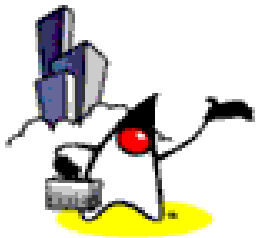
Document-driven Web Services Programming using JAX-RPC



Sang Shin

sang.shin@sun.com
Java™ Technology Evangelist
Sun Microsystems, Inc.
www.javapassion.com/webservices

Disclaimer & Acknowledgement

- Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here are created as his own personal endeavor and thus does not reflect any official stance of Sun Microsystems.
- Sun Microsystems is not responsible for any inaccuracies in the contents
- Acknowledgments
  - Some contents are borrowed from the presentation slides of Roberto Chinnici, Rahul Sharma, Phil Goodwin, Sridhar Reddy (all Sun Microsystems)
  - Some contents are borrowed from Java WSDP tutorial and Web services blueprint documents

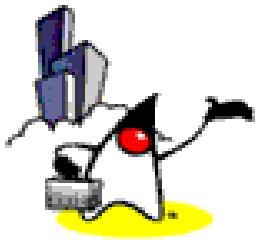
Revision History

- 10/01/2002: version 1, Created (Sang Shin)
- 01/30/2004: version 2, updated (Sang Shin)
- 03/19/2005: version 3, separated from Advanced JAX-RPC presentation, add more slides on document-driven web services based on Sameer Tyagi's Patterns and Strategies for Building Document-based Web Services" article (Sang Shin)
- Things to do
  - speaker notes need to be added

Advanced Topics

- Document-driven Web service
- Formatting vs. Processing
- 8 strategies of implementing document-driven Web services

Document-driven Web Services vs. RPC-based Web Services



RPC vs. Document-driven WS

RPC-based

- Procedure call
- Method signature
- Marshaling
- Tightly-coupled
- Point to point
- Synchronous
- Typically within Intranet

Document-driven

- Business documents
- Schema
- Parsing & Validating

- Loosely coupled
- End to end
- Asynchronous
- Typically over internet

When to use Which model?

RPC-based

- Within Enterprise
- Simple, point-to-point
- Short running business process
- Reliable and high bandwidth
- Trusted environment

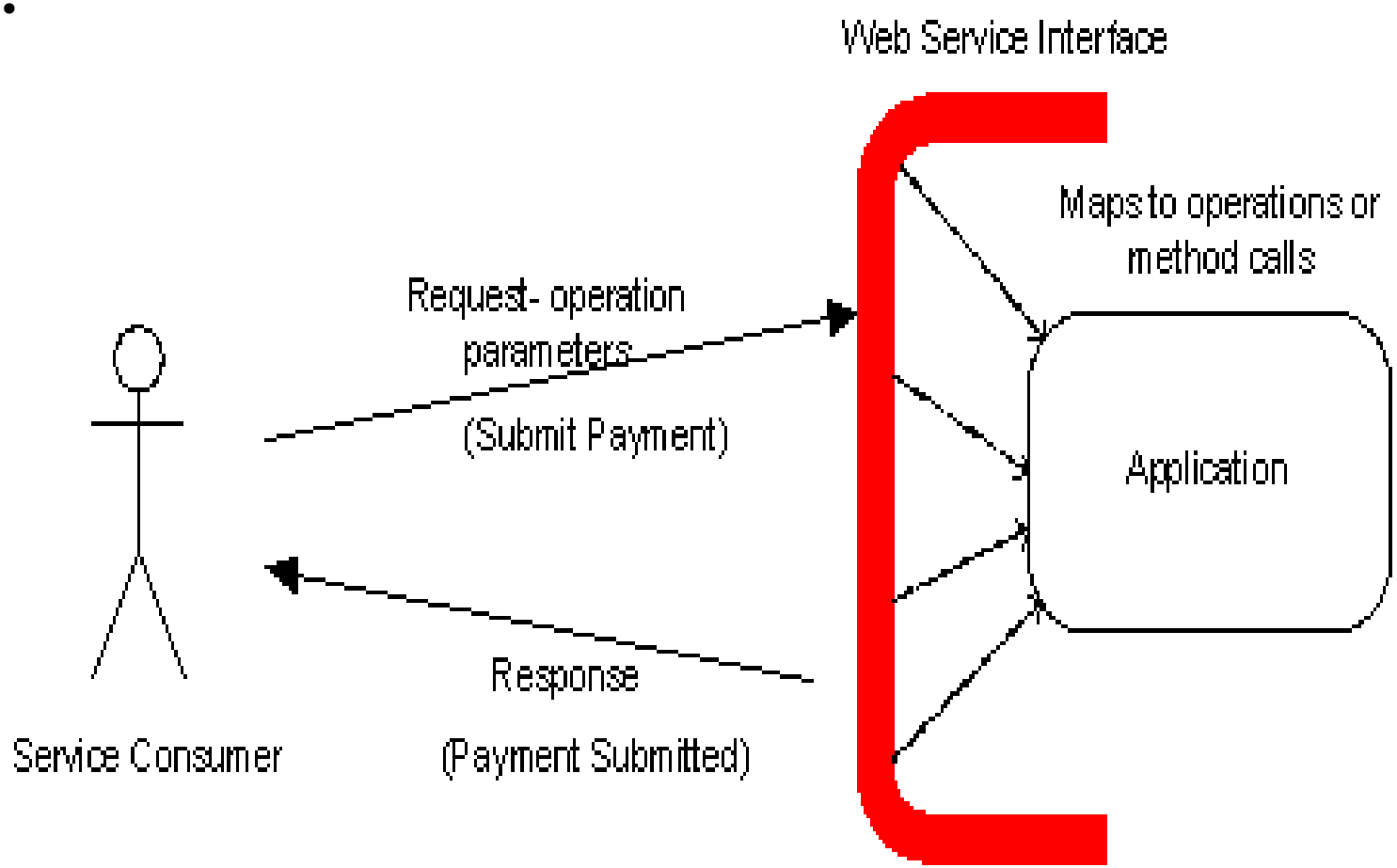
Document-driven

- Between enterprise and enterprise
- Complex, end to end with intermediaries
- Long running business process
- Unpredictable bandwidth
- Blind trust

RPC Based Interaction

- Web service is viewed by consumer as single logical application or component with encapsulated data
- The WSDL describes the exposed interface
- XML in the SOAP messages formatted to map to discrete operations in the application
- Messages directly map to input output parameters of procedure calls or operations
- Typically such invocations occur over a synchronous transport protocol like HTTP
  - SOAP request/response piggy backed on the protocol level
  - request/response to form synchronous request/ response pattern

