Web Services... Where Do You Start?
SOAP, WSDL and UDDI

Denise Hatzidakis
Perficient, Inc.
denise.hatzidakis@perficient.com
Web Services Standards

Evolution

Evolving to resolve pain ...

Phase I
“Connection”

XML Schema
SOAP
WSDL
UDDI

WS-I Basic Profile

Phase II
“Security and Reliability”

XML Digital Signature
XML Encryption
WS-Security
SAML
WS-Policy
WS-ReliableMessaging

Phase III
“Enterprise”

Provisioning Transactions
Workflow/BPEL
Systems management ...

Evolving to resolve pain ...
What Is a Web Service?

- Specifically – Web Services are a stack of emerging standards that describe a service-oriented, component-based application architecture.
- Conceptually – Web Services represent a model in which discrete tasks within an e-business processes are widely distributed.

"...loosely coupled software components that interact with one another via standard technologies."
- Gartner Group

"...loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols.
- Stencil Group

".. a broad based agreement for exposing programmatic behavior over a network and a set of core technologies that enable that capacity."
- Noel Bergman (CSS 2002)
What Is a Service Oriented Architecture?

- Service-oriented architectures are **distributed**.
  - Functional elements of the application are deployed on multiple systems and execute across local and even remote networks.
  - In particular, Web services make use of existing, ubiquitous transport protocols like HTTP. By piggybacking on the same, well understood transport as Web content, Web services leverage existing infrastructure and can comply with basic firewall policies.

- The systems are characterized by **loosely coupled** interfaces.
  - Traditional application design depends upon a tight interconnection of all subsidiary elements.
  - The complexity of these connections requires that developers thoroughly understand and have control over both ends of the connection; moreover, once established, it is exceedingly difficult to extract one element and replace it with another.
  - Loosely coupled systems, on the other hand, require a much simpler level of coordination and allow for more flexible reconfiguration.
What Is a Service Oriented Architecture? *(Continued)*

- Systems are conceived from a **process-centric** perspective.
  - By intent, services are designed with a task-orientation; they function as discrete steps in a larger workflow or business process.
  - A well designed service describes its inputs and outputs in a way that other software can determine what it does, how to invoke its functionality, and what result to expect in return.

- The connections are based upon **vendor-independent standards**.
  - The development of generally open and accepted standards is a key strength of the coalitions that have been building Web services infrastructure.
  - Most previous efforts at distributed computing (*e.g.*, CORBA, DCOM, RMI, and others) were very difficult to implement, because they were dependent upon a particular vendor's product offering, highly specified, or programmatically complex—usually all of the above.
Web Services Architecture

To support the use of Web Services in e-business, IBM, Microsoft, Sun, BEA and others are working to create a concrete Web Services 'stack' that defines how to construct Web Services based solutions.

- An instance of a service oriented architecture
- As defined by the W3C Web Services Architecture Working Group
  - Web Services Architecture Working draft
  - [http://www.w3.org/TR/ws-arch/](http://www.w3.org/TR/ws-arch/)
What Is the Web Services Stack?

The **Web Services Stack** is an emerging architectural model and set of standards for developing and deploying software applications.

- Conceptually, **service-oriented architectures** (SOAs) represent a model in which small, loosely coupled pieces of application functionality are published, consumed, and combined with other applications over a network.
- Specifically, **Web services** are a stack of emerging standards that define protocols and create a loosely coupled framework for programmatic communication among disparate systems.
What Is the Web Services Stack?

- A universal client/server architecture that **allows disparate systems to communicate** with each other without using proprietary libraries.
- The architecture simplifies the process typically associated with client/server applications by **effectively eliminating code dependencies between the client and server**.
Web Services Stack – Key Components

- Web Services are a set of protocols based on XML (Extensible Markup Language).
- Base protocols that formed the initial specification for Web Services.
  - **Simple Object Access Protocol (SOAP)** – defines the runtime message that contains the service request and response. SOAP is independent of any particular transport and implementation technology.
  - **Web Services Description Language (WSDL)** – describes a Web Service and the SOAP Message. It provides a programmatic way to describe what a service does, paving the way for automation.
  - **Universal Discovery, Description, Integration (UDDI)** – UDDI is a cross industry initiative to create a standard for service discovery together with a registry facility that facilitates the publishing and discovery processes.
SOAP, SOAP and More SOAP

- The term is used several ways
  - **Soap Protocol**
    - w3c standard (http://www.w3.org/TR/SOAP/)
  - **Apache Soap project**
    - Open source Web Services engine (http://ws.apache.org/soap/)
    - Implementation which uses the Soap protocol
  - **Soap Encoding**
    - Optional specification within the Soap standard (Section 5)
  - **Soap Engine**
    - Generic term, meaning a Web Services Engine
    - Processes Soap messages, calls implementation code, returns response
So what is SOAP?

- A light weight protocol for exchange of information in a distributed environment

- A standard for encoding messages as XML that can be delivered *via* a variety of transports including:
  - HTTP(S)
  - SMTP
  - Message Oriented Middleware such as WebSphere MQ

- Defines the format of what flows over the wire
What does the SOAP Protocol provide?

- A mechanism for defining the exchange of information between computers – the **SOAP message**
  - A message has both optional headers and a body that comprises the main message
- Rules for describing messages
  - Envelope, body, header
- Rules for describing and processing remote procedure call (RPC) messages
  - “uniform representation of remote procedure calls and responses”
  - RPC is not required
- A built-in error handling mechanism in the form of special Fault messages
- An optional mechanism for data representation in XML (SOAP encoding)
- A binding to HTTP
- A built-in extensibility mechanism through SOAP headers
SOAP vs XML over HTTP

- You may be wondering: Why don’t I just use HTTP GET and POST to transmit XML data over HTTP? Why do I need SOAP?
- Interoperability
  - Your XML over HTTP is almost guaranteed not to be compatible with someone else’s.
    - you would have to write a custom XML RPC processor for each partner’s XML definition of RPC
    - SOAP is a standard supported by many tools
- Economy of Scale
  - You will have to implement your own XML over HTTP libraries
  - SOAP is simple, standard, extensible, and dozens of implementations are available, many for free
- Development productivity
  - toolkits, wizards, and SOAP engines integrated into IDEs
  - standardization of skills
- Headers
  - ability to separate information about the message from the message body in a standard fashion
  - Leverage SOAP header processing services
What Does a SOAP Engine Do?

