ANALYSIS OF WEB SERVICES ON J2EE APPLICATION SERVERS

Adarsh Kumar Gosu, B.E.

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

May 2004

APPROVED:

Robert Brazile, Major Professor
Kathleen Swigger, Committee Member
Karl Steiner, Committee Member
Krishna Kavi, Chair of the Department of Computer Science
Oscar N. Garcia, Dean of the College of Engineering
Sandra L. Terrell, Interim Dean of the Robert B. Toulouse School of Graduate Studies

Gosu, Adarsh Kumar, Analysis of Web Services on J2EE Application Servers. Master of Science (Computer Science), May 2004, 62 pp., 1 table, 4 figures, references, 27 titles.

The Internet became a standard way of exchanging business data between B2B and B2C applications and with this came the need for providing various services on the web instead of just static text and images. Web services are a new type of services offered via the web that aid in the creation of globally distributed applications. Web services are enhanced e-business applications that are easier to advertise and easier to discover on the Internet because of their flexibility and uniformity.

In a real life scenario it is highly difficult to decide which J2EE application server to go for when deploying a enterprise web service. This thesis analyzes the various ways by which web services can be developed & deployed. Underlying protocols and crucial issues like EAI (enterprise application integration), asynchronous messaging, Registry tModel architecture etc have been considered in this research. This paper presents a report by analyzing what various J2EE application servers provide by doing a case study and by developing applications to test functionality.

ACKNOWLEDGEMENTS

I want to thank my advisor Dr. Robert Brazile, for his guidance, patience, insightful ideas and suggestions throughout my thesis work. His valuable guidance throughout my graduate life is whole-heartedly appreciated.

Sincere thanks to Dr. Kathleen Swigger and Dr. Karl Steiner for taking time off their busy schedules to be a part of my thesis committee. I would also like to thank the Computer Science department and faculty for providing me an opportunity to pursue my master's degree at University of North Texas and work on various edge cutting technologies. I would like to thank my manager & my colleagues at Sabre Holdings for helping me in various ways in getting the thesis work done.

Finally and most importantly, I would like to express my deep appreciation to my parents, brother and friends for their moral support and encouragement, without which I would not have reached this stage.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

TRADEMARKS

List of Trademarks used in this thesis:

Registered Trademarks of BEA Systems, Inc. :

- *BEA*
- *WebLogic*
- *Tuxedo*

Trademarks of BEA Systems, Inc. :

- *BEA WebLogic Server*
- *BEA WebLogic Integration*
- *BEA WebLogic Portal*
- *BEA WebLogic Enterprise Platform*

Registered Trademarks of Oracle Corporation :

- *Oracle*

Trademarks of Oracle Corporation :

- *Oracle9i*

- *PL/SQL*

Registered Trademarks of International Business Machines Corporation:

- *IBM*
- *DB2*
- *MQSeries*
- *VisualAge*
- *WebSphere*

Trademarks of International Business Machines Corporation :

- *Cloudscape*
- *DB2 Universal Database*

Registered Trademarks & Trademarks of Sun Microsystems, Inc. :

- *J2EE*
- *Java*
- *JDBC*
- *JMS*

- *Enterprise JavaBeans*
- *SOAP with Attachments API for Java*
- *SUN ONE Application Server*
- *SUN Java System Application Server*
- *SUN ONE Message Queue*
- *JAXM*
- *JAX-RPC*
- *JAXR*
- *JAXP*

CHAPTER 1

INTRODUCTION

With the rapid growth of the Internet in the past few years, came new ways of using the Web. The Internet, which primarily used to be a source of static information, gave birth to sophisticated interactions like e-commerce applications.

Currently Web took a new face with application to application interactions without or little human interference. Business organizations are exchanging huge chunks of data online which can be secure or non-secure in nature. Irrespective of the type of data there needs to be a common way to exchange and handle this data between various organizations. Web services provide a pivoted solution to this problem. Web services are services offered via the Web that aid organizations in creation of distributed applications. Web services are enhanced e-business applications that are easier to advertise and easier to discover on the Internet because of their flexibility and uniformity. Web service popularity is because of their inherent features like interoperability, efficient application integration, flexibility, uniformity, security, use of existing standards and protocols.

Web services are different from Web applications, Web services use SOAP (Simple Object Access Protocol)[1] messages instead of standard MIME (Multipurpose

Internet Mail Extensions) messages, Web services are not HTTP(Hypertext Transport Protocol) specific and Web services provide metadata describing the messages they produce and consume.

All potential Web services needs to be registered in UDDI (Universal Description, Discovery and Integration) [2] registry so that service requestors can find the service they need. Web services publish their information to requestors using WSDL (Web Services Description Language)[3]. The WSDL document contains information like the name of the service, the operations that the service provides to the requestor and the location where the service requestor has to send the request to consume the Web service. A business application sends a request for service to a given URL (uniform resource locator) using SOAP message over HTTP. The Web service receives the request, processes the request and returns a response again as a SOAP message. The sending and receiving of messages can be synchronous or asynchronous in nature based on the type of the service.

XML (Extensible Markup Language) is considered as a standard language for all Web service applications because of its portability and interoperability. Currently most of the Web services are deployed on J2EE (Java 2 Platform Enterprise Edition) application servers since they are scalable, secure and efficient even with the growth of the service. All major J2EE application server developers like BEA®, IBM®, Sun®, Oracle®etc have come up with their support for Web services based on the standards

specified by W3C (World Wide Web consortium)[9]. In a practical scenario it is highly difficult to decide which application server to go for when deploying an enterprise Web service. In order to overcome this problem analysis of various application servers with regards to Web services has been done and surveyed in this research.

1.1 Organization of the Material

Chapter 2 describes Web services, it's architecture and the protocols associated with it.

Chapter 3 describes the J2EE architecture, provides an overview of the application servers and their functionality.

Chapter 4 gives a brief explanation of each and every parameter used in this thesis and the results of the thesis.

Chapter 5 presents a summary of the thesis along with a discussion of future work.

CHAPTER 2

Web Services

Web services are a way to convert Web from a human centric to application centric. With the advent of the Web services the face of B2B (business-to-business) , B2C (business-to-consumer) and EAI (enterprise application integration) has totally changed. Web services can be defined as:

- Web service can be treated as a network accessible interface to a Web application.
- Web service is a way to automate various Web application interactions with minimum human intervention.
- Web service can be treated as an application that interacts with other systems of the network using XML (Extensible Markup Language) messages via lightweight vendor neutral protocols.
- Web service is a system that can be identified by an URI (Universal Resource Identifier), whose public methods and bindings are defined by a WSDL (Web Services Description Language) document and published in a UDDI (Universal Description, Discovery and Integration) registry.

- Web services provide flexibility for applications that run on different hardware and softwares to interact with each other using minimum human intervention.

Web services use a stack of protocols that are developed by vendor neutral organizations like W3C (World Wide Web consortium) and OASIS(Organization for the Advancement of Structured Information Standards).

Web services represent the next step beyond the existing distributed computing protocols like CORBA (Common Object Request Broker Architecture), RMI(Remote Method Invocation), COM(Component Object Model), DCOM(Distributed Component Object Model), RPC(Remote Procedure Call) and sockets etc. Most of the existing distributed protocols provide tight coupling between applications, language dependent, vendor dependent or provide firewall & security problems while integrating. Web services overcome all these problems by providing a flexible & most efficient way to handle distributed computing.

Web service components are loosely coupled and it is possible for a client to query the service broker at run-time, get the interface to a service, and bind to it without having to do any hard-coding of the URL or method names.