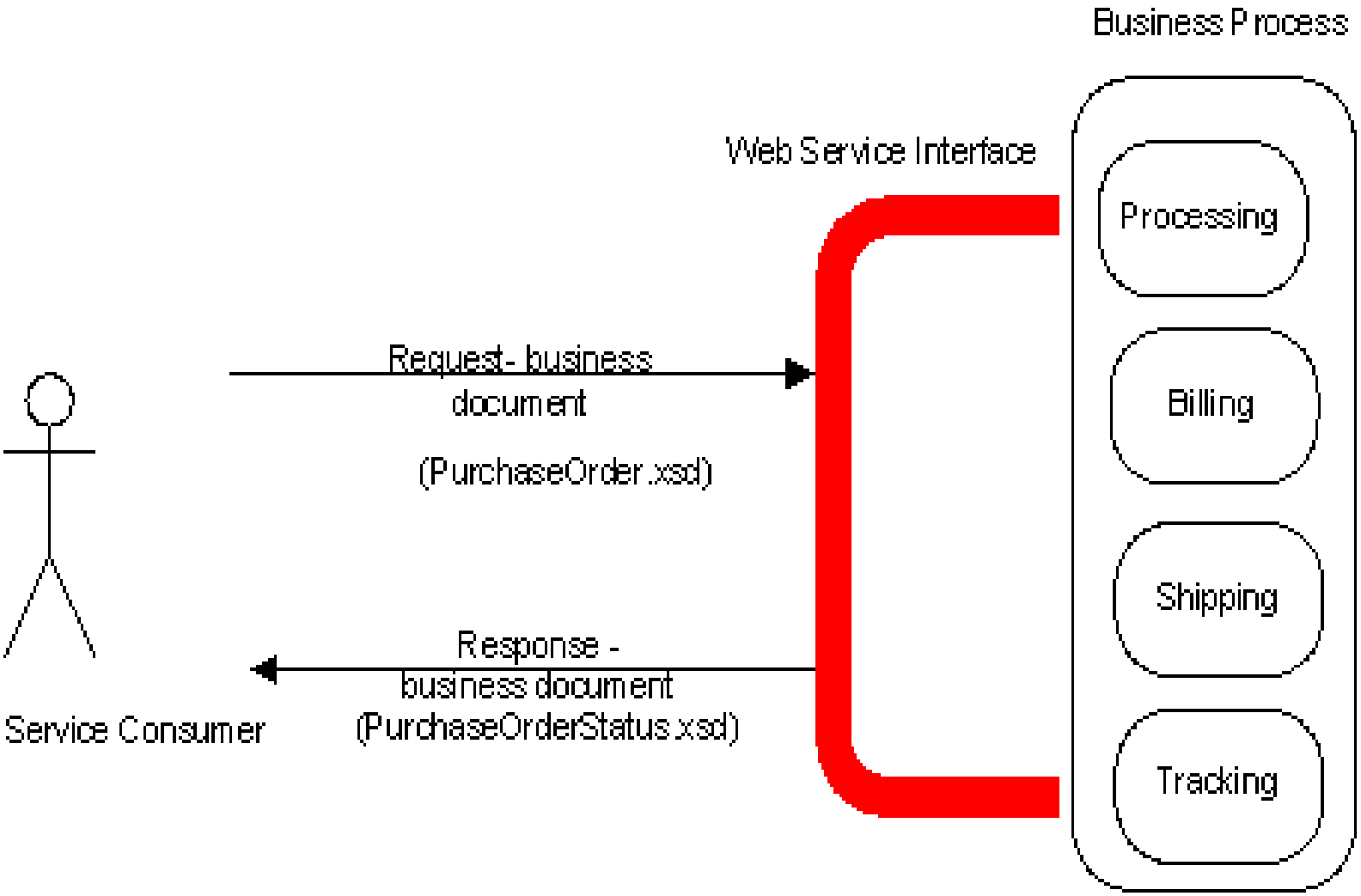
RPC Based Interaction



Document Based Interaction

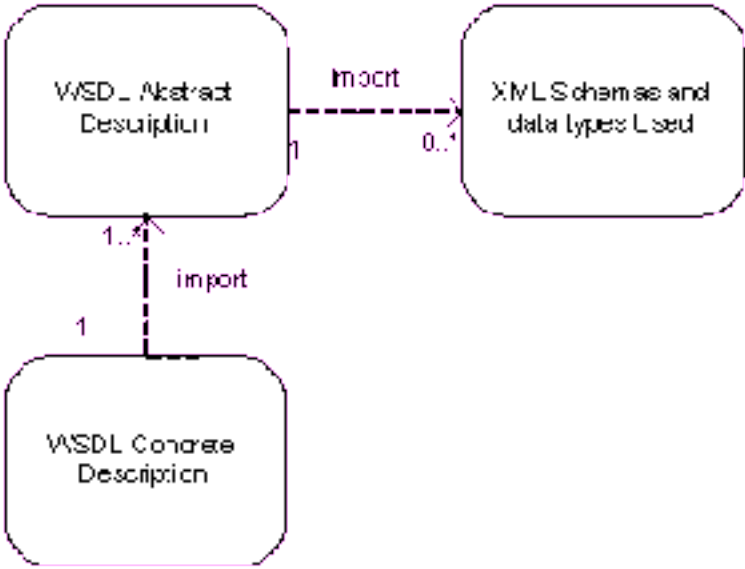
- Service consumers interact with the service using XML documents
- Documents are meant to be processed as complete entities
- Defined by a commonly agreed schema between the service provider and service consumer

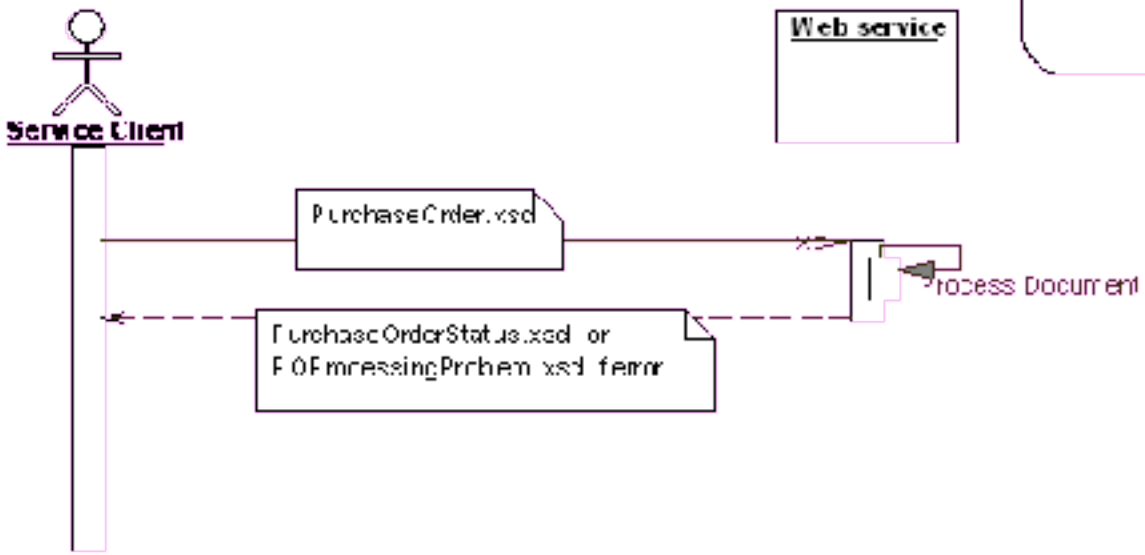
Document Based Interaction



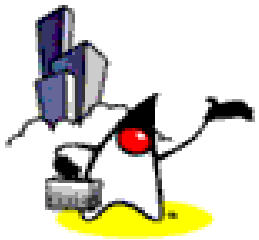
Building Document Driven Services

- Typically start with defining
  - XML documents exchanged
  - Faults
  - WSDL contract





