Learning Material

UNIT – I

Objective:

To differentiate various distributed computing technologies.

Syllabus:

Evolution and Emergence of web services:

Evolution of distributed computing, Core distributed computing technologies-client/ server, CORBA, JAVA RMI, MicroSoft DCOM, MOM, Challenges in Distributed Computing, role of J2EE and XML in distributed computing, emergence of web services and Service Oriented Architecture(SOA).

Learning Outcomes:

Students will be able to

- Understand how to differentiate various distributed computing technologies.
- Identify challenges in distributed Computing.
- Learn the role of J2EE and XML in distributed computing.
- Understand the emergence of Web Services and SOA

LEARNING MATERIAL

Evolution of Distributed Computing:-

In the early years of computing, mainframe-based applications were considered to be the best-fit solution for executing large-scale data processing applications. With the advent of personal computers (PCs), the concept of software programs running on standalone machines became much more popular in terms of the cost of ownership and the ease of application use. With the number of PC-based application programs running on independent machines growing, the communications between such application programs became extremely complex and added a growing challenge in the aspect of application-to-application interaction.

Lately, network computing gained importance, and enabling remote procedure calls (RPCs) over a network protocol called Transmission Control Protocol/Internet Protocol (TCP/IP) turned out to be a widely accepted way for application software communication. Since then, software applications running on a variety of hardware platforms, operating systems, and different networks faced some challenges when required to communicate with each other and share data. This demanding requirement lead to the concept of distributed computing applications. As a definition, "Distributing Computing is a type of computing in which different components and objects comprising an application can be located on different computers connected to a network". Distributed computing model that provides an infrastructure enabling invocations of object functions located anywhere on the network. The objects are transparent to the application and provide processing power as if they were local to the application calling them.

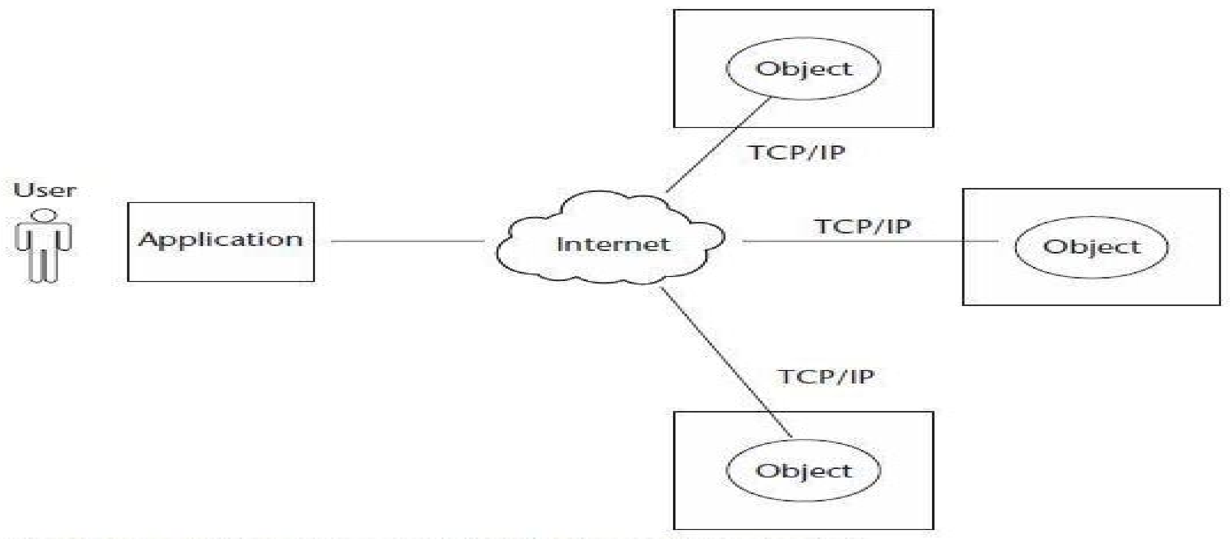


Figure 1.1 Internet-based distributed computing model.

Importance of Distributed Computing:

The distributed computing environment provides many advantages
The following are some of those key advantages:

Higher performance: Applications can execute in parallel and distribute the load across multiple servers.

Collaboration: Multiple applications can be connected through standard distributed computing mechanisms.

Higher reliability and availability: Applications or servers can be clustered in multiple machines.

Scalability: This can be achieved by deploying these reusable distributed components on powerful servers.

Extensibility: This can be achieved through dynamic (re)configuration of applications that are distributed across the network. Higher productivity and lower development cycle time. By breaking up large problems into smaller ones, these individual components can be developed by smaller development teams in isolation.

Reuse: The distributed components may perform various services that can potentially be used by multiple client applications. It saves repetitive development effort and improves interoperability between components.

Reduced cost: Because this model provides a lot of reuse of once developed components that are accessible over the network, significant cost reductions can be achieved. Distributed computing also has changed the way traditional network programming is done by providing a shareable object like semantics across networks using programming languages like Java, C, and C++. The following sections briefly discuss core distributed computing technologies such as Client/Server applications, OMG CORBA, Java RMI, Microsoft COM/DCOM, and MOM.

Client-Server Applications:

The early years of distributed application architecture were dominated by two-tier business applications. In a two-tier architecture model, the first (upper) tier handles the presentation and business logic of the user application (client), and the second/lower tier handles the application organization and its data storage (server). This approach is commonly called client-server applications architecture. Generally, the server in a client/server application model is a database server that is mainly responsible for the organization and retrieval of data. The application client in this model handles most of the business processing and provides the graphical user interface of the application. It is a very popular design in business applications where the user.

Interface and business logic are tightly coupled with a database server for handling data retrieval and processing. For example, the client-server model has been widely used in enterprise resource planning (ERP), billing, and Inventory application systems where a number of client business applications residing in multiple desktop systems interact with a central database server. Figure 1.2 shows an architectural model of a typical client server system in which multiple desktop-based business client applications access a central database server. Some of the common limitations of the client-server application model are as follows:

■Complex business processing at the client side demands robust client systems.

■Security is more difficult to implement because the algorithms and logic reside on the client side making it more vulnerable to hacking.

■Increased network bandwidth is needed to accommodate many calls to the server, which can impose scalability restrictions.

■Maintenance and upgrades of client applications are extremely difficult because each client has to be maintained separately.

■Client-server architecture suits mostly database oriented standalone applications and does not target robust reusable component oriented applications.

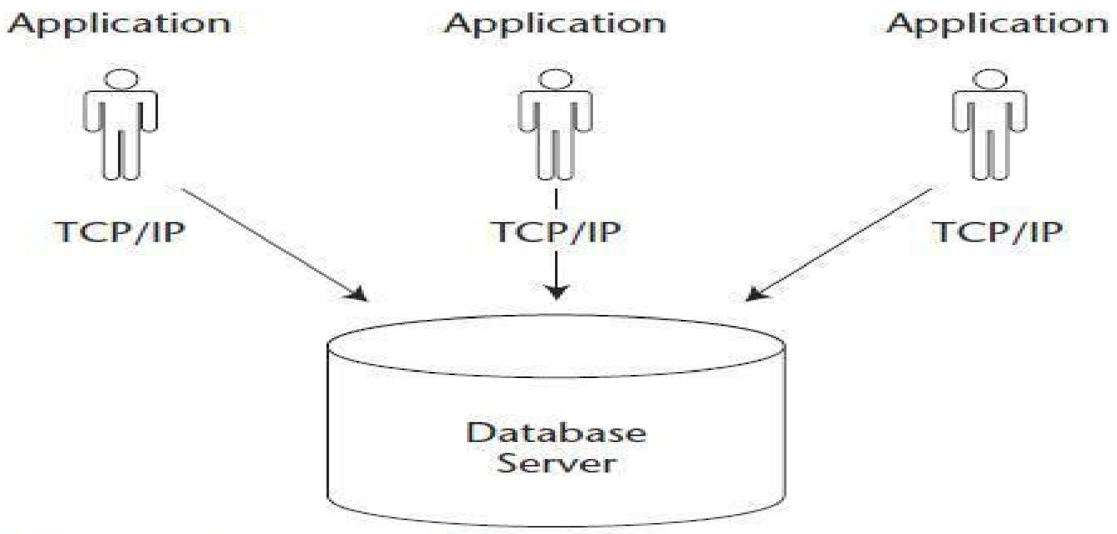


Figure 1.2 An example of a client-server application.

CORBA:

The Common Object Request Broker Architecture (CORBA) is an industry wide, open standard initiative, developed by the Object Management Group (OMG) for enabling distributed computing that supports a wide range of application environments. OMG is a non profit consortium responsible for the production and maintenance of framework specifications for distributed and interoperable object-oriented systems.

CORBA differs from the traditional client/server model because it provides an object-oriented solution that does not enforce any proprietary protocols or any particular programming language, operating system, or hardware platform. By adopting CORBA, the applications can reside and run on any hardware platform located anywhere on the network, and can be written in any language that has mappings to a eutral interface definition called the Interface Definition Language (IDL).

An IDL is a specific interface language designed to expose the services (methods/functions) of a CORBA remote object. CORBA also defines a collection of system-level services for handling low-level application services like life-cycle, persistence, transaction, naming, security, and so forth. Initially, CORBA 1.1 was focused on creating component level, portable object applications without interoperability. The introduction of CORBA 2.0 added interoperability between different ORB vendors by implementing an Internet Inter-ORB Protocol (IIOP).

The IIOP defines the ORB backbone, through which other ORBs can bridge and provide interoperation with its associated services. In a CORBA-based solution, the Object Request Broker (ORB) is an object bus that provides a transparent mechanism for sending requests and receiving responses to and from objects, regardless of the environment and its location. The ORB intercepts the client’s call and is responsible for finding its server object that implements the request, passes its parameters, invokes its method, and returns its results to the client. The ORB, as part of its implementation, provides interfaces to the CORBA services, which allows it to build custom-distributed application environments.

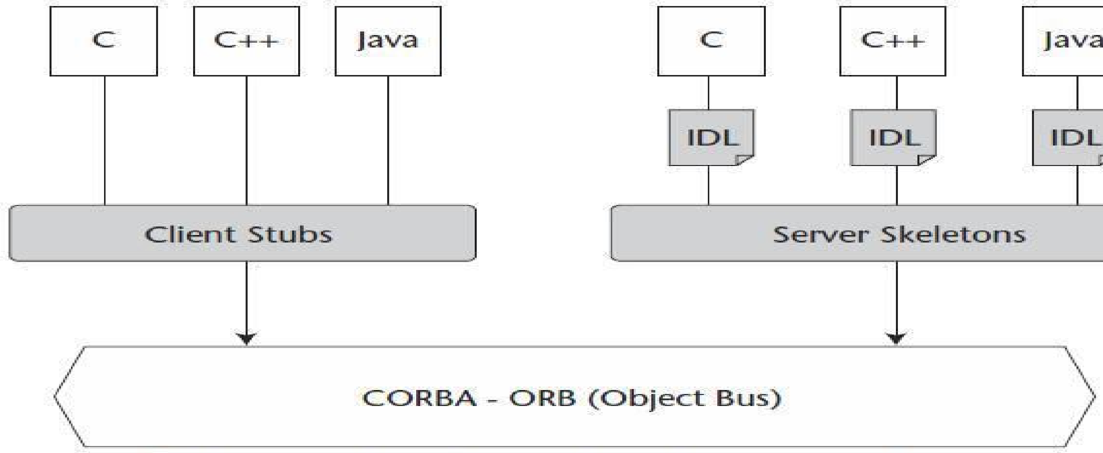


Figure 1.3 An example of the CORBA architectural model.

Figure 1.3 illustrates the architectural model of CORBA with an example representation of applications written in C, C++, and Java providing IDL bindings.

The CORBA architecture is composed of the following components:

IDL: CORBA uses IDL contracts to specify the application boundaries and to establish interfaces with its clients. The IDL provides a mechanism by which the distributed application component's interfaces, inherited classes, events, attributes, and exceptions can be specified.

ORB: It acts as the object bus or the bridge, providing the communication infrastructure to send and receive request/responses from the client and server. It establishes the foundation for the distributed application objects, achieving interoperability in a heterogeneous environment.

Some of the distinct advantages of CORBA over a traditional client/server application model are as follows:

OS and programming-language independence: Interfaces between clients and servers are defined in OMG IDL, thus providing the following advantages to Internet programming: Multi-language and multi-platform application environments, which provide a logical separation between interfaces and implementation.

Legacy and custom application integration: Using CORBA IDL, developers can encapsulate existing and custom applications as callable client applications and use them as objects on the ORB.

Rich distributed object infrastructure: CORBA offers developers a rich set of distributed object services, such as the Lifecycle, Events, Naming, Transactions, and Security services.

Location transparency: CORBA provides location transparency: An object reference is independent of the physical location and application level location. This allows developers to create CORBA-based systems where objects can be moved without modifying the underlying applications.

Java RMI:

Java RMI was developed by Sun Microsystems as the standard mechanism to enable distributed Java objects-based application development using the Java environment. RMI provides a distributed Java application environment by calling remote Java objects and passing them as arguments or return values. It uses Java object serialization—a lightweight object persistence technique that allows the conversion of objects into streams. Before RMI, the only way to do inter-process communications in the Java platform was to use the standard Java network libraries. Though the java.net APIs provided sophisticated support for network functionalities, they were not intended to support or solve the distributed computing challenges.

Java RMI uses Java Remote Method Protocol (JRMP) as the interprocess communication protocol, enabling Java objects living in different Java Virtual Machines (VMs) to transparently invoke one another's methods. Because these VMs can be running on different computers anywhere on the network, RMI enables object-oriented distributed computing. RMI also uses a reference-counting garbage collection mechanism that keeps track of external live object references to remote objects (live connections) using the virtual machine. When an object is found unreferenced, it is considered to be a weak reference and it will be garbage collected.

In RMI-based application architectures, a registry (rmiregistry)-oriented mechanism provides a simple non-persistent naming lookup service that is used to store the remote object references and to enable lookups from client applications. The RMI infrastructure based on the JRMP acts as the medium between the RMI clients and remote objects. It intercepts client requests, passes invocation arguments, delegate's invocation requests to the RMI skeleton, and finally passes the return values of the method execution to the client stub. It also enables call backs from server objects to client applications so that the asynchronous notifications can be achieved.

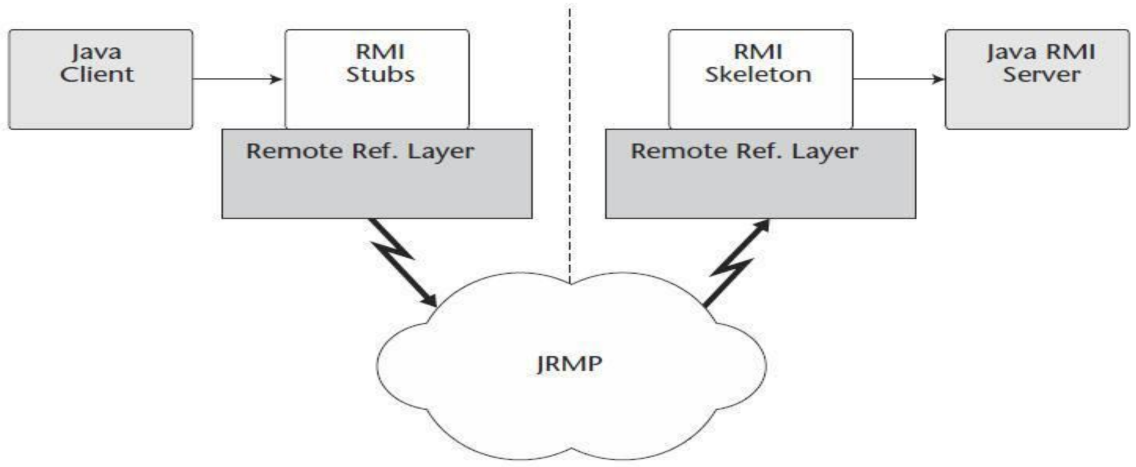


Figure 1.4 A Java RMI architectural model.

Figure 1.4 depicts the architectural model of a Java RMI-based application solution. The Java RMI architecture is composed of the following components:

RMI client: The RMI client, which can be a Java applet or a standalone application, performs the remote method invocations on a server object. It can pass arguments that are primitive data types or serializable objects.

RMI stub: The RMI stub is the client proxy generated by the rmi compiler (*rmic* provided along with Java developer kit—JDK) that encapsulates the network information of the server and performs the delegation of the method invocation to the server. The stub also marshals the method arguments and un marshals the return values from the method execution.

RMI infrastructure: The RMI infrastructure consists of two layers: the remote reference layer and the transport layer. The remote reference layer separates out the specific remote reference behavior from the client stub. It handles certain reference semantics like connection retries, which are unicast/multicast of the invocation requests. The transport layer actually provides the networking infrastructure, which facilitates the actual data transfer during method invocations, the passing of formal arguments, and the return of back execution results.

RMI skeleton: The RMI skeleton, which also is generated using the RMI compiler (rmic) receives the invocation requests from the stub and processes the arguments (un marshalling) and delegates them to the RMI server. Upon successful method execution, it marshals the return values and then passes them back to the RMI stub via the RMI infrastructure.

RMI server: The server is the Java remote object that implements the exposed interfaces and executes the client requests. It receives incoming remote method invocations from the respective skeleton, which passes the parameters after unmarshalling. Upon successful method execution, return values are sent back to the skeleton, which passes them back to the client via the RMI infrastructure. RMI-IIOP doesn't support dynamic downloading of the classes as it is done with CORBA in DII (Dynamic Interface Invocation).Actually RMI-IIOP combines the usability of Java Remote Method Invocation (RMI) with the interoperability of the Internet Inter-ORB Protocol (IIOP).So in order to attain this interoperability between RMI and CORBA, some of the features that are supported by RMI but not CORBA and vice versa are eliminated from the RMI-IIOP specification.

Microsoft DCOM:

The Microsoft Component Object Model (COM) provides a way for Windows-based software components to communicate with each other by defining a binary and network standard in a Windows operating environment. COM evolved from OLE (Object Linking and Embedding), which employed a Windows registry-based object organization mechanism. COM provides a distributed application model for ActiveX components. As a next step, Microsoft developed the Distributed Common Object Model (DCOM) as its answer to the distributed computing problem in the Microsoft Windows platform. DCOM enables COM applications to communicate with each other using an RPC mechanism, which employs a DCOM protocol on the wire.

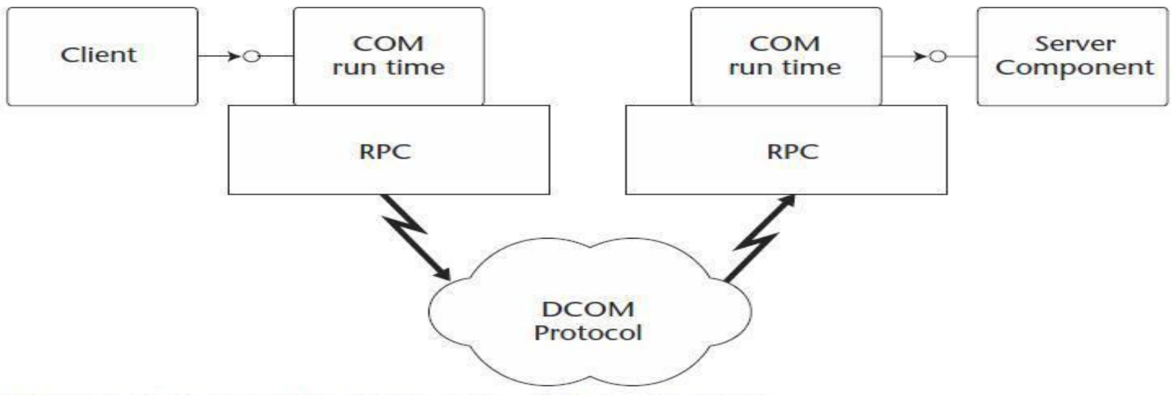


Figure 1.5 Basic architectural model of Microsoft DCOM.

Figure 1.5 shows an architectural model of DCOM. DCOM applies a skeleton and stub approach whereby a defined interface that exposes the methods of a COM object can be invoked remotely over a network. The client application will invoke methods on such a remote COM object in the same fashion that it would with a local COM object. The stub encapsulates the network location information of the COM server object and acts as a proxy on the client side.

The servers can potentially host multiple COM objects, and when they register themselves against a registry, they become available for all the clients, who then discover them using a lookup mechanism.

DCOM is quite successful in providing distributed computing support on the Windows platform. But, it is limited to Microsoft application environments.
The following are some of the common limitations of DCOM:

- Platform lock-in
- State management
- Scalability
- Complex session management issues

Message-Oriented Middleware:

Although CORBA, RMI, and DCOM differ in their basic architecture and approach, they adopted a tightly coupled mechanism of a synchronous communication model (request/response). All these technologies are based upon binary communication protocols and adopt tight integration across their logical tiers, which is susceptible to scalability issues. Message-Oriented Middleware (MOM) is based upon a loosely coupled asynchronous communication model where the application client does no need to know its application recipients or its method arguments.

MOM enables applications to communicate indirectly using a messaging provider queue. The application client sends messages to the message queue (a message holding area), and the receiving application picks up the message from the queue. In this operation model, the application sending messages to another application continues to operate without waiting for the response from that application.

JMS provides Point-to-Point and Publish/Subscribe messaging models with the following features:

■ Complete transactional capabilities

■ Reliable message delivery

■ Security

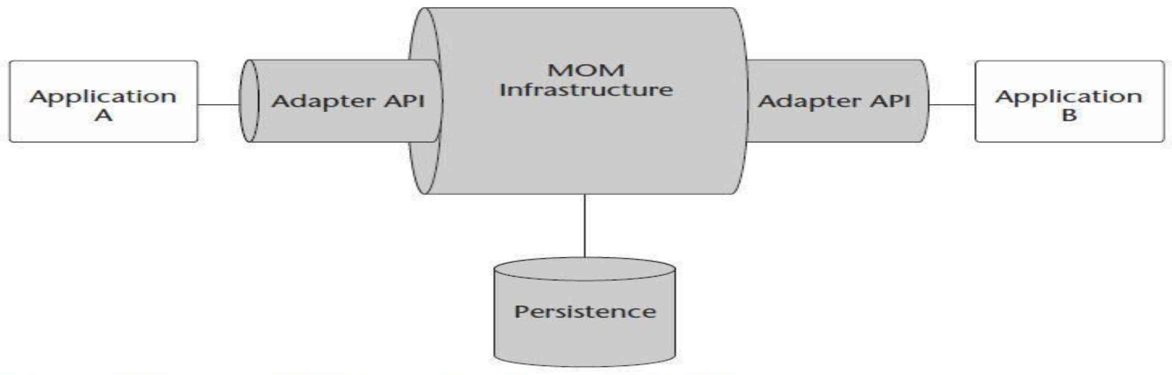


Figure 1.6 A typical MOM-based architectural model.

Some of the common challenges while implementing a MOM-based application environment have been the following:

■ Most of the standard MOM implementations have provided native APIs for communication with their core infrastructure. This has affected the portability of applications across such implementations and has led to a specific vendor lock-in.

■ The MOM messages used for integrating applications are usually based upon a proprietary message format without any standard compliance.

<u>Challenges in Distributed Computing:</u>

Distributed computing technologies like CORBA, RMI, and DCOM have been quite successful in integrating applications within a homogenous environment inside a local area network. As the Internet becomes a logical solution that spans and connects the boundaries of businesses, it also demands the interoperability of applications across networks. This section discusses some of the common challenges noticed in the CORBA-, RMI-, and DCOM-based distributed computing solutions:

■ Maintenance of various versions of stubs/skeletons in the client and server environments is extremely complex in a heterogeneous network environment.

■ Quality of Service (QoS) goals like Scalability, Performance, and Availability in a distributed environment consume a major portion of the application's development time.