The basic Web services are a messaging framework. The most critical requirement of a Web service is it should be able to send and receive XML messages between applications.

The architectural model behind Web services is they are loosely coupled, service-oriented architecture and vendor neutral protocols. The service-oriented architecture is satisfied by declaring the interface of the application as a WSDL document that serves as a contract between the service provider and the service requester. The loosely coupled architecture can be achieved by using vendor neutral protocols like SOAP(Simple Object Access Protocol), UDDI, and WSDL on top of transport protocols like HTTP (Hypertext Transport Protocol), SMTP (Simple Mail Transfer Protocol), and IIOP(Internet Inter-ORB Protocol).

2.1 Web Service Architecture

The three basic components of the Web service are

Registry: Registry is a broker for Web services. This is a logically centralized directory of Web services. The registry is a central repository where providers can publish Web services and consumers can search and consume services. [10]

Service Provider: A service provider publishes a description of the services it provides to the registry. [10]

Service Requester: The service requester searches the registry to find services it need and uses them accordingly. [10]

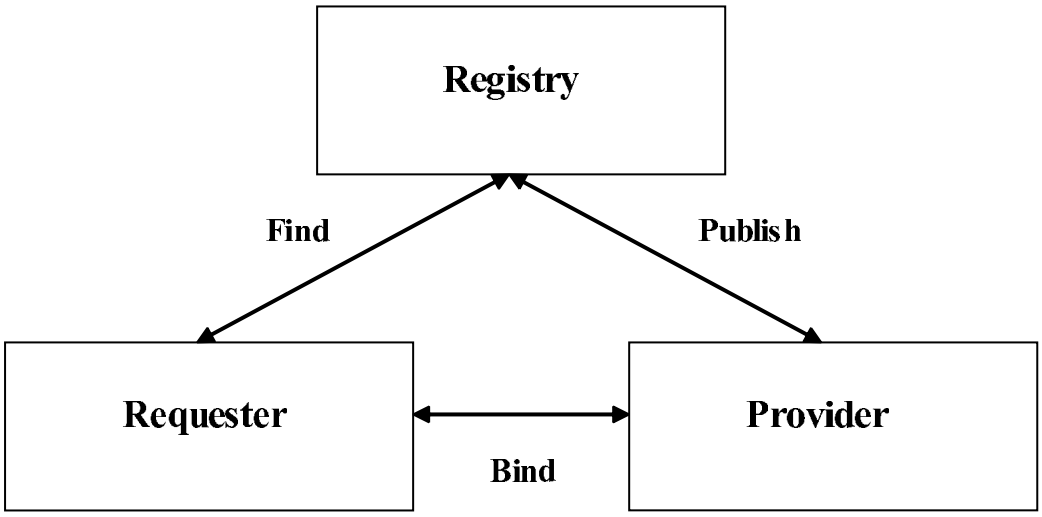

Figure 2.1: Web Service Architecture (Adapted from [10])

2.2 Web Service Conceptual Stack

The Web service conceptual stack can be considered as a layer of different technologies stacked upon one another.

Service Discovery: This layer provides the mechanism for service requestors to fetch the UDDI document of the service providers. The discovery of a service depends on the way it is published in the UDDI registry. [10]

Service Publication: This layer provides the functionality such that the WSDL document is available to a service requestor. In a direct publication mode the service provider can directly send the WSDL document to the service requestor. In an indirect mode the service provider publishes the WSDL document to an UDDI registry and the service requestor retrieves it. [10]

Service Description: Description of the service describes the way service requester can contact and use the service. WSDL is a standard that is used to provide the description information. This layer is actually a stack of description documents. [10]

XML Based Messaging : XML based messaging is a way of packing the data that is moved across applications so that they can understand the data . SOAP over XML is used in this layer. [10]

Network: The network layer is used to do the basic communication, addressing and routing etc. Some protocols for transport used by the network layer are TCP, HTTP, SMTP etc. [10]

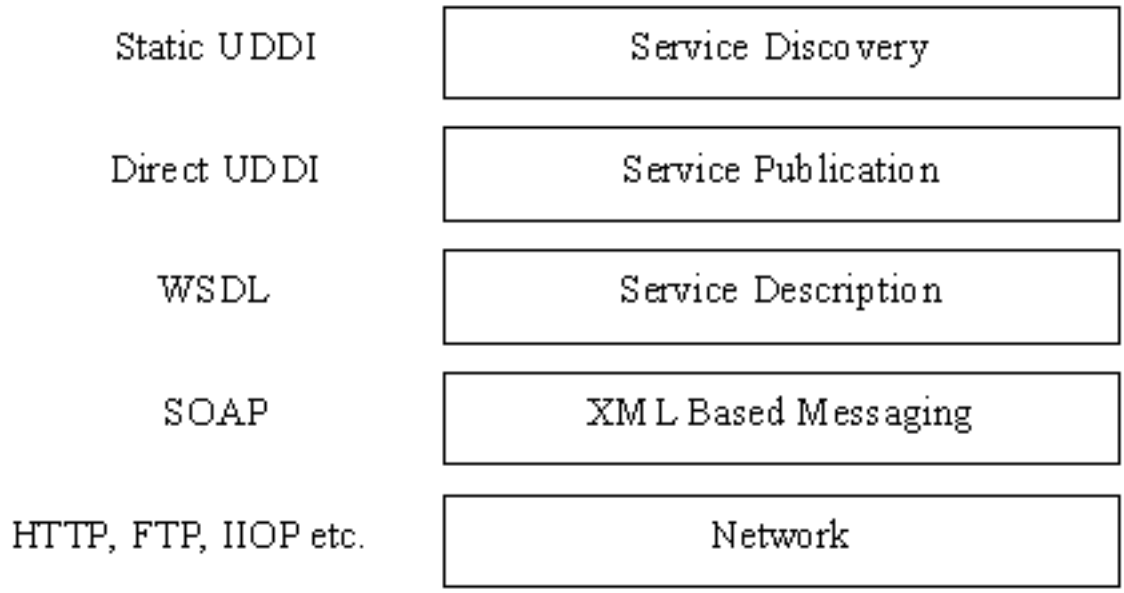

Figure 2.2: Web Service Conceptual Stack (Adapted from [10])

2.3 Web Service Protocols

2.3.1 Simple Object Access Protocol (SOAP)

SOAP is a XML based communication protocol that facilitates the interaction between applications and Web services located on remote computers. SOAP is based on XML and uses XML standards like XML Schema and XML Namespaces for its definition and function.

SOAP Messages: SOAP message is based on XML specification. SOAP messages can be categorized as request messages and response messages.

