

Advanced Web Services & Best Practices

Agenda

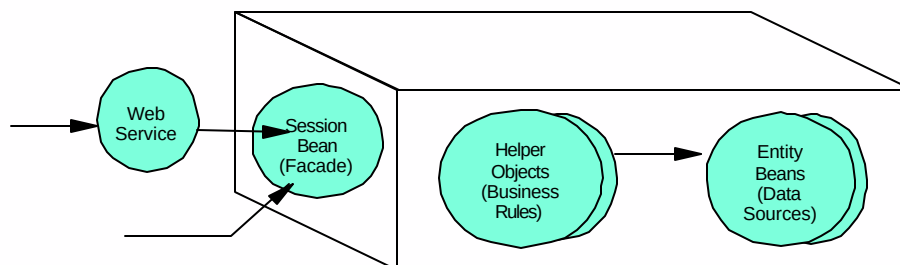
- Design Best Practices
- RPC Vs. Document
- Implementation Best Practices
- XML Best Practices
- Programming Model Best Practices
- Interoperability Best Practices
- Security Best Practices
- Performance Best Practices

Design Best Practices

- Web Services should be coarse-grained interfaces
 - ▶ Apply the Session Facade pattern to reduce chatter
- Web Services should be stateless
 - ▶ Even though it's possible to use transport state mechanisms they should be avoided
- The best web service call is NO web service call
 - ▶ Cache Data on the requestor when reasonable
 - ▶ Minimize the number of requests for the same information
 - ▶ logically group information into larger-grain requests when possible
- Consider asynchronous messaging instead of synchronous RPC-style interaction when return values are not required
- Do NOT use SOAP encoding if you ever expect to interoperate
 - ▶ Use Document/Literal instead
- In these cases, design the WSDL first
 - ▶ Then obtain the implementation from the WSDL
 - ▶ It's ok to start from a "throwaway" implementation to get the first-pass WSDL

Session Facade Pattern

- The Session Facade Pattern ([Alur]) describes how to hide a complex subsystem by creating a coarse-grained "Facade" ([Gamma]) to provide client access to the subsystem
- Session Facades are usually implemented in Java as Stateless Session Beans
 - ▶ Can be done with Standard JavaBeans if you are not using EJB's
 - ▶ The key is that the Facade does no new work -- it's simply there for passthru to existing subsystem functionality



Statefulness

- By default, all Web Services are stateless
 - ▶ There is no way to define a stateful connection in WSDL
 - ▶ Statefulness is not part of the protocol for HTTP or JMS
 - ▶ Many service implementations do not provide for state management
- It is just possible to force statefulness into a Web Service
 - ▶ You could rely on information held in HttpSession or in Singletons
 - ▶ JAX-RPC does allow access to the HttpSession through the ServletEndpointContext
 - You should RESIST the temptation to do so
 - Not scalable on all application servers, ties you to HTTP
 - ▶ Singletons can run into threading problems (standard Servlet programming issues)
- The long and the short of it is that services should remain stateless for scalability

What SOAP bindings are available?

- There are two particular parts of the WSDL Binding we need to examine
 - ▶ The binding mode (Document or RPC)
 - ▶ The encoding style (literal or encoded)

	Document	RPC
Literal	Standard Choice for MS tools	
Encoded		Standard choice for Apache tools

Binding style

- There are two different ways of "binding" your WSDL to the SOAP spec
 - ▶ These refer to different parts of the WSDL and SOAP specification, and have an effect on what XML is actually sent down the wire
- RPC mode says "use the SOAP conventions for RPC"
 - ▶ each part is a parameter or return value and appears inside a wrapper element
 - ▶ The wrapper element is named identically to the operation name
 - ▶ Each message part (parameter) is inside the wrapper represented by an accessor named identically to the corresponding parameter of the call
 - ▶ message parts occur in the same order as the corresponding parameters of the method call
- Document mode says "Do not require a wrapper element"
 - ▶ The message parts appear directly under the SOAP body
 - ▶ Any arbitrary well-formed XML
 - ▶ Up to the SOAP engine what to do the XML Body. Most will pass it a method as a DOM element.
- Looking at a SOAP message, you cannot necessarily tell if it is document or RPC style
 - ▶ important to the SOAP engines because it determines how the body is processed

Example SOAP Message

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <Address>
      <Street>54 First St</Street>
      <City>NYC</City>
      <Zip>3423432</Zip>
    </Address>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Is it Document or is it RPC style?

... You cannot tell looking at this message. Since it has single element in the body it could be either style.

Example Document style SOAP Message

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  <SOAP-ENV:Body>
    <Address>
      <Street>54 First St</Street>
      <City>NYC</City>
      <Zip>34234-4345</Zip>
    </Address>
    <useZipPlus4>true</useZipPlus4>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This message is Document style since it has multiple elements in the body.

