www.tutorial4us.com

Tutorial4Us

College Projects

```
void main()
{
 int no;
 clrscr();
 printf("Hello");
}
```

Basic Program

C Tutorial

JAVA Programming

tutorial4us.com

A Perfect Place for All Tutorials Resources

Core Java | Servlet | JSP | JDBC | Struts | Hibernate | Spring

Java Projects | C | C++ | DS | Interview Questions | JavaScript

College Projects | eBooks | Interview Tips | Forums | Java Discussions

For More Tutorials Stuff Visit

www.tutorial4us.com

A Perfect Place for All Tutorials Resources

Web Services

(Natraj Notes)

www.tutorial4us.com

25th Aug '11

WEB SERVICES

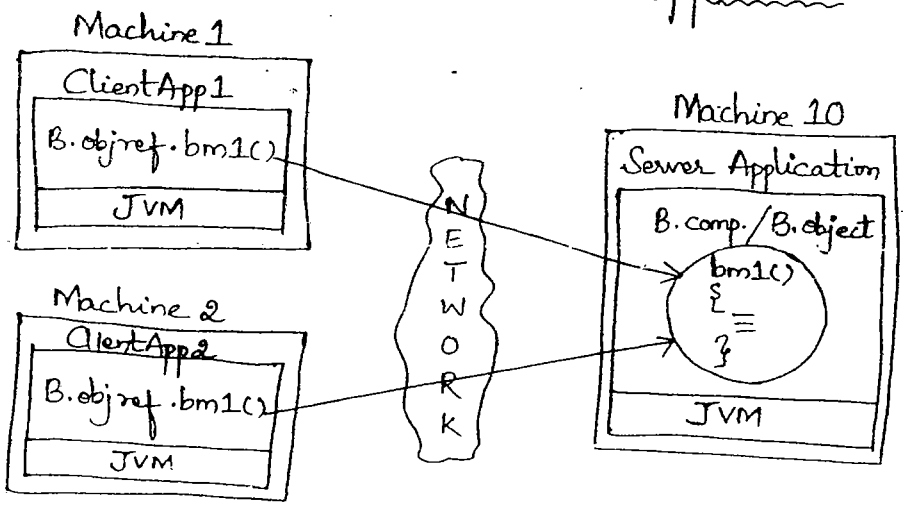
INTRODUCTION:

The Socket programming based applications can be called as Traditional Client-Server applications. In these applications, client application directly communic-tes with the server application from remote or local places having location dependency. That means any change in the location of the server in the application must be informed to all the client applications directly. This creates problems when huge number of clients are there for server application.

Traditional Client Server applications allows both remote and local clients but these applications are location dependent application.

Socket programming applications, Basic Jdbc application, Web applications that are not hosted in the internet are examples for Traditional Client Server applications.

Traditional Client Server Application

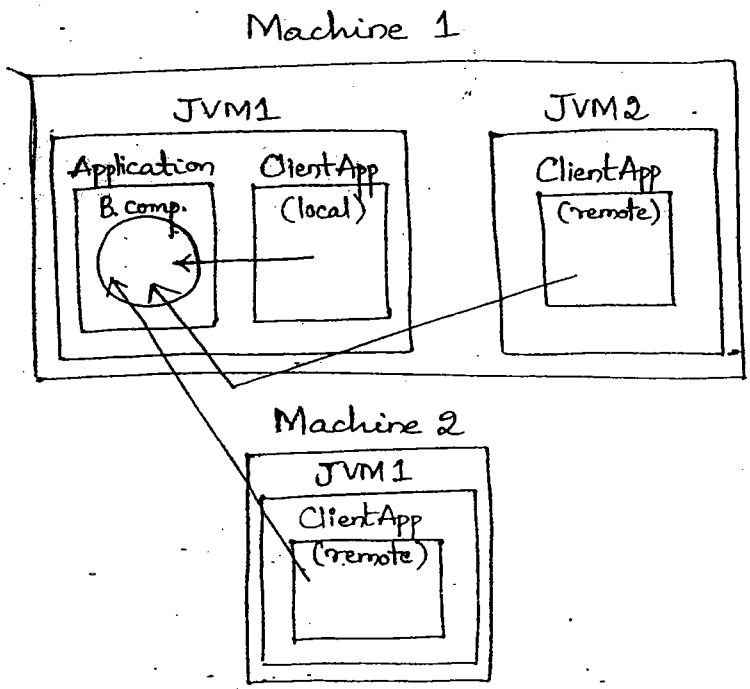


In one computer we can see multiple JVMs activation simultaneously or parallely.

To run each Java application, we need one JVM. So to execute multiple Java applications in a computer, we need multiple JVMs.

If application and its Client resides and executes from the same JVM, then that client is called as Local client to the application.

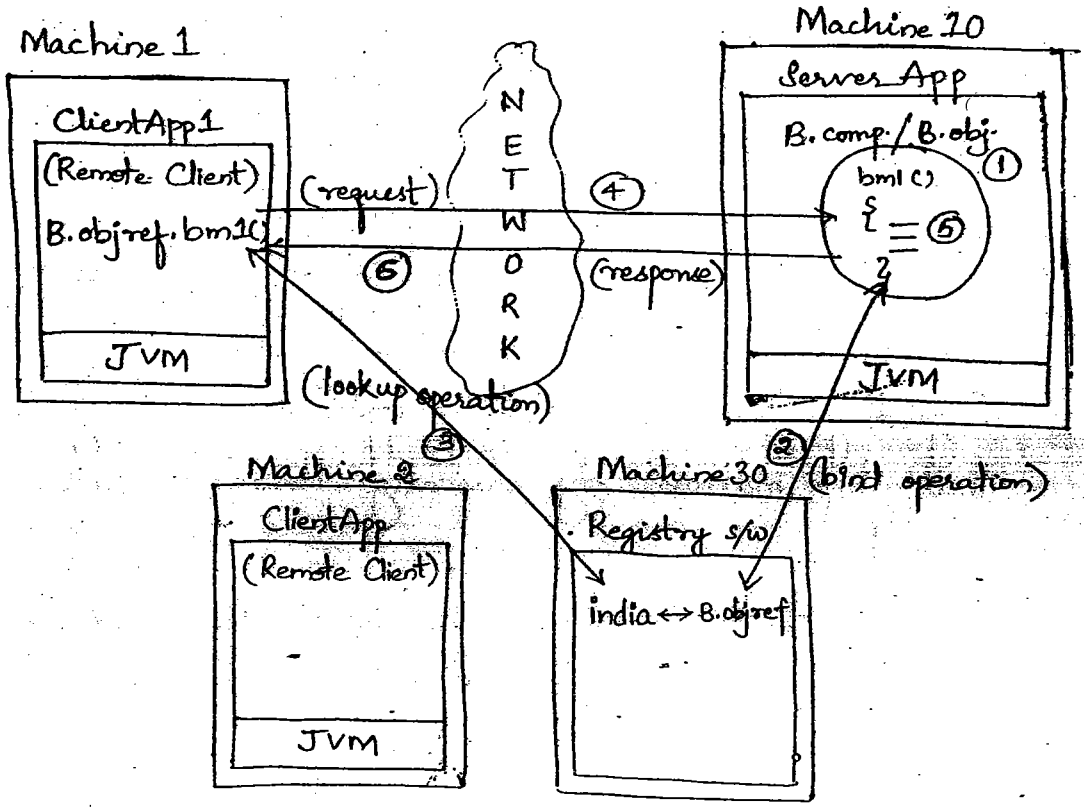
If application and its Client resides and executes from two different JVMs of same computer or two different computers, then that client is called as Remote Client to the application.



Distributed Application:

The Client-Server application that gives location transparency by using Registry software support is called as a Distributed application.

Distributed Application



In order to provide global visibility to Java object or object reference, place them in Registry s/w having Nick name / Alias name.

The DataSource object that represents Jdbc Connection pool, the business objects of Distributed application will be placed in Registry s/w for global visibility.

The process of keeping object or object reference in Registry s/w is called as Binding operation.

The process of searching and gathering the object and object reference from Registry s/w based on Nick name / Alias name is technically called as Lookup operation.

Distributed applications are location transparent because of Registry s/w. Here client application communicates with the server application only after getting business object reference from the Registry s/w. So any change in Server application location need not to be informed to all the Client applications. If it is just informed to Registry s/w, the client application will gather that information dynamically through lookup operations.

In distributed applications, the Registry s/w maintains business object references of multiple server applications. So the location of Registry s/w is fixed and never changes.

Register s/w s:

RMI registry
COS (Common Object Service) registry
DNS (Domain Naming Service) registry
Weblogic registry
Open LDAP (Light weight Directory Access Protocol) registry
Glassfish registry
and etc

The websites placed in the Internet having domain name (like www.yahoo.com) and having registration with DNS registry are called as

Distributed Applications.

The Classroom level web applications where domain name and DNS registry are not involv are called as Traditional Client Server applicatio

A distributed application can be developed eith as web application or non web application.

Wr.t the diagram,

① Server application develops business component/ business object having business methods.

② Server application binds the business object reference with Registry s/w having nick name or alias name for global visibility.

③ Client application gets business object refere nce from Registry through Lookup operation

④ Client application calls business method on business object reference

⑤ The business logic of business method in business component executes at server application.

⑥ The business method generated results goes to Client application.

To develop Distribute Applications, we need Distributed Technologies. They are

* RPC (Remote Procedure Call) — from XOpen Community
* RMI (Remote Method Invocation) (J) — from SUN MS
* CORBA (Java can be there) — from Object Management Group (800+ comp.)

DCOM (Distributed Component Object Model) — from MS
Remoting — from MS (MicroSoft)
EJB (Java based) — from Sun Microsystems
Webservices (Java can be there) — not from any vendor
HttpInvoker (Java based) — from Interface21 (part of Spring s/w)
⋮

Webservices, HttpInvoker gives web based distributed applications (Http Protocol) where as the remaining distributed technologies allow to develop non-web based distributed applications.

Aug 11

Achieving the same performance irrespective of increase or decrease in number of clients/requests is nothing but developing scalable application.

Distributed applications can be developed as Scalable applications.

Specification is a document that contains set of rules and guidelines to develop s/ws. There are 2 types of Specifications. They are

1. Open Specification (any company can implement s/ws based on this specification)
2. Proprietory Specification
   — only specific companies can implement s/ws based on this specification.

Open Specification examples:

Jdbc, Servlets, Jsp, EJB specifications.

Jdbc specification contains set of rules and guidelines to develop Jdbc drivers

Servlets specifications contains set of rules and guidelines to develop Servlet Container s/w.

Proprietory Specification Examples:

.NET

RPC

→ Application should be developed in C, C++.

→ OS dependent (both client and server applicatio must reside and execute in same Operating System). (On client and server machines same Operating system must be installed).

→ Language dependent (both client and server applications must be developed in same langu(like C/C++).

→ Architecture specific and dependent (both client and serve machines must be the, with same architecture

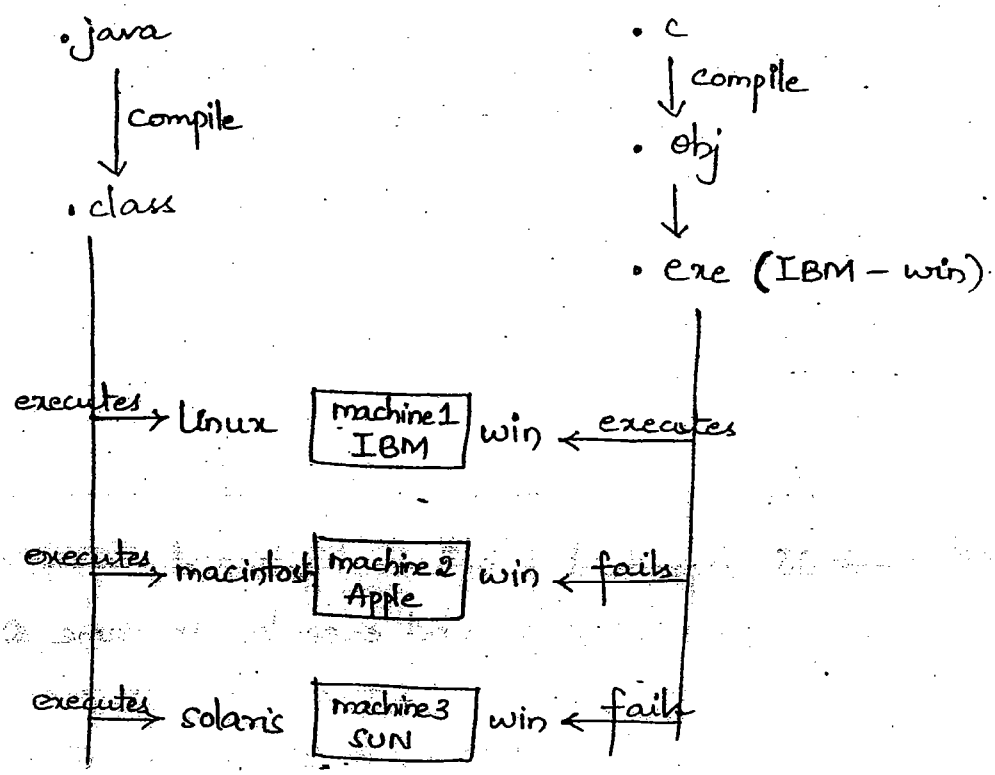
→ No built in middleware services.
like security, Transaction Mgmt. and etc..