- Provides Java Classes to create and parse SOAP messages
- Server side infrastructure for deploying, managing and running SOAP enabled services
- Client-side API for invoking SOAP services
What Does a SOAP Engine Do?

- Receive a message from a Transport
- Check SOAP semantics
- Process the SOAP headers
- Deserialize the message
- Route the message to the service
- Serialize the response (if request/response)
- Process the response SOAP headers
- Send the response back out over the Transport
SOAP Messaging Model

- SOAP messages are fundamentally one way transmissions from a sender to a receiver
- Remote Procedure Call (RPC) can be simulated using two one-way messages
  - one for the RPC request
  - one for the RPC response
  - other communication patterns can be simulated as well but SOAP defines a convention for RPC
- The SOAP specification doesn't enforce anything about the "real" contents of the message as long as it is valid XML.
  - Use of 'SOAP encoding' is anticipated but not required
  - There are strict rules for how the messages elements map to a method call
  - Does not require HTTP
- The SOAP model for message processing allows one or more intermediate nodes (intermediaries) to process a message before the message reaches its final destination
What Are the Components of a SOAP Message?
The SOAP Message

- Separation of concern
  - Layered architecture
  - Implementation code receives only the message
- SOAP defines an "envelope"
  - "envelope" wraps the message itself
  - message is a different vocabulary
  - namespace prefix is used to distinguish the two parts
SOAP and Namespaces

- Namespaces are used heavily throughout SOAP
- There are two SOAP namespaces
  - One for the elements of a SOAP Envelope – required
  - One for elements related to the SOAP encoding – optional
- All content in a SOAP Header must be namespace qualified.
- Content in the SOAP Body (element) doesn't have to be namespace qualified, but it is a best practice to use a namespace
  - application specific namespace for the application specific elements
    - examples: stockQuoteService
  - multiple namespaces may be used in the body of the message
  - often used by the SOAP engine to identifying the Web service being called
  - WS-I restricts – requires a namespace on the Body element
    - Apache SOAP uses this namespace for dispatching

Best Practice: use application specific namespaces for the body
Soap and Namespaces

<SOAP-ENV:Envelope
xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getQuoteResponse
xmlns:ns1="urn:StockQuoteService"
SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:float">100.36</return>
</ns1:getQuoteResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
SOAP and XML Schema

- SOAP requires a rich set of XML tags to represent data passed in with SOAP Call requests and returned as responses
  - DTDs are not adequate to define complex datatypes and to model things like inheritance.
- XML Schema provide a general way to define data, validate data and place constraints on that data
  - Not developed specifically for SOAP but provides the functionality that is needed for SOAP
- SOAP can use XML Schema to encode data
  - Parameters to method calls
  - Return values
- XML Schema is not required by the SOAP specification but most SOAP implementations (e.g. Apache SOAP) use them
- Best Practice: Use XML Schema
Components of a SOAP Message

The SOAP message for a stock quote request as it appears on the WIRE

```xml
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Header>
        <dummy:header xmlns:dummy="http://nowhere.com"/>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <ns1:getQuote xmlns:ns1="urn:StockQuoteService"
            SOAP-ENV:encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/"/>
        <symbol xsi:type="xsd:string">IBM</symbol>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
The SOAP Envelope is the root element of the XML document representing a SOAP message. Everything is inside the Envelope.

The Envelope can contain an optional Header element.

The Envelope contains a required Body element.

Example Envelope:

```xml
<SOAP-ENV:Envelope ...
  <SOAP-ENV:Header>
    ... Optional
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    ... Required
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
SOAP Header

- The header is optional
- Elements in the head can use two attributes from the SOAP Envelope namespace
  - SOAP-ENV:mustUnderstand
  - SOAP-ENV:actor
- Sample Usage
  - Authentication
  - Digital signature support
  - Encryption support
  - Transaction management
- Example Header:

  ```xml
  <SOAP-ENV:Header>
    <dummy:header xmlns:dummy="http://nowhere.com"/>
  </SOAP-ENV:Header>
  ```
SOAP Header

- SOAP Headers are an important feature of SOAP
  - Separate application processing, QoS information from the application data
  - WS-xxxx standards define headers, eg. WS-Security

- Extension point
  - Allows for additional capabilities to be added without impacting the application

- Processed by intermediaries, or the Soap Engine
  - handlers = filters = intermediaries depending on the technology
  - In general, Intermediary is an application operating on a SOAP message between the requestor and the provider
  - A Handler is operating on the header of a SOAP message
Soap Header

- The `<Security>` header block provides a mechanism for attaching security-related information targeted at a specific receiver (SOAP actor). This MAY be either the ultimate receiver of the message or an intermediary.

```xml
<S:Envelope>
  <S:Header>
    ...<Security S:actor="..." S:mustUnderstand="...">
    ...
    </Security>
    ...
  </S:Header>
  ...
</S:Envelope>
```
SOAP Body

- Body element is always required
  - may contain zero or more elements, *e.g.* may be empty
- Body element contains
  - Required to contain well-formed XML
  - application specific XML vocabulary
  - *(should be valid against WSDL and Schema – outside of the SOAP spec)*
- Soap provides an optional specification for RPC calls
  - Body element must have a single child element, named after the method
  - Parameters are children of the method element, and must map exactly to the name, type, and order of the method parameters
- For Document style calls, the body element must be well formed XML