Literal XML vs. Encoded XML

- SOAP specification has an optional *encodingStyle* attribute
 - ▶ attribute MAY appear on any element, but usually is on the envelope
 - ▶ there is no default encoding defined for a SOAP message
 - ▶ there is an OPTIONAL SOAP encoding which is included in the spec
- If the encoding style is not provided, it is 'literal'
- SOAP encoding style is based on a simple type system that is a generalization of the common features found in type systems in programming languages
 - ▶ A type either is a simple (scalar) type or is a compound type constructed as a composite of several parts, each with a type
 - ▶ encodingStyle attribute should be set to <http://schemas.xmlsoap.org/soap/encoding/>
 - ▶ used almost exclusively with RPC style
- Each operation's input and output elements must be declared to use either "literal" or "encoded" style
 - ▶ If use is "encoded" then each message part refers to an abstract type (in the types section) using the types attribute. The encoding specified in the encodingStyle attribute determines how the output will be encoded
 - ▶ If use is "literal" then each part references a concrete schema definition using either the element or the type attribute.

When do you use document mode?

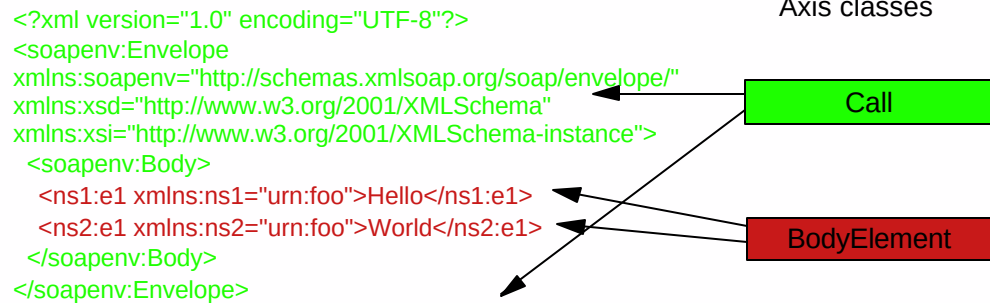
- When do you use RPC Style?
 - ▶ You would choose RPC style when you have an existing implementation
 - Adopting a Java bean or an EJB to be a web services
 - Application is not already using XML
 - ▶ This works best when both client and server are written in Java
- When would you use Document mode?
 - ▶ The primary reason to use document style is when communicating with outside partners
 - Cannot specify the tool set they may use
 - Cannot specify the object design on each end
 - The publisher can't control who use the service and what the consequences are when changes are made
 - ▶ Document mode eases validation of complex documents
 - XML Schema validation makes this simple
 - Although currently neither the Apache SOAP or Axis engines do Schema validation
 - It could be added at the transport layer
 - ▶ Data already exists as XML and/or will be processed as XML throughout the system
 - ▶ Document mode is more tolerant to version changes in the XML

Document (Message) style servers in Axis

- It is possible to write Message style servers in Axis
- All message-style servers have the same signature
 - ▶ void name(Vector elems)
 - ▶ Each element in the Vector is an org.w3c.dom.Element
- It's probably only worthwhile if
 - ▶ You are exchanging existing XML documents that are not in "RPC form"
 - ▶ You need to communicate with an existing Document Style Service
 - ▶ You will need to interoperate with MS (more on this later)

Document (Message) style clients in Axis

- The Axis Call object has an invoke method that takes an array of org.apache.message.SOAPBodyElements (implements the SAAJ SOAPBodyElement interface)
- Each SOAPBodyElement contains an org.w3c.dom.Element



IBM Software Group



Axis Message Style Clients

```
public class TestMsg {
    public String doit(String[] args) throws Exception {
        Options opts = new Options(args);
        opts.setDefaultURL("http://localhost:8080/axis/services/MessageService");
        Service service = new Service();
        Call call = (Call) service.createCall();
        call.setTargetEndpointAddress( new URL(opts.getURL()) );
        SOAPBodyElement[] input = new SOAPBodyElement[2];

        input[0] = new SOAPBodyElement(XMLUtils.StringToElement("urn:foo",
                                                                "e1", "Hello"));
        input[1] = new SOAPBodyElement(XMLUtils.StringToElement("urn:foo",
                                                                "e1", "World"));

        Vector elems = (Vector) call.invoke( input );
        SOAPBodyElement elem = null ;
        Element e = null ;
        elem = (SOAPBodyElement) elems.get(0);
        e = elem.getAsDOM();
        String str = "Res elem[0]=" + XMLUtils.ElementToString(e);
        elem = (SOAPBodyElement) elems.get(1);
        e = elem.getAsDOM();
        str = str + "Res elem[1]=" + XMLUtils.ElementToString(e);
        return( str );
    }
}
```

IBM Software Group



Implementation Best Practices

- In nearly all synchronous request/response cases, a stateless Session EJB should be the preferred vehicle
 - ▶ The only way to have fine-grained security
 - ▶ When a client can use RMI-IIOP (through WSIF) the performance gains are immense
 - ▶ Javabeans web services need to be justified to be used
- When you are using JavaBeans prefer Application Scope
 - ▶ Or consider static methods if that's your preference
- Use the JAX-RPC programming model
 - ▶ Allows for the best portability between application servers