There are three important architectures computer manufacturing architectures

1. IBM architecture
2. Apple architecture
3. Sun architecture

C, C++ languages are not only operating system dependent, they are also architecture dependent.

Java is both Operating system and architectur Independent.



RMI

→ given by SUN Ms (part of Jdk s/w)
→ both client and server applications must be developed in Java language.
→ OS independent
→ Language dependent
→ Architecture independent
→ gives limited number of middleware services.
→ not suitable for Large scale distributed appl.s.

DCOM

→ given by MicroSoft.
→ application should be developed using MicroSoft technologies.
→ language independent (partially)
(only with Microsoft supplied languages)

→ OS dependent
→ Architecture dependent
→ gives built-in middleware services
→ not suitable for large scale applications.

Remoting of .net also contains the same feature

CORBA (Common Object Request Broker Architecture

→ given by OMG (except Microsoft, 800+ companies are there in this group).
→ Application can be developed by using multiple technologies.
→ language independent
→ OS Independent
→ architecture independent
→ gives built-in middleware services

CORBA is a specification. The following are CORBA specification softwares.

Boss → from IBM
IDLJ → from Sun
VisiBroker → from Visigenic

The CORBA specification based s/w s ~~have~~ are really failed to implement specification features in the development of medium scale and large scale projects.

EJB (Enterprise Java Bean)

→ given by Sun MS
→ Application should be developed in ~~Java~~ environment
→ Language dependent

→ OS independent
→ Architecture independent
→ gives more number of middleware services
→ Enhancement of RMI having all the facilities of CORBA except language independency.

To develop pure Java based distributed applications, work with EJB.

Webservices

→ given based on SOA (Service Oriented Architecture) specification.
→ webservices is not given by any specific organization.

Working with Webservices is all about working with multiple technologies, concepts, specifications, protocols together to develop distributed applications.

→ language independent (at max (not 100%))
→ OS independent
→ Architecture independent
→ allows to work with server supplied middleware services.
→ Web based distributed technology.
→ Allows to convert existing business components into web services based business components.

We can use Webservices not only to develop scratch level distributed applications, we can also use them to convert the existing business components

into interoperatable business components.

While working with hardware components, we never bother about their compatibility with other devices and components because they are developed as interoperatable components.

The ability to work with components/resources without worrying about their environment is nothing but Interoperatability (any component can be used in any application).

To develop s/w business components as interoperatable components, we take the support of web services.

To get communication between two incompatible s/w applications, we take the support of Webservices.

In Webservices based distributed application the server application can be developed in any language and the client application can be developed in any language.

e.g., (i). for .net based business component/server application, we can have Java client..

(ii). for Java based business component/server application, we can have .net client.

Based on the distributed technology we use to develop distributed application, the registry s/w will be decided.

29th Aug'11

In RMI applications, RMI Registry is the registry s/w.

In CORBA applications, COS is the Registry s/w.

In EJB applications/components, the Registry s/w will be decided based on the Application Server s/w we use.

In Weblogic ⟶ Weblogic Registry
In Glassfish ⟶ Glassfish Registry
In JBoss ⟶ Jnp Registry

In Webservices based distributed applications, UDDI Registry is the Registry s/w.

Distributed applications are location transparent because of Registry s/w's.

In distributed applications, the business component/business object must be developed as Remote object.

The object whose methods can be invoked from both remote and local clients is called as Remote object.

Java object becomes remote object only when its class implements the marker interface called java.rmi.Remote.
(can also be called as Tag interface)

The interface whose implementation gives special behaviour or capabilities to the objects of class is called as a Marker Interface/Tag interface.

Most of the marker interfaces are Empty interfaces. But marker interfaces can be there with or without methods.

Some examples for Marker Interfaces:

java.rmi.Remote
javax.servlet.SingleThreadModel
java.lang.Cloneable
java.lang.Runnable
java.io.Serializable
java.io.Externalizable

In distributed application development, we can see the following four resources utilization.

1. Service provider
2. Service client
3. Service interface
4. Service registry

Service provider is the server application of distributed application who develops business object having business methods as remote objects.

Service client is a client application of distributed application having the capability to call the business methods of business object from remote or local clients.

Service interface is a common understanding document between service provider and service client having the declaration of business methods. This document will be used by both service provider and

service client.

Service registry maintains business objects/ business object references along with related documents for global visibility (to expose them to clients). Service registry is nothing but Registry s/w of distributed application.

In Webservices environment, service provider application and service client applications can be developed in same or different technologies by using Java/JEE, .Net and etc..

Service interface in Webservices environment is WSDL (WebServices Definition Language) document (it is an XML document). In Webservices environment, UDDI Registry acts as Service registry.

To pass data between two incompatible applications, we take the support of XML. To get communication between two incompatible s/w applications, we take the support of WebServices.

SOA is a s/w design principle having idea, plan to develop interoperatable distributed appl.n. SOA specifies the requirements and benefits of developing interoperatable distributed applications.

WebServices is a technology specification having set of rules and guidelines to develop interoperatable distributed applications.

JAVA/JEE, .Net and etc., technologies can be used to develop interoperatable distributed apps

based on the specification rules and guidelines given in Web Services.

WebServices is not a direct s/w technology to install and use. Working with Web service is nothing but working with multiple technologies, concepts and protocols together to develo interoperatable distributed application.

XML
SOAP (Simple Object Access Protocol)
HTTP
WSDL
UDDI
- Java/JEE APIs
.net APIs
and etc-

For benefits of WebServices refer page no.s 4, 9, 10 in the given handout booklet.

The project that is developed on the old versions of the existing technologies and or outdated technologies is called as Legacy Project e.g., Java 1.1 project, PASCAL project, C project, and etc..

→ Why Servlet, Jsps are not involved in distributed application development eventhough they are web based components?

1. They do not allow programmable clients (allows only browser window as client).
2. They do not allow to place business methods (Servlet container, Jsp container executes them through life cycle methods).
3. Middleware services support is minimum.
4. We cannot register objects of Servlet, Jsp programs with Registry s/w to achieve location transparency (because these objects are created by Container of Servlet/Jsp).

5th Aug'11

In WebServices:

Service Provider
- It is the server application that contains interoperability business component.

Service Client
- It is the client application that calls business method of business component.

Service Interface
- It is an WSDL document.

Service Registry
- It is an UDDI Registry.

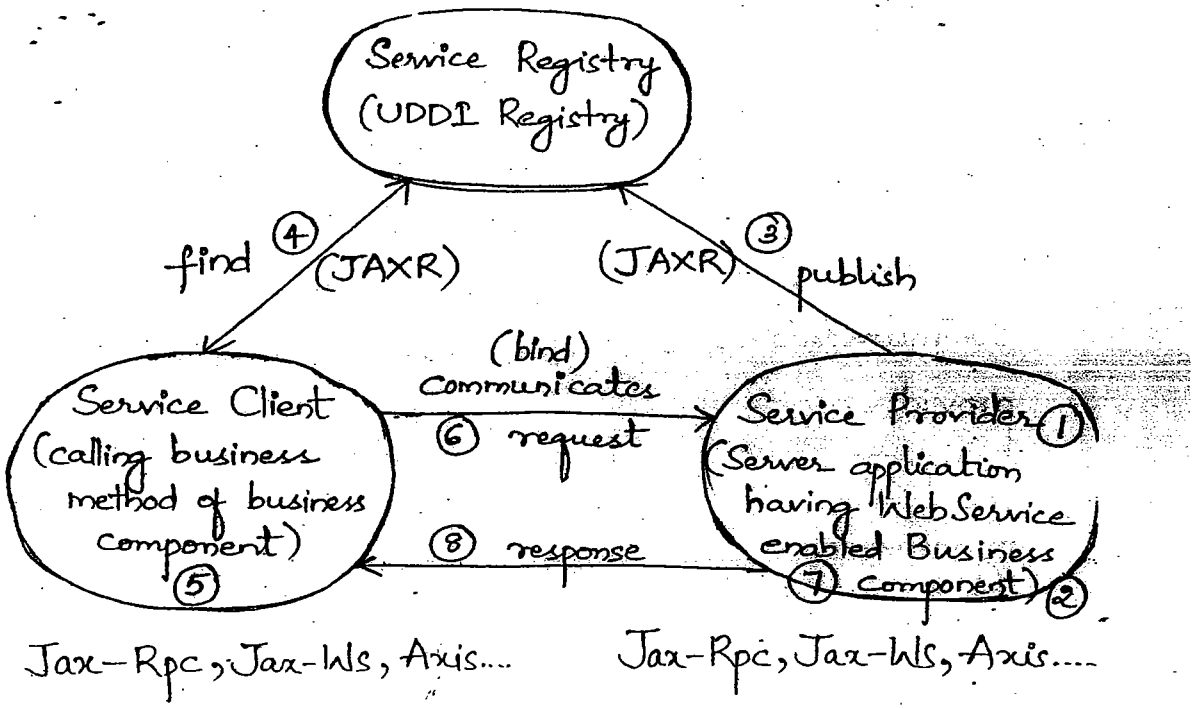
A Complete Web Services Architecture:

With respect to the diagram,

(1) Programmer develops service provider application having Web Service enabled business component.

(2) Programmer generates/develops WSDL

document for that WS enabled business component.



(3). Programmer sends WSDL document to UDDI registry for global visibility.

Note: This WSDL document contains XML based entries having details about the Web Service enabled business component.

(4) Service client gets WSDL document from the Service registry.

(5) Service client understands WSDL document and develops the client application by using certain API.

(6) Service client sends request to service provider application by invoking certain business method.

(7) Business logic in business method executes.

(8) Business method generated result comes to the service client as response.

Service provider application and service client application interacts with service registry by using JAXR API (in Java enabled environment).

Service client and service provider applications will be developed by using Jax-Rpc, Jax-Ws, Axis and etc APIs.

Use Jax-Rpc API to have synchronous communication between Service client and service provider.

Use Jax-Ws and Axis API to have both synchronous and asynchronous communication between Service client and service provider.

If client is blocked or sitting idle to perform next operation until the given request related response comes from server, then that is synchronous communication between client and server.

If client is free to perform next operation without waiting for given request related response from server, then it is called as Asynchronous communication between client and server.

e.g., (i) The login page request of gmail.com is an example for synchronous communication.

(ii) Opening email messages in the Inbox without stopping chatting operations in gmail.com is an example for asynchronous communication.

In WebServices based distributed application, the service provider application/server application is always one web application having business component.

In WebServices environment, service client interacts with service provider by using SOAP over Http Protocol. That means Http request contains SOAP request messages as body. Similarly Http response contains SOAP response messages as body.

For the related information on Http, SOAP over Http, refer page no.s 9 to 10 of the given hand-out on 30th August 2011.

SOAP is XML based protocol. It always carries data in the form of XML tags and attributes. SOAP cannot run as an independent protocol, so it must be embedded with the main application level protocols like Http, SMTP and etc..

In WebServices, SOAP will be used by embedding with HTTP. In WebServices, service client sends its business method call based request to service provider as SOAP request embedded with HTTP Request. Similarly Service provider sends its business method execution result as SOAP response embedded with Http response (refer page no. 12).

3rd Sep '11

XML DTD ; XML XSD (XML Schema Definition) are the two different techniques which can be used to define set of rules to construct XML documents / XML data. This rules contain tags and attributes which have to be used to create XML document, the structure and hierarchy of tags in XML document and etc..