Formatting vs. Processing



Formatting vs Processing

- Design of an RPC or document driven Web service is orthogonal to the formatting and the representation of the SOAP message on the wire
- Often confusing because Document-literal and RPC-encoded nomenclature is used to describe the formatting
  - Interchangeably used to describe processing but there is a difference

Formatting vs Processing

- Formatting refers to representation of the SOAP message on the wire
  - Choice is governed by style attribute in WSDL
  - Either "*RPC*" style or "*document*" style
- Encoding refers to how data is serialized and sent over the wire
  - This is specified by the use attribute in WSDL
  - Can be *"literal"* or *"encoded"*
- Thus four possible combinations

Choosing a Formatting Style

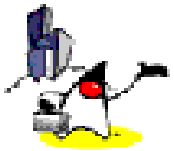
- State maintenance
  - If stubs cannot maintain state, pass the contents of an entire transaction as an XML document
  - Endpoint must ensure the processing sequence, maintain state
- Vertical Vocabularies & Industry standard schemas
  - Document style because its not constrained by RPC oriented encoding
- Validate business documents
  - Endpoint can use validating parser, runtime for syntactic validation

Choosing a Formatting Style

- Performance and memory limitations
  - RPC-encoded scheme is the least performing
  - Other parsing technologies (SAX StAX)
- Interoperability
  - Tools tend to expose programming language structures in WSDL with RPC style
  - WS-I BP limits use of encoding, encourages literal formatting
  - Some toolkits like .NET only support document-literal

Document-Driven Web Services: 8 Strategies

(Based on Sameer Tyagi's “Patterns and Strategies for Building Document-based Web Services” article and sample code)

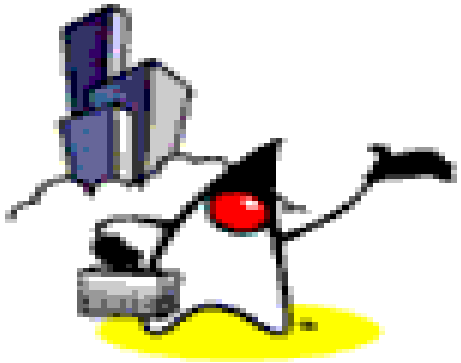


Implementation Strategies

- Using XML in the SOAP body
  - Starting with a WSDL
  - Staring with Java code
- Using String in the SOAP body
- Using base64Encoded or raw bytes in the SOAP body
- Using no data binding
- Using the xsd:any element in WSDL
- Using a the xsd:anyType in WSDL
- Using an external URI to reference the business document
- Using message attachments in the SOAP message

Types of Parameters & Return Values

- Standard Java data types
  - Automatic handling by JAX-RPC runtime
  - Interoperability is assured
- XML document
  - javax.xml.transform.Source
  - javax.Activation.DataHandler
- SOAP document fragment
  - javax.xml.soap.SOAPElement
- Non-standard types
  - Strongly discouraged



Strategy 1:
XML in the SOAP Body

Document-Style WSDL

- WSDL provides a document-style service contract between sender and receiver
- Abstract Message Description
  - Provides name for each part: PARTNAME
  - Provides type of each message part (e.g., schema for XML parts)
- Binding Description
  - Provides messaging packaging format

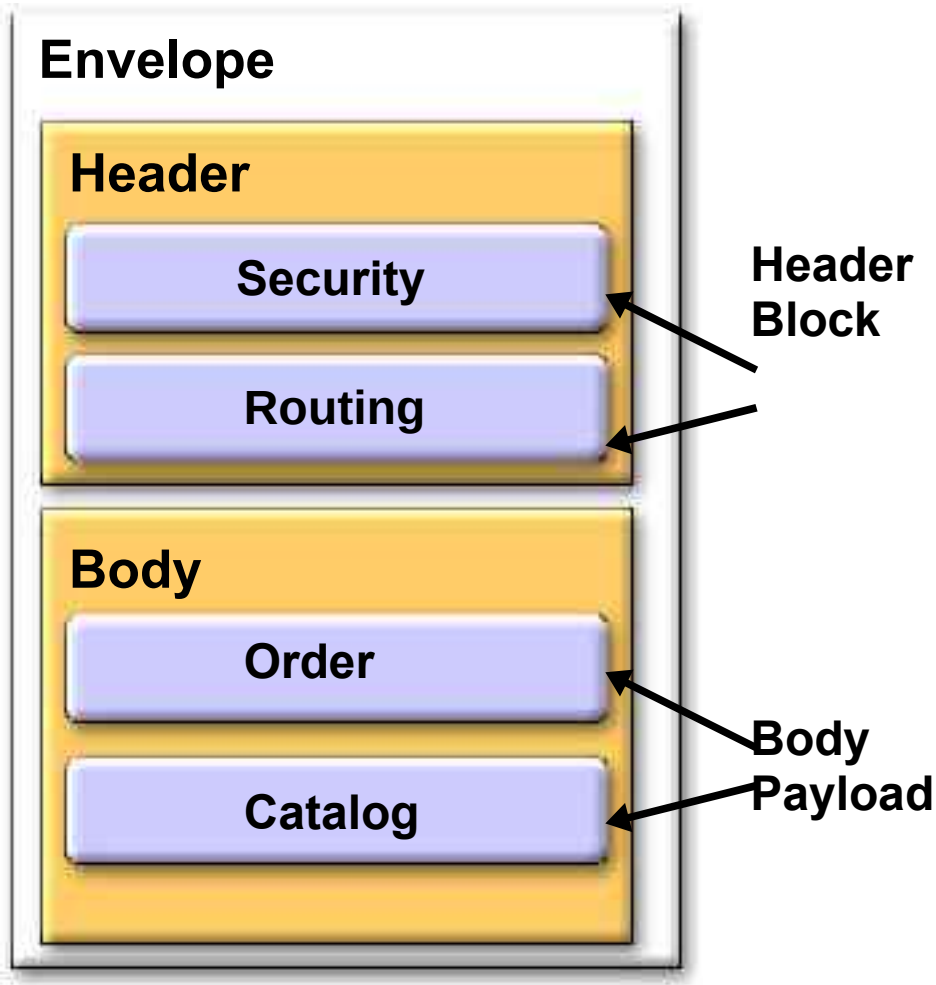
Document-Style SOAP

Header

- Specifies message-level services

Payload

- Opaque
- Schema-defined
- Large
- Complex



Document-Style Web Services

- Supported by JAX-RPC 1.1 via data binding to Java types
- Allowed XML Schema features:
  - Nearly all predefined simple types
  - Simple types derived by restriction
  - List types
  - Attributes
  - Complex types with simple content
  - Complex types with complex content
  - Element wildcards

