CS144 Notes: Web Services

What is a Web service?

• **Q: What is the Web?**
  
  – Web sites and Web pages
    * static textual pages created for human consumption.

• **Q: If we want to build a service that integrates services from expedia, weather, and frommers --- given departure and destination cities, provide maps, weather and tourist destinations --- what needs to be done?**
  
  – Remarks:
    * "screen scraping"
    * extracting data from html pages is labor intensive and fragile
    * how can we make them easy to "integrate" or "mash-up"?

• **"Web service" is the Web for applications**
  
  – Each service provides a particular functionality
    * travel booking, dictionary, weather forecast, ...

  – Each service exchanges request and result through a well-defined standard
    * makes it possible to build web services independently of the platform

  – Each service "describes" its interface using a well-defined standard
    * makes it easy to build the "glue" between services

<show WSDL and SOAP examples>

http://www.webservicex.net

http://www.soapclient.com/soaptest.html

– Remark:
  * all output is formatted in XML: no more screen scraping
  * we can "invoke" a service without writing any software code due to WSDL.
• Important standards

1. SOAP (Simple Object Access Protocol)
   - Standard for making "request" and receiving "response"

2. WSDL (Web Service Definition Language)
   - Standard for "describing" a particular Web service
   - This should provide (1) location (2) list of "operations" and (3) the input and output parameters

Web Service example

• Temperature conversion service
  - Fahrenheit -> Celsius

• In Java

```java
public class Converter {
    public double fahrenheitToCelsius(double fahrenheit) {
        // convert Fahrenheit to Celsius
        return (fahrenheit - 32.0)*5.0/9.0;
    }
}
```

SOAP

• message exchange standard for Web services
  - standard for "method invocation"

  - e.g., calling "fahrenheitCelsius" with parameter 80

  - assume the service is available at http://oak.cs.ucla.edu/cs144/Converter

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body>
    <fahrenheitToCelsius>
      <fahrenheit>80</fahrenheit>
    </fahrenheitToCelsius>
  </soap:Body>
</soap:Envelope>
```

  - The root element is <Envelope>
Any soap message should be wrapped in `<Envelope>` element

- `<Envelope>` has a child element `<Body>`

  * The main message should be wrapped in `<Body>` element

  - e.g., response from "Converter"

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body>
    <fahrenheitToCelsiusResponse>
      <fahrenheitToCelsiusReturn>26.67</fahrenheitToCelsiusReturn>
    </fahrenheitToCelsiusResponse>
  </soap:Body>
</soap:Envelope>
```

- Skeleton SOAP message

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Header> ... </soap:Header>  
  <soap:Body>
    ...  
    <soap:Fault> ... </soap:Fault>  
    ...  
  </soap:Body>
</soap:Envelope>
```

- Remarks:
  * Header
    - optional first subelement of `<Envelope>`
    - contains optional information about the message

  * Body
    - main message

  * Fault
    - optional subelement of `<Body>` to deliver an error message
    - may contain `<faultcode>`, `<faultstring>`, `<detail>`, ...

- Remark: SOAP itself does not specify the exact format of `<Body>`. The exact format of `<Body>` is defined by WSDL. The above format is just a common convention. It could well be
...<soap:Body><Celsius>26.67</Celsius></soap:Body>...

**WSDL**

- **Standard for describing a particular Web service**

- **Q: What information needs to be given in a Web service description?**
  - The location of a service: appears in `<service>`
    - List of "functions": appears in `<portType>`
    - Function parameters: appears in `<types>`

```
<service>: **the location**
  ^
|<binding>: protocol and encoding scheme
  ^
|<portType>: **the list of operations and the parameters**
  ^
|<message>: the message structure
  ^
|<types>: **the format of input/output parameters**
```

<give students time to go over the WSDL example>
e.g.,

