

**(9A05803) WEB SERVICES  
(ELECTIVE - III)**

**UNIT III**

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

**Web services architecture and its characteristics:**

1Q. what are the characteristics of the web services architecture?

**Characteristics of web services:**

The Web services architecture represents the logical evolution of traditional computer-based applications to services-oriented applications over the Internet. It defines a distributed computing mechanism by adopting a service-oriented architecture (SOA), where all of the applications are encapsulated as services and made available for invocation over a network. These services can be leveraged from different applications and platforms with varying technologies adopting common industry standards and platform-independent and language-neutral Internet protocols for enabling application interoperability, thus making them easily accessible over the Internet.

In addition, it provides a mechanism for categorizing and registering the services in a common location by making them available for discovery and collaboration over the Internet or corporate networks.

Using Web services architecture and adhering to its standards also exposes existing and legacy applications as Web services, and the clients invoking these services do not require that they are aware of their target system environment and its underlying implementation model.

**Core Building blocks of Web services:**

2Q. What are the key design requirement of the web services architecture

The basic principles behind the Web services architecture are based on SOA and the Internet protocols. It represents a composable application solution based on standards and standards-based technologies.

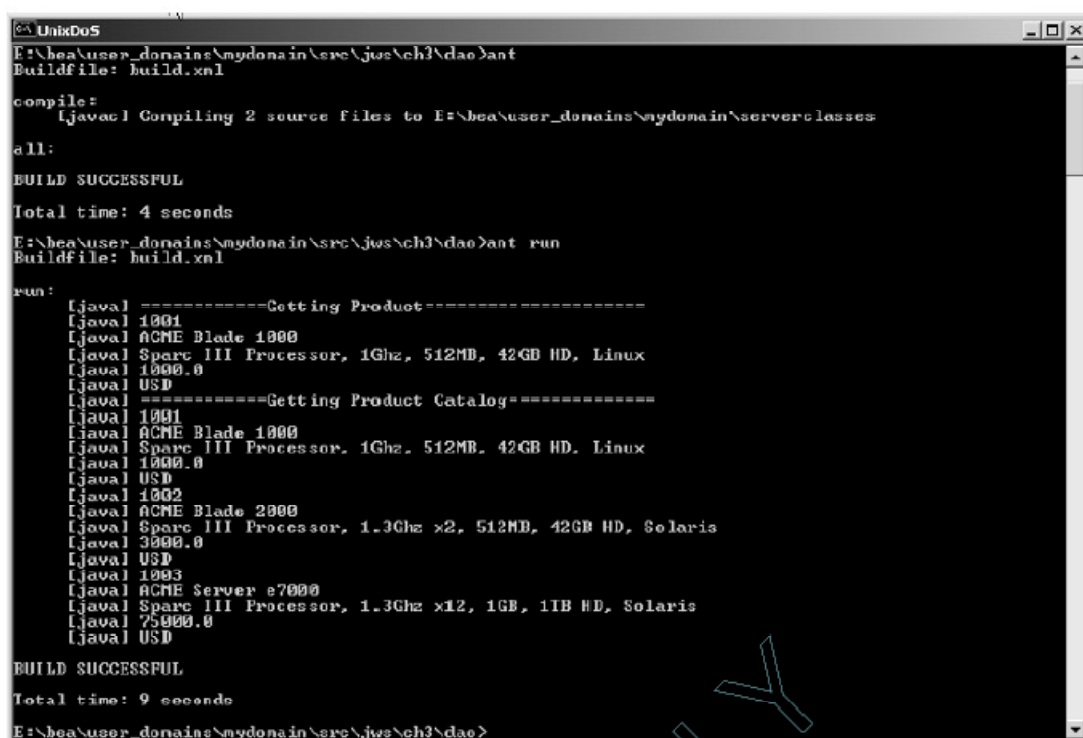
Some of the **key design requirements** of the Web services architecture are the following:

- ✓ 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

3Q. Explain the core building blocks of web services architecture? (or) Draw and explain the web services architecture

A typical Web service architectural model consists of three key logical components as core building blocks mapping the operational roles and relationships of a Web services environment. Figure 3.1 represents the core building blocks of a typical Web services architecture.



```

C:\UnixDos
E:\bea\user_domains\mydomain\src\jws\ch3\dao>ant
Buildfile: build.xml

compile:
[java] Compiling 2 source files to E:\bea\user_domains\mydomain\serverclasses
all:
BUILD SUCCESSFUL
Total time: 4 seconds
E:\bea\user_domains\mydomain\src\jws\ch3\dao>ant run
Buildfile: build.xml

run:
[java] -----Getting Product-----
[java] 1001
[java] ACME Blade 1000
[java] Spare III Processor, 1Ghz, 512MB, 42GB HD, Linux
[java] 1000.0
[java] USD
[java] -----Getting Product Catalog-----
[java] 1001
[java] ACME Blade 1000
[java] Spare III Processor, 1Ghz, 512MB, 42GB HD, Linux
[java] 1000.0
[java] USD
[java] 1002
[java] ACME Blade 2000
[java] Spare III Processor, 1.3Ghz x2, 512MB, 42GB HD, Solaris
[java] 2000.0
[java] USD
[java] 1003
[java] ACME Server e7000
[java] Spare III Processor, 1.3Ghz x12, 1GB, 1TB HD, Solaris
[java] 75000.0
[java] USD
BUILD SUCCESSFUL
Total time: 9 seconds
E:\bea\user_domains\mydomain\src\jws\ch3\dao>

```

**Figure 3.1** Core building blocks of Web services architecture.

**Services container/runtime environment.** The services container acts as the Web services runtime environment and hosts the service provider. Typical to a Web application environment, it defines the Web services runtime environment meant for client communication as a container of Web services interfaces by exposing the potential components of the underlying applications.

It facilitates the service deployment and services administration. In addition, it also handles the registration of the service description with the service registries.

Usually, the Web services platform provider implements the services container. The Web application servers provide system services and APIs that can be leveraged as the Web services container.

**Services registry.** The services registry hosts the published services and acts as a broker providing a facility to publish and store the description of Web services registered by the service providers. In addition, it defines a common access mechanism for the service requestors for locating the registered services.

**Services delivery.** It acts as the Web services client runtime environment by looking up the services registries to find the required services and invoking them from the service provider. It is represented as a presentation module for service requestors, by exposing the appropriate interfaces or markups for generating content and delivery to a variety of client applications, devices, platforms, and so forth.

To build the Web services architecture with these logical components, we need to use standardized components and a communication model for describing and invoking the services that are universally understood between the service providers and their potential service requestors. It also requires a standard way to publish the services by the service provider and store them in the service broker. In turn, service requestors can find them.

**WSDL.** This resides in the services container and provides a standardized way to describe the Web services as a service description. In ebXML-based architecture, ebXML CPP/A provides services descriptions including business partner profiles and agreements.

**UDDI.** This provides a standard mechanism for publishing and discovering registered Web services, and it also acts as the registry and repository to store WSDL-based service descriptions. In ebXMLbased architecture, ebXML Registry & Repository provides a facility to store CPP/CPA descriptions for business collaboration.

### **Standards and Technologies available for implementing web services. or Tools of the Trade**

4Q. Write a short notes on the following:

- I. SOAP
- II. WSDL
- III. UDDI
- IV. ebXML

or

Explain about the standards and technologies for implementing the web services

### **Simple Object Access Protocol (SOAP)**

The Simple Object Access Protocol, or SOAP, plays the role of the messaging protocol for exchanging information between the service provider and the service requestor. It consists of the following:

**SOAP Envelope.** It describes the message, identifying the contents and the envelope's processing information.

**SOAP Transport.** It defines the bindings for the underlying transport protocols such as HTTP and SMTP.

**SOAP Encoding.** It defines a set of encoding rules for mapping the instances of the application-specific data types to XML elements.