While defining XML Schema definition rules, name spaces will be used by specifying their URIs.

XML Name space is a library that contains set of XML Tags.

Every name space will be identified with its URI.

SOAP base related tags are available in http://schemas.xmlsoap.org/soap/envelope URI based XML name space.

The SOAP request and response messages will be constructed based on XSD rules.

For related information on XML Schema, refer page nos 19-21 and also refer page nos 5-8 of the booklet given on 30th August 2011.

For related information on SOAP, refer page numbers 11-19 (30th August 2011).

The top most element in SOAP message is <envelope>.

Every SOAP message contains optional headers and mandatory body.

The tags of SOAP messages are defined in XML Schema name spaces.

An example SOAP request:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/
                    soap/envelope/">
    <S:Header/> (optional)
    <S:Body>
        <ns2:sum xmlns:ns2="http://sathya/">
            <val1>10</val1>
            <val2>20</val2>
        </ns2:sum>
    </S:Body>
</S:Envelope>
```

the XML Schema name space URI when <envelope>, <header>, <body> tags are defined.

name space URI when <sum> is called

Request data in the form of XML

An example SOAP Response:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap
                                    /envelope/">
    <S:Body>
        <ns2:sumResponse xmlns:ns2="http://sathya/">
            <return>30</return>
        </ns2:sumResponse>
    </S:Body>
</S:Envelope>
```

result data in the form of XML

A WSDL document contains the complete details about WebService enabled business component in the form of XML document.

For more information about WSDL, refer page no.s 21 - 26 (30th August 2011).

UDDI is XML based Registry where the WSDL documents of Web Services will be maintained.

www.UDDI.org acts as public UDDI Registry.

The UDDI registry of every Web Server and Application server acts as a private UDDI Registry.

For related information of UDDI, refer page no. 27-29 (30th August 2011).

9sep'11

The client and server applications of Web Services can be developed by using different APIs like

JAX-RPC API } SUN MS
JAX-WS API }

AXIS API - APACHE Foundation

In Web Services application, the server appln should always be developed as web application. So it will be developed either as WAR file or EAR file.

EAR file = WAR file + JAR file + ...
EAR file = WAR file + WAR file + ...
EAR file = JAR file + JAR file + ...

To know the current active version of Jdk, use <cmd prompt> java -ver

Setup required for developing JAX-RPC based Web Service Application:

jdk 1.4 (use jdk that comes with Weblogic 8.1 s/w)
Weblogic 8.1.

To When multiple versions related Java_home\bi directories are placed in path environment var. the first value related Jdk version will be activated.

Weblogic 8.1:

Type : Application Server
Version: 8.x (compatable with jdk 1.4)
Vendor: BEA Systems
Default port no. : 7001 (changable through config.xml fi
Commercial s/w.
Allows to create domains.
Default domain : examplesServer
Each domain acts as one ApplicationServer.

If multiple projects of a company are using same Weblogic s/w, then the Weblogic s/w will be installed only once in a common computer and multi domains will be created in that s/w for multiple projects.

To Download this s/w : www.commerce.bea.com
For documents : www.edocs.bea.com

Procedure to create User defined domain in Weblogic 8.1 server:

Start → Programs → BEA Weblogic Platform 8.1 → Configuration Wizard → ⊙ Create new Weblogic configuration → Next → select Basic Weblogic Server Domain → Next → ⊙Express → Next → User name: [Javaboss] (min. of 8 letters)

User password: [javaboss]

confirm password: [javaboss] → Next →

Next → Configuration Name: [WsBDomain] → Create → Done.

To change the default port no. of above domain server, Goto BEA_home\user_projects\domains\WsBDomain\config.xml file and modify the listen port attribute value of <Server> tag.

To start the above domain server:

Start → Programs → BEA Weblogic Platform 8.1 → User projects → WsBDomain → Start Server

Procedure to develop JAX-RPC based Web Service enabled Distributed Application:

Step-1: Activate Jdk 1.4 in your computer

My Computer → Properties → Advanced Tab → Environment variables →

Variable name: ~~CLASS~~PATH

value : D:\bea 8.x\jdk 1.4.1_05\bin;<other existing val>;

→ OK → OK → OK.

Step-2: Add the following Jar files to CLASSPATH to work with servicegen, clientgen tools supplied by Weblogic.

BEA 8.x_HOME\weblogic 8.1\server\lib\weblogic.jar
dependent jar file to webservices.jar

BEA 8.x_HOME\weblogic 8.1\server\lib\webservices.jar
main Jar file

BEA 8.x_HOME\jdk 1.4.1_05\tools.jar
dependent jar file to webservices.j

Note: Servicegen tool generates EAR file representi web service based on the given Java class.

Note: Clientgen tool generates JAR file having helper code that is required for Client application development and execution.

Step-3: Develop the Java class based on which you want to develop web Service based Server application.

Note: Developing Web Service is nothing but developing the Server application of Web Service

E:\Apps
- WebServices
  - App1
    - Server
      - Calculator.java
      - Calculator.class

Calculator.java:

```
public class Calculator
{
  //Operation 1 (Business method 1)
   public int add (int a, int b)
   {
     return a+b;
   }
   public in sub (int a, int b)  //Operation 2
   {
     return a-b;
   }
}
```

> javac Calculator.java

Step-4: Use the Weblogic supplied servicegen tool to get EAR file representing Web Service based on the above Calculator class.

E:\Apps\WebServices\...\Server > java weblogic.webservice.servicegen

E:\Apps\WebServices\App1\Server > java weblogic.webservice.servicegen -destEar e:\Apps\WebServices\App1\server\calc.ear -warName calc.war -javaClassComponents Calculator -serviceName calcy -serviceURI /sathyacalcy -targetNamespace http://www.sathyacalcy.com

E:\Apps\WebServices\App1\Server> java weblogic.webservice.servicegen
- destEar e:\Apps\WebService\App1\Server\calc.ear
- warName calc.war
- javaClassComponents Calculator
- serviceName calcy
- service URI/satyacalcy (give space b/w URI and /)
- targetNamespace http://www.satyacalcy.com

The above command generates calc.ear representing WS.

To know about the above tool related options type the following command

> java weblogic.webservice.servicegen

Note: The above generated calc.ear file internal contains calc.war file and moreover the generated web service will be identified with logical name calcy and with URI satyacalcy

Step-5: Deploy the above generated web service in Domain server (WSBDomain) of Weblogic

Copy the above calc.ear file to

BEA 8.1_HOME\user_projects\domains\WSBDomain\Applications folder

Step-6: Start the WSBDomain server of Weblogic.

Note: When Calc.ear file is deployed, the generated WSDL documents will be registered automatically with the built-in UDDI Registry of Weblogic Server.

Step-7: Use the following url, to see the WSDL document.

http://localhost:7070/calc/satyacalcy?WSDL
(war file name)

Note: for clients, the above Web service is visible and accessible through the following URL.

http://localhost:7070/calc/satyacalcy
(collect from <service> tag of WSDL document)

Note: Every Web Service will be identified with one URL/URI specified in its WSDL document.

Sep'11

Procedure to develop the Java based Web service Client Application for the above developed WebService:

Step-1: Add the following three Jar files in CLASSPATH
(i) weblogic.jar
(ii) webservices.jar
(iii) tools.jar (JdK 1.4 supplied)

Step-2: Create seperate folder to develop and execute client application.

E:\Apps
|→ WebServices
|→ App1
|→ Client

Step-3: Use clientgen tool to generate helper resources that are required in the development of WebService client application

E:\Apps\webservices\App1\Client> java weblogic.webservice.clientgen
-clientjar E:\Apps\webservices\App1\Client\helper.jar
*1
-wsdl http://localhost:7070/calc/satyacalcy?WSDL
*2
-packageName pack1
*3
-keepGenerated false
*4

*1 – name of the Jar file into which the helper resources will be generated.

*2 – the WSDL URL of above generated webservice.

*3 – package name for the resources (classes) of helper.jar.

*4 – "false" indicates that .java's will be deleted after generating .class files in helper.jar file where as "true" indicates that .java's will be preserved even after generating .class files in helper.jar. The default value is 'true'.

Note: Make sure that only Jdk1.4_HOME\bin directory is there in the path environment variable (other jdk versions related bin directories must not be placed).

Note: Make sure that the WSBDomain of Weblogic 8.1 server is in running mode.

To know various options of clientgen tool, use the following command.

E:\Apps\webservices\App1\Client> java weblogic.webservice.clientgen

Step-1: Develop the WebService Client as shown below.

TestClient.java:(E:\Apps\WebServices\App1\Client)

```
import pack1.*;
class TestClient
{
 p.s.v. main(String[] args) throws Exception
 {
   S.o.pln("TestClient.....!");
   String wsdlUrl = "http://localhost:7070/calc/satyacalcy?
                                                WSDL";
   //locate webservice component on server
   Calcy_Impl service = new Calcy_Impl(wsdlUrl);
   // get Business object reference of Web Service
       component
   CalcyPort port = service.getcalcyPort();

   // call the business methods
   S.o.pln("The sum is : "+port.add(12,30));
   S.o.pln("The sub is : "+port.add(50,30));
  }
}
```

The helper classes generated in helper.jar file

The above client application is running outside the Weblogic Server. So this client (client application) is called as Remote client

Step-5: Add the above generated helper.jar file to CLASSPATH environment variable.

Step-6: Compile and execute the Client application.

```
E:\Apps\WebServices\App1\Client>javac TestClient.java
E:\Apps\WebServices\App1\Client>java TestClient
TestClient ..... !
The sum is 42.
The sub is 20.
```

The above client application is generating Http Request to reach web service component and to call business methods.

Procedure to develop the .NET based Web Service Client Application for the above developed Web Service:

Step-1: Launch Microsoft Visual Studio 2008.

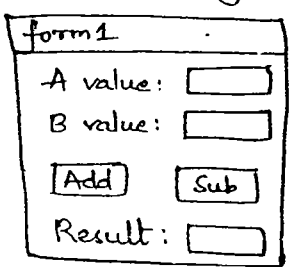
Start → Programs → Microsoft Visual Studio 2008 → Microsoft Visual Studio 2008.

Step-2: Create new project

File menu → new → project → Visual C# → Windows Forms Application →

Name: [MyTestProj1] → Ok

Step-3: Design form page as shown below



Step-4: Add Web Service reference to the project.

Go to Solution Explorer... window → right click on project (MyTestProj1) → Add service reference → Address: http://localhost:7070/calc/satycalcy?WSDL → Go → Name space: S1 → OK

Step-5: Write the following event handling code for Add button.

Double click on Add button and write the following logic in button1_Click(-,-) method.

```
pub. void bu    (object sender, EventArgs e)
    {
      S1.calcyPortClient obj1 = new MyTestProj1.S1.
                                calcyPortClient();
        * points to Web service component

reading values from text boxes { int a = Convert.ToInt32(textBox1.Text);
                                 int b = Convert.ToInt32(textBox2.Text);

      int res = obj1.add(a,b); //calling business method
                               of WebService Component
      textBox3.Text = res.ToString(); //setting result to textBox
    }
```

Step-6: Write the following event handling code for Sub button.

Double click

```
public void button2_Click(-,-)
  {
    S1.calcyPortClient obj1 = new MyTestProj1.S1.
                              calcyPortClient();
    int a = Convert.ToInt32(textBox1.Text);
    int b = Convert.ToInt32(textBox2.Text);

    int res = obj1.sub(a,b);

    textBox3.Text = res.ToString();

  }
```

Step-7: Run the .NET client application.

Procedure to write C# based CUI client for the above Java based Web Service Component:

Step-1: Create new project.

File menu → project → Visual C# → Console Application → MyTestProj2 → OK

Step-2: Add Web Sewice reference to the project having logical name S2.

Step-3: Write the following code in the ~~Main(-)~~ method.

```
class Program
{
  static void Main(string[] args)
  {
    S2.calcyPortClient obj1 = new MyTestProj2.S2.
                              CalcyPortClient();
    int a=10;
    int b=20;
    int res1 = obj1.add(a,b);
    int res2 = obj1.sub(a,b);
    Console.WriteLine("Result1 is :" +res1);
    Console.WriteLine("Result2 is : "+res2);
    Console.Read();
  }
}
```

Step-4: Run the Application (Start debugging)
Press ▷ (Debug) button

nsep'11

Procedure to develop WebService in .NET environment:

Step-1: Launch Visual Studio 2008

Step-2: Create project pointing to ASP.NET Web Service application.

