Unit-1

SHORT ANSWER QUESTIONS:

1. **Discuss the importance of Distributed Computing:-**

The distributed computing environment provides many significant advantages compared to a traditional standalone application. The following are

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

2. What are the Challenges in Distributed Computing?

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.

3. What Are Web Services

Web services are based on the concept of service-oriented architecture (SOA). SOA is the latest evolution of distributed computing, which enables software components, including application functions, objects, and processes from different systems, to be exposed as services.

Web services are loosely coupled software components delivered over Internet standard technologies." In short, Web services are self-describing and modular business applications that expose the business logic as services over the Internet

through programmable interfaces and using Internet protocols for the purpose of providing ways to find, subscribe, and invoke those services. Based on XML standards, Web services can be developed as loosely coupled application components using any programming language, any protocol, or any platform. This facilitates delivering business applications as a service accessible to anyone, anytime, at any location, and using any platform.Consider the simple example shown in Figure 2.1 where a travel reservation services provider exposes its business pplications as Web services supporting a variety of customers and application clients. These business applications are provided by different travel organizations residing at different networks and geographical locations.

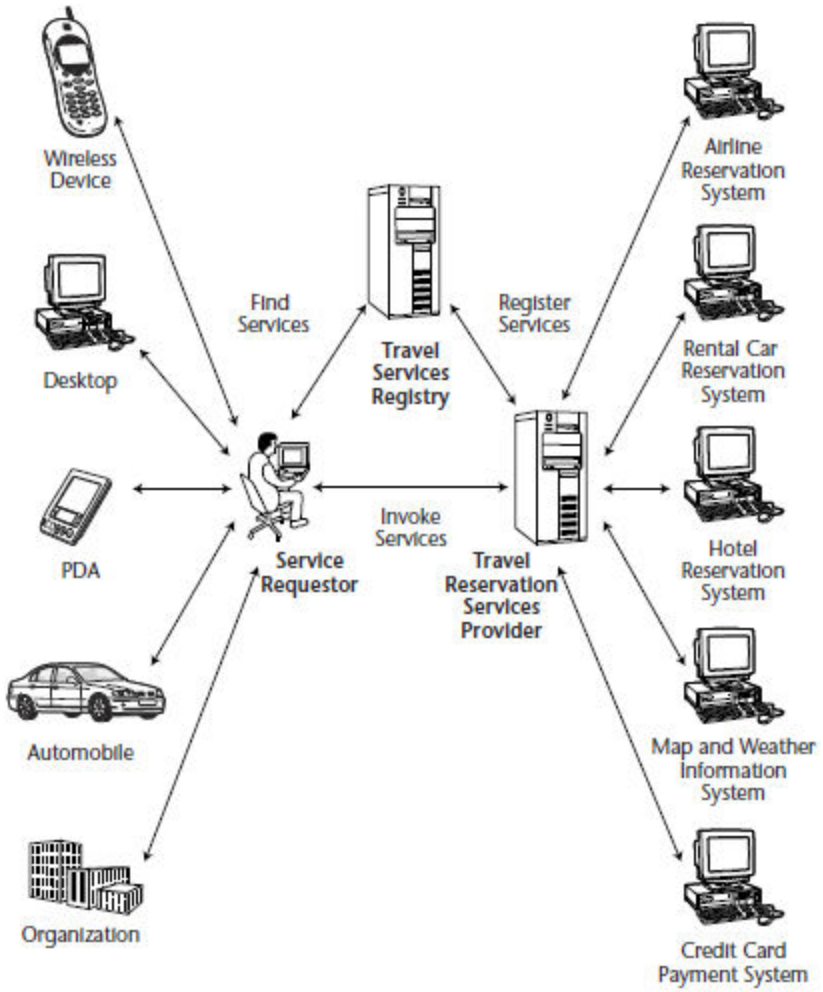


Figure 2.1 An example scenario of Web services.

4. **Discuss the characteristics of a Web services application model?**

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.

5. **What are the design requirements of the Web services architecture?**

To provide a universal interface and a consistent solution model to define the application as modular components, thus enabling them as exposable services

■ To define a framework with a standards-based infrastructure model and protocols to support services-based applications over the Internet

■ To address a variety of service delivery scenarios ranging from e-business (B2C), business-to-business (B2B), peer-to-peer (P2P), and enterprise application integration (EAI)-based application communication

■ To enable distributable modular applications as a centralized and decentralized application environment that supports boundary-less application communication for inter-enterprise and intra-enterprise application connectivity

■ To enable the publishing of services to one or more public or private directories, thus enabling potential users to locate the published services using standard-based mechanisms that are defined by standards organizations

■ To enable the invocation of those services when it is required, subject to authentication, authorization, and other security measures

LONG ANSWER QUESTIONS

1.Briefly discuss core distributed computing technologies?

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.

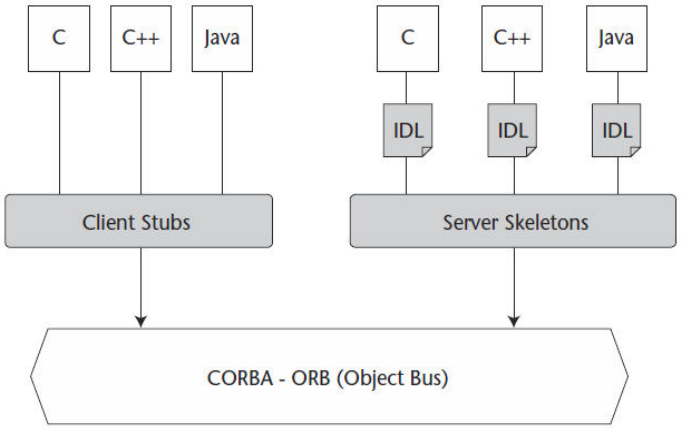


Figure 1.3 An example of the CORBA architectural model.

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 componentoriented applications

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 nonprofit 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 ystem, or hardware platform. By adopting CORBA, the applications can reside and run on any hardwarelatform 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 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 ervices, 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 remoteobject 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, delegates invocation requests to the RMI skeleton, and finally passes the return values of the method execution to the client stub. It also enables callbacks from server objects to client applications so that the asynchronous notifications can be achieved. 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 orserializable 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 unmarshals 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 fomal 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 (unmarshalling) 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 incomingremote 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 RMIinfrastructure.

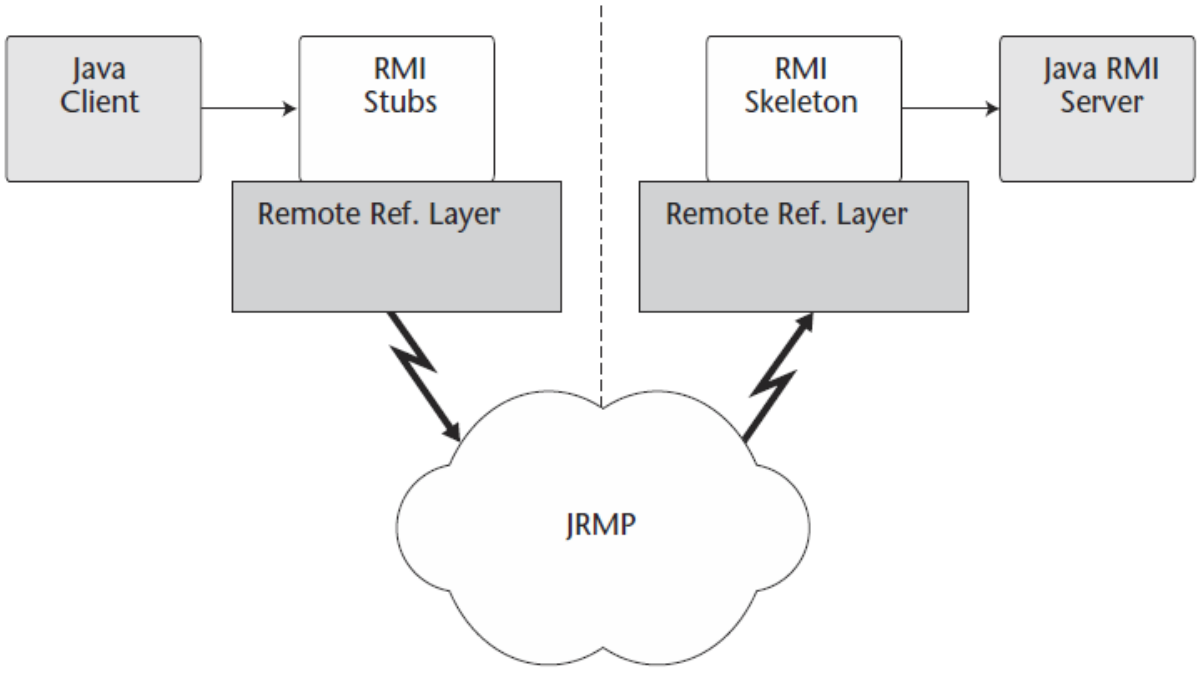


Figure 1.4 A Java RMI architectural model.

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), whichemployed 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

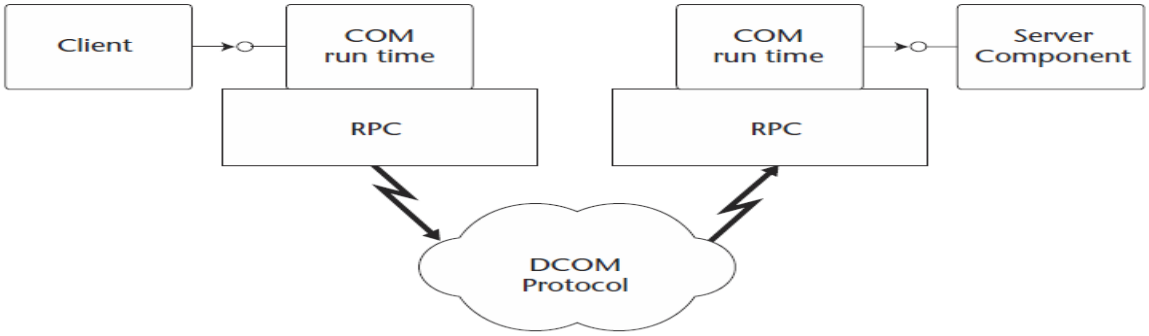


Figure 1.5 Basic architectural model of Microsoft DCOM.

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 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 not

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

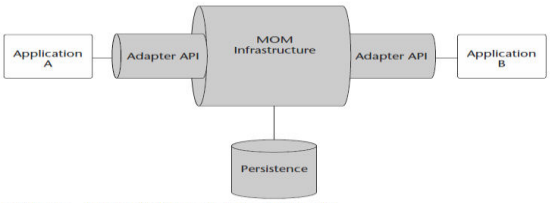


Figure 1.6 A typical MOM-based architectural model.

2. Discuss 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 (HyperText Transfer Protocol). HTTP is a stateless protocol that connects Web browsers to Web servers, enabling the transportation of HTML content to the user. With the high popularity and potential of this infrastructure, the push for a more dynamic technology was inevitable. 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 lead 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 shows various components of the J2EE architecture.

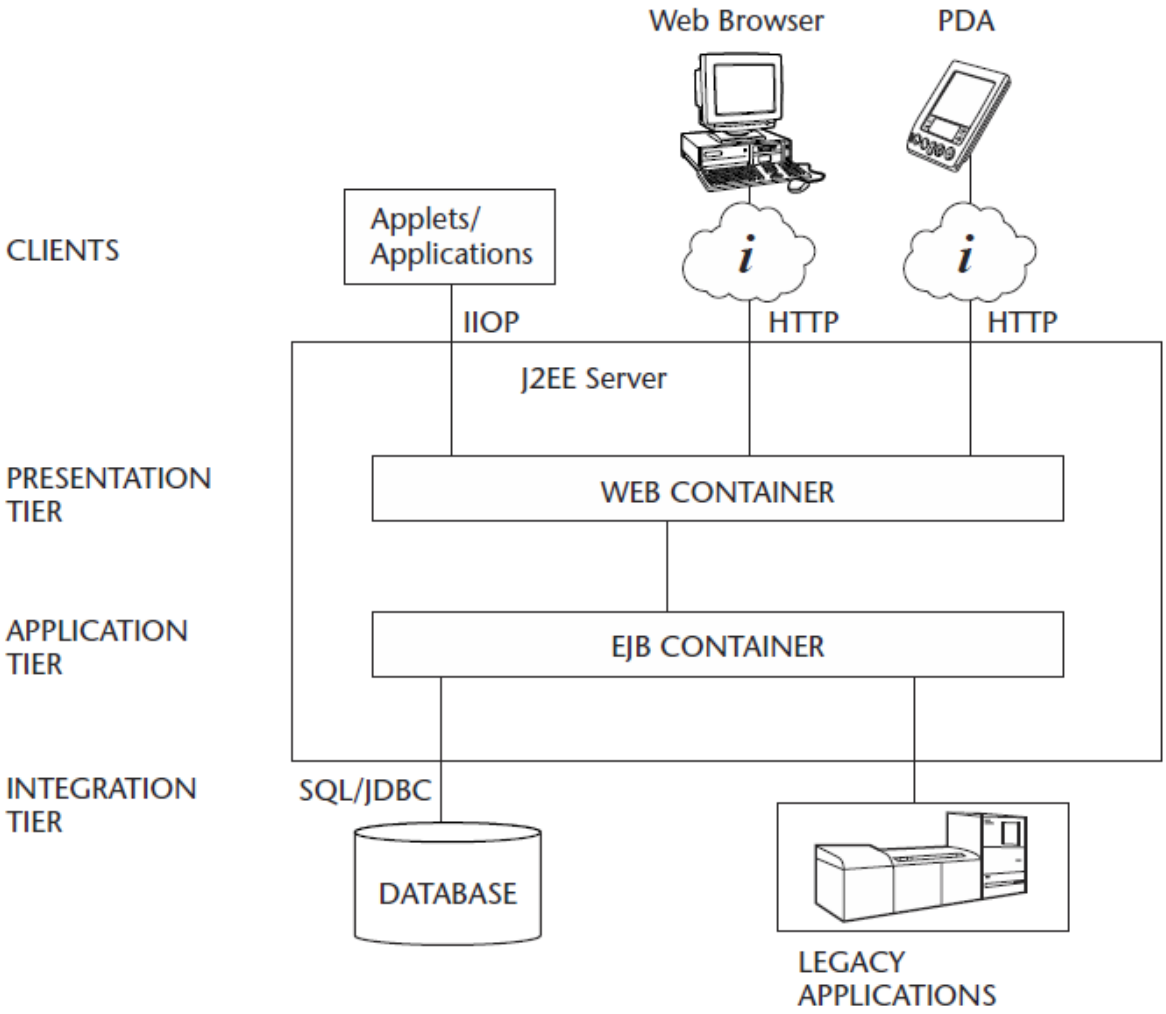


Figure 1.7 J2EE application architecture.

