Ajax: A new programming model for the Web

Juan M. Gimeno, Josep M. Ribó

May, 2009

Contents

Two Splitted Worlds

Rich Desktop Applications

- Difficult to distribute and deploy
- Rich user experience
- Rapid response to the user
- Off-line

Classical Web Applications

- Easy to distribute and deploy
- Poor user experience
- Delayed responses
- On-line

Can we get the best of these two worlds?

Rich Internet Applications

- Rich user experience
- Easy to distribute and deploy
- Easy to actualize
- Rich user experience
- Rapid response to the user
- Can work off-line
- Multiplatform

RIA Platforms

Adobe Flash

- Was the creator of the RIA concept
- More oriented to graphical designers than to developers

Adobe Flex

- Based on flash but oriented to developers
- XML (MXML) + Action Script 3.0

Open Laszlo

- XML (LZX) + ECMAScript
- Compiles into Flash or AJAX

Microsoft Silverlight

- XML (XAML) + .NET (web version of WPF: Windows Presentation Foundation)

Java Web Start

- Autodeployable and updatable applications

JavaFX ????

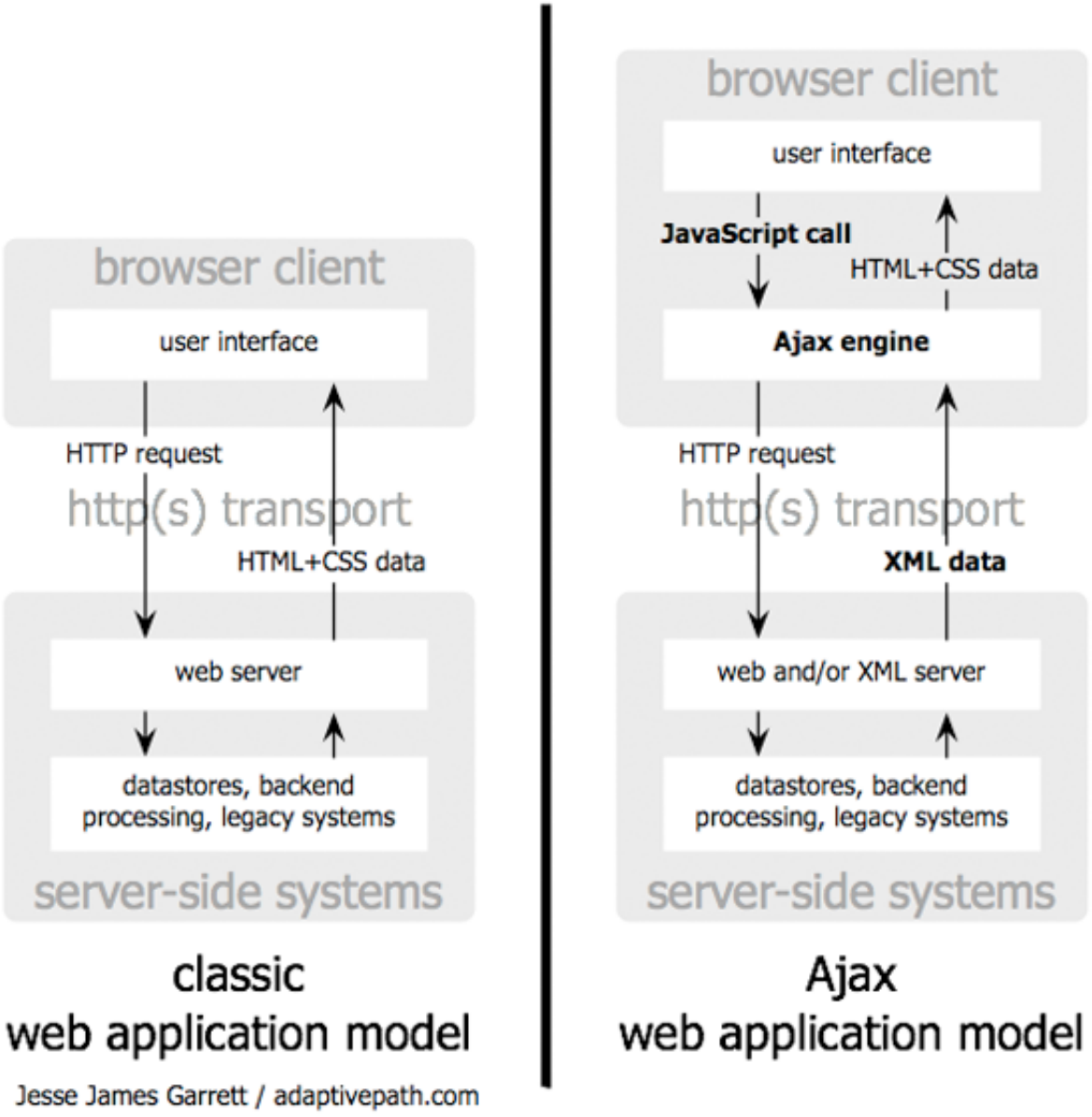
- JavaFX Script
- JavaFX Mobile

AJAX

AJAX

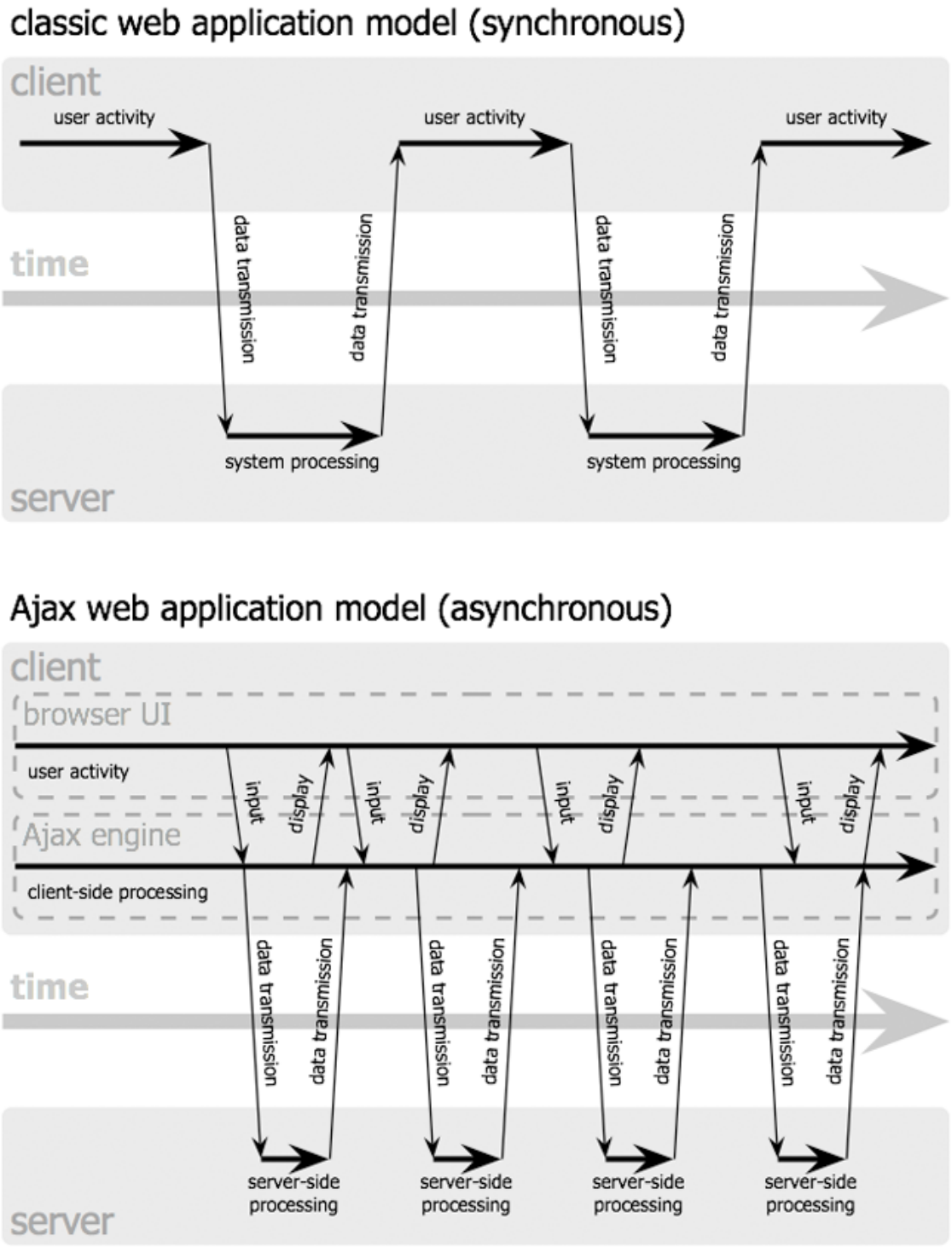
- Based on standards and not owned by any company
- AJAX stands for Asynchronous JavaScript And XML (term coined by Jesse James Garret in a famous article)
- AJAX is not new but it was not popular until Google used it into its products
- Based on the `XMLHTTP` object introduced in IE5.0 (1999) to allow remote scripting of Outlook Web Access
- All browsers have included their own version of the `XMLHTTP` object
- Technologies in AJAX:
  - (X)HTML and CSS for structure and design
  - JavaScript as a programming language
  - DOM (Document Object Model) to dynamically modify the structure
  - XML as a data transport language between client and server (but it can use others such as JSON)

