

CSI 3140

WWW Structures, Techniques and Standards

**Web Services:
JAX-RPC, WSDL, XML Schema, and
SOAP**

Web Services Concepts

- ◆ A web application uses Web technologies to provide functionality to an end user
- ◆ A web service uses Web technologies to provide functionality to another software application

Web Services Concepts

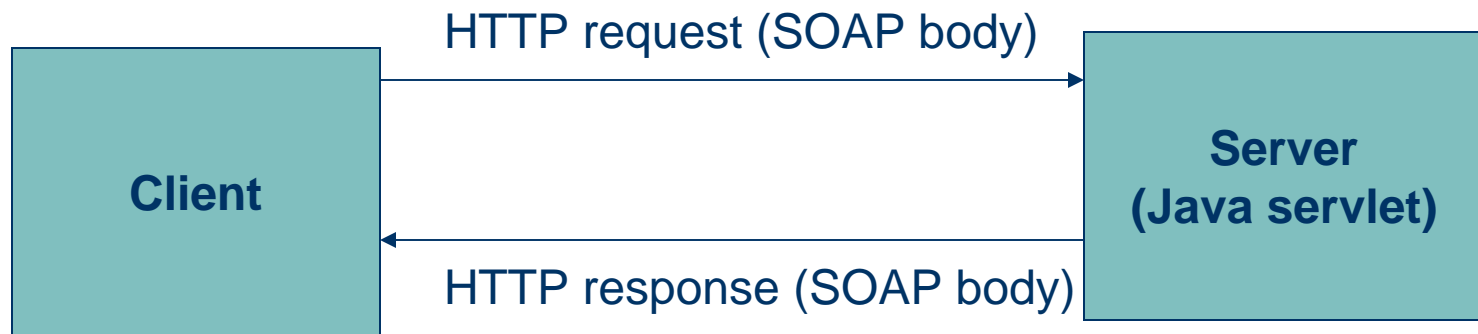
- ◆ Standard web services technologies:
 - Communication via SOAP XML vocabulary documents over HTTP
 - Operations of web service defined by Web Services Definition Language (WSDL) XML vocabulary
 - Data within WSDL defined using XML Schema

Web Services Concepts

- ◆ Higher-level API's are often used to automatically generate web services client and server communication software
 - We will use the Java API for XML-based Remote Procedure Call (JAX-RPC)
 - Microsoft .NET framework is one popular alternative to JAX-RPC

Web Services Concepts

- ◆ Web services conceptually are just specialized web applications:



Web Services Concepts

- ◆ Body of web services request is analogous to calling a method

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope ...>
  <env:Body>
    <ns0:NDFDgen>
      <latitude xsi:type="xsd:decimal">40.28</latitude>
      <longitude xsi:type="xsd:decimal">-79.49</longitude>
    ...
  </env:Body>
</env:Envelope>
```

Operation name (like method name)

Input parameter names

Input parameter values

Web Services Concepts

- ◆ Body of web services response is analogous to value returned by a method

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope ...>
  <SOAP-ENV:Body>
    <NDFDgenResponse>
      <xmlOut xsi:type="xsd:string">
        ...
        <name>Daily Maximum Temperature</name>
        <value>47</value>
        ...
      </xmlOut>
    </NDFDgenResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Return value data type

Return value {

Web Services Concepts

- ◆ WSDL defines web service
 - Operations
 - Parameters
 - Return values
 - Communication protocols
- ◆ Basically an API for the web service
 - Facilitates automated client/server software generation

Web Services Concepts

◆ Portions of NWS WSDL

Input
params

```
<?xml version="1.0"?>
<definitions ...>
  ...
  <message name="NDFDgenRequest">
    <part name="latitude" type="xsd:decimal" />
    <part name="longitude" type="xsd:decimal" />
    ...
  </message>
```

Data types
defined using
XML Schema

Return
type

```
<message name="NDFDgenResponse">
  <part name="xmlOut" type="xsd:string" />
</message>
```

```
<portType name="ndfdXMLPortType">
  <operation name="NDFDgen"> Operation name
    <documentation>...</documentation>
    <input message="tns:NDFDgenRequest" />
    <output message="tns:NDFDgenResponse" />
  </operation>
```

```
  ...
</portType>
  ...
</definitions>
```

Web Services Concepts

- ◆ Java Web Services Developer Pack (JWS DP) *wscompile* tool can implement a Java API from a WSDL