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.

3.Discuss the steps involved in implementing a web service?

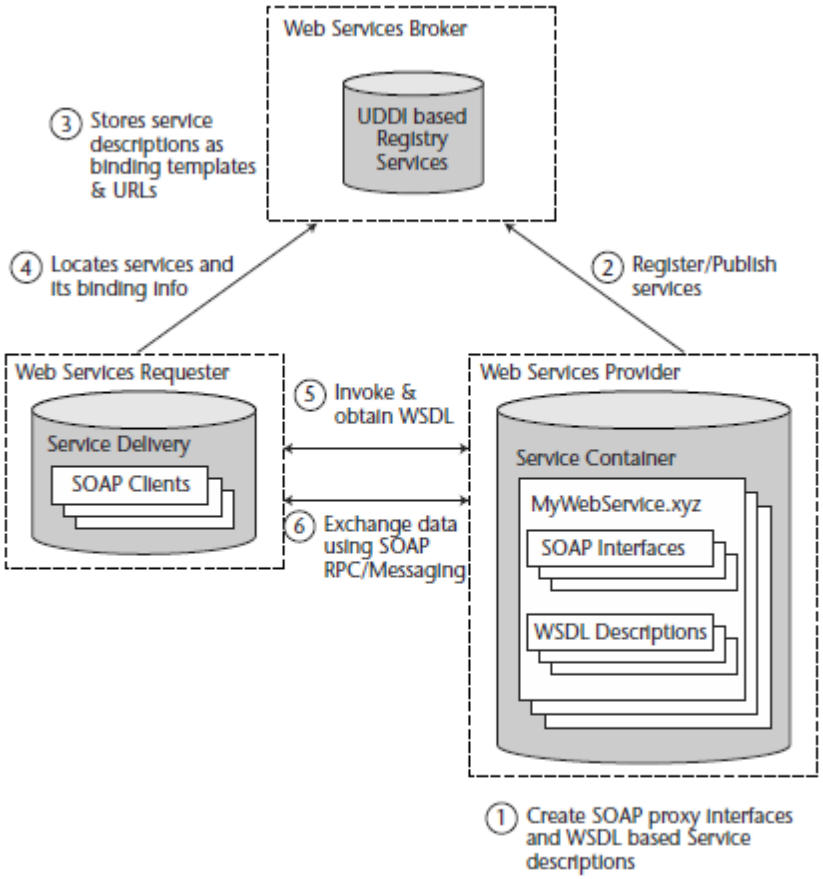


Figure 3.5 Process steps involved in implementing Web services.

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. As illustrated in Figure 3.5, the basic steps of implementing Web services are as follows:

1.The service provider creates the Web service typically as SOAPbased 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

UNIT-2

SHORT ANSWER QUESTIONS

1. Define the Emergence of Soap?

SOAP initially was developed by Develop Mentor, 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. In 1999, the SOAP 1.0 specification was made publicly available as a joint effort supported by vendors like Rogue Wave, IONA, Object Space, Digital Creations, User Land, Microsoft, and Develop Mentor. Later, the SOAP 1.1 specification was released as a W3C Note, with additional contributions from IBM and the Lotus Corporation supporting a wide range of systems and communication models like RPC and Messaging.

Nowadays, the current version of SOAP 1.2 is part of the W3C XML Protocol Working Group effort led by vendors such as Sun Microsystems, IBM, HP, BEA, Microsoft, and Oracle. At the time of this book's writing, SOAP 1.2 is available as a public W3C working draft.

2. What are the Soap Specifications?

The SOAP 1.1 specifications define the following:

- Syntax and semantics for representing XML documents as structured SOAP messages
- Encoding standards for representing data in SOAP messages
- A communication model for exchanging SOAP messages
- Bindings for the underlying transport protocols such as SOAP transport
- Conventions for sending and receiving messages using RPC and messaging.

3. What are t he Features are Not Supported by Soap?

The following features are not supported by SOAP

- Garbage collection
- Object by reference
- Object activation
- Message batching

4. Define Soap message?

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.

5. List out the Structural Elements of the Soap Message Structure?

The structural format of a SOAP message (as per SOAP version 1.1 with attachments) contains the following elements:

- Envelope
- Header (optional)
- Body
- Attachments (optional)

5. Draw the Structure of A SOAP Message With Attachments?

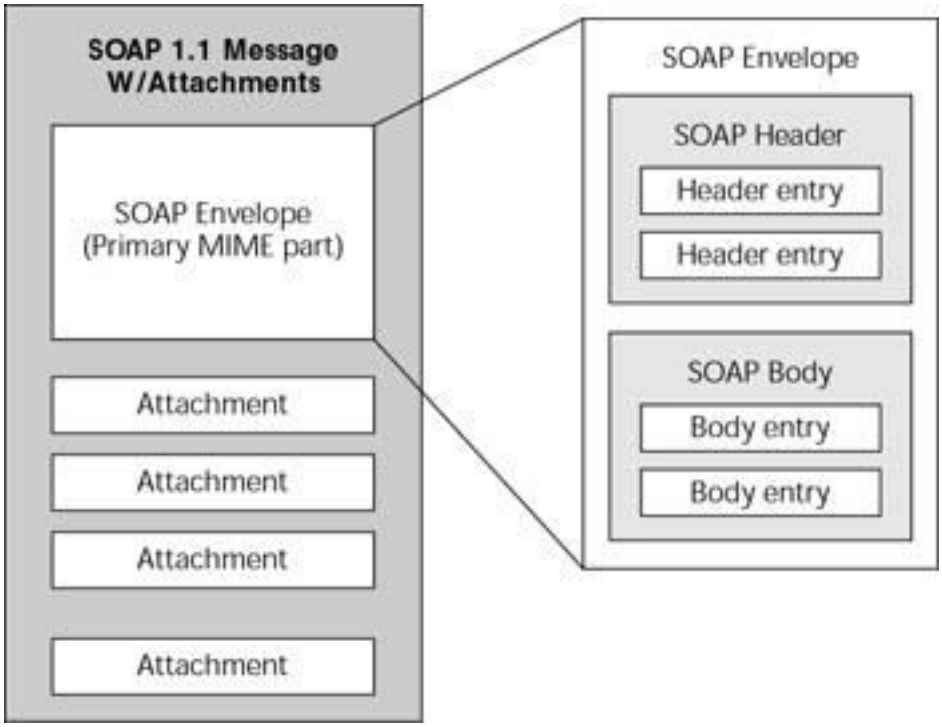


Fig: soap message structure

6. Define Serialization and Deserialization?

In SOAP messages, all data and application-specific data types are represented as XML, and it is quite important to note that there is no generic mechanism to serialize application-specific data types to XML. SOAP implementation provides application-specific encoding for application programming languages (such as Java and C++). It also enables developers to define custom application-specific encoding, especially to handle the data representation required and its data types adopting data types defined by the "W3C XML Schema, Part -2" specification (see www.w3.org/TR/xmlschema-2/). This is usually implemented as application- or programming language-specific serialization and

deserialization mechanisms that represent application-specific data as XML and XML as application-specific data.

Most SOAP implementations provide their own serialization and deserialization mechanisms and a predefined XML Schema supporting the SOAP encoding rules and mapping application-specific data types. These serializers and deserializers supporting SOAP encoding rules provide the encoding and decoding of data on runtime by mapping XML elements to target application objects and vice versa. It leverages interoperability between disparate applications using SOAP messages.

We will study the serializers and deserializers of a SOAP implementation in the example illustration using Apache Axis discussed in the section titled Axis Infrastructure and Components. So far we discussed the structure of a SOAP message and the representation of its data types. Now, let's take a look at how to exchange SOAP messages using SOAP communication

7. Define Intermediaries & Actor?

SOAP defines intermediaries as nodes for providing message processing and protocol routing characteristics between sending and receiving applications. Intermediary nodes reside in between the sending and receiving nodes and process parts of the message defined in the SOAP header. The two types of intermediaries are as follows:

> **Forwarding intermediaries.** This type processes the message by describing and constructing the semantics and rules in the SOAP header blocks of the forwarded message.
>
> **Active intermediaries.** This type handles additional processing by modifying the outbound message for the potential recipient SOAP nodes with a set of functionalities.

In general, SOAP intermediaries enable a distributed processing model to exist within the SOAP message exchange model. By using SOAP intermediaries, features can be incorporated like store and forward, intelligent routing, transactions, security, and logging, as well as other value additions to SOAP applications.

In a SOAP message to represent a target SOAP node, the SOAP actor global attribute with a URI value can be used in the Header element. SOAP defines an actor with a URI value, which identifies the name of the SOAP receiver node as an ultimate destination.

8. Define 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.

9. What are The Limitations of The 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:

1. The specification does not address message reliability, secure message delivery, transactional support, and its communication requirements of a SOAP implementation.
2. The specification does not address issues like object activation and object lifecycle management.
3. The specification discusses HTTP as the primary transport protocol but does not discuss the usage of other transport protocols.
4. 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.

LONG ANSWER QUESTIONS

1. Describe the Anatomy of the Soap Message?

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.

Before we go exploring the SOAP features, let's walk through an existing SOAP message and understand the XML syntax, semantic rules, and conventions. The example is a SOAP request/response message for obtaining book price information from a book catalog service provider. The SOAP request accepts a string parameter as the name of the book and returns a float as the price of the book as a SOAP response.

SOAP request message.

```
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.priceList">
```

```
        <bookname xsi:type='xsd:string'>
              Developing Java Web Services</bookname>
        </m:GetBookPrice>
       </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*.

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

SOAP response message.

```
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>
```

You might have noticed that the SOAP message contains a SOAP Envelope SOAP-ENV:Envelope as its primary root element, and it relies on defined "XML Namespaces" commonly identified with a keyword xmlns and specific prefixes to identify the elements and its encoding rules. All the elements in the message are associated with SOAP-ENV-defined namespaces.

Note that a SOAP application should incorporate and use the relevant SOAP namespaces for defining its elements and attributes of its sending messages; likewise, it must be able to process the receiving messages with those specified namespaces. These namespaces must be in a qualified W3C XML Schema, which facilitates the SOAP message with groupings of elements using prefixes to avoid name collisions.

Usually a SOAP message requires defining two basic namespaces: SOAP Envelope and SOAP Encoding. The following list their forms in both versions 1.1 and 1.2 of SOAP.

SOAP Envelope

- http://schemas.xmlsoap.org/soap/envelope/ (SOAP 1.1)
- http://www.w3.org/2001/06/soap-envelope (SOAP 1.2)

SOAP Encoding

- http://schemas.xmlsoap.org/soap/encoding/ (SOAP 1.1)
- http://www.w3.org/2001/06/soap-encoding (SOAP 1.2)

2. Explain in Detail About the Soap Message Structure?

the structural format of a SOAP message (as per SOAP version 1.1 with attachments) contains the following elements:

- Envelope
- Header (optional)
- Body
- Attachments (optional)

Figure 2.1 represents the structure of a SOAP message with attachments. Typically, a SOAP message is represented by a SOAP envelope with zero or more attachments. The SOAP message envelope contains the header and body of the message, and the SOAP message attachments enable the message to contain data, which include XML and non-XML data (like text/binary files). In fact, a SOAP message package is constructed using the

MIME Multipart/Related structure approaches to separate and identify the different parts of the message.

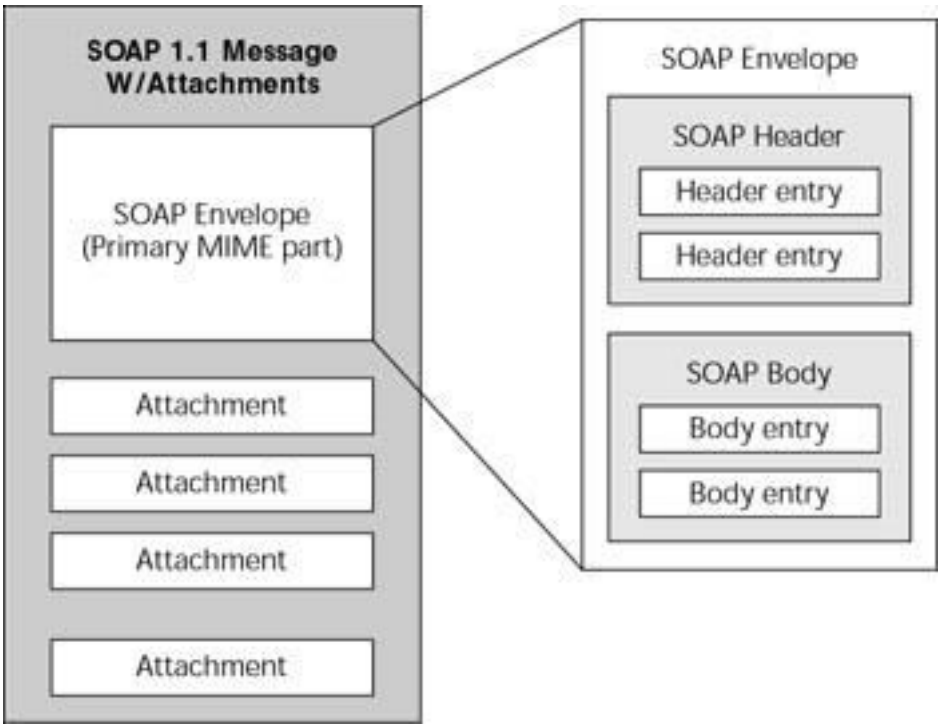


Figure 2.1: Structure of a SOAP message with attachments.

Now, let's explore the details and characteristics of the parts of a SOAP 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.

SOAP Envelope element.

```
<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.w3.org/2001/XMLSchema"
    SOAP-ENV:
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <!--SOAP Header elements - -/>
  <!--SOAP Body element - -/>
  </SOAP-ENV:Envelope>
```

SOAP Header

The SOAP header is represented as the first immediate child element of a SOAP envelope, and it has to be namespace qualified. In addition, it also may contain zero or more optional child elements, which are referred to as SOAP header entries. The SOAP encodingStyle attribute will be used to define the encoding of the data types used in header element entries. The SOAP actor attribute and SOAP mustUnderstand attribute can be used to indicate the target SOAP application node (Sender/Receiver/Intermediary) and to process the Header entries.

SOAP Header element.