**SOAP RPC conventions.** It defines the representation of the RPC requests and responses. These SOAP requests and responses are marshaled in a data type and passed in to a SOAP body.

Example 3.1 represents a SOAP message using an HTTP post request for sending a `getBookPrice()` method with `<bookname>` as an argument to obtain a price of a book.

```
POST /StockQuote HTTP/1.1
Host: www.acmeretailer.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 1000
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: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>
```

**Listing 3.1** SOAP message using HTTP.

### Web Services Description Language (WSDL)

The Web Services Description Language, or WDDL, is an XML schemabased 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.

### Universal Description, Discovery, and Integration (UDDI)

Universal Description, Discovery, and Integration, or UDDI, defines a mechanism to register and categorize Web services in a general-purpose registry that users communicate to in order to discover and locate registered services. While querying a UDDI registry for a service, the WSDL description describing the service interface will be returned. Using the WSDLdescription, the developer can construct a SOAP client interface that can communicate with the service provider

UDDI can be implemented as a public registry to support the requirements of a global community or as a private registry to support an enterprise or a private community.

## ebXML

ebXML provides a standard framework for building an electronic marketplace by enabling the standardization of business processes, business partner profiles, and partner agreements. In general, ebXML complements other Web services standards like SOAP, WSDL, and UDDI.

The following are major features of ebXML:

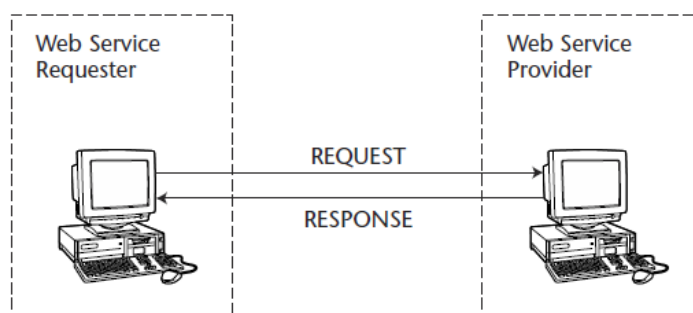
- ■ ebXML Messaging Service (MS) is a value-add over SOAP that provides reliability and security mechanisms.
- ■ ebXML BPSS enables business processes to be described.
- ■ ebXML CPP/CPA is a value-add over WSDL that enables business partner profiles and partner agreements to be described.
- ■ ebXML reg/rep provides a registry and repository, while UDDI is just a registry.
- ■ ebXML Core components provide a catalogue of business process components for the business community.

## Web Services Communication Models

### 5Q. Explain in detail webs services communication models

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

**RPC-Based Communication Model:** The RPC-based communication model defines a request/response-based synchronous communication. When the client sends a request, the client waits until a response is sent back from the server before continuing any operation. Typical to implementing CORBA or RMI communication, the RPC-based Web services are tightly coupled and are implemented with remote objects to the client application. Figure 3.3 represents an RPC-based communication model in Web services architecture.

