

# Introduction to Web Services

Bina Ramamurthy

9/17/2004

1

## Literature Surveyed

- ◆ IBM's alphaworks site:  
<http://www-106.ibm.com/developerworks/webservices/>  
<http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>
- ◆ O'Reilly book on Web Services: Kim Topley's Webservices in a Nutshell: <http://www.oreilly.com/catalog/javawsian/index.html>  
This link has a sample chapter (SAAJ) and zip of all examples in the book.
- ◆ <http://www.w3.org/DesignIssues/WebServices.html>

9/17/2004

2

## Web Services

- ◆ *Web Services* is a technology that allows for applications to communicate with each other in a standard format.
- ◆ A *Web Service* exposes an interface that can be accessed through XML messaging.
- ◆ A Web service uses XML based protocol to describe an operation or the data exchange with another web service. Ex: SOAP
- ◆ A group of web services collaborating accomplish the tasks of an application. The architecture of such an application is called Service-Oriented Architecture (SOA).

9/17/2004

3

## Web Services Suite of Protocols

- ◆ A suite of protocols define the Web Services Technology.
- ◆ These are used to describe, publish, discover, deliver and interact with services.
- ◆ The information about the protocols is from IBM's [developerworks](#).

9/17/2004

4

## WS Suite of Protocols

- ◆ Messaging protocol Simple Object Access Protocol (SOAP) encodes messages so that they can be delivered over the transport protocols HTTP, SMTP or IIOP.
- ◆ Web Services Definition Language (WSDL) is used to specify the service details such as name, methods and their parameters, and the location of the service. This facilitates the registering and discovery of the service.
- ◆ For services to locate each other, the Universal Description, Discovery and Integration (UDDI) protocol defines a registry and associated protocols for locating and accessing services.

9/17/2004

5

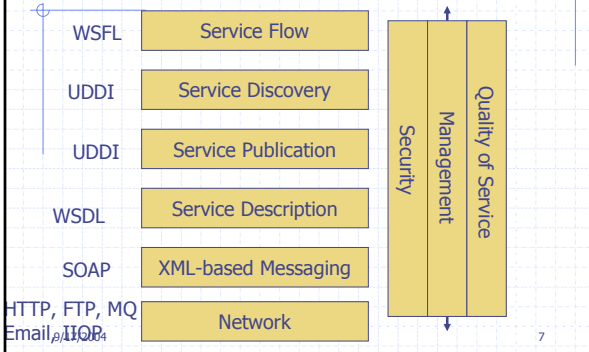
## WS Suite of Protocols (contd.)

- ◆ The WS-Transaction and WS-Coordination protocols work together to handle distributed transactions.
- ◆ The Business Process Execution Language for Web Services (BPEL4WS) defines workflow operations.
- ◆ WS-Security is a family of protocols that cover authentication, authorization, federated security, secure communications, and delivery.
- ◆ WS-Policy is another group of protocols that define the policy rules behind how and when Web services can interact with each other.
- ◆ WS-Trust defines how trust models work between different services.
- ◆ These protocols are for e-business. Are there any available for e-science?

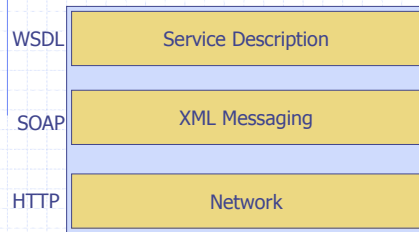
9/17/2004

6

## WS Stack



## WS Interoperability Infrastructure



Do you see any platform or language dependencies here?

9/17/2004

8

## JAX-RPC

- ◆ **JAX-RPC: Java API for XML-based Remote Procedure Call (RPC).**
- ◆ **An API for building Web Services and Web Services clients.**
- ◆ **Some important concepts in JAX-RPC are:**
  - **Type-mapping system (WSDL to Java)**
  - **Service endpoint**
  - **Exception handling**
  - **Service endpoint context**
  - **Message handlers**
  - **Service clients and service context**
  - **SOAP with attachments**
  - **Runtime services**
  - **JAX-RPC client invocation models**

9/17/2004

9

## JAX-RPC (contd.)

- ◆ JAX-RPC is designed to provide a simple way for developers to create Web services server and Web services client.
- ◆ Based on remote procedure calls; so the programming model is familiar to Java developers who have used RMI or CORBA.
- ◆ Major difference between RMI and JAX-RPC is that messages exchanged are encoded in XML based protocol and can be carried over a variety of transport protocols such as HTTP, SMTP etc.
- ◆ You can use JAX-RPC without having to be an expert in XML, SOAP, or HTTP.