```
// Create an object representing
// the NWS web service.
NdfdXMLPortType ndfdWS =
    new NdfdXML_Impl().getNdfdXMLPort();
```

Classes and methods
generated by *wscompile*

```
// Request a forecast from the web service
BigDecimal latitude = new BigDecimal(40.28);
BigDecimal longitude = new BigDecimal(-79.49);
String forecast =
    ndfdWS.NDFDgen(latitude, longitude,
    // other parameters ...
    );
```

This method automatically
handles SOAP request and response

Web Services Concepts

◆ Writing the server for a web service with JWS DP:

- Write a Java interface defining the API
- Implement the interface
- JWS DP generates
 - SOAP and communication handling classes
 - WSDL

Web Services Concepts

◆ Example Java interface

```
public interface NdfdXMLPortType extends Remote {
    public String NDFDgen(BigDecimal latitude,
                          BigDecimal longitude,
                          // other parameters ...
                          )
        throws RemoteException;
}
```

Web Services Concepts

◆ Example implementation

```
public class WeatherImpl implements NdfdxMLPortType {
    public String NDFDgen(BigDecimal latitude,
                          BigDecimal longitude,
                          // other parameters ...
                          )
        throws RemoteException
    {
        // Return dummy string, ignoring parameter values
        String retVal =
            "<?xml version='1.0' ?> <dwml version='1.0' ... >" +
            "... " + "</dwml>";
        return retVal;
    }
}
```

Web Services Examples

Tons of WS available on the internet.

- ◆ <http://www.websvcicex.net> as a good collection
 - Geo IP: <http://www.websvcicex.net/geoipservice.asmx?op=GetGeoIP>
 - Whois: <http://www.websvcicex.net/whois.asmx?op=GetWhoIS>
 - SMS: <http://www.websvcicex.net/sendsmsworld.asmx>
 - Etc..
- ◆ Google:
- ◆ Amazon
 - S3

JWSDP: Server

- ◆ Application: currency converter
 - Three operations:
 - *fromDollars*
 - *fromEuros*
 - *fromYen*
 - Input: value in specified currency
 - Output: object containing input value and equivalent values in other two currencies

JWSDP: Server

1. Write service endpoint interface
 - May need to write additional classes representing data structures
2. Write class implementing the interface
3. Compile classes
4. Create configuration files and run JWSDP tools to create web service
5. Deploy web service to Tomcat

JWSDP: Server

- ◆ Service endpoint interface
 - Must extend *java.rmi.Remote*
 - Every method must throw *java.rmi.RemoteException*
 - Parameter/return value data types are restricted
 - No *public static final* declarations (global constants)

JWSDP: Server

- ◆ Allowable parameter/return value data types
 - Java primitives (*int, boolean, etc.*)
 - Primitive wrapper classes (*Integer, etc.*)
 - *String, Date, Calendar, BigDecimal, BigInteger*
 - *java.xml.namespace.QName, java.net.URI*
 - Struct: class consisting entirely of public instance variables
 - Array of any of the above

JWSDP: Server

- ◆ Struct for currency converter app (data type for return values)

```
package myCurCon;  
  
public class ExchangeValues {  
    public double dollars;  
    public double euros;  
    public double yen;  
}
```

JWSDP: Server

◆ Service endpoint interface

```
package myCurCon;

public interface CurCon extends java.rmi.Remote {
    public ExchangeValues fromDollars(double dollars)
        throws java.rmi.RemoteException;
    public ExchangeValues fromEuros(double euros)
        throws java.rmi.RemoteException;
    public ExchangeValues fromYen(double yen)
        throws java.rmi.RemoteException;
}
```

JWSDP: Server

1. Write service endpoint interface
 - May need to write additional classes representing data structures
2. Write class implementing the interface
3. Compile classes
4. Create configuration files and run JWSDP tools to create web service
5. Deploy web service to Tomcat

JWSDP: Server

- ◆ Class *CurConImpl* contains methods, for example:

```
public ExchangeValues fromDollars(double dollars)
    throws java.rmi.RemoteException
{
    ExchangeValues ev = new ExchangeValues();
    ev.dollars = dollars;
    ev.euros = dollars * dollar2euro;
    ev.yen = dollars * dollar2yen;
    return ev;
}
```

JWSDP: Server

1. Write service endpoint interface
 - May need to write additional classes representing data structures
2. Write class implementing the interface
3. Compile classes
4. Create configuration files and run JWSDP tools to create web service
5. Deploy web service to Tomcat

JWSDP: Server

- ◆ Configuration file input to *wscmpile* to create server

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service
    name="HistoricCurrencyConverter"
    {
      targetNamespace="http://tempuri.org/wsdl"
      typeNamespace="http://tempuri.org/types"
      packageName="myCurCon">
      <interface name="myCurCon.CurCon" />
    }
  </service>
</configuration>
```

Namespaces
used in
WSDL
(normally,
unique URL's
at your
Web site)

JWSDP: Server

◆ Configuration file for web service

```
<?xml version="1.0" encoding="UTF-8"?>
<webServices
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
  version="1.0"
  targetNamespaceBase="http://tempuri.org/wsdl"
  typeNamespaceBase="http://tempuri.org/types"
```

JWSDP: Server

◆ Configuration file for web service

Context path { `urlPatternBase="/converter">`

Like
servlet
in
web.xml

```
<endpoint
  name="CurrConverter"
  displayName="Currency Converter"
  description=
    "Converts between dollars, euros, and yen."
  interface="myCurCon.CurCon"
  model="/WEB-INF/model.xml.gz"
  implementation="myCurCon.CurConImpl"/>
```

Like
servlet-mapping
in
web.xml

```
<endpointMapping
  endpointName="CurrConverter"
  urlPattern="/currency" />
```

```
</webServices>
```

JWSDP: Server

◆ Also need a minimal *web.xml*

```
<web-app
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <display-name>Historic Currency Converter</display-name>
  <description>
    This web service converts between three currencies using their
    exchange rates as of a fixed date.
  </description>
</web-app>
```

JWSDP: Server

- ◆ Run *jar* and *wsdeploy* to create a Web Archive (WAR) file *converter.war*
 - Name must match *urlPatternBase* value

JWSDP: Server

1. Write service endpoint interface
 - May need to write additional classes representing data structures
2. Write class implementing the interface
3. Compile classes
4. Create configuration files and run JWSDP tools to create web service
5. Deploy web service to Tomcat


JWSDP: Server

- ◆ Just copy *converter.war* to Tomcat *webapps* directory
 - May need to use Manager app to deploy
 - Enter *converter.war* in “WAR or Directory URL” text box

JWSDP: Server

◆ Testing success:

- Visit <http://localhost:8080/converter/currency>



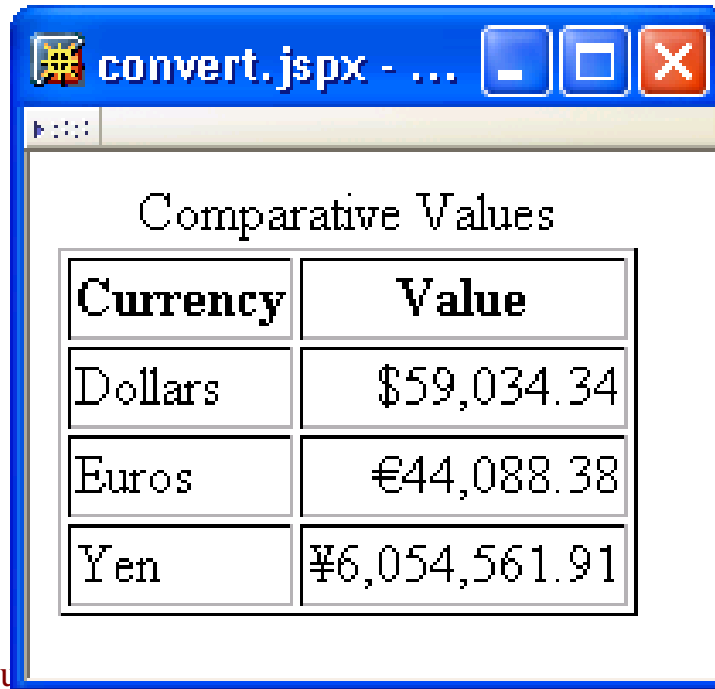
Web Services - Mozilla

Web Services

Port Name	Status	Information
CurrConverter	ACTIVE	Address: http://localhost:8080/converter/currency WSDL: http://localhost:8080/converter/currency?WSDL Port QName: {http://tempuri.org/wsdl} CurConPort Remote interface: myCurCon.CurCon Implementation class: myCurCon.CurConImpl Model: http://localhost:8080/converter/currency?model

JWSDP: Client

- ◆ Goal: write a JSP-based client
 - Input: currency and value
 - Output: table of equivalent values



The screenshot shows a web browser window with the title 'convert.jspx - ...'. The page content is titled 'Comparative Values' and displays a table with two columns: 'Currency' and 'Value'. The table contains three rows of data: Dollars (\$59,034.34), Euros (€44,088.38), and Yen (¥6,054,561.91).