Direct Document Access

- New in JAX-RPC 1.1
- Map any message part to a DOM Element, then use your favorite XML technology

```
public Element processOrder(Element in)
        throws RemoteException;
```

- Operations are RPC, message content is just an XML document fragment
- Useful in conjunction with JAXB 1.0
- Use “-f nodatabinding” option of wscompile

Strategy#1 XML In SOAP Body

- Top down approach
  - Start with WSDL
  - Separately define abstract, concrete WSDL & XML documents
  - Use tools to generate Java SEI and write implementation
    - E.g wscompile
- Bottom up approach
  - Start with Java code, write SEI and implementation
  - Generate WSDL with tools
  - Pass formatting options to tool, typically document/literal
    - Eg wscompile –f:documentliteral

Strategy#1 XML In SOAP Body

- Pros :
  - Interoperability
  - Schema validation
  - Better performance than rpc/encoded
- Cons:
  - Endpoint received object representation
  - If you want XML you have to reconstruct it

Strategy#1 Top-down Approach

- Sample code
  - <Doc-Literal-From-WSDL> directory
- Code review
  - WSDL document (you write)
  - Service interface (generated)
  - Service implementation template (generated) and code you add
  - Client code (you write)
  - Exchanges SOAP request/responses
- Demo

Steps of Building, Deploying Service

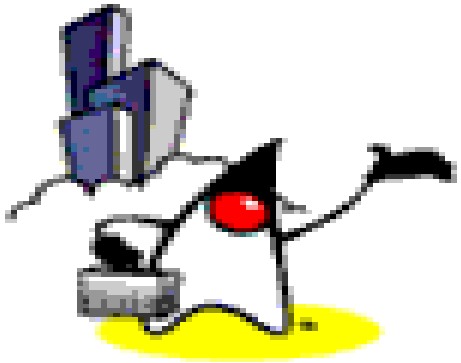
- cd Doc-Literal-From-WSDL
- asant (to get help on the build targets)
- asant create-war
- asant process-war
- asant deploy

Steps of Building and Running Client

- cd Doc-Literal-From-WSDL
- asant run-wsdl-client

Strategy#1 Bottom-up Approach

- Sample code
  - <Doc-Literal-From-Java> directory
- Code review
  - WSDL document
  - Service code
  - Client code
  - Exchanges SOAP request/responses
- Demo



Strategy 2:
Using String in the SOAP Body

Strategy#2 Using String in SOAP Body

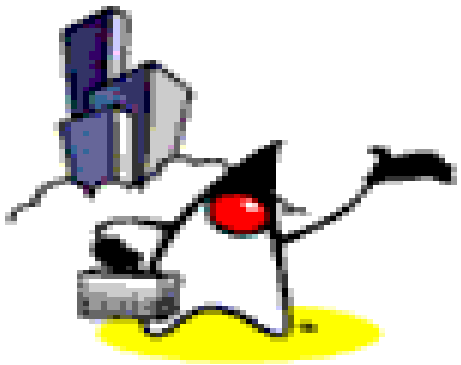
- XML document is passed as a String (xsd:string) between service and service client
- Provides a simple option for passing complex business documents without the need for creating serializers or performing encoding-decoding on the underlying elements

Strategy #2: XML Strings in SOAP

- Pros
  - Simple to develop service
  - Simple to develop clients
- Cons
  - No schema validation: So errors are not detected till everything is in memory
  - Memory intensive: Read entire documents into memory as Strings
  - Loss of business context: What's in the String ?

Strategy#2 Using String in SOAP Body

- Sample code
  - <Doc-Literal-String-XML> directory
- Code review
- Demo



Strategy 3:
Using Base64 or Raw Bytes in SOAP Body

Strategy#3: Using Base64 Or Raw Bytes in SOAP Body

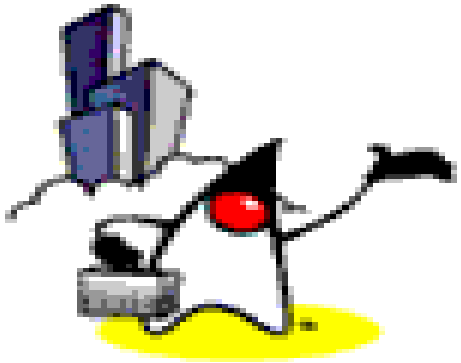
- XML as base64 encoded
  - Same as String, only now encoded with base64 algorithm
  - “A Web service developed in JAX-RPC” encoded into “QSBXZWIgU2VydmljZSBkZXZlbG9wZWQgaW4gSkFYLVJQQw==”.
  - Useful when passing binary data
- XML as raw bytes
  - Defined as xsd:base64binary in WSDL

Strategy#3: Using Base64 Or Raw Bytes in SOAP Body

- Pros :
  - Useful when XML contains characters, declarations not supported by SOAP message info-set or runtime
    - Eg DTD declarations and locale specific character encoding
- Cons
  - Loss of business context – what's the message ?
  - Increased message size, 33% more

Strategy#3: Using Base64 or Raw bytes in SOAP Body

- Sample code
  - <Doc-Literal-String-XML-Base64> directory
- Code review
- Demo



Strategy 4:
Switching off Databinding

Strategy#4.1 Switch Off Data Binding

- Usually endpoint uses standard XML-Java mappings to map XML datatype to Java datatype and vice versa
- A JAX-RPC implementation may be configured to switch off data binding for the literal encoding
  - Eg : wscompile -f:nodatabinding
- Method parameters in generated interfaces bound to javax.xml.soap.SOAPElement

“-f nodatabinding” of wscompile

- All the method parameters are bound to javax.xml.soap.SOAPElement
- Use it when you want to use your own binding framework (e.g. JAXB) instead of that of JAX-RPC

Strategy#4.1 Switch Off Data Binding

- Pros :
  - Useful when application wants to
    - Use a custom binding framework like JAXB
    - Consume the XML representation of the service
- Cons:
  - Runtime specific feature