- SOAP engine executes service code based on the SOAP Body

- The body must be namespace qualified
SOAP Body

- The SOAP Specification is neutral on interpretation of the body
  - The body can represent a remote procedure call
  - The spec describes an element structure for representing RPCs
  - The body can simply represent an XML document fragment(s)
- The WSDL SOAP binding clarifies the interpretation of the body
  - It calls for two interpretations of what the body represents
  - It provides some choices for how typed data in the body (if any) is encoded
- The binding is controlled in the WSDL file
  - You can have bindings which are not SOAP (i.e. don’t conform to the SOAP envelope)
- The SOAP messages do not contain information about the binding
  - Their content is a result of the binding
SOAP Body

- WSDL defines two styles for what the body means:
  - **RPC Style**
    - The body represents a procedure call or method invocation
  - **Document Style**
    - The body represents data OR
    - The body represents a procedure call or method invocation
WSDL SOAP Bindings

- A WSDL SOAP Binding defines
  - The binding style (*Document* or *RPC*)
  - The part encoding (*literal* or *encoded*)

<table>
<thead>
<tr>
<th></th>
<th>Document</th>
<th>RPC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literal</strong></td>
<td>Standard Choice for MS tools</td>
<td></td>
</tr>
<tr>
<td><strong>‘SOAP’ Encoded</strong></td>
<td>Standard choice for Java tools (until recently)</td>
<td></td>
</tr>
</tbody>
</table>

- WS-I says do not use Soap Encoded
  - RPC/Literal support is optional
Encoding and Mapping

- SOAP needs a way to convert programming language types into XML for the SOAP Body when the body represents an RPC
- This is called type encoding
  - SOAP provides a pre-built encoding scheme
    - Based on subset of XML Schema datatypes
    - Defined in the namespace "http://www.w3.org/2001/XMLSchema"
  - You can also choose no encoding – this is called “literal XML”
    - Elements conform to an XML Schema

SOAP allows you to use any encoding scheme you want
provides means to plugin custom encoding code
SOAP -encoding simple examples:

```
<symbol xsi:type="xsd:string">IBM</symbol>
<return xsi:type="xsd:float">550.99</return>
```

WS-I Recommendation
"For interoperability, literal XML is preferred."
Encoding the SOAP view

- SOAP specification has an optional `encodingStyle` attribute
  - Attribute MAY appear on any element, but usually is on the body
  - There is an OPTIONAL SOAP encoding which is included in the SOAP specification
- If the encoding style is not provided, it is 'literal'
  - Most likely will be described by XML schema document
- SOAP encoding style is based on a simple type system
  - A type either is a simple (scalar) type or is a compound type constructed as a composite of several parts, each with a type
  - The problems with encoding comes from use of Array, Struct, and DateTime
  - EncodingStyle attribute should be set to `http://schemas.xmlsoap.org/soap/encoding/`
  - Used exclusively with RPC style (WSDL/SOAP would allow it to be used with document literal, but in practice never used this way)
- You could make up your own encoding, but it would be proprietary to your application. We advise against this.
  - example: `harrysEncoding` could say 'Date' maps to 'dd-mm-yyyy' format
How can the body represent a procedure call / method invocation?

- Explicitly – put the name of the method / procedure call in the body as a wrapper element (RPC Style)
  
  `<methodname>
    <arg1>value</arg1>
    <arg2>value</arg2>
    ...
    <argn>value</argn>
  </methodname>`

- Implicitly – the receiver of the message knows to interpret the data as an invocation so just put the arguments in the body (Document Style)
  
  `<arg1>value</arg1>
  <arg2>value</arg2>
  ...
  <argn>value</argn>`
There are two particular parts of the WSDL Binding to examine:

- The binding style (Document or RPC)
- The encoding use (literal or encoded)

<table>
<thead>
<tr>
<th>WSDL SOAP Binding Attribute</th>
<th>Document</th>
<th>RPC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literal</strong></td>
<td>Default for MS, WebSphere, and most newer tools</td>
<td>MS does not support</td>
</tr>
<tr>
<td>‘SOAP’ Encoded</td>
<td>Should never occur</td>
<td>Earlier Standard choice for Java tools</td>
</tr>
</tbody>
</table>

- First generation of Java tools supported RPC Encoded (often exclusively)
- MS always supported Document literal
- WAS 5.02 and later supports RPC Literal, Document Literal, and RPC SOAP encoded
WSDL Encoding Example

**SOAP Encoded**

```xml
<binding name="TemperatureConverterBinding" type="tns:TemperatureConverter">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="fahrenheitToCelsius">
    <soap:operation soapAction="" style="rpc"/>
    <input name="fahrenheitToCelsiusRequest">
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:TemperatureConverter" use="encoded"/>
    </input>
  </operation>
</binding>
```

**Literal**

```xml
<binding name="CSharpTempConverterSoap" type="s0:CSharpTempConverterSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <operation name="celsiusToFahrenheit">
    <soap:operation soapAction="http://wsbootcamp.com/webservices/celsiusToFahrenheit" style="document"/>
    <input>
      <soap:body use="literal"/>
    </input>
  </operation>
</binding>
```
Encoding

- SOAP encoding is outlawed by the Basic Profile
  - It defined a new type system, together with rules for how to encode it
  - Vendors interpreted it differently, and no language specific mappings were defined
  - This resulted in numerous interoperability problems
- Literal encoding means no encoding at all
  - Created artifacts are direct instances of the XML Schema type definitions referred to in WSDL messages
  - This means that we settle down on something that is well established and supported, without creating new type systems and encoding rules
- Hence, for the remainder of the discussion, we will assume literal encoding

Best Practice: Always use Literal
Invocation *Style*

- **RPC style** 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
  - requires a Response message

- **Document style** says “Send any well-formed XML message”
  - Allows to have more than one child element to body
  - RPC allows only one child, i.e. the element named after the operation
  - Interpretation of the message content completely left to the receiver
    - Does not imply that there are parameters, procedure names, *etc.*
  - Generally harder to manage by the service provider