9/17/2004

10

## The JAX-RPC Programming Model

- ◆ Services, ports and bindings
- ◆ JAX-RPC web service servers and clients
- ◆ JAX-RPC service creation
- ◆ JAX-RPC client and server programming environments
- ◆ Stubs and ties
- ◆ Client invocation modes
- ◆ Static and dynamic stubs and invocation

9/17/2004

11

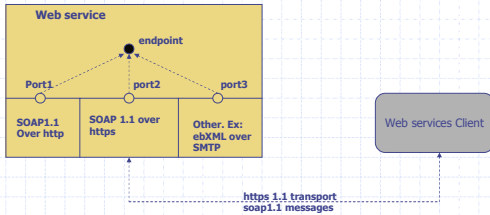
## Services, ports and bindings

- ◆ Service endpoint interface or service **endpoint** that defines one or more operations that the web service offers.
- ◆ Access to an endpoint is provided by binding it to a protocol stack through a **port**.
  - **A port has an address that the client can use to communicate with the service and invoke its operations.**
- ◆ An endpoint can be bound to different ports each offering a different suite of protocols for interaction.

9/17/2004

12

## Endpoint, Port and binding



9/17/2004

13

## Web Service Clients and Servers

- ◆ JAX-RPC maps a
  - web service operation to a java method call.
  - service endpoint to a Java Interface.
- ◆ Thus one way to begin implementation of a web service in JAX-RPC is to define a Java interface with a method for each operation of the service along with a class that implements the interface. Of course, following the rules of remote invocation etc.
- ◆ Now visualize client/server invocation in the same address space and lets compare it with remote invocation.

9/17/2004

14

## Local Date Service

```
//server
public class DateService {
    public Date getDate() {
        return new Date();}
}

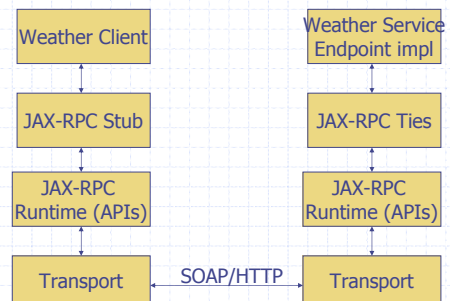
//client
Public class Appln {
    public static void main (..) {
        DateService instance = new DateService();
        Date date = instance.getDate();
        System.out.println (" The date is" + date);
    }
}
```

- ◆ In the case of the remote call a layer of software is used to convey the method call from client to server. This layer of software is provided by JAX-RPC runtime.

9/17/2004

15

## Application Architecture



9/17/2004

16

## JAX-RPC service creation

- ◆ A service definition describes the operations that it provides and the data types that they require as argument and provide as return values.
- ◆ This definition can be made available as a document written in WSDL.
- ◆ From a WSDL document, JAX-RPC can generate the Java code required to connect a client to a server leaving one to write only the logic of the client application itself.
- ◆ Since WSDL is language independent the server can be in .net, Jax-rpc or any other compatible platform.

9/17/2004

17

## JAX-RPC service creation (contd.)

- ◆ Define the service a Java interface.
- ◆ Generate WSDL using the tools provided with JAX-RPC package.
- ◆ Advertise it in a registry for the client to lookup and import it.
- ◆ For publication and lookup any other technology such as J2EE can be used.

9/17/2004

18

## Interface Method Definition

- ◆ Interface must extend `java.rmi.Remote`
- ◆ Must declare that it throws `java.rmi.RemoteException`
- ◆ May declare other service-dependent exceptions such as `java.lang.Exception`
- ◆ May be derived as extensions from other interfaces.

9/17/2004

19

## Client and Server Programming Environment

- ◆ JAX-RPC API is distributed over a set of packages:
  - `javax.xml.rpc`
  - `javax.xml.rpc.encoding`
  - `javax.xml.rpc.handler`
  - `javax.xml.rpc.handler.soap`
  - `javax.xml.rpc.holders`
  - `javax.xml.rpc.server`
  - `javax.xml.rpc.soap`

9/17/2004

20

## Stubs and Ties

- ◆ Client Side: Stub object has the same methods as the service implementation class.
  - Client application is linked with the **stub**.
  - When it invokes a method stub delegates the call to the JAX-RPC runtime so that appropriate SOAP message can be sent to the server.
  - On completion the result return back in the reverse path as above.
- ◆ Server side:
  - Message received must be converted into a method call on actual service implementation. This functionality is provided by another piece of glue called **tie**.
  - Tie extracts method name and parameter from SOAP message.
  - Tie also converts the result of the method call back into a response message to be returned to client JAX-RPC runtime.
- ◆ Developer need not write these classes (tie and stub) since JAX-RPC comes with tools to generate them.