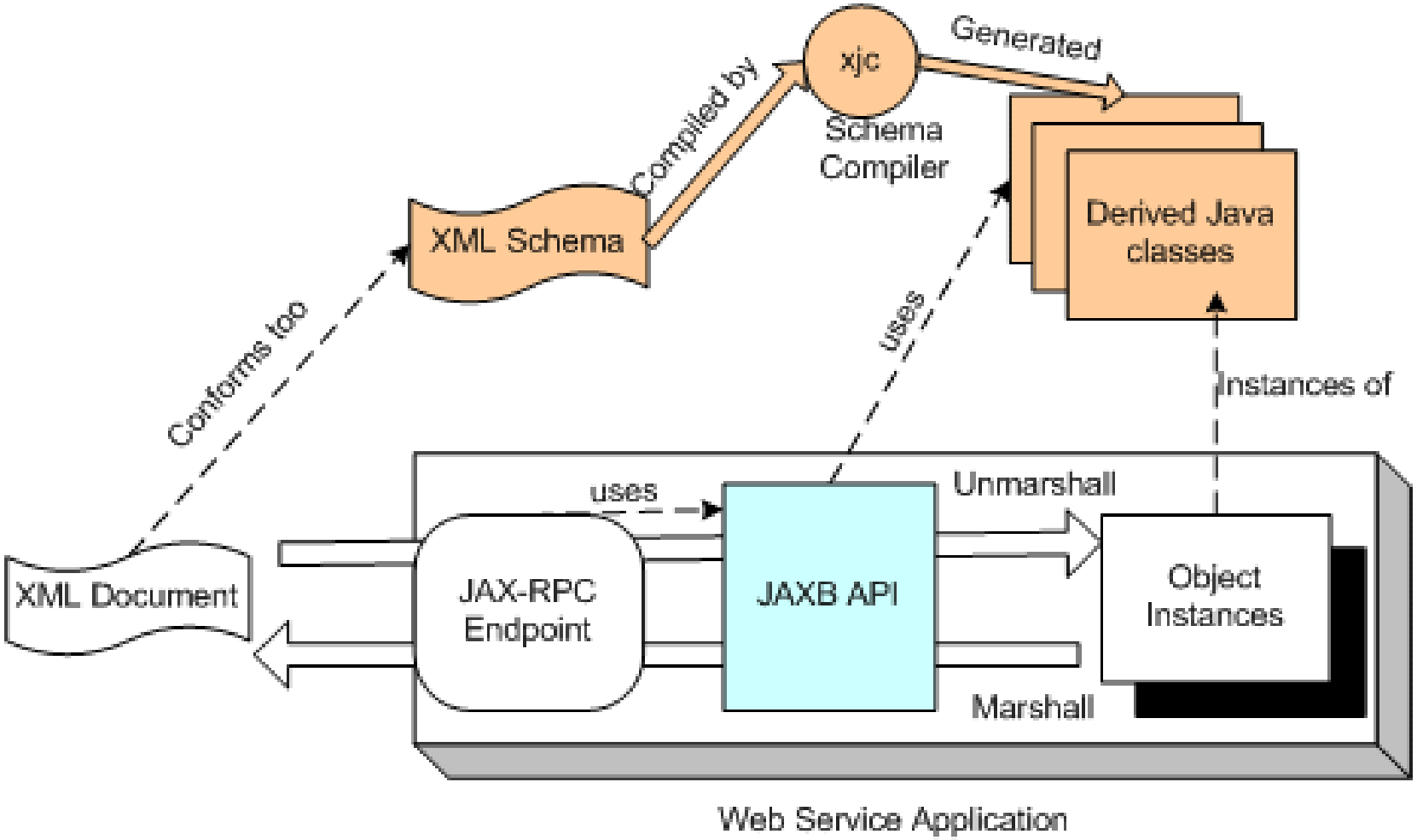
Strategy#4.1: Switch Off Data Binding

- Sample code
  - <Nodatabinding> directory
  - <Nodatabinding-JAXB-Integration> directory
- Code review
- Demo

Strategy#4.2 Integration with JAXB

- Using JAXB as custom binding framework (instead of using built-in databinding from JAX-RPC runtime)

Strategy#4.2 Integration with JAXB

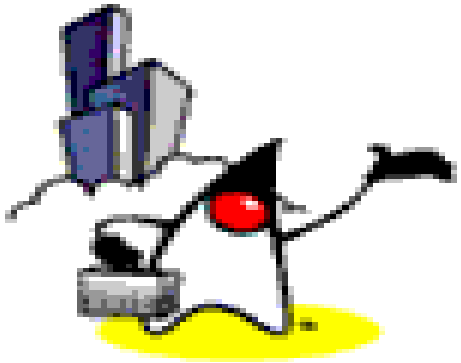


Strategy#4.2 Integration with JAXB

- Pros :
  - Integration with data binding APIs like JAXB that are optimized for parsing the XML documents
- Cons:
  - Runtime specific feature (Implementation dependent)

Strategy#4.2: Integration with JAXB

- Sample code
  - <Nodatabinding-JAXB-Integration> directory
- Code review
- Demo



Strategy 5:
XML

Strategy#5 XML as xsd:any

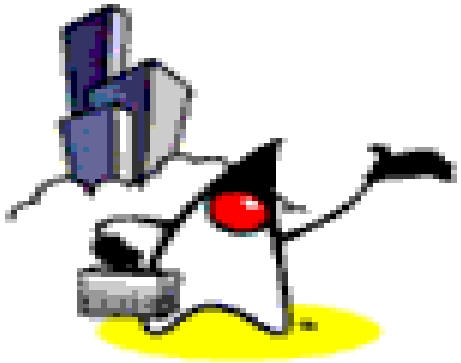
- XML Schema uses xsd:any as wildcard
  - Allows complex types to be extended with elements not specified by the schema
  - Useful when contents of complex type don't need to be defined in schema
- JAX-RPC maps the complex type with xsd:any
  - To usual JavaBean, additional property called "_any" SOAPElement if maxOccurs=1 SOAPElement[] if maxOccurs>1
- Endpoint gets XML document intact

Strategy#5 XML as xsd:any

- Pros
  - Use of a standard mapping
  - Useful for polymorphic processors
  - Can be used with binding frameworks
- Cons
  - You deal with manipulation of SOAPElement
  - Loss of business context, what's in the message ?

