

Java Web Services

Java Web Services with Axis and Soap

Instructor: Rick Palmer, SCWCD

rick@online-ettraining.com

Topics Covered

- ❑ Web Services Overview
- ❑ SOAP
- ❑ Axis

Application Evolution

❑ Monolithic Applications

⇒ Single, large computer program that couples User Interface, Business, and Database logic together tightly.

❑ Client Server Applications

⇒ Stand-alone client applications that connected to a central database server, with some logic embedded in the database.

❑ *n*-tier Applications

⇒ Modular components, each handling separate concerns (e.g. User Interface Logic, Business Logic, and Database Logic).

⇒ Designed to spread user requests across multiple servers, to pool resources, and to support a much larger number of users.

Development Evolution

- ❑ Procedural programming
 - ⇒ FORTRAN, COBOL
- ❑ Object-oriented programming
 - ⇒ Encapsulation of data and functionality
 - ⇒ Re-usable and extensible code
- ❑ Component-based development
 - ⇒ Standardized, “pluggable” components (e.g. spreadsheet control, or calendar control)
 - ⇒ Wrappers around groups of objects that provide specific functionality.
- ❑ System-based development
 - ⇒ Computers around the world communicating with each other using standardized protocols and formats.

Web Services – the current wave

- The Vision: a Web as rich in *functionality* as it currently is with *information*.
 - ⇒ Components and services shared between organizations, not just applications
- The Need: an integration mechanism
 - ⇒ A common, Web-based protocol
 - ⇒ A standard format for requesting a service
 - ⇒ A standard format for describing data
 - ⇒ Must be language-, platform-, and vendor-neutral to support full interoperability.

Web Services Building Blocks

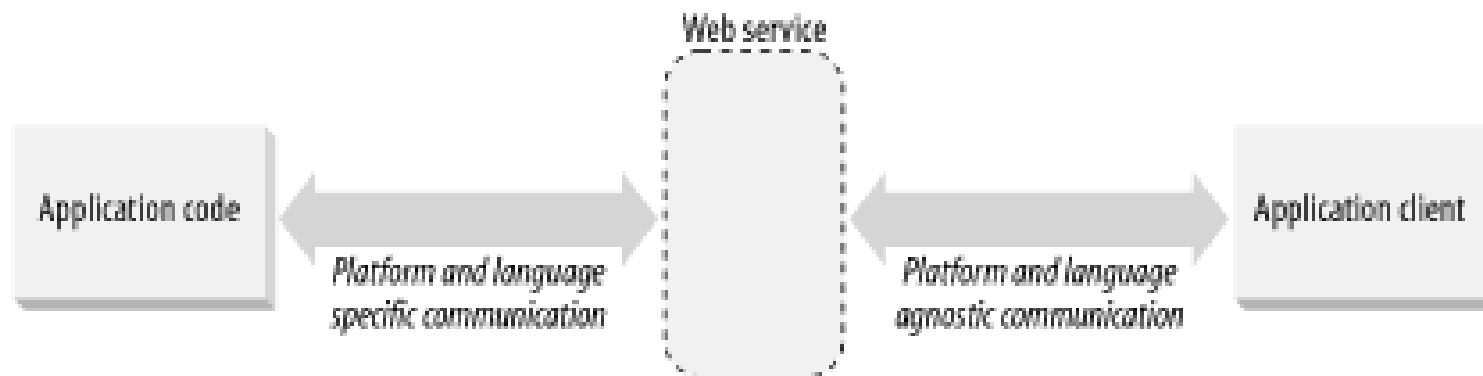
- ❑ HTTP: Standard web transportation protocol
- ❑ SOAP (Simple Object Access Protocol)
 - ⇒ Standard XML-based communication protocol
- ❑ WSDL (Web Services Description Language)
 - ⇒ Describes a web service and its public methods.
 - ⇒ Can be used to generate client proxy classes.
- ❑ Each web service has its own URL *Endpoint*
 - ⇒ Receives SOAP XML request messages
 - ⇒ Parses the XML and calls the requested object method
 - ⇒ Returns the response data encoded in SOAP XML

XML Messaging with SOAP

- Applications request services from each other with SOAP XML messages.