```
<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>
```

The SOAP header represents a transaction semantics entry using the SOAP mustUnderstand attribute. The mustUnderstand attribute is set to "1", which ensures that the receiver (URI) of this message must process it. We will look into the mustUnderstand attributes in the next section.

SOAP headers also provide mechanisms to extend a SOAP message for adding features and defining high-level functionalities such as security, transactions, priority, and auditing.

SOAP Body

A SOAP envelope contains a SOAP body as its child element, and it may contain one or more optional SOAP body block entries. The Body represents the mandatory processing information or the payload intended for the receiver of the message. The SOAP 1.1 specification mandates that there must be one or more optional SOAP Body entries in a message. 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 body representing an RPC call for getting the book price information from www.wiley.com for the book name *Developing Java Web Services*.

SOAP Body element

```
<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>
```

Like other elements, the Body element also must have a qualified namespace and be associated with an encodingStyle attribute to provide the encoding conventions for the payload. In general, the SOAP Body can contain information defining an RPC call, business documents in XML, and any XML data required to be part of the message during communication.

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 faultcode 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 namespace 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.

Let's take a look at the common examples of SOAP fault scenarios.

SOAP mustUnderstand

The SOAP mustUnderstand attribute indicates that the processing of a SOAP header block is mandatory or optional at the target SOAP node. The following example is a SOAP request using mustUnderstand and the response message from the server.

Listing 4.9 shows the request message where the SOAP message defines the header block with a mustUnderstand attribute of 1.

SOAP mustUnderstand attribute.

```
<SOAP-ENV:Header>
    <wiley:Catalog
     xmlns:wiley="http://jws.wiley.com/2002/bookList"
                    SOAP-ENV:mustUnderstand="1">
    </wiley:Catalog>
</SOAP-ENV: Header>
```

Lising 4.10 is an example response message from the server when the server could not understand the header block where the mustUnderstand is set to 1. Listing 4.10 is the server-generated fault message detailing the issues with the header blocks using misUnderstood and qname (faulting SOAP nodes) and providing a complete SOAP fault.

3. **Define The Soap Encoding? Explain Soap Encoding Data Types?**

SOAP 1.1 specifications stated that SOAP-based applications can represent their data either as literals or as encoded values defined by the "XML Schema, Part -2" specification (see www.w3.org/TR/xmlschema-2/). Literals refer to message contents that are encoded according to the W3C XML Schema. Encoded values refer to the messages encoded based on SOAP encoding styles specified in SOAP Section 5 of the SOAP 1.1 specification. The namespace identifiers for these SOAP encoding styles are defined in http://schemas.xmlsoap.org/soap/encoding/ (SOAP 1.1) and http://www.w3.org/2001/06/soap-encoding (SOAP 1.2).

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: Datatypes" 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.

Enumeration

Enumeration defines a set of names specific to a base type.

Enumeration data type.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    elementFormDefault="qualified">
  <xs:element name="ProductType">
     <xs:simpleType base="xsd:string">
        <xs:enumeration value="Hardware">
        <xs:enumeration value="Software">
     </xs:simpleType>
  </xs:element>
</xs:schema>
```

Array of Bytes

An example of an array data type of an array of binary data that is represented as text using base64 algorithms and expressed using a W3C XML Schema.

An array

```
<myfigure xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:enc=" http://schemas.xmlsoap.org/soap/encoding">
     xsi:type="enc:base64">
     sD334G5vDy9898r32323</myfigure>
```

Polymorphic Accessor

The polymorphic accessor enables programming languages to access data types during runtime. SOAP provides a polymorphic accessor instance by defining an xsi: type attribute that represents the type of the value.

The following is an example of a polymorphic accessor named price with a value type of "xsd:float" represented as follows:

<price xsi:type="xsd:float">1000.99</price>

And, the XML instance of the price data type will be as follows:

<price>1000.99</price>

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

An XML Schema of the Structure data type representing the "Shipping address" with subelements like "Street," "City," and "State."

Structure data type.

```
<xs:element name="ShippingAddress"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" >
    <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:element ref="Country" type="xsd:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>.
```

Resulting XML instance of a structure data type.

```
<e:ShippingAddress>
 <Street>1 Network Drive</Street>
 <City>Burlington</City>
 <State>MA</State>
 <Zip>01803</Zip>
 <Country>USA</Country>
</e:ShippingAddress>
```

The structure also can contain both simple and complex data type values that can reference each other . The structure uses the "href" attribute to reference the value of the matching elemen

Structure data type using simple and complex types.

```
<e:Product>
 <product>Sun Blade 1000</product>
 <type>Hardware</type>
 <address href="#Shipping"/>
 <address href="#Payment"/>
<e:/Product>
<e:Address id="Shipping">
 <Street>1 Network Drive</Street>
 <City>Burlington</City>
 <State>MA</State>
 <Zip>01803</Zip>
 <Country>USA</Country>
</e:Address>
<e:Address id="Payment">
 <Street>5 Sunnyvale Drive</Street>
 <City>Menlopark</City>
 <State>CA</State>
 <Zip>21803</Zip>
 <Country>USA</Country>
```

</e:Address>

Array Types

XML Schema of an Array data type representing MyPortfolio — a list of portfolio stock symbols.

Compound array types.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding" >
<xs:import
  namespace="http://schemas.xmlsoap.org/soap/encoding" >
<xs:element name="MyPortfolio" type="enc:Array"/>
</xs:schema>
```

Resulting XML instance of a compound array type.

```
<MyPortfolio xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:enc=" http://schemas.xmlsoap.org/soap/encoding"
        enc:arrayType="xs:string[5]">
 <symbol>SUNW</symbol>
 <symbol>IBM</symbol>
 <symbol>HP</symbol>
 <symbol>RHAT</symbol>
 <symbol>ORCL</symbol>
</MyPortfolio>
```

Multiple References in Arrays

SOAP encoding also enables arrays to have other arrays as member values. This is accomplished by having the id and href attributes to reference the values.

Multiple references in arrays.

```
 <MyProducts xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:enc="
```

```
        http://schemas.xmlsoap.org/soap/encoding"
        enc:arrayType="xs:string[][3]">
 <item href="#product-hw"/>
 <item href="#product-sw"/>
 <item href="#product-sv"/>
<SOAP-ENC:Array id="product-hw"
                SOAP-ENC:arrayType="xsd:string[3]">
 <item>SUN Blade 1000</item>
 <item>SUN Ultra 100</item>
 <item>SUN Enterprise 15000</item>
  </SOAP-ENC:Array>
   <SOAP-ENC:Array id="product-sw"
                SOAP-ENC:arrayType="xsd:string[2]">
 <item>Sun Java VM</item>
 <item>Sun Solaris OS</item>
 </SOAP-ENC:Array>
   <SOAP-ENC:Array id="product-sv"
                SOAP-ENC:arrayType="xsd:string[2]">
 <item>Sun Java Center services</item>
<item>Sun Java Web Services</item>
 </SOAP-ENC:Array>
```

Partially Transmitted Arrays

Partially transmitted arrays are defined using a SOAP-ENC:offset, which enables the offset position to be indicated from the first element (counted as zero-origin), which is used as an offset of all the elements that will be transmitted. using SOAP-ENC:offset="4" transmits the fifth and sixth elements of a given array of numbers (0,1,2,3,4,5).

Partially transmitted arrays.

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[6]"
                        SOAP-ENC:offset="[2]">
   <item> No: 2</item>
```

```
  <item> No: 3</item>
  <item> No: 4</item>
  <item> No: 5</item>
</SOAP-ENC:Array>
```

Sparse Arrays

Sparse arrays are defined using a SOAP-ENC:position, which enables the position of an attribute to be indicated with an array and returns its value instead of listing every entry in the array

Sparse arrays.

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:int[10]">
  <SOAP-ENC:int SOAP-ENC:position="[0]">0</SOAP-ENC:int>
  <SOAP-ENC:int SOAP-ENC:position="[10]">9</SOAP-ENC:int>
</SOAP-ENC:Array>
```

This summarizes the SOAP encoding defined in the SOAP 1.1 specification. Now, let's take a look at how to handle the custom encoding requirements specific to applications.

4. Explain About The Soap Message Exchange Models?

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.

Figure 4.2 represents a basic SOAP message exchange model with different SOAP nodes.

Figure: Basic SOAP message exchange model.

In a SOAP message exchange model, the SOAP message passes from the initiator to the final destination by passing through zero to many intermediaries. In a SOAP messaging path, the SOAP intermediaries represent certain functionalities and provide routing to the next message destination. It is important to note that SOAP does not define the actual SOAP senders, intermediaries, and receivers of the SOAP message along its message path or its order of destination. However, SOAP can indicate which part of the message is meant for processing at a SOAP node. Thus, it defines a decentralized message-exchanging model that enables a distributed processing in the message route with a message chain.

Represents an example of a complete message exchange model with a sender, receiver, and its intermediaries. In the previous example, the message originates from Sender A to Receiver D via Intermediaries B and C as a request chain, and then as a response chain the message originates from Receiver D to Sender A via Intermediary E.

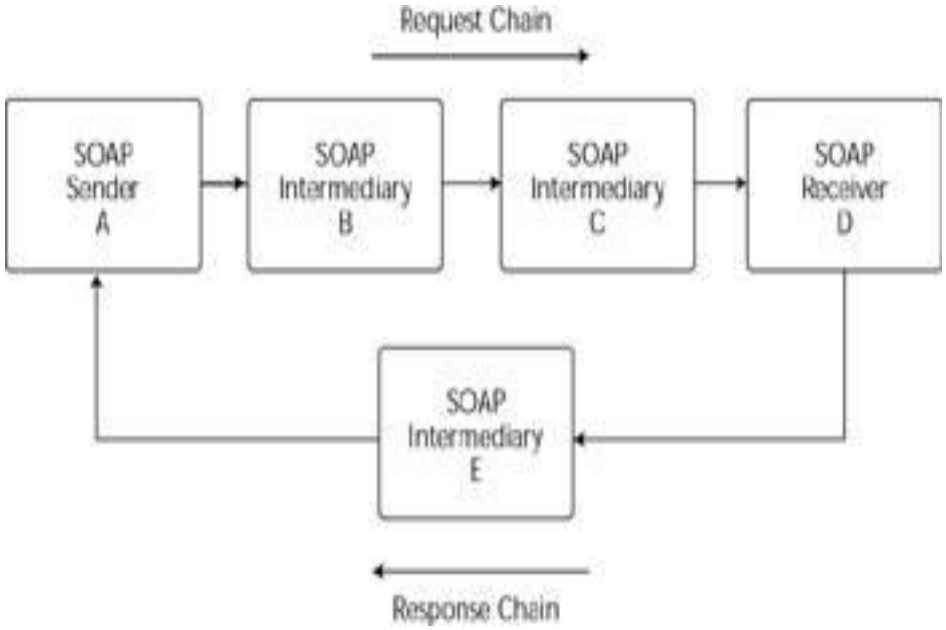


Figure : SOAP message exchange model with intermediaries.

5. Describe the Soap Communication in Detail?

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.

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 "bookname" parameter of "Developing Java Web Services".

SOAP request using RPC-based communication.

```
<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>
  </SOAP-ENV:Header>
  <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>
</SOAP-ENV: Envelope>
```

The SOAP message represents the SOAP RPC response after processing the SOAP request, which returns the result of the GetBookPrice method from the SOAP server namespace http://www.wiley.com/jws.book.priceList using a "Price" parameter with "$50" as its value.

SOAP response message using RPC-based communication.

```
<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>
```

The communication model in is similar to a traditional CORBA- or RMI-based communication model, except the serialized data types are represented by XML and derived from SOAP encoding rules

6. Explain about the 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.

SOAP message using messaging-based communication.

```
<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>
```

So far, we have looked at SOAP messages, conventions, encoding rules, and its communication model. Now, let's take a look at its bindings to the transport protocols required for its messaging environment.

7. Descibe the Soap Message Exchange Patterns?

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 (see Figure 4.4). It is typically found in email messages.

Figure : One-way message pattern.

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 .

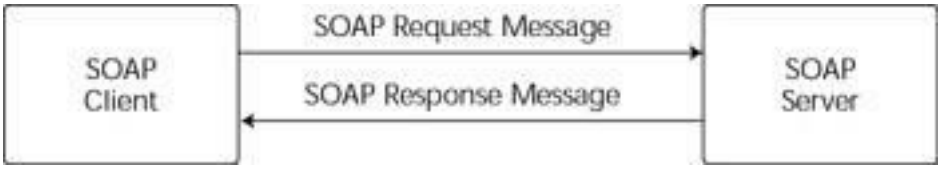


Figure: Request/Response pattern.

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 .

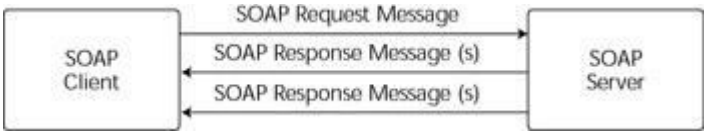


Figure : Request/N*Response pattern.

Notification pattern. In this pattern, the SOAP server sends messages to the SOAP client like an event notification, without regard to a response.

Figure : Notification pattern.

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.

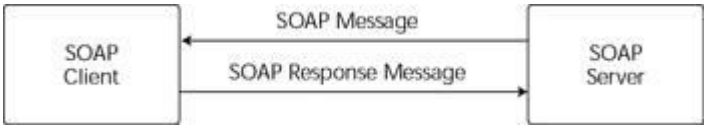


Figure : Solicit-response pattern.

Note that the previous patterns can be implemented based on the transport protocols and their supporting communication models.

8. How to Develop the Soap Web Services Using Java & Axis?

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 communities, 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.

Developing Web Services Using Apache Axis

Apache Axis is an open-source implementation that provides a Java-based SOAP implementation for developing Web services. To implement Web services, it facilitates a SOAP runtime environment and Java-based API framework for implementing the core components of Web services adopting compliant standards and protocols.

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 Weblogic)
- 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. At the time of this book's writing, Axis 1.0B3 provides limited implementation support of JAX-RPC 1.0 and SAAJ 1.1 specifications. To find out the current status of the Axis implementation and its availability for download, go to Apache's XML Web site at http://xml.apache.org/axis/

UNIT 3

DESCRIBING WEB SERVICES

SHORT ANSWERS

1). Define WSDL?

Ans. The Web Services Description Language, or WDDL, is an XML schema based specification for describing Web services as a collection of operations and data input/output parameters as messages. WSDL also defines the communication model with a binding mechanism to attach any transport protocol, data format, or structure to an abstract message, operation, or endpoint.

EX:- Shows a WSDL example that describes a Web service meant for obtaining a price of a book using a GetBookPrice operation.