File menu ⟶ new ⟶ Project ⟶ Visual C# ⟶ ASP.NET Web Service application ⟶

Name: TestProj3 ⟶ OK

S Note: The above project gives one web service

having default business method called HelloWorld()

Step-3: Run the pro application.

Step-4: Gather the WSDL url of the above .NET Web Service

http://localhost:1142/Service1.asmx?WSDL

Procedure to develop JAX-RPC based Java client for the above .NET Web Service:

Step-1: Create Work folder as shown below.

E:\apps
  ⟶ Webservices
    ⟶ JavaClient

Step-2: Generate helper resources related Jar file from the work folder by using clientgen tool and the above WSDL url.

E:\apps\webservices\javaClient>java weblogic.webservice.clientgen
-clientjar E:\apps\webservices\javaClient\helper2.jar
-wsdl http://localhost:1142/Service1.asmx?WSDL
-packageName satya
-keepGenerated false

Note: The above tool gives helper.jar file

Step-3: Add the above step generated helper.jar file to CLASSPATH.

Step-4: Develop the Java Web service client as shown below

```
//TestClient.java : (E:\apps\webservices\javaClient)
import satya.*;
public class TestClient
{
  p.s.v. main(String[] args) throws Exception
  {
    S.o.pln("TestClient -----!");
    String wsdlurl="http://localhost:1142/Service1.asmx?
                                        WSDL";
    //locate .net Web service component
    Service1_Impl service=new Service1_Impl(wsdlurl);

    //get b. object refer. of .net Web service comp.
    Service1Soap port=service.getService1Soap();

    //call the b. method
    S.o.pln("The result is "+port.HelloWorld());
  }
}
```

Step-5: Compile and execute the client application

```
E:\apps\webservices\javaClient>javac TestClient.java
E:\apps\webservices\javaClient>java TestClient
```

While upgrading old projects or existing projects, we need to make the business components of the projects as inter operatable components. For this we need to convert existing business components

Procedure to develop .NET client for the above EJB 2.x based Web Service component:

Step-1: Create project pointing to Visual C# console application.

File menu → new → Visual C# → Console application → Test Name: TestProj4 → OK

Step-2: Add Web Service reference to the project

Right click on the project → Add Service Reference →

Address: http://localhost:7070/MyWls/satyaws1?WSDL

→ go → Name space: S3 → OK

Step-3: Write the following code in the Main() method of the application and then start debugging (press ▷ (debug) button).

```
S3.ws1PortClient obj1 = new TestProj4.S3.ws1Port
                                    Client();
        Console.WriteLine("Enter val1");
String  val1 = Console.ReadLine();
int a= Convert.ToInt32(val1);

Console.WriteLine("Enter val2:");
String val2 = Console.ReadLine();
int b = Convert.ToInt32(val2);

int result = obj1.findSum(a,b); //calling B. method
Console.WriteLine(" Result is: " + result);
Console.Read();
```

08th Sep'11

Procedure to develop Java based JAX-RPC based Web Service Client for the above EJB 2.x component converted into Webservice:

Step-1: Create seperate work folder for this Client application.

E:\apps
  ⟶ Webservices
    ⟶ EJB-2.x WS
      ⟶ Client

Step-2: Use clientgen tool to generate helper jar file containing helper resources.

E:\apps\webservices\EJB2.xWS\client> java weblogic.webservice clientgen
- clientjar E:\apps\webservices\EJB2.xWS\client\wshelper.jar
- wsdl http://localhost:7070/MyWs/satyaws1?WSDL
- packageName satya
- KeepGenerated false

Step-3: Develop the WebService client application

Refer client1.java of supplementary handout (07th Sep'11).

Step-4: Add the generated helper jar file (wshelper.jar) to the CLASSPATH.

Step-5: Compile and execute the Client application

Note: To compile the above .java files, use javac -d option and also add weblogic.jar file to the CLASSPATH.

E:\apps\ ..... \ejbcomp> javac -d . *.java

Step-2: Prepare Jar file on the above deployment directory structure representing EJB component.

E:\apps\...\ejbcomp> jar cf comp1.jar .

Step-3: Use servicegen tool to convert the above EJB component into web service.

E:apps\----\ejbcomp>java weblogic.webservice.servicegen
-destEar E:\apps\webservices\EJB2.xWS\ejbcomp\MyWs.ear
- warName MyWs.war
- ejbJar E:\apps\webservices\EJB2.xWS\ejbcomp\comp1.jar
  (Jar file that represents EJB component.)
- serviceName WS1
- serviceURI /satyaws1
- targetNamespace http://www.satyaws1.com

Step-4: Deploy the above generated MyWs.ear file (represents Web service component) in WsBDomain server of Weblogic 8.1 server.

Copy MyWs.ear file to <BEA 8.x_HOME>\user_projects\domains\WsBDomain\applications folder.

Step-5: Start WsBDomain server of Weblogic 8.2.

Step-6 Use the following URL to get the WSDL document.

http://localhost:7070/MyWs/satyaws1?WSDL
(MyWs: war file name; satyaws1: service URI)

into Web Service components.

In most of the old projects, EJB 2.x is used to develop business components and we can use the JAX-RPC support to convert them into Web Service components.

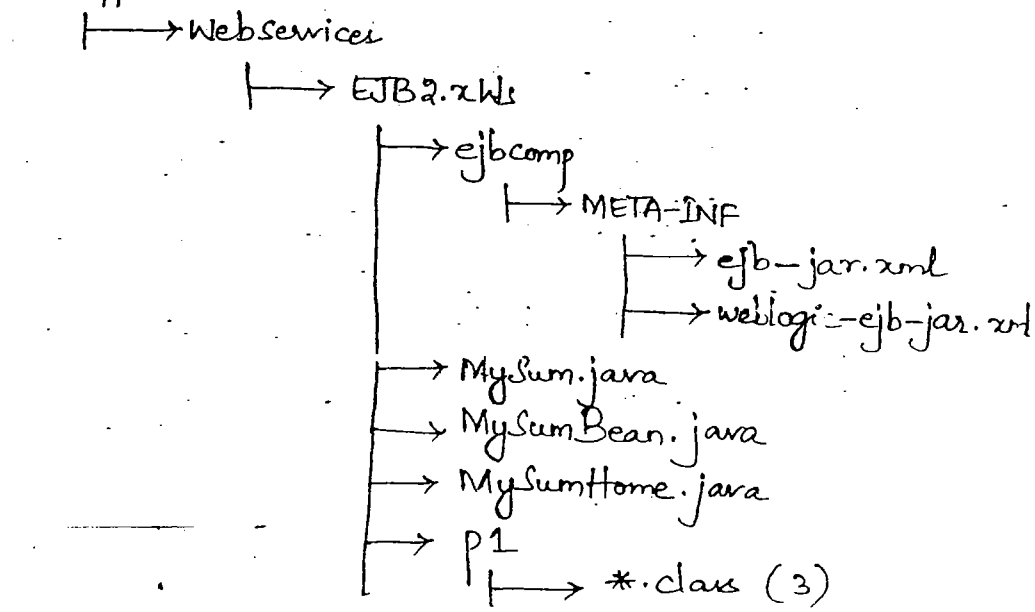
We can use JAX-WS to convert EJB 3.x components into Web services.

The plain EJB component allows only Java clients where as the EJB component that is converted into web service allows both Java and non Java clients.

For EJB 2.x component code, refer the supplementary handout given on 07th September 2011.

Procedure to convert EJB 2.x component give in handout to Web Service component:

Step-1: Prepare the deployment directory structure of EJB 2.x component.



//TestClient.jsm : (F:\ame\webservices\jsmclient)

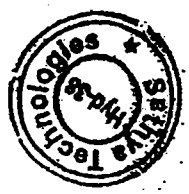
```
</ejb-jar>

-------------------------weblogic-ejb-jar.xml-------------------------
<!DOCTYPE weblogic-ejb-jar PUBLIC "-//BEA Systems, Inc.//DTD WebLogic 8.1.0 EJB//EN" "
<weblogic-ejb-jar>
    <weblogic-enterprise-bean>
        <ejb-name>MyEJB</ejb-name>
        <jndi-name>MyJndi</jndi-name>
    </weblogic-enterprise-bean>
</weblogic-ejb-jar>

-------------------------Client1.java-------------------------
//Client1.java
import satya.*;

public class Client1
{
    public static void main(String args[])throws Exception
    {
        String wsdlUrl = "http://localhost:7070/MyWs/satyaws12WSDL";

    // locate webservice comp
        Ws1_Impl service= new Ws1_Impl(wsdlUrl);
        // get webservice comp obj ref
        Ws1Port bobj=service.getws1Port();

    // call B.method of ejbcomp
        int sum=bobj.findSum(10,20);
        System.out.println("the sum is "+ sum);
    }
}
```

Vendor Specific DD file

into Web Service components.

07th September 2011

Title: Converting EJB2.x comp as webservice



MySum.java — Business Interface

```
//MySum.java (B.Interface)
package p1;
import java.rmi.*;
import javax.ejb.*;
public interface MySum extends EJBObject
{
    public int findSum(int a,int b) throws RemoteException;
}
```

declaration of b. method

MySumBean.java — Bean class

```
//MySumBean.java (Bean class)
package p1;
import javax.ejb.*;

public class MySumBean implements SessionBean
{
    public void ejbActivate(){    }
    public void ejbRemove(){}
    public void ejbPassivate()   {}
    public void setSessionContext(SessionContext cix){}
    public void ejbCreate()throws CreateException{    }
    // Impl of B.method
    public int findSum(int dx,int dy)
    {
        return dx+dy;
    }
}
```

// definition of b. method

MySumHome.java — Home Interface

```
//MySumHome.java (Home Interface)
package p1;
import java .rmi.*;
import javax.ejb.*;

public interface MySumHome extends EJBHome
{
    MySum create()throws CreateException,RemoteException;
}
```

