Outline

- From XML-RPC to SOAP + WSDL
  - Web services background and compatibility
  - SOAP and WSDL basics
- Introduction to Axis
  - JAX-RPC background
  - Developing SOAP clients using Axis
  - TCPMonitor

Historical background

- Many different ways to expose applications:
  - Direct socket connection (custom data formats)
  - Remote Procedure Call (RPC)
  - CORBA cross-language calls (or DCOM, or RMI)
- All have drawbacks for interoperability
  - Especially in terms of coupling
- XML seems ideal as a way around problems
  - Cross-platform and cross-language
  - Wide variety of tools available

XML web services

- Mostly use HTTP because of firewalls
- Exposing applications using XML:
  - Direct XML exchange via HTTP
  - SOAP wrapper with WSDL description
  - HTTP GET queries with XML response (REST)
- All are seeing common use
**Direct XML exchange**

- Typically uses HTTP transport
- POST operation for two-way exchange:
  - Request XML document as POST data
  - Response XML document returned directly
  - Both directions use customized XML formats
    - Industry- or company-specific
- Requires custom handling (security, etc.)
- Common approach for 6+ years

**Direct XML example**

- Aircraft parts catalog viewer application
  - Equivalent of two bookshelves full of manuals (for each individual plane)
  - “Smart” Java applet-based browser client
  - Web server software interfaces to database
  - XML data sent in both directions
  - Images referenced in XML, not embedded
  - Developed in 1999

**SOAP/WSDL XML**

- SOAP defines wrapper for content
  - Both wrapper (envelope) and content are XML
  - Envelope allows for added-value functions
    - Security, routing, transactional support, etc.
- WSDL gives standard service descriptions
  - Operations, data structures, addresses, etc.
  - Allows automatic client code generation
- Widespread use for last 4+ years (.Net, etc.)

**SOAP Example**

- Amazon SOAP web service interface:
  - Provides SOAP services for full range of services
  - WSDL definition allows easy client generation
    - Defines set of operations supported by service
    - Defines XML format for all request and responses
  - Advantage to developers is ease of use
    - Frameworks generate client interface from WSDL
    - Services exposed to application as method calls
REST XML

- **RE**presentational State **T**ransfer
  - Way of looking at the Web as resources (2000)
    - GET to retrieve the representation of a resource, DELETE to remove, POST/PUT for updating or creating
  - Can also be applied to HTTP XML web services
  - More hand development than SOAP/WSDL
    - But simpler resulting code – no framework needed
- Requires custom handling (security, etc.)
- Specialized use over last 3+ years

REST example

- Amazon REST web service interface:
  - Provides XML output for full range of services
  - Request very similar to web browser requests
  - Advantage to developers is simplicity
    - No SOAP framework is required
  - Currently gets more use than SOAP version
    - But not necessarily a typical use case...

Focus for this workshop

- Focus on SOAP/WSDL services
  - Widespread industry backing (MS, IBM, Sun, ...)
  - Client code generation makes for easy use
  - Supports (relatively) easy interface versioning
  - Other protocols being layered on top of SOAP
- Most widely used approach at present

RPC with XML

- XML-RPC the simple form of XML web services
  - Remote calls with XML encoding
  - Less efficient than binary, but much more convenient
- Developed by Dave Winer of Userland
  - In cooperation with Microsoft (SOAP)
  - Publicly released while SOAP still in development
- Largely “self-describing”
XML-RPC structure

```xml
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>
```

Why not use XML-RPC?

- Limited expressiveness:
  - Only a few data types
  - Handles nested structure trees, but not graphs
- Not very “XMLish”
  - Rigid format doesn't allow for general XML
  - In-band type information bulky and redundant
- Microsoft wanted more...

SOAP basics

- SOAP a more-complex version of this:
  - More sophisticated encoding scheme
  - Namespaces to keep components clear
- Microsoft-sponsored effort for 1.0
- IBM joined in promoting as standard in 1.1
- Basis for .NET marketing campaign
- What most people mean by “web services”

What is SOAP?

- Defines wrapper for application data
- Actual data format can vary:
  - Specification defines one encoding (mapping between XML and application objects)
  - Applications can define own encodings
  - Increasingly common to just use XML directly
- Actual connection can vary:
  - Usually request-response pattern
  - Usually HTTP transport (but can be anything)
SOAP Message Structure

- **Envelope**: a wrapper for content, but no useful information
- **Optional header**: can contain control information
- **Body**: contains actual data in XML form
- **Attachments**: can hold other types of data (binary, unencoded text, etc.)

The WSDL Additive

- **Web Services Description Language**
- Based on abstract definitions and bindings:
  - **types**: types defined for use in messages
  - **messages**: the data being exchanged
  - **port types**: collections of operations
  - **binding**: concrete protocol and data format
  - **service**: a collection of bound ports at address
- Gives a comprehensive definition of service

Types

- Types section optionally defines structures for exchange (with embedded schema definitions)

```xml
  <wsdl:types>
    <schema ...>
      ...</schema>
  </wsdl:types>
</wsdl:definitions>
```
Messages

- Each message represents one interaction
- One or more “parts” in message
- Describes the content to be sent in an operation
- Can make use of type definitions (or schema types)

```xml
<wsdl:message name="getAllPersonsResponse">
  <wsdl:part element="impl:getAllPersonsResponse" name="parameters"/>
</wsdl:message>

<wsdl:message name="getPersonResponse">
  <wsdl:part element="impl:getPersonResponse" name="parameters"/>
</wsdl:message>
```