Looking at a SOAP message, you cannot necessarily tell if it is document or RPC style
“Wrapped” Document Literal

- If you use doc literal, the message may be “wrapped” or “unwrapped”

- Wrapped basically means that there is a single complex type in the message
  - That element name can be used as the operation name
- Result is the ability to treat the message as an RPC call
- Resulting SOAP messages looks exactly like RPC
- Avoids ambiguous messages
- Wrapped document literal is the default in WebSphere and .NET tooling
- This is really the way the industry is going
Java code

What does the WSDL and an XML instance look like for

public void myMethod(int x);

for each style/use choice?
RPC Style Request

- StockQuote RPC request (Taken from WSAD 5.1)
  - Method invoked is the same as the name of the SOAP structure (first element in the body)
  - Parameters are XML elements named and typed the same as the corresponding parameter in the method call.
  - Parameters must appear in the same order as in the method signature
  - Note the type information in the symbol element – soap encoding

```xml
<?xml version='1.0' encoding='UTF-8'?>
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
  <ns1:getQuote xmlns:ns1="http://tempuri.org/StockQuoteService"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <symbol xsi:type="xsd:string">ibm</symbol>
  </ns1:getQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
RPC/literal

WSDL

```xml
<message name="myMethodRequest">
  <part name="x" type="xsd:int"/>
</message>

<message name="empty"/>

<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>

<!-- I won't bother with the details, just assume it's RPC/literal. -->
```

SOAP Message

```xml
<soap:envelope>
  <soap:body>
    <myMethod>
      <x>5</x>
    </myMethod>
  </soap:body>
</soap:envelope>
```
Document Style Request

- StockQuote Document request (Taken from WSAD 5.1)
  - Method name and parameters follow the same pattern as an RPC call
  - This is not required. Body must simply be well formed XML
  - Note there is **no type information** in the symbol element – no soap encoding

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getQuote xmlns="http://DefaultNamespace">
      <symbol>ibm</symbol>
    </getQuote>
  </soapenv:Body>
</soapenv:Envelope>
```
WSDL

```xml
<types>
  <schema>
    <element name="xElement" type="xsd:int"/>
  </schema>
</types>

<message name="myMethodRequest">
  <part name="x" element="xElement"/>
</message>

<message name="empty"/>

<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>
```

SOAP Message

```xml
<soap:envelope>
  <soap:body>
    <xElement>5</xElement>
  </soap:body>
</soap:envelope>
```

<!-- I won't bother with the details, just assume it's document/literal. -->
WSDL

```xml
<types>
  <schema>
    <element name="myMethod">
      <complexType>
        <sequence>
          <element name="x" type="xsd:int"/>
        </sequence>
      </complexType>
    </element>
  </schema>
</types>

<message name="myMethodRequest">
  <part name="parameters" element="myMethod"/>
</message>

<message name="empty"/>

<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>

<!-- I won't bother with the details, just assume it's doc/literal. -->
```

SOAP Message

```xml
<soap:envelope>
  <soap:body>
    <myMethod>
      <x>5</x>
    </myMethod>
  </soap:body>
</soap:envelope>
```
Best Practice – Use document/literal wrapped

Document/literal wrapped is the way to go

- It is becoming the industry standard.
- It interoperates best with .NET.
- WSDL 2.0 is moving toward a similar style.

But, there are certain unusual situations where you will want to use RPC/Literal

- .NET will not support it normally – can be ‘hacked’ to produce the correct message
RPC Style Response

- The body of the StockQuote RPC response
  - The response struct can be named anything, but by convention it is named `methodNameResponse`
  - An accessor named `return` represents the return value of the method call

```xml
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <SOAP-ENV:Body>
  <ns1:getQuoteResponse xmlns:ns1="urn:StockQuoteService"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
   <return xsi:type="xsd:float">100.36</return>
  </ns1:getQuoteResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Document Style Response

- The body of the StockQuote Document response
  - The response struct can be named anything, but by convention it is named `methodNameResponse`
  - An element named `methodNameReturn` represents the return value of the method call

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getQuoteResponse xmlns="http://DefaultNamespace">
      <getQuoteReturn>83.46</getQuoteReturn>
    </getQuoteResponse>
  </soapenv:Body>
</soapenv:Envelope>
```
SOAP Body *per WS-I*

- The children of the soap:Body element in a MESSAGE MUST be namespace qualified.

- A PROCESSOR MUST NOT mandate the use of the xsi:type attribute in messages except as required in order to indicate a derived type
  - Apache SOAP requires xsi:type,
  - Apache Axis does not require it.

- A MESSAGE MUST NOT contain soap:encodingStyle attributes on any element that is a child of soap:Body
Soap – Fault element

- SOAP Fault element is used to carry error and/or status information within a SOAP message
- If present, the SOAP Fault element MUST appear as a body entry instead of a xxxMethodName or xxxResult
- Fault element defines the following four subelements:
  - faultcode
    - SOAP-ENV:Version Mismatch
      - Processing party found an invalid namespace for the SOAP Envelope
    - SOAP-ENV:MustUnderstand
      - a header which contained a SOAP-ENV:mustUnderstand attribute could not be processed
    - SOAP-ENV:Client
      - The client sent an incorrectly formed or incomplete message
    - SOAP-ENV:Server
      - The server was unable to process the message. There was nothing wrong with the message.
  - faultstring
  - faultactor
  - detail
SOAP Fault