■ Interoperability of applications implementing different protocols on heterogeneous platforms almost becomes impossible. For example, a DCOM client communicating to an RMI server or an RMI client communicating to a DCOM server.

■ Most of these protocols are designed to work well within local networks. They are not very firewall friendly or able to be accessed over the Internet.

The Role of J2EE and XML in Distributed Computing:

The emergence of the Internet has helped enterprise applications to be easily accessible over the Web without having specific client-side software installations. In the Internet-based enterprise application model, the focus was to move the complex business processing toward centralized servers in the back end. The first generation of Internet servers was based upon Web servers that hosted static Web pages and provided content to the clients via HTTP (Hyper Text Transfer Protocol). HTTP is a stateless protocol that connects Web browsers to Web servers, enabling the transportation of HTML content to the user.

This was the beginning of server-side scripting using technologies like CGI, NSAPI, and ISAPI. With many organizations moving their businesses to the Internet, a whole new category of business models like business-to-business (B2B) and business-to-consumer (B2C) came into existence. This evolution leads to the specification of J2EE architecture, which promoted a much more efficient platform for hosting Web-based applications. J2EE provides a programming model based upon Web and business components that are managed by the J2EE application server.

The application server consists of many APIs and low-level services available to the components. These low-level services provide security, transactions, connections and instance pooling, and concurrency services, which enable a J2EE developer to focus primarily on business logic rather than plumbing. The power of Java and its rich collection of APIs provided the perfect solution for developing highly transactional, highly available and scalable enterprise applications. Based on many standardized industry specifications, it provides the interfaces to connect with various back-end legacy and information systems. J2EE also provides excellent client connectivity capabilities, ranging from PDA to Web browsers to Rich Clients (Applets, CORBA applications, and Standard Java Applications).

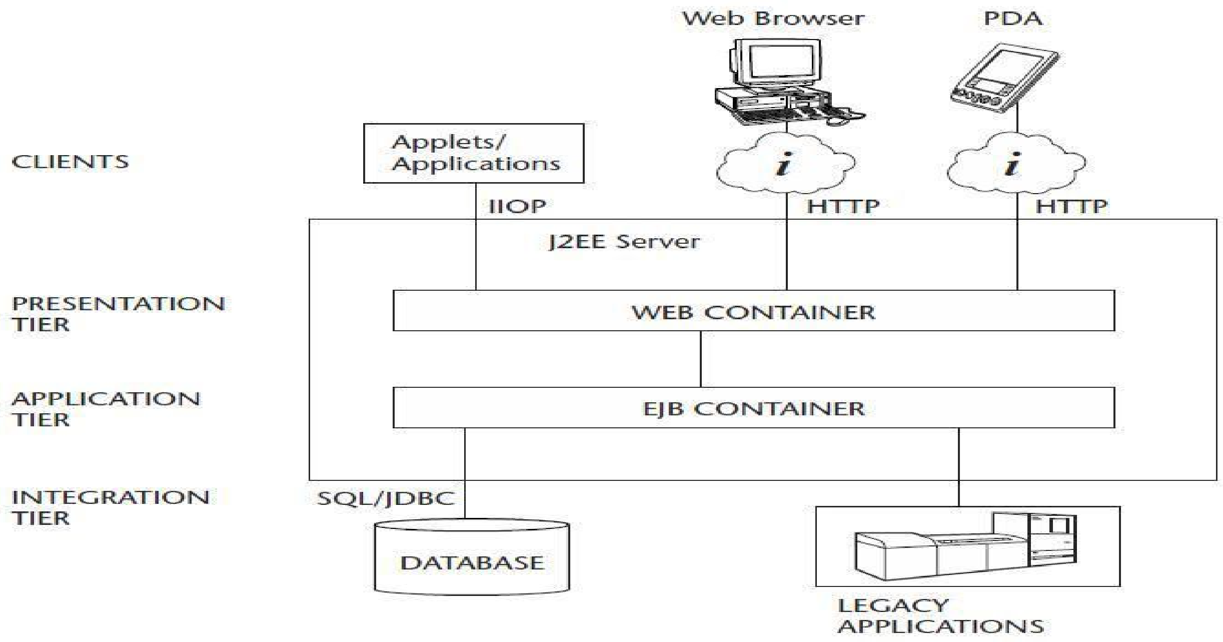


Figure 1.7 J2EE application architecture.

Figure 1.7 shows various components of the J2EE architecture. A typical J2EE architecture is physically divided in to three logical tiers, which enables clear separation of the various application components with defined roles and responsibilities.

The following is a breakdown of functionalities of those logical tiers:

Presentation tier: The Presentation tier is composed of Web components, which handle HTTP quests/responses, Session management, Device independent content delivery, and the invocation of business tier components.

Application tier: The Application tier (also known as the Business tier) deals with the core business logic processing, which may typically deal with workflow and automation. The business components retrieve data from the information systems with well-defined APIs provided by the application server.

Integration tier: The Integration tier deals with connecting and communicating to back-end Enterprise Information Systems (EIS), database applications and legacy applications, or mainframe applications.

Emergence of Web Services and SOA:

Today, the adoption of the Internet and enabling Internet-based applications has created a world of discrete business applications, which co-exist in the same technology space but without interacting with each other.

The increasing demands of the industry for enabling B2B, application-to-application (A2A), and inter-process application communication has led to a growing requirement for service-oriented architectures. Enabling service- oriented applications facilitates the exposure of business applications as service components enable business applications from other organizations to link with these services for application interaction and data sharing without human intervention. By leveraging this architecture, it also enables interoperability between business applications and processes.

By adopting Web technologies, the service-oriented architecture model facilitates the delivery of services over the Internet by leveraging standard technologies such as XML. It uses platform-neutral standards by exposing the underlying application components and making them available to any application, any platform, or any device, and at any location.

Today, this phenomenon is well adopted for implementation and is commonly referred to as Web services. Although this technique enables communication between applications with the addition of service activation technologies and open technology standards, it can be leveraged to publish the services in a register of yellow pages available on the Internet. This will further redefine and transform the way businesses communicate over the Internet. This promising new technology sets the strategic vision of the next generation of virtual business models and the unlimited potential for organizations doing business collaboration and business process management over the Internet.

Web services can be programmed in a variety of languages, old and new. The obvious way to publish a web service is with a web server; a web service client needs to execute on a machine that has network access, usually over HTTP, to the web server. In more technical terms, a web service is a distributed software system whose components can be deployed and executed on physically distinct devices.

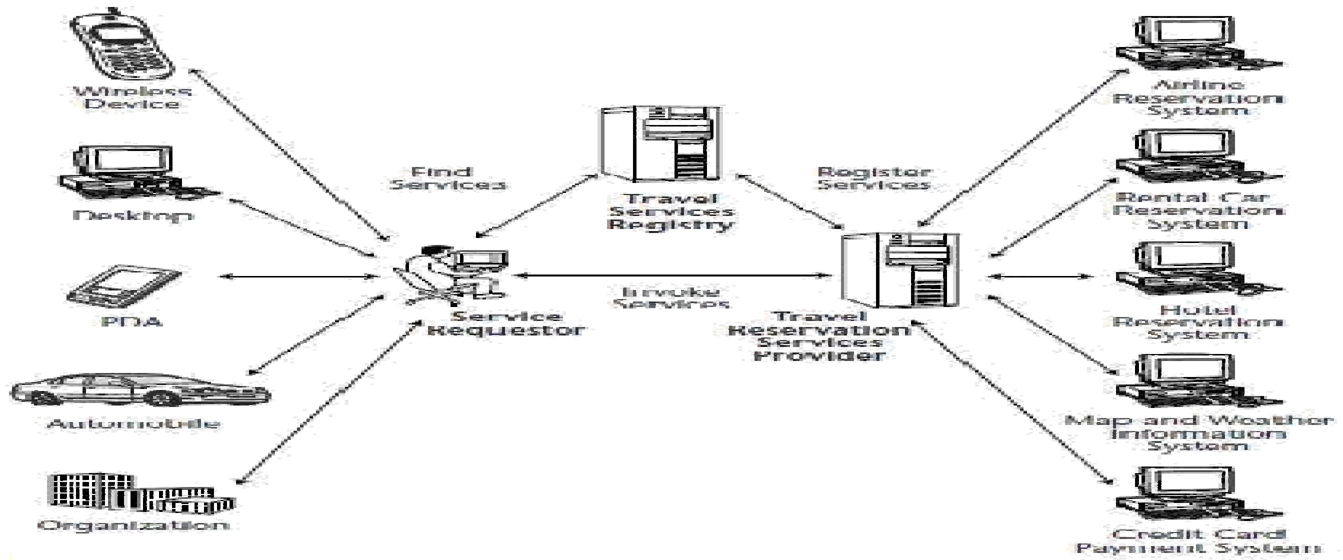


Figure 2.1 An example scenario of Web services.

Consider, for example, a web server *host1* that hosts a web service and a mobile device *host2* that hosts an application issuing requests against the service on *host1* (see Figure 1-9). Web services may be more architecturally complicated than this, of course; for one thing, a service may have many clients issuing requests against it, and the service itself may be composed of other services. For instance, a stock-picking web service might consist of several code components, each hosted on a separate commercial-grade web server, and any mix of PCs, handhelds, and other networked devices might host programs that consume the service.

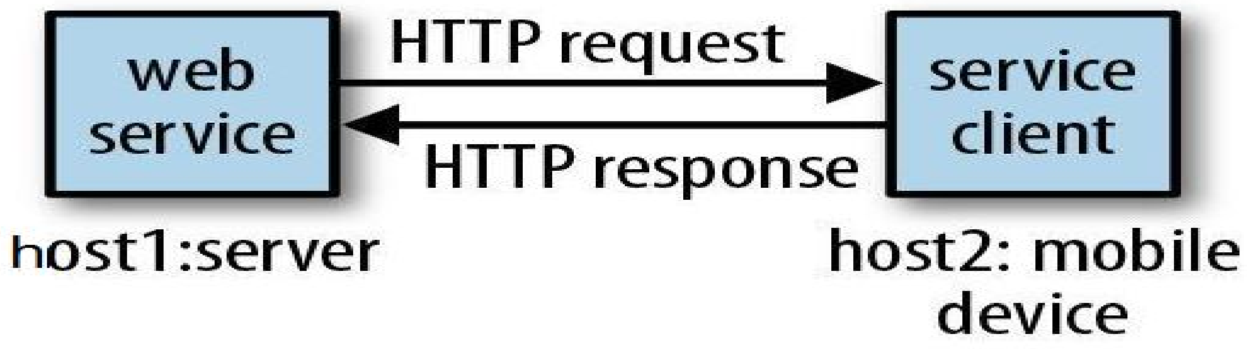


Figure 1-9. A web service and one of its clients

An HTTP request goes, by definition, from client to server, and an HTTP response goes, also by definition, from server to client. For web services over HTTP, the HTTP messages are the infrastructure, and these HTTP messages can be combined into basic conversational patterns that characterize a web service. For example, if the web service conversation starts with an HTTP request that expects an HTTP response, the pattern is the familiar *request/response* conversation. By contrast, if the conversation starts with an HTTP message from the server, a message that expects a message from the client in return, then the pattern is *solicit/response*. Richer conversational patterns can be composed out of such basic two-message patterns. Indeed, these two-message patterns are composed of even more primitive ones: a message from client to server without a response is a pattern known as *one-way,* and the reverse pattern, from server to client without a client response, is known as *notification*. Web services tend to be simple in structure. The four conversational patterns enumerated just now cover most modern web services, and request/response is the pattern that still dominates.

UNIT-I

Assignment-Cum-Tutorial Questions

SECTION-A

I)Objective Questions:

1). CORBA stands for ____________

2).__________ are used to convert your application into Web-Application. [ ]

a).Strut Services b).Web Services

c).Java Services d).Browser Action

3). Web Services are ____________ [ ]

a).None of these b).Application Desigining Tool

c).Application IDE d).Application Components

4).Web services are self contained and self desribing?
(True/False)

5). Web services can be discovered using ____________ [ ]

a) UDDII b) UDDI c) UDDDI d)UDII

6). _____________ is the basis for web services?

a)PHP b)XML c)GGI d)CSS

7). Which of the following is considered as Web Service Platform Elements? [ ]

A) UDDI B) WSDL C) SOAP D) All

8).Java supports RMI,RMI stands for? [ ]

A) Random Method Invocation B)Remote Memory Interface

C)Remote Method Invocation D)Random Memory Invocation

9).In RMI architecture which layer intercepts method calls made by the client redirects these calls to a remote RMI service? [ ]

A)stub & skeleton layer B)Application Layer

C)Remote Reference Layer D)Transport Layer

10).In the ____________client invokes the request and then blocks waiting for the response [ ]

A)Deferred synchronous Invocation B)one way Invocation

C)Synchronous Invocation D)Two-Way Invocation

11). ____________ provides programmers a familiar programming model by executing the local procedural calls to a distributed environment [ ]

A)Distributed Environment B)Permanent Procedural call

C)Process and File D)Remote Procedure Call

12). ____________refers to computing technologies in which the hardware and software components are distributed across the network [ ]

A) Client and server B)User and system

C)User and File Server D)User and DB Server

13). ____________serves as the glue between the client and server applications respectively , and that ORB [ ]

A)ORB&ORB Interface B)CORBA IDL stubs & Skeletons

C)Client and servant D)Client and server

14).An RMI server is responsible for ____________ [ ]

A)Creating an instance of the remote object B)Exporting remote object

C)Binding instance of the remote object to RMI Registry D)All

15). ____________ servers as the glue between CORBA object implementations and the ORB itself [ ]

A)The object Adapter B)Dynamic Skeleton Interface

C)Server Process Activation D)Client process Activation

16).What are the layers of RMI architecture____________ [ ]

A)Stub and Skeleton Layer B)Remote Reference Layer C)Transport Layer D)All

17).Microsoft DCOM remote protocol is also referred to as ____________ [ ]
A)Object RPC or ORPC B)Opinion RPC C)Server RPC D)Client RPC

18). Point out the wrong statement: [ ]
a) SOA provides the standards that transport the messages and makes the infrastructure to support it possible
b) SOA provides access to reusable Web services over a SMTP network
c) SOA offers access to ready-made, modular, highly optimized, and widely shareable components that can minimize developer and infrastructure costs
d) None of the mentioned

19). Which of the following describes a message-passing taxonomy for a component-based architecture that provides services to clients upon demand ? [ ]
a) SOA b) EBS c) GEC d) All of the mentioned

20). Point out the correct statement: [ ]
a) Service Oriented Architecture (SOA) describes a standard method for requesting services from distributed components and managing the results
b) SOA provides the translation and management layer in an architecture that removes the barrier for a client obtaining desired services
c) With SOA, clients and components can be written in different languages and can use multiple messaging protocols
d) All of the mentioned

SECTION-B

II).Descriptive Questions:

1.Briefly explain the evolution of Distributed Computing.
2.List and explain core distributed computing technologies.
3.Write a short note on the following DCT

a) Client / Server architecture b) CORBA

4.Write a short note on the following DCT

c) JAVA RMI d) MS DCOM and MOM

5.Outline the challenges in Distributed Computing.

6.Outline the role of J2EE ,XML in Distributed Computing.

7.Define the emergence of Web Services.

8. Compare the advantages of CORBA over a traditional client / server application model.

9.Analyze the limitations of DCOM Model.

10.Does JAVA RMI-IIOP support dynamic downloading of classes?

11.Illustrate how does Core Distributed Computing Techniques provide web services?

12.Illustrate what happens when you use Distributed Computing in web services?

UNIT – II

Objective:

To learn the fundamentals of web service

Syllabus:

Introduction to web services:

Web services architecture and its characteristics, Core building blocks of web services, standards and technologies available for implementing web services, web services communication, basic steps of implementing web services, developing web services enabled applications.

Learning Outcomes:

- Understand the characteristics and core building blocks of WS.
- Identify standards and technologies available for WS.
- Learn the basic steps to implement WS.
- Understand the design and development of WS enabled application.

LEARNING MATERIAL

Introduction to web services:

- A web service is any piece of software that makes it available over the internet and uses a standardized XML messaging system.
- XML is used to encode all communications to a web service.
- A web service is a collection of open protocols and standards used for exchanging data between applications or systems.
- Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer.
- This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.
- Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains.
- These applications can be local, distributed, or web-based.
- Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.

Following are the features of Web service:

- Is available over the Internet or private (intranet) networks.
- Uses a standardized XML messaging system.
- Is not tied to any one operating system or programming language.
- Is self-describing via a common XML grammar.
- Is discoverable via a simple find mechanism.

How Does a Web Service Work?

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP.

A web service takes the help of:

- XML to tag the data.
- SOAP to transfer a message.
- WSDL to describe the availability of service.

You can build a Java-based web service on Solaris that is accessible from your Visual Basic program that runs on Windows.

You can also use C# to build new web services on Windows that can be invoked from your web application that is based on JavaServer Pages (JSP) and runs on Linux.

The basic characteristics of a Web services application model are as follows:

- Web services are based on XML messaging, which means that the data exchanged between the Web service provider and the user are defined in XML.
- Web services provide a cross-platform integration of business applications over the Internet.
- To build Web services, developers can use any common programming language, such as Java, C, C++, Perl, Python, C#, and/or Visual Basic, and its existing application components.
- web services are not meant for handling presentations like HTML context—it is developed to generate XML for uniform accessibility through any software application, any platform, or device.
- Because Web services are based on loosely coupled application components, each component is exposed as a service with its unique functionality
- web services use industry-standard protocols like HTTP, and they can be easily accessible through corporate firewalls.

- Web services can be used by many types of clients.
- Web services vary in functionality from a simple request to a complex business transaction involving multiple resources.
- All platforms including J2EE, CORBA, and Microsoft .NET provide extensive support for creating and deploying Web services.
- Web services are dynamically located and invoked from public and private registries based on industry standards such as UDDI and ebXML.

Benefits of using Web Services:

Exposing the Existing Function on the network

- A web service is a unit of managed code that can be remotely invoked using HTTP, that is, it can be activated using HTTP requests.
- Web services allows you to expose the functionality of your existing code over the network.
- Once it is exposed on the network, other application can use the functionality of your program.

Interoperability

- Web services allow various applications to talk to each other and share data and services among themselves. Other applications can also use the web services.
- For example, a VB or .NET application can talk to Java web services and vice versa.
- Web services are used to make the application platform and technology independent.

Standardized Protocol

- Web services use standardized industry standard protocol for the communication.

- All the four layers (Service Transport, XML Messaging, Service Description, and Service Discovery layers) use well-defined protocols in the web services protocol stack.
- This standardization of protocol stack gives the business many advantages such as a wide range of choices, reduction in the cost due to competition, and increase in the quality.

<u>Low Cost of Communication</u>

- Web services use SOAP over HTTP protocol, so you can use your existing low-cost internet for implementing web services.
- This solution is much less costly compared to proprietary solutions like EDI/B2B.
- Besides SOAP over HTTP, web services can also be implemented on other reliable transport mechanisms like FTP.

<u>Web services architecture and its characteristics</u>:

- The Web services architecture is a technology stack that identifies standards-based application components, which ensures interoperability among Web services providers and requesters.
- It adopts service-oriented architecture (SOA) concepts using standards-based messages and communication protocols.
- Web-services architecture consists of many layers of interrelated logical components built using standards-based technologies.
- The logical components representing the layers provide standardized components and communication for defining and describing the services, discovering and subscribing to the services, transporting for service communication, aggregating a set of services, and collaborating with services.
- They also facilitate standards-based mechanisms for building end-to-end security, services provisioning and management, and delivering other QoS such as reliability, scalability, manageability, and availability.

There are two ways to view the web service architecture:

The first is to examine the individual roles of each web service actor.

The second is to examine the emerging web service protocol stack.

There are three major roles within the web service architecture:

- Service Provider .
- Service Requestor.
- Service Registry.

Service Provider: This is the provider of the web service. The service provider implements the service and makes it available on the internet.

Service Requestor: This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

Service Registry: This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.

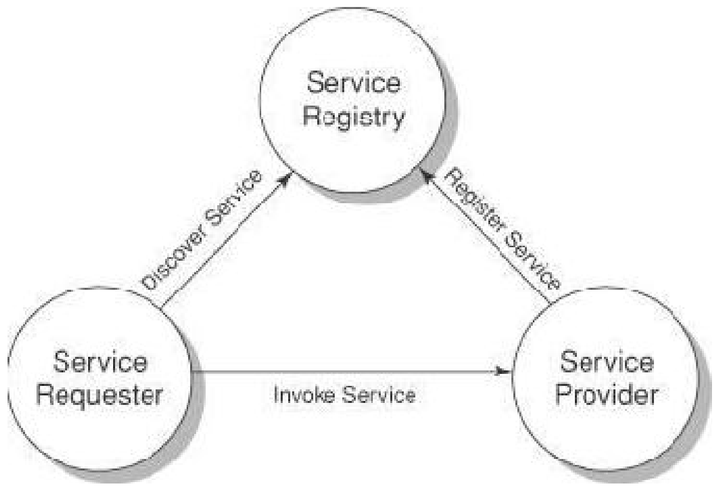


Figure: Operational Models or Roles in Web Services

Web services have the following special behavioral characteristics:

XML-Based : Web Services uses XML at data representation and data transportation layers.

Loosely Coupled: A consumer of a web service is not tied to that web service directly.

Coarse-Grained: Businesses and the interfaces that they expose should be coarse-grained.Web services technology provides a natural way of defining coarse-grained services that access the right amount of business logic.

Ability to be Synchronous or Asynchronous: Asynchronous clients retrieve their result at a later point in time, while synchronous clientsreceive their result when the service has completed. Asynchronous capability is a key factor in enabling loosely coupled systems.

Supports Remote Procedure Calls(RPCs): A web service supports RPC by providing services of its own, equivalent to those of a traditional component, or by translating incoming invocations into an invocation of an EJB or a .NET component.

Supports Document Exchange: Web services support the transparent exchange of documents to facilitate business integration.

2. Core building blocks of web services:

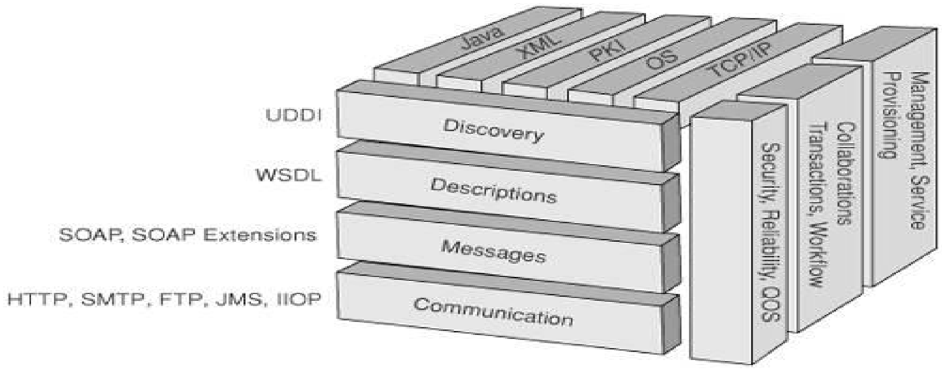

Java
XML
PKI
OS
TCP/IP
UDDI
Discovery
WSDL
Descriptions
SOAP, SOAP Extensions
Messages
HTTP, SMTP, FTP, JMS, IIOP
Communication
Security, Reliability, QOS
Collaborations Transactions, Workflow
Management, Service Provisioning

Figure: Technology/Protocol Stack or Core Building Blocks of Web services architecture.

Components of Web Services:

XML-RPC:

This is the simplest XML-based protocol for exchanging information between computers.

- XML-RPC is a simple protocol that uses XML messages to perform RPCs.
- Requests are encoded in XML and sent via HTTP POST.
- XML responses are embedded in the body of the HTTP response.
- XML-RPC is platform-independent.
- XML-RPC allows diverse applications to communicate.
- A Java client can speak XML-RPC to a Perl server.
- XML-RPC is the easiest way to get started with web services.