Strategy#5: XML as xsd:any

- Sample code
  - <SOAPElement> directory
- Code review
- Demo



Strategy 6:
XML

Strategy#6 XML as xsd:anyType

- XML Schema use xsd:anyType as an abstraction
  - Base type from which all simple & complex types derived
- xsd:anyType type has no restriction,constraints on the data content
- xsd:anyType can be used to pass an XML document “fragments”
- JAX-RPC maps this schema type to a SOAPElement

Strategy#6 XML as xsd:anyType

- Pros
  - Better suited for XML document “fragments”
  - Suited for a strategy where same verb on multiple documents

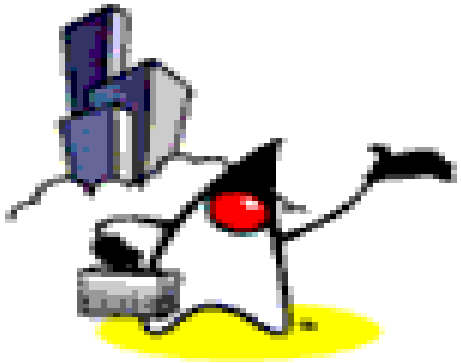
    “Cancel” > Invoice, PurchaseOrder bid etc
- Cons
  - Mapping of anyType is optional in JAX-RPC
  - Since anyType defines the actual data type for a named element in the WSDL

    XML document passed in the SOAP body is located inside this element
    E.g : the PurchaseOrder is inside the BusinessDocumentRequest element.
  - So document being passed now either:

    Have root element identified in the WSDL
    Constructed appropriately or wrapped in the element on the fly

Strategy#6: XML as xsd:anyType

- Sample code
  - <SOAPElement-AnyType> directory
- Code review
- Demo



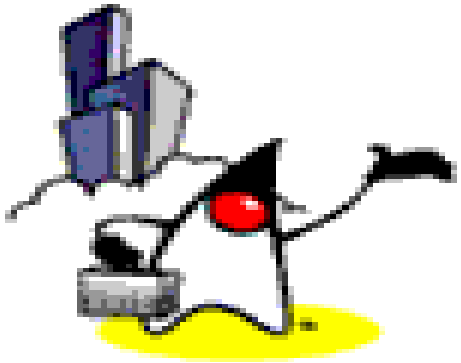
Strategy 7:
Use URI to XML

Strategy#7 Use a URI To XML

- XML Schemas use xsd:anyURI to represent location of XML document
- Mapped to the Java
  - java.net.URI (in J2SE 1.4 only)
  - Usual String
- Service consumer sends SOAP message with a reference to XML document
  - Service provider uses reference to resolve and obtain the business document in a separate call

Strategy#7 Use a URI To XML

- Pros
  - Minimal payload size
  - Data can be dynamically generated when requested
  - Data can be cached in proxy servers
  - Useful for repeated reads
  - Service provider can obtain the document later when it needs it
  - Service provider can use binding API like JAXB directly
- Cons
  - Additional network hops
  - Data may become stale
  - Client needs
    - web server access to serve up documents
    - HTTP Client to download documents
  - Lot of out of band negotiation
  - Many security implications



Strategy 8:
XML as Attachments

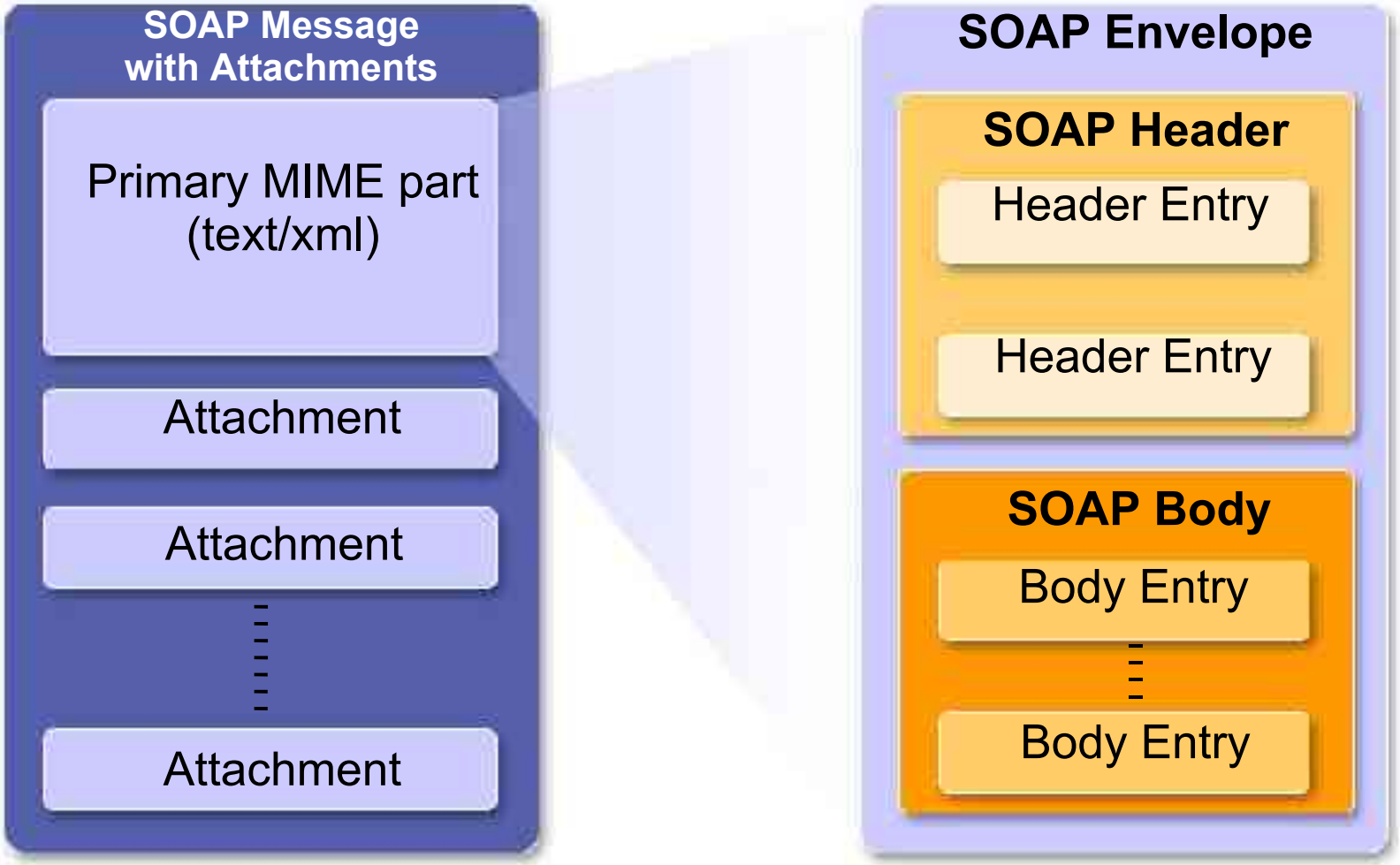
Strategy#8 XML As Attachments

- Send the XML as MIME Attachment to SOAP message
- JAX-RPC uses JavaBeans activation framework
  - Standard mappings defined for common mime types
  - Map others to a javax.activation.DataHandler and vice versa
- WS-I Attachment Profile for interoperability
  - Defines a "swaRef" mechanism for attachment referencing