- A Fault is used to return an error or status information

```xml
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultcode>SOAP-ENV:Server</faultcode>
            <faultstring>Method 'getQuoted' is not supported.</faultstring>
            <faultactor>/StockQuote/servlet/rpcrouter</faultactor>
            <detail>
                ...
            </detail>
        </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
SOAP Fault *per WS-I*

- *Fault* element MUST NOT have element children other than *faultcode*, *faultstring*, *faultactor* and *detail*. Can put your own elements as children to *detail* element.

- PROCESSORs MUST generate a mustUnderstand fault when a message contains a mandatory header that the processor does not understand.

- Where the normal outcome of processing a SOAP message would have resulted in the transmission of a SOAP response, but rather a SOAP Fault is generated instead, a PROCESSOR MUST transmit a SOAP Fault message in place of the response.
Client API

- Provide a proxy for the Web service
- Provide a way to build up a set of parameters for the service request
- Provide a way to invoke the Web service
- Provide a way to get the result of invoking the service
- Provide a way to notify the application of faults that occur
SOAP over HTTP

- HTTP is one transport for SOAP messages
  - also HTTPS which just adds SSL security encryption
- HTTP Request Header must include SOAPAction header field (URI)
  - Indicates intent of HTTP Request
  - Format not specified, can be empty ""
- Some Web Servers and SOAP engines use SOAPAction for routing
- The SOAP Envelope is carried in content of HTTP POST
- Response may contain a SOAP Envelope
- When an Error occurs:
  - Must send HTTP 500 with SOAP envelope containing fault (not 200)
- The use of HTTP as the protocol makes the interaction synchronous

WS-I Recommendation

- header MAY contain any quoted string including "".
- The SOAPAction header is purely a hint to processors.
- All vital information regarding the intent of a message is carried in the Envelope.
- specifies HTTP response codes for variety of situations
SOAP over HTTP – Request

POST /StockQuote/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 454
SOAPAction: "urn:StockQuoteService"

```xml
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
    <ns1:getQuote xmlns:ns1="urn:StockQuoteService"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <symbol xsi:type="xsd:string">IBM</symbol>
    </ns1:getQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAPAction header is required by some SOAP engines to route the request to the matching Web service – non-conformance with WS-I
SOAP over HTTP – Response

HTTP/1.1 200 OK
Server: WebSphere Application Server/4.0
Content-Type: text/xml; charset=utf-8
Set-Cookie: JSESSIONID=0000NMQZPAGA30SVMHZXQ125FRQ:-1;Path=/
Cache-Control: no-cache="set-cookie,set-cookie2"
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Content-Length: 474
Content-Language: en
Connection: close

<?xml version='1.0' >
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getQuoteResponse xmlns:ns1="urn:StockQuoteService"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:float">100.36</return>
</ns1:getQuoteResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
SOAP Attachments

- Sometimes it is desirable to transmit a SOAP message along with some attachments
  - Legal briefs
  - Engineering diagrams
  - Specifications
  - Contract documents

- SOAP Attachments are a proposal for sending attachments along with a SOAP message
  - in the W3C approval process as a Note

- SOAP Attachments rely on packaging a SOAP messages as part of a MIME multipart structure
  - The SOAP message can refer to the attachments using URIs constructed during creation of the multipart

- SOAP Attachments are independent of the transport
SOAP Intermediaries

- We need a way to modularize services to promote reuse. One way is via intermediaries.
- Intermediaries can both receive and forward SOAP messages.
  - can also add headers
- Examples:
  - Authentication
  - Digital signature
  - Encryption
  - Internal message routing
  - Other processing based on envelope characteristics
- SOAP-ENV:actor attribute indicates a header meant for an intermediary
WSDL Service Description

Web Services Description Language

- From the specification: "XML format for describing network services as a set of endpoints operating on messages... The operations and messages are described abstractly, and then bound to a concrete network protocol and message format"

- Service description is the key to interoperability of services

Serves as a recipe for automating the details involved in applications communication
WSDL Definition

- **WSDL describes:**
  - Implementation independent description of the service
    - Service interface definition
  - Where the service is located
    - Service implementation definition

- **WSDL provides:**
  - A simple standardized way for service providers to describe the basic format of requests to their system.
  - A "contract" between the client and server

- **WSDL specification**
  - [http://www.w3.org/TR/2001/NOTE-wsdl-20010315](http://www.w3.org/TR/2001/NOTE-wsdl-20010315)
How Is WSDL Used?

- **Standardized Service Interface Descriptions**
  - Allows advertisement and dynamic discovery of services
  - Communicates to the service requesters all of the information they need to be able to invoke the service
  - Enables dynamic binding to service by service requesters
  - Can be stored in a UDDI registry

- **Used by Tools**
  - To generate client code to bind to a service
  - To generate SOAP deployment descriptors
Our Example

- **EmployeeManager class**
  - 2 methods we will expose
    - public Vector getEmployeeList()
    - public Employee getEmployee(String lastName)

- **Employee Class**
  - return type used in the service

```
EmployeeManager
  getEmployeeList()
  getEmployee()

Employee
  employeeNo, lastName,
  firstName, phoneNumber
  department, salary, hireDate
```
Basic WSDL Syntax

- Describes
  - What a service can do
  - Where the service can be found
  - How to invoke the service

- Describes them as
  - Collection of endpoints that are able to exchange messages
## WSDL Document Overview

### Definition
- The root of the WSDL document
- Contains the definition of one or more services
- Usually contains attributes

