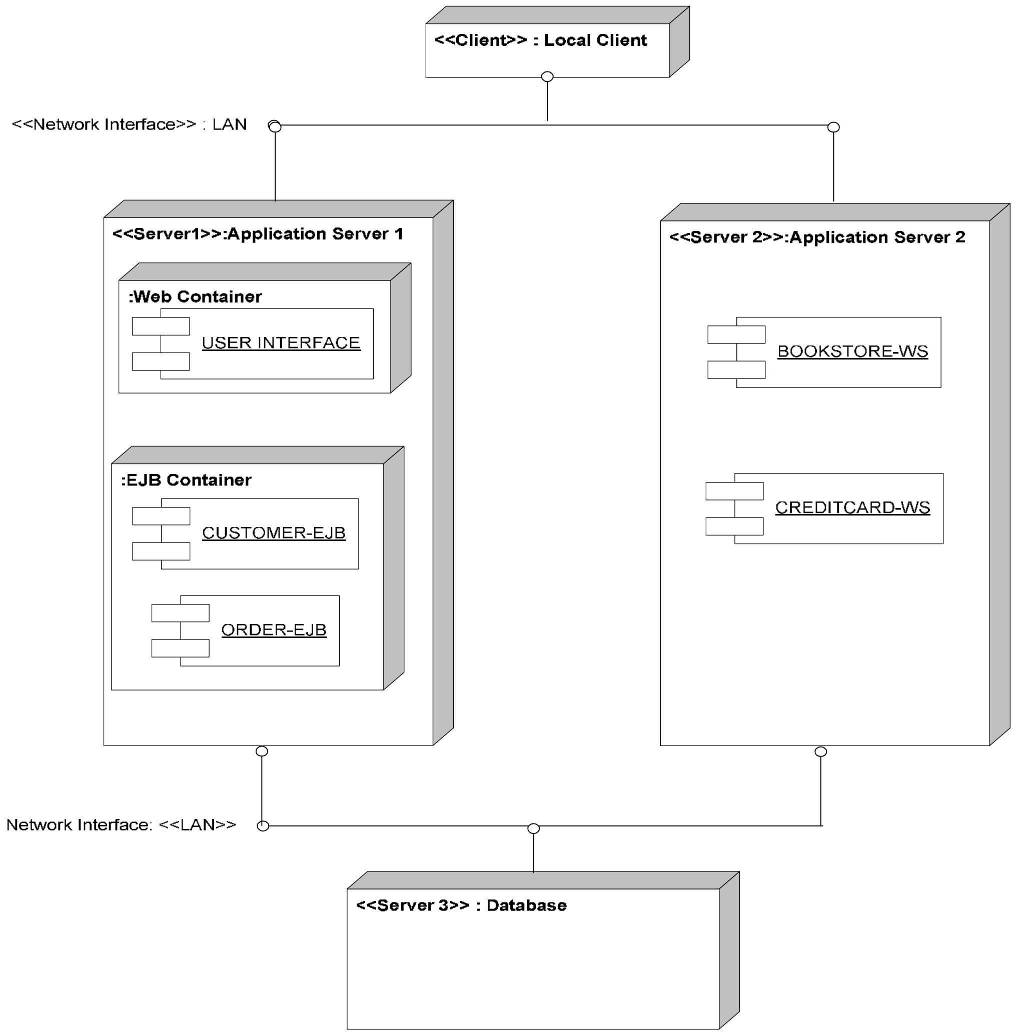

<<Client>> : Local Client
<<Network Interface>> : LAN
<<Server1>>:Application Server 1
:Web Container
USER INTERFACE
:EJB Container
CUSTOMER-EJB
ORDER-EJB
<<Server 2>>:Application Server 2
BOOKSTORE-WS
CREDITCARD-WS
Network Interface: <<LAN>>
<<Server 3>> : Database

Fig 1: UML Deployment Diagram of the Online Bookstore

Chapter 5: Workload details on TPC

Workload characterization is one of the most important steps in the planning of an e-business site's capacity [3]. In order to understand the performance and scalability of an e-commerce site and to provide quality of service to customers, it is necessary to understand the characteristics of its workload. Many studies have been carried out with the purpose of understanding the nature of an e-commerce site's workload. Different types of e-commerce sites may have different kinds of workloads and it is hard to characterize a representative workload for each site. Being able to characterize the workload of an e-commerce site has two important outcomes. Firstly, we will be able to understand the essential features of the workload of the given site and generate an appropriate workload model. This will greatly help in the planning for capacity of the site and identifying potential bottlenecks. Secondly, these workload models can be use to generate synthetic workloads which can later be used as benchmarking tools for evaluating other sites.

A lot of research has been done on the characterization of representative workloads for e-commerce sites. In [4] the author characterizes the workload of an e-business in a multi-layer hierarchical manner. The four layers are business, session, function and protocol. Each layer has its own set of characteristics. The business layer examines the characteristics of the business and how they affect a user's interaction with the site. The session layer deals with characteristics of the session such as session duration, user navigation patterns within a session and the buy-to-visit ratio [4]. The function layer characterization involves an analysis of the number of e-business functions invoked by users within a session and across different time-scales. The protocol or HTTP request layer characterization involves the analysis of the workload in terms of HTTP requests such as number of HTTP requests in a given time period.

In [3], the authors present a detailed study of the characterization of workloads for Internet World Wide Web servers. From the web access logs of different Internet servers they identify ten common characteristics of web server workloads. These characteristics are successful requests, document types, one time referencing, distinct requests, file-size distribution, concentration of references, inter-reference times, remote requests and wide area usage. They then use these characteristics to propose two strategies for the design of a caching system to improve the performance of web servers.

In this thesis we study the performance of an e-commerce application that simulates an online bookstore. In an e-commerce application customers typically interact with the site through a session. In [5] the authors define a session as a sequence of requests to execute e-business functions such as search, browse, add to cart, login, register and pay. Hence the workload of this site is better-described using sessions.

The TPC-W Benchmark

The TPC-W Benchmark is a transactional web benchmark [6]. It is a benchmark developed for a B2C e-commerce scenario. The closed workload is performed in a controlled Internet commerce environment that simulates the activities of a business oriented transactional web site. [6] A typical e-commerce environment is characterized by multiple concurrent browser sessions, service of static and dynamic web content and a significant amount of database activity. The TPC-W benchmark describes such a scenario. The main features of this specification include the generation of multiple online browser sessions, dynamic page generation with database access and update, databases consisting of many tables of different sizes, attributes and relationships, enforcement of ACID properties in the database, and the support for SSL authentication. The TPC-W benchmark is a benchmark specification that must be adapted to work with the e-commerce site under test. It is not a tool that readily generates workloads.

The primary performance metric reported by TPC-W specification is the number of web interactions processed per second (WIPS).

The TPC-W specification uses the concept of an **Emulated Browser (EB)** to represent a user who interacts with the web site using a browser to generate a sequence of HTTP requests to execute various e-business functions.

In this specification the navigational pattern of the user is modeled using a Customer Behavior Model Graph (CBMG). In [5] the authors introduce the concept of a state transition graph called the Customer Behavior Model Graph to describe the behavior of customers who show similar navigational patterns.