The basic building blocks of a SOAP message are

- *SOAP Envelope:*The SOAP envelope element is the root element of any SOAP message. It consists of an optional SOAP header and a mandatory SOAP body. [5] [23]
- *SOAP Header :*The SOAP header element is optional and offers a flexible framework for specifying additional application level requirements. The header provides a mechanism for authentication, transaction management, state information and routing etc. [5] [23]
- *SOAP Body:* The SOAP body element is mandatory for all SOAP messages. The body element contains a XML document of the sender if it is a request message or a XML document of the receiver if it is a response message. [5] [23]
- *SOAP Faults:* SOAP faults are used in case of an error. The response SOAP body element contains the appropriate fault in case of an error. [5] [23]

SOAP Attachments: SOAP attachments element is an optional part. It is generally used to transmit non XML data like images, zip files etc along with the SOAP messages. It can also be used to transmit XML files that are not part of SOAP body element. There can be any number of SOAP attachment elements in a SOAP message. [5] [23]

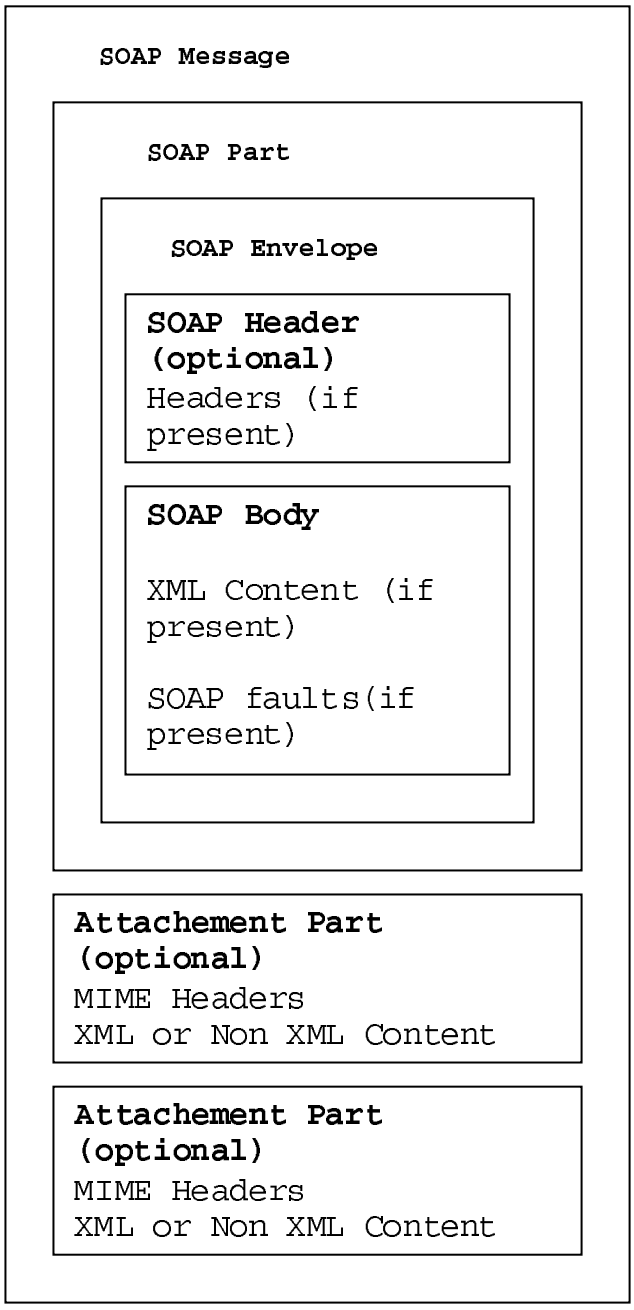

Figure 2.3: SOAP Message Structure (Used with permission) [23]

A sample SOAP request message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope
xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<soap-env:Header/>
<soap-env:Body>
<ResearchStock:GetStockPrice
xmlns:ResearchStock="http://cs.unt.edu/~akg0011/ResearchStock">
<ResearchStock:symbol>Sabre</ResearchStock:symbol>
</ResearchStock:GetStockPrice> </soap-env:Body>
</soap-env:Envelope>
```

A Sample SOAP response message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope
xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<soap-env:Header/>
<soap-env:Body>
<Response>Response for the Stock
          Research Price Request Message </Response>
</soap-env:Body>
</soap-env:Envelope>
```

2.3.2 Web Service Description Language (WSDL)

WSDL document contains the public interface and binding definitions of a Web service. WSDL uses XML specification. WSDL standardizes XML elements that describe a collection of communication endpoints.

The basic XML elements of a WSDL are:

Type Element: The type element contains the definitions of the data types used within the WSDL document message. The data types use the XML schema so that they can be independent. [3] [11]

Message Element: The message element identifies the body of the WSDL document. [3] [11]

PortType Element: The porttype element describes the abstract operations and messages. The element operation is used to uniquely identify the operation within the port type element. [3] [11]

```
<wsdl:operation name="getPrice" parameterOrder="in0">
 <wsdl:input message="intf:getPriceRequest" name="getPriceRequest"/>
 <wsdl:output message="intf:getPriceResponse" name="getPriceResponse"/>
</wsdl:operation>
```

Binding Element: Binding element defines the protocols and the message formats that are referenced in the PortType element. There can be many bindings in a WSDL document. [3] [11]

```
<wsdl:binding name="StockSoapBinding" type="intf:StockInterface">
 <wsdlsoap:binding style="rpc"
                transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="getPrice">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getPriceRequest">
      <wsdlsoap:body namespace="http://stockSession.stock.research"
                     use="literal"/>
  </wsdl:input>
```

```
  <wsdl:output name="getPriceResponse">
     <wsdlsoap:body namespace="http://stockSession.stock.research"
                    use="literal"/>
  </wsdl:output>
</wsdl:operation>
</wsdl:binding>
```

Port Element: The port element identifies the end points of the Web service. It contains the URI of the Web service. [3] [11]

```
<wsdl:port binding="intf:StockSoapBinding" name="Stock">
 <wsdlsoap:address  location=
     "http://localhost:6080/StockWebService/services/Stock"/>
</wsdl:port>
```

Service Element: The service element is used to group all the port elements inside a WSDL document. [3] [11]

```
<wsdl:service name="StockInterfaceService">
 <wsdl:port binding="intf:StockSoapBinding" name="Stock">
  <wsdlsoap:address location=
     "http://localhost:6080/StockWebService/services/Stock"/>
```

```
 </wsdl:port>
</wsdl:service>
```

2.3.3 Universal Description, Discovery and Integration (UDDI)

Web services are registered with a registry along with the name of the service, its description, interfaces etc and this registry is called UDDI registry. The Web service providers register these services with the UDDI registry so that global access to the Web service can be provided, the consumer of the service looks up the registry to find the services he need and consumes them. Each Web service that is published with a registry server is assigned a unique key so that services with same name can be registered in the registry server.

UDDI entries are called as tModel . A tModel is a metadata describing

- Name of business.
- Business contact information.
- Industry code.
- Description of Web service.
- Classification.
- Access rights to Web service.

- Public interfaces & properties.

The UDDI version 2 provides a fixed set of API calls that can be used to access any UDDI complaint registry. They are

find_binding : This is used when a Web service client wants to find out how to bind to a particular Web service. [2] [11]

find_business : Used to find business that are registered in the UDDI registry. [2] [11]

find_relatedBusinesses : Used to find information about one or more registrations that are related to the same business entity is the UDDI registry. [2] [11]

find_service : Used to find one or more services of a business entity in the UDDI registry. [2] [11]

find_tModel : Used to find one or more tModel information structures in the UDDI registry. [2] [11]

Get_bindingDetail : Used to retrieve the full runtime binding template information from the UDDI registry. [2] [11]

Get_businessDetail : Used to retrieve full business entity information from business that are registered in the UDDI registry. [2] [11]

Get_businessDetailExt : Used to retrieve extended information about a business entity from the UDDI registry. [2] [11]

Get_serviceDetail : Used to retrieve all the details related to a service registered in the UDDI registry. [2] [11]

Get_tModelDetail : Used to retrieve full details of a specified tModel from the UDDI registry. [2] [11]

CHAPTER 3

Application Servers

3.1 J2EE Architecture

J2EE(Java 2 Platform, Enterprise Edition) is a multi-tier distributed architecture developed by Sun Microsystems, Inc. All the application servers that currently exist inherit the multi tier J2EE architecture.

A J2EE component is a self-contained functional software unit that is assembled into a J2EE application with its related classes and XML descriptor files that communicate with other components.

