Web Services

© Leonidas Fegaras
Middleware

- Facilitate the interaction among applications across heterogeneous computing platforms
  - they offer programming abstractions that hide complexity
- Remote Procedure Call (RPC)
  - hides communication details behind procedure calls
  - Serialization: marshaling/unmarshaling of values
  - Stubs
- Component-oriented programming
  - CORBA, RMI, EJB
  - COM
  - .NET
- Next generation heterogeneous distributed applications:
  - *Service-oriented Computing*
    - the basic element is a service
**What are Web Services?**

- Services available via the Web
  - Mostly for application-to-application communication
  - Enables business-to-business transactions
- Note:
  - web sites are for humans
  - web services are for software applications
- Examples:
  - stock quote service
  - weather service
  - map service
  - web search service

**Diagram:**

- **Client**
  - Service request: get IBM stock price
  - Service response: 123.78

- **Server**
Why Web Services?

- They are platform and language independent
- They are appropriate for loosely coupled systems
  - don't need to have any prior knowledge of Web Services
- Based on XML
  - SOAP: Simple Object Access Protocol
  - WSDL: Web Service Description Language
  - UDDI: Universal Description, Discovery, and Integration
- Typical Web Service interaction:
SOAP

- A lightweight protocol for exchanging information in a distributed, heterogeneous environment
- Enables cross-platform interoperability
  - programming language neutral
  - hardware independent
  - protocol independent
- Works over existing Internet infrastructure
  - on top of HTTP
- Builds on XML standards
- Defines
  - message format
  - data encoding
  - headers for sending messages & receiving responses
A SOAP Request

- SOAP requests are HTTP POST requests
  
  POST /DictService/DictService.asmx HTTP/1.1
  Host: services.aonaware.com
  Content-Type: text/xml; charset=utf-8
  Content-Length: length
  SOAPAction: "http://services.aonaware.com/webservices/DefineInDict"

  
  <?xml version="1.0" encoding="utf-8"?>
  <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
      <DefineInDict xmlns="http://services.aonaware.com/webservices/"
        >
        <dictId>wn</dictId>
        <word>computer</word>
      </DefineInDict>
    </soap:Body>
  </soap:Envelope>
Message Layout

- **SOAP Message**
  - **Headers**
  - **SOAP Envelope**
    - **SOAP Header**
      - **Headers**
    - **SOAP Body**
      - **Message Name & Data**

- **Complete SOAP message**
- **Protocol binding headers**
- **Encloses payload**
- **Encloses headers**
- **Individual headers**
- **Contains SOAP message name**
- **XML-encoded SOAP message name & data**
Message Format

- The SOAP schema can be defined in an XML document

```xml
<?xml version="1.0"?>
<soap:Envelope ...>
    <soap:Header ...>
        ...
    </soap:Header>
    <soap:Body>
        <add xmlns="http://services.aonaware.com/webservices/">
            <left>1</left>
            <right>2</right>
        </add>
    </soap:Body>
</soap:Envelope>
```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <DefineInDictResponse xmlns="http://services.aonaware.com/webservices/">
            <DefineInDictResult>
                <Word>computer</Word>
                <Definitions>
                    <Definition>
                        <Word>computer</Word>
                        <Dictionary id="wn" />
                        <WordDefinition>computer n 1: a machine for performing calculations automatically [syn: {computing machine}, {computing device}, {data processor}, {electronic computer}, {information processing system}] 2: an expert at calculation (or at operating calculating machines) [syn: {calculator}, {reckoner}, {figurer}, {estimator}]</WordDefinition>
                    </Definition>
                </Definitions>
            </DefineInDictResult>
        </DefineInDictResponse>
    </soap:Body>
</soap:Envelope>
WSDL: Web Services Description Language

- XML Schema for describing Web Services
  - Service interface definition
    - abstract semantics for Web Service
  - Service implementation definition
    - concrete end points and network addresses where Web Service can be invoked
- Used primarily to describe SOAP services
- Describes four critical pieces of data:
  - Interface information describing all publicly available functions
  - Data type information for all message requests and message responses
  - Binding information about the transport protocol to be used
  - Address information for locating the specified service
Main Structure

<definition namespace = ...>
<type> XML schema types </type>
<message> … </message>
<port> a set of operations </port>
<binding> communication protocols </binding>
<service> a list of binding and ports </service>
<definition>
WSDL Parts

- `<types>` define types used in message declaration
  - XML Schema must be supported by any vendor of WSDL conformant products
- The `<message>` element defines the data elements of an operation
  - each message may consist of one or more parts
    - similar to the parameters of a function call in a traditional programming language
- The `<portType>` defines a web service, the operations that can be performed, and the messages that are involved
  - defines the connection point to a web service, an instance of `<portType>`.
  - similar to a function library in a traditional programming language
  - each operation can be compared to a function in a traditional programming language
Example: HelloService.wsdl

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
    targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <message name="SayHelloRequest">
        <part name="firstName" type="xsd:string"/>
    </message>
    <message name="SayHelloResponse">
        <part name="greeting" type="xsd:string"/>
    </message>
    <portType name="Hello_PortType">
        <operation name="sayHello">
            <input message="tns:SayHelloRequest"/>
            <output message="tns:SayHelloResponse"/>
        </operation>
    </portType>
</definitions>
```
Example (cont.)