It is important to note that the CBMG characterizes the navigational pattern of the user as viewed from the server side [5]. It consists of a graph where each node represents a state such as home page, browse, search, add to cart and pay, and transitions between these states. Each transition is assigned a probability. The Customer Behavior Model Graph for the online bookstore is given in Figure 2:

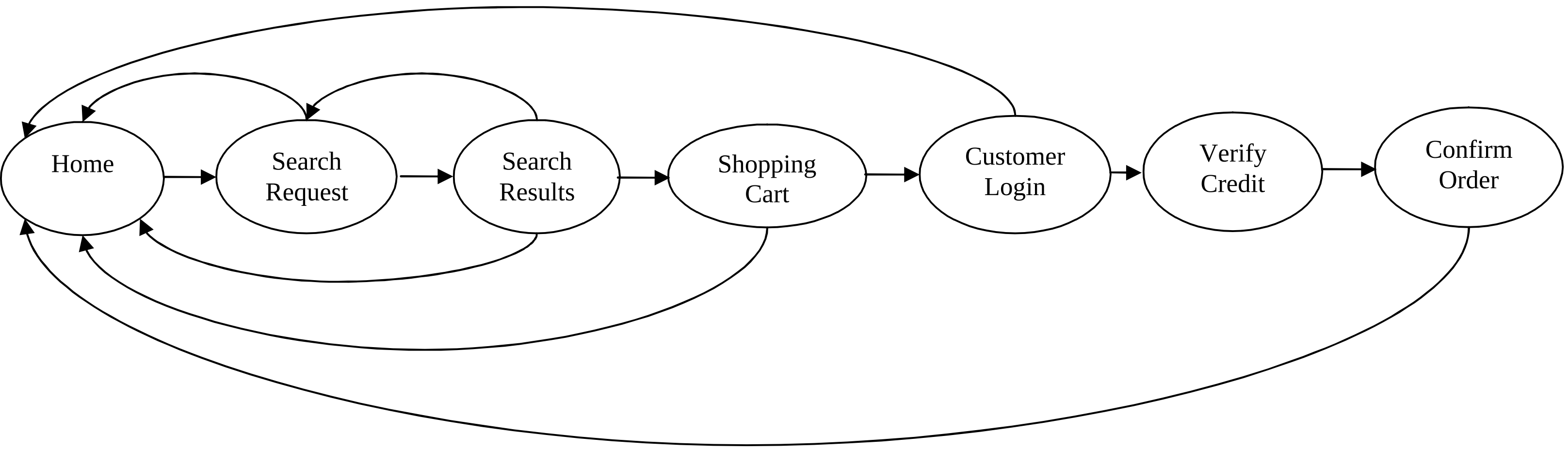


Fig: 2 Customer Behavior Model Graph for the Online BookStore application

TPC-W determines the transition probabilities between each state based on the web interaction mix used to generate the workload. The specification emulates different user browsing scenarios by varying the ratio of the browse to buy activity. As such it defines three different types of user web interactions mixes– Browsing, Shopping and Ordering. A web interaction mix is defined by a matrix of probabilities, where each element of the matrix is the probability of which page a user is most likely to visit. Depending on the type of web interaction mix used, the emulated browser visits some part of the web site more frequently than the others.

	Init	Home	Search Req	Search Res	Shop Cart	Cust Login	Verify Credit	Confirm Order
Init	0	9999	0	0	0	0	0	0
Home	0	0	0	0	0	0	0	0
SearchReq	0	3330	9999	0	0	0	0	0
SearchRes	0	1000	4482	9999	0	0	0	0
ShopCart	0	8125	0	0	9999	0	0	0
CustLogin	0	1667	0	0	0	9999	0	0
VerifyCredit	0	0	0	0	0	0	9999	0
ConfirmOrder	0	9999	0	0	0	0	0	9999

Table 1: Transition Probabilities for the Browsing Profile

	Init	Home	Search Req	Search Res	Shop Cart	Cust Login	Verify Credit	Confirm Order
Init	0	9999	0	0	0	0	0	0
Home	0	0	0	0	0	0	0	0
SearchReq	0	571	9999	0	0	0	0	0
SearchRes	0	1	4482	9999	0	0	0	0
ShopCart	0	667	0	0	9999	0	0	0
CustLogin	0	2500	0	0	0	9999	0	0
VerifyCredit	0	0	0	0	0	0	9999	0
ConfirmOrder	0	9999	0	0	0	0	0	9999

Table 2: Transition Probabilities for the Ordering Profile

TPC-W further categorizes each web interaction as either a **Browse** or an **Order** interaction. A browse interaction typically involves browsing or searching activities, while an order interaction involves activities such as adding an item to a shopping cart, placing an order and making a payment. In this application the interactions categorized as browse interactions are: **Home**, **Search Request**, and **Search Results**. The interactions categorized as order interactions are: **Shopping Cart**, **Customer Login**, **Verify Credit** and **Confirm Order**.

In this thesis, we run experiments for two types of web interaction mixes or workload profiles: Browsing and Ordering. The Browsing mix represents users who spend most of the time browsing or searching for books while the Ordering mix represents users who take part in the buying and ordering activities.

The following table shows the web interaction mixes for the Browsing and Ordering profiles used in the experiments.

	Browsing Profile	**Ordering Profile**
Browse	**76%**	**50%**
Home	17.0%	16.0%
Search Request	30.0%	17.5%
Search Results	29.0%	16.5%
Order	**24%**	**50%**
Shopping Cart	16.0%	15.0%
Customer Login	3.0%	14.0%
Verify Credit	2.5%	10.5%
Confirm Order	2.5%	10.5%

Table 3: Web Interaction Mixes for Browsing and Ordering Profiles

The TPC-W specification imposes certain restrictions on workload generation process. The steady state period during which the workload is generated is known as a Measurement Interval. In this case we have two measurement intervals corresponding to the two web interaction mixes. Over each measurement interval the remote browser emulator must maintain the mix of web interactions as specified in the table above. As a result of the finite random selection process a maximum deviation of (0.05*P) where P is any of the required mix percentages, is allowed.

During each session, the emulated browser starts at the **Home** interaction and navigates through all the states in the CBMG of this application. Each emulated browser cycles through a process of requesting a web interaction, receiving and parsing the response page, measuring the time required to receive it (WIRT), selecting the next navigation option, preparing the next request, and waiting the balance of the Think Time before requesting the next web interaction [6]. The required mix of web interactions is maintained by controlling the selection of the next navigation option.

User Session Minimum Duration (USMD) refers to the minimum duration measured by the emulated browser for which a user session must last. For each user session the emulated browser generates a USMD randomly selected from a negative exponential differentiation [6]. A user session ends when the USMD elapses and the next navigation option is the Home web interaction. The workload generator then terminates the current session and starts a new one.

A test run can consist of one or more measurement intervals. Each test run consists of the following three phase: ramp-up interval, steady-state interval and ramp-down interval. During the ramp-up interval all the emulated browsers are initialized. The system reaches a steady state at the end of the ramp-up interval. All data measurements must be computed over an interval over which the throughput is in a steady state interval condition that represents the true sustainable performance of the system under test [6]. The specification requires that we report the 90th percentile value of the response times measured during the steady state interval.