SOAP:

SOAP is an XML-based protocol for exchanging information between computers.

- SOAP is a communication protocol.
- SOAP is for communication between applications.
- SOAP is a format for sending messages.
- SOAP is designed to communicate via Internet.
- SOAP is platform independent.
- SOAP is language independent.
- SOAP is simple and extensible.
- SOAP allows you to get around firewalls.
- SOAP will be developed as a W3C standard.

WSDL:

WSDL is an XML-based language for describing web services and how to access them.

- WSDL stands for Web Services Description Language.
- WSDL was developed jointly by Microsoft and IBM.

- WSDL is an XML based protocol for information exchange in decentralized and distributed environments.
- WSDL is the standard format for describing a web service.
- WSDL definition describes how to access a web service and what operations it will perform.
- WSDL is a language for describing how to interface with XML-based services.
- WSDL is an integral part of UDDI, an XML-based worldwide business registry.
- WSDL is the language that UDDI uses.

UDDI:

UDDI is an XML-based standard for describing, publishing, and finding web services.

- UDDI stands for Universal Description, Discovery, and Integration.
- UDDI is a specification for a distributed registry of web services.
- UDDI is platform independent, open framework.
- UDDI can communicate via SOAP, CORBA, and Java RMI Protocol.
- UDDI uses WSDL to describe interfaces to web services.
- UDDI is seen with SOAP and WSDL as one of the three foundation standards of web services.
- UDDI is an open industry initiative enabling businesses to discover each other and define how they interact over the Internet.

3. Standards and technologies available for implementing web services:

All the latest standards related to web services are

Transports:

BEEP, the Blocks Extensible Exchange Protocol (formerly referred to as BXXP), is a framework for building application protocols. It has been standardized by IETF and it does for Internet protocols what XML has done for data.

- Blocks Extensible Exchange Protocol (BEEP)

Messaging:

These messaging standards and specifications are intended to give a framework for exchanging information in a decentralized, distributed environment.

- SOAP 1.1 (Note)
- SOAP 1.2 (Specification)
- Web Services Attachments Profile 1.0
- SOAP Message Transmission Optimization Mechanism

Description and discovery:

Web services are meaningful only if potential users may find information sufficient to permit their execution. The focus of these specifications and standards is the definition of a set of services supporting the description and discovery of businesses, organizations, and other web services providers; the web services they make available; and the technical interfaces which may be used to access those services.

- UDDI 3.0
- WSDL 1.1 (Note)
- WSDL 1.2 (Working draft)
- WSDL 2.0 (Working Group)

Security:

Using these security specifications, applications can engage in secure communication designed to work with the general web services framework.

- Web Services Security 1.0
- Security Assertion Markup Language (SAML)

Management:

Web services manageability is defined as a set of capabilities for discovering the existence,availability, health, performance, usage, as well as the control and configuration of a web service within the web services architecture. As web services become pervasive and critical to business operations, the task of man aging and implementing them is imperative to the success of business operations.

l**The following core technologies are used for web services:**

- Extensible Markup Language (XML)
- SOAP
- Web Services Description Language (WSDL)

4. Web Services Communication Model

In Web services architecture, depending upon the functional requirements, it is possible to implement the models with RPC-based synchronous or messaging-based Synchronous/asynchronous communication models. These communication models need to be understood before Web services are designed and implemented.

RPC-Based Communication Model

- The RPC-based communication model defines a request/response-based, synchronous communication.
- When the client sends a request, the client waits until a response is sent back from the server before continuing any operation.
- Typical to implementing CORBA or RMI communication, the RPC-based Web services are tightly coupled and are implemented with remote objects to the client application.
- Figure 3.3 represents an RPC-based communication model in Web services architecture.
- The clients have the capability to provide parameters in method calls to the Web service provider.
- Then, clients invoke the Web services by sending parameter values to the Web service provider that executes the required methods, and then sends back the return values.
- Additionally, using RPCbased communication, both the service provider and requestor can register and discover services, respectively.

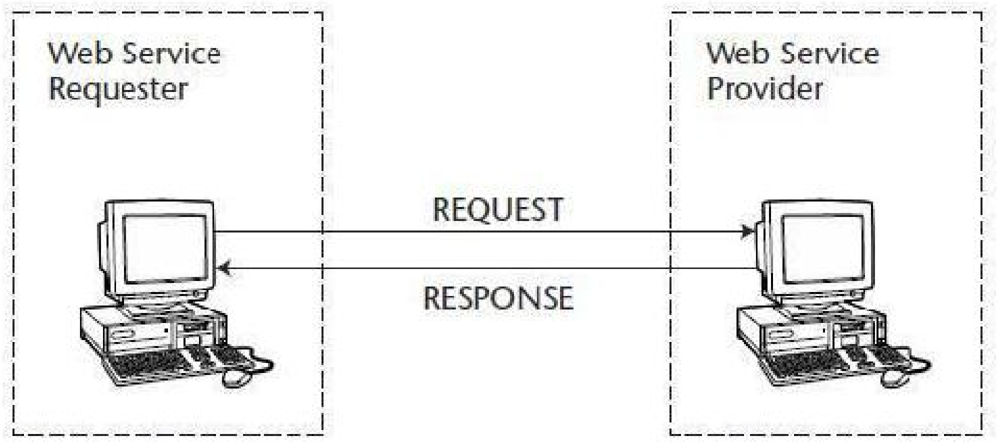


Figure 3.3 RPC-based communication model in Web services.

5. Basic steps of implementing web services

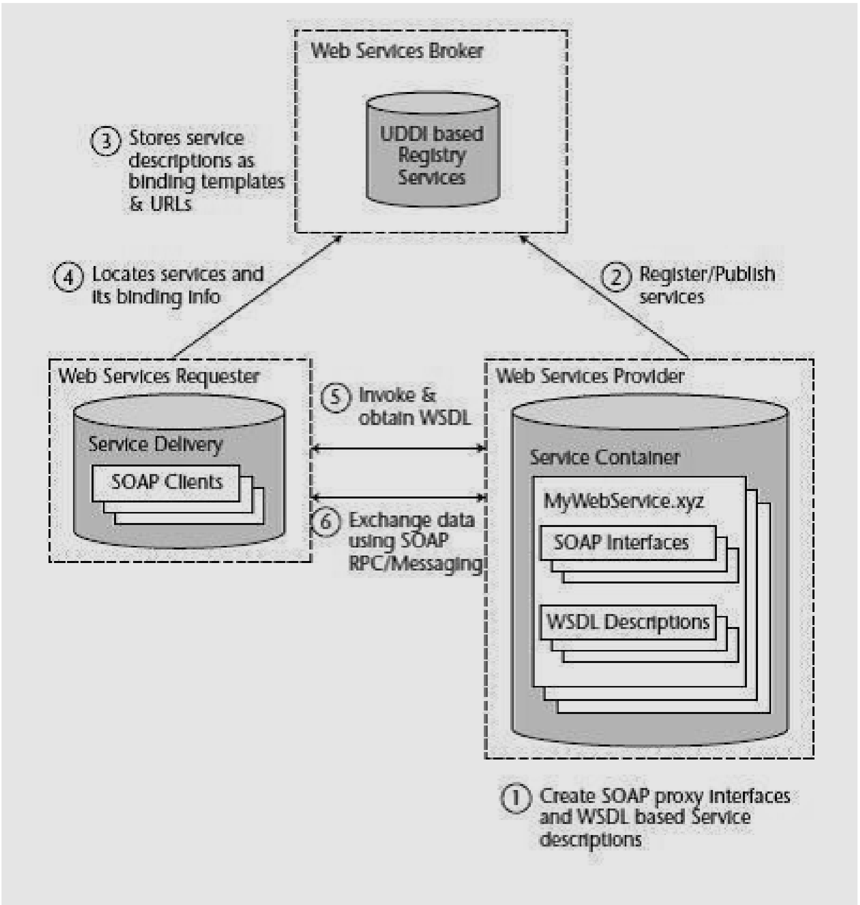

Web Services Broker
UDDI based Registry Services
3 Stores service descriptions as binding templates & URLs
4 Locates services and its binding info
2 Register/Publish services
Web Services Requester
Service Delivery
SOAP Clients
5 Invoke & obtain WSDL
6 Exchange data using SOAP RPC/Messaging
Web Services Provider
Service Container
MyWebService.xyz
SOAP Interfaces
WSDL Descriptions
1 Create SOAP proxy interfaces and WSDL based Service descriptions

Figure 3.5 Process steps involved in implementing Web services.

The process of implementing Web services is quite similar to implementing any distributed application using CORBA or RMI.

However, in Web services, all the components are bound dynamically only at its runtime using standard protocols.

Figure 3.5 illustrates the process highlights of implementing Web services.

1.The service provider creates the Web service typically as SOAP based service interfaces for exposed business applications.

The provider then deploys them in a service container or using a SOAP runtime environment, and then makes them available for invocation over a network.

The service provider also describes the Web service as a WSDL-based service description, which defines the clients and the service container with a consistent way of identifying the service location, operations, and its communication model.

2. The service provider then registers the WSDL-based service description with a service broker, which is typically a UDDI registry.

3. The UDDI registry then stores the service description as binding templates and URLs to WSDLs located in the service provider environment.

4. The service requestor then locates the required services by querying the UDDI registry. The service requestor obtains the binding information and the URLs to identify the service provider.

5. Using the binding information, the service requestor then invokes the service provider and then retrieves the WSDL Service description for those registered services. Then, the service requestor creates a client proxy application and establishes communication with the service provider
using SOAP.

6. Finally, the service requestor communicates with the service provider and exchanges data or messages by invoking the available services in the service container.

6. Developing web services enabled applications:

Preparing for the JAX-WS samples

To prepare for this sample, we import sample code, which is a simple web application that includes Java classes and an EJB.

Importing the sample In this section, prepare the environment for the JAX-WS web services application samples:

1. In the Java EE perspective, select File Import
2. Select General Existing Projects into Workspace.
3. In the Import Projects window, select Select archive file.
4. Click Browse. Navigate to the 4884code\webservices folder and select the RAD8WebServiceStart.zip file. Click Open.
5. Click Select All and click Finish. After the build, no warning or error messages are displayed in the workspace.

Testing the application:

To start and test the application, follow these steps:

1. In the Servers view, start WebSphere Application Server V8.0.
2. Right-click the server and select Add and remove projects.
3. In the Add and Remove Projects window, select RAD8WebServiceEAR, click Add, and then click Finish.
4. Expand WebContent, right-click search.html, RAD8WebServiceWeb and select Run As ▪ Run on Server.
5. Select Choose an existing server and select the v8.0 server to run the application.
Then click Finish.
6. When the search page opens in a web browser, in the Social Security number field, enter an appropriate value, for example, 111-11-1111, and click Search. If everything works correctly, you can see the customer's full name, first account, and its balance, which have been read from the memory data.
7. Test the stateless session EJB, SimpleBankFacade, by using the Universal Test Client (UTC).

The following methods are valid: – getCustomerFullName(ssn): Retrieves the full name (use 111-
11-1111) – getNumAccounts(ssn): Retrieves the number of accounts – getAccountId(ssn, int):
Retrieves the account ID by index (0,1,2,...) – getAccountBalance(accountId): Retrieves the balance.

UNIT-II
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. The __________________is responsible for transporting messages between applications.

2. A ________is a collection of open protocols and standards used for exchanging data between applications or systems.

3. The XML Messaging layer is responsible for transporting messages between applications. [True/False]

4. The service requestor utilizes an existing web service by opening a network connection and sending an XML request. [True/False]

5. A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP.

[True/False]

6. Which of the following is considered as Web Service Platform Elements ?
A) All of these B) UDDI C) WSDL D) SOAP [ ]

7. Web services can be discovered using _______. [ ]

A) UDDII B) UDDI C) UDDDI D) UDII

8. _____ process steps are there for implementing web services. [ ]

A) 12 B) 8 C) 6 D)1

9. __________ are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. [ ]

A) Web Services B) Software Services

C) System Services D) Hardware Services

10. Which of the following is correct about Service Description layer in Web Service Protocol Stack? [ ]

A) This layer is responsible for describing the public interface to a specific web service.

B)Currently, service description is handled via the Web Service Description Language (WSDL).

C) Both of the above. D) None of the above.

11. Which of the following are correct layers in protocol stack of web services.

A) Service Transport layer & XML Messaging layer [ ]

B) Service Description layer & Service Discovery layer

C) Both and A & B D) None of the above

12. Which of the following is true about behavioral characteristics of web services? [ ]

A) Web Services uses XML at data representation and data transportation
B)A consumer of a web service is not tied to that web service directly.

C) Businesses and the interfaces that they expose should be coarse-grained. Web services technology provides a natural way of defining coarse-grained services that access the right amount of business logic.

D) All of the above.

13.Which of the following is true about Web services? [ ]

A - Web services are open standard (XML, SOAP, HTTP etc.) based Web applications.

B - Web services interact with other web applications for the purpose of exchanging data.

C - Web Services can convert your existing applications into Web-applications. D - All of the above.

14) Which of the following is true about Web service? [ ]

A - It is available over the Internet or private (intranet) networks.

B - It uses a standardized XML messaging system.

C - It is not tied to any one operating system or programming language.

D - All of the above.

15) What is the purpose of XML in a web service? [ ]

A - A web services takes the help of XML to tag the data, format the data.

B - A web service takes the help of XML to transfer a message.

C - A web service takes the help of XML to describe the availability of service.

D - None of the above.

16) Which of the following is the benefits of having XML based WEB services?

A) Using XML eliminates any networking, operating system, or platform binding. [ ]

B). Web Services based applications are highly interoperable application at their core level.

C) Both of the above

D)None of the above.

17) Which of the following is correct about XML RPC? [ ]

A) XML-RPC is a simple protocol that uses XML messages to perform RPCs.

B) XML-RPC is platform-independent.

C)XML-RPC allows diverse applications to communicate.

D) All of the above.

18) ________is responsible for describing the public interface to a specific web service. [ ]

A) Service Discovery B) Service Description

C) Service Request D) Service Response

SECTION-B

SUBJECTIVE QUESTIONS

1. Define Web Services and explain introductory concepts of WS.

2. Draw and explain the architecture of Web Services

3. Briefly explain the characteristics of Web Services.

4. List and explain core building blocks of Web Services.

5. Write a short note on the following to implement WS

a) Standards b) Technologies

6. What is the importance of RPC-Based communication model?

7. With a neat sketch explain basic steps of implementing web services.

8. Explain the use of the components in Web Services.

9. Compare and contrast the advantages of Web Services than Distributed Computing Technologies.

10. Distinguish between SOAP and WSDL in Web Services.

11. Do Web services supports Remote Procedure Calls(RPCs)?

12.Distinguish the XML Messaging layer and Service Transport layer in Web Service.

13. Outline the purpose of Service Discovery layer and Service Description Layer in Web Service Protocol Stack.

UNIT – III

WEB SERVICES

Objective:

To learn the use of WSDL for describing web services.

Syllabus:

Describing web services WSDL: WSDL in the world of web services, Web services life cycle, Anatomy of WSDL definition document, WSDL bindings, WSDL tools, limitations of WSDL

Learning Outcomes: Students will be able to

- understand the need of WSDL in web services.
- identify the components in WS life cycle.
- learn the anatomy of WSDL definition document.
- understand WSDL bindings and tools.
- know the limitations of WSDL

LEARNING MATERIAL

WSDL in the world of web services

WSDL stands for Web Services Description Language. It is the standard format for describing a web service. WSDL was developed jointly by Microsoft and IBM.

Features of WSDL :

- WSDL is an XML-based protocol for information exchange in decentralized and distributed environments.
- WSDL definitions describe how to access a web service and what operations it will perform.
- WSDL is a language for describing how to interface with XML-based services.
- WSDL is an integral part of Universal Description, Discovery, and Integration (UDDI), an XML-based worldwide business registry.
- WSDL is the language that UDDI uses.
- WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'. WSDL Usage
- WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet.
- A client program connecting to a web service can read the WSDL to determine what functions are available on the server.
- Any special data types used are embedded in the WSDL file in the form of XML Schema.
- The client can then use SOAP to actually call one of the functions listed in the WSDL.

History of WSDL

WSDL 1.1 was submitted as a W3C Note by Ariba, IBM, and Microsoft for describing services for the W3C XML Activity on XML Protocols in March 2001.

WSDL Elements

A WSDL document contains the following elements:

Definition : It is the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.

Data types : The data types to be used in the messages are in the form of XML schemas.

Message : It is an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.

Operation : It is the abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message.

Port type : It is an abstract set of operations mapped to one or more end-points, defining the collection of operations for a binding; the collection of operations, as it is abstract, can be mapped to multiple transports through various bindings.

Binding : It is the concrete protocol and data formats for the operations and messages defined for a particular port type.

Port : It is a combination of a binding and a network address, providing the target address of the service communication.

Service : It is a collection of related end-points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions.

Web Services Description Language (WSDL)

- Microsoft and IBM released the first version of the WSDL specification jointly, in September 2000, briefly after announcing a UDDI specification along with 36 other companies.
- This version of WSDL was based on two precedent description languages: Network Accessible Services Specification Language (NASSL) and (SOAP Contract Language SCL), from IBM and Microsoft, respectively.
- Later on in March 2001, the same companies joined by a few others submitted the WSDL 1.1 specification to W3C. Thus, currently the WSDL specification is in works at W3C. Officially, it is a W3C Note that forms the basis of the upcoming WSDL 1.2 specification from W3C.
- This chapter goes into detail in understanding WSDL 1.1.
- WSDL in the World of Web Services
- WSDL, as we know, is a description language for Web services. So what does this exactly mean? This means that WSDL

represents information about the interface and semantics of how to invoke or call a Web service. A WSDL definition contains four important pieces of information about the Web service:

- Interface information describing all the publicly available functions.
- Data type information for the incoming (request) and outgoing (response) messages to these functions.
- Binding information about the protocol to be used for invoking the specified Web service.
- Address information for locating the specified Web service.

- Once we develop a Web service, we create its WSDL definition.
- We can create this definition either manually or by using a tool.
- Many tools are available for generating a WSDL definition from existing Java classes, J2EE components (such as Servlets/EJBs), or from scratch. Once the WSDL definition is created, a link to it is published in a Web services registry (based on UDDI, for instance), so that the potential user(s) of this Web service can follow this link and find out the location of the Web service, the function calls that it supports, and how to invoke these calls.
- Finally, the user(s) would use this information to formulate a SOAP request or any other type of request based on the binding protocol supported, in order to invoke the function on a Web service.

Web Service Life Cycle:

In Figure 3.1, all of the communication over the wire takes place onSOAP. The following list explains the steps depicted in Figure 3.1.

Step 1: illustrates a service provider publishing its Web service to a UDDI
registry. This is when the service provider would create a WSDL definition and publish a link to this definition along with the rest of theWeb service information to a UDDI registry.

Step 2: illustrates an interested service user locating the Web service and
finally obtaining information about invoking the Web service from the
published WSDL definition. This step involves downloading a WSDL
definition to the service user system and desterilizing WSDL to a Java
class (or any other language). This Java interface serves as a proxy to the
actual Web service. It consists of the binding information of the Web
service.

Step 3: shows the service user binding at runtime to the Web service. In
this step, the service user's application would make use of the Java interface representing WSDL as a proxy, in order to bind to the Web
service.

Step 4: finally shows the service user invoking the Web service based on
the service invocation information it extracted from the Web service WSDL definition. This is when the service user's application would make use of the Java interface representing WSDL as a proxy, in order to invoke the methods/functions exposed by the Web service.

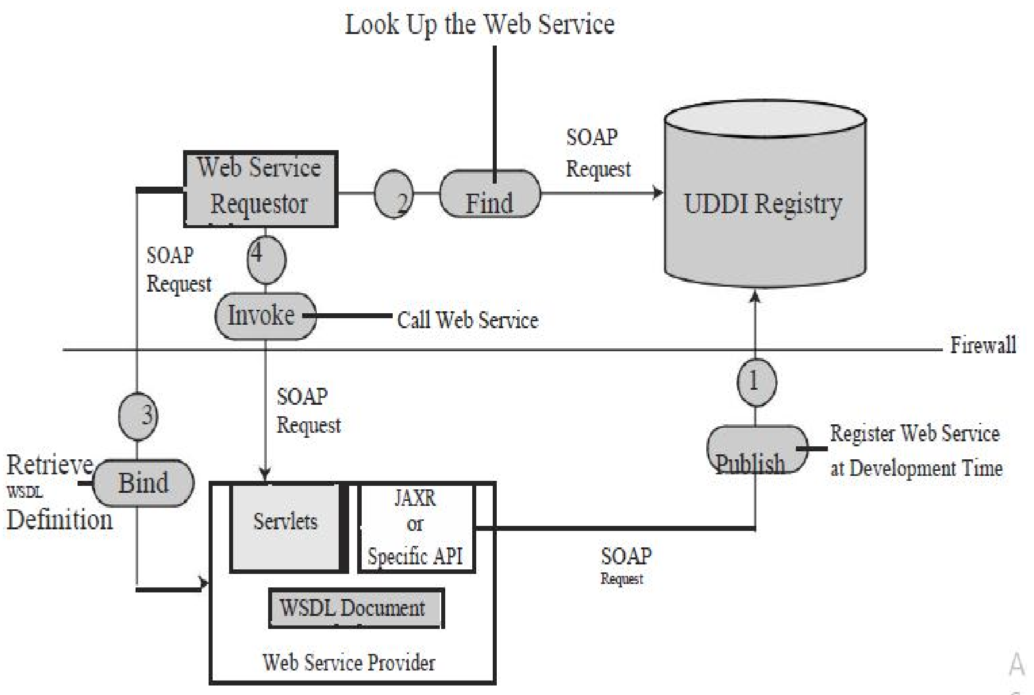


Figure 3.1 Web service life cycle.

Anatomy of a WSDL Definition Document

A WSDL definition document consists of the following seven key structural elements:

<definitions>: A WSDL document is a set of definitions. These definitions are defined inside the <definitions> element, which is the root element in a WSDL document. It defines the name of the Web service and also declares the name spaces that are used throughout the rest of the WSDL document.

<types>: This element defines all of the data types that would be used to describe the messages that are exchanged between the Web service and the service user. WSDL does not mandate the use of a specific typing system. However, as per the WSDL specification, XML Schema is the default typing system.

<message>: This element represents a logical definition of the data being transmitted between the Web service and the service user. This element describes a one-way message, which may represent a request or response sent to or from the Web service. It contains zero or more message <part> elements, which basically refer to the request parameters or response return values.