The various tiers are:

Client-tier : Client applications that runs on client machine come under this tier. [11]

Web-tier Components : Components like JSP(JavaServer Pages), servlet that run on J2EE server. [11]

Business-tier Components : Components like session beans, entity beans, message driven beans runs in this tier of the J2EE server. [11]

Enterprise Information System (EIS) tier : Components like databases, ERP(Enterprise Resource Planning) etc runs on EIS tier. [11]

J2EE containers are the interface between a component and the low-level platform specific functionality that supports the J2EE components. All J2EE components must be assembled into J2EE applications and deployed into appropriate J2EE containers. While assembling the components container specific settings should be set. Some of them include security, transaction model, remote connectivity, database connectivity, naming and directory lookup.

The J2EE containers that exist currently are:

Enterprise JavaBeans container : This container is used to hold the enterprise beans and message-driven beans for J2EE applications.

Web Container : This container manages the execution of JavaServlets and JaveServer pages. This container is also responsible to providing the JSP & servlet access to bean container.

3.2 Application Servers

An application server is a software server program in a distributed network that provides the business logic for an application program. Application servers provide middleware services for security, state maintenance, database access and persistence.

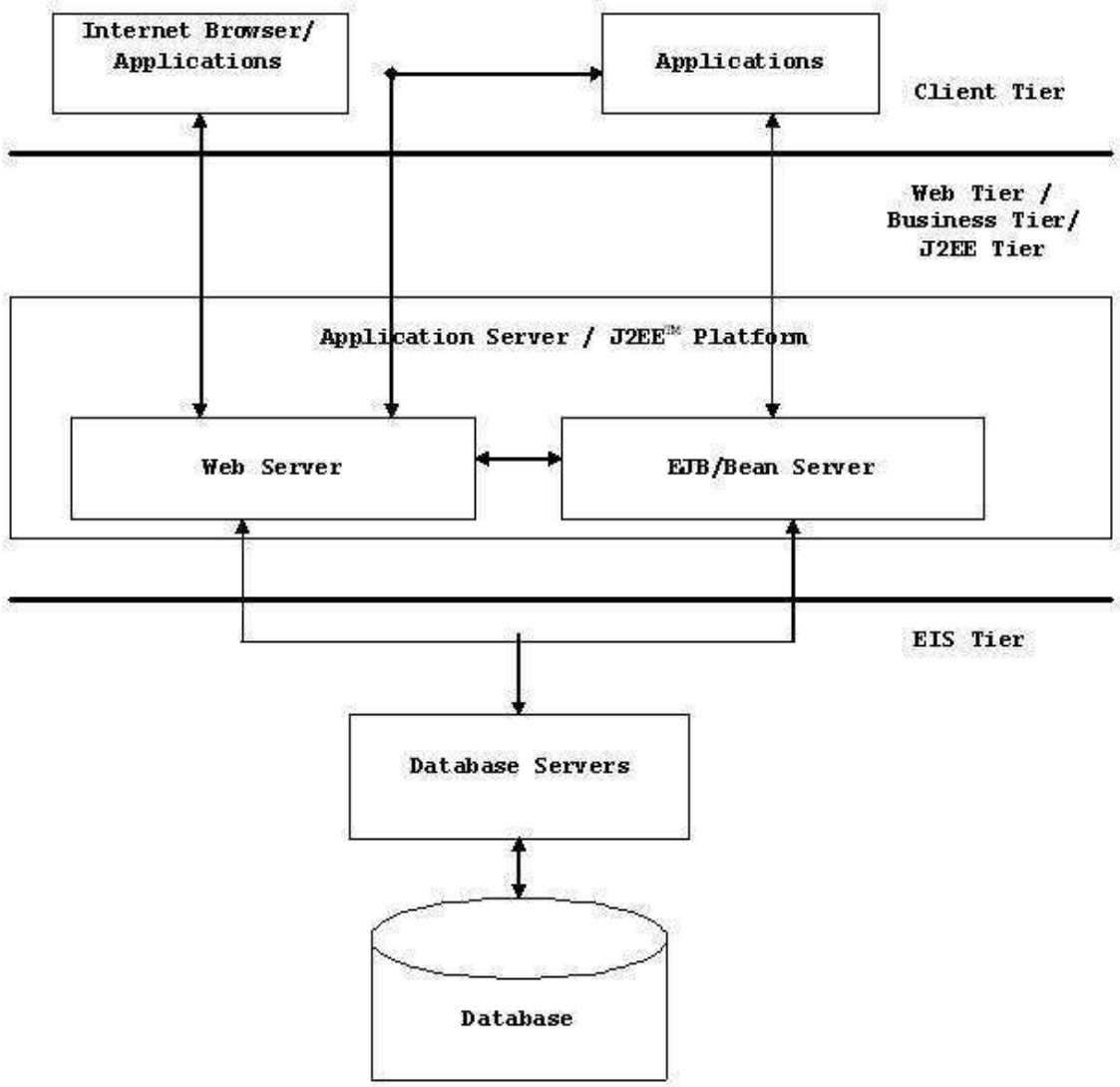

Figure 3.1: J2EE Architecture

Application server also handles all application operations between users and back-end business applications or databases. Application servers handle complex transactions while providing stability, redundancy, high availability, high performance and complex database querying capability.

Application servers are also called as appservers. The application servers are based on the Java platform and J2EE architecture. The database access is made through JDBC (Java Database Connectivity), SQLJ or JDO API.

Some of the functions and services that application servers provide are:

- Load balancing.
- Web service capability.
- Fault tolerance.
- Network transparency.
- Transaction management.
- Security.
- Synchronous & asynchronous messaging.
- Multi-threading.
- Integration with traditional applications.

- Database connectivity.
- Persistence.
- Resource pooling.

Different vendors have come with different version of application servers based on the J2EE architecture. Some application servers are:

- BEA WebLogic ®Application Server.
- IBM WebSphere ®Application Server.
- iPlanet ®Application Server.
- Oracle ®Application Server.
- Sybase ®Enterprise Application Server.
- Sun Java System Application Server formerly SUN One Application Server.
- JBoss ®Application Server.

The application servers that are used in this thesis for analysis are

- BEA WebLogic Application Server 8.1.
- Sun Java System Application Server 7 formerly SUN One Application Server 7.

- IBM WebSphere Application Server 5.0.
- Oracle9i Application Server.

3.2.1 BEA WebLogic Application Server 8.1

BEA WebLogic server is a fully featured application server, which provides functionality to deploy and handle enterprise applications. BEA WebLogic server adheres to the multi-tier J2EE architecture by separating the presentation, business logic & data connectivity.

In a Web service perspective a BEA WebLogic server provides [15]:

- High performance reliable SOAP implementation.
- Support for WSDL & UDDI.
- Automatic generation of Web services helper code.
- Support for Web services security.
- Graphic tools for development. & deployment of Web services.
- Support for XML parsing.

WebLogic server provides the flexibility to create Web services using enterprise application components or use existing Web services.

