Interactive Applications Using Ajax

- Ajax is a set of technologies that collectively enable interactive applications
  - Hallmark: Part of a page changing asynchronously
- Ajax = Asynchronous JavaScript and XML
- Extensive use in Gmail, Yahoo! Finance, etc.

Only one really new primitive:
- JavaScript uses a (client side) XMLHttpRequest to asynchronously communicate with the server
- http://www.w3.org/TR/XMLHttpRequest/

So Far in Class...

Client Communicates Synchronously with Server

Even if the new page is almost identical to starting web page, the server engages in total re-computation, creation and transmission of new response

POOR INTERACTION
Ajax

Client Communicates **Asynchronously** with Server

In response to some event, a JavaScript function creates XMLHttpRequest object and makes request to server. Browser does not redraw or reload.

The request has also defined which JavaScript function will be activated upon receiving the response. This function typically re-computes part of the page.

New JavaScript Material

- You can assign functions to variables and object properties
  - We will assign the function that handles the HTTP response to a property of XMLHttpRequest object

Ajax Example 1

```html
<body>
  Write your story here:
  <form action="nowhere" method="GET">
    <p/>
    <textarea rows="10" cols="80" name="story"
              onkeyup="lastTimeFunction();"/>
    </textarea>
  </p>
  <span id="lastTime">
    You have not typed anything in the above box yet
  </span>
  </form>
</body>
```
date.jsp used (that's all)

<%= new java.util.Date() %>

Ajax Example 1 (cont’d)

function lastTimeFunction() {
    var xmlHttp = new XMLHttpRequest();
    var responseHandler = function() {
        if (xmlHttp.readyState == 4)
            document.getElementById("lastTime").innerHTML = "You last typed on " + xmlHttp.responseText;
    }
    xmlHttp.onreadystatechange = responseHandler;
    xmlHttp.open("GET","date.jsp",true);
    xmlHttp.send(null);
}

The value of the variable is a function and NOT the result of a function call (think of C++ function pointers)

Call the function stored in this property whenever the server produces the HTTP response

3rd argument is asynchronous communication flag (versions with user & password also avail)

Initiates request. If it was POST, argument would be body

XMLHttpRequest

The `readyState` Property of XMLHttpRequest

- 0 : request not initialized yet
- 1 : request is set up
- 2 : request has been sent
- 3 : request is in process
- 4 : request is complete
XMLHttpRequest (cont'd)

// request
• open(DOMString method, DOMString url, boolean async,
  DOMString? user, DOMString? password);
• setRequestHeader(DOMString header, DOMString value);

// response
• unsigned short status; // holds the HTTP status code
• DOMString statusText; // holds the HTTP status text
• DOMString responseText; // holds the HTTP status text
• Document responseXML; // Document

Browser Compatibility

var xmlHttp;
try {
  xmlHttp = new XMLHttpRequest(); // Firefox, Opera, Safari
} catch (e) {
  // Internet Explorer
  try {
    xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
  } catch (e) {
    try {
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (e) {
      alert("Your browser does not support Ajax!");
      return false;
    }
  }
}

How to transfer complex data from the server to the client? (Strings not enough) The ‘x’ in Ajax : XML

<html>
<head><script src="selectCustomerXML.js"></script></head>
<body>
<form action="">
  Select a Customer:
  <select name="custs" onchange="showCust(this.value)">
    <option value="AV">Art Vandelay</option>
    <option value="JP">Jim Progress</option>
    <option value="ND">Nick Dummy</option>
  </select>
  <b><span id="company"></span></b><br/>
  <span id="contact"></span><br/>
  <span id="address"></span><br/>
  <span id="city"></span><br/>
  <span id="country"></span>
</form>
</body>
</html>
function showCust(str) {
    var xmlHttp = new XMLHttpRequest();
    var url="getCustomerXML.jsp";
    url = url + "?q=" + str;
    xmlHttp.onreadystatechange = stateChanged;
    xmlHttp.open("GET", url, true);
    xmlHttp.send(null);
}

An example XML response by the server to the client

<?xml version='1.0' encoding='ISO-8859-1'?><company><comp>Vandelay Industries</comp><cont>Inc.</cont><addr>9500 Gilman Drive</addr><city>La Jolla</city><cntr>USA</cntr></company>

A dummy jsp producing XML

<% response.setContentType("text/xml"); %>
<% String person=request.getParameter("person"); %>
<% if (person!=null && person.equals("AV")) { %>
<company><compname>Vandelay Industries</compname><contname>Inc.</contname><address>9500 Gilman Drive</address><city>La Jolla</city><country>USA</country></company>
<% } else if (person!=null && person.equals("JP")) { %>
<company><compname>Acme Industries</compname></company>
<% } %>
<% } %>
How to access XML:
It is not just text; it is a DOM object

An example including attributes

```xml
<?xml version='1.0' encoding='ISO-8859-1'?>
<company id='1'>
  <comp>Vandelay Industries</comp>
  <cont>Inc.</cont>
  <addr>9500 Gilman Drive</addr>
  <city>La Jolla</city>
  <cntr>USA</cntr>
</company>
```

How to access XML:
It is not just text; it is a DOM object
The response handler code navigates in the XML DOM object to pick data

```javascript
function stateChanged() {
    if (xmlHttp.readyState==4) {
        var xmlDoc = xmlHttp.responseText.XML.documentElement;
        document.getElementById("company").innerHTML =
            xmlDoc.getElementsByTagName("comp")[0].childNodes[0].nodeValue;
        document.getElementById("contact").innerHTML =
            xmlDoc.getElementsByTagName("cont")[0].childNodes[0].nodeValue;
        document.getElementById("address").innerHTML =
            xmlDoc.getElementsByTagName("addr")[0].childNodes[0].nodeValue;
        document.getElementById("city").innerHTML =
            xmlDoc.getElementsByTagName("city")[0].childNodes[0].nodeValue;
        document.getElementById("country").innerHTML =
            xmlDoc.getElementsByTagName("cntr")[0].childNodes[0].nodeValue;
    }
}
```