9/17/2004

21

## Client Invocation Modes

- ◆ Synchronous request-response mode (tightly coupled).
- ◆ One-way RPC (loosely coupled): no value returned, no exception thrown, need to bypass stub layer, use Dynamic Invocation Interface (DII).

9/17/2004

22

## Approaches to Web Service Implementation

- ◆ Top down: Start with WSDL and map onto Java
- ◆ Bottom up: Start with Java and end up all the supporting classes needed. Tools are available for converting Java to WSDL.

9/17/2004

23

## WS Development Lifecycle

- ◆ Build:
  - Definition of service interface
  - Definition of service implementation
    - New services
    - Existing application into WS
    - Composing a WS out of other WS and applications
  - Source compiled and Stubs and Ties are generated.
- ◆ Deploy:
  - Publication of the service interface and service implementation to service registry or service requestor.
  - Deployment of executables in an execution environment.

9/17/2004

24

## WS Development Lifecycle (contd.)

- ◆ Run: A WS is available for invocation. Requestor can perform find and bind operation.
- ◆ Manage: on going management and administration for security, availability, performance, QoS, and business processes.

9/17/2004

25

## A Simple Example from Sun Microsystems

- ◆ HelloWorld distributed application:
- ◆ Files of interest:
  - **HelloIF.java**: service definition interface
  - **HelloImpl.java**: Service definition implementation.
  - **HelloClient.java**: remote client to invoke the service.
  - **config-interface.xml**: configuration file used by **wscm**
  - **jaxrpc-ri.xml**: a configuration file used by **wsdeploy**
  - **web.xml**: a deployment descriptor for the web component that dispatches to the service.
  - **build.xml** for running the various steps such as compile, **wscm**, **deploy** etc. Used by the ant tool.
  - **build.properties**: contains the details of various context roots or paths.

9/17/2004

26

## Building and Deploying the Service

- ◆ Code the service definition interface and implementation class.
- ◆ Compile the service definition code written above.
- ◆ Package the code in a WAR (web archive) file.
- ◆ Generate the ties and WSDL files.
- ◆ Deploy the service.

9/17/2004

27

## Coding the interface and implementation classes

- ◆ The interface extends `java.rmi.Remote` interface.
- ◆ No constant declarations allowed.
- ◆ Methods must throw `java.rmi.RemoteException`
- ◆ Method parameters and return types must be supported by JAX-RPC types.

9/17/2004

28

## Compiling and Packaging

- ◆ To compile:
  - `ant compile-server`
- ◆ To package:
  - `ant setup-web-inf`
  - `ant package`
- ◆ These two commands will generate and place the executables in appropriate directories. (Details will be given to you later in another handout).

9/17/2004

29

## Generating Ties and WSDL file and deploy the service

- ◆ To generate Ties and WSDL:
  - `ant process-war`
  - Will invoke `wsdeploy` to generate the tie classes and the WSDL file `MyHello.wsdl`
- ◆ To deploy the service:
  - `ant deploy`
- ◆ To verify deployment:
  - <http://localhost:8080/hello-jaxrpc/hello>
  - The details of the web service will be displayed.
- ◆ To undeploy:
  - `ant undeploy`

9/17/2004

30

## Building and Running the client

- ◆ Generate the stubs.
- ◆ Code the client.
- ◆ Compile the client code.
- ◆ Package the client classes into a JAR file.
- ◆ Run the client.

9/17/2004

31

## Client steps

- ◆ To generate stubs:
  - ant generate-stubs
  - This will call *wscmpile* to generate the stubs.
- ◆ Coding the client: is a stand alone program. It calls the service through the generated stub which acts as a proxy for the remote service.

9/17/2004

32

## Clients steps (contd.)

- ◆ Compile the client code:
  - ant compile-client
- ◆ Package the client:
  - ant jar-client
- ◆ Running the client:
  - ant run

9/17/2004

33

## Iterative Development

- ◆ Test the application.
- ◆ Edit the source files.
- ◆ Execute *ant build* to create and deploy war files.
- ◆ Execute *ant redeploy* to undeploy and deploy the service.
- ◆ Execute *ant build-static* to create jar files with static stubs.
- ◆ Execute *ant run* to run the client.

9/17/2004

34

## Other features

- ◆ Read about the data types supported by JAX-RPC
- ◆ An advanced feature of interest is the dynamic proxy.
- ◆ Read about the directory structure and paths.

9/17/2004

35

## Summary

- ◆ That was an overview of Webservices realized using JAX-RPC.
- ◆ Best way to reinforce the concepts we discussed is by developing a WS-based application and testing it.
- ◆ Next class we will look at hands-on WS-based service implementation.

9/17/2004

36