```xml
<binding name="Hello_Binding" type="tns:Hello_PortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHello">
        <soap:operation soapAction="sayHello"/>
        <input>
            <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="urn:examples:helloservice"
                use="encoded"/>
        </input>
        <output>
            <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="urn:examples:helloservice"
                use="encoded"/>
        </output>
    </operation>
</binding>
```
Example (cont.)

<service name="Hello_Service">  
  <documentation>WSDL File for HelloService</documentation>  
  <port binding="tns:Hello_Binding" name="Hello_Port">  
    <soap:address  
      location="http://localhost:8080/soap/servlet/rpcrouter"/>  
  </port>  
</service>

</definitions>
Another Example

<s:element name="DefineInDict">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="dictId" type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="word" type="s:string"/>
        </s:sequence>
    </s:complexType>
</s:element>

<s:element name="DefineInDictResponse">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="DefineInDictResult" type="tns:WordDefinition"/>
        </s:sequence>
    </s:complexType>
</s:element>

<wsdl:operation name="DefineInDict">
    <documentation>Define given word, returning definitions from specified dictionary</documentation>
    <wsdl:input message="tns:DefineInDictSoapGetIn"/>
    <wsdl:output message="tns:DefineInDictSoapGetOut"/>
</wsdl:operation>
UDDI

- **UDDI** = Universal Description, Discovery, and Integration
- **Industry initiative to address discovery**
  - a registration database for Web Services
- **Specifications**
  - schema for service providers and descriptions
  - API for publishing and searching
  - developed on industry standards (XML, HTTP, TCP/IP, SOAP)
  - applies to both XML and non-XML services
- **Implementation**
  - public and private instances of specification
Web Services and Sun's J2EE

- JWSDP: Sun Java Web Services Developer Pack
  - JAXP: Java API for XML processing
  - JAXB: Java architecture for XML binding
    - marshaling/unmarshaling: mapping Java objects to XML
  - JAXM: Java API for XML (SOAP) messaging
    - SAAJ: SOAP with attachments API for Java
  - JAX-RPC: Java API for XML-based RPC
  - JAXR: Java API for XML registries (UDDI)
    - Java WSDR registry server
JAXB

- JAXB uses the XML Schema to convert Java objects to XML and vice versa
  - generates a set of Java classes from the XML Schema
  - also generates javadoc
- Based on Java Beans
  - For each element with tagname “tag”, generates
    - a getter getTag()
    - a setter method setTag(value)
  
```
List<Gradstudent> sl = dept.getGradstudent();
Iterator<Gradstudent> l = sl.iterator();
while (l.hasNext()) {
    Gradstudent s = l.next();
    System.out.println(s.getName().getLastname());
}
```
- Provides the classes Marshaller and Unmarshaller
JAXM

MessageFactory msgFactory = MessageFactory.newInstance();
SOAPMessage msg = msgFactory.createMessage();
SOAPPart soapPart = msg.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPHeader header = envelope.getHeader();
SOAPBody body = envelope.getBody();

header.detachNode();
Name methodName
SOAPBodyElement getHoroscope = body.addBodyElement(methodName);
Name dobParamName = envelope.createName("DateOfBirth");
SOAPElement dobParam = getHoroscope.addChildElement(dobParamName);
dobParam/addTextNode("05/03/1964");
Name colorParamName = envelope.createName("FavoriteColor");
SOAPElement colorParam = getHoroscope/addChildElement(colorParamName);
colorParam/addTextNode("Purple");

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
SOAPMessage response = conn.call(msg, endPoint);
conn.close();
System.out.println(“Received reply”);
SOAPPart soapRespPart = response.getSOAPPart();
SOAPEnvelope soapRespEnv = soapRespPart.getEnvelope();
SOAPBody soapRespBody = soapRespEnv.getBody();
Iterator it = soapRespBody.getChildElements();
SOAPBodyElement bodyElement = (SOAPBodyElement)it.next();
String horoscope = bodyElement.getValue();
System.out.println(“Horoscope returned: ” + horoscope);
JAX-RPC

- The generated stubs client model
  - Stub and tie classes are generated from WSDL
- Dynamic proxy
  - acquires the signature of the service call at run-time and constructs a proxy that can then be used to call the service