Port types

- Abstract operation definitions
- Message(s) involved in each operation

```xml
<wsdl:portType name="Person">
  <wsdl:operation name="getPerson">
    <wsdl:input message="impl:getPersonRequest" name="getPersonRequest"/>
    <wsdl:output message="impl:getPersonResponse" name="getPersonResponse"/>
    <wsdl:fault message="impl:NoPersonException" name="NoPersonException"/>
  </wsdl:operation>
  ...
</wsdl:portType>
```

Bindings

- Protocol-specific binding data for port type
- SOAP communication style, usage, transport

```xml
<wsdl:binding name="personSoapBinding" type="impl:Person">
  <wsdlsoap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getPerson">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getPersonRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    ...
  </wsdl:operation>
</wsdl:binding>
```

Service and port definition

- Service gives name for a set of ports
- Each port definition specifies a single address for a binding
- Can be multiple ports, multiple addresses

```xml
<wsdl:service name="PersonService">
  <wsdl:port binding="impl:personSoapBinding" name="person">
    <wsdlsoap:address locations="http://localhost:8080/axis/services/person"/>
  </wsdl:port>
</wsdl:service>
```
SOAP Body

- Message payload container
- Three commonly-used formats:
  - RPC encoded – defined by 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)

RPC encoded SOAP

- **Remote Procedure Call** – like CORBA or RMI
  - Call parameters and result encoded in body
    - encodingStyle defines the rules for conversions
    - Messages may include xsi:type information, or not
  - Encoding allows for graphs of objects (multiRefs)
- Optionally used by .NET – but not in future...
- Usually abbreviated rpc/enc

Document literal

- An XML document is passed each way
- Flexible, but less automatic for code:
  - Working with XML documents (but may still map to method calls behind the scenes)
  - Requires code at each end to convert to data
- Equivalent to direct XML exchange embedded in SOAP
- Usually abbreviated doc/lit

Wrapped

- Microsoft approach for easy service interface:
  - Given a call with parameters "a", "b" and "c"
    - Define a top-level element with children "a", "b" and "c"
    - Pass the element as document literal SOAP body
    - Uses Microsoft encoding scheme
  - Preferred .NET approach
- Represented as doc/lit
- Limited by the encoding
**rpc/enc sample**

```xml
<soapenv:Body>
  <ns1:handleQuery
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns1="urn:axis.sosnoski.com">
    <in0 href="#id0"/>
  </ns1:handleQuery>
</soapenv:Body>
```

**doc/lit sample**

```xml
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
      <qk:min-date>2001-05-04T02:31:00.601</qk:min-date>
      <qk:max-date>2001-07-02T04:33:22.61</qk:max-date>
      <qk:min-long>-91.997185</qk:min-long>
      <qk:max-long>78.09824</qk:max-long>
      <qk:min-lat>-48.434654</qk:min-lat>
      <qk:max-lat>152.46613</qk:max-lat>
    </qk:search-params>
  </SOAP:Body>
</SOAP:Envelope>
```

**rpc/enc issues**

- Like database schema generation from code
  - Automatic conversion does not allow tuning
  - Some code structures can't be used (hashmaps, etc.)
  - Okay for small, standalone applications
  - Distributed applications need cleaner structure
- doc/lit gives better control
  - doc/lit schemas XML equivalent of DBA
  - Can use cleaner and more compact XML structure

**SOAP in Java**

- Java API for XML-based RPC
  - “Portable and interoperable” web services in Java
  - Makes web services look like RMI
  - Converts to and from Java objects (limited)
- Now available as 1.1.X release (JWSDP 1.X)
  - Unfortunately very difficult to use
Apache Axis SOAP

- Successor to Apache SOAP (itself originally based on IBM SOAP4J):
- Tools for WSDL to Java and Java to WSDL generation
- Automatic WSDL generation for deployed services
- JAX-RPC compatibility
- Currently at 1.2.1, 1.3 in progress

Building Axis clients

- Person sample service demonstration
  - Supplied person.wsdl defines the service
  - Supplied build.xml takes care of (most) work
    - clean deletes generated code and class files
    - generate-java generates and organizes code
    - build-client compiles client code
    - from-wsdl does all the above
    - run runs the client with test data
    - tcpmon runs TCPMonitor program
    - run-monitored runs client using TCPMonitor

How it works

- Code generation to client/gen directory
- Supplied Test.java program in client/impl tree
- Compile first generated classes, then supplied code, to client/bin
- Run directly from there

Exercise 1a

- Run the same demonstration on your system
  - Build the client
  - Run without TCPMonitor
  - Start TCPMonitor (in separate console window)
  - Run with TCPMonitor
- Now modify this to work with supplied seismic.wsdl
Seismic service

- Web service for retrieving seismic events
  - Get count of events in database
  - Retrieve by date, latitude, and longitude ranges
  - Retrieve by “similarity” to quake of interest

Exercise 1b

- Now modify to build client for `seismic.wsdl`
  - Copy the whole directory first!
  - Change “person” name to “seismic” name (including package for `Test.java`)
  - Try the initial `generate-java` step
  - Compare generated classes to those from example
  - Modify the `Test.java` code to use the appropriate class, and implement the `getQuakeCount()` and `findQuakes()` operations

Hints

- The query parameters used in `test` should return 3 QuakeSets, containing 9 total quakes
- Java code for parameters:

```java
TimeZone utc = TimeZone.getTimeZone("UTC");
SimpleDateFormat format =
    new SimpleDateFormat("yyyy-MM-dd'/'HH:mm:ss");
Calendar mndate = new GregorianCalendar(utc);
mndate.setTime(format.parse(args[3]));
...
Float mmlat = new Float(args[5]);
...```

Exercise 2

- Now try doing the same thing for the Amazon Web Service `aws.wsdl`
  - You'll need to register with Amazon for an id to actually use this
  - Check details at http://www.amazon.com/webservices
  - Will demonstrate in workshop, if time available