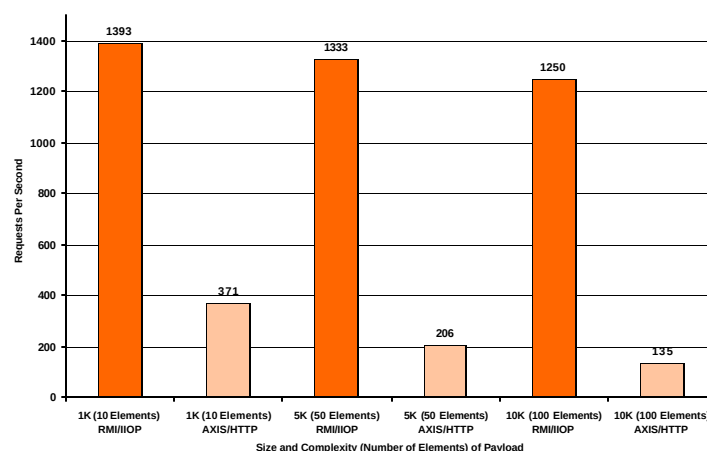
IBM Software Group



SOAP AXIS vs. RMI/IIOP

Results: RMI/IIOP far out performs Axis over HTTP. If any of the requestors are going to be Java applications/middle tier, use EJBs to implement the web service. Extremely significant.

EJBs RMI/IIOP vs. EJBs AXIS "Alpha 3" - Primitive Performance
Examines - IIOP/GIOP Marshaling vs. XML/AXIS/HTTP Marshaling



Notes: If the application server may have Java as a service requestor, have that Java application make an RMI/IIOP call to an EJB instead of using SOAP over HTTP. Support both RMI/IIOP and web services interfaces.

IBM Software Group





XML Best Practices

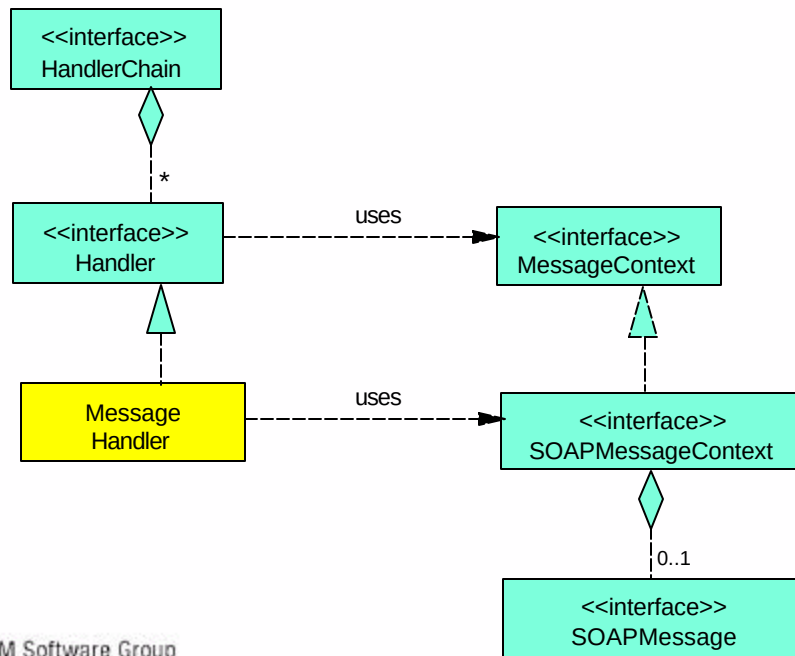
- Always use company or domain specific namespaces in your WSDL
- Use easily-readable, descriptive names for your elements
 - ▶ It's worth the few extra bytes
- Explicitly name your types in XML Schema
- Annotate your WSDL
 - ▶ It's hard enough to understand as it is...
- Be careful when designing your schema
 - ▶ Don't send an entire object when a reference will do
- Use elements for data, attributes for meta-data



Programming Model Best Practices

- Use Headers for Infrastructure-related information, not application-related information
 - ▶ Plan to use the JAX-RPC Header handler features for headers
 - ▶ Use the Axis Header handling features as a fallback position
- Use custom serialization for odd encodings or where the client and server do not match

Header handling classes



IBM Software Group



Header Handling Architectures

- Handlers implement the `javax.xml.rpc.handler.Handler` interface
- The `MessageContext` provides access to the message context (the request or response)
- Handlers are invoked in order of a `HandlerChain`

IBM Software Group



Header handling

- The `handleRequest()` method does one of the following
 - ▶ return true to indicate that the processing of the handler chain should continue
 - ▶ return false to indicate that handler chain is blocked. In this case, start processing of the `handleResponse()` methods of this handler instance and the handlers backward in the chain
 - ▶ throw a `SOAPFaultException` to indicate a SOAP Fault (which begins the invocation of the `handleFault()` chain)
 - ▶ throw a `JAXRPCException` or other `RuntimeException` for a runtime error (in which case the `HandlerChain` will generate a SOAP fault)
- The `handleResponse()` mechanism is similar, but without the extra handler processing on the false case (after blocking no other handlers are processed)

```
package javax.xml.rpc.handler;  
public interface Handler {  
    boolean handleRequest(MessageContext ctx);  
    boolean handleResponse(MessageContext ctx);  
    boolean handleFault(MessageContext ctx);  
}
```

