Outline

- The issue of service styles
  - Background rpc/enc vs. doc/lit
  - WS-I Basic Profile

- Developing services with Axis
  - Starting from Java code
  - Modifying WSDL for better structures
  - Implementing and deploying server code

SOAP Formats

- Three commonly-used formats:
  - RPC encoded (SOAP encoding) – defined by original SOAP specification
  - Document literal – XML message with format determined by schema
  - Wrapped – document literal variation, with call parameters as children of root element
- Service defines the format to use (WSDL)

WSDL differences

- rpc/enc vs. doc/lit in WSDL:
  - rpc/enc can use type for message parts
    - Defined in schema, or basic types
  - doc/lit uses schema elements as message parts
    - Specified in the bindings section of WSDL
- What about other combinations?
  - rpc/lit like doc/lit, but message parts use types
  - doc/enc is not a valid combination
**SOAP encoding**

- SOAP encoding a primitive form of data binding:
  - Types based on schema types
  - “Automatic” conversions to native types for language
  - Cross-language and cross-platform
- But limited...

**Encoding issues**

- Data types not cross-language
- Complex structures even more of a problem:
  - Constructs such as HashMap have no standard form
  - Custom serializers / deserializers needed everywhere
- Incompatible quirks in encoding
  - Use xsi:type or not?
  - multiRef handling
- Creates interoperability and extensibility problems
WS-I

- “Web Services Interoperability Organization”
  - Original primary backers IBM, Microsoft, BEA, etc.
  - Selecting options rather than writing standards

- Basic Profile a first goal:
  - Decide on “best practices”
  - Reduce choices to promote interoperability

- Basic Profile 1.0a final as of August, 2003
  - Updated to 1.1 August, 2004
  - SOAP Binding Profile and Attachments Profile 1.0

WS-I Basic Profile

- Based on many standards, including:
  - Simple Object Access Protocol (SOAP) 1.1
  - Extensible Markup Language (XML) 1.0
  - RFC2616: Hypertext Transfer Protocol -- HTTP/1.1
  - RFC2965: HTTP State Management Mechanism
  - Web Services Description Language (WSDL) 1.1
  - XML Schema Part 1: Structures
  - XML Schema Part 2: Datatypes
  - etc.

WS-I Basic Profile

- Key transport constraints:
  - Mandatory support of HTTP binding for SOAP
    - HTTP 1.1 recommended, but HTTP 1.0 allowed
  - Requires the use of the HTTP POST method
  - Standardizes HTTP response code interpretation
  - TLS 1.0 or SSL 3.0 (HTTPS) security
    - Service may require HTTPS
    - Service may require mutual authentication

- Key XML constraints:
  - Prohibits XML DTDs and PIs
  - Requires UTF-8 or UTF-16

- Key WSDL constraints:
  - Requires the use of WSDL 1.1
  - Clarifies:
    - `<wsdl:import>` restricted to import from other WSDL
    - Order must be `[<wsdl:import>], <wsdl:types>`, then rest
    - Basically looking toward WSDL 2.0
WS-I Basic Profile

- SOAP constraints:
  - Clarifies Fault usage and syntax
  - Requires namespace-qualified Body child elements
  - And the biggie - prohibits encodingStyle
- Basic change in direction of Web services:
  - rpc/encoded forbidden
  - rpc/literal allowed (but not currently much used)
  - doc/lit (and wrapped variation) the clear winners

What's the effect?

- Implementations can still use an encoding
  - Automatically convert XML to and from structures
  - Just don't rely on other end interpreting XML the same way, and build a schema
- Schema defines XML document payload
  - XML-centric rather than code-centric
- Up to the application to work with the XML
  - Directly, via document model or such
  - As data, using some form of data binding

Installing Axis server

- Installing Axis servlet under Tomcat
  - Copy Axis *webapps/axis* directory to Tomcat *webapps/axis*
  - Start Tomcat
  - Browse to http://localhost:8080/axis
  - Check Validation link
  - Check List link

Axis usage

- Can start from Java code, using default conversions defined by JAX-RPC / Axis
- Always use doc/lit or wrapped/lit modes
- Can modify generated schema as part of WSDL to:
  - Improve interoperability
  - Correct mistaken assumptions
  - Improve schema structuring
Building Axis services

- Basic recommended approach (part 1):
  - Start from template version of Java code
  - Define the service API in a Java class
  - Include any data structures used in API
  - Doesn't need to actually work, as long as it compiles
  - Run Axis Java2WSDL tool to generate WSDL
  - Converts to XML representation using JAX-RPC rules
  - Best to use wrapped form of doc/lit
  - Let's take a look...