- SOAP XML web services insulate other applications from platform/language-specific communication.



SOAP Introduction

- Typical RPC (Remote procedure call):

```
String strYear = objVehicle.getYear("779RIQ");
```

- XML-RPC

- ⇒ **Request:**

```
<getYear>779RIQ</getYear>
```

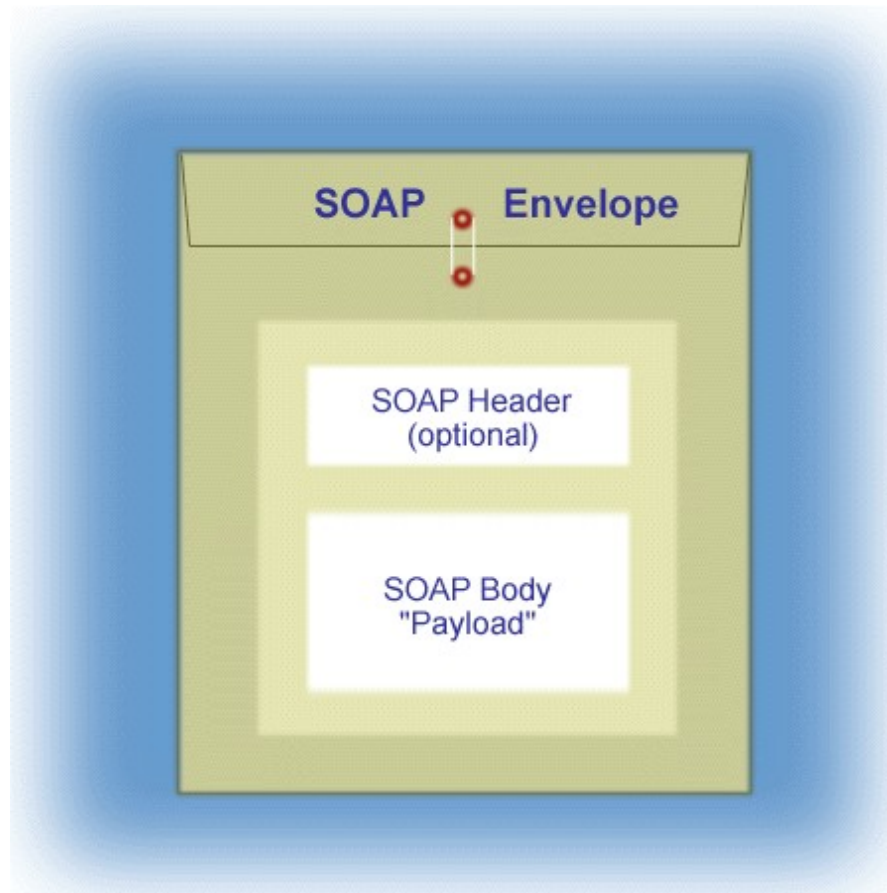
- ⇒ **Response:**

```
<getYearResponse>1996</getYearResponse>
```

- SOAP simply provides an industry-recognized standard for XML-RPC.
 - ⇒ Provides a consistent communication mechanism.
 - ⇒ Allows vendors to provide tools built around SOAP.