Handler Chains

- A Handler is associated with a SOAP header block using the qualified name of the outermost element of each header block
- A Handler chain performs the following steps during SOAP processing
 - ▶ Identify the set of SOAP actor roles in which this Handlerchain is to act
 - ▶ Identify all mandatory header blocks for this node
 - ▶ If one or more of the mandatory blocks are not handled by this node then generate a SOAP `MustUnderstand` fault
 - ▶ else
 - ▶ Process all header blocks by invoking the chain of handlers
 - ▶ If processing is unsuccessful, generate exactly one SOAP fault to propagate to the client

JAX-RPC Handlers in Axis

- In Axis JAX-RPC handlers are declared in the deployment descriptor in a requestFlow or responseFlow element
- The JAX-RPC example for Axis shows how they are used:

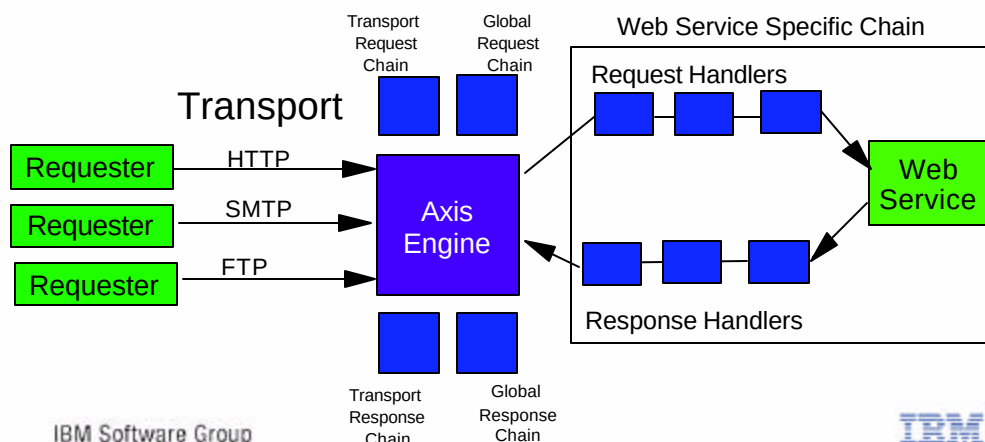
```
<service name="HelloPort" provider="java:RPC">
  <parameter name="wsdlTargetNamespace" value="http://hello.jaxrpc.samples"/>
  <parameter name="wsdlServiceElement" value="HelloWorld"/>
  <parameter name="wsdlServicePort" value="HelloPort"/>
  <parameter name="className" value="samples.jaxrpc.hello.HelloBindingImpl"/>
  <parameter name="wsdlPortType" value="Hello"/>
  <parameter name="scope" value="session"/>
  <operation name="sayHello" qname="operNS:sayHello" xmlns:operNS="http://hello.jaxrpc.samples" returnQName="result" >
    <parameter name="String_1" type="tns:string" xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
  </operation>
  <parameter name="allowedMethods" value="sayHello"/>
  <requestFlow>
    <handler type="java:org.apache.axis.handlers.JAXRPCHandler">
      <parameter name="scope" value="session"/>
      <parameter name="className" value="samples.jaxrpc.hello.ClientHandler"/>
    </handler>
  </requestFlow>
  <responseFlow>
    <handler type="java:org.apache.axis.handlers.JAXRPCHandler">
      <parameter name="scope" value="session"/>
      <parameter name="className" value="samples.jaxrpc.hello.ClientHandler"/>
    </handler>
  </responseFlow>
</service>
</deployment>
```

IBM Software Group



Axis Header handling (pre JAX-RPC)

- Axis has a more flexible header handling system than JAX-RPC (for instance, you can attach header handlers to transport handlers)
- However, over time it will be probably deprecated in favor of the JAX-RPC standard
 - JAX-RPC guarantees portability across Application Servers



IBM Software Group



List of Simple Mapped Types

- Remember that in encoded style interfaces the Axis deployment descriptor describes how to map an accessor to a type
- The following simple types are directly mapped to XML in JAX-RPC

XSD	Java Type
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int*	int
xsd:long*	long
xsd:short*	short
xsd:float*	float
xsd:double*	double
xsd:boolean*	boolean
xsd:byte*	byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	java.util.Calendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]

*If the "nillable" attribute is set for an element, the Wrapper type is used

Complex Type mappings

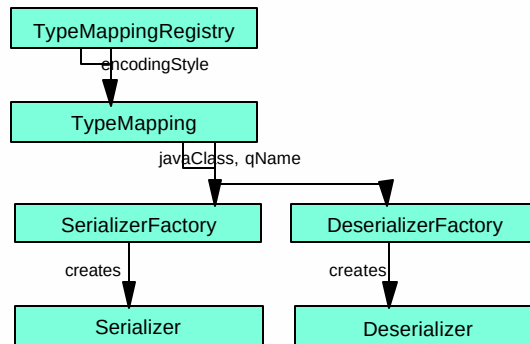
- There are also two sets of complex types that JAX-RPC supports

Java Type	XML Schema
BigDecimal	xsd:decimal
Date	xsd:dateTime

- Java Arrays are mapped to SOAP:ENC encoded arrays
 - Unless they are part of a Value Object, in which case they are mapped to an xsd:complexType with the maxOccurs attribute set to a non-negative number or unbounded
- You can also choose to use soap encoding for standard classes like String, int, etc.