<portType>: This element defines the abstract definition of the operations supported by a Web service, by combining various request and response messages defined by <message> elements. Each operation refers to an input
message and an output message.
<binding>: This element specifies a concrete protocol and data format used for representing the operations and messages defined by a particular <portType> on the wire.
<port>: This element specifies an address for binding to the Web service.
<service>: This element aggregates a set of related <port> elements,each which uniquely specify the binding information of the Web service. <service> consisting of multiple <port> elements essentially represents the capability of the service to be invoked over multiple bindings. More information on WSDL bindings is discussed in the next section.First, let's take a look at Listing 3.1, which shows a WSDL document describing First,a weather information Web service, WeatherInfoService.This WSDL definition is present in the WeatherInfo.wsdl file.

```
 <?xml version=”1.0”?>

<definitions name=”WeatherInfo”
targetNamespace=”http://myweather.com/weatherinfo.wsdl”
xmlns:tns=”http://myweather.com/weatherinfo.wsdl”
xmlns:xsd1=”http://myweather.com/weatherinfo.xsd”
xmlns:soap=”http://schemas.xmlsoap.org/wsdl/soap/”
xmlns=”http://schemas.xmlsoap.org/wsdl/”>
<types>
<schema targetNamespace=
“http://myweather.com/weatherinfo.xsd” xmlns=
“http://www.w3.org/2000/10/XMLSchema”>
<element name=”WeatherInfoRequest”>
<complexType>
<all>
<element name=”Country”
type=”string”/>
<element name=”Zip”
type=”string”/>
```

```
<element name=”Instant”
type=”string”/>
</all>
</complexType>
</element>
<element name=”WeatherInfo”>
<complexType>
<all>
<element name=”FTemp”
type=”float”/>
<element name=”Humidity”
type=”float”/>
</all>
</complexType>
</element>
</schema>
</types>
<message name=”GetWeatherInfoInput”>
<part name=”WeatherInfoRequestSpec”
element=”xsd1:WeatherInfoRequest”/>
</message>
<message name=”GetWeatherInfoOutput”>
```

Listing 5.1 WeatherInfo.wsdl.

```
 element=”xsd1:WeatherInfo”/>
</message>
<portType name=”WeatherInfoPortType”>
<operation name=”GetWeatherInfo”>
<input message=”tns:GetWeatherInfoInput”/>
<output message=”tns:GetWeatherInfoOutput”/>
</operation>
</portType>
<binding name=”WeatherInfoSoapBinding”
```

```
type=”tns:WeatherInfoPortType”>
<soap:binding style=”document”
transport=”http://schemas.xmlsoap.org/soap/http”/>
<operation name=”GetWeatherInfo”>
<soap:operation soapAction=
“http://myweather.com/GetWeatherInfo”/>
<input>
<soap:body use=”literal”/>
</input>
<output>
<soap:body use=”literal”/>
</output>
</operation>
</binding>
<service name=”WeatherInfoService”>
<documentation>
Provides Weather Information
</documentation>
<port name=”WeatherInfoPort”
binding=”tns:WeatherInfoSoapBinding”>
<soap:address location=
“http://myweather.com/provideweatherinfo”/>
</port>
</service>
</definitions>
```

Listing 5.1 WeatherInfo.wsdl. *(continued)*

Now, let’s understand how exactly WeatherInfo.wsdl describes the WeatherInfoService Web service.

<definitions> Element:

The <definitions> element specifies the name of the document in which this

WSDL definition is stored, which is WeatherInfo in our case.This element specifies namespaces that would be used in the rest of the WSDL document. The following are the two important namespaces that the <definitions> element defines:

WSDL instance specific namespace: The targetNamespace attribute of the <definitions> element lets the WSDL document make references to itself as an XML Schema namespace. Note how-ever that it is not required for the WSDL document to actually exist at the address specified targetNamespace attribute. This attribute is just a mechanism to refer to our WSDL definition in a unique way.

Default namespace for a WSDL document: The default namespace is specified by xmlns=”http://schemas.xmlsoap.org/wsdl/”. The default namespace indicates that all of the elements in this WSDL definition without a namespace prefix, such as <types>, <message>, and <portType>, are part of this namespace.

<message> Element:

WeatherInfo.wsdl defines two <message> elements.

The first <message> definition named GetWeatherInfoInput will be used later to define the input message of the GetWeatherInfo operation.

The second <message> definition named GetWeatherInfoOutput will be used later to define the output message of the GetWeatherInfo operation. This binding of <message> definitions to an actual operation is defined in the <portType> element.

Again, each of these <message> definitions consists of a <part> element. In case of the GetWeatherInfoInput message, <part> essentially specifies the name,that is, WeatherInfoRequestSpec, and type, that WeatherInfoRequest, of the request parameter to GetWeath-erInfo operation. Whereas, in case of the GetWeatherInfoOutput message, <part> refers to the name and type of the return value sent within the response of the GetWeatherInfo operation. Note that both WeatherInfoRequest and WeatherInfo, which were referred to by the type attribute of <part> element also were defined by the preceding <types> element.

Also in cases where operations take multiple arguments or return multiple values, the <message> element can define multiple <part> elements.

<portType> Element:

The <portType> element in WeatherInfo.wsdl defines a single operation named GetWeatherInfo by combining the <input> message as defined by the

GetWeatherInfoInput <message> element and the <output> message as defined by the GetWeatherInfoOutput <message> element. Note the use of WeatherInfo.wsdl as a target namespace by the <input> and <output> elements.

Four types of operations are supported by WSDL:

One-way operation. One-way operation represents a service that just receives the message, and thus a one-way operation is typically defined by

a single <input> message element.

Request-response operation. A request-response operation repre-sents a

service that receives a request message and sends a response message.

Typically, a request-response operation is defined by one <input> message

followed by one <output> message. An optional <fault> element also can

be included in the definition of a request-response operation to specify the

abstract message format for any error messages that may be output as a

result of the operation.

The GetWeatherInfo operation follows the request-response transmission pattern.

Solicit-response operation: A solicit-response operation represents a service that sends a request message and that receives the response message. Such operations are therefore defined by one <output> message, followed by an <input> message. A solicit-response opera-tion also can include a <fault> element in order to specify the for-mat of any error messages resulting from the operation.

Notification operation. This operation represents a service that sends a message, hence this kind of operation is represented by a single <output> element.

Figure 5.2 provides the pictorial representation of the previous four transmission types.

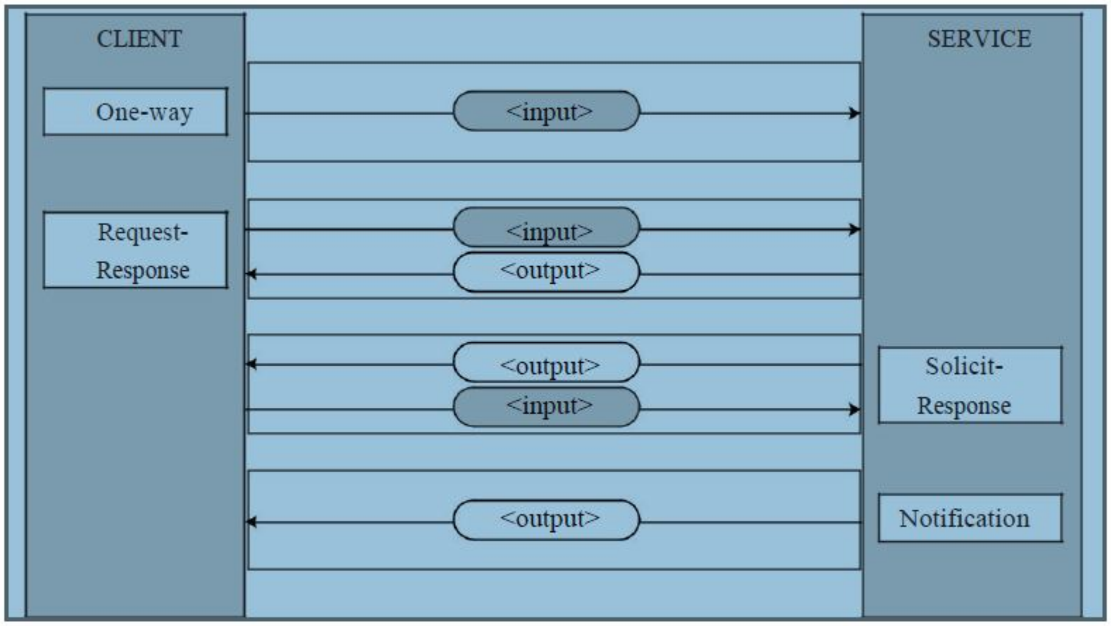


Figure:WSDL operational types.

<binding> Element:

A binding defines the message format and protocol details for operations and messages defined by a particular <portType>. There may be any number of bindings for a given <portType>. The type attribute of the <binding> element refers to the <portType> that it binds to, which is WeatherInfoPortType in our case. Our WeatherInfoService specifies a SOAP binding, as is defined in the WSDL 1.1 specification. The WSDL 1.1 SOAP binding is discussed in detail in a later section titled *SOAP Binding*.

<service> Element:

The <service> element specifies the location of the service. Because our WeatherInfoService is bound to SOAP, we use the <soap:address> element and specify the service URL as http://myweather.com /provideweatherinfo/.

Now, let’s take a look at the support for various bindings in the WSDL 1.1 specification.

WSDL Bindings:

In WSDL, the term binding refers to the process of associating protocol or data format information with an abstract entity such as <message>, <operation>, or <portType>. In this section, we examine the support for bindings in the WSDL 1.1 specification. Let’s begin with the WSDL binding extensions.

WSDL Binding Extensions:

WSDL allows user-defined elements, also known as *Extensibility Elements*,
under various elements defined by a default WSDL namespace. These elements are commonly used to specify some technology-specific binding, although they can be used for other purposes as well. Extensibility elements, when used to specify a technology-specific binding, are known as *WSDL Binding Extensions*.

Extensibility elements provide a powerful mechanism for extending WSDL because they enable support for network and message protocols to be revised without having to revise the WSDL specification.

The base specification of WSDL defines three WSDL binding extensions, which are as follows:

- SOAP binding
- HTTP GET & POST binding
- MIME binding

We will take a look at the most commonly used WSDL binding extension, the
SOAP binding, in a later section titled *SOAP Binding*.

WSDL Binding Support for Operations:

All four types of operations supported by WSDL—one-way, request-response, solicit-response, and notification—represent an abstract notion only.

Binding describes the concrete correlation to these abstract notions. Binding determines how the messages are actually sent, for instance, within a single communication (for example, an HTTP request/response) or as two independent communications (for example, two HTTP requests). Thus, binding for a specific operation type must be defined in order to successfully carry out that type of operation. Note that although the WSDL structure supports the bindings for these four operations, the WSDL specification defines bindings for only one-way and request-response operations.

Hence, in order to use WSDL to describe services that support solicit-response and/or notification types of operations, the communica-tion protocol of the Web service must define the WSDL binding extensions, thus enabling the use of these operations.

Let’s now take a look at SOAP binding as defined by the WSDL 1.1
specification.

SOAP Binding:

WSDL 1.1 defines a binding for SOAP 1.1 endpoints. This binding provides the following SOAP protocol specific information:

- An indication that the binding is bound to the SOAP 1.1 protocol.
- A way of specifying an address for a SOAP endpoint.
- The URI for the SOAP action HTTP header for the HTTP binding of SOAP
- A list of definitions of headers that are transmitted as part of the SOAP envelope

Let's examine the SOAP binding of the request-response RPC operation over HTTP as defined in the WeatherInfo.wsdl file shown earlier (see the section titled *Anatomy of a WSDL Definition Document*).

<soap:binding>

The <soap:binding> element is defined in WeatherInfo.wsdl as follows:

<soap:binding style=”document”
transport=”http://schemas.xmlsoap.org/soap/http”/>

The <soap:binding> element says that the binding is bound to the SOAP protocol format, that is, envelope, header, and body. However, this element does not give any information about the format or encoding of the message. This element must be present whenever describing services that have a SOAP binding.

The style attribute indicates whether the operations supported by this bindingare RPC-oriented or document-oriented. In RPC-oriented commu-nication, themessages contain parameter and return values, whereas in document-oriented communication, the messages contain document(s). This information about the style of communication can be useful because it helps in selecting the programming model for communicating with the Web service. For example, if a Web service is described to support RPC, we can choose a JAX-RPC programming model to communicate with it, or if a Web service is described to support document-style communication, we can appropriately choose a JAXM programming model.

The transport attribute specifies the transport binding for the SOAP protocol. The URI value of http://schemas.xmlsoap.org/soap/http corresponds to the HTTP binding as defined in the SOAP specification. Similarly, respective URIs can be used to indicate other types of transports such as SMTP or FTP.

<soap:operation>

The <soap:operation> element is defined in WeatherInfo.wsdl as follows:

<soap:operation soapAction="http://myweather.com/GetWeatherInfo"/>

The <soap:operation> element defines the information with regard to communication style and the SOAP action header at that specific operation level.

The semantics of the style attribute remains the same as that for a <soap:binding> element.

The soapAction attribute specifies the value of a SOAP action header in the form of a URI. "Developing Web Services Using SOAP."

<soap:body>

The <soap:body> element is defined in WeatherInfo.wsdl as follows:

<soap:body use="literal"/>

This element defines how the message <part> elements appear inside the SOAP body element. Based on the style of communication, RPC-oriented or document-oriented, the <Body> element of the SOAP message is constructed.

The use attribute indicates whether the message <part> elements are encoded using some encoding rules or whether the <part> elements already define the concrete schema of the message.

If the value of the use attribute is "encoded", then each message <part> refers to an abstract type defined through the type attribute. These abstract types are then used to produce a concrete definition of the message by applying the encoding specified by an encodingStyle attribute.

Consider the following example:

```
<output>
<soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:acmens:acmeservice"
use="encoded"/>
        </output>
```

The <soap:body> element in this code depicts a SOAP binding wherein the

body of the output SOAP message consists of abstract <part> elements that are used to produce the concrete definition of the message by applying the
encodingStyle as defined in http://schemas .xmlsoap.org/soap/encoding/.

<soap:address>

The <soap:address> element is defined as follows in

WeatherInfo.wsdl:

<soap:address location=

“http://myweather.com/provideweatherinfo”/>

The <soap:address> element specifies an address for the given service port.

WSDL Tools:

WSDL tools typically provide functionality in terms of the following:

WSDL generation: Generating WSDL from an existing service Component for example, a J2EE component or a Java Bean component or from scratch.

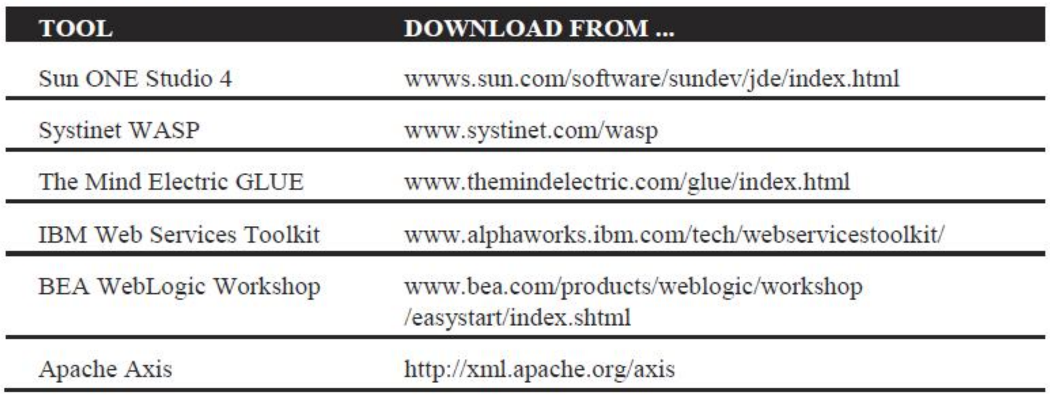
WSDL compilation: A typical WSDL compiler would generate the necessary
data structures and a skeleton for the implementation of the service. The generated implementation skeleton contains all the methods (operations) that are described in the given WSDL definition.

WSDL proxy generation: This functionality can read a WSDL and produce a specific language binding (for example, Java or Perl) consisting of all the code required to bind the Web service and to invoke the Web service functions at runtime. This functionality is typically used at the client end.

Many WSDL tools provide support for these three functionalities.

Table 3.1 lists some of the famous ones in the Java Web Services space.

Table 3.1 WSDL Tools



TOOL	DOWNLOAD FROM ...
Sun ONE Studio 4	wwws.sun.com/software/sundev/jde/index.html
Systinet WASP	www.systinet.com/wasp
The Mind Electric GLUE	www.themindelectric.com/glue/index.html
IBM Web Services Toolkit	www.alphaworks.ibm.com/tech/webservicestoolkit/
BEA WebLogic Workshop	www.bea.com/products/weblogic/workshop /easystart/index.shtml
Apache Axis	http://xml.apache.org/axis

In the following section, we examine the WSDL tools provided by the Systinet WASP platform

Support for WSDL in Systinet WASP 4.0:

Systinet WASP provides two tools for working with WSDL: Java2WSDL and WSDL Compiler. Both of these tools accomplish two different types of functionalities related to WSDL:

Generating WSDL from a Java class that is a potential candidate for a Web service: This functionality is taken care of by the Java2WSDL tool.

Generating Java code from an existing WSDL: This functionality is taken care of by the WSDL Compiler.

We will check out both these tools in the following two sections.

Generating WSDL from a Java Class

In situations in which an implementation of the Web service has already been created first, the Java2WSDL tool can be used to generate WSDL. This tool provides a lot of options for generating WSDL from an existing Java implementation.

To understand the functioning of this tool, consider the following Java class:

```
package jws.ch5;
public class WeatherInfoJavaService
{
public float GetWeatherInfo (String sCountry, String sZip,String sInstance)
{
Talk to some backend services to get hold
of the weather information
Return the weather information;
a manipulated value for now.
return 65.0f;
}
public void SetWeatherInfo (String sCountry, String sZip, String sInstance, String sTemperature)
{
Update the backend weather databases
with this information
                }
}
```

- As can be seen from the previous listing, this class provides get and set methods. The main job of this class is to manage information related to weather. Note that this is a very simplistic version of such a weather information service.
- For example, we want to expose this class as a Web service. In which case, we also decide to provide the description of the interface of this Web service as a WSDL. Our Web service supports SOAP-based communication, and hence, a SOAP binding as well. Thus, this fact also should be considered while generating WSDL using the Java2WSDL tool.
- Once the WSDL has been generated, it can be registered in a registry such as UDDI accompanied by the business- and other service-related information. So when the prospective Web service users find the Web service, they can obtain the WSDL description corresponding to this Web service and start using it.

Limitations of WSDL:

- WSDL 1.1 has an obvious limitation: its incapability of being able to describe complex business Web services, which typically are constituted by orchestrating multiple finer-grained Web services.
- This drawback is due to the lack of support for workflow descriptions in WSDL.
- To overcome these limitations of WSDL, standards such as ebXML Collaborative Protocol Profile/Collaborative Protocol Agreement (CCP/A), Business Process Specification Schema (BPSS), and Web Services Choreography Interface (WSCI) can be leveraged. An EbXML set of technologies can be used to build business Web services.

UNIT-III
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1).What is WSDL? [ ]

a) Web Services Data Language b) Web services Description Language

c) Web Services Describing Language d) None of These

2)WSDL was developed jointly by [ ]

a)Microsoft and IBM b)Microsoft and Intel

c)IBM and Oracle d)Microsoft and Oracle

3)WSDL is pronounced as [ ]

a)Wis-dell b)wis-dull c)wiz-dull d)wiz-dell

4). ________ is an extension of WSDL. [ ]

a)XLink b)XLang c)XQuery d)XPath

5). Which of these is not the major element of WSDL document which describes the describes a web service

[ ]

a) portType b) message c) binding d) attribute

6). WSDL port describes the interfaces exposed by a [ ]

a) web servicer b)web service c)web browser d)None of these

7). Which directory of web service interface described by WSDL? [ ]

a)HTTP b)DNS c)UDDI d)XML

8). The information about the names of the methods the parameters that can be passed,and the values that are returned from the functions is controlled in some Webservices by a description specified in ________ [ ]

a)XML b)SOAP c)WSDL d)WSL

9) A resource on the Web is uniquely identified by its URI,which means________ [ ]

a) Uniform Resource Identifier b) Universal Registered Identifier

c) Uniform Registered Identifier d) Universal Resource Identifier

10)WSDL is written in [ ]

a)WML b)XML c)HTML d)CSS

11). The information about the names of the methods, the parameters that can be passed, and the values that are returned from the functions is controlled in some Web services by a description specified in ____

a)XML b)SOAP c)WSDL d)WC [ ]

12) WSDL contains information about the names of the methods, the parameters that can be passed, and the values that are returned from the functions. (True/False)

13)What are the web service platform elements? [ ]

a)SOAP,UDDI,XML b)HTTP.WSDL

c)UDDI,XML,SOAP d)SOAP,UDDI,WSDL

14)The binding element has two attributes,they are [ ]

a)name and type b)style and transport

c)prototype and name d)port and service

15).Which of the following is used to locate and describe web service?[ ]

a)SOAP B)Web page c)WSDL D)UDDI

SECTION-B

SUBJECTIVE QUESTIONS

1.Define WSDL and explain briefly.

2.With a neat sketch explain web services lifecycle.

3.Outline the Anatomy of WSDL definition document.

4.What are the different types of bindings available in WSDL?

5.List and explain the tools of WSDL.

6.Briefly explain the limitations of WSDL.

7.Identify six major elements provided by WSDL.

8.Give four transaction primitives/operation types supported by WSDL.

9.Apply the concepts of WSDL in the world of web services.

10.Differentiate between WSDL binding and WSDL port.

11.Illustarate the relationship of ports with in a service.

12.Mention what things need to be taken care for ports while binding?

13.Outline how endpoints are defined inWSDL?

14.Illustrate how the HTTP GET/POST Binding Extends WSDL?

UNIT – IV

Objective:

To learn the fundamentals of SOAP.

Syllabus:

Core fundamentals of SOAP:

Anatomy of a SOAP Message (SOAP message structure), SOAP Encoding, SOAP Message Exchange Models, SOAP Communication, SOAP Messaging, SOAP Security, Developing web services using SOAP, Building SOAP Web Services, Developing SOAP Web Services Using Java, Limitations of SOAP.

Learning Outcomes:

- Students will be able to
- know the anatomy of SOAP message.
- identify SOAP exchange models, communication and security.
- learn the basic steps to build SOAP web services.
- develop SOAP Web Services using java.
- identify the limitations of SOAP.

LEARNING MATERIAL

Introduction:

- SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. SOAP uses XML technologies to define an extensible messaging framework, which provides a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation-specific semantics.

- SOAP initially was developed by DevelopMentor, Inc., as a platform-independent protocol for accessing services, objects between applications, and servers using HTTP-based communication.
- SOAP used an XML-based vocabulary for representing RPC calls and its parameters and return values.
- SOAP stands for Simple Object Access Protocol.
- SOAP is a W3C recommendation for communication between two applications.

SOAP message structure:

SOAP defines the structure of an XML document, rules, and mechanisms that can be used to enable communication between applications.

It does not mandate a single programming language or a platform, nor does it define its own language or platform.

A SOAP message is an ordinary XML document containing the following elements −

- **Envelope −** Defines the start and the end of the message. It is a mandatory element.
- **Header −** Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end-point. It is an optional element.
- **Body −** Contains the XML data comprising the message being sent. It is a mandatory element.
- **Fault −** An optional Fault element that provides information about errors that occur while processing the message.

All these elements are declared in the default namespace for the SOAP envelope –

http://www.w3.org/2001/12/soap-envelope and the default namespace for SOAP encoding and data types is – http://www.w3.org/2001/12/soap-encoding

The following block depicts the general structure of a SOAP message

```
<?xml version="1.0"?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">

  <SOAP-ENV:Header>

    ...

    ...

  </SOAP-ENV:Header>
  <SOAP-ENV:Body>

    ...

    ...

    <SOAP-ENV:Fault>

      ...

      ...

    </SOAP-ENV:Fault>


    ...

  </SOAP-ENV:Body>
</SOAP_ENV:Envelope>
```

In the scenario in Listing 4.1, the SOAP message is embedded in an HTTP request for getting the book price information from www.wiley.com for the book *Developing Java Web Services*.

```
            POST /BookPrice HTTP/1.1
            Host: catalog.acmeco.com
            Content-Type: text/xml; charset=”utf-8”
            Content-Length: 640
            SOAPAction: “GetBookPrice”
<SOAP-ENV:Envelope
```

```
        xmlns:SOAP

        ENV="http://schemas.xmlsoap.org/soap/envelope/"

        xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"

        xmlns:xsd=http://www.w3c.org/2001/XMLSchema

SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Header>
        <person:mail
        xmlns:person="http://acmeco.com/Header/">xyz@acmeco.com
        </person:mail>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
        <m:GetBookPrice
        xmlns:m="http://www.wiley.com/jws.book.price
List"> <bookname xsi:type='xsd:string'>

Developing Java Web Services</bookname>
        </m:GetBookPrice>
</SOAP-ENV:Body>
</SOAP-ENV: Envelope>
```

Listing 4.1 SOAP request message.