BEA WebLogic server provides necessary support to create thin client stubs while deploying Web services, SOAP encoding/decoding, generation of WSDL scripts , management and monitoring of Web services, graphical administration console, private UDDI registry. Security of Web services is provided by transmitting the SOAP messages using SSL(Secure Socket Layer). SOAP by nature is not reliable and it is made reliable by BEA WebLogic by providing support for asynchronous communication, receipts and notifications.[12]

BEA WebLogic enterprise platform is an integrated suite of application infrastructure built on open standards with support for high volume transactions, business process management, application integration and business collaboration.[12]

BEA WebLogic platform infrastructure includes:

BEA WebLogic Server: An application server based on J2EE architecture to handle enterprise applications.[12]

BEA WebLogic Workshop ™: Unified graphical development environment and runtime framework to develop, test and deploy enterprise Web service applications.[12]

BEA WebLogic Integration ™: A solution for delivering application server, application integration, business process management and B2B integration functionality.[12]

BEA WebLogic Portal™: Used for delivering integrated enterprise platform including portal framework with portal foundation services, personalization, interaction management and integration services.[12]

BEA Liquid Data for WebLogic™: A virtual data access and aggregation product for information visibility allowing a real-time unified view of distinct enterprise data.[12]

BEA Tuxedo ®: A platform for building rock-solid, easy-to-manage enterprise systems that enable businesses to rapidly launch new products and services. [12]

BEA WebLogic JRockit ™: A highly-optimized JVM (Java Virtual Machine) that delivers superior application performance, reliability, and manageability for mission-critical Java applications running on the Intel 32 and 64 bit architecture.[12]

3.2.2 IBM WebSphere Application Server 5.0

IBM WebSphere application server is a J2EE based application server that provides the foundation for business-on-demand infrastructure. WebSphere server provides integration of deployment model, programming model, administration point and integrated application development environment. It also provides a visual work flow support for Web service standards.

WebSphere provides :

- Full J2EE 1.3 compatibility.
- Configuration and administration based on Java Management Extensions (JMX)
- Support for JMS, Web services and a private UDDI registry.
- Security using Java 2 security, JAAS (Java Authentication and Authorization Service) and WebSphere's pluggable security architecture.

Web Service Standard Development Kit : The WSDK contains WebSphere 5.0 application server and a set of tools to deploy Web services. It also includes a private UDDI registry server, used to publish and discover Web services using the UDDI protocols. The UDDI server runs as an enterprise application (EAR) within the application server. WSDK is based on Axis but does not expose the Axis programming model, it uses the WebSphere programming model based on JAX-RPC and Web services for J2EE . [17]

WebSphere Studio : The WebSphere studio is a suite of development tools for enterprise e-business Java-based applications. It also provides functionality to test and deploy applications. The WebSphere studio combines the functionality of VisualAge for Java and WebSphere Studio (classic). The studio contains Eclipse Workbench as a standard. [17]

3.2.3 Sun Java System Application Server 7

The Sun Java System Application Server 7, is an enterprise application server developed based on multi-tier J2EE architecture. The components that are included in SUN Java System Application Server 7 are :

- Sun Java System Message Queue 3.5 formerly SUN ONE Message Queue 3.0.1.
- PointBase database server. & Type 4 JDBC drivers.
- Java 2 Software Development Kit.
- SUN ONE Studio 4, Enterprise Edition for Java Plugin Module.

The following specifications that are supported by the Sun Java System Application Server 7 have been taken directly from Introduction to Sun ONE Application Server 7 [22]:

- *Support for Java Web services by using JAXP, JAXM, JAX-RPC, JAXR, SAAJ.*
- *Runtime including runtime control, process and thread management.*
- *HTTP server based on SUN ONE Web Server. Highly scalable and high performance CMP runtime.*
- *JTS-based, Java Transaction Manager.*

- *Object Request Broker (ORB).*
- *SNMP Agent.*
- *HTTP Proxy Plugin.*
- *J2EE verifier utilities.*
- *Apache Ant and Sun Optional Tasks.*
- *Web-based administration interface with local and remote administration.*
- *Server JMX technology used for internal underlying administration infrastructure.*
- *Full-featured, Command-line interface local and remote operation.*
- *Command line with interactive and scripting mode.*
- *Multiple administrative domains capabilities for creating wholly separate application server configurations.*
- *Provides multiple instances in each domain.*
- *JAAS support.*
- *Multiple authentication realms including file, Solaris realms and LDAP(Lightweight Directory Access Protocol).*

- *SUN ONE Directory for authentication.*

3.2.4 Oracle 9i Application Server

Oracle9iAS is a J2EE based application server that provides a platform for running J2EE applications & Web services. Oracle9iAS provides support for

J2EE and Web Services : Oracle9iAS framework provides functionality to design, develop, and deploy dynamic Web sites, portals and transactional applications. With Oracle9iAS Web services can expose business functions to authorized parties over the Internet from any Web device. [24]

Portals : Oracle9iAS provides functionality to support enterprise portals. Oracle9iAS allows for self-service content management and publishing, wizard-based development, deploying, publishing, and consuming Web services. [24]

Wireless : Oracle9iAS wireless provides the ability to deliver content to any device by using any protocol on any wireless network. It can also provide location-based services. [24]

Caching : Oracle9iAS can cache both static and dynamically generated Web content thus improving the performance and scalability of heavily loaded websites. [24]

Business Intelligence : Oracle9iAS provides comprehensive personalization and business intelligence services through Web analytic applications. [24]

E-Business Integration : Oracle9iAS provides communications and integration capabilities for e-business applications. It provides seamless query and transaction access to many non-Oracle data sources, provide integrated e-mail, voice, and fax messaging for access through multiple client devices. [24]

Management and Security : Oracle9iAS provides flexibility to configure and monitor Oracle9iAS instances, optimize them for performance and scalability, respond to problem conditions from a centralized console, use Secure Sockets Layer (SSL) connections, user and client certificate-based authentication, single sign-on capability across applications, LDAP directory that provides a single repository and administration environment for user accounts . [24]

Oracle9iAS components that are needed to deploy J2EE and Web services are:

- Oracle9iAS Web services.
- Oracle9iAS Containers for J2EE.
- Oracle HTTP Server.
- Oracle PL/SQL.
- Oracle9iAS Forms services.
- Oracle XML Developer Kit.
- Oracle9i Client.

CHAPTER 4

Analysis Parameters

4.1 Parameters Used for Analysis

The analysis of the application servers with respect to Web services is done using a set of parameters. The detail explanation of each parameter is provided in the next section followed by the results explaining what application servers provide.

4.1.1 Application Server Supported Protocols

This parameter is used to give an overview of all the Web service related protocols supported by each application server that are used directly/indirectly to build and deploy an application server.

SOAP 1.1, 1.2 : Simple Object Access Protocol is a W3C standardized XML based light weight protocol used to exchange information in a distributed environment. SOAP 1.2 differs from SOAP 1.1 slightly by the syntax and semantics. SOAP 1.2 is still a W3C recommendation while SOAP 1.1 is a W3C standard. [1]

WSDL 1.1 : Web service Description Language is a W3C standardized XML based specification that describes a Web service, the functionality provided by that service, how to invoke the service, how to connect to service etc. [3]

UDDI 2.0 : Universal Description, Discovery and Integration is a specification that describes the process of registering with a public registry, invoking the service from registry etc. UDDI is a cross-industry effort driven by major software/e-business providers within the OASIS standards consortium. [2]

JAX-RPC 1.0 : The Java API for XML-based RPC (JAX-RPC) is a Sun Microsystems, Inc protocol used to develop SOAP based interoperable and portable Web services on the Java platform. The primary advantage of using JAX-RPC is that the developer of Web services is relieved from the complexity of the underlying runtime mechanisms. The JAX-RPC runtime takes care of the marshalling and un-marshalling , SOAP protocol level mechanisms etc.[4]

JAXM 1.1 : The Java API for XML Messaging is a Sun Microsystems, Inc protocol used to develop Web services that can send and receive document oriented XML messages on the Java platform. The JAXM implements the SOAP 1.1 with Attachments and includes the notion of messaging profiles. Profiles enables the support to use higher level messaging protocols beyond basic SOAP based messaging. [5]

SAAJ 1.1 : The SOAP with Attachments 1.1 is a Sun Microsystems, Inc used by Web services that consume and produce messages using SOAP 1.1 specification and an additional SOAP with Attachment note. SAAJ is used internally by

JAXM messages. [6]

JAXR 1.0 : The Java API for XML Registries is a Sun Microsystems, Inc protocol used to provide an uniform and a standard API for accessing different XML registries. JAXR uses unified JAXR information model, which describes content and metadata with in XML registries. JAXR provides rich metadata capabilities for classification, association and rich query capabilities. JAXR supports both the ebXML registry and the UDDI registry v2.0 specifications. [7]

4.1.2 Publish Web service from Java Class

This parameter analyzes whether the application server has got the capability to convert a Java class or a Java interface to a Web service and deploy the Web service.