Custom Serialization

- You can choose to provide your own mechanism for interpreting incoming XML and creating Java objects from it, and likewise creating XML from a Java object type
- This is called Extensible Type Mapping in the JAX-RPC specification



Why use Custom Serializers?

- The main reason is so that client and server can have different implementations
 - ▶ You can't assume you have the same infrastructure and the same parsing structure on both ends of the conversation (ex. a C++ client with an RPC-style server)
 - ▶ Perhaps you need to maintain communication with older clients while you change the underlying implementation
 - ▶ JAX-RPC will try to match fully qualified Java class names
- This is yet another reason not to rely on RPC and SOAP Encoding for the long term
 - ▶ Literal XML use would have avoided this problem altogether
- Can also use with unusual encodings
 - ▶ Such as when a String-valued or binary-encoded accessor encodes information
 - ▶ (e.g. perhaps you want compression on some fields, or you want to use binary serialization)
- Also, you would choose to use a custom serializer in the cases where in normal Java serialization you would need to use a transient field
 - ▶ for instance for socket connections or EJB references

Interoperability Best Practices

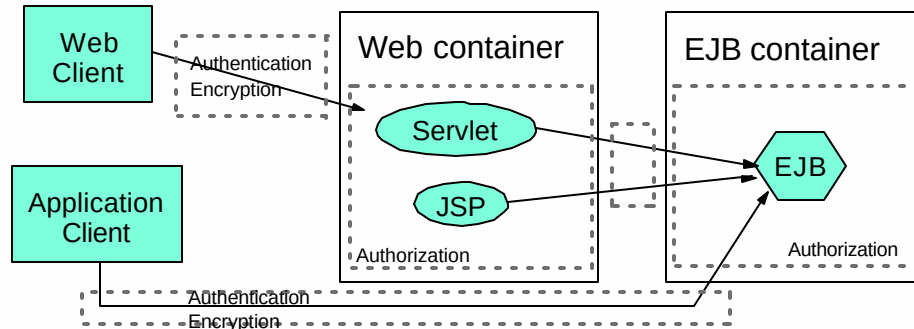
- Include SOAPAction in your HTTP headers
 - ▶ even though it's not required by all SOAP engines, some need it
 - ▶ Axis (for instance) uses a 3-step dispatching System
 - By URL
 - By SOAP Action
 - By namespace
- Keep your schema and WSDL as simple as possible
- Test for interoperability from the start
 - ▶ Beware of Forgiving SOAP engines
- Stay informed
 - ▶ SOAP-Builders, Sam Ruby's website and articles

Security Best Practices

- Enable Application Server Security
 - ▶ Lock down your RPC-Router and Message router servlets to at least "all authenticated users"
 - ▶ Using EJB's allows you finer-grained authorization
- Use Basic-Auth for user identification
 - ▶ Don't try embedding this kind of stuff in the message yet
- Use HTTPs for encryption over the wire
 - ▶ especially when using basic-auth

J2EE Supported Topology

- J2EE and WebSphere protect a set of interactions that the user can have with the system
 - ▶ Encryption (guarantee private conversations)
 - ▶ Authentication (proving identity)
 - ▶ Authorization (only certain people can access J2EE resources)

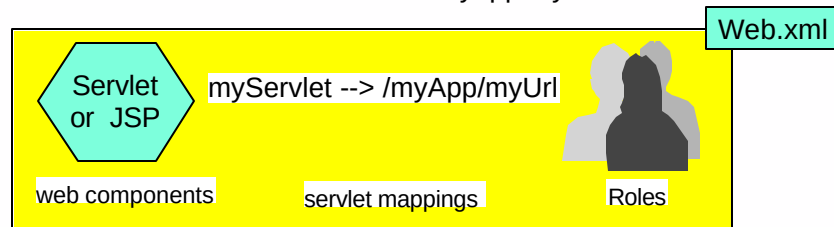


IBM Software Group

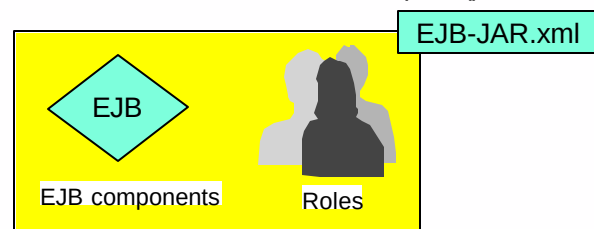


J2EE Role Mapping

"Users who are Admins can POST to /myapp/myUrl"



"Users who are Tellers can send deposit() to instances of the Account EJB"



IBM Software Group



Using HTTP Basic-Auth with JAX-RPC

- In JAX-RPC you can set the HTTP Basic-Auth username and password either through obtaining a provider with them pre set, or set the username and password properties in the Stub instance

```
StockQuoteService sqs = // ... Get access to the service
StockQuoteProvider sqp = sqs.getStockQuoteProviderPort(
    "<username>", "<password>");
float quote = sqp.getLastTradePrice("ACME");
```

```
StockQuoteProvider_Stub sqp = // ... get to the stub;
sqp._setProperty(Stub.USERNAME_PROPERTY, "<username>");
sqp._setProperty(Stub.PASSWORD_PROPERTY, "<password>");
```