Person service template

```java
public class Person {
    public PersonBean getPerson(int index) throws NoPersonException {
        ...        return s_beans[index];
    }
    public PersonBean[] getAllPersons() {
        return s_beans;
    }
    public PersonBean findPerson(String fname, String lname, Date born)
        throws NoPersonException {
        ...    }
}
```

Person data template

```java
public class PersonBean {
    private int m_index;
    private String m_fName;
    private String m_lName;
    private Date m_born;
    public PersonBean() {
    }
    public PersonBean(int index, String fname, String lname, Date born) {
        ...        m_index = index;
        m_fName = fname;
        m_lName = lname;
        m_born = born;
    }
    public int getIndex() {
    }
    public String getFName() {
    }
    public ... m_lName;
    public Date getBorn() {
    }
    public void setIndex(int index) {
        m_index = index;
    }
}
```

Person exception template

```java
public class NoPersonException extends Exception {
    private String m_addedInfo;
    public NoPersonException(String ...
        return m_addedInfo;
    }
    public void setAddedInfo(String addedInfo) {
        m_addedInfo = addedInfo;
    }
}
```
WSDL from Java code

- Person sample service demonstration
  - Supplied template code defines the service
  - Supplied build.xml takes care of (most) work
    - compile-template compiles the template code
    - generate-wsdl generates WSDL from template
    - from-java does all the above
    - modify-wsdl changes schema type used for date
    - build-server compiles server code to Axis installation
    - deploy deploys the service to running Tomcat/Axios
    - undeploy removes service from running Tomcat/Axios

How it works

- Template code in template/src directory
- Compiled to template/bin directory
- Code generation to gen/src directory
- Copied from there to server/gen and client/gen directories
- Server implementation code in server/impl
- All server code compiled to server/bin
- Copied to Axis installation under Tomcat

Exercise 1

- Run the same demonstration on your system
  - Install Axis under Tomcat
  - Run WSDL generation with from-java target
  - Run modify-wsdl target to correct schema
  - Run from-wsdl to generate Java code and compile
  - Start Tomcat
  - Run deploy target to activate service
  - Run run to verify proper operation

Axis service details

- WSDL2Java tool generates client only, or both client and server
  - No option for server-only generation
  - Must separate out client code from server code
  - Done by build.xml in example
Axis classes

- Client-only classes are `{service-name}Locator.java`, `{binding-name}Stub.java`, and `{service-name}Service.java
- Server-only classes is `{binding-name}Impl.java` – but this should be replaced by your own implementation code
- Remaining (data) classes and `{portType-name}.java` are shared
- `deploy.wsdd` and `undeploy.wsdd` are only required for server code deployment

Building Axis services

- Generated code may not match template
  - Some types are changed (java.util.Calendar in place of java.util.Date, for instance)
  - Data class constructors use alphabetical parameters
  - Arrays in place of collections
- Can minimize changes if you build template with these issues in mind

Axis standard deployment

- Done by `build.xml` for example
  - Copy code to `axis/WEB-INF/classes` directory tree
  - Use Axis admin tool to deploy to running server
  - Merges service information into `axis/WEB-INF/server-config.wsdd` configuration

Modifying server code

- Changing server interface requires undeploy:
  ```xml
  <!-- Undeploy service in Axis, which must be running -->
  <target name="undeploy">
    <admin port="8080" hostname="localhost" servletpath="axis/services/AdminService"
      xmlfile="generate/src/${package-path}/undeploy.wsdd"/>
  </target>
  
  Can then update code and redeploy
  Changing code generally requires stopping and restarting Tomcat
```
Exercise 2

• Now modify to build Seismic service:
  – Change “person” name to “seismic” name (including package for Test.java)
  – Substitute supplied service template code
  – Use `generate-wsdl` to generate the WSDL
  – Build code and merge with client code from yesterday
  – Once that's running, modify the schema to match yesterday

Alternative Axis deployment

• Build complete service as web application
  – Merge `deploy.wsdd` directly into Axis `server-config.wsdd`
    • Only problem is you need to generate one first
    • Axis application creates on first use
  – Rest as in standard deploy
    • Copy your classes in to `axis/WEB-INF/classes`
    • Copy your library jars in to `axis/WEB-INF/lib`
  – Then package it up as a war file