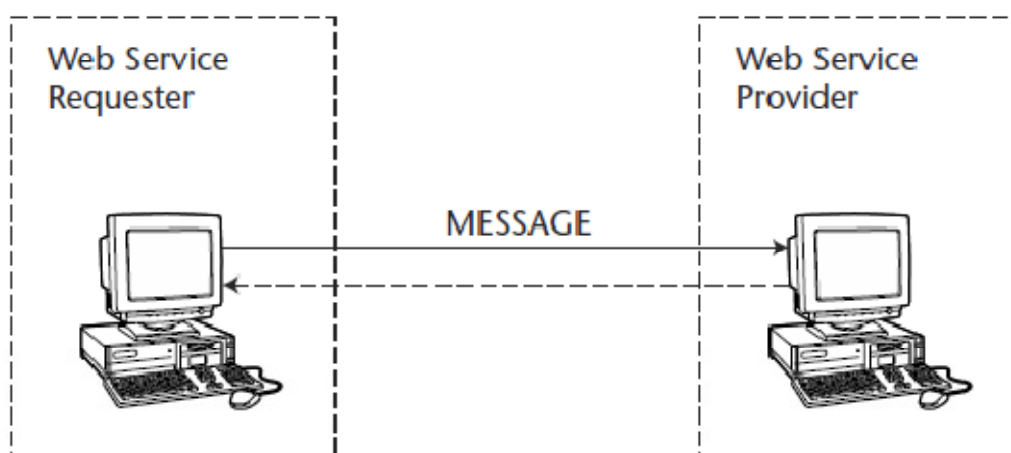


**Figure 3.3** RPC-based communication model in Web services.

The clients have the capability to provide parameters in method calls to the Web service provider. Then, clients invoke the Web services by sending parameter values to the Web service provider that executes the required methods, and then sends back the return values. Additionally, using RPCbased communication, both the service provider and requestor can register and discover services, respectively.

### Messaging-Based Communication Model

The messaging-based communication model defines a loosely coupled and document-driven communication. The service requestor invoking a messaging-based service provider does not wait for a response. Figure 3.4 represents a messaging-based communication model in Web services architecture.



**Figure 3.4** Messaging-based communication model.

In Figure 3.4, the client service requestor invokes a messaging-based Web service; it typically sends an entire document rather than sending a set of parameters. The service provider receives the document, processes it, and then may or may not return a message. Depending upon the implementation, the client can either send or receive a document asynchronously to and from a messaging-based Web service, but it cannot do both functionalities at an instant.

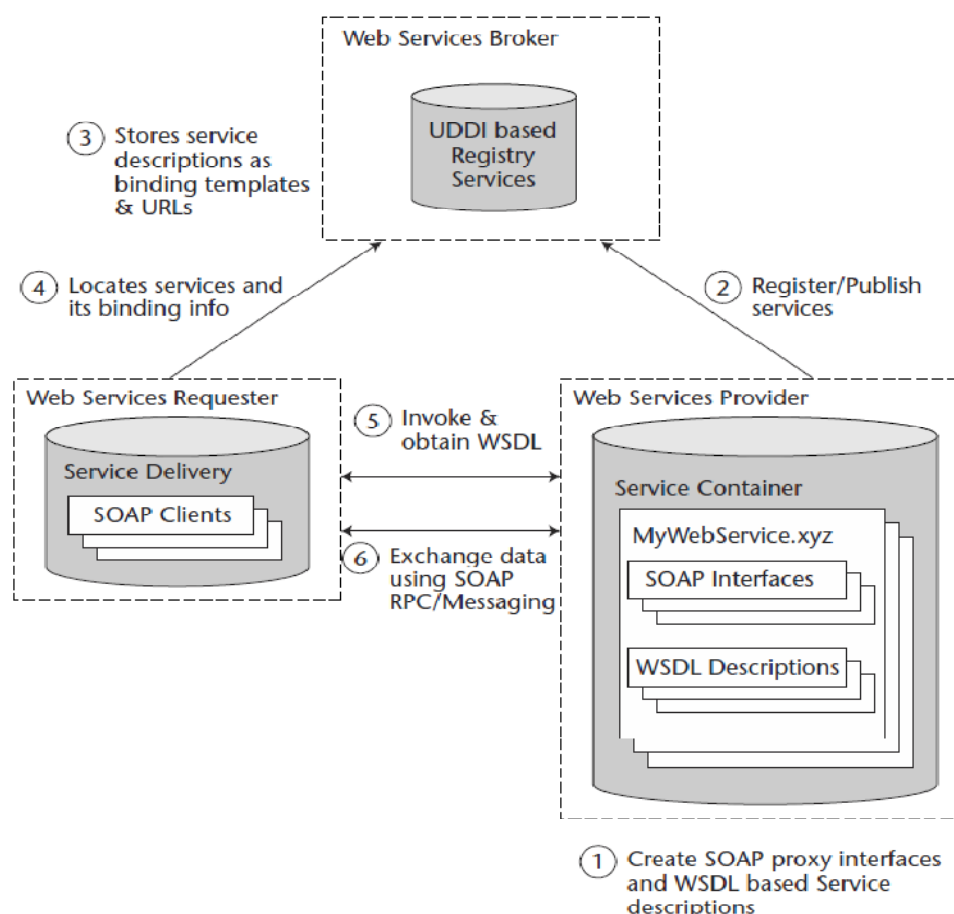
In addition, it also is possible to implement messaging with a synchronous communication model where the client makes the service request to the service provider, and then waits and receives the document from the service provider.