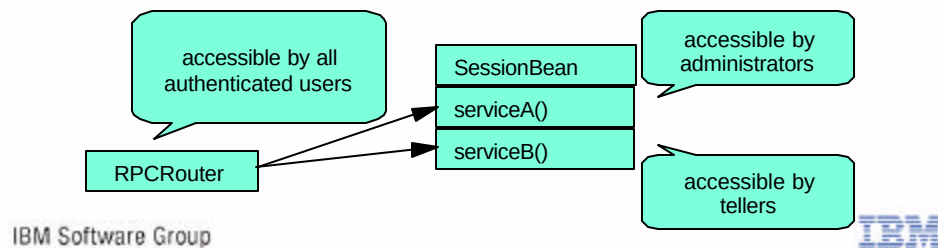
Basic HTTP-Auth with Axis

- When you are using a standard Axis (non-JAX-RPC) client then you simply use the setUsername() and setPassword() methods on the Call object

```
URL url = new URL(opts.getURL());
String user = opts.getUser();
String passwd = opts.getPassword();
Service service = new Service();
Float res = new Float(0.0F);
Call call = (Call) service.createCall();
call.setTargetEndpointAddress( url );
call.setOperationName( new QName("urn:xmltoday-delayed-quotes", "getQuote") );
call.addParameter( "symbol", XMLType.XSD_STRING, ParameterMode.IN );
call.setReturnType( XMLType.XSD_FLOAT );
call.setUsername( user );
call.setPassword( passwd );
Object ret = call.invoke( new Object[] {symbol} );
if (ret instanceof String) {
    System.out.println("Received problem response from server: "+ret);
    throw new AxisFault("", (String)ret, null, null);
}
res = (Float) ret;
return res.floatValue();
```

Securing Web Services Resources

- Since WAS only knows about EJB's and Servlets, we have two options for securing our Web Services
 - ▶ Very high granularity (ALL web services in that WAR file)
 - ▶ Should be done in any case, but should be supplemented
 - ▶ Probably should be limited to "all authenticated users"
- Secure the RPCRouter Servlet
 - ▶ Can secure each Web Service method individually that way
 - ▶ Can specify more specific roles for each method
- Implement your Web Services as Session EJB's
 - ▶ Can secure each Web Service method individually that way
 - ▶ Can specify more specific roles for each method
- Note that this means you cannot secure Java Beans Web Services at a fine grained level with WebSphere Security

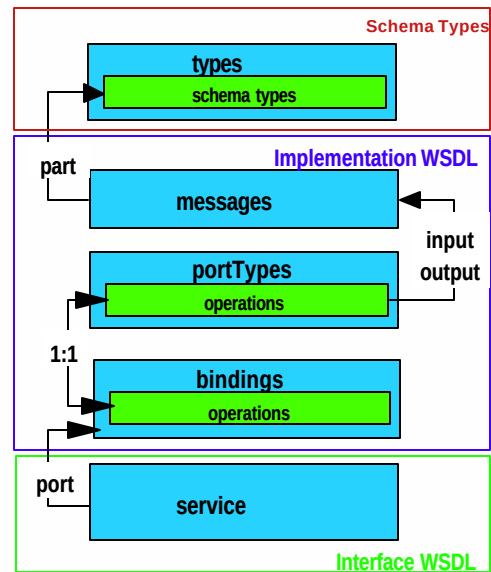


WSDL Implementation Best Practices

- Use one service per WSDL document
 - ▶ helps with interoperability
- Divide the WSDL into separate interface and implementation files
 - ▶ Know how to make this work in Visual Studio
- Divide out the Schema types into separate documents
- Make the WSDL available by URL

WSDL Elements

- A WSDL document defines Web Service via
 - **Messages**
 - Defines a single interaction with the service
 - **Types**
 - Defines data types used in a message
 - **Operations**
 - Description of an action
 - **Port Types**
 - Describes the set of operations supported by the service.
 - **Bindings**
 - A concrete protocol and data format for a particular port type.
 - **Port**
 - Describes the network address where the service is being hosted.
 - **Service**
 - Ties together all the elements of the service.



Java Implementation Best Practices

- Plan for incremental development
- Don't initialize required variables in your default constructors for serialized objects
- Plan for doing automated testing
 - ▶ Using JUnit -- search the web for this
 - ▶ Axis will generate JUnit tests automatically