Listing 4.2 shows the SOAP message embedded in an HTTP response returning the price of the book.

```
        HTTP/1.1 200 OK
        Content-Type: text/xml; charset="utf-8"

         Content-Length: 640
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
        xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
<SOAP-ENV:Header>
        <wiley:Transaction
        xmlns:wiley="http://jws.wiley.com/2002/booktx"SOAP-
ENV:mustUnderstand="1"> 5
</wiley:Transaction>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<m:GetBookPriceResponse
xmlns:m="http://www.wiley.com/jws.book.priceList">
<Price>50.00</Price>
</m:GetBookPriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4.2 SOAP response message.

SOAP Envelope

- The SOAP envelope is the primary container of a SOAP message's structure and is the mandatory element of a SOAP message.

- It is represented as the root element of the message as Envelope
- As we discussed earlier, it is usually declared as an element using the XML namespace http://schemas.xmlsoap.org/soap/envelope/.
- As per SOAP 1.1 specifications, SOAP messages that do not follow this namespace declaration are not processed and are considered to be invalid.
- Encoding styles also can be defined using a namespace under Envelope to represent the data types used in the message.

Listing 4.3 SOAP envelope element in SOAP message.

```
<SOAP-ENV:Envelope

xmlns:SOAPENV=http://schemas.xmlsoap.org/soap/envelo

pe/xmlns:xsi=http://www.w3c.org/2001/XMLSchema-

instance xmlns:xsd=http://www.w3.org/2001/XMLSchema

SOAP-ENV    encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”/>
<!--SOAP Header elements - -/>

<!--SOAP Body element - -/><SOAP-ENV:Envelope>
```

Listing 4.3 SOAP Envelope element.

- Envelope is a mandatory part of SOAP message.
- Every Envelope element must contain exactly one Body element.
- If an Envelope contains a Header element, it must contain no more than one, and it must appear as the first child of the Envelope, before the Body.
- The envelope changes when SOAP versions change.
- The SOAP envelope is specified using the *ENV*namespace prefix and the Envelope element.
- The optional SOAP encoding is also specified using a namespace name and the optional *encodingStyle*element, which could also point to an encoding style other than the SOAP one.
- A v1.1-compliant SOAP processor generates a fault upon receiving a message containing the v1.2 envelope namespace
- v1.2-compliant SOAP processor generates receives a message that does not include a *VersionMismatch* fault if it the v1.2 envelope namespace.

<u>SOAP Header</u>

- It is an optional part of a SOAP message.
- Header elements can occur multiple times.
- Headers are intended to add new features and functionality.

- The SOAP header contains header entries defined in a namespace.
- The header is encoded as the first immediate child element of the SOAP envelope.
- When multiple headers are defined, all immediate child elements of the SOAP header are interpreted as SOAP header blocks.

SOAP Header Attributes

A SOAP Header can have the following two attributes −

- Actor attribute

  The SOAP protocol defines a message path as a list of SOAP service nodes. Each of these intermediate nodes can perform some processing and then forward the message to the next node in the chain. By setting the Actor attribute, the client can specify the recipient of the SOAP header.

- MustUnderstand attribute

  It indicates whether a Header element is optional or mandatory. If set to true, the recipient must understand and process the Header attribute according to its defined semantics, or return a fault.

```
<SOAP-ENV:Header>
     <wiley:Transaction
     xmlns:wiley=”http://jws.wiley.com/2002/booktx”
```

```
        SOAP-ENV:mustUnderstand=”1”>
        <keyValue> 5 </keyValue>


        </wiley:Transaction>
</SOAP-ENV:Header>
```

Listing 4.4 SOAP Header element.

SOAP Body:

- The SOAP body is a mandatory element that contains the application-defined XML data being exchanged in the SOAP message.
- The body must be contained within the envelope and must follow any headers that might be defined for the message.
- The body is defined as a child element of the envelope, and the semantics for the body are defined in the associated SOAP schema.
- A Body block of a SOAP message can contain any of the following:
- RPC method and its parameters
- Target application (receiver) specific data
- SOAP fault for reporting errors and status information

```
<SOAP-ENV:Body>
<m:GetBookPrice
        xmlns:m=”http://www.wiley.com/jws.book.priceList/”>
        <bookname xsi:type=’xsd:string’>
        Developing Java Web services</bookname>
</m:GetBookPrice>
</SOAP-ENV:Body>
```

Listing 4.5 SOAP Body element.

SOAP Fault

- In a SOAP message, the SOAP Fault element is used to handle errors and to find out status information.

- This element provides the error and/or status information. It can be used within a Body element or as a Body entry.

- It provides the following elements to define the error and status of the SOAP message in a readable description, showing the source of the information and its details:
- **Faultcode :** The faultcode element defines the algorithmic mechanism for the SOAP application to identify the fault.
- It contains standard values for identifying the error or status of the

SOAP application.

- The namespace identifiers for these faultcode values are defined in

http://schemas.xmlsoap.org/soap/envelope/.

The following fault-code element values are defined in the SOAP 1.1 specification:

VersionMismatch : This value indicates that an invalid namespace is defined in the SOAP envelope or an unsupported version of a SOAP message.

MustUnderstand : This value is returned if the SOAP receiver node cannot handle and recognize the SOAP header block when the MustUnderstand attribute is set to 1. The MustUnderstand values can be set to 0 for false and 1 for true.

Client This faultcode is indicated when a problem originates from the receiving client. The possible problems could vary from an incorrect SOAP message, a missing element, or incorrect name-space definition.

Server This faultcode indicates that a problem has been encountered during processing on the server side of the application, and that the application could not process further because the issue is specific to the content of the SOAP message.

- **Faultstring:** The faultstring element provides a readable description of the SOAP fault exhibited by the SOAP application.
- **Faultactor:**The faultactor element provides the information about the ultimate SOAP actor (Sender/Receiver/Intermediary) in the message who is responsible for the SOAP fault at the particular destination of a message.
- **Detail:** The detail element provides the application-specific error or status information related to the defined Body block.

  <SOAP-ENV:Envelopexmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”

```
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding
/">
 <SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcode>SOAP-ENV:MustUnderstand</faultcode>

<faultstring>Header element missing</faultstring>
<faultactor>http://jws.wiley.com/GetBookPrice</faultactor>
<detail>
<wiley:error
xmlns:wiley="http://jws.wiley.com/GetBookPrice">
<problem>The Book name parameter missing.</problem>
</wiley:error>

</detail>
</SOAP-ENV:Fault>
</SOAP_ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4.6 SOAP Fault in a SOAP message.

<u>SOAP Encoding</u>

- The SOAP encoding defines a set of rules for expressing its data types.
- It is a generalized set of data types that are represented by the programming languages, databases, and semi-structured data required for an application.
- SOAP encoding also defines serialization rules for its data model using an encodingStyle attribute under the SOAP-ENV namespace that specifies the serialization rules for a specific element or a group of elements.

- SOAP encoding supports both simple- and compound-type values.

Simple Type Values

The definition of simple type values is based on the “W3C XML Schema, Part -2: Data types” specification.

Examples are primitive data types such as string, integer, decimal, and derived simple data types including enumeration and arrays.

The following examples are a SOAP representation of primitive data types:

```
<int>98765</int>
<decimal> 98675.43</decimal>
<string> Java Rules </string>
```

The derived simple data types are built from simple data types and are expressed in the W3C XML Schema.

Compound Type Values

Compound value types are based on composite structural patterns that represent member values as structure or array types.

The following sections list the main types of compound type values.

Structure Types

Listing 4.14 is an XML Schema of the Structure data type representing the “Shipping address” with sub elements like “Street,” “City,” and “State.”

```
 <xs:elementname=”ShippingAddress”
            xmlns:xs=”http://www.w3.org/2001
            /XMLS chema” >

<xs:complexType>

<xs:sequence>
<xs:element ref=”Street”type=”xsd:string”/>

<xs:element ref=”City” type=”xsd:string”/>

<xs:element ref=”State” type=”xsd:string”/>

<xs:element ref=”Zip” type=”xsd:string”/>

<xs:elementref=”Country”

type=”xsd:string”/>

</xs:sequence>

</xs:complexType>

</xs:element>
```

Listing 4.14 Structure data type.

<u>SOAP Message Exchange Models</u>

Basically, SOAP is a stateless protocol by nature and provides a compos-able one-way messaging framework for transferring XML between SOAP applications which are referred to as SOAP nodes. These SOAP nodes represent the logical entities of a SOAP message path to perform message routing or processing.

In a SOAP message, SOAP nodes are usually represented with an endpoint URI as the next destination in the message.

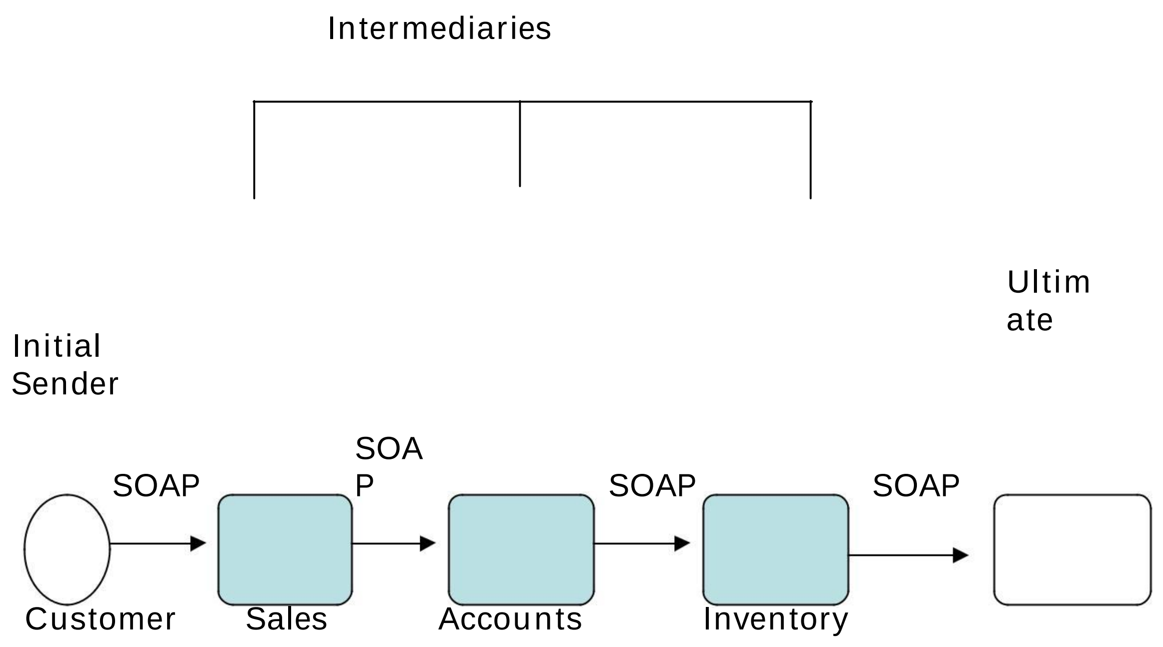
In a SOAP message, a SOAP node can be any of the following:

SOAP sender. The one who generates and sends the message.

SOAP receiver. The one who ultimately receives and processes the message with a SOAP response, message, or fault.

SOAP intermediary. The one who can play the role of a SOAP sender or SOAP receiver. In a SOAP message exchange model, there can be zero or more SOAP

intermediaries between the SOAP sender and receiver to provide a distributed processing mechanism for SOAP messages.



SOAP Communication

SOAP is designed to communicate between applications independent of the underlying platforms and programming languages.

To enable communication between SOAP nodes, SOAP supports the following two types of communication models:

SOAP RPC. It defines a remote procedural call-based synchronous communication where the SOAP nodes send and receive messages using request and response methods and exchange parameters and then return the values.

SOAP Messaging. It defines a document-driven communication where SOAP nodes send and receive XML-based documents using synchronous and asynchronous messaging.

Now, let’s explore the details of both the communication model and how it is represented in the SOAP messages.

SOAP RPC

The SOAP RPC representation defines a tightly coupled communication model based on requests and responses.

Using RPC conventions, the SOAP message is represented by method names with zero or more parameters and return values.

Each SOAP request message represents a call method to a remote object in a SOAP server and each method call will have zero or more parameters.

Similarly, the SOAP response message will return the results as return values with zero or more out parameters.

In both SOAP RPC requests and responses, the method calls are serialized into XML-based data types defined by the SOAP encoding rules.

Listing 4.24 is an example of a SOAP RPC request making a method call GetBookPrice for obtaining a book price from a SOAP server namespace

http://www.wiley.com/jws.book.priceList using a ”book-name” parameter of ”Developing Java Web Services”.

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
```

```
        xmlns:xsi=”http://www.w3c.org/2001/XMLSchema-
            instance”
        xmlns:xsd=”http://www.w3c.org/2001/XMLSchema”
         SOAP-ENV:encodingStyle
=”http://schemas.xmlsoap.org/soapencoding/”>
<SOAP-ENV:Header>
</SOAP-ENV:Header>
        <SOAP-ENV:Body>
             <m:GetBookPrice
xmlns:m=”http://www.wiley.com/jws.book.price   List”>
<bookname xsi:type=’xsd:string’>

Developing    Java    Web    services</bookname>
</m:GetBookPrice>
</SOAP-ENV:Body>
</SOAP-ENV: Envelope>
```

Listing 4.24 SOAP request using RPC-based communication.

The SOAP message in Listing 4.25 represents the SOAP RPC response after processing the SOAP request, which returns the result of the Get-BookPrice method from the SOAP server namespace http://www. wiley.com/jws.book.priceList using a “Price” parameter with “$50” as its value.

```
 <SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi=”http://www.w3c.org/2001/XMLSchema-instance”
```

```
        xmlns:xsd=http://www.w3c.org/2001/XMLSchema
              SOAP-ENV:encodingStyle
="http://schemas.xmlsoap.org/soap/encoding/"/>
              <SOAP-ENV:Body>
                    <m:GetBookPriceResponse xmlns:m="
                            http://www.wiley.com/jws.book.priceList">
                       <Price>50.00</Price>
                     </m:GetBookPriceResponse>
           </SOAP-ENV:Body>
           </SOAP-ENV:Envelope>
```

Listing 4.25 SOAP response message using RPC-based communication.

SOAP Messaging

- SOAP Messaging represents a loosely coupled communication model based on message notification and the exchange of XML documents.
- The SOAP message body is represented by XML documents or literals encoded according to a specific W3C XML schema, and it is produced and consumed by sending or receiving SOAP node(s).
- The SOAP sender node sends a message with an XML document as its body message and the SOAP receiver node processes it.

Listing 4.26 represents a SOAP message and a SOAP messaging-based communication. The message contains a header block InventoryNotice and the body product, both of which are application-defined and not defined by SOAP. The header contains information required by the receiver node and the body contains the actual message to be delivered.

```
<env:Envelope
 xmlns:env="http://www.w3.org/2001/12/soap-
 envelope"> <env:Header>

   <n:InventoryNotice
    xmlns:n="http://jws.wiley.com/Inventory">
    <n:productcode>J6876896896</n:productcode>
    </n: InventoryNotice>
 </env:Header>
 <env:Body>
 <m:product
    xmlns:m="http://jws.wiley.com/product">
    <m:name>Developing    Java     Web
    Services</m:name>
    <m:quantity>25000</n:quantity>
    <m:date>2002-07-01T14:00:00-05:00</n:date>
   </m:product>
 </env:Body>
</env:Envelope>
```

Listing 4.26 SOAP message using messaging-based communication.

<u>SOAP Messaging</u>

Based on the underlying transport protocol, to enhance the communication and message path model between the SOAP nodes, SOAP chooses an interaction pattern depending upon the communication model.

Although it depends upon SOAP implementation, SOAP messages may support the following messaging exchange patterns to define the message path and transmission of messages between SOAP nodes, including intermediaries.

It is important to note that these patterns are introduced as part of SOAP 1.2 specifications.

The most common SOAP messaging patterns are as follows:

- **One-way message.** In this pattern, the SOAP client application sends SOAP messages to its SOAP server without any response being returned. It is typically found in email messages.

- **Request/response exchange.** In this pattern, the SOAP client sends a request message that results in a response message from the SOAP server to the client.
- **Request/N*Response pattern.** It is similar to a request/response pattern, except the SOAP client sends a request that results in zero to many response messages from the SOAP server to the client.
- **Notification pattern.** In this pattern, the SOAP server sends messages to the SOAP client like an event notification, without regard to a response.

- **Solicit-response pattern.** In this pattern, the SOAP server sends a request message to the SOAP client like a status checking or an audit and the client sends out a response message.

SOAP Security

- Security in SOAP messages plays a vital role in access control, encryption, and data integrity during communication.

- In general, SOAP messages do not carry or define any specific security mechanisms.

However, using the SOAP headers provides a way to define and add features enabling the implementation of application-specific security in a form of XML-based metadata.

- The metadata information can be application-specific information incorporating message security with associated security algorithms like encryption and digital signatures
- More importantly, SOAP supports various transport protocols for communication, thus it also is possible to incorporate transport protocol-supported security mechanisms like SSL/TLS for SOAP messages.
- The W3C SOAP Security Extensions specifications were available as a Note to define encryption, authorization, and digital signatures in SOAP messages.
- But all of the security-related elements are identified using a single namespace identifier using the prefix

SOAP-SEC and with an associated URI using http://schemas.xmlsoap.org/soap /security/.

- It also defines the three security element tags <SOAP-SEC: Encryption>, <SOAP-SEC:Signature>, and <SOAP-SEC:Authorization>.

Use of these security tags enables the incorporation of encryption, digital signatures, and authorization in SOAP messages.

The following section takes a look at how to represent these security tags in a SOAP message.

SOAP Encryption

The use of XML-based encryption in SOAP permits secure communication and access control to be implemented by encrypting any element in the SOAP envelope. The W3C XML Encryption WG (XENC) defines the mechanisms of XML encryption in the SOAP messages. In SOAP communication, encryption can be done at the SOAP sender node or at any of the intermediaries in the message path.

Listing 4.33 is a sample representation of a SOAP message

SOAP Digital Signature

The use of an XML-based digital signature in SOAP messages provides message authentication, integrity, and non-repudiation of data during communication.

The SOAP sender node that originates the message applies an XML-based digital signature to the SOAP body and the receiver node validates the signature.

Listing 4.34 is a sample representation of a SOAP message using XML digital signatures.

using XML encryption for encrypting its data elements.

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”>
   <SOAP-ENV:Header>
   <SOAP-SEC:Encryption
        xmlns:SOAP-SEC=”http://schemas.xmlsoap.org/soap/security/”
        SOAP-ENV:actor=”some-URI”
        SOAP-ENV:mustUnderstand=”1”>
        <SOAP-SEC:EncryptedData>
          <SOAP-SEC:EncryptedDataReference URI=”#encrypted-
element”/>
        </SOAP-SEC:EncryptedData>
        <xenc:EncryptedKey xmlns:xenc=
“http://www.w3.org/2001/04/xmlenc#” Id=”myKey”
CarriedKeyName=”Symmetric Key”
Recipient=”Bill Allen”>
<xenc:EncryptionMethod
        Algorithm=”http://www.w3.org/2001/04/xmlenc#rsa-1_5”/>
        <ds:KeyInfo  xmlns:ds=“http://www.w3.org/2000/09/xmldsi  g#”>
        <ds:KeyName>Bill Allen’s RSA Key</ds:KeyName>
        </ds:KeyInfo>
        <xenc:CipherData>
        <xenc:CipherValue>ENCRYPTED KEY</xenc:CipherValue>
        </xenc:CipherData>
```

```
          <xenc:ReferenceList>
<xenc:DataReference URI="#encrypted-element"/>
          </xenc:ReferenceList>
          </xenc:EncryptedKey>
         </SOAP-SEC:Encryption>
        </SOAP-ENV:Header>
    <SOAP-ENV:Body>
.. </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4.33 SOAP message using XML encryption

```
 <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
   <SOAP-SEC:Signature
xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/"
        SOAP-ENV:actor="Some-URI"
        SOAP-ENV:mustUnderstand="1">
        <ds:Signature            Id="TestSignature"
              xmlns:ds="http://www.w3.org/2000
              /02/xmld sig#">

        <ds:SignedInfo>
<ds:CanonicalizationMethod
                Algorithm="http://www.w3.org/TR/2000/CR-
                xml-c14n-20001026">
             </ds:CanonicalizationMethod>
             <ds:SignatureMethod
```

```
            Algorithm=”http://www.w3.org/2000/09/xmldsig#hmac-
            sha1”/>

            <ds:Reference URI=”#Body”>

             <ds:Transforms>

              <ds:Transform  Algorithm=”http://www.w3.org/TR/2000/CR-
              xml-c14n-20001026”/>

             </ds:Transforms>

             <ds:DigestMethod
             Algorithm=”http://www.w3.org/2000/09/xmldsig#sha1”/>

               <ds:DigestValue>vAKDSiy987rplkju8ds:DigestValue>

              </ds:Reference>

          </ds:SignedInfo>

        <ds:SignatureValue>JHJH2374e<ds:SignatureValue>

      </ds:Signature>

      </SOAP-SEC:Signature>
```

Listing 4.34 SOAP message using XML digital signatures

SOAP Authorization

Using XML-based authorization in SOAP messages enables the authorization of the SOAP messages using certificates from the originating SOAP sender nodes.

SOAP authorization applies an XML-based digital certificate from an independent authorization authority to the SOAP message from the sender.

Listing 4.35 is a sample representation of a SOAP message using an XML-based authorization.

```
<SOAP-ENV:Envelope
```

```
 xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”>
<SOAP-ENV:Header>
  <SOAP-SEC:Authorization
    xmlns:SOAP-SEC=”http://schemas.xmlsoap.org/soap/security/”
    SOAP-ENV:actor=” actor-URI”
    SOAP-ENV:mustUnderstand=”1”>
    <AttributeCert xmlns=
     “http://schemas.xmlsoap.org/soap/security/AttributeCert”>
       An encoded certificate inserted here as
       encrypted using actor’s public key.
    </AttributeCert>
</SOAP-SEC:Authorization>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4.35 SOAP message using an XML-based authorization.

Developing web services using SOAP

- With the emergence of Web services, SOAP has become the *de facto* communication protocol standard for creating and invoking applications exposed over a network.
- SOAP is similar to traditional binary protocols like IIOP (CORBA) or JRMP (RMI), but instead of using a binary data representation, it adopts text-based data representation using XML.

- Using XML notation, SOAP defines a lightweight wire protocol and encoding format to represent data types, programming languages, and databases.
- SOAP can use a variety of Internet standard protocols (such as HTTP and SMTP) as its message transport, and it provides conventions for representing communication models like remote procedural calls (RPCs) and document-driven messaging.
- This enables inter application communication in a distributed environment and interoperability between heterogeneous applications over the networks.
- With its widespread acceptance by leading IT vendors and Web developers, SOAP is gaining popularity and adoption in most popular business applications for enabling them as Web services.
- It is important to note that SOAP is an ongoing W3C effort in which leading IT vendors are participating in order to come to a consensus on such important tasks associated with XML-based protocols and to define their key requirements and usage scenarios.

Building SOAP Web Services

We all are aware that SOAP provides an XML-based communication protocol solution for bridging disparate applications in a distributed environment using XML-based messaging or by remotely invoking methods.

From a Web services point of view, it defines and provides the following:

- A standardized way to transmit data using Internet-based protocols and a common-wire format (XML) between the Web service provider and its requestors.
- An extensible solution model using an XML-based framework enabling the Web service providers and requestors to interoperate with each other in a loosely coupled fashion and without knowing the underlying application architecture (such as programming languages and operating systems).
- This enables the creation of Web services over existing applications without modifying the underlying applications.