### Service
- Defines the service

### Messages and PortTypes
- Describes the actions available for the service

### Bindings
- How to communicate with the service
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.
WSDL Elements

**Service Interface**

Abstract, reusable service definition

Represents a type of service that can be implemented

**Service Implementation**

Implementation of one or more service interfaces

Contains the endpoint reference
The root element of the WSDL document
Usually defines a single service

```xml
<definitions
    targetNamespace="http://employee.webservices.wsad.ibm.com"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://employee.webservices.wsad.ibm.com"
    xmlns:intf="http://employee.webservices.wsad.ibm.com"
    xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wSDLsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">"
<definitions> element attributes

- **targetNamespace**
  - Chosen to be unique for this service
  - targetNamespace="http://employee.webservices.wsad.ibm.com"

- **xmlns**
  - Default namespace – the WSDL defined name space
  - xmlns="http://schemas.xmlsoap.org/wsdl/"

- **xmlns:apachesoap**
  - Apache SOAP Type namespace
  - xmlns:apachesoap="http://xml.apache.org/xml-soap"

- **xmlns:impl**
  - Implementation namespace
  - xmlns:impl="http://employee.webservices.wsad.ibm.com"

- **xmlns:intf**
  - Interface namespace
  - xmlns:intf="http://employee.webservices.wsad.ibm.com"

- **xmlns:wsdl**
  - WSDL namespace
  - xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

- **xmlns:wsdlsoap**
  - SOAP namespace
  - xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"

- **xmlns:xsd**
  - Schema namespace
  - xmlns:xsd="http://www.w3.org/2001/XMLSchema"
<types> element attributes

- Contains data type definitions other than those defined by the base schema
- May be imported into the WSDL document

```xml
<wsdl:types>
  <schema elementFormDefault="qualified"
    targetNamespace="http://employee.webservices.wsad.ibm.com"
    xmlns="http://www.w3.org/2001/XMLSchema">
    <complexType name="Employee">
      <sequence>
        <element name="department" nillable="true" type="string"/>
        <element name="phoneNumber" nillable="true" type="string"/>
        <element name="lastName" nillable="true" type="string"/>
        <element name="hireDate" nillable="true" type="dateTime"/>
        <element name="firstName" nillable="true" type="string"/>
        <element name="salary" type="double"/>
        <element name="employeeNo" nillable="true" type="string"/>
      </sequence>
    </complexType>
    <element name="getEmployeeResponse">
      <complexType>
        <sequence>
          <element name="getEmployeeReturn" nillable="true" type="intf:Employee"/>
        </sequence>
      </complexType>
    </element>
  </schema>
</wsdl:types>
```
<message> and <part> Elements

- <message>
  - A single piece of information moving between the requester and provider
  - A single interaction between requester and provider
  - By convention
    - methodNameRequest and methodNameResponse

- <part>
  - Describes a piece of data associated with the message
  - Optional

```
<message name="getEmployeeRequest">
  <part name="lastName" element="impl:getEmployee"></part>
</message>

<message name="getEmployeeResponse">
  <part name="result" element="impl:getEmployeeResponse"></part>
</message>
```

Request and Response messages for public Employee getEmployee(String lastName)
<operation> and <portType>

- Defines what operations the Web Service provides
  - <operation>
    - An action
    - Like a Java Method
    - Three messages
      - input message
      - output message
      - fault message
  - <portType>
    - A collection of operations
    - Like a Java Class

```xml
<message name="getEmployeeRequest">
  <part name="lastName" element="impl:getEmployee"></part>
</message>
<message name="getEmployeeResponse">
  <part name="result" element="impl:getEmployeeResponse"></part>
</message>

<portType name="EmployeeManager">
  <operation name="getEmployee">
    <input message="impl:getEmployeeRequest" name="getEmployeeRequest" />
    <output message="impl:getEmployeeResponse" name="getEmployeeResponse" />
  </operation>
</portType>
```
**<binding> Elements**

- **<binding>**
  - HOW the operation is invoked
  - Ties to the <portType> element to the protocol defined for the binding
  - Defines the transport **protocol** and **style** (rpc or document)
  - If a service supports more than one protocol then there should be multiple bindings for the port type.
  - For each operation of the <portType> each <message> is detailed
  - For SOAP, the <soap:body> element defines **use** (encoding or literal) and **namespace**
  - If using encoding, <soap:body> specifies **encodingStyle**

```xml
<portType name="EmployeeManager">
  <operation name="getEmployee">
    <input message="impl:getEmployeeRequest" name="getEmployeeRequest" />
    <output message="impl:getEmployeeResponse" name="getEmployeeResponse"/>
  </operation>
</portType>

<binding name="EmployeeManagerSOAPBinding" type="impl:EmployeeManager">
  <wsdlsoap:binding
    transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <wsdl:operation name="getEmployee">
    <input name="getEmployeeRequest" />
    <output name="getEmployeeResponse" />
  </wsdl:operation>
</binding>
```
<service> Element Attributes

- **<service>**
  WHERE the service is located
  A collection of <ports>s
- **<port>**
  defines the availability of a particular binding at a specific endpoint
  binding attribute must correspond to a <binding> element
  <soap:address> defines the actual location of the service