```
<?xml version=”1.0”?>
<definitions name=”BookPrice”
targetNamespace=”http://www.wiley.com/bookprice.wsdl”
xmlns:tns=http://www.wiley.com/bookprice.wsdl
```

2). List out the structural elements of WSDL?

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

1. <definition>
2. <types>
3. <portType>
4. <Port>
5. <message>
6. <Binding>
7. <service>

3). What are the operations supported by WSDL?

Ans. Four types of operations are supported by WSDL:

One – way operation:- Represents a service that just receives the message, it is defined by a single <input> message element.

Request – response operation:- A request – response operation represents a service that receives a request message and sends a response message. It is defined by one <input> message followed by one <output> message.

Solicit – response operation:- It represents a service that sends a request message and that receives the response message. It is defined by in <output> message followed by an <input> message.

Notification operation:- this operation represents a service that sends a message, hence, it is defined by single <output> message.

4). Define <Definition> element?

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

5). What are the namespaces of <Definition> element?

Ans. The two important namespaces that the <definition> element defines:

WSDL instance Specific namespace:-

The target namespace attribute of the <definition> element lets the WSDL document make references to itself as an XML schema namespace.

Default namespace for a WSDL document:-

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

6). Define <Type> element?

Ans. <Types>:- This elements defines all of the data types that would be used to describe the messages that are exchanged b/w the web service and the service user. It does not mandate the use of a specific typing system. However as per WSDL specification it uses XML schemas as a default typing system.

```
<schema targetNamespace="http://example.com/stockquote.xsd"
  xmlns="http://www.w3.org/2000/10/XMLSchema">

    <element name="TradePriceRequest">
       <complexType>
          <all>
             <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
    </element>

    <element name="TradePrice">
       <complexType>
          <all>
             <element name="price" type="float"/>
          </all>
    </complexType>
    </element>

  </schema>
</types>
```

7). Define <Message> Element?

Ans. <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 refer to the request parameters or response return values.

```
<message name="SayHelloRequest">
        <part name="firstName" type="xsd:string"/>
        </message>

        <message name="SayHelloResponse">
        <part name="greeting" type="xsd:string"/>
        </message>
```

8). Define <Port> Element?

Ans. <PortType>:- This element defines the abstract of the operations supported by a web service by combining various request na response messages defined by <message> elements, each operation refers to an input message and an output message.

```
<portType name="Hello_PortType">
  <operation name="sayHello">
      <input message="tns:SayHelloRequest"/>
      <output message="tns:SayHelloResponse"/>
   </operation>
</portType>
```

9). Define <Binding> element?

Ans. <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.

10). Define <Service> element?

Ans. <Service>:- This element aggregates a set of related <port> elements>, each which uniquely specify the binding information of the web service.

A <service> consisting of multiple <port> elements.

```
<service name="Hello_Service">
```

```
   <documentation>WSDL File for HelloService</documentation>
   <port binding="tns:Hello_Binding" name="Hello_Port">
      <soap:address
         location="http://www.examples.com/SayHello/">
   </port>
</service>
```

11). Define <Port> element?

Ans. <Port>:- this element Specifies an address for binding to the web service.

```
<service name="Hello_Service">
   <documentation>WSDL File for HelloService</documentation>
   <port binding="tns:Hello_Binding" name="Hello_Port">
      <soap:address
        location="http://www.examples.com/SayHello/">
   </port>
</service>
```

12). what is HTTP GET & POST binding?

Ans. WSDL includes a binding for HTTP 1.1's GET and POST verbs in order to describe the interaction between a Web Browser and a web site. This allows applications other than Web Browsers to interact with the site. The following protocol specific information may be specified:

- An indication that a binding uses HTTP GET or POST
- An address for the port
- A relative address for each operation (relative to the base address defined by the port)

HTTP GET/POST Examples

The following example shows three ports that are bound differently for a given port type.

If the values being passed are part1=1, part2=2, part3=3, the request format would be as follows for each port:

port1: GET, URL="http://example.com/o1/A1B2/3"
port2: GET, URL="http://example.com/o1?p1=1&p2=2&p3=3
port3: POST, URL="http://example.com/o1", PAYLOAD="p1=1&p2=2&p3=3"

13). what are the limitations of WSDL?

Ans. WSDL 1.1 has an obvious limitations:

- Its incapability of being able to describe complex business web services, which typically are constituted by orchestrating multiple finer-grained web services.

RPC-encoded WSDL documents are not supported. For these documents, use `createClassFromWsdl`.

The following WSDL documents are not supported:

- Documents that the Apache™ CXF program cannot compile into complete code.
- Documents that import other WSDL documents that contain WSDL type definitions.
- On Windows®, documents that import other WSDL documents might fail if the imported URI contains certain punctuation characters.
- Some documents with messages containing multiple parts.
- Some documents with schemas containing anonymous complex types.
- Some documents defining an input parameter to an operation as a simple type. When you invoke such an operation, for example `GetMyOp`, MATLAB® displays one of the following errors.
- Error using xxx/GetMyOp. Too many input arguments.

14) Explain what is WSDL?

WSDL stands for Web Services Description Language. It is a simple XML document that contains a set of definitions to describe or locate a web service.

15) **Explain what is the WSDL document structure?**

The WSDL document structure consists of these major elements

- <types>: A container for data type definitions used by the web services
- <message>: A typed definition of the data being communicated
- <portType>: A set of operations supported by one or more endpoints

- <binding>: A protocol and data format for a specific port type

16) What is the prefix used for the target namespace for the WSDL document?

Prefix "xmlns:tns=target name" is used for target namespace for the WSDL document.

17) Explain what is message element in WSDL?

- A message is protocol independent, and it describes the data being exchanged between the consumers and web service providers
- Each web service has two messages input and output. The input determines the parameters for the web service and the output determines the return data from the web service
- Each <message> element contains zero or more <part> parameters, one for each parameter of the web service function
- <Part> element relates to the parameter or return value in the RPC call
- The <part> name order reflects the order of the parameters in the RPC signature

18) Explain what is "soap:body"?

- "soap:body" is a SOAP extension element used as a sub element of the "wsdl:input/output" inside the wsdl binding and operation. It is used to provide information on how the content of the SOAP body element is constructed.

19) Explain how endpoints are defined in WSDL?

Endpoints represent an instantiated service; they are determined by combining a binding and the networking details used to expose the endpoint. Endpoints are defined in a contract using a combination of the WSDL port elemennt and WSDL service element. The port elements define the actual endpoints

20) Explain what is the difference between Message type and Element in WSDL?

Message type: It creates variable based on a message type that you defined in WSDL

Element type: It creates variable based on an element that you defined in Schema

21) How to validate WSDL file?

WSDL file is a contract between consumer and web service clients. A WSDL validator verifies whether the file can be consumed by other applications before you give the url to your end-users. To validate your file you need to set your criteria like

- Does it require to be validate according to WSDL XML schema
- Does it require to fulfill with known best practices
- Does it require to be parsed correctly by common soap stacks

You can use a commercial tool like XMLSpy to validate WSDL file.

LONG ANSWER QUESTIONS

1). Explain WSDL in the world of web services?

Ans. 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 services.

- ❖ Interface information describing all the publicly available functions.
- ❖ Data type information for the incoming and outgoing messages to these functions.
- ❖ Binding information about the protocol to be used for invoking the specified web services.
- ❖ Address information for locating the specified web services.

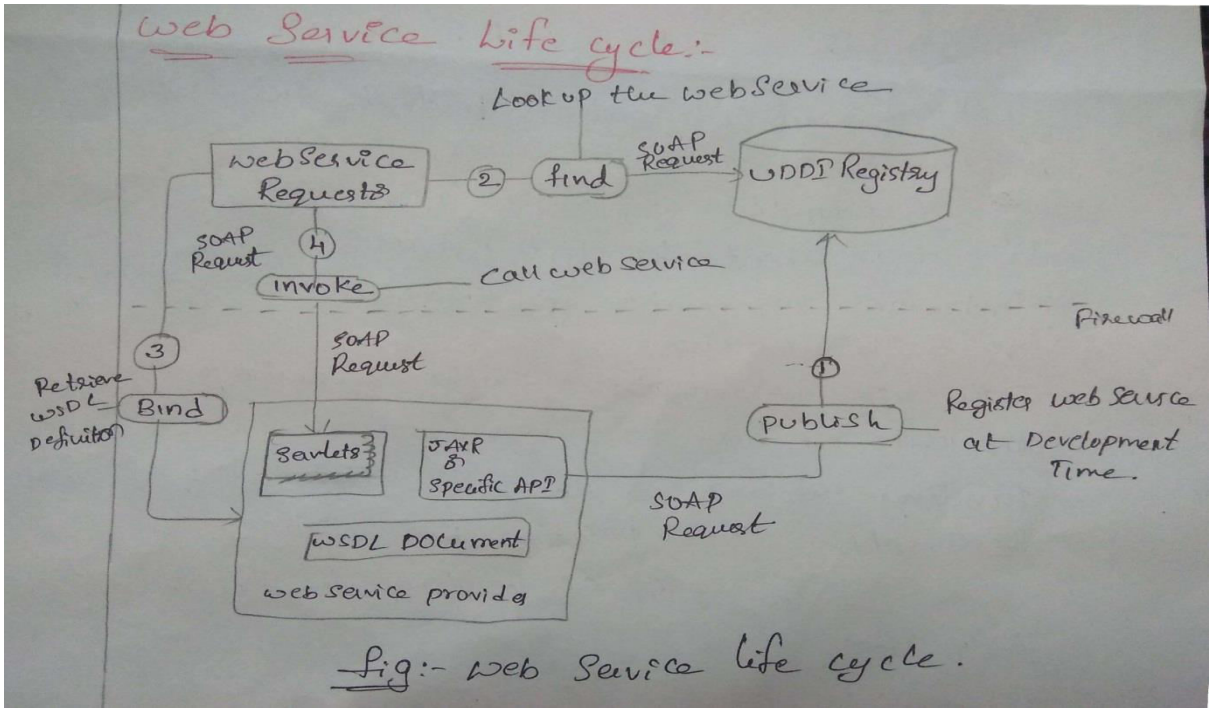
Once the web service is developed, then create its WSDL definition.

It can be created manually or by using a tool. Many tools are available for generating a WSDL definition from existing java classes, J2EE components, or from scratch.

Once the WSDL definition is created, a link to it is published in the web service registry.
Then the users can follow that link and find out the location of the web service; the function calls that it supports and how to invoke these calls.
Finally, the users would use this information to formulate a SOAP request or any other type of request in order to invoke the function on a web service.

2). Illustrate web service life cycle?

Ans.



Step 1:- A service provider is publishing its web services 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 web services information to a UDDI registry.

Step 2:- An interested service user locating the web service and finally obtaining information about the invoking web service from the published WSDL definition.

This step involves download the WSDL definition to the service user system and deserializing WSDL to a Java class.

This java interface services as a proxy to the actual web services.

It consists of the binding information of web service.

Step 3:- The service user binding to the web service at run time.

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 extracted from the web service WSDL definition.

The service user's application uses the Java interface representing WSDL as a proxy in order to invoke the methods/functions exposed by the web services.

3). Explain the Anatomy of a WSDL definition document?

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

1. **<Definition>:-** A WSDL document is a set of definitions. These definitions are defined inside the <definition> element, which is the root element in a WSDL document. It defines the name of the web service and also declares the namespaces that are used throughout the rest of the WSDL document.

   The two important namespaces that the <definition> element defines:

WSDL instance Specific namespace:-

The target namespace attribute of the <definition> element lets the WSDL document make references to itself as an XML schema namespace.

Default namespace for a WSDL document:-

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

EX:- <definitions name="HelloService"
targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"

```
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   ..............................................
</definitions>
```

2. **<Types>:-** this elements defines all of the data types that would be used to describe the messages that are exchanged b/w the web service and the service user. It does not mandate the use of a specific typing system. However as per WSDL specification it uses XML schemas as a default typing system.

```
<types>
   <schema targetNamespace="http://example.com/stockquote.xsd"
     xmlns="http://www.w3.org/2000/10/XMLSchema">

        <element name="TradePriceRequest">
           <complexType>
              <all>
                 <element name="tickerSymbol" type="string"/>
              </all>
          </complexType>
        </element>

        <element name="TradePrice">
           <complexType>
              <all>
                 <element name="price" type="float"/>
              </all>
        </complexType>
        </element>

     </schema>
  </types>
```

3. **<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 refer to the request parameters or response return values.

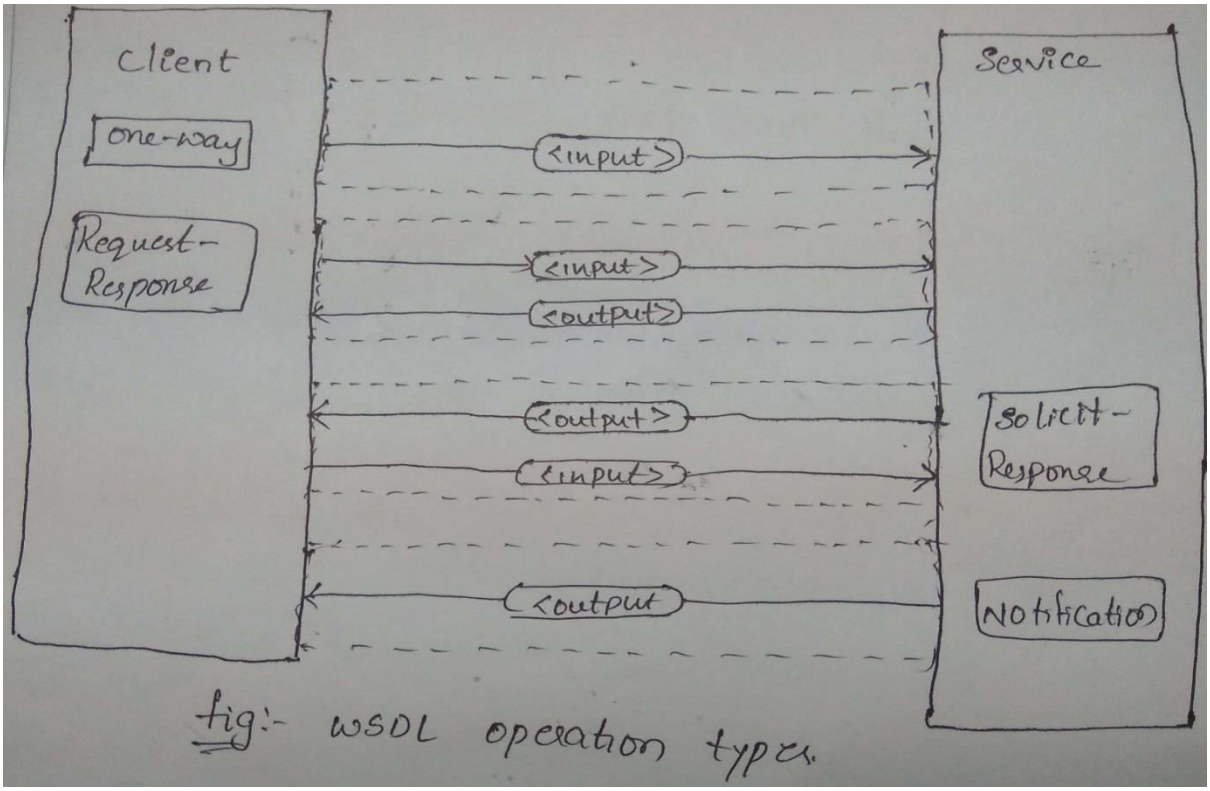
```
<message name="SayHelloRequest">
<part name="firstName" type="xsd:string"/>
</message>

<message name="SayHelloResponse">
<part name="greeting" type="xsd:string"/>
</message>
```

4. **<PortType>:-** This element defines the abstract of the operations supported by a web service by combining various request na response messages defined by <message> elements, each operation refers to an input message and an output message.