In these experiments, the ramp-up interval is 5 minutes, the steady-state interval is 30 minutes and the ramp-down interval is 1 minute.

The throughput of the benchmark is controlled by the activity of the emulated browsers. Each EB emulates only a single user session at a given time. Therefore increasing the number of emulated browsers can increase the throughput demand on the system under test. The benchmark also specifies scaling requirements in order to maintain the ratio between the load presented to the system under test, the number of records in the database tables, the required storage space and the number of emulated browsers. Database scaling is defined by the number of rows in the database tables and the number of emulated browsers.

Since we ran the experiments for 50, 100, 150 and 200 clients, as per the specification the customer table is populated with 576,000 records. The bookstore table has 100,000 items. The average think time is set to 7 seconds while the average user session duration is 15 minutes.

Chapter 6: Measurement Methodology

The primary performance metric of the TPC-W specification is the number of Web Interactions per Second (WIPS). The term Web Interaction Response Time (WIRT) is defined as the time elapsed from the first byte sent by the EB to request an interaction until the last byte received by the EB to complete that interaction. According to the specification, even though the intent of this benchmark is to measure the response times as experienced by the user the WIRT does not take into account the time needed to display web pages and objects [6].

The primary goal of this thesis is to study the contribution of the web services and EJB components to the overall server-side response time of a given interaction. In addition to this, the thesis also aims to study the effect of the e-commerce application on the hardware resources of the system.

The following table presents the web interactions in the online bookstore application and the software components executed during each interaction.

Web Interaction	**Software Component Executed**
Home	Home page
Search Request	Search page
Search Results	Bookstore-WS and Search Results servlet
Shopping Cart	Bookstore-WS and Shopping Cart servlet
Customer Login	Login page, Customer-EJB and Login servlet
Verify Credit	Card details page, CreditCard-WS and CreditCard servlet
Confirm Order	Bookstore-WS, Orders-EJB

Table 4: Software Components involved in Web Interactions

We measure the response times at the software architecture level by logging the time taken to execute each of the above components during each web interaction. The Java Logging API (**java.util.logging)** provides a way to capture information about the operation of a J2EE or Java-based application. Once captured, the information can be used for many purposes, but it is particularly useful for debugging, troubleshooting, and auditing.

The Logging API is designed to let a Java program, servlet, applet, EJB, etc. produce messages of interest to end users. The Logging APIs capture information such as security failures, configuration errors, performance bottlenecks, and/or bugs in the application or platform. The core package includes support for delivering plain text or XML-formatted log records to memory, output streams, consoles, files, and sockets. The API also supports dynamic configuration, hierarchical loggers and multiple logging levels. To make sure that logging can be left in a production program, the API is designed to make logging as inexpensive as possible.

The application server settings were modified to enable the web container and the EJB container to log timestamps of the events taking place in the application. The application components themselves were modified to use the java.util.logging API to persist the component response times into the application server logs.

Since, the server log files produced during the experiments were of sizes running into several hundreds of Megabytes, the AWK scripting language [7] was used to parse the log files to extract the response times of each component.

The hardware resource usage of the system is measured by using the Windows 2000 performance-monitoring tool. The Performance Logs and Alerts service is a measurement infrastructure that reflects the current state of the system through performance counters. The counter logs record data about the hardware resources and system services.

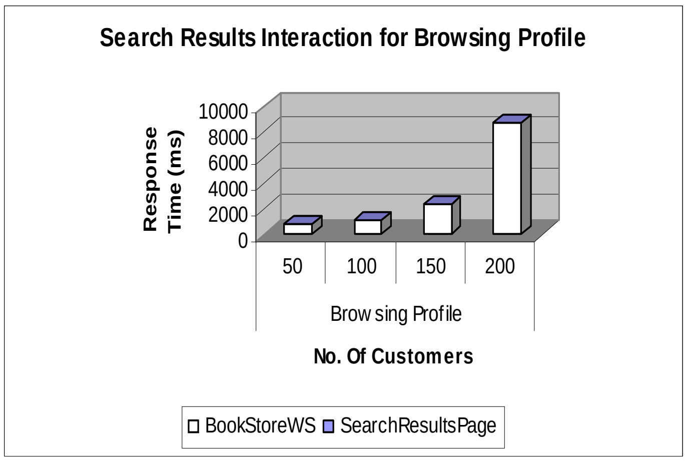
Since the components of the online bookstore are deployed across different computers, the performance counters on each machine record the percentage of non-idle processor time spent in user mode (*%User Time*) and the rate of read and write operations on the disk (*Disk Transfers/sec*).

Chapter 7: Experimental Results

The e-commerce prototype developed represents an online bookstore. The application allows customers to shop for books based on author last name or book title.

The Home, Search Request, Customer Login and Credit Card Details interactions serve only static HTML content. These interactions are very similar in nature and the response times for these interactions are almost zero. An increasing workload does not have an effect on these interactions.

We first analyze the response times of the Search Results Interaction. The Search Results interaction involves a call to the Bookstore Web Service. Figure 3 shows the 90th percentile response times of the search results interaction. The response times for the Ordering and Browsing profiles for 50 and 100 clients are approximately the same, though for 150 and 200 clients the response times are much higher. This increase in response times could be attributed to the increase in the response time of the web service component. Also, it must be noted that the response times for the Browsing profile are significantly higher than that of the Ordering profile. The call to the Bookstore Web Service is a database intensive activity. The Browsing profile consists of 76% browsing activity and 24 % ordering activity. The browsing activity involves calls to the search operations of the Bookstore Web Service while the ordering activity involves calls to the store order operation of the Order EJB and update store operations of the Bookstore web service. The search operation returns one or more Book objects depending upon the search string used in the call. As a result, the size of the SOAP messages being returned by the search query in each case will depend upon the number of books returned in the result set of the query. This results in the response times of the Browsing profile being higher than the results of the Ordering profile.



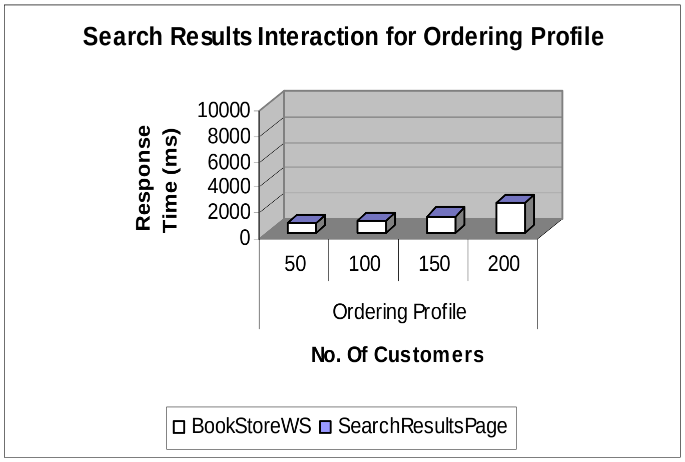
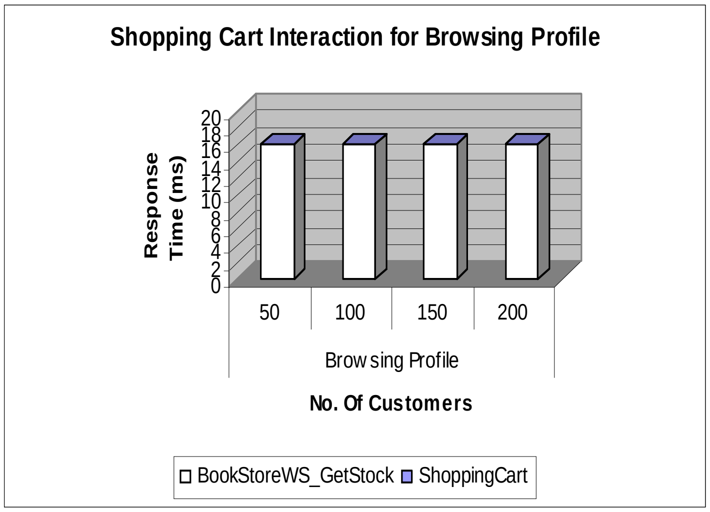