4.1.3 Publish Web service from Enterprise JavaBean

This parameter analyzes whether the application server has got the capability to convert an EJB (Enterprise JavaBean) to a Web service and deploy the Web service in the bean container of the application server. The published Web service should also have a capability to invoke an entity JavaBean to cater the user requests.

4.1.4 Publish Web service from Message-Driven Bean (JMS)

Message-Driven JavaBeans are used to develop enterprise applications that support asynchronous communication between client and server. There is no direct point to point contact between the provider/requester, instead all the communication is diverted through a messaging provider. This parameter will decide whether the message-driven bean can be published as a Web service, so that the concept of asynchronous communication still works.

4.1.5 Support for built in & non built in data types

Built in data types are those that are supported by the XML Schema. Some of them include int, string, float etc. Every Java data type that is used inside a Web service should be either mapped as a built in or non built in data type. If a non built in data type is used as a parameter or a return value, then the Web service developer should provide a serialization class that converts the data between the XML and java data types.

4.1.6 RPC Oriented Web Services

Web services can be implemented as both RPC-oriented and document-oriented services. This parameter is used to analyze the application server capability in handling

RPC oriented Web services. The SOAP messages that use RPC-oriented style contain input parameters and output values. The input parameters are transmitted to the Web service via a SOAP message, Web service maps the set of input parameters received to the native programming method parameters, processes the request and returns the values to the service requesters by constructing a response SOAP message. RPC oriented services has got the flexibility to handle any number of parameters as long as they match the parameter types in the calling procedure.

By default most of the currently existing application servers use RPC oriented services.

4.1.7 Document-Oriented Web Service

Web services that contain an XML document inside the SOAP message come under this category. Document oriented Web services create an XML document with all the data that needs to be transmitted and embeds this XML document inside the SOAP message.

The advantages of using document oriented services are high flexibility, dynamic, optimized & interpreted.

Application servers can handle either one of RPC-oriented style or Document-oriented style in a single SOAP message but not both simultaneously.

4.1.8 JAX-RPC protocol Support

Application servers that provide Web services functionality should support either one of JAX-RPC or JAXM. The Java API for XML-based RPC (JAX-RPC) is a Sun Microsystems, Inc protocol used to develop SOAP Web services on the Java platform. The JAX-RPC runtime takes care of the marshalling and un-marshalling, SOAP protocol level mechanisms etc.

The JAX-RPC requires javax.xml.soap , which is a low level SOAP package provided by Sun Microsystems, Inc to handle Web services. This SOAP package provides SOAP message access & SOAP message communication. JAX-RPC relies on SAAJ. The whole concept of JAX-RPC is based primarily on three interfaces: service, stub and call. The call interface is a dynamic invocation interface (DII) that supports synchronous & asynchronous messaging, service & stub support synchronous request/response invocations by using the RMI architecture.

4.1.9 JAXM protocol Support

The Java API for XML Messaging is a Sun Microsystems, Inc protocol used to develop Web services that can send and receive document oriented XML messages on Java platform. The JAXM protocol requires javax.xml.soap a low level SOAP package provided by Sun Microsystems, Inc to provide SOAP message access and communication and javax.xml.messaging a higher level messaging package that supports SOAP

communication.

The advantage of using the messaging package is it auto generates SOAP envelope using the low level SOAP package based on predefined profiles. Profiles enables support to use higher level messaging protocols beyond basic SOAP based messaging. JAXM is primarily used when there is a need to handle high level messaging profiles like ebXML and asynchronous communication. Currently asynchronous communication can also be handled using JMS. JAXM uses the Simple Object Access Protocol (SOAP) 1.1 with Attachments (SAAJ) and includes the notion of messaging profiles.

4.1.10 Support for ebXML

Electronic Business Extensible markup language (ebXML) is a specification that allows business organizations to exchange electronic business data by using XML specifications & Web service capability.

ebXML provides a framework that can be used by business organizations or individual components. ebXML provides support for SOAP based ebXML messaging service. ebXML also provides support for security, reliable messaging, registry service, mechanism to define high level profiles and a mechanism to define electronic trading agreements. [8]

SUN ONE application server provides the capability to handle ebXML profiles. The JAXM messaging package provided by SUN ONE server has got the capability to

generate a SOAP envelope based on predefined ebXML profile. SUN One application server has also got the capability to provide security and reliable messaging for the ebXML profiles.

4.1.11 Tool to Convert Java class/EJB to Web service

The process of converting an enterprise bean or a Java interface to a Web service is often tedious and cumbersome. So every application server has come up with a set of tools to convert the source program to Web service interfaces and publish them as Web services. Some important things that needs to be done in this phase are creating a Web service deployment descriptor, checking for dependencies and other runtime issues , generating client JAR files, creating EAR files by packing the necessary archive files and deployment descriptors. All application servers used in this thesis use a variant of Ant tool. Ant is a Java based build tool that uses XML based configuration files to execute tasks written in Java. The tools provided by application servers are BEA WebLogic server: Servicegen, Source2Wsdd; IBM Web Sphere: Bean2WebService, EJB2WebService; SUN Java System Server: Wscompile, Wsdeploy; Oracle 9iAS: WebServiceAssembler.

4.1.12 Tool to Convert Web service to Java Interface

Client applications that need to invoke a Web service should have a capability to convert the WSDL document to a Java interface/class which can be inherited by other classes. To accomplish this there should be a way to convert the Web service deployment descriptor to an interface that provide all the methods and parameters that are declared in the WSDL document. Application servers provide a tool to support this functionality and the tools are: BEA WebLogic server: Wsdl2service; IBM Web Sphere: WSDL2Java; SUN One Server: Wscompile; Oracle 9iAS: wsdl2ejb.

4.1.13 Support for SOAP with Attachments

This parameter is used to analyze whether the Web service deployed in a particular application server has got the capability to transmit non XML based documents as SOAP messages when a client requests. Non XML based documents include images, zip files etc.

4.1.14 Components that provide support for Messaging and Application Integration

Web services can be developed and accessed in many ways, but providing support for messaging and enterprise application integration is a key issue. This parameter is used to analyze the various queuing and application integration components provided by application servers. Queuing is a way by which various application communicate

asynchronously.