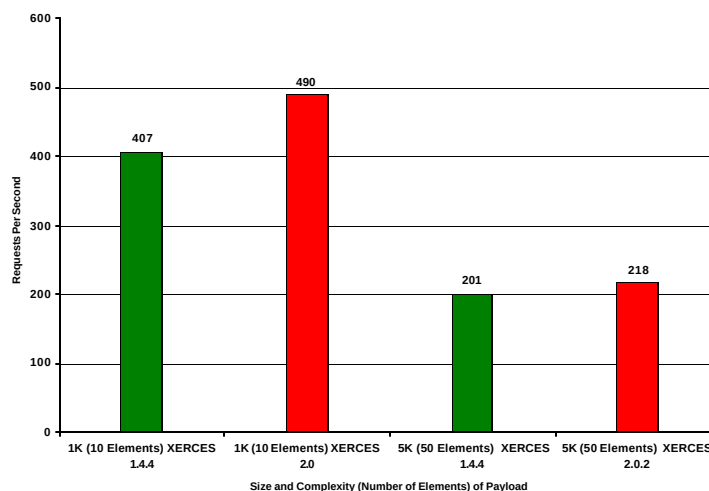
Performance Best Practices

- Use the fastest XML Parser available
 - ▶ Keep your eye on the latest Apache releases
- Keep your messages as small and "flat" as you can
 - ▶ Fewest bytes
 - ▶ Least complexity (fewest elements)
- Use the fastest SOAP Engine available
 - ▶ For instance, Apache Axis is significantly faster than Apache SOAP
- Use SOAP Attachments for sending binary data
 - ▶ Much faster and smaller than base-64 encoding

Xerces Parser Comparison

Results: Use the fastest parser available. Do not take default. Use Xerces 2.0 over 1.4.4 with AXIS.

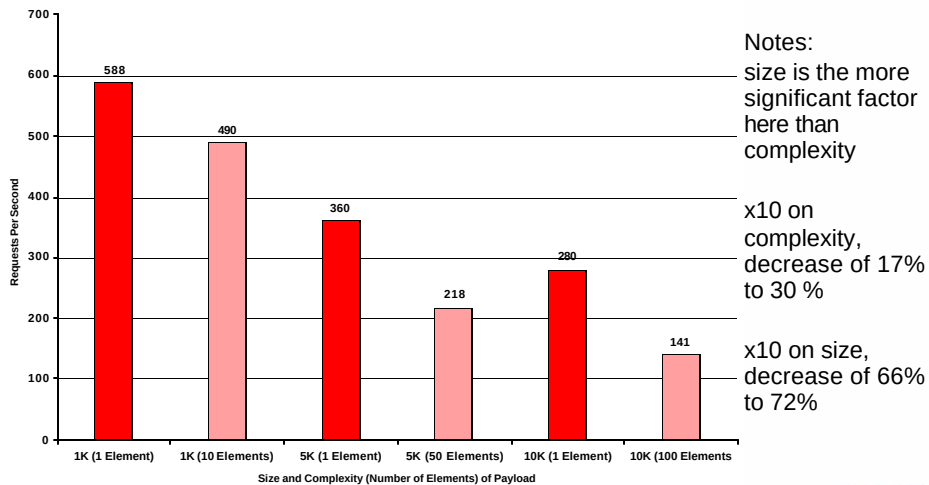
AXIS "Alpha 3" Effect of the Parser - XERCES 2.0 vs. XERCES 1.4.4



Impact of Message Size and Complexity

- Results: 1) Smaller messages are better
2) Fewer elements are better (less significant)

AXIS - "Alpha 3" - Primitive Performance - Size and Complexity of Payload



IBM Software Group



Binary Data

- Sometimes you want to send binary data within a SOAP message
 - ▶ JPG files, other legacy binary formats
- Apache Axis allows you to transmit binary data as part of the message by using base64 encoding
 - ▶ When you send use byte array in Java as a Web Service parameter, it's mapped using the Base64Serializer to SOAP-ENC:base64#
 - ▶ Translates the binary data to plain text
 - ▶ Increases the size by nearly 33%
- So, instead, to send the data without the increase in size, use SOAP with Attachments

```
<Picture xsi:type = "SOAP-ENC:base64">  
aG93IG5vDyBicm73biBj....  
</Picture>
```

IBM Software Group



SOAP with Attachments

- In SOAP with Attachments the entire message plus attachments is constructed as a multipart MIME message

```
MIME-Version: 1.0
Content Type: Multipart/Related; boundary=MIME_boundary;...
start="http://myUrl/soap/attachments/fileupload1"
--MIME_boundary

Content-Type: text/xml
...
<?xml version='1.0'?>
<SOAP: Envelope>
...
</SOAP:Envelope>

--MIME_boundary

Content-Type: image/gif
Content-Transfer-Encoding: binary
...
gif image

--MIME_boundary
```

IBM Software Group



Using SOAP With Attachments in JAX-RPC

- JAX-RPC makes using SOAP with Attachments automatic
- If you are using an RPC-style service, then any parameter that is an instance of `javax.activation.DataHandler` is automatically sent as an attachment
- MIME types defined in WSDL using the MIME binding are automatically mapped to the following classes:

image/gif	java.awt.Image
image/jpeg	java.awt.Image
text/plain	java.lang.String
multipart/*	javax.mail.internet.MimeMultipart
text/xml or application/xml	javax.xml.transform.Source

**However, the Axis JAX-RPC support doesn't handle SOAP w/attachments yet

IBM Software Group



Sample Code for SOAP with Attachments

- This example uses a `javax.activation.DataHandler` and renders a file as a MIME multipart attachment

```
package com.ibm.demo.services;
import javax.activation.*;

public class SourceFileService {
    public SourceFileService() {
    }

    public javax.activation.DataHandler getFileNamed(String name) {
        try {
            FileDataSource datasource = new FileDataSource(name);
            DataHandler handler = new DataHandler(datasource);
            java.io.InputStream stream = handler.getInputStream();
            return handler;
        } catch (Exception e) {
            System.out.println("Exception caught on file " + name + " " + e);
            return null;
        }
    }
}
```