Fig 3: Response time for Search Results Interaction

Next, we discuss the response times of the Shopping Cart interaction and the Verify Credit interactions. The Shopping Cart interaction involves a call to the GetStock operation of the Bookstore web service. The response times do not increase much with increasing workload. There is also not much of a difference between the response times for the Browsing and the Ordering profiles. This is because the Get Stock operation of the Bookstore web service involves a relatively inexpensive database call and also returns just an integer value thereby reducing the size of the SOAP message that is returned by the call.



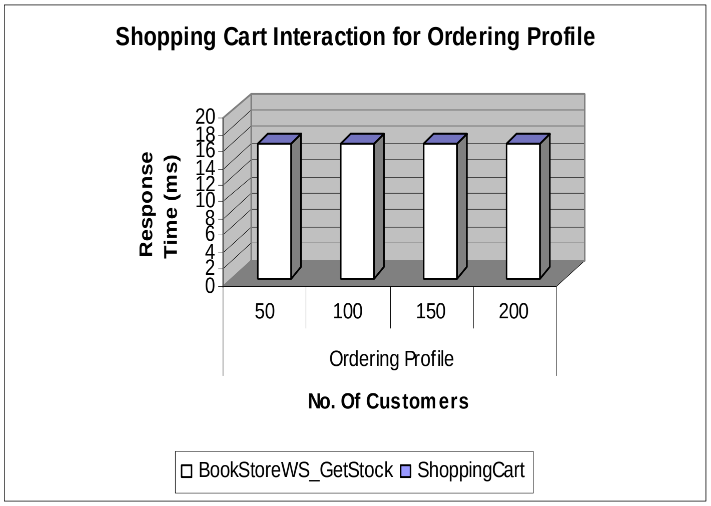
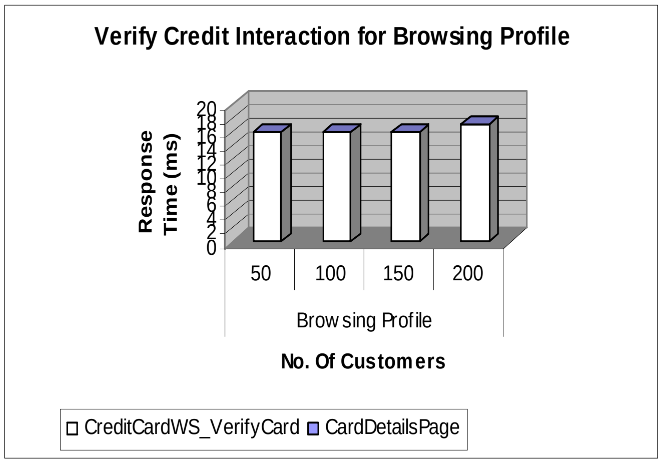


Fig 4: Response time for Shopping Cart Interaction

The Verify Credit interaction exhibits a similar behavior to the Shopping Cart interaction. An increasing workload does not have much of an effect on the response times of this interaction. The response times are also the same for both the Browsing and the Ordering profile. Like the Get Stock operation of the Bookstore web service, the Verify Card operation of the Credit Card web service involves an inexpensive database call and returns a single String value. This again reduces the size of the SOAP message returned by the web service call. Hence even an increasing workload does not have much of an effect on the response times of these interactions.



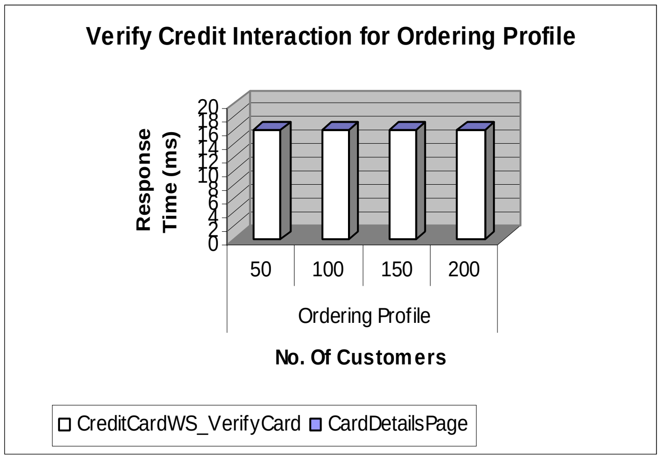
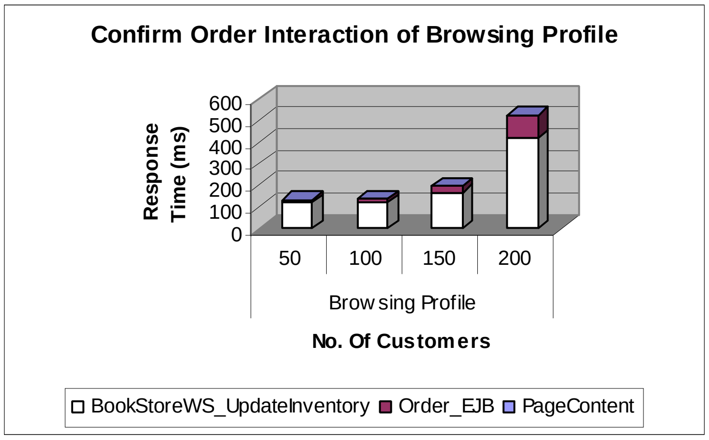


Fig 5: Response time for Verify Credit Interaction

The response times for the Search Results interaction and the Confirm Order interaction display similar behavior. That is the response times' increase with increasing workload. Both interactions make web service calls. The Confirm Order interaction involves calls to the Update Inventory operation of the Bookstore Web Service and the Order_EJB. The increase in response time for the Confirm Order interaction is mainly due to the web service component. It must be noted that for the Confirm Order interaction nearly 60-80% of the time is spent in executing the web services components.



