Web Services

Web service: “any service available on the Web that has been designed for consumption by programs, independent of the technology being used”

Two primary camps

- **REST** (sometimes just “HTTP”)
  - “Representational State Transfer”
  - Exchanging documents
  - Many HTTP actions

- **SOAP**
  - Exchanges messages in XML
  - Mainly HTTP POST
REST

• Use all of HTTP for sensible document/data handling:
  – POST – tell the server to make new document
  – HEAD – request the server to send document metadata
  – GET – retrieve document from the server (no state change)
  – PUT – update document on the server
  – DELETE – delete document on the server

• Requests – all state info is in the URL
• Response format?

• In practice – often GET for everything
  – Works in browser
  – But violates “no side effects” rule

SOAP

• Originally “Simple Object Access Protocol”
• Two views
  – 1. Exchanging messages
  – 2. Performing “remote procedure calls” (RPC)
• Request
  – Mostly POST (but need not be just HTTP!)
  – A complex XML document
• Response format: XML
• What parameters/functions are legal??
  • A “WSDL” file specifies legal tags that can be used
  • Useful if really need standardization and checking
  • Heavyweight, so not widely used!
REST vs SOAP

**REST**

Request method: GET
Request data: GET parameters
Example: wunderground.com?p1=zip&p2=time&…

Returned data: XML or JSON or text.

**SOAP**

Request method: POST
Request data: sent in POST as XML
Example: myB2Bsite.com
// send an XML document along

Returned data: XML

---

**REST Example #1**

- Get the weather from wunderground.com
- http://www.wunderground.com/weather/api/d/docs

```
GET http://api.wunderground.com/api/21401.xml
```

```
XML or JSON or text.
```

---

Standard Request URL Format

```
GET http://api.wunderground.com/api/21401.xml
```

---

**conditions or forecast**

(or)

**conditions/forecast**

(often settings)

21401.xml
(Ex 1) Weather XML Data

From: http://api.wunderground.com/api/XXX/conditions/forecast/q/21409.xml

```xml
<current_observation>
  <display_location>
    <full>Annapolis, MD</full>
    <city>Annapolis</city>
    <state>MD</state>
    <state_name>Maryland</state_name>
    <country>US</country>
    <country_iso3166>US</country_iso3166>
    <zip>21409</zip>
    <latitude>39.02930832</latitude>
    <longitude>-76.43528748</longitude>
    <elevation>6.00000000</elevation>
  </display_location>
  <weather>Clear</weather>
  <temperature_string>56.1 F (13.4 C)</temperature_string>
  <temp_f>56.1</temp_f>
...```

Problem: Same-origin policy

- Browser security restriction: AJAX from usna.edu can ONLY request content from usna.edu
  – Not wunderground.com!

- So how to get content from the web service?
  1. Use a workaround directly!
  2. Use site-specific workaround!
  3. Only make requests to our same domain but…
You have a lot of content on the page, and want to localize it for the user. One easy way is to provide the weather!

Let's paste in your local weather using the wunderground.com web service, ask for a zipcode, and then use XSLT to transform the result into some nice XHTML.

We'll paste the result below.

Type your zip code: <input type="text" id="zipcode" />
<input type="button" value="Get weather!" onclick="getWeather()" />

This is where the transformed XML in XHTML form will appear.

```
(Ex 1) transform.js

var xml = null;
var xsl = null;

function getWeather() {
    var zip = $('#zipcode').val();
    var url = "wunderground.pl?zipcode=" + zip;
    transform(url, "wunderground.xsl");
}

function transform (xmlFileName, xslFileName) {
    // Make two ajax calls
    ...
    // Call insertXML when ajax returns with (1) XML, and (2) XSLT
    insertXML();
}

function insertXML() {
    // same from before!!
    ...
    o.parentNode.replaceChild(n, o);
}
(Ex 1) wunderground.pl

#!/usr/bin/perl
use CGI "standard"; use strict;

# Required libraries to make an HTTP request from Perl.
use LWP::Simple "!head";
use LWP::UserAgent;
use HTTP::Request;
use HTTP::Response;

# We want to forward the XML from wunderground.com to the client.
print "Content-type: text/xml\n\n";

# Construct URL to get the weather
my $zip = param("zipcode");
my $URL = "http://api.wunderground.com/api/XXXX/conditions/forecast/q/$zip.xml";