ejb-jar.xml — DD file

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN" "http://java.sun.com
<ejb-jar>
    <enterprise-beans>
    <session>
        <ejb-name>MyEJB</ejb-name>
        <home>p1.MySumHome</home>
        <remote>p1.MySum</remote>
        <ejb-class>p1.MySumBean</ejb-class>
        <session-type>Stateless</session-type>
        <transaction-type>Container</transaction-type>
    </session>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>MyEJB</ejb-name>
                <method-intf>Remote</method-intf>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
```

| | |⟶ *.class (3)

Annotations:

Annotations are Java statements and alternate for the XML files based resources configuration and METADATA operations.

Data about Data is called MetaData.

Passing details to underlying container by specifying the details of resource in XML file is some kind of MetaData operation.

The XML based resources configuration gives good flexibility for modification but poor performance because of the XML Parsers or heavy weight applications to read and process XML documents.

Annotations based MetaData, resources configu-ration gives good perfoimance but poor flexibilit of modification.

In Core Java programming, Annotations are given to instruct JRE/JVM regarding various resources of the application.

Documentation Annotations are there in Java from jdk 1.0. onwards.

Annotations for programming are introduced from jdk 1.5.

All Java technologies that are compatible with jdk 1.5+ are giving support for Annotations

e.g.s: Servlet 3.0, Struts 2.x, EJB 3.x, Spring 2.5+, etc..

· JAX-WS supports Annotations based programming.

e.g:

@<annotation-name>(param1=val1, param2=val2, ...)

these parameters are like attributes of XML Tags

Every Annotation is a special Interface in Java and parameters of Annotation come as methods declared in the special Interface. Every technology related API supplies certain predefined annotations for resources configuration.

The underlying container/server/framework/JRE uses the annotation to gather the details about annotations.

We can apply annotations at 3 levels.

1. Class/Interface Level (Resource Level)
2. Field Level
3. Method Level

JAX-RPC supports only Synchronous communication and is compatible with jdk 1.4 environment and does not give support for Annotations.

JAX-WS supports both Synchronous and Asynchronous communication between Web Service client and Web Service component. JAX-WS is compatible with jdk 1.5+ and also gives support for Annotations.

09th Se

Setup required for working with JAX-WS:

jdk 1.6
Glassfish 2.x / Weblogic 10.3

Note: JAX-WS API is built-in API of jdk 1.6

Glassfish:

Type :

We can work with NetBeans IDE supplied Glassfish server with or without NetBeans IDE.

Procedure to modify the port no. of domain1 server:

Use <Glassfish_home>\AppServer\domains\domain1\config\domain.xml file and modify the port attribute value of first <http-listener> (line no. around 64).

09th Sep'11

Differences between JAX-RPC and JAX-WS

1. JAX-RPC supports SOAP 1.1 where as JAX-WS supports SOAP 1.1 and SOAP 1.2.
2. JAX-RPC supports WSDL 1.1 where as JAX-WS supports WSDL 1.1 and WSDL 1.2.
3. JAX-RPC does not support HTTP binding (i.e.,

sending XML messages over HTTP with SOAP) where as JAX-WS supports Http Binding.

4. JAX-RPC supports WS-I BP 1.0 where as JAX-WS supports WS-I BP 1.1.

Note: WS-I is the web services interoperability organisation providing standards to develop Interoperatable Web service component.

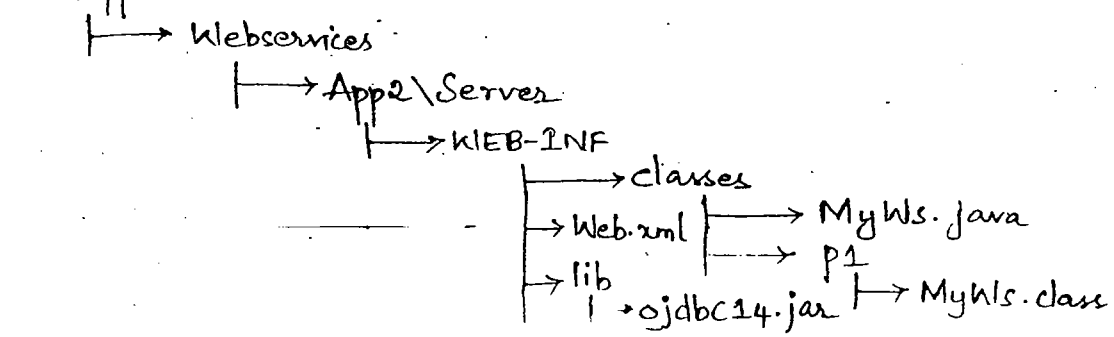
5. JAX-RPC maps to Java 1.4, J2EE 1.4 features where as JAX-WS maps to Java 5, Java EE 5.0 features.

6. JAX-RPC supports only synchronous communication between Web service client and web service component. JAX-WS supports both synchronous and asynchronous communication.

Procedure to develop JAX-WS based Web Service component:

Step-1: Activate jdk1.6 in your computer.

Step-2: Develop the Web Application Deployment structure as shown below having the Java class as Web service component.



MyWs.java

```
import package p1;
import javax.jws.WebService;
import javax.jws.WebMethod;
import java.util.*;
import java.sql.*;

@WebService     //class level Annotation
public class MyWs
 {
   @WebMethod   /method level annotation
   public String generateWishMsg(String uname)
    {
      Calendar cl = Calendar.getInstance();
      int h = cl.get(Calendar.HOUR_OF_DAY);
      if (h<12)
        return "Good morning: "+uname;
      else if(h<16)
        return "Good afternoon: "+uname;
      else if (h<20)
        return "Good evening: "+uname;
      else
        return "Good night: "+uname;
    } //method

   @WebMethod     //method level annotations
   public float fetchSalary(int eno)
    {
```

```
float bs=0.0f;
try
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con= DriverManager.getConnection(
      "jdbc:oracle:thin:@localhost:1521:orcl",
                              "scott","tiger");
  Statement st= con.createStatement();
  ResultSet rs = st.executeQuery("select sal
                  from emp where empno="+eno);
   if(rs.next())
   {
     bs = rs.getFloat(1);
   }
 }//try
catch(Exception e)
 {
   e.printStackTrace();
 }
 return bs;
}//method
}// class
```

// > javac -d . MyUtls.java

Note: No need of adding any Jar files to the CLASSPATH.

Web.xml:

```
<web-app/>
```

Step-3: Prepare WAR file representing the above web application.

```
E:\apps\Webservices\App2\Server>jar cf MyWsApp.war .
```

Step-4: Start the domain1 server of Glassfish

Start → Programs → Sun Microsystems → Application serve → Start default server (domain1)

Step-5: Deploy the above web application that represents web service component in domain1 server of Glassfish.

Copy the MyWsApp.war file to the

D:\Sun\AppServer\domains\domain1\autodeploy
<GlassfishFish_home>
folder.

Step-6: Get the WSDL document for the above WebService component. (Type the below URL)

http://localhost:6060/MyWsApp/MyWsService?WSDL
port no. WAR file name class name fixed word

Procedure to develop JAX-WS based Web Service Client for the above Web Service Component:

Step-1: Create work folder for Client application

E:Apps
|→ Webservices
|→ App2
|→ Client

Step-2: Use "wsimport" tool by specifying the WSDL url to generate helper resources that are required for Client application development.

Note: wsimport is the built-in tool of jdk 1.6 software.

E:\apps\WebServices\App2\Client > wsimport -p mypack -keep http://localhost:6060/MyWsApp/MyWsService?WSDL
*1 *2 *3

*1 The package in which the generated helper resources will be placed.

*2 This preserves .java files of helper resources.

*3 The WSDL url of the above web service component.

Step-3: Develop Web Service Client application.

TestClient.java (E:\apps\Webservices\App2\Client)

```
import mypack.MyWs;
import mypack.MyWsService;   } helper resources

public class TestClient
{
  p.s.v. main(String args[]) throws Exception
  {
    //locate web service component
    MyWsService cs = new MyWsService();
    //get access to b.obj ref. of Web Service comp.
    MyWs bobj = cs.getMyWsPort();
    //call b. methods
    String msg = bobj.generateWishMsg("raja");
    S.o.println("Wish Message "+msg);
```

```
        S.o.pln ("Emp. sal is "+bobj.fetchSalary(7934));
    } // main()
} // class
```

Step-4: Compile and execute the client application

```
E:\apps\webservices\App2\Client>javac TestClient
E:\apps\webservices\App2\Client>java TestClient
```

We can also develop .NET Client application by using the above WSDL url.

Note: While running these Client applications, make sure that the Glassfish server is in running mode.

We can develop JAX-RPC based Client appl. for the above JAX-WS based Web Service compone by taking the support of clientgen tool.

Step-1: Take seperate work folder for JAX-RP based Client.

```
E:\apps
    |--> WebServices
            |--> App2
                    |--> Client (jax-rpc)
```

Step-2: Make sure that the following 3 jar file are added to the CLASSPATH.

(i) weblogic.jar
(ii) webservices.jar
(iii) tools.jar

Step-3: Use clientgen tool as shown below.

```
E:\apps\.....\Client(jax-rpc)>java weblogic.webservice.clientgen
  -clientjar E:\apps\....\Client(jax-rpc)\helper.jar
  -wsdl http://localhost:6060/MyWlsApp/MyWlsService?WSDL
  -packageName pack1
  -keepGenerated false
```

10th Se

Step-4: Add the above generated helper.jar file to CLASSPATH.

Step-5: Develop the client application as shown below and compile, execute the application.

TestClient1.java:

```
import pack1.*;
class TestClient1
 {
  p.s.v. main(String args[]) throws Exception
   {
     S.o.pln("TestClient1 ---!");
     String wsdlUrl="http://localhost:6060/MyWlsApp/
                     MyWlsService?WSDL";
     //locate web service comp. on server
    MyWlsService_Impl service=new MyWlsService_Impl(
                                  "wsdlUrl");
```

```
        //get b. obj ref to Web service component
        MyWs bobj = service.getMyWs();
        //call the b. methods
        S.o.pln("Wish Msg "+bobj.generateWishMsg("ravi"));
        S.o.pln("Salary is "+bobj.fetchSalary(7934));
    }//main
}//class
```

10th Sep '11

We cannot write JAX-WS based Web Service Client for JAX-RPC based Web Service component (the reason is the incompatibility with the supported SOAP versions).

JAX-WS support gives its web service component and client based on SOAP 1.2 where as JAX-RPC gives web service component and client based on SOAP 1.1.

NetBeans

Type : IDE s/w for Java environment
Version : 7.0.1 (compatible with jdk 1.6/1.7)
Vendor : Sun MicroSystems
Open source s/w
Gives Glassfish as a built-in server (Glassfish 3.1.1)

Procedure to develop JAX-WS based Web Service component using NetBeans 7.x:

Step-1: Create Web project in NetBeans IDE

File menu → New Project → Java Web → Web appl. → Next → Project name: [TestWsProj] → Next → Server: [Glassfish 3.1] Context path: [/TestWsProj] → Next → finish → delete index.jsp

Step-2: Add Web Service to the project

Rt. cl. on project → New → Web Service → Web service name: [ArithWs] package: [p1] → ⊙ Create Web Service from scratch → finish

Step-3: Add operations / business methods in the generated Web service related class (add sum(+,+), sub(-,-) methods).

As example: Rt. click inside ArithWs class → Insert code → Add Web Service operation → Name: [sum] Return type: [int] → Parameters tab → Add x, y parameters → OK

Step-4: Run the project

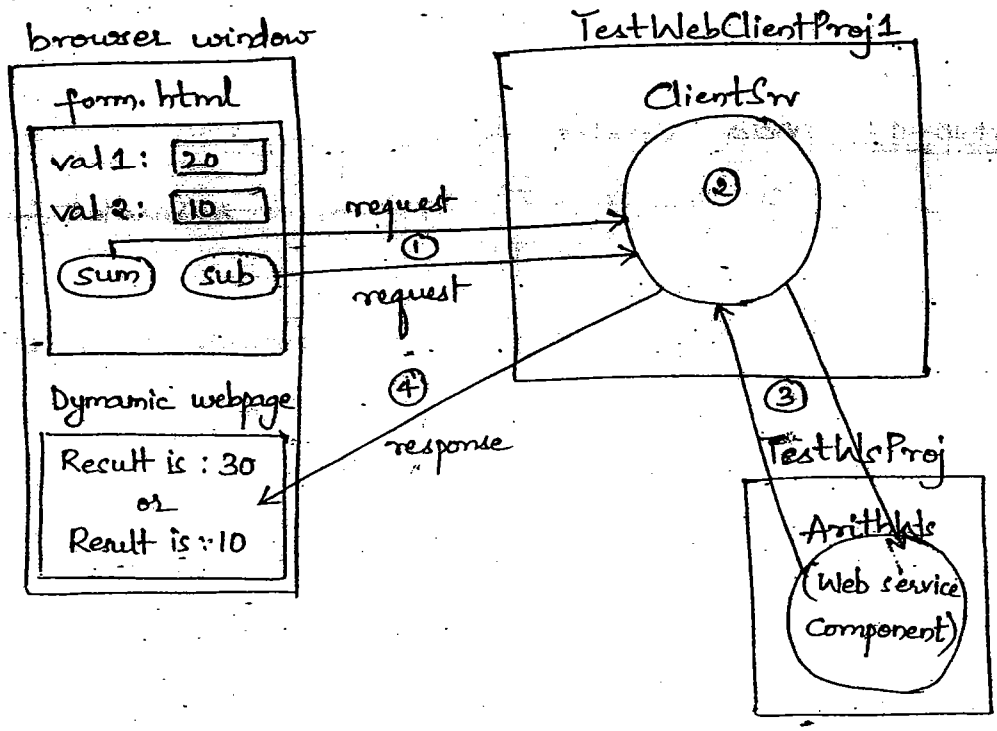
Rt. cli. on project → Run

Step-5: Test the Web Service

Expand project → Expand Web services → Rt. cl. on ArithWs → Test Web service →

The WSDL url of the above Web Service compoen is http://localhost:6060/TestWsProj/ArithWs?WSDL
(ArithWs: Service Name)

Procedure to develop Java Web application as Web Service client for the above Web Service Compone



Step-1: Create project in NetBeans IDE having name TestWebClientProj1 (Web project)

Step-2: Add Web Service client to the project
Rt. cli. on project → New → Web Service Client → ⊙ Project → Browse → select ArithWs of TestWsProj → package: P1 → Finish (This will add Web service reference to the project).

Step-3: Add form page to the Web pages folder of the project.

form.html