Adopting a communication model also depends upon the Web service provider infrastructure and its compliant protocol for RPC and Messaging. The current version of SOAP 1.2 and ebXML Messaging support these communication models; it is quite important to ensure that the protocols are compliant and supported by the Web services providers. It also is important to satisfy other quality of services (QoS) and environmental requirements like security, reliability, and performance.

### Basic steps for implementing the web services:

#### 6Q. With a neat diagram explain the process of implementing the 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.



**Figure 3.5** Process steps involved in implementing Web services.

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.

In the case of an ebXML-based environment, the steps just shown are the same, except ebXML registry and repository, ebXML Messaging, and ebXML CPP/CPA are used instead of UDDI, SOAP, and WSDL, respectively. The basic steps just shown also do not include the implementation of security and quality of service (QoS) tasks.

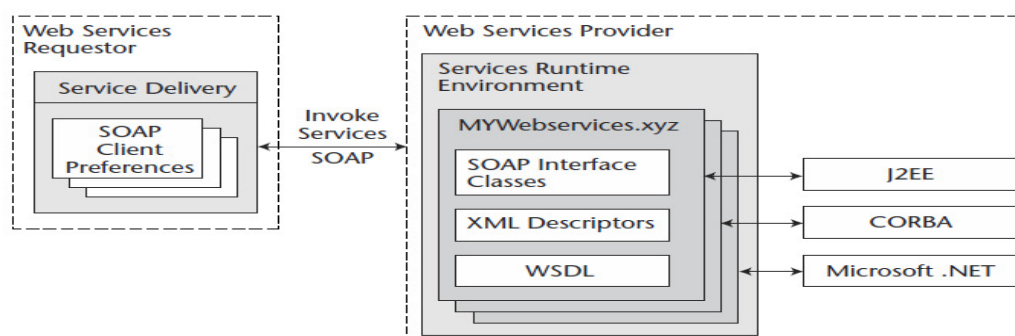
#### Developing Web services enabled applications:

#### 7Q. Explain how to develop web services enabled applications

The design and development process of creating a Web services-enabled application is not different from the typical process of developing a distributed application. In case of Web services, it can be created as a new application or from using an existing application by repurposing them as services.

In a Web services implementation, it also is possible to expose existing/ legacy applications as services by encapsulating the core business functionalities of those underlying applications. The underlying applications can be of any application implemented in any programming language and running on any platform.

Figure 3.6 represents a typical Web services implementation model providing service-oriented interfaces supporting a variety of back-end application environments.