Strategy#8 XML As Attachments

- Pros
  - Quite efficient, SOAP bodies processed faster
    - Message contains only a reference to the data and not the data itself
    - Reduces the translation time in mapping XML<>Java
  - Useful for documents with prohibited declarations in a SOAP message (DTD)
  - Useful for large documents (can be compressed-decompressed)
  - Good for building additional facilities with handlers
    - Eg PBE on attachment and then GZIP algorithm
- Cons
  - Interoperability- not all vendors support attachments
    - Eg .NET uses DIME though extension pack

SOAP Message with Attachments



SOAP 1.1 With Attachments

- Submitted to W3C for basis of XMLP
  - http://www.w3.org/TR/SOAP-attachments
- Uses MIME "multipart/related" as a container for:
  - SOAP envelope
  - Arbitrary "attachments"
- SOAP envelope and payload can reference "attachments" via relative URLs (href) in the SOAP envelope

Use of Attachments

- Efficient since no marshaling is required
- Used to send both XML document and non-XML contents
- JAX-RPC implementation automatically performs the mapping from certain Java types to Attachments (and vice versa)
- Possible interoperability issues

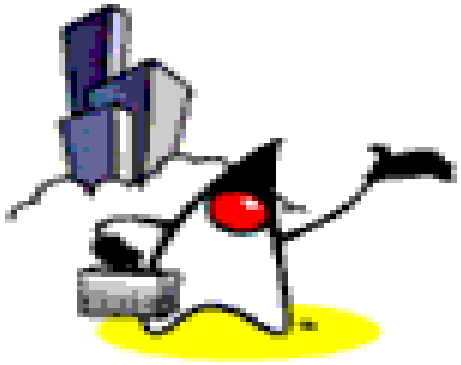
When to Use Attachments

- Non-XML document
- XML document is big and needs to sent as a batch
- XML document's original form needs to be preserved (for legal reason for example)
  - Comments and external references
- XML document uses non-standard schema

Mapping of MIME Types

- image/gif
- image/jpeg
- text/plain
- Multipart/*
- text/xml or application/xml

- java.awt.Image
- java.awt.Image
- java.lang.String
- javax.mail.internet.MimeMultipart
- javax.xml.transform.Source



Document-Style Web Services

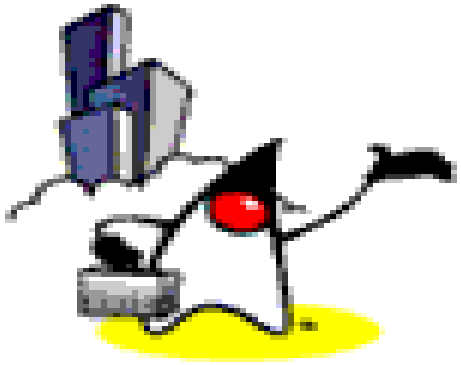
How to send Attachment

3 Different Ways

1. Send javax.xml.transform.Source object as a parameter or return value in JAX-RPC method
2. Send javax.activation.DataHandler object as a parameter in JAX-RPC method
3. Send javax.activation.DataHandler object as an attachment in SAAJ

Current Status & Roadmap of Attachment Support in JAX-RPC 1.3

- Supported only in rpc/encoding mode for now
- Supported only from Java interface type definition (not from WSDL)
- WS-I Attachment profile compliance work is in progress
  - Literal mode support
  - Starting from WSDL document



Document-Style Web Services

Sending Attachment as javax.xml.transform.Source

Example: AttachmentServiceIF.java

```
package attachment;

import java.rmi.Remote;
import java.rmi.RemoteException;
import javax.activation.DataHandler;
import javax.xml.transform.Source;

public interface AttachmentServiceIF extends Remote{

    public String storeDocumentService(DataHandler dh,String filename)
                        throws RemoteException ;

    public String storeDocumentXML(Source source,String filename)
                        throws RemoteException ;
}
```

Example: AttachmentServiceImpl.java

```
public class AttachmentServiceImpl implements AttachmentServiceIF {

   ...

    public String storeDocumentXML(Source source, String xmlfilename) {

        try {
            TransformerFactory factory = TransformerFactory.newInstance();
            Transformer transformer = factory.newTransformer();

            // Print the contents of the file onto Tomcat console
            transformer.transform(source, new StreamResult(System.out));

        } catch (Exception e) {
            System.out.println(e);
        }
        return (":The service received the attachment XML file " + xmlfilename + "!");
    }

}
```

Example: AttachmentServiceClient.java (page 1)

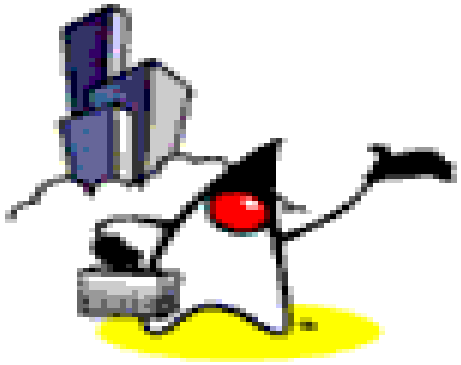
```
public class AttachmentServiceClient {
   public static void main(String[] args) {
      try {
         String filename = "Uploadme.doc";
         String xmlfilename = "bidrequest.xml";
         if (args.length == 2) {
            filename = args[1];
         }

         // Create proxy object
         Stub stub = createProxy();

         // Set endpoint address
         stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
                           args[0]);
```

Example: AttachmentServiceClient.java (page 3)

```
        // Cast the stub to be the correct type
        AttachmentServiceIF astub = (AttachmentServiceIF) stub;

        // Send attachment file as DataHandler object
        sendAttachmentWithDataHandler(astub, filename);

        // Send attachment file as Source object
        sendAttachmentWithSource(astub, xmlfilename);

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

private static Stub createProxy() {
    // Note: Attachment_Impl is implementation-specific.
    return (Stub)(new MyAttachmentService_Impl().getAttachmentServiceIFPort());
}
```

Example: AttachmentServiceClient.java (page 2)

```
    private static void sendAttachmentWithDataHandler(AttachmentServiceIF stub,
                                        String filename) throws Exception {
        DataHandler dh = new DataHandler(new FileDataSource(filename));
        String response = stub.storeDocumentService(dh, filename);
        System.out.println("Response from server " + response);
    }

    private static void sendAttachmentWithSource(AttachmentServiceIF stub,
                                        String xmlfilename) throws Exception{
        Source src= new StreamSource(new File(xmlfilename));
        String response = stub.storeDocumentXML(src, xmlfilename);
        System.out.println("Response from server " + response);
    }

}
```