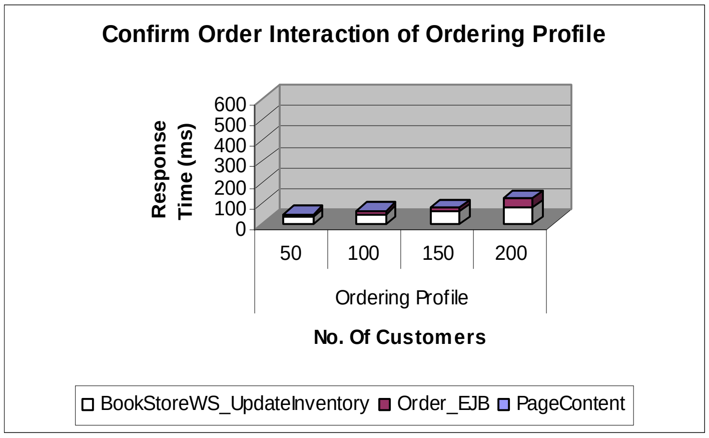
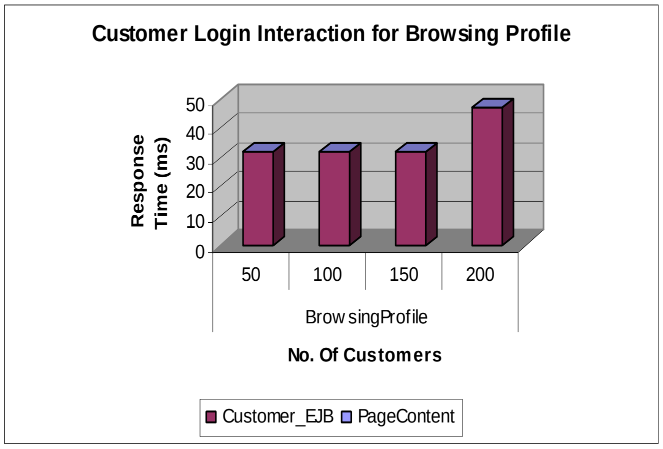


Fig 6: Response time for Confirm Order Interaction

We finally discuss the Customer Login interaction. This interaction involves the processing of the Customer_EJB component and the page content. The page serves static HTML content and has a response time that is almost zero. The Customer_EJB call is also an inexpensive database call. For the Ordering profile, the response time of the Customer Login interaction increases as the workload increases from 50 clients to 100 clients but does not change much for 150 and 200 clients. For the Browsing profile, while the response times do not change much for a workload of 50, 100 and 150 clients, there is a significant increase in the response time for 200 clients.



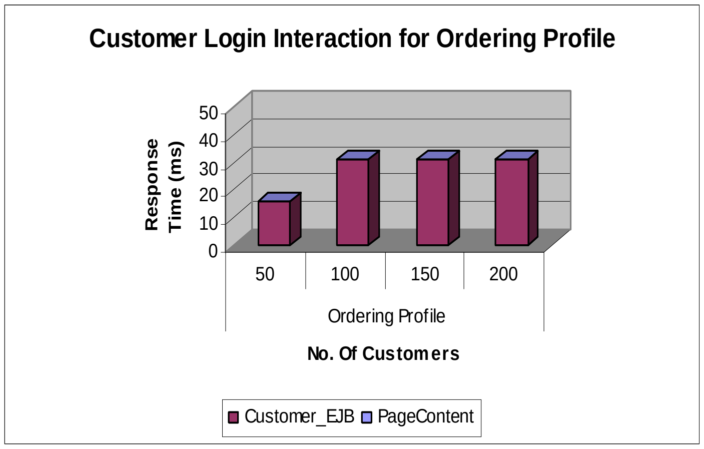
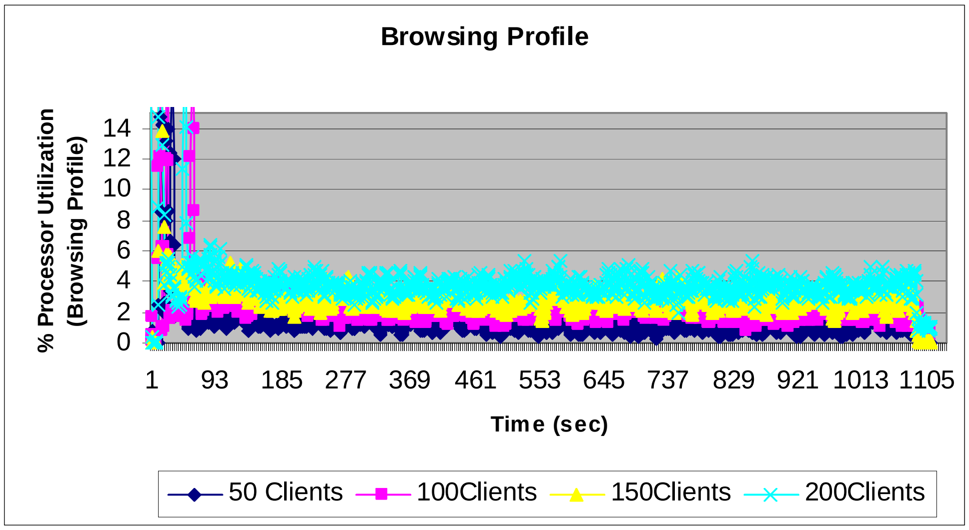


Fig 7: Response time for Customer Login Interaction

SOAP is the standard protocol for Web services. SOAP uses XML as its payload to marshal data that is transported to a software application. The performance of SOAP depends on the following factors:

- Extracting the SOAP envelope from the SOAP packet.
- Parsing the XML information in the SOAP envelope using a XML parser.
- XML data cannot be optimized too much.
- SOAP encoding rules make it mandatory to include typing information in all the SOAP messages sent and received.
- Encoding binary data in a form acceptable to XML results in overhead of additional bytes due to the encoding as well as processor overhead incurred in performing the encoding/decoding.

XML's way of representing data usually results in a substantially larger size than representing the same data in binary, which is on average 400% larger. This increase of the message size creates a critical problem when data has to be transmitted quickly, which effectively results in increase of the data transmission time. The XML parser must be loaded, instantiated, and fed with the XML data. Then the method call argument information must be discovered. This involves a lot of overhead as XML processors grow to support more XML features. Most existing XML parsers are too expensive in terms of code size and processing time because these parsers have to support a number of features like type checking and conversion, wellformedness checking, or ambiguity resolution. All these make XML parsers require more computing resources. All this marshalling and unmarshalling of data makes web service calls very expensive. As the XML payload size in the SOAP message increases, the time taken to encode and parse these messages also increases. Also these are CPU intensive activities. We therefore study the CPU utilization on the machine that hosted the web service components.