IBM Software Group



Axis Deployment Descriptor

```
<!-- This file can be used to deploy the SourceFileService sample -->
<!-- using this command: java org.apache.axis.client.AdminClient attachdeploy.wsdd -->

<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:ns1="urn:SourceFileService" >
    <service name="urn:SourceFileService" provider="java:RPC" >
        <parameter name="className" value="com.ibm.demo.axis.jaxrpc.mime.SourceFileService"/>
        <parameter name="allowedMethods" value="getFileNamed"/>
        <operation name="echo" returnQName="returnqname" returnType="ns1:DataHandler" >
            <parameter name="name" type="xsd:string"/>
        </operation>

        <typeMapping deserializer="org.apache.axis.encoding.ser.JAFDataHandlerDeserializerFactory"
            languageSpecificType="java:javax.activation.DataHandler" qname="ns1:DataHandler"
            serializer="org.apache.axis.encoding.ser.JAFDataHandlerSerializerFactory"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            />
    </service>
</deployment>
```

IBM Software Group



Axis Client

```
public DataHandler getFileNamed(String filename) throws Exception {
    Service service = new Service();
    Call call = (Call) service.createCall();
    call.setTargetEndpointAddress(new URL(SERVICE_URL));
    //Set the target service host and service location.
    call.setOperationName(new QName("urn:SourceFileService", "getFileNamed"));
    //This is the target services method to invoke.
    call.addParameter("fileName", XMLType.XSD_STRING, ParameterMode.IN);
    //Add serializer for attachment.
    QName qnameAttachment = new QName("urn:SourceFileService", "DataHandler");
    call.registerTypeMapping(javax.activation.DataSource.class,
        qnameAttachment,
        JAFDataHandlerSerializerFactory.class,
        JAFDataHandlerDeserializerFactory.class);
    call.setReturnType(qnameAttachment);
    Object ret = call.invoke(new Object[] { filename }); //Add the attachment.
    if (null == ret) {
        System.out.println("Received null ");
        throw new AxisFault("", "Received null", null, null);
    }
    if (ret instanceof String) {
        System.out.println("Received problem response from server: " + ret);
        throw new AxisFault("", (String) ret, null, null);
    }
    if (!(ret instanceof DataHandler)) {
        //The wrong type of object that what was expected.
        System.out.println("Received problem response from server:" + ret.getClass().getName());
        throw new AxisFault("",
            "Received problem response from server:" + ret.getClass().getName(), null, null);
    }
    //Still here, so far so good.
    DataHandler rdh = (DataHandler) ret;
    return rdh;
}
```

IBM Software Group



More Performance Best Practices

- Requestor side caching of data
 - ▶ The requestor is almost always another server - not an 'end user' or browser
 - ▶ The "Best Web Services Call" is the one that you don't have to make
 - ▶ 60% of data can be obtained and cached
 - Lazy as needed
 - Up Front
 - ▶ Cache the Transferred Business Objects
 - Proxy Pattern
- Keep Exchanges Simple
 - analyze business requirement
 - ex. acknowledge a Purchase Order by referencing the PO number, not resending the PO

IBM Software Group

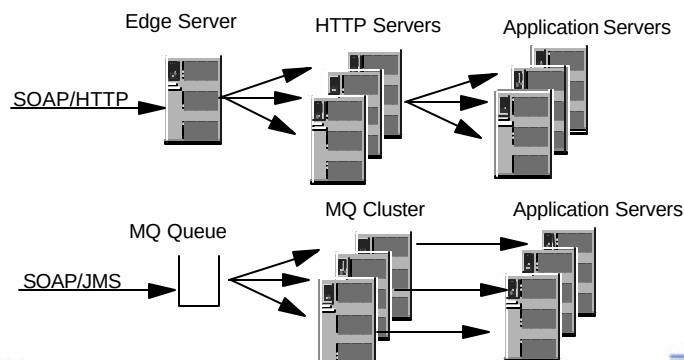


Scaleability

- Web Services per se have nothing to say about scaleability
 - ▶ This is due to the protocol independence of SOAP and WSDL
 - ▶ The scalability of your application is dependent largely on the scalability of the underlying protocol and application server
- So what you have (generally) is the same scaling options as you have for EJB's, JMS/MQ Series and for Java Servlets

Scaling WebSphere Web Services

- For HTTP transport services deploy the Web Service on multiple machines in an application server cluster
 - ▶ For instance, in WebSphere, Edge Server and WebSphere plugin load-balancing spreads the load appropriately
 - ▶ Do not have to specify any HTTPSession pinning options
- For JMS transport services, use transport level scaling (MQ Series clustering)
 - ▶ A single "virtual" Queue is spread over many machines in a cluster





Summary

- We've seen:
 - ▶ Design Best Practices
 - ▶ RPC Vs. Document
 - ▶ Implementation Best Practices
 - ▶ XML Best Practices
 - ▶ Programming Model Best Practices
 - ▶ Interoperability Best Practices
 - ▶ Security Best Practices
 - ▶ Performance Best Practices