```
<form action="testurl" method="get">
    value 1 : <input type="text" name="t1"> <br>
    value 2 : <input type="text" name="t2"> <br>
    <input type="submit" value="sum" name="s1" />
    <input type="submit" value="sub" name="s1" />
</form>
```

Step-4: Add Servlet to the project

Rt. cl. on project → New → Servlet →

class name : ClientSrv → Next →

URL pattern : /testurl → finish → place your request processing logic in processReq(-,-) generated source code. (It is helper method called from doGet(-,-), doPost(-,-) methods).

In processRequest(-,-) method,

```
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
//read form data
int a = Integer.parseInt(request.getParameter("t1"));
int b = Integer.parseInt(request.getParameter("t1"));
String cap= request.getParameter("s1");
if(cap.equals("sum"))
 {
    int res= sum(a,b);
    pw.println("Sum is :"+res);
 }
```

```
else
 {
    int res = sub(a,b);*
    pw.println("Sub is: "+res);
 }
```

* → the helper methods which will be generated through next step (Step-5)

Step-5: Add Web service client code to Servlet calling business methods.

Expand TestWebClientProj1 ⟶ Expand Web Service References ⟶ Expand "ArithWs" (client name) ⟶ Expand ArithWs (service name) ⟶ Expand ArithWsPo ⟶ Drag and drop sum, sub methods to any part of the Servlet (ClientSrv) (generate 2 helper methods calling Web Service compone business methods internally).

Step-6: Run the Client project.

Rt. cl. on Client project (TestWebClientProj1) ⟶ Run ⟶ Type this url.

http://localhost:6060/TesthlebClientProj1/form.htm

Developing Standalone Web Service Client by using NetBeans IDE for the above Web Service Component

Step-1: Create Java project in NetBeans IDE

file menu ⟶ New project ⟶ Java ⟶ Java Application ⟶ Next ⟶ Project Name: TestProj1 ⟶ finish.

Step-2: Add Web Service client to project pointing to the above ArithWs Web Service Component.

Step-3: Add application to project having main() method.

Rt. cl. on project → New → Java class → Class name: TestClient → finish

TestClient.java

```
public class TestClient
{
  p.s.v. main(String[] args)
  {
     S.o.pln (sum(10,20));
     S.o.pln (sub(20,10));
  }
}
```

- Drag and drop business methods to TestClient.java from Web Service References of the project.

Step-4: Run the client application (TestClient)

Rt. cli. in the source code of TestClient.java → Run file

15ep'11

Using JAX-WS, we can convert EJB 3.x component into Web Service. To develop EJB 3.x component, 2 resources are required.

1. Business Interface
   - Declaration of business methods
2. Bean class
   - Implementation of business methods

To develop SessionBean EJB Component, the following Annotations are required.

(i) @Remote
- to make the business interface as remote b.interface

(ii) @Stateless
- to make/develop SessionBean component as Stateless SessionBean component.

(iii) @Stateful
- to develop SessionBean component as a Stateful Session bean component.

Stateless SessionBean component does not remembers Client data during a Session.

Stateful SessionBean component remembers Client data during a Session.

Set of continuous and related operations performed on EJB component by a Client through b. method calls is called as a Session.

(iv) @Local
- to make B. interface as Local B. interface.

Procedure to convert EJB 3.x Stateless SessionBean component into Web Service:

~~Step-1~~: To develop EJB Component (in NetBeans 6.7.1)

Step-1: Create the JEE based EJB project.

file menu → New Project → Java EE → EJB module → Next → Project Name: TestProj11 → Next → finish → Server: GlassFish 2.1 → finish.

Step-2: Add Stateless Session Bean component to the project.

Rt. click on project → New → Session Bean →

EJB Name: Hello1 → package: p1 →

Session Type: ⊙ Stateless

Create Inteface: ☑ Remote → finish

Note: The above wizard generates business interface Hello1Remote.java and Bean class Hello1Bean.java.

Step-3: Add b. method to Bean class.

Inside the class definition, Rt. click → insert code → Add business method →

Name: sayHello — Return type: java.lang.String

→ Parameters tab → Add uname type: java.lang.String → Use in interface: ⊙Remote

→ OK → Add business logic.

Hello1Bean.java:

```
public class HelloBean implements Hello1Remote
 {
   public String sayHello(String uname)
    {
      return "Good moining "+uname;
    }
}
```

```
package p1;
import javax.ejb.Stateless;
```

Hello1Remote.java:

```
package P1;
import javax.ejb.Remote;
@Remote
public interface Hello1Remote
{
    String sayHello(String uname);
}
```

Step-4: Run the project to deploy the component in Glassfish server

Rt. click on project ⟶ Run

To Convert the above EJB Component into Web Service

Step-1: Create Web project in NetBeans IDE.

file menu ⟶ New Project ⟶ Java Web ⟶ Web Application ⟶ Next ⟶ Project name: TestWebProj ⟶ Next ⟶ Server: Glassfish 2.1 ⟶ Next ⟶ finish ⟶ delete Index.jsp.

Step-2: Copy the business interface of the above EJB component to the source packages folder of current web project (make sure that here also it is there in p1 package) (Hello1Remote.j

Step-3: Convert the above EJB component into Web Service from this Web project.

Rt. cli. on Web project (TestWebProj) ⟶ New ⟶ Web Service ⟶ Web Serv. name: TestWs1

package: P1

⊙ Create Web Service from Existing Session Bean → Browse and select the above project related Hello1Bean (EJB Component) → OK → Finish

Step-4: Run the Web project to deploy the generated Web service component.
Rt. click on project → Run.

Step-5: Use the following URL to get the WSDL document of the project
Rt Expand project → Expand Web Services folder → Rt. cli. on TestWs → Test Web Service → get WSDL URL

http://localhost:6060/TestWebProj/TestWsService?WSDL

Procedure to develop Jsp program as Client for the above Web Service component:

Step-1: Create Web project in NetBeans IDE having name ClientProj

Step-2: Add Web service client to the project 13th Sep
Rt. cli. on project → New → Web Service Client → Project → Browse → select TestWs1 of TestWebProj → finish

Step-3: Drag and drop TestWs1 Service from Web Services reference to index.jsp page (Drag and drop SayHello(-) method).

Step-4: Run the project

Rt. click on the code of the Clt → Run

Note: We can also develop .NET client for the above Web Service component

AXIS API:

AXIS is Apache foundation supplied software which provides complete environment for programme to work with Web Services.

Axis 1.x (1.3) gives

- a built-in sample server to deploy and execu Web service
- a built-in servlet called AxisServlet ~~to~~ as Front Controller to trap the request and te pass them to Web Service Component.
- an API to develop Web Service compone and Web Service Client.

To keep Axis 1.3 software ready, download axis-1.3-all-bin.zip file to a folder

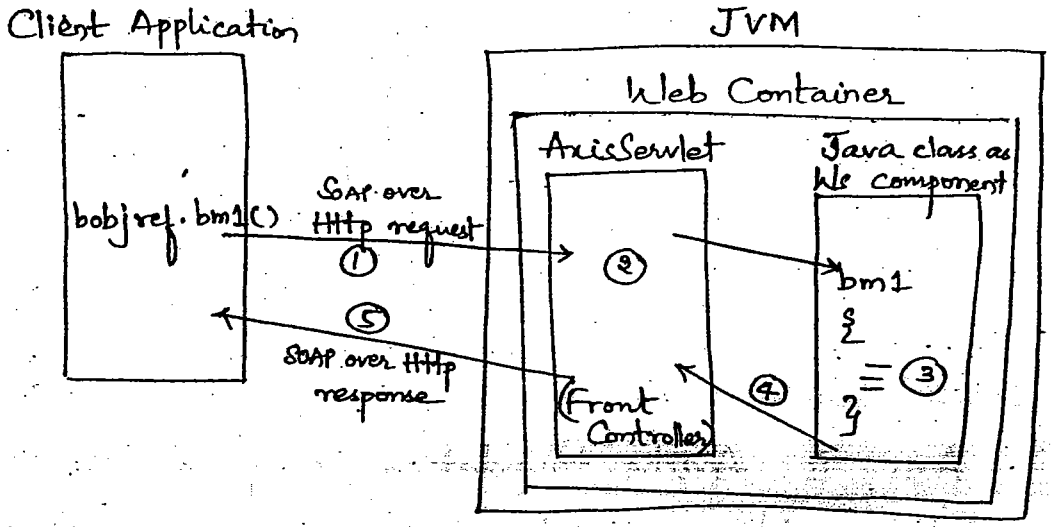
13th Sep 11

Note: A special servlet in web application which can trap and take the request coming from clients is called as Front Controller

W.r.t the the given diagram,

① Client application calls business method and generates SOAP over Http request

Axis 1.x based Scenario



② As a Front Controller, AxisServlet traps and takes the request.

③ AxisServlet calls the business method of Web service component.

④ The business method generated result comes back to AxisServlet

⑤ AxisServlet sends the result to the Client application as SOAP over Http Response

Note: (i). The above Web Container also contains a built-in UDDI registry.

(ii). The above Web Container will be started without taking the support of Web server or Application server s/w.s.

Axis:

Type: Toolkit to work with Java based web services

Version: 1.x, 2.x

Vendor: Apache foundation

Open source s/w

To download s/w : www.apache.org.

Procedure to develop Axis 1.x / 1.3 based Web Service Component:

Step-1: Install Axis 1.3 s/w

Extract axis-1.3-all-bin.zip file to a folde

Step-2: Develop Java class having business method

E:\apps
⟶ webservices
⟶ AxisApp
⟶ App1
⟶ server
⟶ Hello.java

Hello.java:

```
package satya.hello;
public class Hello
 {
   public String sayHello(String name)
    {
      return "Good Morning : " + name ;
    }
```

Step-3: Prepare .cmd or .bat file adding multip jar files of Axis 1.3 s/w to CLASSPATH.

SetEnv.bat

```
SET AXIS_LIB=E:\Apps\webservices\Axis\Apps\softwares\AXIS 1.3Soft\axis 1.3\lib
```
(<Axis 1.3_HOME>)

```
SET CLASSPATH=%.CLASSPATH%;%AXIS_LIB%\axis.jar;%AXIS_LIB%\axis-ant.jar;%AXIS_LIB%\axis-schema.jar
```

* represents the exicting values of CLASSPATH

%AXIS_LIB%\commons-discovery-0.2.jar;
%AXIS_LIB%\common-logging-1.0.4.jar;
%AXIS_LIB%\log4j-1.2.8.jar; %AXIS_LIB%\jaxrpc.jar;
%AXIS_LIB%\saaj.jar; %AXIS_LIB%\wsdl4j-1.5.1.jar

Note: In the above file, all the 9 Jar files of <AXIS 1.3_HOME>\lib folder are placed in CLASSPATH.

Step-4: Run the above batch file (SetEnv.bat)

E:\Apps\-----\App1\Server> SetEnv

Step-5: Compile the above Jav class (Hello.java)

E:\Apps\----\App1\Server> javac -d . Hello.java

Step-6: Use Java2WSDL tool to generate the WSDL document based on the above Java class.

E:\Apps\----\App1\Server> java org.apache.axis.wsdl.Java2WSDL
-o Hello.wsdl
-l http://localhost:65535/axis/servlet/AxisServlet
-p tns:Hello
-b tns:HelloBinding
-A OPERATION
-n urn:Hello satya.hello.Hello
The above developed classname

Note: To know the details about the above tool (Java2WSDL), refer <AXIS_HOME>/docs/reference.html file.

Step-7: Copy deploy.wsdd, undeploy.wsdd files from AXIS 1.3 s/w to current "Server" folder and modify them as needed.

deploy.wsdd:

```
<deployment name="test" xmlns="..." xmlns:java="...">
  <service name="urn:Hello" provider="java:RPC">
    <parameter name="className" value="satya.hello.Hello"/>
                                      classname
    <parameter name="allowedMethods" value="sayHello"/>
                                           operation name
    <parameter name="wsdlServicePort" value="tns:Hello"/>
                                      given for -P opt in step-6.
  </service>
</deployment>
```

undeploy.wsdd:

```
<undeployment name="test" xmlns:"------">
  <service name="urn:Hello"/>
</undeployment>
```

Step-8: Start the built-in Server supplied by Axis s/w

E:\---\Server> start java org.apache.axis.transport.http.SimpleAxisServer -p 65535 port no. of the Server

Step-9: Deploy the above Web Service component

E:\---\Server> java org.apache.axis.client.AdminClient -p 65535 deploy.wsdd

Procedure to develop Client Application (Axis 1.3 based) for the above Web Service Component:

Step-1: Create seperate folder for Client application and copy SetEnv.bat file to that folder

E:\Apps
  ⟶ WebServices
    ⟶ AxisApp
      ⟶ App1
        ⟶ Client
          ⟶ SetEnv.bat

Step-2: Run this SetEnv.bat file from Client folder.

E:\Apps\WebServices\AxisApp\App1\Client > SetEnv

Step-3: Develop the Client application in the above Client folder as shown below.

HelloClient.java:

```
import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import org.apache.axis.encoding.XMLType;
import javax.xml.rpc.*;
import javax.xml.rpc.*;
import javax.xml.namespace.QName;

public class HelloClient
{
  p.s.v. main(String[] args) throws Exception
  {
    // define Service (starting point)
    Service s = new Service();

    // define call from service
    Call c = (Call)s.createCall(); // 'c' holds details to
                                   call b. methods
```

14th Se

```
// set Operation (B.method) name
c.setOperationName(new QName("urn:Hello", "sayHello"
// target End point name (to locate Web service)
c.setTargetEndpointAddress("http://localhost:65535/ax
                            /servlet/AxisServlet");
// register b.method parameter with data types
c.addParameter("in0", XMLType.XSD_STRING, ParameterMo
                                          .IN);
// set return type which is returned by B.metho
c.setReturnType(XMLType.XSD_STRING);
// call the b.method (Operation)
Object o = c.invoke(new Object[]{args[0]});
// display results  S.o.pln(o.toString());
} }
```

Step-1: Compile and execute the Client application

E:\Apps\----\Client> javac HelloClient.java

E:\Apps\-----\Client> java HelloClient ramesh
                                        args[0]

Note: By using the following WSDL url, we can also develop .net client for the above Web Service component.

"http://localhost:65535/axis/servlet/AxisServlet"

14th Sep'11

Axis 2.x is totally different from Axis 1.x. Axis 2.x takes the support of a Third party supplied Web server or Application server s/ws for executing Web service Component.

Axis 1.x supports SOAP 1.1 and WSDL 1.1 specifications where as Axis 2.x supports both SOAP 1.1, SOAP 1.2 and WSDL 1.1, WSDL 2.0 specifications.

Axis 1.x gives built-in server to execute Web Service component where as Axis 2.x depends up on Third Party Web server or Application Server s/w s.

Steps to develop Axis 2.x based Web Service Component:

Step-1: Download the following zip files from www.apache.org website.

(i) axis2-1.5.zip file (to work with Axis API)
— (basic AXIS s/w)

(ii) axis2-1.5.5-war.zip file
— gives a special WAR file as plugin to make Tomcat server recognize and work with Web Services.

Step-2: Extract both zip files to 2 different folders.

Step-3: Gather axis2.war file (special plugin type web application) to make Tomcat Server working with Web service (from axis2-1.5.5-war.zip file extraction and deploy that file in <Tomcat_Home>\webapps folder).

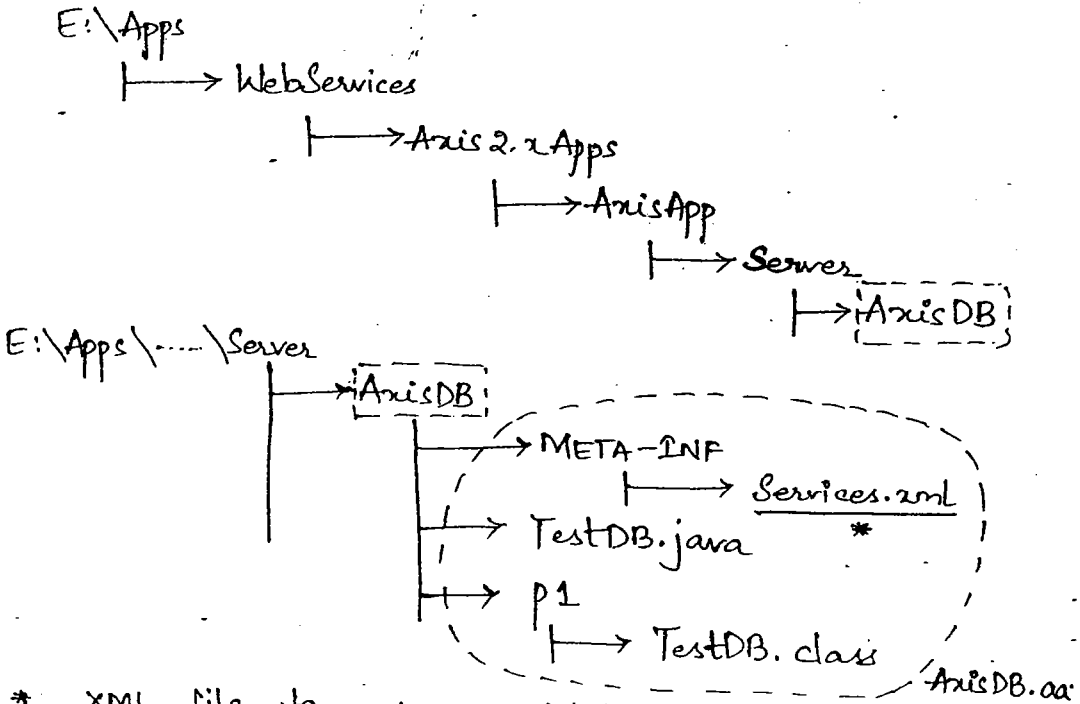
Step-4: Start Tomcat Server

Step-5: Launch Axis 2 Web application related home page by using the following URL.

http://localhost:2020/axis2

In the above url generated home page, click on Validate hyperlink and get access to Axis 2 Happiness Page in order to confirm that Axis2.war file has been deployed successfully.

Step-6: Develop Axis 2 Web Service component as shown below.



* XML file to configure Web Service component and its operation names (b. method names)

TestDB.java:

```
package p1;
import java.sql.*;

public class TestDB
{
  Connection con;
  PreparedStatement ps;
  ResultSet rs;
  public TestDB()
  {
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con= DriverManager.getConnection("jdbc:oracle:
            thin:@localhost:1521:orcl","scott","tiger");
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }

  //B.method1 (Operation 1)
  public int insert(int no, String name, String addrs)
  {                                    throws Exception
    ps= con.prepareStatement("insert into Axis_Student
                    values(?,?,?)");
    ps.setInt(1,no);
    ps.setString(2,name);
    ps.setString(3,addrs);
```

```
        return ps.executeUpdate();
    }//insert

    // B. method 2 (Operation 2)
    public int update(int no, String name, String addrs
                                        throws Exception
    {
        ps = con.prepareStatement("update Axis_Student
                set sname=?, sadd=? where sno=?");
        ps.setString(1, name);
        ps.setString(1, addrs);
        ps.setInt(3, no);
        return ps.executeUpdate();
    }//update
}//class.
```

Now compile the above TestDB.java using javac -d option.

```
E:\Apps\....\AxisDB> javac -d . TestDB.java
```

DB Table:

```
SQL> create table Axis_Student (sno number, sname varchar2
     (20), sadd varchar2(20));
Table created.
```

```
E:\Apps\....\AxisDB > javac -d . TestDB.java
```

Note: For this compilation, no special jar files are required in the CLASSPATH.

Services.xml: (DD file / Configuration file)

```
<service>
  <parameter name="ServiceClass">p1.TestDB</parameter>
  <operation name="insert">
    <messageReceiver class="org.apache.axis2.rpc.
        receivers.RPCMessageReceiver"/>
  </operation>
  <operation name="update">
    <messageReceiver class="org.apache.axis2.rpc.
        receivers.RPCMessageReceiver"/>
  </operation>
</service>
```

(TestDB: the above developed Java class; update: business method)

Step-7: Prepare .aar file (Axis Archive file) on the directory structure that represents Web Service component (AxisDB).

```
E:\Apps\-----\AxisDB> jar cf AxisDB.aar .
```

Step-8: Place the above generated AxisDB.aar file in <Tomcat_HOME>\Webapps\Axis2\WEB-INF\services folder.

Note: In the above step (Step-8) the AxisDB.aar file related TestDB class will be converted into WebService component.

Step-9: Place the Oracle Thin driver related ojdbc14.jar file in <Tomcat_HOME>\Webapps\Axis2\WEB-INF\lib folder.

Step-10: Generate/Gather the WSDL url for the above Web Service component.

Type this URL in browser window

http://localhost:2020/axis2 ⟶ Services

⟶ select AxisDB

Note: The above process gives the following WSDL url.

http://localhost:2020/axis2/services/AxisDB?wsdl

Procedure to develop Axis 2.x based Client application for the above Web Service component:

Step-1: Create the following environment variables. (→ MyComputer)

PATH = <java_home>\bin; <Axis2_Home>\bin; <other>;.

(<java_home>\bin: Jdk s/w installation folder; <Axis2_Home>\bin: Axis2 s/w installation folder)

AXIS2_HOME = <Axis2_home>

Axis2 s/w installation folder like E:\Axis2.2Soft\axis2-1.5.5

The directory that comes after extracting axis2-1.5.5.zip file.

JAVA_HOME = <java_home>

the Jdk s/w installation folder like D:\Java\jdk1.6.0_11

Note: For AXIS2_HOME, JAVA_HOME, we must not place ; symbols at the end of their values.

Step-2: Use wsdl2java tool from AxisApp folder to generate the helper classes that are required in Web Service Client application development.

E:\Apps\.....\AxisApp > wsdl2java -uri http://localhost:2020/axis2/services/AxisDB?wsdl
-o client

WSDL url

The folder where generated helper resources will be saved.

Note: The above step generates Client folder under AxisApp folder having files and folders representing helper resources using which we can develop Client application.

Sep'11

The above step generates helper classes (.java) in the following folder.

E:\Apps\Webservices\....\AxisApp\Client\src\p1 folder

Step-3: Develop Client application and place that application in the above generated Client\src folder.

E:\Apps\.....\AxisApp\Client\src\TestClient.java

TestClient.java:

```
import p1.*;
public class TestClient
{
  p.s.v. main(String[] args) throws Exception
  {
    AxisDBStub stud = new AxisDBStub();
  // locate B.method1 (insert)
  AxisDBStub.Insert request1 = new AxisDBStub.
                                    Insert();
```

```
// set arg. values to B.method 1 (insert)
request1.setArgs0(2);
request1.setArgs1("satya");
request1.setArgs2("hyd");

//invoke the B.method 1 (insert)
AxisDBStub.InsertResponse response1 = stub.insert(request1);
S.o.ph("Insert result is : "+response1.get_return());

//locate B.method 2 (update)
AxisDBStub.Update request2 = new AxisDBStub.Update();

//set arg. values to B.method 2 (update)
request2.setArgs0(2);
request2.setArgs1("raja");
request2.setArgs2("vizag");

//invoke B. method 2 (update)
AxisDBStub.Update response2 = stub.update(request2);
S.o.ph("Update result is : "+response2.get_return());
}

}
```

Note: The above code has utilized both of Inner classes generated in p1 folder using "wsdl2java" tool.

Step-4: Prepare the batch file in "p1" folder as shown below to add the Axis2 supplied (collected from axis2-1.5.zip file extraction) 14 jar files to the classpath.

setCpath.bat:

```
SET AXIS2_LIB = E:\Apps\WebServices\Axis2.xApps\Axis
                2.xSoft\axis2-1.5.5\lib