```
<portType name="Hello_PortType">
  <operation name="sayHello">
      <input message="tns:SayHelloRequest"/>
      <output message="tns:SayHelloResponse"/>
   </operation>
</portType>
```

Four types of operations are supported by WSDL:
One – way operation:- Represents a service that just receives the message, it is defined by a single <input> message element.
Request – response operation:- A request – response operation represents a service that receives a request message and sends a response message. It is defined by one <input> message followed by one <output> message.
Solicit – response operation:- It represents a service that sends a request message and that receives the response message. It is defined by in <output> message followed by an <input> message.
Notification operation:- this operation represents a service that sends a message, hence, it is defined by single <output> message.



5. **<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.
6. **<Service>:-** This element aggregates a set of related <port> elements>, each which uniquely specify the binding information of the web service.

```
<service name="Hello_Service">
   <documentation>WSDL File for HelloService</documentation>
   <port binding="tns:Hello_Binding" name="Hello_Port">
      <soap:address
         location="http://www.examples.com/SayHello/">
   </port>
</service>
```

A <service> consisting of multiple <port> elements.

7. **<Port>:-** this element Specifies an address for binding to the web service.

```
<service name="Hello_Service">
   <documentation>WSDL File for HelloService</documentation>
   <port binding="tns:Hello_Binding" name="Hello_Port">
```

```
        <soap:address
          location="http://www.examples.com/SayHello/">
      </port>
   </service>
```

4). Explain WSDL Binding?

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>.

WSDL Binding Extensions:-

WSDL allows user-defined elements, also known as Extensibility elements, defined by a default WSDL namespace. These elements are commonly used to specify some technology-specific binding. Extensibility elements used to specify a technology-specific binding are known as WSDL Binding Extensions.

Extensibility elements provides 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:

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

WSDL Binding support for operations:-

All four types of operations supported by WSDL are one-way, request-response, solicit-response, and notification. Binding determines how the messages are actually sent for instance within a single communication or two independent communications. Thus, binding for a specific operation type must be defined in order to successfully carry out that type of operation. Although the WSDL structure supports the bindings for four operations, the WSDL specification defines bindings for only one-way and request-response operations.in order to use WSDL to describe services that support solicit-response and/or notification types

operations, the communication protocol of web services must define the WSDL binding extensions.

5). Explain SOAP binding in WSDL?

Ans. SOAP Binding:-

WSDL 1.1 includes built-in extensions for SOAP 1.1. It allows you to specify SOAP specific details including SOAP headers, SOAP encoding styles, and the SOAP Action HTTP header. The SOAP extension elements include the following:

- <soap: binding>
- <soap: operation>
- <soap: body>

<soap: binding>

This element indicates that the binding will be made available via SOAP. The *style* attribute indicates the overall style of the SOAP message format. A style value of *rpc* specifies an RPC format.

The *transport* attribute indicates the transport of the SOAP messages. The value http://schemas.xmlsoap.org/soap/http indicates the SOAP HTTP transport, whereas http://schemas.xmlsoap.org/soap/smtp indicates the SOAP SMTP transport.

<soap: operation>

This element indicates the binding of a specific operation to a specific SOAP implementation. The *soapAction* attribute specifies that the SOAPAction HTTP header be used for identifying the service.

<soap: body>

This element enables you to specify the details of the input and output messages. In the case of HelloWorld, the body element specifies the SOAP encoding style and the namespace URN associated with the specified service.

Here is the piece of code from the Example chapter:

<binding name="Hello_Binding" type="tns:Hello_PortType">

```
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/ >
  <operation name="sayHello">
    <soap:operation soapAction="sayHello"/>

    <input>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:examples:helloservice" use="encoded"/>
    </input>

    <output>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:examples:helloservice" use="encoded"/>
    </output>
  </operation>
</binding>
```

6). What are WSDL Tools?

Ans. The Web Services Description Language tool generates code for XML Web services and XML Web service clients from WSDL contract files, XSD schemas, and .discomap discovery documents.

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 that are described in the given WSDL definition.

WSDL proxy generation: - This functionality can read a WSDL and produce a specific language binding 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. Some of the famous WSDL tools in the Java Web Services space are:

TABLE: WSDL Tools

TOOL	DOWNLOAD FROM
Sun ONE Studio 4	www.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/insex.shtml
Apache Axis	http://xml.apache.org/axis

The WSDL tools provided by the Systinet WASP platform.

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.

Generating WSDL from a java class

Implementation of the web service has already been created first, the Java2WSDL tool can be used to generate WSDL.

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

Package jws.ch5;

```
Public class WeatherInfoJavaService
{
    Public float GetWeatherInfo(String sCountry, String sZip, String sInstance)
    {
        Return 65.0f;
`   }
    Public void SetWeatherInfo(String sCountry, String sZip, String sInstance,
String sTemperature)
    {
    }
}
```

Our web service supports SOAP – based communication, and hence, a SOAP binding as well. Thus, this fact also should be considered while generating the WSDL using the Java2WSDL tool.

Once the WSDL has been generated, it can be stored in the UDDI registry along with its service related information. Then the service users can obtain the services according to their need.

The following command line instruction shows the usafe of the Java2WSDL tool such that it would generate a WeatherInfo.wsdl from the WeatherInfoJavaService class:

```
>Java@WSDL jws.ch5.WeatherInfoJavaService ---package-mapping
"jws.ch5=http://www.myweather.com/weatherInfo" –output-file-mapping
http://www.myweather.com/weatherInfo=WeatherInfo.wsdl –output-directory
jws/ch5
```

The output WeatherInfo. wsdl generated by the Java2WSDL tool is

```
<?xml version='1.0'?>
<wsdl:definitions name='jws.ch5.WeatherInfoJavaService'
targetNamespace='http://www.myweather.com/WeatherInfo'
xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
xmlns:tns='http://www.myweather.com/WeatherInfo'
xmlns:ns0='http://systinet.com/xsd/SchemaTypes/'
xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
xmlns:map='http://systinet.com/mapping/'>

    <wsdl:types>
        <xsd:schema elementFormDefault="qualified"
        targetNamespace=
        "http://systinet.com/xsd/SchemaTypes/"
        xmlns:tns="http://systinet.com/xsd/SchemaTypes/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

            <xsd:element name="sCountry" nillable="true"
```

```
            type="xsd:string"/>

            <xsd:element name="sZip" nillable="true"
            type="xsd:string"/>

            <xsd:element name="sInstance" nillable="true"
            type="xsd:string"/>




            <xsd:element name="float_res"
            type="xsd:float"/>

            <xsd:element name="sTemperature"
            nillable="true" type="xsd:string"/>
        </xsd:schema>
</wsdl:types>

<wsdl:message name=
'WeatherInfoJavaService_GetWeatherInfo_1_Request'>
        <wsdl:part name='sCountry' element='ns0:sCountry'/>
        <wsdl:part name='sZip' element='ns0:sZip'/>
        <wsdl:part name='sInstance' element='ns0:sInstance'/>
</wsdl:message>

<wsdl:message name=
'WeatherInfoJavaService_GetWeatherInfo_Response'>
        <wsdl:part name='response' element='ns0:float_res'/>
</wsdl:message>

<wsdl:message name
='WeatherInfoJavaService_SetWeatherInfo_Response'/>

<wsdl:message name=
'WeatherInfoJavaService_SetWeatherInfo_1_Request'>
        <wsdl:part name='sCountry' element='ns0:sCountry'/>
        <wsdl:part name='sZip' element='ns0:sZip'/>
        <wsdl:part name='sInstance' element='ns0:sInstance'/>
        <wsdl:part name='sTemperature'
        element='ns0:sTemperature'/>
</wsdl:message>

<wsdl:portType name='WeatherInfoJavaService'>
        <wsdl:operation name='GetWeatherInfo'
        parameterOrder='sCountry sZip sInstance'>
            <wsdl:input message=
```

```
            'tns:WeatherInfoJavaService_GetWeatherInfo_1_Request'/>

            <wsdl:output message=
            'tns:WeatherInfoJavaService_GetWeatherInfo_Response'/>

        </wsdl:operation>

        <wsdl:operation name='SetWeatherInfo'
        parameterOrder='sCountry sZip sInstance
        sTemperature'>

            <wsdl:input message=
            'tns:WeatherInfoJavaService_SetWeatherInfo_1_Request'/>

            <wsdl:output message=
            'tns:WeatherInfoJavaService_SetWeatherInfo_Response'/>
        </wsdl:operation>
    </wsdl:portType>

    <wsdl:binding name='WeatherInfoJavaService'
    type='tns:WeatherInfoJavaService'>

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

        <wsdl:operation name='GetWeatherInfo'>
        <map:java-operation name=
        'GetWeatherInfo' signature='KExq...'/>

        <soap:operation soapAction='_10'
        style='document'/>

            <wsdl:input>
                <soap:body use='literal'
                namespace='http://www.myweather.com/
                WeatherInfoWeatherInfoJavaService'/>
            </wsdl:input>

            <wsdl:output>
                <soap:body use='literal' namespace=
                'http://www.myweather.com/
                WeatherInfoWeatherInfoJavaService'/>
            </wsdl:output>
        </wsdl:operation>

        <wsdl:operation name='SetWeatherInfo'>
            <map:java-operation name='SetWeatherInfo'
            signature='KExq...'/>
```

```
            <soap:operation soapAction='_11'
            style='document'/>

            <wsdl:input>
                <soap:body use='literal' namespace=
                'http://www.myweather.com/
                WeatherInfoWeatherInfoJavaService'/>
            </wsdl:input>
                    .*
            <wsdl:output>
                <soap:body use='literal' namespace=
                'http://www.myweather.com/
                WeatherInfoWeatherInfoJavaService'/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

    <wsdl:service name='JavaService'>
        <wsdl:port name='WeatherInfoJavaService'
        binding='tns:WeatherInfoJavaService'>

            <soap:address location=
            'urn:unknown-location-uri'/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Listing: WeatherInfo.wsdl generated by the Java2WSDL tool

Generating Java Code From an Existing WSDL:

In this situation in which WSDL definitions are created before actually implementing a web service, the WSDLCompiler tool of WASP can be used to generate the skeleton of a java interface. A java class consisting of the actual method implementations then can imoplement this generated java interface.

The usage of the WSDLCompiler tool is as follows:

>WSDLCompiler WeatherInfo.wsdl

In this case, a java interface with the name WeatherInfoJavaService is created as follows:

```
/**
  */
Public interface WeatherInfoJavaService {/
        /**
          */
        Float GetWeatherInfo(java.lang.String sCountry, java.lang.String sZip, java.lang.String sInstance);
        /**
          */
        Void SetWeatherInfo(java.lang.String sCountry, java.lang.String sZip, java.lang.String sInstance, java.lang.String sTemperature);
}
/*
* Generated by WSDLCompiler, (c) 2002, Systinet Corp.
*                                    http://www.systinet.com
```

UNIT 4

SHORT ANSWER QUESTIONS

1) What is Service discovery?

Ans: Service discovery is the process of locating Web service providers, and retrieving Web services descriptions that have been previously published.

Interrogating services involve querying the service registry for Web services matching the needs of a service requestor.

- A query consists of search criteria such as:
  - ✓ The type of the desired service, preferred price and maximum number of returned results, and is executed against service information published by service provider.
- Discovering Web services is a process that is also dependent on the architecture of the service registry.

After the discovery process is complete, the service developer or client application should know the exact location of a Web service (URI), its capabilities, and how to interface with it

2) What are the types of service discovery?

Ans: There are two types of service discovery:

1) Static

- The service implementation details are bound at design time and a serviceretrieval is performed on a service registry.
- The results of the retrieval operation are examined usually by ahuman designer and the service description returned by theretrieval operation is incorporated into the application logic.

2) Dynamic

- The service implementation details are left unbound at design time so that they can be determined at run-time.

- The Web service requestor has to specify preferences to enable the application to infer/reason which Web service(s) the requester is most likely to want to invoke.
- Based on application logic quality of service considerations such as best price, performance, security certificates, and so on, theapplication chooses the most appropriate service, binds to it, andinvokes it

3) What is UDDI?

Ans: The universal description, discovery, and integration is aregistry standard for Web service description and discoverytogether with a registry facility that supports the WSpublishing and discovery processes.

UDDI enables a business to:

- Describe its business and its services;
- Discover other businesses that offer desired services;
- Integrate (interoperate)with these other businesses.

Conceptually, a UDDI business registration consists ofthree inter-related components:

- "White pages" (address, contact, and other key points of contact);
- "Yellow pages" classification info. based on standard industry taxonomies; and
- "Green pages", the technical capabilities and information about services

4) What are the uses of UDDI Registry?

Ans: Business 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, and contact information, unique identifiers such as D-U-N-S numbers or Tax IDs, or web services use UDDI as white pages.

Green pages level: Businesses that publish the technical information describing the behavior and supported functions on their web services make use of the UDDI registry as green pages.

Yellow pages level: businesses that intend to classify their information based on categorizations make use of the UDDI registry as yellow pages.

5) Write short notes on publishing API?

Ans: 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. 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 deines functions that deal with the authentication of the registry users, which is required inorder to successfully execute the rest of the functions of this API:

- <get_authToken>
- <discard_authToken>

6) What are the limitations of UDDI?

Ans: some of the limitations of UDDI are:

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

LONG ANSWER QUESTIONS

1) What is service discovery and what are the types of service discovery?

Ans: Service discovery is the process of locating Web service providers, and retrieving Web services descriptions that have been previously published.

Interrogating services involve querying the service registry for Web services matching the needs of a service requestor.

- A query consists of search criteria such as:
  - ✓ The type of the desired service, preferred price and maximum number of returned results, and is executed against service information published by service provider.
- Discovering Web services is a process that is also dependent on the architecture of the service registry.

After the discovery process is complete, the service developer or client application should know the exact location of a Web service (URI), its capabilities, and how to interface with it**.**

There are two types of service discovery:

1) Static

- The service implementation details are bound at design time and serviceretrieval is performed on a service registry.
- The results of the retrieval operation are examined usually by ahuman designer and the service description returned by theretrieval operation is incorporated into the application logic.

2) Dynamic

- The service implementation details are left unbound at design time so that they can be determined at run-time.
- The Web service requestor has to specify preferences to enable the application to infer/reason which Web service(s) the requester is most likely to want to invoke.

- Based on application logic quality of service considerations such as best price, performance, security certificates, and so on, theapplication chooses the most appropriate service, binds to it, andinvokes it.