Currency	Value
Dollars	\$59,034.34
Euros	€44,088.38
Yen	¥6,054,561.91

JWSDP: Client

- ◆ Configuration file input to *wscmpile* to create client

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl
    location=
      "http://localhost:8080/converter/currency?WSDL"
    packageName="myCurConClient" />
  </wsdl>
</configuration>
```

JWSDP: Client

- ◆ Directory structure (*wscmpile* generates content of *classes* and *src*)

```
webapps
  [[ other web application document base directories ]]
  ConverterClient
    WEB-INF
      classes
        myCurConClient
      src
        myCurConClient
```

JWSDP: Client

◆ Starting point for writing a client (if the web service author does not explain how):

- In the WSDL, find the *name* attribute of the *service* element

```
<service name="HistoricCurrencyConverter">
```

- Look in Java file with this name to see how to obtain a proxy object for the service

```
public interface HistoricCurrencyConverter
    extends javax.xml.rpc.Service {
    public myCurConClient CurCon getCurConPort ();
}
```

Data type of proxy object

Method called to obtain object

JWSDP: Client

◆ Obtaining the proxy object:

- Java file consisting of service name followed by *_Impl* defines a class implementing the proxy-generating interface
- Client code begins with method call on this class:

```
CurCon curCon =  
    (new HistoricCurrencyConverter_Impl()).getCurConPort();
```

JWSDP: Client

◆ Using the proxy object:

```
public interface CurCon extends java.rmi.Remote {
    public myCurConClient.ExchangeValues fromDollars(double double_1)
        throws java.rmi.RemoteException;
    public myCurConClient.ExchangeValues fromEuros(double double_1)
        throws java.rmi.RemoteException;
    public myCurConClient.ExchangeValues fromYen(double double_1)
        throws java.rmi.RemoteException;
}
```

JWSDP: Client

- ◆ Structs will be represented as JavaBeans classes, regardless of how they are defined on the server

```
public class ExchangeValues {
    protected double dollars;
    protected double euros;
    protected double yen;
    ...
    public double getDollars() {
        return dollars;
    }

    public void setDollars(double dollars) {
        this.dollars = dollars;
    }
    ...
}
```

JWSDP: Client

◆ Bean obtaining and calling proxy object:

```
public ExchangeValues getExValues() {
    ExchangeValues ev = null;
    CurCon curCon =
        (new HistoricCurrencyConverter_Impl()).getCurConPort();
    try {
        if (currency.equals("euros")) {
            ev = curCon.fromEuros(value);
        }
        else if (currency.equals("yen")) {
            ev = curCon.fromYen(value);
        }
        else {
            ev = curCon.fromDollars(value);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return ev;
}
```

JWSDP: Client

◆ JSP document using the bean:

Call to *getExValues()*

```
<c:set var="exvals" value="\${client.exValues}" />
...
<tr>
  <td>Euros</td>
  <td style="text-align:right">
    <fmt:formatNumber
      type="currency" currencySymbol="&#8364;">
      \${exvals.euros} Call to getEuros()
    </fmt:formatNumber>
  </td>
</tr>
```


WSDL Example

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<definitions name="HistoricCurrencyConverter"
```

```
  targetNamespace="http://tempuri.org/wsdl" ←
```

```
  xmlns:tns="http://tempuri.org/wsdl" ←
```

```
  xmlns="http://schemas.xmlsoap.org/wsdl/" ←
```

```
  xmlns:ns2="http://tempuri.org/types" ←
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" ←
```

```
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
```

Namespaces
specified in
config files

XML Schema NS

WSDL
namespaces

- ◆ Target namespace: namespace for names (*e.g.*, of operations) defined by the WSDL