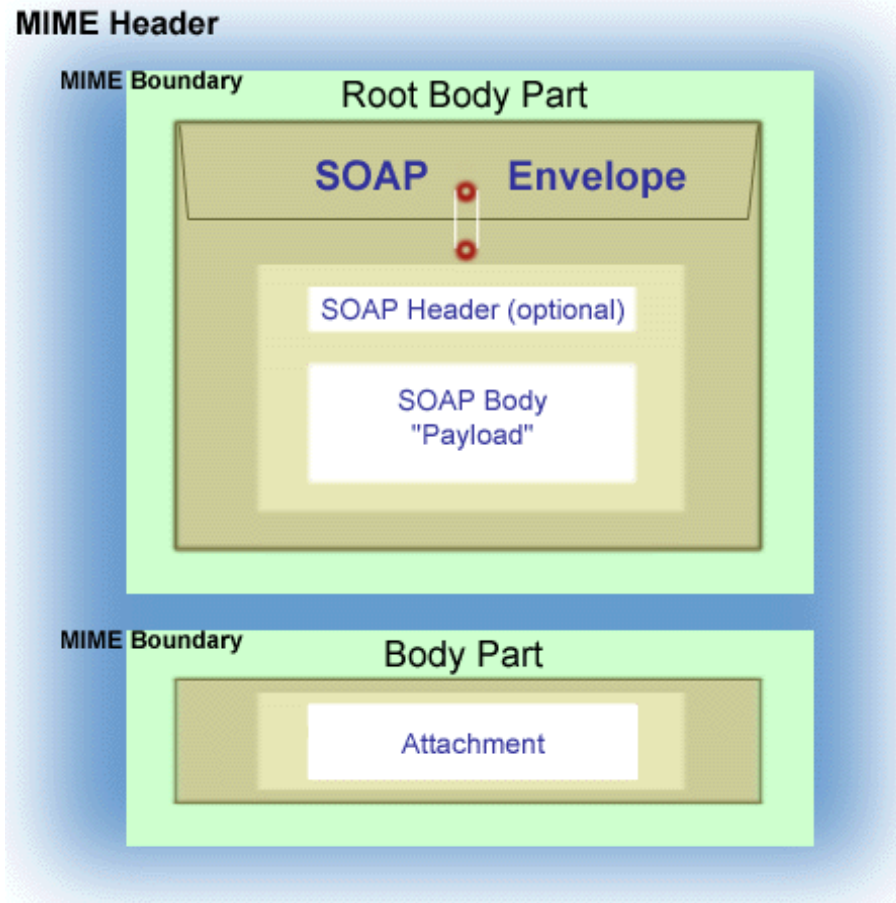
SOAP: A Communications Envelope

- A SOAP message is defined as an XML “envelope”, complete with a Header and Body.



SOAP with Binary Attachments

- A SOAP message can also contain binary attachments using MIME encoding (like email).



SOAP Request Message

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">

  <soap:Header>
    <!-- Header elements go here, and are optional. -->
  </soap:Header>

  <soap:Body>
    <!-- Message or method call elements go here.-->
    <getYear>
      <LicensePlate>779RIQ</LicensePlate>
    </getYear>
  </soap:Body>

</soap:Envelope>
```

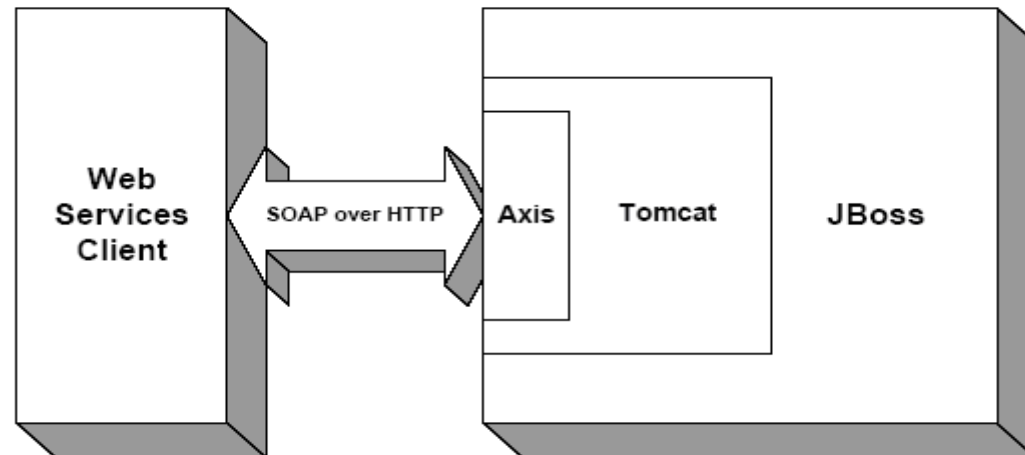
SOAP Response Message

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <!-- Response from web service. -->
    <getYearResponse>1996</getYearResponse>
  </soap:Body>
</soap:Envelope>
```

- ❑ The “soap” portion of the namespace declaration (**xmlns:soap**) is not important, but the URI is **very** important (<http://schemas.xmlsoap.org/soap/envelope/>).
- ❑ Interoperability is centered around the xmlsoap schema, which is the shared reference point between web services written in Java, .NET, and other languages.