2) What is the role of service discovery in SOA?

Ans: In **Service Oriented Architectures/Micro Service Architectures**, we are seeing an application built as a number of restful services which do one thing great. Since that one thing could be compute /any other resource intense, it makes sense to scale each service independently. From the end user's perspective he is interested in the end result, which is all of these services working together to deliver a result. So each service would end up calling one or many other services

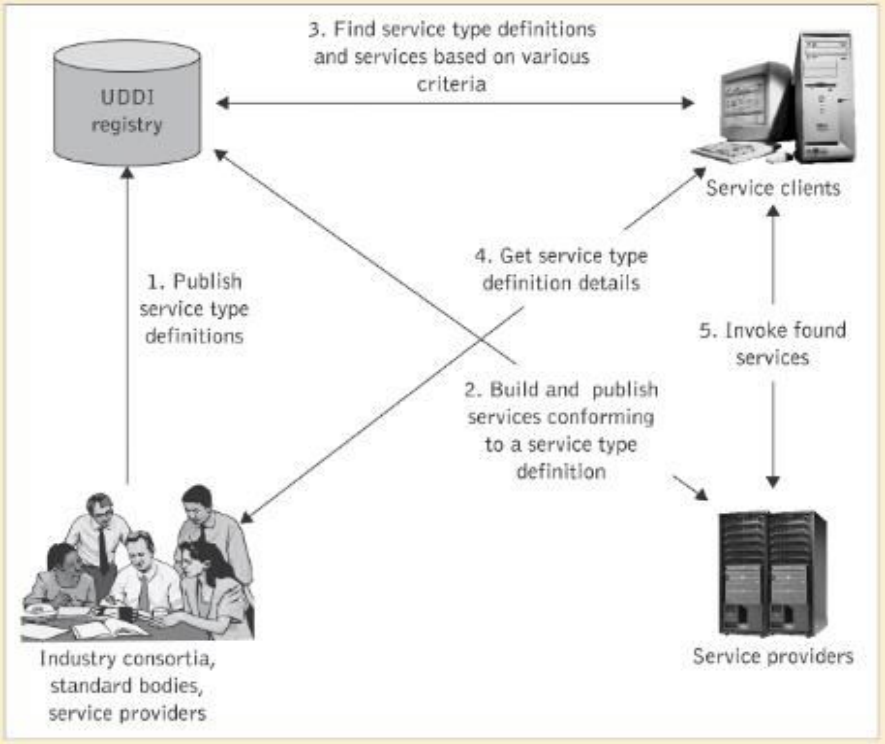
As a result, In **SOA/distributed systems**, services need to find each other. For example, a web service might need to find a caching service or another mid-tier component service etc. A Service Discovery system should provide a mechanism for:

- 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 or 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

3) Explain UDDI Usage Model and its characteristics?
Ans:

- An enterprise may set up multiple private UDDI registries in-house to support intranet and e-Business operations.
- Public UDDI registries can be set up by customers and business partners of an enterprise.
- Services must be published in a public UDDI registry so that potential clients and service developers can discover them.



UDDI Characteristics:

1) UDDI provides a mechanism to categorize businesses andservices using taxonomies
   - Service providers can use a taxonomy to indicate that a serviceimplements a specific domain standard, or that it provides services toa specific geographic area
   - UDDI uses standard taxonomies so that information can bediscovered on the basis of categorization.
2) UDDI business registration: an XML documentused todescribe a business entity and its Web services.
3) UDDI is not bound to any technology. In other words,

- An entry in the UDDI registry can contain any type of resource,independently of whether the resource is XML based or not, e.g., theUDDI registry could contain information about an enterprise'selectronic document interchange (EDI) system or even a service that,uses the fax machineas its primary communication channel.
- While UDDI itself uses XML to represent the data it stores, it allowsfor other kinds of technology to be registered.

4) Explain UDDI Registries?

Ans: 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 etc.,

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.

A UDDI registry can be operated in two modes: public mode and private mode.

Public UDDI Registries:

A public UDDI registry is avaliable for everyone to publish/query the business and service information on the internet.

Such public registries can be 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.

Private UDDI Registeries:

A private UDDI registry is operated by a single organization or a group of collaborating organizations to share the information that would be available 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.

Interacting with a UDDI registry

Typically, vendorsimplementating a UDDI registry provides two ways of interacting with a UDDI registry service.

1) A graphical user interface (GUI), for implementing with a UDDI registry. These GUIs also can be browser-based. The following are the list of public UDDI registries, operated by various companies such as Microsoft, IBM, etc.,that provide a browser-based interface to these registries:
   - http://uddi.rte.microsoft.com/search/frames.aspx
   - http://www-3.ibm.com/services/uddi/v2beta/protect/registry.html
   - http://uddi.hp.com/uddi/index.jsp
   - http://udditest.sap.com/
   - http://www.systinet.com/uddi/web
2) A programmatic interface for communicating with the UDDI registry. These programmatic interfaces are based on SOAP, because the UDDI registry supports SOAP as the communication protocol.
   - Most of the vendors providing the UDDI registry implementations support both of these types of access to the registry.

5) Explain programming with UDDI?

Ans: UDDI programming API: The UDDI specification defines two XML-based programming APIs for communicating with the UDDI registry node: inquiry API and publishing API.

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. 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 following 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. 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 deines functions that deal with the authentication of the registry users, which is required inorder to successfully execute the rest of the functions of this API:

- <get_authToken>
- <discard_authToken>

6) Explain about UDDI Data Structures?

Ans: 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. This UDDI data structures are acts as the input and output parameters for the UDDI programming API.

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

1) <businessEntity>
2) <publisherAssertion>
3) <businessService>
4) <bindingTemplate>
5) <tModel>

Here all the data structures except <publisherAssertion> are identified and referenced through 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.

1) **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.
   The generic white and yellow pages information about a serviceprovider is stored in the <businessEntity> data structure.

Here is an example of a fictitious business's UDDI registry entry:

```
<businessEntitybusinessKey="uuid:C0E6D5A8-C446-4f01-99DA-
70E212685A40"
  operator="http://www.ibm.com" authorizedName="John Doe">
<name>Acme Company</name>
<description>
    We create cool Web services
</description>
<contacts>
<contact useType="general info">
<description>General Information</description>
<personName>John Doe</personName>
<phone>(123) 123-1234</phone>
<email>jdoe@acme.com</email>
</contact>
</contacts>

<businessServices>
    ...
</businessServices>
<identifierBag>
<keyedReferencetModelKey="UUID:8609C81E-EE1F-4D5A-B202-
3EB13AD01823"  name="D-U-N-S" value="123456789" />
</identifierBag>

<categoryBag>
```

<keyedReferencetModelKey="UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2" name="NAICS" value="111336" />
</categoryBag>
</businessEntity>

2) publisherAssertion Data Structure

This is a relationship structure putting into association two or more businessEntity structures according to a specific type of relationship, such as subsidiary or department.

The publisherAssertion structure consists of the three elements: fromKey (the first businessKey), toKey (the second businessKey), and keyedReference.

The keyedReference designates the asserted relationship type in terms of a keyNamekeyValue pair within a tModel, uniquely referenced by a tModelKey.

<element name="publisherAssertion" type="uddi:publisherAssertion" />
<complexType name="publisherAssertion">
<sequence>
<element ref="uddi:fromKey" />
<element ref="uddi:toKey" />
<element ref="uddi:keyedReference" />
</sequence>
</complexType>

3) businessService Data Structure

The business service structure represents an individual web service provided by the business entity. Its description includes information on how to bind to the web service, what type of web service it is, and what taxonomical categories it belongs to.

Here is an example of a business service structure for the Hello World web service.

<businessServiceserviceKey="uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
businessKey="uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">

<name>Hello World Web Service</name>
<description>A friendly Web service</description>
<bindingTemplates>
...
</bindingTemplates>
<categoryBag />
</businessService>

4) bindingTemplate Data Structure

Binding templates are the technical descriptions of the web services represented by the business service structure. A single business service may have multiple binding templates. The binding template represents the actual implementation of the web service.

Here is an example of a binding template for Hello World.

<bindingTemplateserviceKey="uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
bindingKey="uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">

<description>Hello World SOAP Binding</description>
<accessPointURLType="http">http://localhost:8080</accessPoint>
<tModelInstanceDetails>
<tModelInstanceInfotModelKey="uuid:EB1B645F-CF2F-491f-811A-4868705F5904">
<instanceDetails>
<overviewDoc>
<description>
references the description of the WSDL service definition
</description>

<overviewURL>
http://localhost/helloworld.wsdl
</overviewURL>
</overviewDoc>
</instanceDetails>
</tModelInstanceInfo>

</tModelInstanceDetails>
</bindingTemplate>

As a business service may have multiple binding templates, the service may specify different implementations of the same service, each bound to a different set of protocols or a different network address.

5) tModel Data Structure

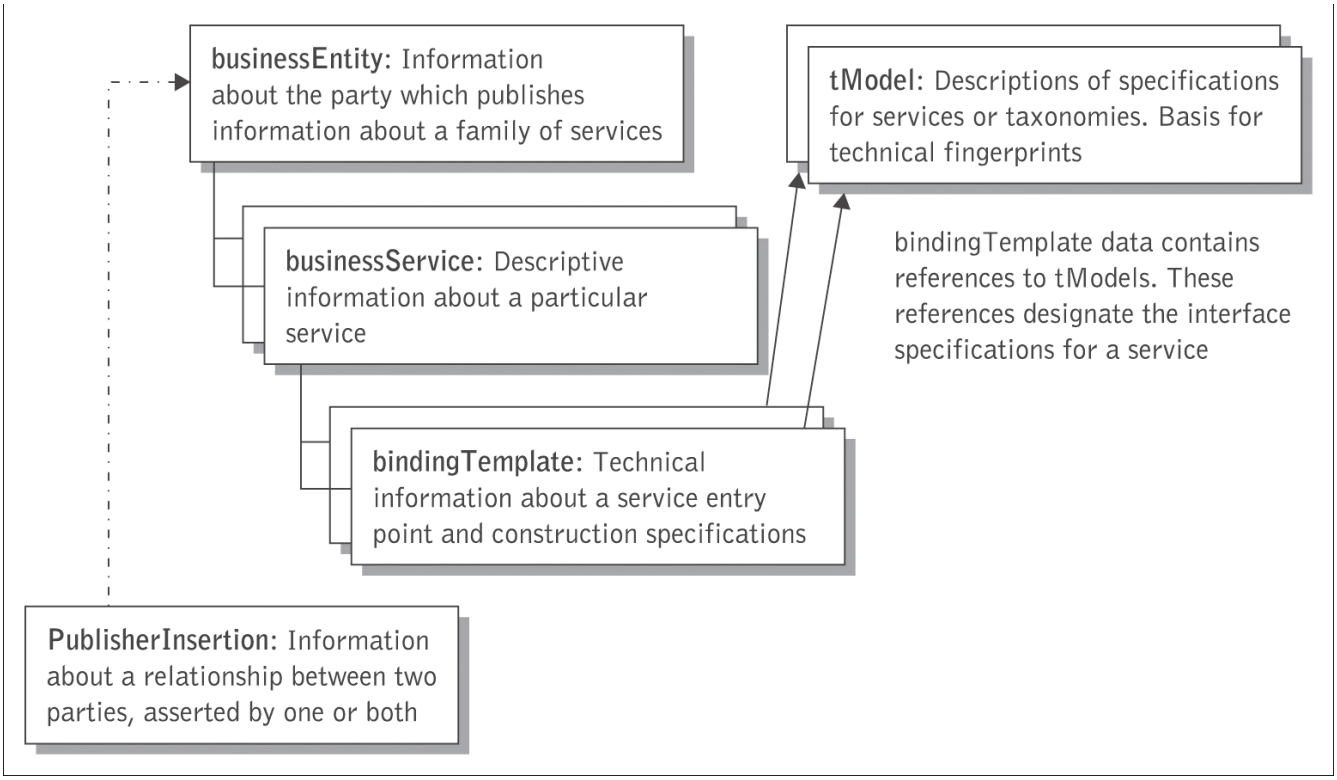
tModel is the last core data type, but potentially the most difficult to grasp. tModel stands for technical model.

tModel is a way of describing the various business, service, and template structures stored within the UDDI registry. Any abstract concept can be registered within the UDDI as a tModel. For instance, if you define a new WSDL port type, you can define a tModel that represents that port type within the UDDI. Then, you can specify that a given business service implements that port type by associating the tModel with one of that business service's binding templates.

Here is an example of a tModel representing the Hello World Interface port type.

```
<tModeltModelKey="uuid:xyz987..." operator="http://www.ibm.com"
authorizedName="John Doe">
<name>HelloWorldInterface Port Type</name>
<description>
    An interface for a friendly Web service
</description>

<overviewDoc>
<overviewURL>
      http://localhost/helloworld.wsdl
</overviewURL>
</overviewDoc>
</tModel>
```



7). Explain the Publishing API?

Ans: XML messages that perform the functionality of adding/modifying/deleting information from a UDDI registry. Publishing a UDDI registry requires an authenticated access. UDDI specification does not define authentication mechanisms and hence, authentication is dependent upon the implementations of UDDI.

List of publishing API functions:

get_authToken : Retrieves an authorization token. All of the Publisher interface operations require that a valid authorization token be submitted with the request.

discard_authToken : Tells the UDDI registry to no longer accept a given authorization token. This step is equivalent to logging out of the system.

save_business : Creates or updates a business entity's information contained in the UDDI registry.

save_service : Creates or updates information about the web services that a business entity provides.

save_binding : Creates or updates the technical information about a web service's implementation.

save_tModel : Creates or updates the registration of abstract concepts managed by the UDDI registry.

delete_business : Removes the given business entities from the UDDI registry completely.

delete_service : Removes the given web services from the UDDI registry completely.

delete_binding : Removes the given web services technical details from the UDDI registry.

delete_tModel: Removes the specified tModels from the UDDI registry.

get_registeredInfo : Returns a summary of everything the UDDI registry is currently keeping track of for the user, including all businesses, all services, and all tModels.

set_publisherAssertions : Manages all of the tracked relationship assertions associated with an individual publisher account.

add_publisherAssertions : Causes one or more publisherAssertions to be added to an individual publisher's assertion collection.

delete_publisherAssertions : Causes one or more publisherAssertion elements to be removed from a publisher's assertion collection.

get_assertionStatusReport : Provides administrative support for determining the status of current and outstanding publisher assertions that involve any of the business registrations managed by the individual publisher account.

get_publisherAssertions : Obtains the full set of publisher assertions that is associated with an individual publisher account.

8) Explain Inquiry API?

Ans: 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. 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.