The Ajax page incremental update pattern

**Browser**

- **HTML DOM**
- **JavaScript Components**

**Event triggers JavaScript function**
- Receive string or XML or JSON describing diff
- Change HTML DOM or JavaScript components to reflect diff

**Application Server**

- JSP j that prepares the diff
- Compute the diff (i.e., data that describe how the page changes).
- Format as string, XML, JSON

**Students example revisited: XHTML compliance & why it matters**

Our Step 1 code was not XHTML compliant (form element directly within tr). Most browsers get confused by non-XHTML compliant incremental updates.

```html
<table id="studentsTbl" border="1">
    <thead id="students">
        <tr>
            <th>ID</th>
            <th>PID</th>
            <th>First Name</th>
            <th>Middle Name</th>
            <th>Last Name</th>
            <th><input onClick="studentAction(null, 'insert');" type="button" value="Insert"></th>
        </tr>
    </thead>
</table>
```
Parameter collection: id's for the insert part and calls to "f" studentAction

```html
<tr>
  <td><input id="pid" value="" name="id" size="10" /></td>
  <td><input id="first" value="" name="first" size="15" /></td>
  <td><input id="middle" value="" name="middle" size="15" /></td>
  <td><input id="last" value="" name="last" size="15" /></td>
  <td><input onClick="studentAction(null,'insert');" type="button" value="Insert" /></td>
</tr>
```

Parameter collection: Synthesize id's that will make it easy to spot updated values during parameter collection

```html
<tr id="<%=((Student)pair.getValue())().getPID()%>">
  <td> <%= id %> </td>
  <td> <input id="pid_<%= id %>
```
var response = eval('(' + responseDoc + ')');
switch (actionType) {
  case 'insert':
    if (response.success) {
      var table = document.getElementById("studentsTbl");
      var row = table.insertRow(table.rows.length);
      row.id = response.pid;
      var html = '<td>' + response.id + '</td><td><input id="pid_' + response.id + '" value="' + response.pid + '" name="row.innerHTML = html;
      document.getElementById('response').innerHTML = "Insert Complete";
    }

---

**JavaScript Object Notation (JSON)**

- Self-describing notation (as XML), tuned for Javascript
- Offered by Groupon, Twitter etc

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Array value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;movieslist&quot;</td>
<td>['Friday the 13th', 'Friday the 13th Part 2', 'Friday the 13th Part III', 'Friday the 13th: The Final Chapter', 'Friday the 13th: A New Beginning']</td>
</tr>
</tbody>
</table>

- Object start
- Object end

<table>
<thead>
<tr>
<th>Text value</th>
<th>Number value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;reviewer&quot;: &quot;Pam&quot;, &quot;stars&quot;: 3, &quot;text&quot;: &quot;Pretty good, but could have used more Jason&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;title&quot;: &quot;Friday the 13th&quot;, &quot;year&quot;: 1980, &quot;reviews&quot;: [{&quot;reviewer&quot;: &quot;Pam&quot;, &quot;stars&quot;: 3, &quot;text&quot;: &quot;Pretty good, but could have used more Jason&quot;}, &quot;reviewer&quot;: &quot;Alice&quot;, &quot;stars&quot;: 4, &quot;text&quot;: &quot;The end was good, but a little unsettling&quot;}</td>
</tr>
</tbody>
</table>

---

**Receiving a JSON response and converting it into a JavaScript object**

```javascript
// assume server has sent a JSON-encoded text
var responseDoc = xmlHttp.responseText;
var response = eval('(' + responseDoc + ')');
```
Creating a JSON response in Java

```java
JSONObject result = new JSONObject();
result.put("success",true);
result.put("id",pair.getKey());
result.put("pid",pid);
result.put("first",student.getFirstName());
result.put("middle",student.getMiddleName());
result.put("last",student.getLastName());
out.print(result);
```

Common Use Cases of Ajax

- Today's busy pages have multiple almost independent sections
- Reduce load by updating only the relevant section
- Quick response to inputs
  - "Illusion" that the page is faster even when it is not, simply because there is always something on screen

Common Ajax Downsides

- The "revisions" do not automatically register with browser's history
  - Back button behaves weirdly
- GET is good for bookmarking; Ajax is bad
  - Mitigation: fragment identifier
- Non-crawlable web
- New opportunities for malicious hackers
- Complicates structure
- As of now, very poor interaction with server-side frameworks (Struts, Spring) and their custom tags
  - Ruby's partial templates probably the best extension of an MVC compliant framework for Ajax
- As of now, poor interaction with non-HTML-standard features supported by browsers
Why is Ajax so hard?

• Ajax introduces difficulties of distributed computation
  – part of computation in Javascript, part in Java
  – Need to make requests from Javascript to server, vice versa “package” results of small server-side functions into http response and XML
• Ajax computations are often not fully compatible across browsers
  – Eg, “hacks” that operate on load, do not operate after an Ajax update of the page
• Half-baked, pre-paradigmatic technology

The difficulties of (unaided) Ajax

• The students example revisited to include Ajax
  – Fasten your seatbelts, it will be a bumpy ride
  – Browser oddities revealed, low level coding
• Project advice
• Ajax frameworks
• Make a contribution to the web & db lab’s FORWARD framework for Ajax applications
  – Natl Science Foundation REU support possible