```xml
<binding name="EmployeeManagerSOAPBinding" type="impl:EmployeeManager">

<service name="EmployeeManagerService">
  <port name="EmployeeManagerServicePort" binding="impl:EmployeeManagerSOAPBinding">
    <wsdlsoap:address location="http://localhost:9080/WSWSDLWeb/services/EmployeeManager" />
  </port>
</service>
```
The WSAD WSDL Editor
What Is UDDI?

Universal Description, Discovery and Integration

- Purpose
  - To facilitate service discovery both at design time and at runtime
  - A platform independent framework for describing services, discovering businesses, and integrating business services via the Internet

- Based in existing standards such as
  - SOAP and XML

- It is a technical discovery layer defining
  - The structure for the registry of service provider and services
  - The API that can be used to access registries with this structure
  - The organization and project defining this registry structure and its API

- A search engine for application clients
  - Some implementations provide browser human readable interfaces
Uddi.org

- Partnership among industry & business leaders
  - Founders (IBM, Microsoft, Ariba) developed basic version 1 specification

- Eventually grew to over 320 UDDI community members in uddi.org
  - Members jointly developed Versions 2 & 3
  - Included companies such as Accenture, Compaq, Fujitsu, HP, Intel, NTT, Oracle, SAP, Sun & many others
Publish, Find and Bind

Service Broker Registry

Service Consumer

Service Provider

Publish WSDL description via UDDI

Describe via WSDL

Connect via SOAP

Find via UDDI
The Evolution of UDDI

- **UDDI Version 2 – (June 2001)**
  - Business Relationships
    - Modeling of Complex Organizations – business units, departments, divisions, and subsidiaries
    - Uses such as membership, certification, *etc.*
  - Additional categorization and identifier schemes
  - Support for externally 'checked' taxonomies
  - Richer search options: more expressive query parameters, more powerful & complex filters
    - Better results with less work
  - Improved Internationalization
    - Names and descriptions in multiple languages
  - Peer based replication for improved scaling
The Evolution of UDDI

- UDDI Version 3 – OASIS Committee draft
  - Multi-registry topologies
  - New subscription API
  - New security features
  - Improved WSDL support
  - Core information model improvements
The Evolution of UDDI

- e-mail, FTP, HTTP Get
  - Simplest approach
  - Static
    - If service changes, difficult to communicate to all users of the service

- IBM's ADS, Microsoft's .NET DISCO
  - Provider publishes service information in a format specific to one or more websites and then publishes the URL to potential service requesters
  - Better update capability
  - Limited information about the service

- WSDL Repository
  - *e.g.* XMethods
  - Similar to previous
  - Additional capability to notify requester when service changes

- UDDI
  - Provides very sophisticated publish and find capability
  - Provides much more information about the service
  - Does not depend on a particular mechanism of service description
UDDI Registry

- There are three roles defined for UDDI data
  - Described in terms of telephone directories
  - Conceptual only

<table>
<thead>
<tr>
<th>White Pages</th>
<th>Yellow Pages</th>
<th>Green Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains contact information for service providers so that a requester can contact the provider directly (negotiation, technical support, etc.)</td>
<td>Contains service information; service categories</td>
<td>Technical information about the individual service</td>
</tr>
</tbody>
</table>
UDDI Provides

- UDDI provides three functions common to all "name services"
  - **Publish**
    - A provider of a Web Service Registers itself
  - **Find**
    - An application finds a provider of a Service
  - **Bind**
    - An application connects to the Service
UDDI Registry Interactions

1. Companies and Organizations describe themselves
2. Companies and Organizations publish their services
3. Client applications invoke remote services using WSDL information

- UDDI Registry
  - Business Entities
  - Services, Bindings, tModels
  - WSDL

- Human users and client applications search for business entities
- Human users and client applications search for services
UDDI Registry Interactions

- **Registry operator**
  - An organization that provides public Web service registries. As noted earlier, examples include IBM, Microsoft, HP, SAP, and Ariba for UDDI-based registries, and at least one publicly accessible ebXML registry operated by Hong Kong University.

- **Submitting organization**
  - The business that registers its services with public registries. It retains ownership over the information it submits to a registry.

- **Content submitter**
  - The person or software performing the registry submission for a submitting organization.

- **Registry guest**
  - A user (human or software) that browses or searches a registry's contents.
UDDI and SOAP

- SOAP [Simple Object Access Protocol] is just XML
- UDDI requests use Message Based SOAP
- Example UDDI SOAP request:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <delete_business generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>myAuthInfoToken</authInfo>
      <AuthInfo>
        <businessKey>0000097D-274B-4166-8E9C-2E3F2F95277F</businessKey>
      </AuthInfo>
    </delete_business>
  </Body>
</Envelope>
```

- SOAP has an Envelope and a Body
- UDDI requests are placed within the SOAP Body
- Entire SOAP message sent as content of HTTP Post request
UDDI and EJB

- Provides an EJB interface to the UDDI Registry
  - This is entirely separate from the SOAP interface.
- Consists of 2 EJBs –
  - "Inquiry" for Inquiry API
  - "Publish" for Publish API.
- Supports only UDDI version 2.0.
- Not quite all UDDI v 2.0 methods are implemented
  - *e.g.* get_ and discard_authToken, get_businessDetailExt.
- Arguments for the EJB methods are java objects in the package com.ibm.uddi.datatypes.
UDDI and EJB

- First obtain the Inquiry and/or Publish bean home interfaces.

- For each UDDI call in an EJB application:
  - First build up the datatypes which will form the arguments to the required method on the bean, using the setter methods.
  - Pass these datatypes to the bean method.
  - Interrogate the datatype returned, using its getter methods.
UDDI Structure

- UDDI Structure is defined in XML
- Defines 4 basic elements
  - Business Entities
  - Business Services
  - Binding Templates
  - tModels

Diagram:

- Business Entity
  - Business Service
    - Binding Template
      - Interface (binding.wsdl)
    - Binding Template
      - Implementation (service.wsdl)
    - Binding Template
      - tModel
    - Binding Template
      - tModel
    - Binding Template
      - tModel
  - Business Service
    - Binding Template
      - tModel
    - Binding Template
      - tModel
    - Binding Template
      - tModel
businessEntity

- The white and yellow pages of the registry
- Contains business information about the service provider
  - business name, business contacts, descriptions, identifiers, categories, etc...

```xml
<?xml version="1.0" encoding="utf-8" ?>
<businessDetail generic="1.0" xmlns="urn:uddi-org:api" operator="www.ibm.com/services/uddi"
truncated="false">
  <businessEntity authorizedName="100000A8B6"
operator="www.ibm.com/services/uddi"
businessKey="2CEF9630-9118-11D6-B746-000C0E00ACDD">
    <discoveryURLs>...</discoveryURLs>
    <name>WSADWS</name>
    <description xml:lang="en">WSAD Web Services Bootcamp</description>
    <businessServices>...</businessServices>
    <identifierBag>...</identifierBag>
    <categoryBag>...</categoryBag>
  </businessEntity>
</businessDetail>
```
Green pages (part 1)
- Provides non technical service information
- Used to group a set of related Web Services

Categorization is available at this level
- Maps to a WSDL service

```xml
<businessServices>
  <businessService serviceKey="51A0BEA0-9118-11D6-B746-000C0E00ACDD"
    businessKey="2CEF9630-9118-11D6-B746-000C0E00ACDD">
    <name>TemperatureConverterService</name>
    <description xml:lang="en">Temperature Converter Service</description>
  </businessService>
</businessServices>
```
bindingTemplate