SET CLASSPATH = %CLASSPATH% ; %AXIS2_LIB%\axis2-
                kernel-1.5.5.jar ; %AXIS2_LIB%\axis2-
                adb-1.5.5.jar ; %AXIS2_LIB%\axiom-api-1.2.11.
                jar ; %AXIS2_LIB%\wsdl4j-1.6.2.jar ;
                %AXIS2_LIB%\XmlSchema-1.4.3.jar ;
                %AXIS2_LIB%\commons-logging-1.1.1.jar ;
                %AXIS2_LIB%\axiom-impl-1.2.11.jar ;
                %AXIS2_LIB%\neethi-2.0.5.jar ; %
                AXIS2_LIB%\axis2-transport-http-1.5.5.
                jar ; %AXIS_LIB%\axis2-transport-lo
                cal-1.5.5.jar ; %AXIS2_LIB%\commons-
                httpclient-3.1.jar ; %AXIS2_LIB%\mail-
                1.4.jar ; %AXIS2_LIB%\httpcore-4.0.
                jar ; %AXIS2_LIB%\commons-codec-1.3.jar
```

Step-5: Run the above batch file (i.e, the above setCpath.bat file).

E:\Apps\---- \AxisApp\Client\src\p1 > setCpath

Step-6: Compile all the resources of p1 folder.

E:\Apps\.....\AxisApp\Client\src\p1 > javac *.java

Step-7: Compile and execute the Client application from Client\src folder.

E:\Apps\...\AxisApp\Client\src > javac -d . *.java

E:\Apps\...\AxisApp\Client\src > java TestClient

Note: Make sure that the Tomcat Web Server is in running mode while executing the Client application.

Note: We can also develop .net based Web Service Client for the above developed Axis 2.0 Web Service Component.

List of Jar files collected from the extraction of axis2-1.5.zip file (14 jar files):

1. axis2-kernel-1.5.5.jar
2. axis2-adb-1.5.5.jar
3. axiom-api-1.2.11.jar
4. axiom-impl-1.2.11.jar
5. wsdl4j-1.6.2.jar
6. XmlSchema-1.4.3.jar
7. commons-logging-1.1.1.jar
8. commons-httpclient-3.1.jar
9. axis2-transport-http-1.5.5.jar
10. axis2-transport-local-1.5.5.jar
11. neethi-2.0.5.jar
12. mail-1.4.jar
13. httpcore-4.0.jar
14. commons-codec-1.3.jar

16th Sep '2011

In Bottom-up approach, first Web Service component will be developed by using certain technology like Java, .net, etc. then the WSDL document will be generated.

The web service components that we have developed so far comes under Bottom-up approach based Web Service components.

In Top-down approach, pure WSDL document will be prepared, then the Web Service components will be developed based on that component in different technologies.

In Top-down approach, programmers can develop different flavours of Web Service components for single WSDL document.

We can say that Top-down approach based Web Service components are more inter-operatable in Large scale environment when compared to Bottom-up approach based Web Service components.

Both MyEclipse, NetBeans IDEs give support for developing Bottom-up and Top-down approaches based Web Service components.

MyEclipse:

Type : IDE s/w for Java environment
Version: 8.2 (compatable with jdk 1.6)
Vendor: Eclipse org.
Commercial IDE

Gives Tomcat as a built-in server and also allows to configure any external server.

To download trail version: www.myeclipseide.com

Cheat code

Subscriber name: gocto.cn

Subscription code: tLR8ZC-855444-666653573987614 2

Procedure to develop JAX-WS based Web Service Component based on Bottom-up approach by using MyEclipse 8.x:

Step-1: Launch myEclipse 8.x IDE by choosing Work space folder. * The folder where my projects will be saved.

Step-2: Submit Cheat code

MyEclipse menu → Subscribe information →

Subscriber : [gocto.cn]

Subscription code : [tLR8ZC-855444-666653573987614 2]

Step-3: Create Web Service project.

file menu → new → Web Service project →

Project name : [SathyaWsProj] → Framework : ⊙ JAX-WS

→ finish

Step-4: Add Java class to the project having business methods

Rt. cli. on proj. → New → Class →

Name : [Calc1] → finish

Calc1.java

```
public class Calc1
 {
   // B. method
   public int sum(int x, int y)
    {
      return x+y;
    }
 }
```

Step-5: Add JAX-WS libraries to the project.

Rt. clic. on proj. → Build path → Add Libraries... → MyEclipse Libraries → Next → select ☑ JAX-WS 2.1 Runtime Libraries and ☑ JAX-WS 2.1 API Libraries → finish

Step-6: Develop Web Service component based on the above Java class.

Rt. clic. on proj. → New → other → MyEclipse → Web Services → Web service → Next → Framework: ⊙ JAX-WS → Strategy: ⊙ Create Web Service from Java class (Bottom-up Scenario) → Next → Java class: [Calc1] → Finish.

*** Step-10: Test the Web Service component

Launch SOAP Web Service Explorer from the tool bar (icon) → Launch WSDL Page → Select WSDL main → Type the WSDL url → http://localhost:2020/SathyaWsProj/Calc1Port?WSDL

Step-7: Configure Tomcat Server with MyEclipse IDE

Window menu → preferences → MyEclipse → Servers → Tomcat → Tomcat 6.x → ⊙ Enable → Tomcat home directory: browse and select Tomcat installation folder (D:\Tomcat 6) → Apply → OK.

Step-8: Start Tomcat Server

Go to the server icon in the tool bar (▶) → Tomcat 6 → Start

Step-9: Deploy the project in Tomcat server →

Go to the deploy icon (📁) in the toob bar → Project: SathyaWSProj → Add Server: Tomcat 6 → finish

Step-10: ***

Procedure to develop Web Service client for the above Web Service component by using MyEclipse IDE:

Step-1: Create Java project.

file menu → new → Java project → Project name: ClientProj1 → Next → finish

Step-2: Add Web service client to the project

Rt. cl. on proj. (ClientProj1) → New → Others → MyEclipse → Web Services → Web Service Client → Next → Framework: ⊙ JAX-WS → Next → WSDL URL: http://localhost:2020/.../Calc1Port?WSDL → Next → Next → finish

* collect from port name attribute of @WebService generated in Calc1Delegate

Note: The above step-2 generates some helper classes required for the Client application development.

Step-3: Add Java class as Web Service client calling business methods.

Rt. cl. on proj (ClientProj1) → New → Class → Name: TestClient1 → select ☑ p.s.v.m() → finish → Write the following code in the main().

TestClient1.java:

```
public class TestClient1;
{
  p.s.v.main(String[] args)
  {
    Calc1Service service = new Calc1Service();
    Calc1Delegate delegate = service.getCalc1Port();
    S.o.pln("Result is :" + delegate.sum(10,20));
  }
}
```

Step-4: Run the Client application

Rt. cli. on the source code of TestClient1.java → Run as → Java application.

Procedure to use MyEclipse 8.x IDE to develop JAX-WS based and Top-down based Web Service component:

Step-1: Create Web Service project in MyEclipse IDE. having name SathyaWsProj1.

Step-2: Create WSDL document in the project

Rt. cli. on proj. → New → Other → MyEclipse → Web Services → WSDL → Next → file name: Test1.wsdl → Next → finish →

N Replace New operation word with "add" word in the Test1.wsdl file → add parameters (specify) names of the business method (B. method name) (make sure that the following two lines are there at line no.s 8 and 9)

```
<xsd:element name="x" type="xsd:int"/>
<xsd:element name="y" type="xsd:int"/>
```

→ Make sure that the following line is there at line no. 16 specifying return type of the business method.

```
<xsd:element name="z" type="xsd:int"/>
```

Step-3: Add the Web Service component to project based on the above WSDL document.

Rt. cli. on proj. → New → Other → MyEclipse → Web Services → Web Service → Next →

Framework: ⊙ JAX-WS → Strategy: ⊙ Create Web Service from WSDL document (Top-down Scenario) → Next → WSDL file: [ ] Browse

and select the above Test1.wsdl file → Next → Finish

Note: The above step generates lots of helper classes based on WSDL document.

Step-4: Add business logic for business method in the generated Test1SOAPImpl.java file.

Step-5: Add JAX-WS Libraries to the project.

Step-6: Configure Tomcat and Start the server.

Step-7: Deploy proj. in Tomcat server

Step-8: Launch SOAP Web Service explorer and test the Web Service component by using the following url.

http://localhost:2020/SathyaWsProj1/Test1SOAP?wsdl

Note: for this Web Service component, we can develop Web Service client in a regular manner as shown in previous example.

Note: In NetBeans IDE, there is no provision to create New WSDL document but it can be used to develop Web Service component based on the existing ~~Web~~ & WSDL file.

Procedure: (Top-down approach).

Step-1: Create Web project in NetBeans IDE

Step-2: Add Web service component to project based on WSDL file

Rt. cli. on proj. → New → Other →

Web Services → Web service from WSDL → Next → * Package : [P1] → Browse and select WSDL file (select the above WSDL file i.e., Test1.wsdl) → finish →

* Web Service Name : [MyWs] →

Add business logic in the add(,) business method of MyWs.java file

Step-3: Run the project.

Step-4: Test the project

17th Sept'11

The Registry s/ws that are specific to one organization are called as Private UDDI Registries.

The UDDI registry that is common for all is called as public registry

x-point.com, uddi.org, service-repository.com are the examples for public UDDI registries.

The UDDI registries of every web server\application s/w is called as Private UDDI registry.

Procedure to consume Web service component that is available in www.service-repository.com Public UDDI Registry:

Step-1: Launch www.service-repository.com website → select Calculator service → Gather the WSDL url

http://soaptest.parasoft.com/calculator.wsdl.

Step-2: We can write Web service Client application based on the above WSDL url in regular manner.

Note: We can also use this repository to(www.service-repository.com) to publish our Web Services and to make them as Globaly visible Web service Components.

REST (Representational State Transfer) Mechanism:

REST is a new mechanism to develop Web Services by directly using XML in HTTP protocol environment (no need of working with SOAP). The SOAP request, response messages are always complex and heavy to embed with HttpRequest, HttpResponse messages. SOAP REST mechanism avoids this SOAP and directly embeds simple XML Tags in HttpRequest, Response messages.

For related information on SOAP versus REST, refer page no.s 1-4 of the Supplementary handout given on 17th Sep 2011.

* JAX-RS is also called as Jersy

Sun Microsystems gives JAX-RS* with Annotations support as built-in APIs of Jdk 1.6 to develop RESTful Web Services and to consume them by writing REST-ful Web Service clients.

In REST-ful Web Service development, a Java class will be taken having multiple business

methods and these business methods will be annotated with @GET, @PUT, @DELETE, @POST to make them responding for HTTP Request methods (GET, PUT, DELETE, POST) based requests given by Clients.

For basics of REST programming, refer page no.s 5 and 6 of Supplementary hand out given on 17th Sep 2011.

Procedure to develop REST full Web Service Component by using NetBeans 7.0.1 IDE:

Step-1: Create Web Project in NetBeans IDE having Name SathyaRSProj

File menu → New project → Java Web → Web application → Name: SathyaRsProj → Next → Next → Finish

Step-2: Add Web Service component to the project.

Rt. cli. of proj (SathyaRSProj) → New → Other → Web Services → RESTful WebService from Patterns → Next → ⊙ Simple Root Resource → Next → package name: P1 Path: myrs1 Class Name: TestWs1 → Finish → Rest Resources path: /resources → OK

Step-3: Keep the following code in TestWs1.java file

TestWs1.java:

```
package p1;
   // import statements

import javax.ws.rs.core.*;
import javax.ws.rs.*;

@Path("myrs1/{num1}/{num2}") // to Identify the RESTful Web Service

public class TestWs1
{
  @GET  //here sum B. method responds for HTTP GET request
  @Produces("text/html")
  public String sum(@PathParam("num1") int n1,
                    @PathParam("num2") int n2)
    {
      return "-"+(n1+n2);
    }
}
```

Step-4: Run the project

Rt. cli on proj (SathyaRsProj) ⟶ Run

Step-5: Test the RESTful Web Service component.

Expand SathyaRsProj ⟶ Rt. cli on RESTful Web Services ⟶ Test-RESTful Web Services ⟶ OK ⟶ ....

(or)

Step-5: Type this URL in the browser window.

http://localhost:6060/SathyaRsProj/resources/myrs1/10/20

resources * refer step-2; myrs1 * refer step-2; 10, 20 ↓ argument values of B. method

Procedure to develop REST-ful Web Service Client Application:

Step-1: Create Java project having name ClientRsPrj

Step-2: Add REST-ful Java Client to the project to generate helper code.

Rt. cli. on proj. → New → Other → Web Services → RESTful Java Client → Next → Class Name: MyHelper → package: p1 → Select REST source → Browse → SathyaRsProj → TestWs1 → OK → Finish.

Step-3: Develop Web Service Client application in the project.

Rt. cli. on proj (ClientRsProj) → New → Java class → Class Name: TestClient → finish → Write the following code in the main()

TestClient.java

```
import p1.MyHelper;
public class TestClient
 {
   p.s.v.main(String[] args)
    {
      MyHelper client = new MyHelper("10","20");
      Object response = client.sum();
      S.o.pln("Result is :" +response);
    }
 }
```

B. method arguments

Step-4: Run the Client application

Rt. clic. in the source code of TestClient.java

⟶ Run file