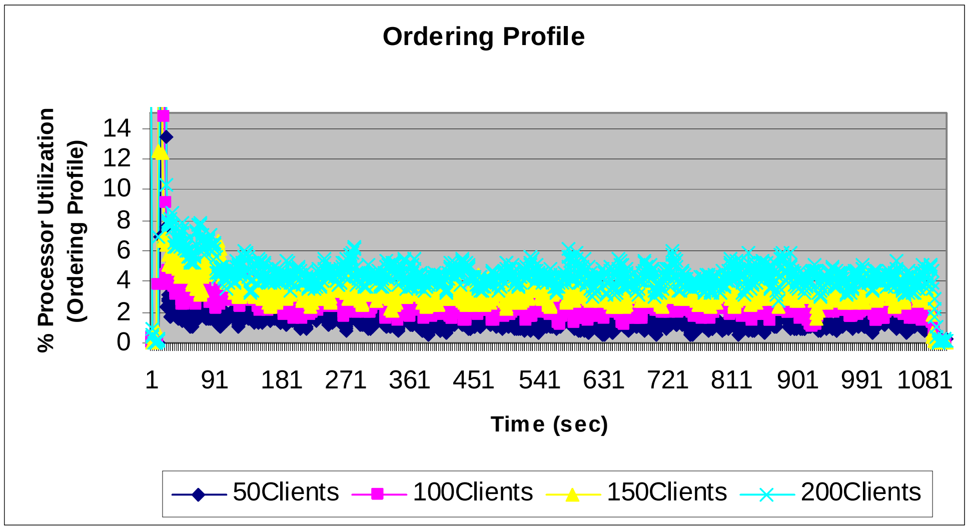
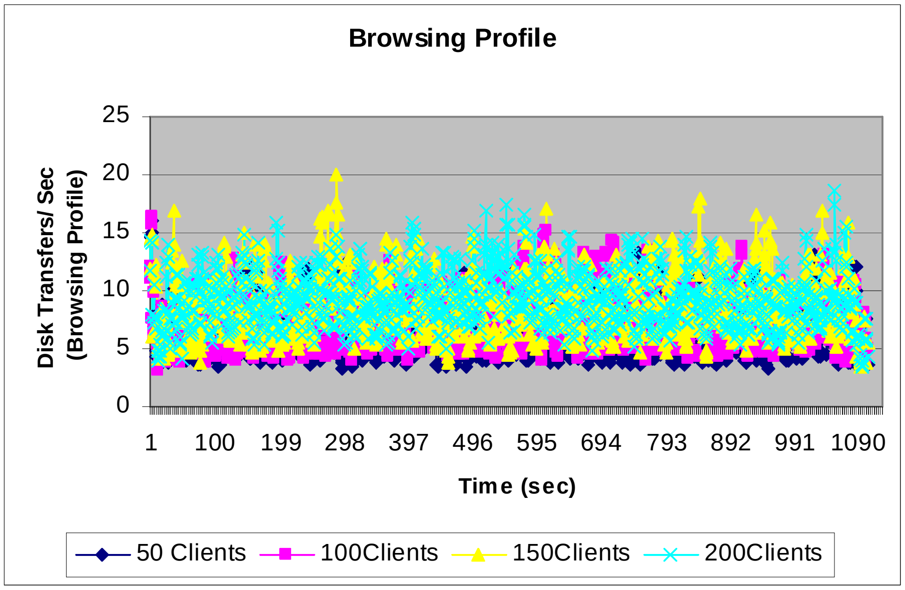


Fig 8: CPU Utilization in Browsing & Ordering Profiles at Application Server 2

As expected, we observe from Figure 8 that the CPU utilization increases with the number of clients for both Ordering and Browsing profiles. One of the reasons for increased response time of Web services components under higher workload is the overhead incurred due to parsing and encoding the XML messages which leads to increased CPU utilization.

In this application, both the web services as well as the EJB components make database calls to satisfy user requests. We therefore study the disk activity on the Database server (i.e., Server 3 in Figure 9). It can be observed from Figure 9 that the number of disk transfers per second for both the Ordering profile and Browsing profile increases with the workload intensity, which clearly explains the increase in the response times of Web services and EJB components.



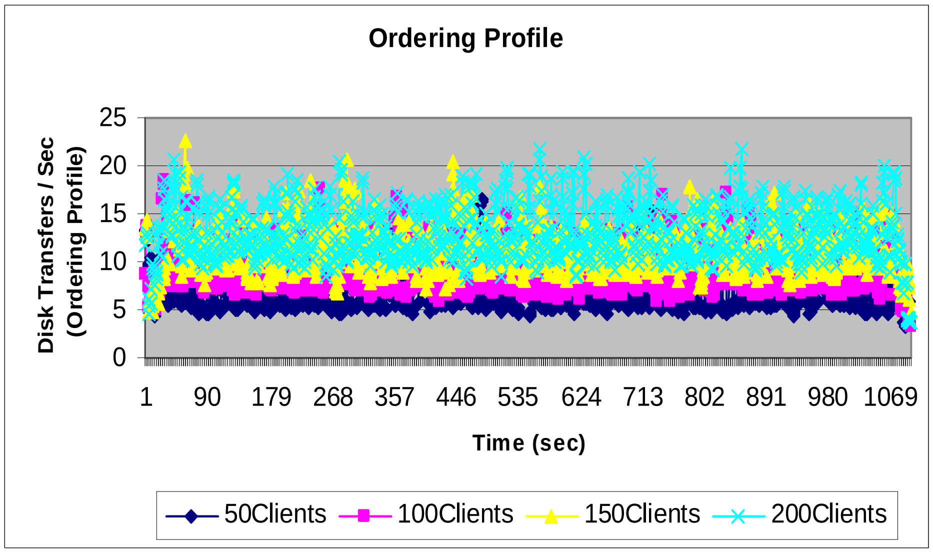


Fig 9: Database Disk activity in Browsing & Ordering Profiles at Application Server 3

Chapter 8: Conclusion

The work presented in this thesis focuses on the performance aspect of web services and studies their contribution to the overall server-side response time when integrated with other middle-tier components like Enterprise Java Beans. A three-tier e-commerce application representing an online bookstore was developed for this purpose and was deployed on several machines. A synthetic workload was used to simulate the activities of online users browsing an e-commerce site.

The results for two workload profiles – Browsing and Ordering are presented. The goal of this thesis was to study the performance of the web services components and the EJB components and the effect they have on the hardware resources of the system. Measurements at the software architecture level were taken by instrumenting the application to log the component level response times in the application. These measurements helped understand the behavior of the web service and EJB components. Hardware resource usage was measured using the Windows 2000 Performance Monitoring tool. Measurements at the hardware level helped study the effect the web services and EJB components had on the hardware resources.

From the results it can be seen that the web services components had higher response times than the EJB components when the size of the data being returned was large. Some of the web service methods showed response times almost similar to the response times of the EJB components. This could be attributed to the fact that in these cases the size of the data returned by the methods was small.

The web service components introduce a contention for database resources that affects the performance of the other web services and EJB components as well. Also the size of the data returned by a web service introduces additional overhead in the marshaling and unmarshalling of the XML payload in the SOAP messages.

Performance is an important quality of service attribute. This thesis studies the performance aspect of an e-commerce application that uses Web services to execute business operations. By focusing on the software architectural view of the e-commerce prototype we analyze the performance aspects of Web services components under controlled workload conditions. We also measure the impact of the application execution on the hardware resources of the system. Such a study allows developers of large applications to identify potential bottlenecks in the systems. As web service technologies become more and more popular, a very significant factor that will contribute to their adoption in large enterprise systems is the performance.