# Get the XML document and send it back to the client (unchanged!)
my $contents = get($URL);
print $contents;

(Ex 1) wunderground.xsl

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <html>
      <head><title>Current weather in: <xsl:value-of select="/response/termsofService"/></title></head>
      <body>
        <div id="planet">
          <h1>Current weather in: <xsl:value-of select="/response/current_observation/display_location/full"/></h1>
          <xsl:apply-templates select="/response/current_observation"/>
          <xsl:apply-templates select="/response/forecast/simpleforecast"/>
        </div>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
REST Example #2: Flickr

• Search flickr.com and show photos on your page.

• Example request:
  https://api.flickr.com/services/rest/?method=flickr.photos.search&name=value&per_page=5&tags=tigers &api_key=XYZ_MYKEY_XYZ

• flickr.pl??
  – Same idea as wunderground.pl
  – But using URL like the one above

Online API: http://www.flickr.com/services/api/flickr.photos.search.html

(Ex 2) XML Data Returned

```xml
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
  <photos page="1" pages="20668" perpage="5" total="103339">
    <photo id="2944625312" owner="41086422@N00" secret="1975114cb7" server="3057" farm="4" title="Bad Mascot" ispublic="1" isfriend="0" isfamily="0" />
    <photo id="2944368362" owner="29542413@N07" secret="0f3f076cd1" server="3020" farm="4" title="_MG_3447" />
    <photo id="2943510303" owner="29542413@N07" secret="7c04e22d9b" server="3283" farm="4" title="_MG_3462" />
    <photo id="2944369890" owner="29542413@N07" secret="fe9271a3b0" server="3035" farm="4" title="_MG_3454" />
    <photo id="2944370484" owner="29542413@N07" secret="451a349bb0" server="3184" farm="4" title="_MG_3456" />
  </photos>
</rsp>
```
(Ex 2) photos.html

<html>
  <head>
    <title>XSLT Example with Web services</title>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
    <script type="text/javascript" src="transform2.js"></script>
  </head>

  <body>
    <p>Lets now use SOAP to access photos from flickr!</p>
    <p>Tags to search for:
      <input type="text" id="tags" />
      <input type="button" value="Get photos!" onclick="getPhotos()" />
    </p>

    <div id="planet">
      <h2>This is where the transformed XML in XHTML form will appear.</h2>
    </div>
  </body>
</html>

(Ex 2) flickr.xsl

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <html><head><title>Flickr test</title></head>
    <body>
      <div id="planet">
        <xsl:variable name="var_total" select="/rsp/photos/@total" />
        <p>There were <xsl:value-of select="$var_total" /> results. Here are just some: </p>
        <ul>
          <xsl:apply-template select="/rsp/photos/photo" />
        </ul>
      </div>
    </body>
  </xsl:template>
</xsl:stylesheet>
(Ex 2) transform2.js

```javascript
var xml = null;
var xsl = null;

function getPhotos() {
    var tags = $('tags').val();
    var url = "flickr.pl?tags=" + tags;
    transform("flickr.xsl", url);
}

function transform (xmlFileName, xslFileName) {
    // Make two ajax calls
    ...
    // Call insertXML when ajax returns with (1) XML, and (2) XSLT
    insertXML();
}

function insertXML() {
    // same from before!!
    ...
    o.parentNode.replaceChild(n, o);
}
```

More XML: What do the tags mean?

Your program finds XML from usna.edu that includes…

```xml
<person>
    <title>Captain</title>
    ...
</person>
```

You also find XML from bn.com that includes…

```xml
<object>
    <title>Captain</title>
    ...
</object>
```

Does <title> mean the same thing in both places?
Solution: XML namespaces
(Note: using fake namespace URIs!)

Your program finds XML from usna.edu that includes…

```xml
<mil:person xmlns:mil="http://milstandards.gov/peoplev2.3">
  <mil:title>Captain</mil:title>
  ...
</mil:person>
```

You also find XML from bn.com that includes…

```xml
<upc:object xmlns:upc="http://nist.gov/b2bcorev3.1">
  <mil:title>Captain</mil:title>
  ...
</upc:object>
```

Does `<title>` mean the same thing in both places?