In a Web services implementation model, SOAP can be implemented as a client, as a server application, or both, as follows:

- A SOAP-based client application plays the role of a Web services requestor, which typically handles an XML-based request/response, a message containing a XML document, parameters required to invoke a remote method, or the calling of a SOAP server application.
- A SOAP client can be a Web server or a traditional application running a SOAP-based proxy, which send SOAP requests or SOAP messages using HTTP or any other supporting protocol.
- A SOAP server application plays the role of a Web services provider, which processes the SOAP requests and messages from calling SOAP-based clients.
- The SOAP server application interacts with its encapsulated applications to process the requests or

messages and then sends a response to the calling SOAP client.

- SOAP server applications also can act as SOAP intermediaries, which allows the extensibility of the application to enable the processing and forwarding of messages through a series of SOAP nodes or a final destination.
- In case of acting SOAP intermediaries, the SOAP server application typically works as a SOAP client application to the final destination of the message.

Developing SOAP Web Services Using Java

SOAP does not mandate a single programming model nor does it define programming language-specific bindings for its implementation.It is up to the provider to choose a language and to define the implementation of its language-specific bindings.In this context, to use Java as a language for developing SOAP applications requires its Java implementation for SOAP-specific bindings.

As of today, there are many SOAP application vendors that have made Java-based SOAP implementations for developing Web applications to Web services.

In general, the use of Java for developing SOAP applications enables scalable and portable applications to be built that also can interoperate with heterogeneous applications residing on different platforms by resolving the platform-specific incompatibilities and other issues.

Additionally, having SOAP-based applications that adopt a J2EE-based infrastructure and component framework allows the inheritance of the characteristics of J2EE

container-based services such as transactions, application security, and back-end application/databases connectivity.

The release of the Java Web Services Developer Pack (JWSDP) also provides a full-fledged API solution for developing SOAP-based Web services.

A long list of open source com-munities, Web services platform providers, and J2EE vendors also have released their SOAP implementations adopting Java platform and Java-based APIs.

To study and explore the features of a Java-based SOAP implementation, we chose to use Apache Axis, a Java-based toolkit from Apache Software foundation for developing SOAP-based Web services.

Axis also supports the JAX-RPC, JAXM, SAAJ, and SOAP 1.2 specifications in its forthcoming implementations. Axis follows its predecessor efforts of Apache SOAP.

Apache refers to Axis as the next generation of Apache SOAP implementation that provides a complete solution kit for Web services, which is more than sending and receiving SOAP messages.

The Axis toolkit is available for download at http://xml.apache.org/axis.

As a packaged solution, the Apache Axis environment provides the following:

- A SOAP-compliant runtime environment that can be used as a standalone SOAP server or as a plug-in component in a compliant Java servlet engine (such as Tomcat, iPlanet, and
- An API library and runtime environment for developing SOAP RPC and SOAP messaging-based applications and services
- A transport-independent means for adopting a variety of transport protocols (such as HTTP, SMTP, and FTP)
- Automatic serialization and deserialization for Java objects to and from XML in SOAP messages
- Support for exposing EJBs as Web services, especially the methods of stateless session EJBs
- Tools for creating WSDL from Java classes and vice-versa
- Tools for deploying, monitoring, and testing the Web services

Axis also provides full-fledged implementation support for Sun JWSDP 1.0 APIs, especially JAX-RPC and SAAJ.

Limitations of SOAP

Although the SOAP specifications define a promising communication model for Web services, the following limitations exist that are not currently addressed by the SOAP specifications:

- The specification does not address message reliability, secure message delivery, transactional support, and its communication requirements of a SOAP implementation.
- The specification does not address issues like object activation and object lifecycle management.
- The specification discusses HTTP as the primary transport protocol but does not discuss the usage of other transport protocols.
- The specification does not address how to handle SOAP messages out of a SOAP implementation.

Note that the limitations of SOAP have been currently well addressed by the ebXML framework as part of the ebXML messaging service, which complements SOAP and other Web services standards.

UNIT-IV
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. SOAP stands for [ ]
   A. Safe Object Access Protocol B. Simple Object Access Protocol
   C. Single Object Access Protocol D. Syntax Open Access Protocol

2. SOAP is [ ]
   A. Both platform and language independent protocol
   B. Only platform independent and not language independent protocol
   C. Only language independent and not platform independent protocol
   D. Neither platform independent nor language independent protocol

3. SOAP is based on XML [ ]
   A. TRUE B. False

4. ________and _______ technology are used to satisfy security requirement

5. Using _____________ we can satisfy requirement message authentication

6. SOAP message has ______ elements. [ ]
A. Header B. Body C. Envelope D. All of the above

7. Every SOAP message has a root _______________________ element.

8. SOAP is a ________. [ ]
A) Language B) Carrier C) Protocol D) Markup

9 .SOAP encoding supports both ______________________ type values

10. In a SOAP message, ______ are usually represented with an endpoint URI as the next destination in the message.

11 SOAP is a ________ to let applications exchange information over HTTP

   A) XML-based protocol B) JAVA-based protocol [ ]
   C) PHP-based protocol D) .NET-based protocol

12. SOAP is a format for sending messages and is also called as ___[ ]
   A) None of these B) Network protocol
   C) Data Transfer protocol D) Communication protocol

13. In a SOAP message, the SOAP ______ element is used to handle errors and to find out status information [ ]
A. Header B. Body C. Envelope D. Fault

14) ________element is the top most tag which identifies the XML document as a SOAP message. [ ]
A. Header B. Body C. Envelope D. Fault

15) What are the advantages of SOAP? [ ]

A) It is platform and programming language independent.
B) RPC (Remote procedure calls) are sometimes blocked by firewalls and proxy servers
C) Both A and B D) None of the above

16) The use of _________ for developing SOAP applications enables scalable and portable applications to be built that also can interoperate with heterogeneous applications residing on different platforms by resolving the platform-specific incompatibilities and other issues. [ ]
A) C++ B) JAVA C) DBMS D) C

17) __________ contains the XML data comprising the message being sent. It is a mandatory element [ ]

A. Header B. Body C. Envelope D. Fault

18) In a SOAP message exchange model, there can be zero or more _______between the SOAP
sender and receiver to provide a distributed processing mechanism for SOAP messages. [ ]

A) SOAP intermediaries B) SOAP sender C) SOAP Receiver D) None

SECTION-B

SUBJECTIVE QUESTIONS

1. Briefly explain the anatomy of SOAP message structure.

2. List and explain data types supported in SOAP encoding.

3. Write a short note on SOAP message exchange models.

4. Distinguish between RPC Communication and Message Communication in SOAP communication.

5.Outline the security issues in SOAP.

6. What is the role of SOAP in developing web services?

7. Outline the role of SOAP to build web services

8. Discuss the limitations of SOAP.

9. Compare and contrast the advantages of Web Services than Distributed Computing Technologies.

10. Analyze the syntax rules for SOAP message.
11. Identify the advantages of SOAP.
12. Outline the importance of different elements that are used in SOAP message format.
13. Explain how SOAP provides security.
14. Illustrate the important characteristics of a SOAP envelop element?
15. Mention what are the major obstacles faced by the users using SOAP?

WEB SERVICES

UNIT – V

Objective: To learn the fundamentals of SOAP.

Syllabus:

Discovering web services

Services discovery, role of service discovery in a SOA Service discovery mechanisms, UDDI Registries and their uses Programming with UDDI, UDDI data structures

Support for categorization in UDDI Registries Operations on UDDI Registry:

Publishing, Searching, Deleting information in a UDDI registry

Limitations of UDDI.

Learning Outcomes:

Students will be able to know the role of service discovery in a SOA. Identify UDDI Registries and their uses learn programming and data structures in UDDI identify operations on UDDI registry.

LEARNING MATERIAL

Discovering web services

Services discovery, role of service discovery in a SOA

In **SOA/distributed systems**, services need to find each other. for example, a web service might need to find a caching service or another mid-tier component service etc. A Service Discovery system should provide a mechanism for

- Service Registration
- Service Discovery
- Handling Fail over of service instances
- Load balancing across multiple instances of a Service
- Handling issues arising due to unreliable network.

Other aspects to consider when choosing or implementing a service registration/discovery solution

- Integrating service registration and discovery into the application or using a sidekick process
- Application Software stack compatibility with service discovery solution
- Handling Failure/outage of service discovery solution itself

Using a DNS

The simplest solution to registration and discovery is to just put all of your backends behind a single DNS name. To address a service, you contact it by DNS name and the request should get to a random backend. Details of how to use DNS for service registration and discovery can be found in the links below.

Advantages

- Easy to implement and has been there for a long time

Disadvantages

1. Service Instances have to poll for all changes – there's no way to push state. A monitoring component has to be implemented which will detect server failures or additions and propagates the state changes to the consumers.

2. DNS suffers from propagation delays; even after a server failure is detected a de-registration command issued to DNS, there will be at least a few seconds before this information gets to the consumers. Also, due to the various layers of caching in the DNS infrastructure, the exact propagation delay is often non-deterministic.

3. If a service is identified just by name, there's no way to determine which boxes get traffic. We will get the equivalent of random routing, with loads chaotically piling up behind some backends while others are left idle.

4. Most services use a front-end reverse proxy like nginx(to handle more connections, load balancing, ssl offloading, static file caching etc). These proxies cache the dns configuration unless you use some configuration file hack resulting in issues with detecting state. The same is true of HAProxy.

Introduction UDDI

UDDI technology is the core and one of the building blocks of Web services apart from SOAP and WSDL.UDDI enables the businesses providing services (in electronic form or in any other medium) to register information to enable the discovery of their services and business profile by prospective customers and/or partners.

Similarly, it enables businesses to discover other businesses for expanding potential business partnerships. Thus, UDDI presents businesses with an opportunity to step into new markets and services. It powers all kinds of businesses, large, medium, or small, to accelerate their business presence in this global market.UDDI initially started as a joint effort from IBM, Microsoft, and Arriba Since then, a number of companies joined the UDDI community. As of this book's writing, the UDDI project community is looking forward to releasing version 3.0 of the UDDI specification. This chapter covers version 2.0 of the UDDI specification because it is widely implemented and adopted as of this writing.To find more information on the UDDI effort, visit the UDDI official Web site **at www.uddi.org.**

Service discovery mechanisms

Manual

- o A human queries and decides Automatic
- o Discovery by a requester agent Centralized
- o UDDI registry: Centralized, authoritative repository of service descriptions

Decentralized

- o Distant ancestors of Whois++, rWhois systems
- o UDDI Federations
- o P2P systems

UDDI Registries and their uses

An implementation of the UDDI specification is termed as a *UDDI registry*.*UDDI registry services* are a set of software services that provide access to the UDDI registry.Meanwhile, registry services can perform a plethora of other activities such as authenticating and authorizing registry requests, logging registry requests, load-balancing requests, and so on.

Public and Private UDDI Registries

A UDDI registry can be operated in two modes: public mode and private mode.A *public UDDI registry* is available for everyone to publish/query the business and service information on the Internet. Such public registries can be a logical single system built upon multiple UDDI registry nodes that have their data synchronized through replication.Thus, all the UDDI registry node operators would each host a copy of the content and accessing any node would provide the same information and quality of service as any other operator node.Such global grouping of UDDI registry nodes is known as a *UDDI Business Registry,* or UBR.Content can be added into a UBR from any node, however, content can be modified or deleted only at a node at which it was inserted.A *private UDDI registry* is operated by a single organization or a group of collaborating organizations to share the information that would be avail-able only to the participating bodies.Private UDDI registries can impose additional security controls to protect the integrity of the registry data and to prevent access by unauthorized users.Note that a private node also can participate in information replication.

A UDDI registry in itself is a Web service. A Web service consumer queries the UDDI registry using the SOAP API defined by UDDI specification.Also, the UDDI specification publishes a WSDL description of the UDDI registry service.The UDDI project community members operate a UBR.This registry is available to everyone for free publishing/querying of businesses and ser-vices information.To find more information on this publicly operated UDDI registry, visit the UDDI Web site at www.uddi.org.

Interacting with a UDDI Registry

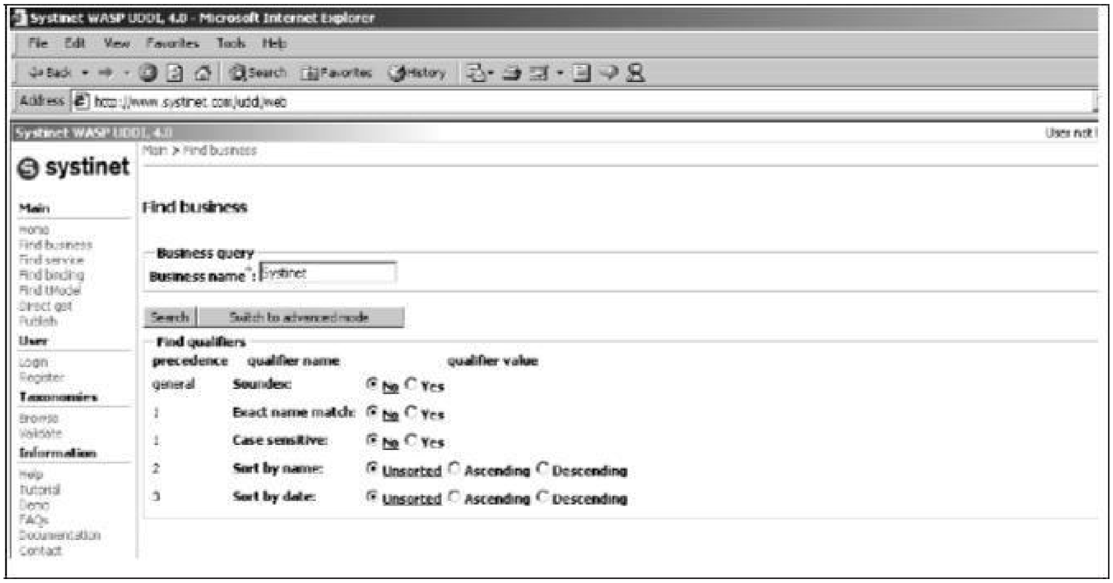
Typically, vendors implementing a UDDI registry provide two ways of interacting with a UDDI Registry Service.A graphical user interface

(GUI), for interacting with a UDDI registry. These GUIs also can be browser-based.The following is a list of public UDDI registries,

operated by various companies such as Microsoft, IBM, Hewlett Packard, and so on, that provide a browser-based interface to these registries:https://uddi.rte.microsoft.com/search/frames.aspx
https://www-3.ibm.com/services/uddi/v2beta/protect
/registry.html
https://uddi.hp.com/uddi/index.jsp
http://udditest.sap.com/
http://www.systinet.com/uddi/web

Below Figure shows a browser-based GUI provided by Systinet in order to interact with its publicly hosted UDDI registry.

This screenshot depicts the interface provided for searching businesses registered with the Sistine registry.



Find business

Business query
Business name*: Systinet

Find qualifiers

precedence	qualifier name	qualifier value
general	Soundex:	[✓ No] Yes
1	Exact name match:	[✓ No] Yes
1	Case sensitive:	[✓ No] Yes
2	Sort by name:	[✓ Unsorted] Ascending Descending
3	Sort by date:	[✓ Unsorted] Ascending Descending

Figure : Web-based GUI to UDDI registry.

Uses of UDDI Registry

Businesses can use a UDDI registry at three levels:

White pages level.

Businesses that intend to register just the very basic information about their company, such as company name, address, contact information, unique identifiers such as D-U-N-S numbers or Tax IDs, or Web services use UDDI as white pages.

Yellow pages level.

Businesses that intend to classify their informa-tion based on categorizations (also known as classification schemes or taxonomies) make use of the UDDI registry as yellow pages.

Green pages level.

Businesses that publish the technical information describing the behavior and supported functions on their Web ser-vices make use of the UDDI registry as green pages.

Programming with UDDI

This section introduces the APIs used for communicating with a UDDI registry. Also, important data structures and categorization support of UDDI are discussed.

UDDI Programming API

The UDDI specification defines two XML-based programming APIs for communicating with the UDDI registry node: inquiry API and publishing API. The following sections describe each of these.

Inquiry API

The inquiry API consists of XML messages defined using a UDDI Schema, which can be used to locate information about a business, such as the services a business offers and the technical specification of those services (such as a link to a WSDL document describing the interface of the service, the binding of the service and the URL where the service is running, and so on).

A UDDI programmer would use these inquiry APIs to retrieve information stored in the registry. To retrieve information, a registry user does not need to be authenticated.The following is a list of inquiry API functions that can be used for finding information in a UDDI registry:

<find_business>
<find_relatedBusinesses>
<find_service>
<find_binding>
<find_tModel>

To get further detailed information from the UDDI registry, the follow-ing inquiry API functions are available:

<get_businessDetail>
<get_businessDetailExt>
<get_serviceDetail>
<get_bindingDetail>
<get_tModelDetail>

Publishing API

The publishing API consists of functions represented by a UDDI Schema, which defines XML messages that can be used to create, update, and delete the information present in a UDDI registry.Note that in order to publish to a UDDI registry, the registry user needs to be authenticated.

The following is a list of publishing API functions that can be used for adding/modifying information to a UDDI registry:

<save_business>
<set_publisherAssertions>
<add_publisherAssertions>
<save_service>
<save_binding>
<save_tModel>

The following is a list of publishing API functions that can be used for deleting information from a UDDI registry:

<delete_business>
<delete_publisherAssertions>
<delete_service>

<delete_binding>
<delete_tModel>

Apart from the functions just mentioned, the publishing API also defines functions that deal with the authentication of the registry users, which is required in order to successfully execute the rest of the functions of this API:

<get_authToken>
<discard_authToken>

We will discuss each of the aforementioned APIs in detail in the sections titled *Inquiry API* and *Publishing API,* which follow.The XML messages constituting the UDDI programmer APIs are defined using a UDDI XML Schema.These XML messages are wrapped in a SOAP message and then sent to the UDDI registry.In other words, all of the XML messages are enveloped within a SOAP <body> element and then sent as an HTTP POST request to the UDDI registry.The UDDI registry then processes these SOAP messages and gets hold of the actual API function represented by the XML message, which further instructs the registry services to provide either publishing or querying services.A UDDI registry node typically enables access to both inquiry and publishing functionalities through different access point URLs. Table 5.3 lists the URLs for publicly operated UDDI registry nodes.

As we can see from Table 5.3, all of the URLs corresponding to the publishing access points support HTTPS, because publishing operations need authenticated access.

Table 5.3 Access Point URLs

OPERATO	INQUIRY URL	PUBLISHING URL
Microsoft	http://uddi.microsoft.con /inqui re	https://uddi.microsoft.con /publish
IBM	http://www-3.ibm.com/services /uddi/inquiryapi	https://www-3.ibm.com /services/uddi/protect /publishapi
HP	http://uddi.hp.com/inqui e	https://uddi.hp.com/ ublis h
SAP	http://udditest.sap.com/uddi https://udditest.sap.com /api/inquiry	/uddi/api/publish

Note that all the UDDI invocations follow a synchronous request/response mechanism and are stateless in nature.
This statelessness has a significant impact on the authentication of a registry user to the UDDI registry, which is required when performing a publishing operation on the registry.

Because of the stateless nature of the UDDI programming API, each time a registry user uses a publishing programming API, the security credentials of the identity associated with the registry user also are passed with each UDDI invocation.

UDDI data structures

The information managed by a UDDI registry is represented as XML data structures also known as UDDI data structures. The UDDI data structures specification document defines the meaning of these data structures and the relationship between them. Ultimately, it is these data structures with which a UDDI client needs to work.

A UDDI client makes use of these, in conjunction with the XML messages of programming APIs, to manipulate a specific type of information in a registry.

Similarly, response to a search operation received from the UDDI registry also would consist of these data structures.

Hence, the UDDI data structures are more or less input and out-put parameters for the UDDI programming API.

The following are the five primary UDDI data structures defined in the specification:

<businessEntity>
<publisherAssertion>
<businessService>
<bindingTemplate>
<tModel>

Note that all of these data structures except <publisher Assertion> are identified and referenced through a 128-bit globally unique identifier also known as UUID.These UUIDs can later be used as keys to access the specific data within the registry.Now, let’s take a look at each of these one by one.

<businessEntity>

The <businessEntity> data structure represents the primary information about a business, such as contact information, categorization of the business according to a specific taxonomy or classification scheme, identifiers, relationships to other business entities, and descriptions about that particular business.

<publisherAssertion>

A business registered in a UDDI registry can have active business relationships with other businesses. This relationship can be of any form, for example, a relationship of business partners or a business-to-customer relationship. Such relationships are represented by a <publisherAssertion> data structure in a UDDI Registry. The <publisherAssertion> structure is used to establish a relationship between two <businessEntity> structures.

A very interesting aspect about relationships in a UDDI registry is its ability to not make the relationship visible to the public unless and until both of the parties establishing this association assert for the same. This means that if a <businessEntity> structure representing Company A asserts its relationship with a <businessEntity> structure representing Company B through a <publisherAssertion> structure, a UDDI registry

would not make this relationship public until Company B has created another similar <publisherAssertion> structure. This provision is supported by the UDDI registries in order to ensure that a company can claim a business relationship with another company, only if the other partner also asserts for the same relationship.

<businessService>

The <businessService> data structure represents the service of a busi-ness. These services can be Web services or any other type of service. For example, the <businessService> data structure may represent a service that is offered over the telephone, such as a telephone banking service. The <businessService> data structure is merely a logical representation of services that a business has to offer.

A <businessEntity> structure contains one or more <businessService> structures. The same <businessService> structure also can be used by multiple <businessEntity> structures. For example, if a business has two departments—say, manufacturing and sales— that are each published to a UDDI registry as a <businessEntity> structure, then both of them can use the same <businessService> structure representing another business service—say, legal counseling.

<bindingTemplate>

The <bindingTemplate> structure consists of pointers to technical descriptions and access URLs of the service. Each <businessService> structure can contain one or more <bindingTemplate> structures. So, for example, if the <businessService> structure represents a Web ser-vice, then its <bindingTemplate> would refer to a PDF document pro-viding the technical description of this Web service and the URL at which the Web service can be accessed. Also, the <bindingTemplate> structure can provide an optional description of the Web service.Note that the <bindingTemplate> structure does not provide the details of the service specification, such as the interface of a service. That information is provided by the <tModel> structures, and <bindingTemplate> simply refers to one or more of such <tModel> structures.

<tModel>

The <tModel> structure provides a description of a particular specification or behavior of the service. The <tModel> structure does not contain the service specification directly; instead, it contains a link to the service specification, which is managed elsewhere. The <tModel> thus defines the interaction pattern in order to use the service. For example, a

business may provide a Web service whose WSDL interface may be referenced through a link from within the <tModel> structure.

Thus, <tModel> defines the lowest-level and most concrete piece of information about the services offered by a business. A UDDI client typically gets hold of the service specification pointed out by the <tModel> structure in order to use a publicly available Web service registered by a particular business.The linking between these five core data structures of UDDI is depicted in Figure.

Apart from these five primary data structures, two other structures exist that represent the category and identification information of the primary data structures: <identifierBag> and <categoryBag>. Let’s take a look at each of them now.

<identifierBag>

The <identifierBag> structure enables <businessEntity> or <tModel> structures to include information about the common forms of identification such as D-U-N-S numbers and tax IDs. This data can be used to signify the identity of <businessEntity>, or it can be used to signify the identity of the publishing party. Including such identification information is optional. However, when a published <businessEntity> or <tModel> carries such common forms of identification, it greatly enhances the search behaviors exposed via inquiry API functions.

<categoryBag>

The <categoryBag> structure enables <businessEntity>, <businessService>, and <tModel> structures to be categorized according to any categorization system, such as an industry categorization system or a geography categorization system. Categorizing the data structures mentioned previously is optional.