SOAP Processing with Axis

- ❑ Axis is a popular open source framework for creating and deploying web services.
- ❑ Clients send SOAP request messages to the server.
- ❑ Axis parses the SOAP XML, calls the requested web service method, and returns a SOAP-encoded response.



Using Axis for Web Services

- ❑ Approach 1: Add your app to Axis
 - ⇒ Install Axis as a web application in <tomcat_home>\webapps (unzip axis.war to create an axis subfolder).
 - ⇒ Copy your web services to <tomcat_home>\webapps\axis (.java files renamed with .jws extension).
 - ⇒ Axis automatically creates wsdl and deployment descriptors when you first run the web service.
 - ⇒ Easy to implement, and the approach we'll use.
- ❑ Approach 2: Add Axis to your app
 - ⇒ Add Axis libraries to your WAR file.
 - ⇒ Copy the Axis Servlet declarations and mappings from axis/WEB-INF/web.xml and add them to your own web.xml.
 - ⇒ Create web service deployment descriptors.
 - ⇒ Build and deploy your webapp.

Web Service Example

```
//Class that Axis will convert to a web service
public class CarService {
    public String getYear(String LicensePlate) {
        System.out.println("License: " + LicensePlate);
        return "1996";
    }
}
```

- ❑ Save the class defined above as CarService.jws in the <tomcat_home>\webapps\axis folder.
- ❑ Enter <http://localhost:8080/axis/CarService.jws?wsdl> in your browser to see the wsdl that Axis generates.
- ❑ Test the web service from your browser using <http://localhost:8080/axis/CarService.jws?method=getYear&LicensePlate=779RIQ>

Creating a SOAP Envelope

- Web service clients need to build and send a SOAP message representing the desired method call, starting with the Envelope:

```
import javax.xml.soap.*;  
import javax.xml.messaging.*;
```

```
//Create Soap Message
```

```
MessageFactory msgFactory =  
    MessageFactory.newInstance();  
SOAPMessage soapMsg = msgFactory.createMessage();
```

```
//Create main Soap objects
```

```
SOAPPart soapPart = soapMsg.getSOAPPart();  
SOAPEnvelope soapEnv = soapPart.getEnvelope();  
SOAPBody soapBody = soapEnv.getBody();
```

Building the SOAP Message

```
//Create the getYear and LicensePlate elements
Name year = soapEnv.createName("getYear");
Name license = soapEnv.createName("LicensePlate");

//Add a request element to the SoapBody
SOAPBodyElement yearElement =
    soapBody.addBodyElement(year);
SOAPElement licenseElement =
    yearElement.addChildElement(license);

//Set the request element's value
licenseElement.addTextNode("779RIQ");
```

Sending a SOAP Message

```
//Create Soap Connection
SOAPConnectionFactory scFactory =
    SOAPConnectionFactory.newInstance();
SOAPConnection sc = scFactory.createConnection();

//Create URLEndpoint and send Soap Message
URLEndpoint endPoint = new
    URLEndpoint("http://localhost/axis/CarService.jws");
SOAPMessage response = sc.call(soapMsg, endPoint);
sc.close();

//TODO: Process the response SOAPMessage object...
```

Processing the SOAP Response

```
//Create Soap objects for the Soap Response
SOAPPart soapRespPart = response.getSOAPPart();
SOAPEnvelope soapRespEnv = soapRespPart.getEnvelope();
SOAPBody soapRespBody = soapRespEnv.getBody();

Name yearResponse = soapRespEnv.createName("getYearResponse");
Name yearReturn = soapRespEnv.createName("getYearReturn");

Iterator it = soapRespBody.getChildElements(yearResponse);
while (it.hasNext()) {
    SOAPElement next = (SOAPElement) it.next();
    Iterator it2 = next.getChildElements(yearReturn);
    while (it2.hasNext()) {
        SOAPElement yearReturnElement = (SOAPElement) it2.next();
        System.out.println(yearReturnElement.getValue());
    }
}
```