```xml
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://oak.cs.ucla.edu/cs144/Converter"
  xmlns:tn="http://oak.cs.ucla.edu/cs144/Converter">
  <!-- types element specifies the "type" of input/output parameters -->
  <types>
    <xs:schema targetNamespace="http://oak.cs.ucla.edu/cs144/Converter">
      <!-- fahrenheitToCelsius will be the root request element -->
      <xs:element name="fahrenheitToCelsius">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="fahrenheit" type="xs:double"/>
          </xs:sequence>
        </xs:complexType>
```
<!-- fahrenheitToCelsiusResponse is the root response element -->
<xs:element name="fahrenheitToCelsiusResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="fahrenheitToCelsiusReturn" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

<!-- message element specifies request and response message format 
message element may consist of multiple parts like header, body, 
faultcode, etc-->
<message name="convertRequest">
  <part name="requestBody" element="tn:fahrenheitToCelsius"/>
</message>
<message name="convertResponse">
  <part name="responseBody" element="tn:fahrenheitToCelsiusResponse"/>
</message>

<!-- portType element specifies the available operations (= methods) 
and the associated input/output message format -->
<portType name="ConverterPortType">
  <operation name="fahrenheitToCelsius">
    <input message="tn:convertRequest"/>
    <output message="tn:convertResponse"/>
  </operation>
</portType>

<!-- binding element specifies the transfer protocol 
and the message MIME encoding of the service 
every operation specified here should have a corresponding 
one operation in the associated portType -->
<binding type="tn:ConverterPortType" name="ConverterBinding">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="fahrenheitToCelsius">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>

<!-- service element specifies the URL where the service is available -->
<service name="ConverterService">
  <port binding="tn:ConverterBinding" name="Converter">
    <soap:address location="http://oak.cs.ucla.edu/cs144/Converter"/>
- WSDL:

  * everything should be wrapped in `<definitions>` ... `</definitions>`
  * `<definition>` has 5 children

    - `<types>`
    - `<message>`
    - `<portTypes>`
    - `<binding>`
    - `<service>`

  * of which the following three contain the "core" information

    - `<service>`: service URL - location attr of address element
    - `<portType>`: list of functions - operation element
    - `<types>`: XML type definitions for message format

    - `<service>` is associated with `<portType>` through `<binding>`
      - `<binding>` makes it possible to use non-HTTP protocol and different encoding
        * `use="literal"` or "encoded".
          * "encoded" means special encoding is used (like mime64)

        * `style="rpc"` or "document".
        * "rpc" implies that "request and response" model and that all requests are wrapped in an element with the name of the invoked operation.

    - `<portType>` is associated with `<types>` through `<message>`
      - `<message>` allows complex message formatting with multiple parts
        * like header, body, etc.
        * in many cases, the actual "message format" is defined as a type
* Notes on Namespace
  • most of the elements should belong to the WSDL namespace
  • Elements in <types> should belong to XML Schema namespace
  • SOAP binding related elements should belong to SOAP namespace
  • User defined types and elements should belong to its own namespace

REST (Representational State Transfer)

  • SOAP is good, but it is too complicated
    - too much additional layers
    - very difficult to read
    - difficult to read and understand messages and description

  • REST
    - Instead of complex SOAP request, request is encoded in URL (in most cases)
      e.g., Yahoo Map
    - http://oak.cs.ucla.edu/cs144/Converter?method=fahrenheitToCelsius&fahrenheit=80
    - response is typically encoded either in simple XML (without additional messaging layer) or in JSON (more about JSON later)

<xml version="1.0">
<Celsius>26.67</Celsius>

• REST web services are:
  - Lightweight - not a lot of extra xml markup
  - Human Readable Results
  - Easy to build - no toolkits required

• SOAP also has some advantages:
  - Rigid - type checking, adheres to a contract
  - Automatic development tools

• REST interface tends to be more popular
  - easier to learn and use using simple tools
• Q: How is a Web Service different from RPC?

A: - XML based
  - typically use HTTP
  -> less compatibility issues

Distributed transactions

• Example: a travel site that arranges both flight and hotel
  - communicates with banks, hotels, and airlines through Web services
  - the user wants to book the flight F and hotel H using credit card C

  - Q: How should the site handle the booking?

  - Q: What if the credit card authorizations fail?
    What if the flight is no longer available?
    What if the hotel is no longer available?

  - Remark: No one can "commit" unless everyone else commits.

Two-phase commit

• Before commit, ask everyone whether they are ready

  - PREPARE -> VOTE-COMMIT/VOTE-ABORT
  - Note: Anyone who said ready cannot say otherwise later

• If everyone says yes, commit

  - if anyone says no, abort
- Q: Any potential problem of two phase commit?

- Q: Where should we add timeout to avoid indefinite wait?

- Remark: Do not get confused with two-phase locking

**Asynchronous transaction**

- Q: What if one participant is very slow?
  

- Q: What does two-phase commit mean in this scenario?

- Q: How can we let each participant go ahead without waiting for the slow one?
  
  - Q: What does Starbucks do?
• **Asynchronous transaction**
  − Each participant "commits" whenever he is done and moves ahead
    * Transaction = sequence of smaller transactions by each participant
  − The entire transaction is done when every participant commits
  − No coordinated wait and synchronous commit

• **Q: What if the coffee machine breaks down after customer paid?**

  − Compensating transaction
    * a transaction that "rolls back" a committed transaction
  − The coordinator should keep track of the "dependency" of transactions
    * together with their compensating transactions
  − If any transaction aborts, run compensating transaction for all committed transactions

• **Q: When should we use two-phase commit/asynchronous transaction?**

  − importance of individual commit guarantee
  − duration of individual transaction
  − probability of abort

• **Remark**

  − There exist a number of standards for distributed transactions on Web Services
    * WS-Coordination, WS-AtomicTransaction, ...
  − Popular Web Application Servers support some of them
    * JBoss, BEA WebLogic, IBM WebSphere, ...