WebSphere MQ : IBM WebSphere MQ is an integrated middleware providing the intelligence and infrastructure to integrate business applications. It provides support to provide messaging functionality between clients and servers. WebSphere application servers JMS functionality is based on the WebSphere MQ architecture. WebSphere MQ supports application integration by allowing applications to exchange data as messages synchronously/asynchronously across different platforms. Some of the additional features provided by WebSphere MQ are automatic dealing with network protocols, dynamic distribution of workload, fault tolerant recovery, application portability and scalability.[19] [20]

Sun Java System Message Queue formerly Sun ONE Messages Queue : Sun Java System Message Queue 3.5 is a middleware product that provides support for reliable messaging between applications. Sun Java System Message Queue is an integral part of Sun Java System application server which allows components to deliver JMS messages between applications. The components of the Message Queue are the client API that implements the JMS API; the broker (Message Server) and an administration component. The primary functionality of the broker is to route the messages between the applications.It is extremely scalable with high-performance. [22]

WebLogic JMS Server : WebLogic JMS Server is a message oriented middleware that provide reliable messaging service between applications that run on BEA WebLogic server. The components of WebLogic JMS Server are : WebLogic JMS Server that provide messaging functionality; client API, JNDI(Java Naming and Directory Interface) and backing stores for persistence of JMS Queues/Topics. Some of the features provided by WebLogic JMS server are support for unified messaging, clustering support, messaging between applications that run on different hardwares/operating systems, persistence of message stores and multicast capability allowing messages to be delivered using IP Multicasting. [16]

Oracle Message Broker : Oracle Message Broker is a middleware that provides support for enterprise application integration & messaging. It provides support for asynchronous system-independent message based communication mechanism. The components of the Oracle Message Broker are Message Broker Core : JMS provider that supports publishing messages, polling of message servers etc. [26] Drivers and Message Servers : Drivers provide access to message servers, message servers stores and manage messages. Drivers are also responsible for message translation to native storage formats. Administrative Components : Support for LDAP directory configuration & Lightweight configuration.[26]

4.1.15 Global Registry

Global registry is a database repository that facilitates the participation of companies that provide Web services in e-commerce and business-to-business places. Global registry makes service provider companies more visible and accessible to the external world. Global registry is an UDDI registry where Web service providers register their Web services with information like the name of the service, description, interfaces etc. so that global access to the Web service can be provided. The consumer of the Web service looks up the registry to find out the services it needs and access them. Each Web service that is published with the registry server is assigned an unique key so that services with same name and different providers can be present in the registry server. Organizations that wish to publish their services to UDDI registry have to register with global registry service provider. Operations like publishing, deletion of Web service from global registry need the verification of security credentials, while searching the registry can be done with out any security credentials.

4.1.16 Local/Private Registry

The local registry provides the same functionality provided by the global registry except that it is used for internal applications of business organization, development and testing etc. The local registry can also be exposed to the external world but it will not have the performance and scalability of a global registry, and it might

also not have a global access point. Local/private registries are generally used by various departments inside an organization which communicate with each other via Web services. Application of security credentials is optional and is based on the requirements of the organization.

4.1.17 Support to Query Local/ Global Registry

This parameter is used to analyze whether global & local registries can be accessed programmatically. The concept of Web services stand on the idea that human intervention should be minimized. This can be achieved only if the registries can be accessed programmatically for publishing, retrieving or searching by organizations that need services.

4.1.18 GUI to perform operations on private registry

All global registries by default provide a Graphical User Interface (GUI) to access them and this might not be the case with local/private registries. Even though most of the Web service operations are handled automatically by computers with out human intervention, this is not true always as there are lot of administrative and maintenance tasks which needs man power to do them. This parameter analyzes whether there is a GUI for the private registries that aid in performing registry operations and the flexibility provided to do the task.

4.1.19 Application Server Configuration Difficulty

Application server configuration is a tedious task, as the overall performance of a Web service application depends on how the application server is configured. Some major issues that need to be taken care are database configuration, network configuration, security issues etc. Clustering/Load balancing should also be considered if the traffic on the system is too high. Application server configuration is not something that's needs to be concerned only for production systems but it is a major issue at every point in development life cycle. This is a one time task that needs to be done while the application server is installed. Changes has to be made to the application server when new services, databases etc are added.

4.1.20 Level of Web Service Deployment Difficulty

Developing a Web service is an important issue, but deploying Web service is much tedious and crucial task as maximum productivity can be achieved only by deploying it accurately. Some of the major steps while deploying a Web service are: compilation of the Web service code, proper mapping of the interfaces with the business logic, validation of the deployment descriptors, marshalling and un-marshalling routines that aid in the SOAP message creation and communication, creation of a WSDL document if not already present or validation of the WSDL document against the Web service interfaces to check the application integrity, creating Web components if

needed by the application, creating enterprise archive files with the necessary classes, stub ,skeleton code and XML deployment descriptors, migrating the archive file to the respective physical deployment container under the application server, registering the service with JNDI by resolving any naming issues and finally cleaning up temporarily created unused files.

This parameter analysis how much complex the whole process is under each application server and the difficulty associated with the use of automated tools.

4.2 Final Results Table

Table 4.1: Final Results Table

Analysis Parameter	BEA WebLogic Server 8.1	IBM WebSphere Server 5.0	Sun Java System Application Server 7	Oracle9i Application Server
Application server supported protocols	SOAP 1.1 & 1.2, SAAJ, WSDL 1.1, UDDI 2.0, JAX-RPC 1.0, JAXR	SOAP 1.1, WSDL 1.1, UDDI 2.0, JAX-RPC 1.0, JAXR	SOAP 1.1, WSDL 1.1, UDDI 2.0, JAX-RPC 1.0, JAXM, JAXR, SAAJ	SOAP 1.1, WSDL 1.1, UDDI 2.0, JAX-RPC 1.0, JAXR, SAAJ
Publish Web service from Java class	Yes	Yes	Yes	Yes
Publish Web service from Enterprise JavaBean (EJB)	Yes	Yes	Yes	Yes
Publish Web service from Message-Driven Bean (JMS)	Yes	No	Yes (Partially supported)	Yes
Support for built in & non built in data types	Yes	Yes	Yes	No
RPC-Oriented Web services	Yes	Yes	Yes	Yes

Document-Oriented Web services	Yes	Yes	Yes	Yes
JAX-RPC protocol support	Yes	Yes	Yes	Yes
JAXM protocol support	No	No	Yes	No
Support for ebXML	No	No	Yes	No
Tool to convert Java class/EJB to Web service	Servicegen, source2wsdd	Bean2Web Service, EJB2Web Service	Wscompile, Wsdeploy	WebService Assembler
Tool to convert Web service to Java interface	Wsdl2service	WSDL2Java	Wscompile	wsdl2ejb
Support for SOAP with Attachments	Yes	No	Yes	No

Components that provide support for Messaging and Application Integration	WebLogic JMS Server	IBM MQ Series	Sun Java System Message Queue formerly SUN ONE MQ Series	Oracle Message Broker
Global Registry	No	Yes	No	Yes
Local/Private Registry	Yes	Yes	Yes	Yes
Support to query local/global registry	Yes	Yes	Yes	Yes
GUI to perform operations on private registry	Yes	Yes	Yes	Yes

Application Server configuration difficulty	Easy	Medium	Medium	High
Level of Web service deployment difficulty	Easy	Easy	Easy	High

CHAPTER 5

Conclusion And Future Work

This thesis has parameterized a set of parameters that analyze the functionality of Web services. The application servers have been surveyed based on these parameters and the results are documented. The results show that most of the application servers provide the most basic features needed to deploy Web services. Application servers also provide some additional functionality that can enhance the way Web services are deployed.

Based on the type of functionality, it can be concluded that BEA WebLogic server can be used to develop traditional & sophisticated Web services. If there is a strong need for application integration and messaging IBM WebSphere & Sun Java System application servers can be considered. Sun Java System application server is also advisable to use when there is a need to automate legacy system interaction, which can be handled by ebXML. Oracle application server can be used when the Web service has to do a lot of database interactions , wireless & portal integration.