WSDL Example

```
<types>
  <schema
    targetNamespace="http://tempuri.org/types"
    xmlns:tns="http://tempuri.org/types"
    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/"
    xmlns="http://www.w3.org/2001/XMLSchema">
    <import
      namespace="http://schemas.xmlsoap.org/soap/encoding/" />
    <complexType name="ExchangeValues">
      <sequence>
        <element name="dollars" type="double" />
        <element name="euros" type="double" />
        <element name="yen" type="double" />
      </sequence>
    </complexType>
  </schema>
</types>
```

Namespace for
data type definitions
(*ns2* in rest of document)

Defines struct
using XML
Schema

WSDL Example

Data type defined by
XML Schema

```
<message name="CurCon_fromDollars">
  <part name="double_1" type="xsd:double"/>
</message>
<message name="CurCon_fromDollarsResponse">
  <part name="result" type="ns2:ExchangeValues"/>
</message>
<message name="CurCon_fromEuros">
  <part name="double_1" type="xsd:double"/>
</message>
<message name="CurCon_fromEurosResponse">
  <part name="result" type="ns2:ExchangeValues"/>
</message>
<message name="CurCon_fromYen">
  <part name="double_1" type="xsd:double"/>
</message>
<message name="CurCon_fromYenResponse">
  <part name="result" type="ns2:ExchangeValues"/>
</message>
```

Input
messages
(parameter
lists)

Output
messages
(response
data types)

WSDL Example

```
<portType name="CurCon">
  <operation name="fromDollars" parameterOrder="double_1">
    <input message="tns:CurCon_fromDollars"/>
    <output message="tns:CurCon_fromDollarsResponse"/>
  </operation>
  <operation name="fromEuros" parameterOrder="double_1">
    <input message="tns:CurCon_fromEuros"/>
    <output message="tns:CurCon_fromEurosResponse"/>
  </operation>
  <operation name="fromYen" parameterOrder="double_1">
    <input message="tns:CurCon_fromYen"/>
    <output message="tns:CurCon_fromYenResponse"/>
  </operation>
</portType>
```

WSDL Example

```
<binding name="CurConBinding" type="tns:CurCon">
  <operation name="fromDollars">
    <input>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        use="encoded" namespace="http://tempuri.org/wsdl"/>
    </input>
    <output>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        use="encoded" namespace="http://tempuri.org/wsdl"/>
    </output>
    <soap:operation soapAction="" />
  </operation>
  ...
</operation>
<soap:binding
  transport="http://schemas.xmlsoap.org/soap/http"
  style="rpc" />
</binding>
```

Implement the operations using SOAP encoding of data structures and RPC (JWS DP defaults)

WSDL Example

```
<service name="HistoricCurrencyConverter">  
  <port name="CurConPort" binding="tns:CurConBinding">  
    <soap:address location="REPLACE_WITH_ACTUAL_URL"/>  
  </port>  
</service>  
</definitions>
```

Replaced by server
when WSDL is visited

WSDL Example

◆ Summary:

- *types* uses XML Schema to define data types
- *message* elements define parameter lists and return types using *types* and XML Schema
- portType defines abstract API for *operation*'s using *message*'s
- *binding* specifies how *message*'s will be communicated and *operation*'s called
- *service* associates URL with *binding*

XML Schema

- ◆ How do we send a Java *double* value to a web service using XML?
 - Is scientific notation allowed?
 - How large can the value be?
 - *Etc.*
- ◆ What if we want to send an object?
 - And what if the object contains references to other objects?

XML Schema

- ◆ XML Schema addresses such questions
 - Defines a number of simple data types, including
 - Range of allowed values
 - How values are represented as strings
 - Provides facilities for defining data structures in terms of simple types or other data structures
- ◆ Can also be used in place of XML DTD

XML Schema

◆ Built-in data types

Built-in type

```
<part name="latitude" type="xsd:decimal" />
```

- Types corresponding to Java primitive types: *boolean, byte, int, double, etc.*
 - String representations much as Java
 - ◆ Exception: can use *0* for *false*, *1* for *true*
 - No *char*; use *string* instead
- XML DTD types (*ID, CDATA, etc.*)

XML Schema

◆ Built-in data types

- *integer* and *decimal* (arbitrary precision)
- dates, times, and related subtypes
- URLs
- XML namespace qualified names
- binary data
- some restricted forms of the above, *e.g.*,
nonNegativeInteger

XML Schema

- ◆ XML Schema namespace defining built-in types is called the document namespace

`http://www.w3.org/2001/XMLSchema`

- Standard prefix for this namespace is *xsd*

XML Schema

TABLE 9.1: JAX-RPC mappings between supported Java classes and XML Schema built-in data types.

