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)**

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

**Ajax Example 1 (cont'd)**

```javascript
function lastTimeFunction() {
    var xmlDoc = new XMLHttpRequest(); // var xmlDoc = new XMLHttpRequest();
    xmlDoc.onreadystatechange = function() { // Call the function stored in this property whenever the server produces the HTTP response
        if (xmlDoc.readyState == 4) // The value of the variable is a function and NOT the result of a function call (think of C++ function pointers)
            document.getElementById("lastTime").innerHTML = "You last typed on " + xmlDoc.responseText;
    }
    xmlDoc.open("GET", "/date.jsp", true); // 3rd argument is asynchronous communication flag (versions with user & password also avail)
    xmlDoc.send(null); // 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; // DOMString
• 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)"
          value="AV">
    <option value="AV">Art Vandelay</option>
    <option value="JP">Jim Progress</option>
    <option value="ND">Nick Dummy</option>
  </select>
</form>
<br />
<span id="company"></span><br/>
<span id="contact"></span><br/>
<span id="address"></span><br/>
<span id="city"></span><br/>
<span id="country"></span>
</body>
</html>
function showCust(str) {
    var xmlhttp = new XMLHttpRequest();
    var url = 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 &amp; person.equals( "AV" )){
    <company><compname>Vandelay Industries</compname><contname>Inc.</contname><address>9500 Gilman Drive</address><city>La Jolla</city><cntrname>USA</cntrname></company>
} else if (person!=null &amp; person.equals( "JP" )){
    <company><compname>Acme Industries</compname></company>
} else {
    <company><compname>Vandelay Industries</compname><contname>Inc.</contname><address>9500 Gilman Drive</address><city>La Jolla</city><cntrname>USA</cntrname></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.responseXML.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 h
Collect arguments from page
Optionally make some page changes
Make XHR object
Define response handler h in XHR object
XHR request to Jsp j prepping 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

Book-keeping, change session/db state

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">
  <tbody 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>
    <tr>
      <td>&nbsp;</td>
      <td><input id="pid" value="" name="pid" 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>
  </tbody>
</table>
```
Parameter collection: id's for the insert part and calls to "f" studentAction

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

```html
<input onClick="studentAction(<%= id %>,"update");" type="button" value="Update"></td>
```

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.id"'+ response.id + '" value="' + response.pid + '" name="row.id"></td>
        row.innerHTML = html;
        document.getElementById('response').innerHTML = "Insert Complete";
    }
}
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", "G");
result.put("last", "Papakonstantinou");
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