**Figure 3.6** Exposing applications through Web services.



The implementation steps generally involved in developing Web services solutions by exposing back-end business applications are as follows:

1. The potential business component of the underlying application will be encapsulated as service-oriented interfaces using SOAP and then exposed as Web services by deploying them in a Web services service container or a SOAP runtime environment. Using those SOAP-based interfaces, the service container handles all the incoming SOAP requests/responses or messaging-based operations and maps them as methods and arguments of the underlying business application.
2. WSDL-based service descriptions will be generated and then reside in a service container. WSDL defines the communication contract required for invoking the SOAP-based service interfaces. These WSDL-based service descriptions will be published in a UDDI registry as service templates and its location URLs. The interfaces required for publishing in the UDDI registry are usually provided by the Web service container provider.
3. The service requester finds the services using the discovery mechanisms (registry API) and obtains the service description and its provider location URL. It then connects to the service provider to obtain WSDL.
4. To invoke the services exposed by the service provider, the service requestor (service delivery environment) is required to implement SOAP-based client interfaces according to the service description defined in the WSDL.

The Web services container/runtime environment provider generally provides the tools required for creating SOAP-based services interfaces from existing applications and generating WSDL-based service descriptions. Depending upon the Web services runtime environment.

The previous steps are usually common at all levels of Web services development, irrespective of the target application environment such as J2EE, CORBA, Microsoft .NET, or standalone applications based on Java, C++, Microsoft Visual Basic, and legacy applications based on, the Mainframe environment. As a result, implementing Web services unifies J2EE, CORBA, .NET, and other XML-based applications with interoperability and data sharing.

### **How to Develop Java-Based Web Services**

#### **8Q. Explain how to develop web services from J2EE applications**

With the overwhelming success of Java in Web and pervasive applications running on a variety of platforms and devices, the Java platform has become the obvious choice for enterprise architects and developers. In addition to the Java platform, today the J2EE-based application environment also has become the preferred solution for running Web servicesbased solutions.

#### ***Building Web Services in the J2EE Environment***

The process of building Web services using a J2EE environment involves exposing J2EE components such as servlets and EJBs. In addition, J2EE applications also can access these exposed services using standard protocols.

In a typical implementation, a J2EE-based Web services model defines another way of exposing their business components similar to Web applications and RMI/IIOP-based application connectivity and without changing the architectural model or code of the existing J2EE components. For example, in a J2EE-based application server environment, J2EE components can be exposed for remote access through RMI/IIOP. In the case of a Web service provider using a J2EE environment, in addition to RMI/IIOP, it also is possible to expose those components as a service via WSDL and handle the exposed service by sending and receiving SOAP-based requests/responses or messages.

The following steps are commonly involved in creating Web services from a J2EE-based application component:

1. Select a Web services platform provider, which provides a consistent platform for building and deploying Web services over the J2EE applications.
2. Define a Web service-enabled application and its behavior.
  - a. Select the potential J2EE components (for example, EJBs, Servlets, and JMS applications) that are required to be exposed as services or are using the existing services.
  - b. Choose the communication model (RPC-based synchronous or messaging-based asynchronous) depending upon the required behavior of the underlying components (for example, Session or Entity EJBs using RPC-based communication or JMS applications using messaging-based communication).
  - c. Ensure that the service uses only built-in/custom data types mapping for XML and Java supported by the Web services container. This applies only to RPC-based communication models.
3. Develop the Web service by writing the interfaces required for accessing the exposed components (for example, EJBs, Servlets, and JMS applications).
  - a. Develop the potential J2EE component (for example, EJBs, Servlets, and JMS applications) that are required and deploy them in a J2EE-compliant container. Ensure that the data types used by the components are supported in the XML/Java mappings defined by the provider.
  - b. Implement the SOAP message handlers.
4. Assemble the required components into a required structure (defined by the Web services platform provider), additionally creating the deployment descriptors for the services (as defined by the Web services platform provider) and package them as a deployable EAR.
  - a. Most Web service platform vendors provide utility tools to generate Web services components (SOAP interfaces) by introspecting the components (especially its methods and values) and mapping them to its supported data types.

b. Also it is important to note, the upcoming release of the J2EE 1.4 specification is expected to provide a complete J2EE-based Web services platform and would enable the deployment of J2EE components as Web services.