However, when these data structures are published along with their categorization information, it greatly enhances the search behaviors exposed via inquiry API functions. The categorization support in a UDDI registry is discussed in the following section.

Support for categorization in UDDI Registries

Categorization—also known as classification in JAXR terminology—is considered to be the prominent functionality of any registry.Categorization enables the data to be classified with the help of various categorization systems (also known as taxonomies or classification schemes), such as an industry categorization system or a geography categorization system.For example, a business can be classified as being located in the United States with the help of a standard geography categorization system such as ISO-3166.

Categorizing data aids in searching for a particular piece of data.

For example, searching for a software organization whose name begins with the letter M is much easier when that data is categorized as being located in Redmond, Washington, than when it is not.

Searching by the letter M for an organization that does not have a geographical categorization returns a much broader set of results, thus making it much more difficult to discover the business in which one is interested.

Hence, categorization is especially useful in the *discovery* of information managed by a UDDI registry.

UDDI registries have built-in support for three industry standard categorization systems. Also, the registry specification enables support for an open-ended categorization system that can be used in specific ways by a UDDI registry node operator.

In UDDI, the categorization system is represented by a <tModel> structure. These <tModel> structures have a unique name across all the UDDI registry node operators; however, the <tModel> UUID may change between the node operators.

UDDI-Supported Categorization Systems

The UDDI supported categorization systems and their <tModel> names are shown in Table 5.4.

Checked and Unchecked Categorization System

UDDI version 2.0 included the capability of validating the categorization of a particular UDDI data structure. Depending upon whether an

organization chooses to use the validation service of a UDDI registry, one of the two types of categorization systems will be supported:

Checked categorization system.

Checked categorization systems are used when the publisher of a categorization system wishes to ensure that the categorization code values registered represent accurate and validated information.

The categorization code values represented by UDDI structure <categoryBag> would be checked for valid values during a <save_business>, <save_service>, or <save_tModel> API call.

Table 5.4. UDDI-Supported Categorization Systems and Their <tModel> Names

CATEGOR- ZATION SYSTEM	<TMODEL> NAME	DESCRIPTION
NAICS	ntis-gov:naics :1997	This is a standard industry ar services categorization system. NAIC abbreviates to the North American Industry Classification System. This system is the mos elaborate and comprehensive industry classification

		scheme defined so far. Further information on this categorization syste can be obtained from www.census.gov/epcd/www /naics.html.
UNSPSC	unspcs-org :unspsc:3-1	This standard industry ar services categorization syste abbreviates to the Universal Standard Produc and Services Classification. This was the fir such industry classification schem defined for electronic businesses. Further information on this categorization syste can be obtained from www.unspsc.org.
ISO 3166	iso-ch:3166 :1999	This is the standard geograph based categorization system. Further information can be found at

		www.din.de/gremien/nas/nabd/iso3166ma.
Operator Specific	uddi-org:general_keywords	This categorization system operator specific. This is an open-ended categorization system that is n pre-defined. As a result, it ca consist of any category entries that may b defined specifically for that UDI registry node.

UDDI version 2 also enables third parties registering new categorization systems to control the categorization validation process. In such case, the third party would implement a Web service, in the same manner as UDDI does, that exposes a single XML API function named <validate_values>.

Unchecked categorization system. Unchecked categorization systems are used for categorization without the need for a UDDI to perform validation of categorization code values. Businesses can choose to make their categorization system available for categorization as an unchecked categorization system. Registering a new <tModel> structure and categorizing that <tModel> as a categorization system would register it as an unchecked categorization system.

Now, let’s take a look at the available programming APIs for searching information in a UDDI registry.

Unchecked categorization system.

Unchecked categorization systems are used for categorization without the need for a UDDI to perform validation of categorization code values. Businesses can choose to make their categorization system available for categorization as an unchecked categorization system. Registering a new <tModel> structure and categorizing that <tModel> as a categorization system would register it as an unchecked categorization system.Now, let’s take a look at the available programming APIs for searching information in a UDDI registry.

Inquiry API

This section will cover all of the XML messages that perform the functionality of inquiring certain information from a UDDI registry. Inquiry API constitutes of two types of functions:Functions that return zero or more homogeneous data structures containing abbreviated informationFunctions that return zero or more homogeneous data structures containing detailed information

Operations on UDDI Registry:

Publishing

SubmitBusiness.java shows us how to publish a business named *ACME Computer Services* along with its description. In the coming sections, we will examine the source code of SubmitBusiness.java, followed by its compilation and execution.

Programming Steps for Publishing

The entire publishing logic is provided by the doSubmit() method of the jws.ch5.SubmitBusiness class, and hence, its implementation is of most interest to us. The following are the steps of doSubmit(): Construct the UDDIApiPublishing object. This is the object that we will use to actually publish to the registry. Get hold of the authentication token from the registry with the help of the get_authToken() API call on the UDDIApiPublishing object. Once we have the authentication token, we should be able to publish to the registry.

- Create the BusinessEntity structure and populate it with the name and description of the business to submit. Note that we do not have to create the key for this business because the registry, upon submitting the business information, would generate it.

- Now, get hold of the SaveBusiness object. This object represents a collection of businesses that we wish to submit at a time. Hence, we will need to add the BusinessEntity object that we just created to the SaveBusiness object using the addBusinessEntity() method.

- Now, publish the business information through a save_business() call on UDDIApiPublishing object. This method call takes the SaveBusiness object as an argument and returns the BusinessDetail object upon completion.

- After the publishing operation has been executed, discard the authentication token. Finally, check whether the publishing operation was successful or not.

Searching

SearchBusiness.java shows us how to search for businesses based on the name pattern provided by the user. In the coming sections, we will examine the source code of SearchBusiness.java, followed by its compilation and execution.

Programming Steps for Searching

The entire querying logic is provided by the doSearch() method of the jws.ch5.SearchBusiness class, and hence, its implementation is of most interest to us. The following are the steps to implementing a doSearch():

1. Construct the FindBusiness object. This object represents the criteria for the search operation. Hence, we will need to add our criteria, that is, the name pattern that the user supplied, using the addName() method on this object.

2. Construct the UDDIApiInquiry object that we would use for placing the inquiry call.

3. Finally, invoke the business inquiry operation through the find _business() method on the UDDIApiInquiry object. This method returns a BusinessList object containing the BusinessInfo structures.
4. Now, check whether the businesses are found matching the given criteria. If there are matching businesses, we need to traverse through their BusinessInfo structures and get hold of the name and key UUID of the business.

Deleting

5. DeleteBusiness.java demonstrates how to delete a business from the UDDI registry based on its key UUID, which is passed by the user as a command line argument.

6. You can get hold of the business key either by browsing the Systinet registry on the Web or by executing SearchBusiness.

In the coming sections, we will examine the source code of DeleteBusiness .java, followed by its compilation and execution.

Programming Steps for Deleting

The deletion logic is provided by the doDelete() method of the jws.ch5.DeleteBusiness class, and hence, its implementation is of most interest to us. The following are the steps to implementing doDelete():

- Construct the UDDIApiPublishing object. This is the object that we would use to actually delete information from the registry.
- Get hold of the authentication token from the registry with the help of the get_authToken() API call on the UDDIApiPublishing object. Once we have a valid authentication token, we should be able to delete from the registry.
- Now, get hold of the DeleteBusiness object. This object represents a collection of businesses that we wish to delete at a time. Hence, we will need to add businesses referenced through BusinessKey to this object, using the addBusinessKey() method on DeleteBusiness.
- Now, delete the business information through the delete_business() call on the UDDIApiPublishing object. This method call takes the DeleteBusiness object as an argument and returns the DispositonReport object upon completion.
- Check the DispositionReport object to see if this operation was a success or a failure.

Limitations of UDDI.

UDDI is an evolving standard. Currently, the most deployed version of UDDI (2.0) is limiting in terms of the information model that it supports, especially when compared to other registry specifications such as ebXML Registry/Repository.

UDDI provides support for storing only the basic data structures, such as businesses, users, services, and service technical descriptions.

However, storing information about business Web services requires more than just the basic support.

For example, potential users of business Web services should be able to publish/query extensive business-oriented information, such as the business process models that a particular business Web service relies upon.

This is possible only if the target registry provides a data structure representing the business process model.

Also, UDDI is just a registry as opposed to ebXML Registry/Repository, which is, as the name suggests, a registry as well as repository.

The basic difference between a registry and repository is that a registry holds just the metadata of the objects submitted, whereas a repository actually stores the submitted objects.

UNIT-V
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. UDDI stands for [ ]
A- Uniform Description, Discovery, and Integration
B - Universal Description, Discovery, and Integration
C - Uniform Discovery, Description, and Integration
D - Uniform Discovery, Delivery, and Integration

2.The simplest solution to registration and discovery is ____________independent protocol

3. _______enables the businesses providing services in electronic form or in
any other medium to register information to enable the discovery of their
services and business profile by prospective customers and/or partners.
4. An implementation of the UDDI specification is termed as a ____________.

5. Businesses can use a UDDI registry at three levels. (TRUE/FALSE)

6.What are the five primary UDDI data structures defined in the specification:

7. The ________ data structure represents the primary information about a business, such as contact information, categorization of the business.

8. The _____ structure enables ___________ structures to be categorized according to any categorization system

9 . ______ are used when the publisher of a categorization system wishes to ensure that the categorization code values registered represent accurate and validated information

10_______ are used for categorization without the need for a UDDI to perform
validation of categorization code values.

11. UDDI enables a business to [ ]
A. describe its business and its services,
B. discover other businesses that offer desired services
C. integrate with these other businesses.
D. All of the above

12. A Service Discovery system should provide a mechanism for_________ [ ]
A. Service Registration and Discovery
- Handling Fail over of service instances
- Load balancing across multiple instances of a Service
- Handling issues arising due to unreliable network.

B. All of the above C. None of the above

13.Registry services can perform a plethora of other activities such as__ [ ]
A. Authenticating and authorizing registry requests
B. Logging registry requests
C. Both A & B D. None of the above.

14 .Businesses that intend to register just the very basic information about their company, such as company name, address, contact information, unique
identifiers such as D-U-N-S numbers or Tax IDs, or Web services use UDDI
as _____________. [ ]
A. White pages level. B. Yellow pages level.
C. Green pages level D. Red pages level.

15.Businesses that intend to classify their information based on categorizations (also

known as classification schemes or taxonomies) make use of the UDDI registry
as _________. [ ]

A. White pages level. C. Yellow pages level.
B. Green pages level D. Red pages level.

16. Businesses that publish the technical information describing the behavior and supported functions on their Web ser-vices make use of the UDDI registry as________. [ ]

A. White pages level. C. Yellow pages level.
B. Green pages level D. Red pages level.

17. The ____________ structure provides a description of a particular specification or behavior of the service. [ ]

A. <businessEntity> D. <publisherAssertion>
B. <businessService> E. <bindingTemplate>
C. <tModel>

18. Operations on UDDI Registry: [ ]

A. Publishing, Searching & Deleting
B .Publishing, Searching, Running & Deleting
C .Publishing, Finding & Running
D. All of the above

19. The ______________structure consists of pointers to technical descriptions
and access URLs of the service. [ ]

A. <businessEntity> C. <publisherAssertion>
B. <businessService> D. <bindingTemplate>

20. The basic difference between a registry and repository is that [ ]

A.A registry holds just the metadata of the objects submitted, whereas a repository actually stores the submitted objects.
B. A repository holds just the metadata of the objects submitted, whereas a registry actually stores the submitted objects.
C. Both A & B
D. None of the above

SECTION-B

SUBJECTIVE QUESTIONS:

1. Briefly explain the role of service discovery in SOA.
2. List and explain service discovery mechanisms.
3. Write a short note on UDDI Registries and their uses.
4. Distinguish between white, yellow and green pages.
5. Outline the UDDI data structures in UDDI
6. What is the Support for categorization in UDDI Registries
7. Outline the following operations on UDDI Registry:
    - Publishing.
    - Searching.
    - Deleting.
8. Discuss the Limitations of UDDI.
9. Demonstrate the Service discovery mechanisms in UDDI.
10. Apply the concept of UDDI Registries for registering Web services.
12. Analyze the operations that are used in UDDI Registry.
13. Mention what are the major obstacles faced by the users using UDDI.
14. Where can I find information about UDDI-related products and tools?
15. Who developed UDDI? And illustrate the purpose of UDDI.

WEB SERVICES

UNIT - VI: Web services Interoperability

Means of ensuring interoperability

One of the goals of Web services is to solve the interoperability problem by adopting industry standard protocols and data formats, which enable transparent application-to-application communication and data exchange between applications, systems, networks, and devices.

Although Web services promote interoperability, creating and testing interoperability between Web services becomes a real challenge when differences and limitations exist among implementations, especially because of application-specific dependencies and characteristics such as transport protocols, data types, XML processing, and compatibility.

In real-world scenarios involving business partner collaborations, the Web service provider needs to take particular care to define standard interoperability mechanisms and communication protocol for the partner applications, enabling them to build their own service clients.

This enables partner applications using different systems to easily interact with the Web service provider and conduct seamless transactions with them.

Interoperability in Web services becomes a real challenge when a service requestor finds problems while invoking a method in the service provider environment or when it does not understand a message sent by the service provider.

This is usually caused by prerequisites and factors exposed by the service provider or service requestor environments, and it is mostly caused by the dependencies of the underlying SOAP runtime provider implementation.

Thus, it becomes essential for Web services offered by a service provider to ensure that the services are usable by a variety of service requestor clients to the best possible accommodation of both con-forming and non-conforming SOAP implementations.

Different ways exist to ensure service requestor interoperability with the service providers.

The following are the ways to interoperability
- o Declaring W3C XML Schemas
- o Exposing WSDL
- o Creating SOAP Proxies
- o Testing Interoperability

Importance of .NET and J2EE.

Microsoft .NET Framework: An Overview

Microsoft .NET is part of the Microsoft .NET platform— Microsoft's strategy for developing distributed applications through XML Web services.

The Microsoft .NET Framework provides a full-fledged development environment for developing XML Web services in a Microsoft Windows–based environment.

It facilitates a runtime infrastructure and APIs for developing Web services applications using a variety of object-oriented programming languages such as C#, Visual Basic, and so forth.

The .NET Framework pro-vides the infrastructure for defining the overall .NET platform. Microsoft provides .NET compilers that generate a new code referred to as Microsoft

Intermediate Language (MSIL). MSIL is a CPU-independent code instruction, which is able to run on any system supporting its native machine language. The .NET compilers provided by Microsoft are as follows:

VB.NET (Visual Basic for .NET)

C++ .NET (Visual C++ for .NET)

ASP.NET (Microsoft ASP for .NET)

C# .NET (New language for .NET)

JScript (Jscript for .NET)

The Microsoft .NET Framework consists of two core components, which are described in the following sections.

Common Language Runtime (CLR)
The Common Language Runtime, or CLR, provides a managed runtime environment (.NET Engine) for the .NET Framework. CLR enables appli-cations to install and execute code, and it provides services such as mem-ory management, including garbage collection, threading, exception handling, deployment support, application runtime security, versioning, and so on.

CLR provides a set of JIT (just-in-time) compilers, which compile MSIL to produce native code specific to the target system. CLR defines a set of rules as Common ype System (CTS) and Common Language System (CLS) that specifies the .N T-supported languages required to use for developing compilers supporting a .NET platform. This enables the com-piler vendors to develop .NET-compliant compilers and to perform cross-language integration. Cross language integration enables .NET-compliant languages to run and interact with one another in a .NET environment.

.NET Framework Class Library
The .NET Framework class library acts as the base class library of the .NET Framework. It provides a collection of classes and a type system as foundation classes for .NET to facilitate CLR. It is included as part of the .NET Framework SDK. The class libraries are reusable object-oriented classes that support .NET programming tasks like establishing database connectivity, data collection, file access, and so on. The class libraries also support the rapid development of software applications such as the following:

o Console applications

o Windows GUI applications
0 Windows services

o ASP .NET applications

o .NET XML Web services

o .NET Scripting applications

o .NET Client applications

The .NET class libraries can work with any CLS-compliant language and can use CLR. At the time of this book's writing, the supported languages include Microsoft Visual Studio .NET, C#, and ASP.NET.

Web services security:-XML Security frame work, XML encryption

SOAP is simplistic: it allows for basic communication between services through structured data exchange, independent of language or platform. It depends upon underlying transport protocols (e.g., HTTPS and others) for its security. The initial SOAP specification focused primarily on extensibility and made security a second-class citizen.

One principle of Web services is to build on and extend what already exists and implement by merging existing technologies. SOAP is the foundation of the Web services infrastructure and is the focus of emerging security efforts. To create enterprise Web services, security considerations must extend past the SOAP specifications and go deeper into the underlying messaging approach. Many business transactions, such as establishing trust relationships and exchanging confidential information, require building additional security into SOAP.

Some of the XML specifications covered in this chapter are encryption, digital signatures, and key management services. Other standards-based organizations, such as OASIS and WS-I, are working on additional specifications related to SOAP security, including Extensible Access Control Markup Language (XACML) and Security Assertion Markup Language (SAML). The above standards provide the security foundation for SOAP and other XML-based messaging paradigms. This will become crucial for XML messages that pass through intermediaries.

Message Layer Security

Compared to transport-based security schemes such as SSL, incorporating security into the SOAP message provides several important advantages in Web services architecture. First, the interoperable nature of SOAP allows it to use a variety of transport protocols, including HTTP, SMTP, and others. In these scenarios, the message is transported from the originator through one-to-many intermediaries to the ultimate destination. When an intermediary receives a SOAP message, it processes entries contained in the header intended for itself and removes them before sending the message to the next destination.

Design

Privacy for Web services and the sensitivity of the messages over the public Internet could mandate the use of encryption in your architecture. *Encryption* is the act of taking data (usually referred to as clear text) and a short string (the key) and producing data (cipher text). The resulting cipher text is meaningless to a third party who does not know the key. Decryption is the inverse of encryption: taking cipher text and key and producing clear text.

Password encryption derives an encryption key from a user-supplied password. To make the task of discovering the key from the password more time-consuming, many implementations mix in a random variable, known as a *salt,* to create the key.

Several industry-standard algorithms can be chosen for a custom encryption mechanism. One of the elements to consider is whether the encrypted data needs to be decrypted once encrypted. Some algorithms are appropriate for one-way encryption, typically used in login and authentication scenarios. *One-way encryption* prevents the data from being reversed, which is important for credentials. *Two-way encryption* allows for reversible encryption.

A *key agreement* is a protocol in which two or more parties establish the same cryptographic keys without exchanging any secret information. Message authentication codes are a way for two parties to check the integrity of information stored in or transmitted over an unreliable medium and are based on a secret key. Typically, both parties have the same key, referred to as a *shared secret.*

Using Secure Sockets Layer to access your Web service will prevent "man-in-the-middle" attacks and stop data from being read or modified in transit. As previously mentioned, SSL does require additional processing overhead if done in software. This can be alleviated by using SSL accelerator network-interface cards.

XML Digital Signatures

XML Digital Signatures is a standard that allows for specifying the syntax and processing rules for attaching digital signatures to XML documents. An XML digital signature takes data objects, calculates a

digest (fixed-length representation of a variable-length stream), and places the result into the signature element.

The standard allows XML to functionally sign itself over an insecure network. XML signatures can be attached to any form of digital content, including XML (data objects). An XML signature can sign more than one type of resource, such as HTML, binary-encoded data (GIFs and JPEGs), or an XML-encoded section of an XML file.

Security Assertions Markup Language

Security Assertions Markup Language (SAML) is an XML-based framework used to exchange security information between business partners over the Internet. The driving force behind the creation of SAML is to enable interoperability between different security service providers. Prior to Web services, security was implemented primarily within a single organization. Now that organizations need to collaborate with business partners electronically, the ability to authenticate a user or service across organizations becomes imperative.

A Web services transaction started by one Web service can be completed at a different Web service and may require security information to be shared across all services involved in the transaction. SAML allows services to exchange authentication, authorization, and attribute information without organizations and their partners having to modify their current security solutions. SAML is designed to work with multiple industry-standard protocols such as HTTP and SMTP and integrates document-exchange protocols such as SOAP, BizTalk, and EbXML.

Extensible Access Control Markup Language

Extensible Access Control Markup Language (XACML) defines standardized security access control using XML to state authorization rules over a public connection. XACML also allows validation and revocation, based on defined authorization rules.

Key Management Specification

XML Key Management Specification (XKMS) is a standard that detail protocols for registration and distribution of public keys, so that the keys can be used in combination with XML digital signatures and encryption. XKMS was created to simplify the integration of digital certificates and public key infrastructure (PKI) with a multitude of applications. Applications that use this specification can easily integrate authentication, digital signature, and encryption services. XKMS includes support for certificate processing and revocation status checking.

Encryption Algorithm Selection

Algorithms can be generically categorized as either *symmetric* or *asymmetric.* A *symmetric-key algorithm,* better known as a shared secret, uses a single key for encryption and decryption. This is suitable when two parties have established a relationship in advance. bellow Table shows some of the algorithms that can be used in a symmetric scenario.

Table: Symmetric Algorithms

Algorithm	Length of key	Block size
AES	256	128
Blowfish	576	64
CAST-256	256	128
GOST	256	64
IDEA	128	64
RC-6	2040	128
Serpent	256	128
Twofish	256	128

Asymmetric algorithms are better known as *public/private-key.* This encryption is best used between two parties who have no prior knowledge of each other but want to exchange data securely. Unlike symmetric algorithms, asymmetric algorithms use two different cryptographic keys to encrypt and decrypt plain text. The two keys have a mathematical relationship. A message encrypted by the algorithm using one key can be decrypted by the same algorithm using the other key. Some asymmetric algorithms have the property that one key is deducible from the other. These algorithms are typically incorporated into public/private-key algorithms commonly used by certificate providers.

Encryption mechanisms typically use various algorithms for their routines. Listed below are some of the algorithms that can be used to develop your own encryption mechanism, along with their relative strengths and weaknesses. Many other encryption algorithms can be part of your toolkit but are not in widespread usage. If you want to go down this path, we recommend *Applied Cryptography,* by Bruce Schneier (Wiley, 1996).

Blowfish

Blowfish is a 64-bit block cipher algorithm. This essentially means that data is encrypted in 64-bit chunks. The Blowfish algorithm allows for varying key lengths, from 32 to 448 bits, and uses sixteen iterations of the main algorithm. The number of iterations is exponentially proportional to the time required to find a key using a brute-force attack. As the number of iterations increases, so does the algorithm's security.

SkipJack

SkipJack is 64-bit algorithm that transforms a 64-bit input block into a 64-bit output block. The transformation is parameterized by an 80-bit key and involves performing 32 iterations of a nonlinear complex function. In a key-based algorithm, the number of possible keys is directly related to the length of the key. Since SkipJack uses 80-bit keys, it means that there are 280, or more than one trillion trillion, possible keys.

Twofish

Twofish is a 128-bit block cipher algorithm. This essentially means that data is encrypted in 128-bit chunks. The Twofish algorithm allows for varying key lengths. It also uses sixteen iterations of its main algorithm, to ensure maximum security. This algorithm has been compromised with five iterations but never with sixteen. More
than sixteen iterations can be used, but the tradeoff in slower speed is not worth the higher security.

Triple DES

The DES algorithm was invented by IBM around 1970 and was initially designed with a key size of 128 bits. This algorithm has been successfully cracked by a group of Internet users (DESCHALL) using spare computer cycles. Based on current computer technology, this

algorithm can be cracked in anywhere from six hours to as little as three minutes. Triple DES uses the DES algorithm but encrypts data with DES three times, using three different keys. It is useful for securing low-security data, such as grade books or diaries.