List of Inquiry API:

find_binding : Returns a list of web services that match a particular set of criteria based on the technical binding information.

find_business : Returns a list of business entities that match a particular set of criteria.

find_ltservice : Returns a list of web services that match a particular set of criteria.

find_tModel : Returns a list of tModels that match a particular set of criteria.

get_bindingDetail : Returns the complete registration information for a particular web service binding template.

get_businessDetail : Returns the registration information for a business entity, including all services that entity provides.

get_businessDetailExt : Returns the complete registration information for a business entity.

get_serviceDetail : Returns the complete registration information for a web service.

get_tModelDetail : Returns the complete registration information for a tModel.

find_relatedBusinesses : Discovers businesses that have been related via the uddi-org:relationships model.

9) Explain in detail about publishing information to a UDDI Registry?

Ans: 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.ut4.SubmitBusiness class, and hence, its implementation is of most interest to us. The following are the steps of doSubmit():

1. Construct the UDDIApiPublishing object. This is the object that we will use to actually publish to the registry.
2. 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.
3. 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.
4. 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.
5. 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.
6. After the publishing operation has been executed, discard the authentication token. Finally, check whether the publishing operation was successful or not.

SubmitBusiness.java Source Code

```
package jws.ut4;

import org.idoox.uddi.client.api.v2.*;
import org.idoox.uddi.client.api.v2.request.publishing.*;
import org.idoox.uddi.client.structure.v2.business.*;
import org.idoox.uddi.client.structure.v2.base.*;
import org.idoox.uddi.client.api.v2.response.*;
importorg.idoox.uddi.*;

public class SubmitBusiness
{
```

```
public static void main(String args[]) throws Exception
     {
          // Call the method in order to submit new business to
          // the public registry hosted by Systinet.

doSubmit();
     }

public static void doSubmit() throws Exception
     {
          String sBusinessName = "ACME Computer Services";
          String sBusinessDescription = "Provides professional
services in the areas of Computer Software";

System.out.println("Saving business with the
following details:");

System.out.println("Name: " + sBusinessName);

System.out.println("Description: " +
sBusinessDescription);

          // Get hold of the UDDI Publishing API
          // Note our usage of Publishing URL for the Systinet
          // UDDI Registry

UDDIApiPublishingobjUDDIApiPublishing =
UDDILookup.getPublishing("https://www.systinet.com:443
          /wasp/uddi/publishing/");

          // First we get hold of Authentication token from the
          // Registry so that we can publish to the UDDI
          // Registry. Note that registered a user in Systinet
          // Public Registry with registry_user ID and
          // registry_user password.

AuthTokenobjAuthToken  =objUDDIApiPublishing.
```

```
get_authToken (new GetAuthToken(new
UserID("registry_user"), new Cred("registry_user")));

          // Create the BusinessEntity Structure
BusinessEntityobjBusinessEntity =
newBusinessEntity();

          // Set the empty businessKey since we are creating a
          // new business
objBusinessEntity.setBusinessKey
           (newBusinessKey(""));

          // Set the name of the business
objBusinessEntity.addName(new Name(sBusinessName));

          // Set the description of the business
objBusinessEntity.addDescription
           (new Description(sBusinessDescription));

          // Get hold of the SaveBusiness interface
SaveBusinessobjSaveBusiness = new SaveBusiness();

          // Set the Authentication Information on SaveBusiness
objSaveBusiness.setAuthInfo
          (objAuthToken.getAuthInfo());

          // Now add the BusinessEntity to save to the
          // SaveBusiness interface
objSaveBusiness.addBusinessEntity(objBusinessEntity);

          // Finally publish the SaveBusiness object to the
          // registry
BusinessDetailobjBusinessDetail =
objUDDIApiPublishing.save_business(objSaveBusiness);

          // Discard the Authentication token now
objUDDIApiPublishing.discard_authToken
```

```
            (newDiscardAuthToken(objAuthToken.getAuthInfo()));

          // See if the Business has been published
          // successfully
if (objBusinessDetail==null)
          {
System.err.println("\nUnsuccessful in
submitting the new business information to
registry.");
          }
else
          {
System.err.println("\nSuccessful in submitting
the new business information to registry.");
          }
return;
     }
}
```

10) Explain in detail about searching information in a UDDI registry?

Ans:Searching Information in a UDDI Registry

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.ut4.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.

SearchBusiness.java Source Code

```
package jws.ut4;

import org.idoox.uddi.client.api.v2.request.inquiry.*;
import org.idoox.uddi.client.structure.v2.tmodel.*;
import org.idoox.uddi.client.structure.v2.base.*;

import org.idoox.uddi.client.api.v2.response.*;
import org.idoox.uddi.client.structure.v2.business.*;
import org.idoox.uddi.client.api.v2.*;

importorg.idoox.uddi.*;

public class SearchBusiness
{
public static void main(String args[]) throws Exception
     {
if (args.length != 1)
          {
printUsage();
          }
else
          {
               String sNameOfBusiness=  args[0];

               // Invoke the search operation
doSearch(sNameOfBusiness);
          }
     }
```

```
private static void printUsage()
     {
System.err.println("\nUsage: java
          jws.ch5.SearchBusiness <BusinessNamePattern>");

System.err.println("\nwhere<BusinessNamePattern>
represents name of the business you want to
search.");
     }

public static void doSearch(String sNameOfBusiness) throws
     Exception
     {
          // Create a FindBusiness object
FindBusinessobjFindBusiness = new FindBusiness();

          // Send the find criteria
objFindBusiness.addName(new Name(sNameOfBusiness));

          // Set the maximum number of rows to return
objFindBusiness.setMaxRows(new MaxRows("10"));

          // Get hold  ofUDDILookup object to place the query
UDDIApiInquiryobjUDDIApiInquiry =
UDDILookup.getInquiry("http://www.systinet.com:80/
wasp/uddi/inquiry/");

          // Invoke the query on the UDDI Inquiry API
BusinessListobjBusinessList=
objUDDIApiInquiry.find_business(objFindBusiness);

          // Check whether anything was found matching the
          // criteria
if (objBusinessList==null)
          {
System.err.println("No businesses were found
matching the criteria.");
```

```
          }
else
          {
               // Get hold of the BusinessInfo objects,
               // contained by BusinessList
BusinessInfosobjBusinessInfos =
objBusinessList.getBusinessInfos();

System.out.println("\n" +
objBusinessInfos.size() + " businesses found
matching the criteria...\n");

BusinessInfoobjBusinessInfo =
objBusinessInfos.getFirst();

BusinessKeyobjBusinessKey;

if (objBusinessInfo != null)
               {
objBusinessKey=objBusinessInfo.
getBusinessKey();

                    // Traverse through the results.
while (objBusinessInfo!=null)
                    {
System.out.println("Business Name =
" +objBusinessInfo.getNames().
getFirst().getValue());

System.out.println("Business UUID = " +
objBusinessInfo.getBusinessKey());

System.out.println("----------------
                         ----------------------------------");

                         // Next BusinessInfo
objBusinessInfo =
```

```
objBusinessInfos.getNext();
                        }
                    }
                }
            }
}
```

11) Explain in detail about deleting information from a UDDI registry?

Ans: 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. 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():

1. Construct the UDDIApiPublishing object. This is the object that we would use to actually delete information from the registry.
2. 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.
3. 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.
4. 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.
5. Check the DispositionReport object to see if this operation was a success or a failure.

DeleteBusiness.java Source Code

```
package jws.ut4;

importorg.idoox.uddi.*;
import org.idoox.uddi.client.api.v2.*;
import org.idoox.uddi.client.api.v2.request.publishing.*;
import org.idoox.uddi.client.api.v2.response.*;
import org.idoox.uddi.client.structure.v2.business.*;

public class DeleteBusiness
{
public static void main(String args[]) throws Exception
     {
if (args.length != 1)
          {
printUsage();
          }
else
          {
BusinessKeyobjBusinessKey =
newBusinessKey (args[0]);

doDelete(objBusinessKey);
          }
     }

private static void printUsage()
     {
System.err.println("\nUsage: java
          jws.ch5.DeleteBusiness <BusinessKey>");

System.err.println("\nwhere<BusinessKey> is a string
representation of UUID corresponding to Business you
want to delete.");
     }

public static void doDelete(BusinessKeyobjBusinessKey)
```

```
throws Exception
     {
System.out.println("\nDeleting Business with Key: ");
System.out.println(objBusinessKey.toString());

UDDIApiPublishingobjUDDIApiPublishing =
UDDILookup.getPublishing("
          https://www.systinet.com:443/wasp/uddi/publishing/");

          // First we get hold of Authentication token from the
          // Registry so that we can delete
          // business from the UDDI Registry. Note that
          // registered a user in SystinetPublich Registry
          // with registry_user ID and registry_user password.

AuthTokenobjAuthToken = objUDDIApiPublishing.
get_authToken(new GetAuthToken(
newUserID("registry_user"),
new Cred("registry_user")));

          // Now get hold of the DeleteBusiness structure
org.idoox.uddi.client.api.v2.request.
publishing.DeleteBusinessobjDeleteBusiness =
new org.idoox.uddi.client.api.v2.request.
publishing.DeleteBusiness();

          // Set the login information on DeleteBusiness
objDeleteBusiness.setAuthInfo
          (objAuthToken.getAuthInfo());

          // Add business to delete to the DeleteBusiness
          // Structure
objDeleteBusiness.addBusinessKey(objBusinessKey);

          // Call Publishing API method delete_business
DispositionReportobjDispositionReport =
objUDDIApiPublishing.delete_business
```

```
          (objDeleteBusiness);

          // Discard the Authentication token now
objUDDIApiPublishing.discard_authToken
          (newDiscardAuthToken(objAuthToken.getAuthInfo()));

          // Check to see if the delete operation was
          // successful
if (objDispositionReport == null)
          {
System.err.println("Unsuccessful in deleting
the business information from the registry.");
          }
else
          {
if (objDispositionReport.
resultIs(UDDIErrorCodes.E_SUCCESS))
               {
System.out.println("\nSuccessful in
deleting the business information from the
registry.");
               }
else
               {
System.out.println("\nUnsuccessful in
deleting the business information due to
following reason(s):");

System.out.println(
objDispositionReport.toXML());
               }
          }
     }

}
```

12) Describe service discovery mechanism?

Ans:Mechanisms for Web Service Discovery based on Centralized Approach :

1) Quality-driven centralized discovery approach using Web service broker (WSB). In this model, service providers publish service information in the UDDI or search engines. WSB performs the tasks of gathering web services spread over the web and monitoring their behavior based on various QWS metrics automatically without needing any human intervention. Its interface allows clients to express appropriate service queries based on QWS. As clients receive response, they can invoke services.
2) Web Service Crawler Engine (WSCE), a crawler that can capture service information from various accessible resources over the Web like UBR, search engines and service portals, to help in searching of Web services on the Web.
3) METEOR-S Web Service Discovery Infrastructure(MWSDI), an ontology based infrastructure to provide access to private and public registries divided based on business domains and grouped into federations for enhancing the discovery process. METEOR-S provides a discovery mechanism for publishing Web services over a federated registry sources but, similar to the centralized registry environment, it does not provide any means for advanced search techniques which are essential for locating appropriate business applications. In addition, having a federated registry environment can potentially provide inconsistent policies to be employed which will significantly have an impact on the practicability of conducting inquiries across the federated environment and can at the same time significantly affect the productiveness of discovering Web services in a real-time manner across multiple registries.
4) Discovering Web services through a centralized UDDI registry. Centralized registries can provide effective methods for the Web service discovery, but they suffer from problems associated with having centralized systems such as a single point of failure, and delayed delivery of notifying updated service description. Other issues relating to the scalability of data replication, and handling versioning of services from the same provider are driving researchers to find other alternatives.

Mechanisms for Web Service Discovery based on Decentralized Approach :

1) Web Service Relevancy Function (WsRF) used for measuring the relevancy ranking of a particular Web service based on QoS metrics and client preferences for the purpose of finding the best available Web service during Web services' discovery process based on a set of given client QoS preferences.
2) structured peer-to-peer framework for web service discovery in which web services are located based on both service functionality and process behavior. In addition, they integrate a scalable reputation model in this distributed peer-to-peer framework to rank web services based on both trust and service quality.
3) Semantic Web Service discovery that splits the task of finding the most appropriate offer from available web services into static discovery and then dynamic offer discovery. Initially it will find all available web services fulfilling the user's goal, by interacting with discovered services, find offers relevant for the goal, filter out the offers which do not meet the user's constraints, rank the offers according to user's preferences and choose one to be invoked and then use the selected service.

UNIT 5

Short Answer Questions

1) Define interoperability?

Ans: Interoperability is a characteristic of a product or system, whose interfaces are completely understood, to work with other products or systems, present or future, in either implementation or access, without any restrictions.

Interoperability describes the extent to which systems and devices can exchange data, and interpret that shared data. For two systems to be interoperable, they must be able to exchange data and subsequently present that data such that it can be understood by a user.

2) What is Common language runtime (CLR)

Ans: The CLR provides a managed runtime environment for the .NET Framework. CLR enables applications to install and execute code, and it provides services such as memory management, including garbage collection, threading, exception handling, deployment support, application runtime security, versioning, etc.

CLR provides a set of just-in-time (JIT) compilers, which compile MSIL to produce native code specific to the target system. CLR defines a set of rules as commom type system (CTS) and common language system (CLS) that specifies the .NET supported languages required to use for developing compilers supporting a .NET platform.

3) What is Common Language System?

Ans:A **Common Language Specification** (**CLS**) is a document that says how computer programs can be turned into MSIL code. When several languages use the same bytecode, different parts of a program can be written in different languages. Microsoft uses a **Common Language Specification** for their .NET Framework.

Common language system (CLS) that specifies the .NET supported languages required to use for developing compilers supporting a .NET platform.

4) What is Common Type System?

Ans: The **Common Type System** (**CTS**) is a standard that specifies how type definitions and specific values of types are represented in computer memory. It is intended to allow programs written in different programming languages to easily share information. As used in programming languages, a type can be described as a definition of a set of values (for example, "all integers between 0 and 10"), and the allowable operations on those values (for example, addition and subtraction).

5) What is common language Specification?

Ans: The Common Language Specification (CLS) is a fundamental set of language features supported by the Common Language Runtime (CLR) of the .NET Framework. CLS is a part of the specifications of the .NET Framework. CLS was designed to support language constructs commonly used by developers and to produce verifiable code, which allows all CLS-compliant languages to ensure the type safety of code. CLS includes features common to many object-oriented programming languages. It forms a subset of the functionality of common type system (CTS) and has more rules than defined in CTS.

6) What is .NET framework class library?

Ans: The .NET framework class library acts as the base class library of the .NET framework. It provides the collection of classes and type system as foundation classes for .NET to facilitate CLR. The class libraries are reusable object-oriented classes that support tasks like establishing database connectivity, data collection, file access etc.

7) What are the goals of cryptography?