Document-Style Web Services

Sending Attachment as javax.activation.DataHandler

Example: AttachmentServiceIF.java

```
package attachment;

import java.rmi.Remote;
import java.rmi.RemoteException;
import javax.activation.DataHandler;
import javax.xml.transform.Source;

public interface AttachmentServiceIF extends Remote{

    public String storeDocumentService(DataHandler dh,String filename)
                        throws RemoteException ;

    public String storeDocumentXML(Source source,String filename)
                        throws RemoteException ;
}
```

Example: AttachmentServiceImpl.java

```
public class AttachmentServiceImpl implements AttachmentServiceIF {
   public String storeDocumentService(DataHandler dh, String filename) {
      try {
         System.out.println(" >>>> storeDocumentService storing file to  "
                           + "/tmp/" + filename);
         BufferedOutputStream out
           = new BufferedOutputStream(new FileOutputStream("/tmp/" + filename));
         BufferedInputStream in
           = new BufferedInputStream(dh.getInputStream());

         byte[] buffer = new byte[256];
         while (true) {
            int bytesRead = in.read(buffer);
            if (bytesRead == -1)
               break;
            out.write(buffer, 0, bytesRead);
         }
         in.close();
         out.close();
      } catch (Exception e) {
         System.out.println(e);
         return e.toString();
      }
      return (":The service saved the attachment file " + filename + "in /tmp directory!");
   }
   ...
}
```

Example: AttachmentServiceClient.java (page 1)

```
public class AttachmentServiceClient {
   public static void main(String[] args) {
      try {
         String filename = "Uploadme.doc";
         String xmlfilename = "bidrequest.xml";
         if (args.length == 2) {
            filename = args[1];
         }

         // Create proxy object
         Stub stub = createProxy();

         // Set endpoint address
         stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
                           args[0]);
```

Example: AttachmentServiceClient.java (page 3)

```
        // Cast the stub to be the correct type
        AttachmentServiceIF astub = (AttachmentServiceIF) stub;

        // Send attachment file as DataHandler object
        sendAttachmentWithDataHandler(astub, filename);

        // Send attachment file as Source object
        sendAttachmentWithSource(astub, xmlfilename);

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

private static Stub createProxy() {
    // Note: Attachment_Impl is implementation-specific.
    return (Stub)(new MyAttachmentService_Impl().getAttachmentServiceIFPort());
}
```

Example: AttachmentServiceClient.java (page 2)

```
    private static void sendAttachmentWithDataHandler(AttachmentServiceIF stub,
                                        String filename) throws Exception {
        DataHandler dh = new DataHandler(new FileDataSource(filename));
        String response = stub.storeDocumentService(dh, filename);
        System.out.println("Response from server " + response);
    }

    private static void sendAttachmentWithSource(AttachmentServiceIF stub,
                                        String xmlfilename) throws Exception{
        Source src= new StreamSource(new File(xmlfilename));
        String response = stub.storeDocumentXML(src, xmlfilename);
        System.out.println("Response from server " + response);
    }

}
```

Example: WSDL of AttachmentService (page1)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="urn:Foo"
    xmlns:ns2="http://java.sun.com/jax-rpc-ri/internal"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="MyAttachmentService"
    targetNamespace="urn:Foo">
 <types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://java.sun.com/jax-rpc-ri/internal" xmlns:soap11-
    enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://java.sun.com/jax-rpc-ri/internal">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>

    <simpleType name="datahandler">
      <restriction base="base64Binary"/></simpleType>
    <simpleType name="text_xml">
      <restriction base="string"/></simpleType></schema></types>
```

Example: WSDL of AttachmentService (page2)

```
<message name="AttachmentServiceIF_storeDocumentService">
  <part name="DataHandler_1" type="ns2:datahandler"/>
  <part name="String_2" type="xsd:string"/></message>
<message name="AttachmentServiceIF_storeDocumentServiceResponse">
  <part name="result" type="xsd:string"/></message>

<message name="AttachmentServiceIF_storeDocumentXML">
  <part name="Source_1" type="ns2:text_xml"/>
  <part name="String_2" type="xsd:string"/></message>
<message name="AttachmentServiceIF_storeDocumentXMLResponse">
  <part name="result" type="xsd:string"/></message>

<portType name="AttachmentServiceIF">
  <operation name="storeDocumentService" parameterOrder="DataHandler_1
    String_2">
    <input message="tns:AttachmentServiceIF_storeDocumentService"/>
    <output
    message="tns:AttachmentServiceIF_storeDocumentServiceResponse"/></operation>
  <operation name="storeDocumentXML" parameterOrder="Source_1 String_2">
    <input message="tns:AttachmentServiceIF_storeDocumentXML"/>
    <output
    message="tns:AttachmentServiceIF_storeDocumentXMLResponse"/></operation>
</portType>
```

Example: WSDL of AttachmentService (page2)

```
<binding name="AttachmentServiceIFBinding" type="tns:AttachmentServiceIF">
  <operation name="storeDocumentService">
    <input>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo"/></input>
    <output>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo"/></output>
    <soap:operation soapAction=""/></operation>
  <operation name="storeDocumentXML">
    <input>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo"/></input>
    <output>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="urn:Foo"/></output>
    <soap:operation soapAction=""/></operation>
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="rpc"/></binding>
<service name="MyAttachmentService">
  <port name="AttachmentServiceIFPort" binding="tns:AttachmentServiceIFBinding">
    <soap:address xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    location="http://localhost:8080/attachment/attachment"/></port></service></definitions>
```

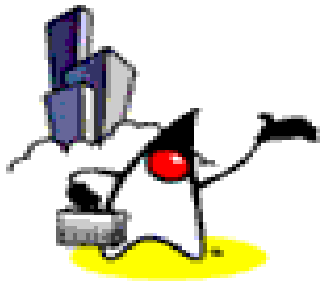
Example: SOAP Request Message (Source)

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:n="urn:Foo"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns="http://java.sun.com/jax-rpc-ri/internal"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <n:storeDocumentXML>
      <Source_1 xsi:type="tns:text_xml"></Source_1>
      <String_2 xsi:type="xs:string"></String_2>
    </n:storeDocumentXML>
  </soap:Body>
</soap:Envelope>
```

Example: SOAP Request Message (DataHandler)

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:n="urn:Foo"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns="http://java.sun.com/jax-rpc-ri/internal"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <n:storeDocumentService>
      <DataHandler_1 xsi:type="tns:datahandler"></DataHandler_1>
      <String_2 xsi:type="xs:string"></String_2>
    </n:storeDocumentService>
  </soap:Body>
</soap:Envelope>
```

Resources



Resources

- JAX-RPC Home
  - http://java.sun.com/xml/jax-rpc/index.html
- Java Web Services Developer Pack Download
  - http://java.sun.com/webservices/downloads/webservicespack.html
- Java Web Services Developer Pack Tutorial
  - http://java.sun.com/webservices/downloads/webservicetutorial.html
- J2EE 1.4 SDK
  - http://java.sun.com/j2ee/1.4/download-dr.html

Passion!