Java Class	XML Schema Type
String	string
java.math.BigDecimal	decimal
java.math.BigInteger	integer
java.util.Calendar	dateTime
java.util.Date	dateTime
java.xml.namespace.QName	QName
java.net.URI	anyURI

◆ Plus Java primitive types (*int*, *etc.*)

XML Schema

- ◆ Mapping from XML Schema data types to Java:
 - Primitives: one-for-one mapping
 - *date*, *time*, *dateTime*: map to *Calendar*
 - most others: map to *String*

XML Schema

- ◆ Elements in the document namespace can declare user-defined data types
- ◆ Two XML Schema data types:
 - Complex: requires markup to represent within an XML document
 - Simple: can be represented as character data

XML Schema

- ◆ User-defined data types are declared in the *types* element of a WSDL
 - Example: *ExchangeValue*
- ◆ In WSDL, user-defined types can be used
 - To define other data types within *types* element
 - To specify data types of parameters and return values in *message* elements

XML Schema

```
<types>
  <schema
    targetNamespace="http://tempuri.org/types"
    xmlns:tns="http://tempuri.org/types"
    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/"
    xmlns="http://www.w3.org/2001/XMLSchema">
    <import
      namespace="http://schemas.xmlsoap.org/soap/encoding/" />
    <complexType name="ExchangeValues">
      <sequence>
        <element name="dollars" type="double"/>
        <element name="euros" type="double"/>
        <element name="yen" type="double"/>
      </sequence>
    </complexType>
  </schema>
</types>
```

XML Schema

- ◆ An XML schema is markup that
 - Is written according to the XML Schema vocabulary
 - Defines an XML vocabulary
- ◆ A schema document is an XML document consisting entirely of an XML schema
- ◆ A document conforming with an XML schema vocabulary is call an instance of the schema

XML Schema

- ◆ Root element of the markup of an XML schema is *schema*
- ◆ Define data types with elements:
 - `complexType`
 - `simpleType`
- ◆ An XML schema can also define other vocabulary aspects (allowed elements, element content) that we won't cover

XML Schema

- ◆ One way to define simple types: restrict an existing simple base type

```
<simpleType name="memberType">  
  <restriction base="string"> Base type  
    <enumeration value="platinum" />  
    <enumeration value="preferred" />  
    <enumeration value="gold" />  
    <enumeration value="member" />  
  </restriction>  
</simpleType>
```

XML Schema

- ◆ Built-in types all have facets, that is, aspects that can be restricted
 - *enumeration* is a facet that applies to all built-in types except *boolean*
 - *length*, *minLength*, *maxLength* apply to string-like types (e.g., *string*, *QName*, *anyURI*)
 - *minInclusive*, *maxInclusive*, *minExclusive*, *maxExclusive* apply to numeric and time-oriented types
 - *totalDigits*, *fractionDigits* apply to numeric types

XML Schema

◆ Restricting multiple facets:

```
<simpleType name="priorityType">  
  <restriction base="int">  
    <minExclusive value="10" />  
    <maxInclusive value="100" />  
  </restriction>  
</simpleType>
```

XML Schema

◆ *pattern* facet

- applies to most types (except a few DTD)
- specifies regular expression

```
<simpleType name="phoneNumType">  
  <restriction base="string">  
    <pattern value="\d{3}-\d{3}-\d{4}" />  
  </restriction>  
</simpleType>
```

XML Schema

◆ Other simple types

- Union: combine two or more types

```
<simpleType name="oddType">  
  <union memberTypes="memberType phoneNumType" />  
</simpleType>
```

- Lists of values of simple type

```
<simpleType name="intList">  
  <list itemType="int" />  
</simpleType>
```


XML Schema

◆ Complex type

■ Defined in an XML schema

```
<complexType name="ExchangeValues">  
  <sequence>  
    <element name="dollars" type="double"/>  
    <element name="euros" type="double"/>  
    <element name="yen" type="double"/>  
  </sequence>  
</complexType>
```