- Green pages (part 2)
- Contains service access information
- Points to a service implementation description
  - Wraps a WSDL port

```xml
<businessService serviceKey="51A0BEA0-9118-11D6-B746-000C0E00ACDD"
businessKey="2CEF9630-9118-11D6-B746-000C0E00ACDD">
  <name>TemperatureConverterService</name>
  <description xml:lang="en">Temperature Converter Service</description>
  <bindingTemplates>
    <bindingTemplate serviceKey="51A0BEA0-9118-11D6-B746-000C0E00ACDD"
      businessKey="2CEF9630-9118-11D6-B746-000C0E00ACDD">
      <description xml:lang="en"> </description>
      <accessPoint URLType="http">http://localhost:8080/WSSWeb/servlet/rpcrouter</accessPoint>
    </bindingTemplate>
    ...<tModelInstanceDetails>....<tModelInstanceDetails>
  </bindingTemplates>
</businessService>
```
tModel

- **Technology Model**
- The technical fingerprint (also known as the service type)
- Points to a service interface description
- Example
  - Use a tModel to define a new WSDL port type
  - Specify a business service implements that port type by associating the tModel with one of the services binding templates

```xml
(bindingTemplate bindingKey="51A838B0-9118-11D6-B746-000C0E00ACDD"
serviceKey="51A0BEA0-9118-11D6-B746-000C0E00ACDD">
...
(tModelInstanceDetails>
(tModelInstanceInfo tModelKey="UUID:CF2D5680-9104-11D6-B746-000C0E00ACDD">
(instanceDetails>
(overviewDoc>
(overviewURL>
http://localhost:8080/WSSWeb/wsdl/TempConverter-service.wsdl#TemperatureConverterPort
(overviewURL>
</overviewDoc>
</instanceDetails>
</tModelInstanceInfo>
</tModelInstanceDetails>
</bindingTemplate>
```
WSDL and UDDI

**WSDL**
- Service Implementation
  - import
- Service
  - port
  - port
- Service Interface
  - types
  - message
  - portType
  - binding

**UDDI**
- Business Entity
- Business Service
  - Binding Template
  - Binding Template
  - tModel
UDDI4J

<table>
<thead>
<tr>
<th>Business</th>
<th>Service</th>
<th>Binding</th>
<th>tModel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save/Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>save_business</td>
<td>save_service</td>
<td>save_binding</td>
<td>save_tModel</td>
</tr>
<tr>
<td>Delete</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete_business</td>
<td>delete_service</td>
<td>delete_binding</td>
<td>delete_tModel</td>
</tr>
<tr>
<td>Find</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>find_business</td>
<td>find_service</td>
<td>find_binding</td>
<td>find_tModel</td>
</tr>
<tr>
<td>GetDetail</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_businessDetail</td>
<td>get_serviceDetail</td>
<td>get_bindingDetail</td>
<td>get_tModel</td>
</tr>
</tbody>
</table>

Business Entity

Business Service

Binding Template

tModel
Finding a Service with UDDI4J

- Access the registry
- Collect the tModels
  - Based on the service name
  - "http://www.temperatureconverter.com/definitions/TemperatureConverterRemoteInterface"
- Find the Business Entity
  - Based on the business Name
    - WSADWS
- Find the Business Services of the Business Entity
- Find the Bindings of the Service that implements the tModel
  - Match the Service keys with the Binding Template keys
- Collect the Access Points from the Bindings
- Set the Access Point in the Proxy generated for the Service
- Invoke the Service based on the Access Point
Finding a Service with UDDI4J

Provider
WSADWS

Business Entity
(BusinessInfo)

BusinessService
(ServiceInfo)

Service

"http://www.temperatureconverter.com/
definitions/TemperatureConverterRemoteInterface"

Service Interface
(TModelInfo)

TModelBag

Service ID
(serviceKey)

Binding Detail
(BindingTemplate)

EntryPoint
(AccessPoint)

http://localhost:8080/WSWeb/servlet/rpcrouter

UUIDs
(TModelKey)
**UDDI V3... What’s Coming...**

- UDDI V3 creates a more robust registry for the growth of Web Services, addressing:
  - Affiliated Registry Environments
  - Policy & Security
  - Modeling Enhancements
  - Tooling Needs
  - Extensions

- Now as OASIS Specifications, they will grow to address infrastructure needs of Web Services and improve compatibility with other standards
References

- SOAP V1.1 Specification
  - http://www.w3c.org/TR/SOAP/

- SOAP V1.2 Specification
  - http://www.w3.org/TR/soap12-part1
  - http://www.w3.org/TR/soap12-part2/

- Apache SOAP V2.2

- Apache AXIS

- SOAP Interoperability Lab
  - http://www.xmethods.net/ilab/ilab.html

- Directory for SOAP 1.1 developers
  - http://www.soapware.org/

- SOAP developerWorks articles by Graham Glass
UDDI References

- White papers, product offerings
  - http://www.ibm.com/webservices

- Tom Bellwood:

- Doug Tidwell: Introduction to UDDI4J
  - ibm.com/developerWorks/library/ws-uddi4j.html

  - www.oasis-open.org/committees/uddi-spec/

- The UDDI Business Registry nodes:
  - uddi.ibm.com
  - uddi.microsoft.com
  - www.ntt.com/uddi
  - uddi.sap.com
UDDI References

- **Providing Taxonomies in V2 (external validation)**
  

- **February 2002 developerWorks Journal article**
  "Build and deploy your own Web services"

- **Book by Steve Graham et al, 2002.**
  *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*

- **Software:**
  - UDDI4J – open-source Java API to access UDDI
  - WebSphere v5.0 includes UDDI V2 support
  - Web Services ToolKit (WSTK)