MD5 and SHA1

A *digest,* such as MD5 or SHA1, takes an arbitrary-sized byte array and generates a fixed-size output, commonly referred to as a digest or hash. The fundamental requirements of a digest are that it should never reveal anything about the input used to generate it. While two different messages could potentially generate the same hash value, it should be computationally infeasible to do so. These algorithms are typically used for "fingerprinting" or digital signatures.

S/MIME

S/MIME is an emerging standard that uses a 40-bit symmetrical encryption for all messages. The message contains a digital signature the receiving party must receive before decrypting the message.

Ralph Merkle's Puzzle Protocol

A puzzle is a string that takes precisely a known amount of time to decrypt. For example, one way to create a puzzle is to encrypt a message with a symmetric cipher and a very short key of 20 bits.
Let us say that no better way exists for attacking the cipher than brute force. Therefore, anyone attempting to crack the puzzle will have to try every possible 20-bit key. Searching the entire key space will take 220 operations. The odds are good that the key will be discovered halfway through, so it is expected to take 219 operations.

Diffie-Hellman

The Diffie-Hellman key agreement protocol, developed in 1996, allows two users to exchange a secret key over an insecure medium without any prior secrets. This protocol depends on a discrete logarithmic problem for its security. It makes the assumption that it is computationally unfeasible to calculate a shared secret key, given two public values that are sufficiently large. This protocol has limitations, in that it does not validate either party.

DSA

The National Institute for Standards and Technology (NIST) published the Digital Signature Algorithm (DSA) as part of the government's Capstone project, which seeks to develop a standard for publicly available cryptography. The Capstone project used 80-bit symmetric keys.

DSA signature generation is significantly faster than signature verification and is therefore not an optimal algorithm compared to RSA. Typically, a message may be signed once but read many times. Therefore, it is advantageous to have faster signature verification.

RSA

The first known asymmetric algorithm was invented by Clifford Cocks but was not public. It was therefore reinvented by Ronald Rivest, Adi Shamir, and Leonard Adelman, (RSA) at MIT during the 1970s. RSA is a public-key cryptographic approach that allows for both encryption and digital signatures. The RSA algorithm relies for its security on factoring very large integers. Encryption and authentication occur without sharing private keys. Each party uses the other's public key or its own private key for operations. Any party can send an encrypted message and/or verify a signed message, but only the party that possesses the correct private key can decrypt or sign a message. RSA has certain weaknesses and is vulnerable to attack by factoring the modulus part of the public key.

Elliptic Curves

Elliptic-curve algorithms, created by Victor Miller and Neal Koblitz in the mid-1980s, are analogs of existing public-key approaches in which elliptic curves replace modular arithmetic operations. An elliptic curve is a mathematical construction from number theory and algebraic geometry and can be defined over any field.

Example: Asymmetric Puzzles

Let us look at the procedure for using Merkle's Puzzle to encrypt legal documents sent between Flute Bank's Loan Officer (Rodney) and a customer (Alicia):

Alicia creates a puzzle using the signed contract received from Rodney. Alicia encrypts her signed contract with a very long, randomly chosen key, using a symmetric algorithm such as Blowfish. Since the key is large, Alicia will not wait for Rodney to read the contract, because it will take a long time to be decrypted. Alicia sends the puzzle to Rodney and asks for a return receipt.

Alicia receives the return receipt, whereby Rodney asks for a "hint" for the puzzle. Rodney will use this hint to solve the puzzle instead of computing it himself. The hint does not reveal the contract (message).

Alicia in turn sends Rodney the first few bits of the key and asks for a return receipt for the hint.

When Alicia receives no additional requests for hints, this means that either the mathematical combinations have been reduced to a point where they are easy to calculate or the puzzle has been solved.

When Rodney sends Alicia a return receipt for the puzzle, she knows he possesses enough information to reconstruct the message. Every time she receives a request for a hint from him, she knows how much time is left until he can read the message. Also, because she does not give out hints until Rodney requests them by sending a return receipt, she knows how much of a hint he has. You may have noticed that asymmetric algorithms are slower than comparably secure symmetric algorithms—sometimes on the order of magnitude of one hundred times slower. Many cryptographic systems use a combination of both approaches, where a receiver's public key encrypts a symmetric-key algorithm used to encrypt a message. This uses the best of both worlds when properly done.
Organizations that operate outside the United States and Canada must be aware of national laws and export regulations. Many of the encryption algorithms that use large keys cannot be exported to certain foreign countries: Afghanistan, Cuba, Iran, Iraq, Libya, North Korea, Serbia, Sudan, Syria, and others. France also has its own unique laws in this regard. For further information on export rules, visit *www.bxa.doc.gov*.

Many open source activities, such as *www.openjce.org*, provide additional algorithms that are secure and do not have export restrictions. It is up to you to determine which of the listed algorithms fits your business needs and falls within legal guidelines.

XML digital signature

XML Signature (also called XMLDSig, XML-DSig, XML-Sig) defines an XML syntax for digital signatures and is defined in the W3C recommendation XML Signature Syntax and Processing.

Functionally, it has much in common with PKCS#7 but is more extensible and geared towards signing XML documents. It is used by various Web technologies such as SOAP, SAML, and others.

XML signatures can be used to sign data–a resource–of any type, typically XML documents, but anything that is accessible via a URL can be signed.

An XML signature used to sign a resource outside its containing XML document is called a detached signature; if it is used to sign some part of its containing document, it is called an

enveloped signature; if it contains the signed data within itself it is called an enveloping signature.

An XML Signature consists of a Signature element in the http://www.w3.org/2000/09/xmldsig# namespace. The basic structure is as follows:

```
<Signature>
<SignedInfo>
<CanonicalizationMethod />
<SignatureMethod />
<Reference>
<Transforms>
<DigestMethod>
<DigestValue>
</Reference>
<Reference />
</SignedInfo>
<SignatureValue />
<KeyInfo />
<Object />
</Signature>
```

The SignedInfo element contains or references the signed data and specifies what algorithms are used.

The SignatureMethod and CanonicalizationMethod elements are used by the SignatureValue element and are included in SignedInfo to protect them from tampering.One or more Reference elements specify the resource being signed by URI reference; and any transforms to be applied to the resource prior to signing. A transformation can be a XPath-expression that selects a defined subset of the document tree.

DigestMethod specifies the hash algorithm before applying the hash.

DigestValue contains the Base64 encoded result of applying the hash algorithm to the transformed resource(s) defined in the Reference element attributes.

The SignatureValue element contains the Base64 encoded signature result - the signature generated with the parameters specified in the SignatureMethod element - of the SignedInfo element
after applying the algorithm specified by the CanonicalizationMethod.

KeyInfo element optionally allows the signer to provide recipients with the key that validates the signature, usually in the form of one or more X.509 digital certificates. The relying party must identify the key from context if KeyInfo is not present.

The Object element (optional) contains the signed data if this is an enveloping signature. When validating an XML Signature, a procedure called Core Validation is followed.

Reference Validation: Each Reference's digest is verified by retrieving the corresponding resource and applying any transforms and then the specified digest method to it. The result is compared to the recorded DigestValue; if they do not match, validation fails.

Signature Validation: The SignedInfo element is serialized using the canonicalization method specified in CanonicalizationMethod, the key data is retrieved using KeyInfo or by other means, and the signature is verified using the method specified in SignatureMethod.
This procedure establishes whether the resources were really signed by the alleged party.

However, because of the extensibility of the canonicalization and transform methods, the verifying party must also make sure that what was actually signed or digested is really what was present in the original data, in other words, that

the algorithms used there can be trusted not to change the meaning of the signed data.

Because the signed document's structure can be tampered with leading to "signature wrapping" attacks, the validation process should also cover XML document structure.

Signed element and signature element should be selected using absolute XPath expression, not getElementByName methods

Benefits

XML Signature is more flexible than other forms of digital signatures such as Pretty Good Privacy and Cryptographic Message Syntax, because it does not operate on binary data, but on the XML Infoset, allowing to work on subsets of the data, having various ways to bind the signature and signed information, and perform transformations.

Another core concept is canonicalization, that is to sign only the "essence", eliminating meaningless differences like whitespace and line endings.

XML Digital Signatures

Consider a scenario where you are developing a healthcare Web service for a hospital that exposes medical records and other patient information. An insurance company or HMO would need to see the details of a lab test and the interactions between the doctor and patient. A doctor would need to see details not only of a patient's current stay at the hospital but also any past visits. The doctor should not need to know whether the patient has insurance deductibles. A nurse may need to see information related only to the current visit. A medical researcher may need to see medical history but not personal details. Many other combinations of information could, of course, be contained in an XML message.
The W3C and the Internet Engineering Task Force (IETF) have put together a joint proposal for using XML-based digital signatures. Java Community Process *(www.jcp.org)* is also working to define a specification for XML Digital Signatures (JSR-105) within Java. Digitally signing the entire XML document is simplistic. What if a document such as a medical record requires digital signatures on different portions by different medical personnel? Furthermore, signature in this scenario implies order. The admissions department should ideally sign its section before the discharge department does.

The primary use for digital signatures is for nonrepudiation. For example, Flute Bank can receive a signed document and know exactly who sent it, because the signature contains a message digest signed by using the sender's private key. XML signatures define a signature element that contains the information to process a digital signature. Each digital signature refers to one of three things:

An XML element in the signature element

An ext■rnal XML document referenced by a URI in the document

An ext■rnal non-XML resource referenced by a URI in the XML document

Consider the following XML document for placing a check order:

```
<flutebank:checkOrder
xmlns:flutebank="http://flutebank.com/CheckOr
<flutebank:checkType>flutebank:Dilbert</flutebank:checkType>
<flutebank:quantity>1,000</flutebank:quantity>
<flutebank:account>ABC123</flutebank:account>
<flutebank:startnum>2000</flutebank:startnum>

</flutebank:checkOrder>
```

If a customer of Flutebank.com sent this order to our Web service, the service will need to validate that the order truly came from the account holder, regardless of the specified account number. One approach would be for the customer to digitally sign the above XML document, as in Listing 15.1.

Listing 15.1: Signed check order

```
<flutebank:signedCheckOrder
xmlns:flutebank="http://flutebank.com/C <Signature
xmlns="http://www.w3.org/2000/09/xmldsig#">

<SignedInfo>

<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/

<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig
```

```
<Reference URI=""/>

<Transforms>

<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#e

</Transforms>


<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
<DigestValue>j6lbgp5EPmSfTb3atsSuNbeVu8nk=</DigestValue>

</Reference>

</SignedInfo>

<SignatureValue>aiYECAxGoPiL0v3sSamm3rXup5zJa ...
</SignatureVa <KeyInfo>
<X509Data>

<X509Certificate>MIIDa1sY+mAyIBA ... </X509Certificate>
</X509Data>
</KeyInfo>

</Signature>

<flutebank:checkOrder>

<flutebank:OrderID>64B4A0D1-814E-4FF6-918A-
DD7E7E1AECEA</fluteb
<flutebank:checkType>flutebank:Dilbert</flutebank:checkType>

<flutebank:quantity>1,000</flutebank:quantity>

<flutebank:account>ABC123</flutebank:account>

<flutebank:startnum>2000</flutebank:startnum>

</flutebank:checkOrder>

</flutebank:signedCheckOrder>
```

Here, the original checkOrder now becomes a child element of the root signedCheckOrders and adds an additional child, the signature element, which contains the digital signature information. Included it are elements that identify the algorithms used to canonicalize the data, the digest algorithm, and the signature algorithm.

The reference element points to the data we are interested in validating. In Listing 15.1, the element contains a null URI, which means that the entire XML document containing the signature element should be signed. This attribute could also include an XPointer reference, to state a specific portion of the document used to compute the signature.

The transforms element performs an XPath transform that removes the signature element. This is required in this scenario, because we are signing the entire document that envelops the signature. The DigestValue, calculated as part of the signing process, will obviously change if you include the digest itself. The transforms element removes the entire signature element from the data that must be digested and signed. This approach prevents recursion issues.

XPath is a language that describes a method to locate and process items within XML documents. The method uses an addressing syntax on a path through the document's logical structure or hierarchy.

The keyinfo element holds key information required for validation of the signature. In our scenario, it contains an X.509 certificate, which holds the public key for the account holder. This is how Flute Bank can validate that the check order originated from a specific account holder.

The final feature that is critical in this example is the OrderID element. It contains a unique identifier that could be based on a timestamp or other calculated value. We have provided an additional level of protection by tagging each order with a unique identifier, preventing the success of a replay attack. Without the unique identifier, an attacker could simply replay the signed message, causing hundreds of duplicate orders to be created. This could be used to drain the bank account. The Flute Bank Check Order Web service checks to see if it has already processed an order with this unique identifier and ignores any duplicates.

In the above example, the check order was sent directly from the client to Flute Bank. If the check order were routed through other intermediaries, changing the order identifier would cause the signature validation check to fail. Remember that a digest is calculated based on the contents of the message. Rearranging the order within the document will also cause the message digest value to change. Likewise, replaying the same order would be flagged and ignored. Detection of replay attacks can be part of the message or header and can use one or more of the following approaches: timestamps, sequence numbers, expiration message, and correlation.

The XML digital signature specification also incorporates the XML canonical specification for generating the physical document, for scenarios where two XML documents may differ in their exact textual representation but are logically equivalent. Otherwise, documents may become suspect without valid cause. Both digital signature generation and validation are typically done using message digests. For a signature, the digest is calculated on the XML's canonical form. If the digests between two canonical forms of XML match, you can be sure the document has not been tampered with, even though the textual forms may vary.

A message digest takes an input message of arbitrary length and produces a fixed-length output.

If you decide to use digital signatures as part of your Web service, you must become aware of the security implications involved. Digital signatures require public and private key pairs, and the validity of the public key has to be provided by other technologies. A certificate trust model is required—either peer-to-peer or hierarchical. A method to generate and maintain trusted key pairs and certificates is also necessary. Finally, it must be possible to validate that the certificate has not been revoked.

For additional information on XML digital signatures, please visit

www.w3.org/TR/2002/REC-xmldsig-core-20020212.

Many implementations of XML digital signatures are available, including:

IAIK XML Signature Library:
http://jcewww.iaik.tu-graz.ac.at/products/ixsil/index.php

IBM's XML Security Suite:
www.alphaworks.ibm.com/tech/xmlsecuritysuite

InfoMosaic SecureXML: *www.infomosiac.net*

NEC XML Digital Signature Software Library:

www.sw.nec.co/jp/soft/xml_s/appform_e.html

Phaos XML: *www.phaos.com/e_security/dl_xml.html*

RSA BSAFE: *www.rsasecurity.com/products/bsafe/certj.html*

Verisign XML Signature SDK:

www.xmltrustcenter.org/xmlsig/developer/verisign/index.htm

Providing digital signatures as part of a Web services architecture increases the ability to support nonrepudiation and signer authentication aspects of your services.

XML Encryption

The W3C and IETF are working on an XML encryption specification. Java Community Process (*www.jcp.org*) is working to define a standard *Java API for XML Encryption* (JSR-106). In our discussion above, we outlined how to use SSL for encryption of messages over the transport. SSL addresses the needs to secure the document over the transport but does not handle the security requirements of a document once it is persisted. It also does not address when different parts of a document require different levels of protection.

The core element in XML encryption is the EncryptedData element, used with the EncryptedKey element to transport encryption keys from the originator to a known recipient. It derives from the EncryptedType abstract type. When an element is encrypted, the EncryptedData element replaces the element in the encrypted version of the XML document. The easiest way to demonstrate XML encryption is by showing unencrypted and encrypted XML documents, as in Listings 15.4 and 15.5.

Listing 15.4: Unencrypted XML document

```
<?xml version='1.0'?>

<InsuranceInfo

xmlns='http://insurance.org/HMOv2'>
<Name>Sylvester James</Name> <Employer>Planet
Fruit</Employer>
<IDCard                Deductible='5,000'

Currency='TT'> <Number>XJABAC
34534</Number> <Issuer>Dispute
Insurance</Issuer>
<Expiration>04/02</Expiration>
</IDCard>

<Insured>Soogia

Rattan</Insured>
</InsuranceInfo>
```

Listing 15.5: Encrypted XML document

```
<?xml version='1.0'?>

<InsuranceInfo

xmlns='http://insurance.org/HMOv2'>
<Name>Sylvester James</Name> <Employer>Planet
Fruit</Employer>
<EncryptedData

Type='http://www.w3.org/2001/04/xmlenc#Element'
xmlns='http://www.w3.org/2001/04/xmlenc#'>
```

```
<CipherData><CipherValue>A1B2C3D4E5F</CipherValue></Ci pherD
</EncryptedData>

<Insured>Soogia Rattan</Insured>

</InsuranceInfo>
```

In the above example, we have hidden the currency amounts. Sometimes it is necessary to encrypt the entire document (Listing 15.6).

Listing 15.6: Completely encrypted XML document

```
<?xml version='1.0'?>

<EncryptedData

xmlns='http://www.w3.org/2001/04/xmlenc#'
Type='http://www.isi.edu/in-
notes/iana/assignments/media-types/
<CipherData><CipherValue>B1CH3TR1N1</CipherValue><</Ci
pherData>

</EncryptedData>
```

Business messages must ensure authenticity (Who was the sender?), data privacy (Was it modified in transit?) and nonrepudiation (Can the sender deny sending it?) So far, we have discussed using Web server authentication for knowing the sender. We have considered use of SSL to secure the transport. We have thought about using either custom or XML encryption to make sure documents cannot be modified by intermediate parties.

In our solution, we have not considered whether we need to support non-repudiation. Flute bank needs to ensure that transactions sent to Flute Bank

services come from authorized customers who cannot deny having sent transactions. This can be handled using multiple techniques.

Public key infrastructure (PKI) describes the processes, policies, and standardgovern the issuance and revocation of the certificates and public and private keys that encryption and signing operations use. Public-key cryptography allows Web services to exchange data across an insecure network such as the Internet with the assurance that messages will neither be modified nor inappropriately accessed.

The basis premise is that data is transformed according to an algorithm parameterized by a pair of keys (public and private). Each participant in the exchange has such a pair of keys. Users make their public key freely available to anyone wishing to establish a dialog with their services and keep the other key private and appropriately secured. Information encoded using one key can be decoded using the other. If someone else intercepts a message, it would remain unusable, because the private key is needed decrypt the message.

The advantages of using public-key infrastructure over other approaches based on shared secrets are not only scalability, in not having to distribute the secret to unknown parties in advance, but also the ability to validate the sender, because everyone has a unique private key.

To digitally sign a message, an algorithm that accepts the sender's private key transforms the data in the message. The detransformation can occur in reverse only if it uses the sender's public key, This

assures the recipient of the message's true origin. If the data can be confirmed using the sender's public key, it must have been signed using the corresponding private key, to which only the sender has access. A certificate authority provides an assertion of a public key's validity and asserts that it actually does belong to the sender. Otherwise, an impostor could arbitrarily create or steal others' certificates, presenting himself as the owner.

Typically, public-key approaches do not encode an entire message. Instead, they create a small, unique thumbprint of the document, typically referred to as a digest or hash. Hashing algorithms are aware of changes to a source message and therefore provide a way for the recipient to validate that the message has not been altered. The digest is transformed using the sender's private key, creating a digital signature. This also allows the recipient to verify that the sender performed the transformation.

Listed below are some of the implementations of XML encryption:

Phaos XML: www.phaos.com/e_security/prod_xml.html

Trust Services Integration Kit:

www.xmltrustcenter.org/developer/verisign/tsik/indes.htm

XML Security Library: *www.aleksey.com/xmlsec*

XML Security Suite: *www.alphaworks.ibm.com/tech/xmlsecuritysuite*

XKMS structure

XML Key Management Specification (XKMS) uses the web services framework to make it easier for developers to secure inter-application communication using public key infrastructure (PKI).

XML Key Management Specification is a protocol developed by W3C which describes the distribution and registration of public keys.

Services can access an XKMS compliant server in order to receive updated key information for encryption and authentication.

XKMS consists of two parts:

X-KISS

XML Key Information Service Specification

X-KRSS

XML Key Registration Service Specification

The X-KRSS defines the protocols needed to register public key information. X-KRSS can generate the key material, making key recovery easier than when created manually.

X-KISS

The X-KISS outlines the syntax that applications should use to delegate some or all of the tasks needed to process the key information element of an XML signature to a trust service.

In both cases the goal of XKMS is to allow all the complexity of traditional PKI implementations to be offloaded from the client to an external service.

While this approach was originally suggested by Diffie and Hellman in their New Directions paper this was generally considered impractical at the time leading to commercial development focusing on the certificate based approach proposed by Loren Kohnfelder.

X-KISS allows a client to delegate part or all of the tasks required to process XML Signature <ds:KeyInfo> elements to a *Trust service*.

A key objective of the protocol design is to minimize the complexity of applications using XML Signature.

By becoming a client of the trust service, the application is relieved of the complexity and syntax of the underlying PKI used to establish trust relationships, which may be based upon a different specification such as X.509/PKIX, SPKI or PGP.

By design, the XML Signature Specification does not mandate use of a particular trust policy. The signer of a document is not required to include any key information but may include a <ds:KeyInfo> element that specifies the key itself, a key name, X.509 certificate, a PGP Key Identifier etc. Alternatively, a link may be provided to a location where the full <ds:KeyInfo> information may be found.

The information provided by the signer may therefore be insufficient by itself to perform cryptographic verification and decide whether to trust the signing key, or the information may not be in a format the client can use. For example:

The Key may be specified by a name only.

The local trust policy of the client may require additional information in order to trust the key.The Key may be encoded in an X.509 certificate that the client cannot parse.In the case of an encryption operation:

The client may not know the public key of the recipient.

X-KRSS

X-KRSS describes a protocol for registration of public key information. A client of a conforming service may request that the Registration Service bind information to a public key. The information bound may include a name, an identifier or extended attributes defined by the implementation.

The key pair to which the information is bound may be generated in advance by the client or, to support key recovery, may be generated on request by the service. The Registration protocol may also be used for subsequent recovery of a private key

The protocol provides for authentication of the applicant and, in the case that the key pair is generated by the client, Proof of Possession (POP) of the private key. A means of communicating the private key to the client is provided in the case that the private key is generated by the Registration Service.

Guidelines for signing XML documents.

Guidelines for Signing XML Documents

Rules for digitally signing XML.

Just as in nonelectronic life, a user should only sign what is seen.

Because XML relies on transformations and substitutions during the processing of an XML document, special care needs to be taken when working with the XML Security Framework.

For instance, if an XML document includes an embedded style sheet (such as when XSLT is used), it is the transformed document that should be represented to the user and signed rather than the document without the style sheet.

In addition, when a document references an external style sheet, the content of that external style sheet should also be signed.

Content presentation may introduce changes.

If signing is intended to convey the judgment of a user about document content, then it is important that what gets signed is the information that was presented to that user.

However, when content is presented on a screen or viewed in a printout based on some XML source, the signer must be careful to sign not only the original XML but also any style sheets or other information that may affect the presentation.

Transformations may alter content.

Some applications might operate with the original or intermediary data, but a signer should be careful about potential weaknesses introduced between the original and transformed data.

This is a trust decision about the character and meaning of the transforms that an application needs to make.

Consider a canonicalization algorithm that normalizes character case (lower to upper) or character composition ("e and accent " to " accented -e").

An adversary could introduce changes that are normalized and thus inconsequential to signature validity but material to a Document Object Model processor.

For instance, by changing the case of a character one might influence the result of an XPath selection, introducing a serious risk if that change is normalized for signature validation but the XML processor operating over the original data returns a different result than intended.

Care should be taken that all documents associated with a core XML document be part of the signature process.

Similarly, care must be taken by applications executing algorithms specified in an XML signature when additional information is supplied as parameters such as XSLT transforms.

The algorithms specified in the document will often be implemented via a trusted library, yet perverse parameters might cause unacceptable processing or memory demand.

As in any security infrastructure, the security of an overall system will depend on the security and integrity of procedures and personnel as well as procedural enforcement.