5. Deploy the Web service components in the Web services container and make them available to its remote clients (based on the required protocol bindings such as HTTP and SMTP).

6. Create test clients for invoking the deployed Web services.

7. Register and publish your Web service in a UDDI registry, in case you require enabling the service available by searching public/private UDDI registries for Web services.

### *J2EE and Java Web Services Developer Pack (JWSDP)*

#### **9Q. Write a short notes on JWSDP**

Sun Microsystems as part of its Java community process has already released its Java API for Web Services for the developer community as the Java Web Services Developer Pack (JWSDP). It provides a full-fledged solution package for developing and testing Web services using the Java APIs. In addition, leading Web services platform providers like Systinet, CapeClear, and Mind Electric and leading J2EE vendors like BEA, IBM, and Sun iPlanet also released their Web services capabilities, adopting a Java platform and supporting Java APIs for Web services as per JWSDP.

JWSDP 1.0 provides a one-stop Java API solution for building Web services using a Java platform. The key API components include the following:

- ■ Java API for XML Messaging (JAXM)
- ■ Java API for XML Processing (JAXP)
- ■ Java API for XML Registries (JAXR)
- ■ Java API for XML Binding (JAXB)
- ■ Java API for XML-Based RPC (JAX-RPC)
- ■ Java WSDP Registry Server (JWSDP)
- ■ Java Server Pages Standard Tag Library (JSTL)

Leading J2EE application server vendors have announced their support to this effort and also started releasing their JWSDP API implementation. This helps the developers to build Web services by exposing their existing J2EE applications.

## Exposing J2EE Components as Web Services

### 10Q. Explain in detail the development of web services by exposing J2EE components deployed in a J2EE application server

The J2EE environment delivers platform-independent Java component-based applications providing a multi-tiered distributed application model with several advantages like security, scalability, administration tools, portability between vendor implementations, and reliability of deployed applications. In general, it defines the following components residing in different logical tiers:

- ■ JavaServer Pages (JSP) and Java Servlet-based components act as Web components running on the Web/Servlet container of the J2EE server.
- ■ Enterprise JavaBeans (EJB)-based components act as business or persistence components running on the EJB container of the J2EE server.
- ■ JDBC (Java Database connectivity) and J2EE connector architecturebased components act as the integration tier of the J2EE server for integrating database applications and enterprise information systems.

The key differences between J2EE components and traditional Java applications is that J2EE components are assembled and deployed into a J2EE application server in compliance with the J2EE specification. These components are managed by J2EE server system services such as synchronization, multithreading, and connecting pooling. Additionally, the J2EE server implementation also provides capabilities like clustering, transaction coordination, messaging, and database connection pooling.

In short, developing Web services from J2EE-based applications requires the implementation of components using J2EE component APIs (such as EJBs and servlets), then packaging and deploying them in a J2EE container environment as target enterprise applications. The components are then hosted in a J2EE-compliant application server. Exposing these J2EE components as Web services also requires a Web services container environment, which enables the creation and deployment of SOAP-based proxy interfaces.

A typical scenario, exposing a J2EE-based application component as Web services involves the steps in the following list:

#### STEPS FOR THE SERVICE PROVIDER

1. The potential J2EE component deployed in an application server environment will be encapsulated as a service-oriented interface using SOAP and then deployed in a Web services runtime environment.
2. WSDL-based service descriptions are generated and then reside in the services runtime environment. The service requestor clients create SOAP-based client interfaces using the WSDL-based descriptions.
3. Using registry APIs, WSDLs are used for publishing the services in a public/private UDDI registry.

## **STEPS FOR THE SERVICE REQUESTOR**

1. The service requestor clients create SOAP-based client interfaces using the WSDL-based descriptions exposed by the service provider.
2. The service requestor may choose to use any language for implementing the client interfaces, but it must support the use of SOAP for communication.
3. These client interfaces then are used to invoke the service provider deployed services