References

1. D.A. Menascé and V.A.F. Almeida, “Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning”, *Prentice Hall, 2000*.
2. K. Gottschalk, S. Graham, H. Kreger, and J. Snell “Introduction to Web services architecture”, *IBM Systems Journal, Volume 41, Number 2, 2002, New Developments in Web Services and E-commerce,* pp. 2-3
3. Martin F. Arlitt and Carey L. Williamson, “Internet Web Servers: Workload Characterization and Performance Implications”, *IEEE/ACM Transactions on Networking, Vol. 5, No. 5, October 1997.* pp. 1-15
4. Ronald C. Dodge, Daniel A. Menascé, Daniel Barbara, “Testing E-Commerce Site Scalability with TPC-W”, Proc. *2001 Computer Measurement Group Conference, Orlando, FL, Dec., 2001.*
5. Daniel A. Menascé, Virgilio A. F. Almeida, “A Methodology for Workload Characterization of E-commerce Sites”, 1999, *ACM*
6. TPC-W Transactional Web Commerce Benchmark, Transaction Processing Performance Council, http://www.tpc.org/tpcw.
7. GNU AWK utility, http://www.gnu.org/software/gawk/gawk.html
8. Yutu Liu, Anne H.H Ngu, Liangzhao Zeng, “QoS Computation and Policing in Dynamic Web Service Selection”, *WWW May 2004, ACM*
9. Chintan Patel, Kaustubh Supekar, and Yugyung Lee, “A QoS Oriented Framework for Adaptive Management of Web Service based Workflows”, School of Interdisciplinary Computing and Engineering, University of Missouri-Kansas City, pp. 2-10
10. Shuping Ran, “A Model for Web Services Discovery With QoS”, CSIRO Mathematical and Information Sciences, GPO Box 664, Canberra, ACT 2601, Australia, *ACM 200,* pp. 2-10
11. , L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, “Quality Driven Web Services Composition”, *In Proceedings of the Twelfth International World Wide Web Conference (WWW'2003).* pp.1-11

12. Daniel A. Menascé, Virgilio A. F. Almeida, “Capacity Planning: An Essential Tool For Managing Web Services”, *IEEE 2002, IT Pro July-August 2002,* pp. 38

13. Kai S. Juse, S. Kuonev, and A. Buchmann, “Petstore-WS: Measuring The Performance Implications of Web Services”, *29th International Conference of the Computer Measurement Group (CMG) on Resource Management and Performance Evaluation of Enterprise Computing Systems*, December 2003.

14. Marin Litoui, “Migrating to Web Services: A Performance Engineering Approach”, Center for Advanced Studies, IBM Toronto Laboratory

15. Christina Cately, Dorina C. Petriu, Monique Frize, “Software Performance Engineering of a Web Service-Based Clinical Decision Support Infrastructure”, *4th International Workshop on Software Performance, Redwood Shores, California, 2004,* pp 130-138

16. Senthilanand Chandrasekaran, Gregory Silver, John A. Miller, Jorge Cardoso, Amit P. Sheth, “Web Services and their Synergy with Simulation”, *Proceedings of the 2002 Winter Simulation Conference,* pp.6-10

17. M. Tian, T. Voigt, T. Naomowicz, H. Ritter, J. Schiller, “Performance Considerations for Mobile Web Services”, Freie Universitat, Berlin, Germany, pp. 1-12

18. D Venu, K Goseva-Popstojanova, “Measurement based Performance Analysis of Ecommerce Applications with Web services components”, *IEEE International Conference on E-business Engineering, 2005.*

19. M. Tian, T. Voigt, T. Naomowicz, H. Ritter, J. Schiller, “A Concept for QoS Considerations in Web Services”, *Proceedings on the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW ’03)*

20. M. Tian, T. Voigt, T. Naomowicz, H. Ritter, J. Schiller, “Performance Impact of Web Services on Internet Servers”, *International Conference on Parallel and Distributed Computing and Systems*, Marina Del Rey, USA, Nov. 2003.

21. Sun Microsystems -- Java 2 Platform Enterprise Edition Specification v1.4, http://java.sun.com/j2ee/j2ee-1/_4-fr-spec.pdf.

22. Sun Microsystems – EJB specification, http://java.sun.com/products/ejb

23. Sun Microsystems, Java Logging API's, http://java.sun.com/j2se/1.4.2/docs/guide/util/logging.
24. Web Services Architecture, Working Draft-2002, http://www.w3.org/TR/wd-ws-arch/
25. Web Services Description Language (WSDL) 1.1, W3C March 2001, http://www.w3c.org/TR/wsdl
26. UDDI Technical White paper, uddi.org, 2001-2002.
27. Barnes Bookstore Web Service, www.xmethods.net/Barnes Case Study
28. Venu Datla, "Measurements based Performance Analysis of Web Service", Master's Thesis, West Virginia University, 2005

APPENDIX 1

1. WSDL for BookStore Web Service

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="MyInventoryService" targetNamespace="urn:Foo"
    xmlns:tns="urn:Foo" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ns2="http://java.sun.com/jax-rpc-ri/internal"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
- <types>
- <schema targetNamespace="http://java.sun.com/jax-rpc-ri/internal"
    xmlns:tns="http://java.sun.com/jax-rpc-ri/internal" xmlns:soap11-
    enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
  <import namespace="urn:Foo" />
- <complexType name="vector">
- <complexContent>
- <extension base="tns:list">
  <sequence />
    </extension>
    </complexContent>
    </complexType>
- <complexType name="list">
- <complexContent>
- <extension base="tns:collection">
  <sequence />
    </extension>
    </complexContent>
    </complexType>
- <complexType name="collection">
- <complexContent>
- <restriction base="soap11-enc:Array">
  <attribute ref="soap11-enc:arrayType" wsdl:arrayType="anyType[]" />
    </restriction>
    </complexContent>
    </complexType>
    </schema>
- <schema targetNamespace="urn:Foo" xmlns:tns="urn:Foo" xmlns:soap11-
    enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
  <import namespace="http://java.sun.com/jax-rpc-ri/internal" />
- <complexType name="Book">
- <sequence>
```

```
  <element name="ISBN" type="string" />
  <element name="a_firstname" type="string" />
  <element name="a_id" type="int" />
  <element name="a_lastname" type="string" />
  <element name="barnesPrice" type="float" />
  <element name="bbarnesprice" type="float" />
  <element name="bisbn" type="string" />
  <element name="bprice" type="double" />
  <element name="bpublisher" type="string" />
  <element name="bstockAvailable" type="int" />
  <element name="btitle" type="string" />
  <element name="i_id" type="int" />
  <element name="price" type="double" />
  <element name="publisher" type="string" />
  <element name="stock" type="int" />
  <element name="title" type="string" />
    </sequence>
    </complexType>
    </schema>
    </types>
- <message name="CheckInventoryIF_checkStock">
  <part name="String_1" type="xsd:string" />
  <part name="double_2" type="xsd:double" />
    </message>
- <message name="CheckInventoryIF_checkStockResponse">
  <part name="result" type="xsd:int" />
    </message>
- <message name="CheckInventoryIF_getBookAuthor">
  <part name="String_1" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_getBookAuthorResponse">
  <part name="result" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_getBookByAuthor">
  <part name="String_1" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_getBookByAuthorResponse">
  <part name="result" type="ns2:vector" />
    </message>
- <message name="CheckInventoryIF_getBookByISBN">
  <part name="String_1" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_getBookByISBNResponse">
  <part name="result" type="tns:Book" />
    </message>
- <message name="CheckInventoryIF_getBookByTitle">
  <part name="String_1" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_getBookByTitleResponse">
```