Ans: In essence, cryptography concerns four main goals. They are:

1. message**confidentiality** (or privacy): Only an authorized recipient should be able to extract the contents of the message from its encrypted form. Resulting from steps to hide, stop or delay free access to the encrypted information.

2. message**integrity**: The recipient should be able to determine if the message has been altered.
3. sender**authentication**: The recipient should be able to verify from the message, the identity of the sender, the origin or the path it traveled (or combinations) so to validate claims from emitter or to validated the recipient expectations.
4. sender**non-repudiation**: The emitter should not be able to deny sending the message.

Long Answer Questions

1) What is the means of ensuring interoperability?

Ans: In a web service environment, the SOAP is a standard communication model. This protocol provides conventions for representing data and application interaction models like RPCs and messaging. This facilitates inter-application communication and seamless data sharing among applications residing on a network, regardless of their language, OS, hardware platforms.

It also enables the development of compatible web services by introducing interoperability among business applications running over a network of devices.

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. It is caused by the dependencies of the 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 conforming and non-conforming SOAP implementations.

Different ways exists to ensure service requestor interoperability with the service providers. Some of them are:

1) Declaring W3C XML schemas

Defining W3C XML schema definitions (XSD) and target namespaces for all the applications data types and having a supporting SOAP implementation for both the

service provider and service requestor resolves almost all interoperability issues specific to the data types. This helps to create compliant SOAP proxy-based clients for the service requestors with all the defined data types by providing automatic encoding and mapping for the service provider-dispensed XSD data types.

2) Exposing WSDL

Most web services platforms and SOAP implementations provide this as an automatic mechanism by delivering WSDL for all its exposed services. This exposed WSDL defines the service provider information and service specific parameters required by a service requestor for invoking the services, which enables the building service clients to interact with the service provider, thus ensuring interoperability based on the WSDL. The service clients also can be dynamically generated from a service provider's WSDL lookup.

3) Creating SOAP Proxies

For a web service, the client SOAP proxies can be created manually or can be generated dynamically based on the WSDL provided details of the service provider. In the automatic generation of SOAP proxies, sometimes they may throw SOAP faults during service invocation and may require some modifications in the SOAP headers or the encoded RPC calls. In the most cases this problem occurs due to non-conforming WSDL and SOAP implementation in the infrastructure of the service provider or requestor.

4) Testing interoperability

To ensure that interoperability between the service provider and requestor exists, the SOAP implementation also can be tested. In that case the SOAP implementations also tested. In that case, the SOAP implementation of the service provider and the requestor must agree and conform to the following SOAP-specific dependencies:

- The defined SOAP transport protocol bindings
- The supported version of SOAP
- The version of WSDL from the service provider and its ability to support by the service requestor client

- The version of W3C XML schema supported by the SOAP message, especially the SOAP envelope and its body elements

2) Explain the Overview of .NET framework in web services?

Ans: Microsoft .NET is part of the Microsoft .NET platform. Microsoft uses XML web services for developing distributed applications.

This provides a development environment for developing XML web services in Microsoft windows based environment.

The .NET framework provides the infrastructure for defining the overall .NET platform. Microsoft provides .NET compilers that generate a new code referred to as Microsoft intermediate language (MSIL).

The .NET compilers provided by Microsoft are as follows:

- VB.NET
- C++.NET
- ASP.NET
- C#.NET
- Jscript

The Microsoft .NET framework consists of two core components:

Common language runtime (CLR)

The CLR provides a managed runtime environment for the .NET Framework. CLR enables applications to install and execute code, and it provides services such as memory management, including garbage collection, threading, exception handling, deployment support, application runtime security, versioning, etc.

CLR provides a set of just-in-time (JIT) compilers, which compile MSIL to produce native code specific to the target system. CLR defines a set of rules as commom type system (CTS) and common language system (CLS) that specifies the .NET supported languages required to use for developing compilers supporting a .NET platform.

.NET framework class library

The .NET framework class library acts as the base class library of the .NET framework. It provides the collection of classes and type system as foundation classes for .NET to facilitate CLR. The class libraries are reusable object-oriented classes that supports tasks like estabilishing database connectivity, data collection, file access etc.

The class libraries support the develepment of software applications such as:

- Console applications
- Windows GUI applications
- Windows services
- ASP.NET applications
- .NET XML web services
- .NET Scripting applications
- .NET client applications

The .NET class libraries can work with CLS and can use CLR.

3) What are the challenges in creating web services interoperability?

Ans: There are many platforms including SOAP implementations are available to provide web services support for a variety of languages, APIs, applications and systems.

But not all the services exposed from these SOAP implementations are guaranteed to interoperate and run across disparate applications and systems.

Most interoperability problems occur in RPC-based web services because of the type mapping issues between the service provider and requestor, which are due to lack of type mapping support in SOAP processing.

In message based web services this is not the case, as the SOAP body is represented with an XML document.

The challenges that affect interoperability in web services:

1) Common SOAP/HTTP Transport Issues

At the core of web services, the transport protocols estabilish the communication and enable the services to send and receive messages. In case of using HTTP protocol, if the service provider requires a SOAPAction with a null value, most HTTP clients may not be able to provide a SOAPAction with a null value. The possible solutions are to fix the service client APIs and to ensure that certain service provider implementations require SOAPAction with a null value. To solve these problems, test the client to see if they can handle those scenarios.

2) XML schema and XML related issues

XML schema validation handling causes a lot of interoperability issues among SOAP implementations. So defining data types using XMS schema definitions must occur in both the service client and provider implementations.

Some SOAP implementations specify the encoding of data as UTF-8 and UTF-16 in the content-type header as

Content-Type: text/xml; charset=utf-8

And some implementations do not specify the charset in the content Type header, which causes some SOAP implementations to be unable to process messages. To correct this problem, ensure that the SOAP implementation and its encoding standards are compliant with the W3C specifications.

3) SOAP/XML message discontinuities

Message discontinuities cause a major problem between a SOAP implementation in fulfilling the request and response between the service client and service provider. To overcome these issues, ensure that the application is aware of the message discontinuities and throw SOAPFaults in the case of missing elements in the SOAPBody of the request message.

4) Version and Compatibility

The version of the supported XMLSchema, and the SOAP and WSDL specifications, and its compatibility between SOAP implementations affect interoperability. To ensure that these issues are handled, the web services platform providers and its SOAP implementations must be tested for the compatible versions of XML Schema definitions and the SOAP and WSDL specifications.

5) Explain the Goals of cryptography?

Ans: cryptography is the art of secret writing, the enciphering and deciphering of messages in secret code, as many would put it.

Goals of cryptography are

1) Confidentiality:

It deals with ensuring that only authorized parties are able to understand the data. Unauthorized parties may know that the data exists, but they should not be able to understand what the data is. When the data is transmitted on the wire, unauthorized parties can view the data by sniffing in between.

Confidentiality made possible through encryption. Encryption is the process of converting a particular message into scrambled text also known as ciphertext, by applying some cryptographic algorithms and secret information.

2) Authentication:

It ensures the identity of the party in a given security domain. Usually it involves having some sort of password or key through which the user would prove his or her identity in a particular security domain.

It is important for services to be able to tell to whom all they are providing their services. Likewise, it important for consumers of services, so that they know exactly with whom they are interacting.

Many approaches are for authentication among them either keys, digital certificates or passwords are famous.

3) Integrity:

Integrity is about protecting sensitive information from unauthorized modifications. In the case of a message being transmitted on the wire, integrity ensures that the message received by the recipient was the same message that was sent originally by the sender that the message has not been tampered with since it was sent.

Different hashing algorithms are used to generate a sort of a checksum to guarantee integrity.

4) Non-repudiation:

Repudiation is to refuse to accept something. Non-repudiation is a technique in which one party ensures that another party cannot repudiate, or cannot refuse to accept, a certain act. Non-repudiation forms the basis of electronic commerce

EX: a supplier of raw materials would want to ensure that the customer does not repudiate later its placing of an order for materials.

Digital signatures can be used to provide non-repudiation in computer security systems.

6) Explain digital signatures?

Ans:A digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document.

The digital equivalent of a handwritten signature or stamped seal, but offering far more inherent security, a digital signature is intended to solve the problem of tampering and impersonation in digital communications. Digital signatures can provide the added assurances of evidence to origin, identity and status of an electronic document, transaction or message, as well as acknowledging informed consent by the signer.

In many countries, including the United States, digital signatures have the same legal significance as the more traditional forms of signed documents. The United States Government Printing Office publishes electronic versions of the budget, public and private laws, and congressional bills with digital signatures.

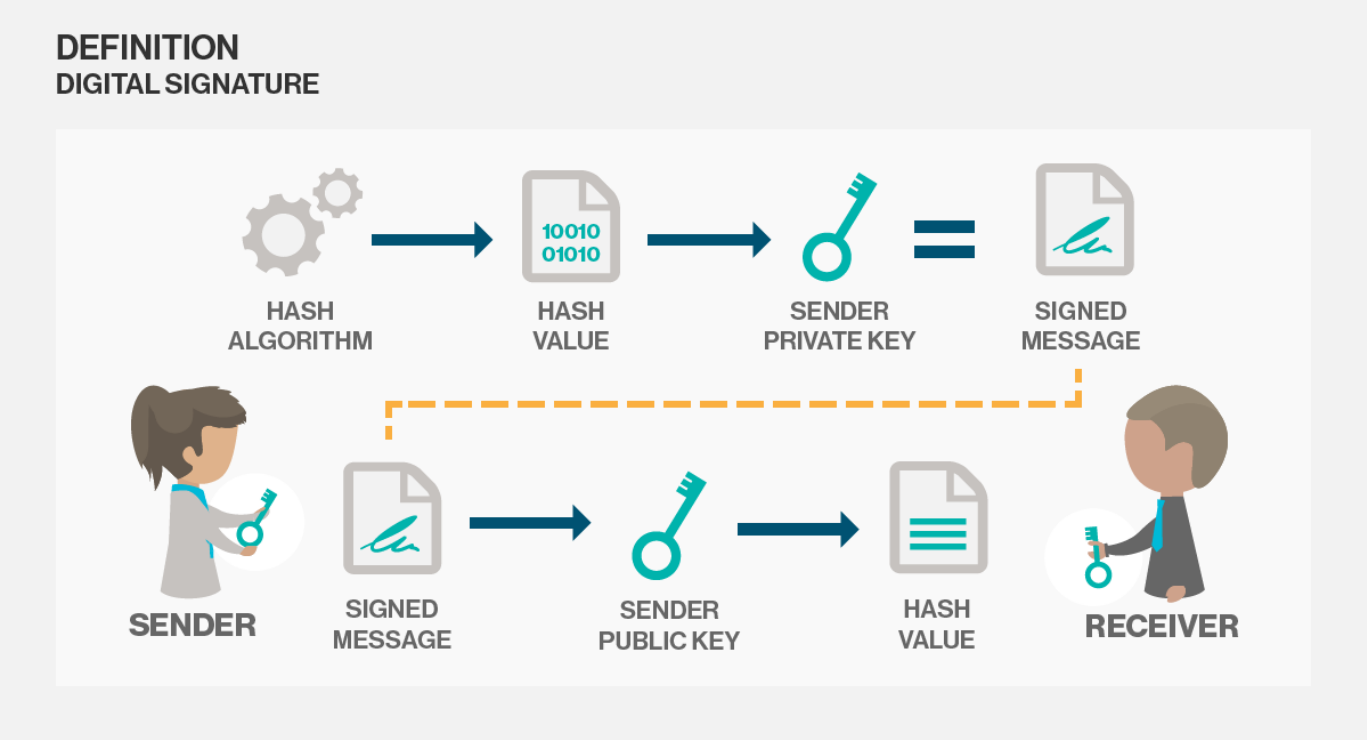
How digital signatures work

Digital signatures are based on public key cryptography, also known as asymmetric cryptography. Using a public keyalgorithm such as RSA, one can generate two keys that are mathematically linked: one private and one public.To create a digital signature, signing software (such as an email program) creates a one-way hash of the electronic data to be signed. The private key is then used to encrypt the hash. The encrypted hash -- along with other information, such as the hashing algorithm -- is the digital signature. The reason for encrypting the hash

instead of the entire message or document is that a hash function can convert an arbitrary input into a fixed length value, which is usually much shorter. This saves time since hashing is much faster than signing.

The value of the hash is unique to the hashed data. Any change in the data, even changing or deleting a single character, results in a different value. This attribute enables others to validate the integrity of the data by using the signer's public key to decrypt the hash. If the decrypted hash matches a second computed hash of the same data, it proves that the data hasn't changed since it was signed. If the two hashes don't match, the data has either been tampered with in some way (integrity) or the signature was created with a private key that doesn't correspond to the public key presented by the signer (authentication).

A digital signature can be used with any kind of message -- whether it is encrypted or not -- simply so the receiver can be sure of the sender's identity and that the message arrived intact. Digital signatures make it difficult for the signer to deny having signed something (non-repudiation) -- assuming their private key has not been compromised -- as the digital signature is unique to both the document and the signer, and it binds them together. A digital certificate, an electronic document that contains the digital signature of the certificate-issuing authority, binds together a public key with an identity and can be used to verify a public key belongs to a particular person or entity.



Most modern email programs support the use of digital signatures and digital certificates, making it easy to sign any outgoing emails and validate digitally signed incoming messages. Digital signatures are also used extensively to provide proof of authenticity, data integrity and non-repudiation of communications and transactions conducted over the Internet.

7) What is digital certificate?

Ans 1:A digital certificate is an electronic "passport" that allows a person, computer or organization to exchange information securely over the Internet using the public key infrastructure (PKI). A digital certificate may also be referred to as a public key certificate.

Just like a passport, a digital certificate provides identifying information is forgery resistant and can be verified because it was issued by an official, trusted agency. The certificate contains the name of the certificate holder, a serial number, expiration dates, a copy of the certificate holder's public key (used for encrypting messages and digital signatures) and the digital signature of the certificate-issuing authority (CA) so that a recipient can verify that the certificate is real.

To provide evidence that a certificate is genuine and valid, it is digitally signed by a root certificate belonging to a trusted certificate authority. Operating systems and browsers maintain lists of trusted CA root certificates so they can easily verify certificates that the CAs have issued and signed. When PKI is deployed internally, digital certificates can be self-signed.

OR

Ans 2: An attachment to an electronic message used for security purposes. The most common use of a digital certificate is to verify that a user sending a message is who he or she claims to be, and to provide the receiver with the means to encode a reply.

An individual wishing to send an encrypted message applies for a digital certificate from a *Certificate Authority (CA)*. The CA issues an encrypted digital certificate containing the applicant's public key and a variety of other identification information. The CA makes its own public key readily available through print publicity or perhaps on the Internet.

The recipient of an encrypted message uses the CA's public key to decode the digital certificate attached to the message, verifies it as issued by the CA and then obtains the sender's public key and identification information held within the certificate. With this information, the recipient can send an encrypted reply.