XML with namespaces

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://realPrefixRemovedForThisDemo.org#"
  xmlns="http://purl.org/rss/1.0/
  xmlns:dc="http://purl.org/dc/elements/1.1/" >

  <channel rdf:about="http://web2.0thebook.org/channel.rss">
    <title>Planet web2.0thebook</title>
    <link>http://web2.0thebook.org</link>
    <description>Aggregated content relevant to the upcoming book "Professional Web 2.0 Programming".</description>
  </channel>
  ...
</rdf:RDF>
```
XML with namespaces

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://realPrefixRemovedForThisDemo.org#" xmlns="http://purl.org/rss/1.0/"
xmlns:dc="http://purl.org/dc/elements/1.1/"
><channel rdf:about="http://web2.0thebook.org/channel.rss">
<title>Planet web2.0thebook</title>
<link>http://web2.0thebook.org</link>
<description>Aggregated content relevant to the upcoming book “Professional Web 2.0 Programming”</description>
</channel>
<item rdf:about="http://www.orbeon.com/blog/2006/06/13/firebug-a-must-have-firefox-extension-for-web-developers/">
<title>XForms Everywhere » FireBug: A Must-Have Firefox Extension for Web Developers</title>
<link>http://www.orbeon.com/blog/2006/06/13/firebug-a-must-have-firefox-extension-for-web-developers/</link>
<description>Alessandro Vernet recommends FireBug, an absolute godsend, the greatest web developer extension out there! It's absolute brilliance. It's also the most recommended extension... </description>
<dc:creator>evlist</dc:creator>
<dc:date>2006-06-15T05:56:16Z</dc:date>
<dc:subject>ajax debugger dom firefox javascript tools web2.0thebook webdev</dc:subject>
</item>
<item rdf:about="http://eric.vandervlist.com/blog/2504_Web_2.0_at_XML_Prague.item">
<title>Web 2.0 at Prague</title>
<link>http://eric.vandervlist.com/blog/2504_Web_2.0_at_XML_Prague.item</link>
<description>Eric van der Vlist will do a presentation about Web 2.0 at XML Prague 2006. </description>
</item>
</rdf:RDF>

Which mean the same thing?

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://realPrefixRemovedForThisDemo.org#" xmlns="http://purl.org/rss/1.0/"
xmlns:dc="http://purl.org/dc/elements/1.1/"
><channel rdf:about="http://web2.0thebook.org/channel.rss">
<title>Planet web2.0thebook</title>
<link>http://web2.0thebook.org</link>
</channel>
<item rdf:about="http://www.orbeon.com/blog/2006/06/13/firebug-a-must-have-firefox-extension-for-web-developers/">
<title>XForms Everywhere » FireBug: A Must-Have Firefox Extension for Web Developers</title>
<link>http://www.orbeon.com/blog/2006/06/13/firebug-a-must-have-firefox-extension-for-web-developers/</link>
<description>Alessandro Vernet recommends FireBug, an absolute godsend, the greatest web developer extension out there! It's absolute brilliance. It's also the most recommended extension... </description>
<dc:creator>evlist</dc:creator>
<dc:date>2006-06-15T05:56:16Z</dc:date>
<dc:subject>ajax debugger dom firefox javascript tools web2.0thebook webdev</dc:subject>
</item>
<item rdf:about="http://eric.vandervlist.com/blog/2504_Web_2.0_at_XML_Prague.item">
<title>Web 2.0 at Prague</title>
<link>http://eric.vandervlist.com/blog/2504_Web_2.0_at_XML_Prague.item</link>
<description>Eric van der Vlist will do a presentation about Web 2.0 at XML Prague 2006. </description>
</item>
</rdf:RDF>
XPath with Namespaces

<booklist listtitle="Science Fiction" xmlns:bn="http://barnes..">
  <bn:book>
    <title>The Naked Sun</title>
    <author>Isaac Asimov</author>
    <isbn>0553293397</isbn>
    <price>30</price> <!-- add by hand to online demo -->
  </bn:book>
</booklist>

Grab all books:

So what if my web service uses namespaces?

Step 1: Understand general idea of namespaces
(see previous slides)

Step 2: See online examples (for basic XML and XSLT)
(XML, Xpath, and XSLT examples – on Calendar)