```
  <part name="result" type="ns2:vector" />
    </message>
- <message name="CheckInventoryIF_getBookPrice">
  <part name="String_1" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_getBookPriceResponse">
  <part name="result" type="xsd:double" />
    </message>
- <message name="CheckInventoryIF_getBookPublisher">
  <part name="String_1" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_getBookPublisherResponse">
  <part name="result" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_getBookStock">
  <part name="String_1" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_getBookStockResponse">
  <part name="result" type="xsd:int" />
    </message>
- <message name="CheckInventoryIF_getBookStockByID">
  <part name="int_1" type="xsd:int" />
    </message>
- <message name="CheckInventoryIF_getBookStockByIDResponse">
  <part name="result" type="xsd:int" />
    </message>
- <message name="CheckInventoryIF_getBookTitle">
  <part name="String_1" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_getBookTitleResponse">
  <part name="result" type="xsd:string" />
    </message>
- <message name="CheckInventoryIF_updateInventory">
  <part name="int_1" type="xsd:int" />
  <part name="String_2" type="xsd:string" />
    </message>
  <message name="CheckInventoryIF_updateInventoryResponse" />
- <portType name="CheckInventoryIF">
- <operation name="checkStock" parameterOrder="String_1 double_2">
  <input message="tns:CheckInventoryIF_checkStock" />
  <output message="tns:CheckInventoryIF_checkStockResponse" />
    </operation>
- <operation name="getBookAuthor" parameterOrder="String_1">
  <input message="tns:CheckInventoryIF_getBookAuthor" />
  <output message="tns:CheckInventoryIF_getBookAuthorResponse" />
    </operation>
- <operation name="getBookByAuthor" parameterOrder="String_1">
  <input message="tns:CheckInventoryIF_getBookByAuthor" />
  <output message="tns:CheckInventoryIF_getBookByAuthorResponse" />
```

```
   </operation>
- <operation name="getBookByISBN" parameterOrder="String_1">
  <input message="tns:CheckInventoryIF_getBookByISBN" />
  <output message="tns:CheckInventoryIF_getBookByISBNResponse" />
   </operation>
- <operation name="getBookByTitle" parameterOrder="String_1">
  <input message="tns:CheckInventoryIF_getBookByTitle" />
  <output message="tns:CheckInventoryIF_getBookByTitleResponse" />
   </operation>
- <operation name="getBookPrice" parameterOrder="String_1">
  <input message="tns:CheckInventoryIF_getBookPrice" />
  <output message="tns:CheckInventoryIF_getBookPriceResponse" />
   </operation>
- <operation name="getBookPublisher" parameterOrder="String_1">
  <input message="tns:CheckInventoryIF_getBookPublisher" />
  <output message="tns:CheckInventoryIF_getBookPublisherResponse" />
   </operation>
- <operation name="getBookStock" parameterOrder="String_1">
  <input message="tns:CheckInventoryIF_getBookStock" />
  <output message="tns:CheckInventoryIF_getBookStockResponse" />
   </operation>
- <operation name="getBookStockByID" parameterOrder="int_1">
  <input message="tns:CheckInventoryIF_getBookStockByID" />
  <output message="tns:CheckInventoryIF_getBookStockByIDResponse" />
   </operation>
- <operation name="getBookTitle" parameterOrder="String_1">
  <input message="tns:CheckInventoryIF_getBookTitle" />
  <output message="tns:CheckInventoryIF_getBookTitleResponse" />
   </operation>
- <operation name="updateInventory" parameterOrder="int_1 String_2">
  <input message="tns:CheckInventoryIF_updateInventory" />
  <output message="tns:CheckInventoryIF_updateInventoryResponse" />
   </operation>
   </portType>
- <binding name="CheckInventoryIFBinding" type="tns:CheckInventoryIF">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
- <operation name="checkStock">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </output>
   </operation>
- <operation name="getBookAuthor">
  <soap:operation soapAction="" />
```

```
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </output>
    </operation>
- <operation name="getBookByAuthor">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </output>
    </operation>
- <operation name="getBookByISBN">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </output>
    </operation>
- <operation name="getBookByTitle">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </output>
    </operation>
- <operation name="getBookPrice">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </input>
- <output>
```

```
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </output>
   </operation>
- <operation name="getBookPublisher">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </output>
   </operation>
- <operation name="getBookStock">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </output>
   </operation>
- <operation name="getBookStockByID">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </output>
   </operation>
- <operation name="getBookTitle">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
   </output>
   </operation>
- <operation name="updateInventory">
  <soap:operation soapAction="" />
```

```
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     use="encoded" namespace="urn:Foo" />
    </output>
    </operation>
    </binding>
- <service name="MyInventoryService">
- <port name="CheckInventoryIFPort" binding="tns:CheckInventoryIFBinding">
  <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </port>
    </service>
    </definitions>
```

2. WSDL for Credit Card Service

```
    <?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="MyCreditCardService" targetNamespace="urn:Foo"
     xmlns:tns="urn:Foo" xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
- <types>
- <schema targetNamespace="urn:Foo" xmlns:tns="urn:Foo" xmlns:soap11-
     enc="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
- <complexType name="ArrayOfstring">
- <complexContent>
- <restriction base="soap11-enc:Array">
  <attribute ref="soap11-enc:arrayType" wsdl:arrayType="string[]" />
    </restriction>
    </complexContent>
    </complexType>
    </schema>
    </types>
- <message name="CreditCardIF_getcarddetails">
  <part name="String_1" type="xsd:string" />
    </message>
- <message name="CreditCardIF_getcarddetailsResponse">
  <part name="result" type="tns:ArrayOfstring" />
    </message>
- <message name="CreditCardIF_verifycard">
  <part name="String_1" type="xsd:string" />
  <part name="String_2" type="xsd:string" />
  <part name="String_3" type="xsd:string" />
  <part name="String_4" type="xsd:string" />
    </message>
- <message name="CreditCardIF_verifycardResponse">
  <part name="result" type="xsd:string" />
    </message>
- <portType name="CreditCardIF">
- <operation name="getcarddetails" parameterOrder="String_1">
  <input message="tns:CreditCardIF_getcarddetails" />
  <output message="tns:CreditCardIF_getcarddetailsResponse" />
    </operation>
- <operation name="verifycard" parameterOrder="String_1 String_2 String_3
    String_4">
  <input message="tns:CreditCardIF_verifycard" />
  <output message="tns:CreditCardIF_verifycardResponse" />
    </operation>
    </portType>
- <binding name="CreditCardIFBinding" type="tns:CreditCardIF">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
- <operation name="getcarddetails">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
    </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
    </output>
    </operation>
- <operation name="verifycard">
  <soap:operation soapAction="" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
    </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo" />
    </output>
    </operation>
    </binding>
- <service name="MyCreditCardService">
- <port name="CreditCardIFPort" binding="tns:CreditCardIFBinding">
  <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </port>
    </service>
    </definitions>
```