This thesis can be extended in future to analyze the security issues concerned with Web service. Security for Web services is needed to ensure that the confidentiality & integrity of the SOAP message is not compromised while in transit. This thesis can be extended to make an enhanced study of how security can be applied to SOAP

messages in synchronous and asynchronous mode. Security issues associated with the global registry authentication also can be analyzed.

This thesis can also be further extended to make an enhanced study of various enterprise application integration components with regards to their functionality, performance & security. Web service interoperability issues between different software platforms and application servers also needs to be studied.

This thesis can also be extended to study the workflow and programming models used by application servers. Finally this thesis can be extended to further analyze various high level messaging profiles like ebXML its message structure, data formats and ebXML registry structures.

APPENDIX A

WSDL Document

```
<?xml version="1.0" encoding="UTF-8"?>
 <wsdl:definitions
       targetNamespace="http://stockSession.stock.research"
       xmlns="http://schemas.xmlsoap.org/wsdl/"
       xmlns:apachesoap="http://xml.apache.org/xml-soap"
       xmlns:impl="http://stockSession.stock.research"
       xmlns:intf="http://stockSession.stock.research"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
       xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types/>
<wsdl:message name="sellResponse">
 <wsdl:part name="sellReturn" type="xsd:string"/>
</wsdl:message>
```

```
<wsdl:message name="buyRequest">
 <wsdl:part name="in0" type="xsd:string"/>
 <wsdl:part name="in1" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="showListRequest">
</wsdl:message>

<wsdl:message name="sellRequest">
 <wsdl:part name="in0" type="xsd:string"/>
 <wsdl:part name="in1" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="buyResponse">
 <wsdl:part name="buyReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getPriceResponse">
 <wsdl:part name="getPriceReturn" type="xsd:float"/>
</wsdl:message>
```

```
<wsdl:message name="showListResponse">
 <wsdl:part name="showListReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getPriceRequest">
 <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>

<wsdl:portType name="StockInterface">
  <wsdl:operation name="showList">
    <wsdl:input message="intf:showListRequest" name="showListRequest"/>
    <wsdl:output message="intf:showListResponse" name="showListResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getPrice" parameterOrder="in0">
    <wsdl:input message="intf:getPriceRequest" name="getPriceRequest"/>
    <wsdl:output message="intf:getPriceResponse" name="getPriceResponse"/>
  </wsdl:operation>
  <wsdl:operation name="sell" parameterOrder="in0 in1">
    <wsdl:input message="intf:sellRequest" name="sellRequest"/>
```

```
    <wsdl:output message="intf:sellResponse" name="sellResponse"/>
  </wsdl:operation>
  <wsdl:operation name="buy" parameterOrder="in0 in1">
    <wsdl:input message="intf:buyRequest" name="buyRequest"/>
    <wsdl:output message="intf:buyResponse" name="buyResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="StockSoapBinding" type="intf:StockInterface">
 <wsdlsoap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http"/>

 <wsdl:operation name="showList">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="showListRequest">
       <wsdlsoap:body namespace="http://stockSession.stock.research"
                      use="literal"/>
   </wsdl:input>
   <wsdl:output name="showListResponse">
       <wsdlsoap:body namespace="http://stockSession.stock.research"
```

```
                    use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getPrice">
  <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getPriceRequest">
      <wsdlsoap:body namespace="http://stockSession.stock.research"
                    use="literal"/>
    </wsdl:input>
   <wsdl:output name="getPriceResponse">
      <wsdlsoap:body namespace="http://stockSession.stock.research"
                    use="literal"/>
   </wsdl:output>
</wsdl:operation>

<wsdl:operation name="sell">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="sellRequest">
      <wsdlsoap:body namespace="http://stockSession.stock.research"
```

```
                    use="literal"/>
    </wsdl:input>
    <wsdl:output name="sellResponse">
      <wsdlsoap:body namespace="http://stockSession.stock.research"
                     use="literal"/>
    </wsdl:output>
 </wsdl:operation>

 <wsdl:operation name="buy">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="buyRequest">
     <wsdlsoap:body namespace="http://stockSession.stock.research"
                    use="literal"/>
   </wsdl:input>
   <wsdl:output name="buyResponse">
     <wsdlsoap:body namespace="http://stockSession.stock.research"
                    use="literal"/>
   </wsdl:output>
 </wsdl:operation>
</wsdl:binding>
```

```
<wsdl:service name="StockInterfaceService">
  <wsdl:port binding="intf:StockSoapBinding" name="Stock">
  <wsdlsoap:address location=
                    "http://localhost:6080/StockWebService/services/Stock"/>
  </wsdl:port>
</wsdl:service>


</wsdl:definitions>
```

BIBLIOGRAPHY

[1] Simple Object Access Protocol (SOAP) 1.1 W3C Note, http://www.w3.org/TR/SOAP/

[2] Universal Description, Discovery, and Integration (UDDI) project: http://www.uddi.org/

[3] Web Services Description Language (WSDL) 1.1 W3C Note, http://www.w3.org/TR/wsdl

[4] Java API for XML-Based RPC (JAX-RPC) Specification 1.0, http://java.sun.com/xml/downloads/jaxrpc.html

[5] Java API for XML Messaging Specification 1.1, http://java.sun.com/xml/jaxm/

[6] SOAP with Attachments API for Java (SAAJ) v1.1, http://java.sun.com/xml/saaj/

[7] Java API for XML Registries Specification 1.0, http://java.sun.com/xml/jaxr/index.html

[8] ebXML, http://www.ebxml.org/

[9] World Wide Web Consortium (W3C), www.w3.org

[10] Web Services Conceptual Architecture (WSCA 1.0) May 2001 By Heather Kreger IBM Software Group

[11] *The Complete Reference J2EE* by James Edward Keogh, Jim Keogh

[12] BEA White Paper - BEA WebLogic Server 8.1 Overview

[13] BEA WebLogic Platform 8.1 Online Documentation, http://edocs.bea.com/wls/docs81/index.html

[14] BEA WebLogic Workshop : building next generation Web services visually / Sean Christofferson ... [et al.].

[15] BEA Weblogic Server, Programming Weblogic WebServices version 8.1 revised:December 2003.

[16] Introduction to WebLogic JMS, BEA Weblogic Online Documentation

[17] IBM Redbooks: WebSphere Version 5 Web Services Handbook by Ueli Wahli, Matija Drobnic, Christian Gerber, Gustavo Garcia Ochoa, Michael Schramm

[18] *Professional IBM WebSphere 5.0 Application Server* by Tim Francis, Eric Herness, Rob High, Jim Knutson, Kim Rochat, Chris Vignola

[19] IBM Redbooks: WebSphere Application Server V5 and WebSphere MQ Family Integration by Jill Lennon, Ashok Ambati, Bill Moore, John W. Mount, Fred Plassman, Mark Smith, Peter von Hirschfeld

[20] IBM Redbooks: IBM MQSeries Primer by Dieter Wackerow

[21] Sun Product Documentation: Sun ONE Application Server 7 Product Introduction

[22] Sun Product Documentation: Introduction to Sun ONE Application Server 7

[23] The Java ™Web Services Tutorial by Eric Armstrong, Stephanie Bodoff, Debbie Carson, Maydene Fisher, Scott Fordin, Dale Green, Kim Haase, Eric Jendrock

[24] Oracle9i Application Server, An Oracle White Paper September 2002

[25] Oracle Documentation: Oracle9i Application Server Web Services Developers Guide Release 2 (9.0.3) August 2002

[26] Oracle Documentation: Oracle Message Broker Administration Guide Release 2.0.1.0

[27] Developing Java Web Services by Ramesh Nagappan, Robert Skoczylas John Wiley & Sons Copyright Wiley Publishing, Inc.