■ Used in an instance document

```
<anExchangeValue xsi:type="ExchangeValues">  
  <dollars>1.0</dollars>  
  <euros>0.746826</euros>  
  <yen>102.56</yen>  
</anExchangeValue>
```

XML Schema

◆ Complex type can be used in place of XML DTD content specification

- sequence element is equivalent to , operator in DTD

```
<!ELEMENT anExchangeValue (dollars, euros, yen)>
```

XML Schema

```
<complexType name="Arguments">
  <sequence>
    <element name="optArg" type="string"
      minOccurs="0" />
  </sequence>
</complexType>
```

```
<complexType name="ExchangeValues">
  <all>
    <element name="dollars" type="double"/>
    <element name="euros" type="double"/>
    <element name="yen" type="double"/>
  </all>
</complexType>
```

XML Schema

◆ Instance namespace

`http://www.w3.org/2001/XMLSchema-instance`

- Normally associated with prefix `xsi`

◆ Used within instance documents to

- define null-valued elements

```
<optArg xsi:nil="true"></optArg>
```

- define data types

```
<latitude xsi:type="xsd:decimal">40.28</latitude>  
<longitude xsi:type="xsd:decimal">-79.49</longitude>
```

SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:ns0="http://tempuri.org/wsdl"
  xmlns:ns1="http://tempuri.org/types"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <env:Body>
    <ns0:fromDollars>
      <double_1 xsi:type="xsd:double">1.0</double_1>
    </ns0:fromDollars>
  </env:Body>
</env:Envelope>
```

SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:ns0="http://tempuri.org/types"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <env:Body>
    <ans1:fromDollarsResponse
      xmlns:ans1="http://tempuri.org/wsdl">
      <result href="#ID1"/>
    </ans1:fromDollarsResponse>
    <ns0:ExchangeValues id="ID1" xsi:type="ns0:ExchangeValues">
      <dollars xsi:type="xsd:double">1.0</dollars>
      <euros xsi:type="xsd:double">0.746826</euros>
      <yen xsi:type="xsd:double">102.56</yen>
    </ns0:ExchangeValues>
  </env:Body>
</env:Envelope>
```

SOAP

◆ Alternate form

```
<ans1:fromDollarsResponse
  xmlns:ans1="http://tempuri.org/wsdl">
  <result xsi:type="ns0:ExchangeValues">
    <dollars xsi:type="xsd:double">1.0</dollars>
    <euros xsi:type="xsd:double">0.746826</euros>
    <yen xsi:type="xsd:double">102.56</yen>
  </result>
</ans1:fromDollarsResponse>
```

SOAP

◆ SOAP encoding of arrays in WSDL

```
<import
  namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="ArrayOfdouble">
  <complexContent>
    <restriction base="soap11-enc:Array">
      <attribute ref="soap11-enc:arrayType"
        wsdl:arrayType="double []" />
    </restriction>
  </complexContent>
</complexType>

<message name="CurCon_fromDollarsResponse">
  <part name="result" type="ns2:ArrayOfdouble"/></message>
```


SOAP

◆ Array in SOAP document

```
<env:Body>
  <ans1:fromDollarsResponse
    xmlns:ans1="http://tempuri.org/wsdl">
    <result href="#ID1"/>
  </ans1:fromDollarsResponse>
  <ns0:ArrayOfdouble id="ID1"
    xsi:type="enc:Array"
    enc:arrayType="xsd:double[3]">
    <item xsi:type="xsd:double">1.0</item>
    <item xsi:type="xsd:double">0.746826</item>
    <item xsi:type="xsd:double">102.56</item>
  </ns0:ArrayOfdouble>
</env:Body>
```

SOAP

- ◆ If SOAP is sent via HTTP, request must include SOAPAction header field
 - Either empty or a URI
 - Can be used to pass operation rather than embedding in body of message

Web Services Technologies

- ◆ Other implementation of JAX-RPC and/or Java-based web services
 - Apache Axis
 - IBM WebSphere
- ◆ Microsoft support for web services: .NET
- ◆ PHP also has web services tools

Web Services Technologies

- ◆ Universal Discovery, Description, and Integration (UDDI)
 - Technology for creating directories of web services
- ◆ Web Services-Interoperability Organization (WS-I) Basic Profile
 - Specification of how web services should be used to enhance interoperability
 - Must use XML Schema and literal encoding (rather than SOAP encoding)