Classic and AJAX Application Model



(from http://www.adaptivepath.com/ideas/essays/archives/000385.php)

Classic and AJAX Application Flow



(from http://www.adaptivepath.com/ideas/essays/archives/000385.php)

XMLHTTP Object

- History:
  - 1998, Microsoft presents Remote Scripting to allow, by means of an applet, make requests from JS without the user noticing it
  - 1999, IE5.0 incorporates, under ActiveX, the XMLHTTP object that replaces the applet
  - 5 years later, and driven by Google, used to create rich webweb applications

- Compatibility:
  - As an ActiveX XMLHTTP object in IE from 5 up to 7
  - As a native JS XMLHttpRequest object in Firefox (≥ 1.0), Netscape (≥ 7.1), Opera (≥ 8.0), Safari (≥ 1.2), IE 7, Konqueror, Opera Mobile, Nokia Web Browser, ...

- It is the heart of an AJAX application because it allows to asynchronically make requests to the server without changing the page

Using the XMLHTTP Object

To make a request to the server from JavaScript using the XMLHTTP Object, one must follow these steps:

1. Obtain the XMLHTTP object

2. Configure and open the connection with the server

3. Define a JavaScript callback function to administer the evolution of the request (because it is asynchronous)

4. Send the request and data to the server

5. The before mentioned function must:

   - Administer the state of the request
   - Get the result from the server
   - Process the result (i.e. updating the page)

Obtaining the XMLHTTP object

```
1  function getXHR() {
2    req = false;
3    if (window.XMLHttpRequest) {
4       req = new XMLHttpRequest();
5    } else {
6       if (ActiveXObject) {
7         msVersions = [ "MSXML2.XMLHttp5.0", "MSXML2.XMLHttp4.0",
8                        "MSXML2.XMLHttp3.0", "MSXML2.XMLHttp",
9                        "Microsoft.XMLHttp" ];
10        for (var i=0; i<msVersions.length; i++) {
11          try {
12            req = new ActiveXObject(msVersions[i]);
13            return req;
14          } catch (e) {}
15        }
16      }
17    }
18    return req;
19 }
```

XMLHTTP Object Methods

The methods of the `XMLHTTP` object are:

open It does not ope the connection to the server but configures the request and leaves it prepared to be sent. Its parameters are:

- **method** string with the method used to make the request ("GET"or "POST")
- **url** string with the url of the resource. It can be relative to the current document or absolute. If the method is "GETit must include the parameters.
- **asynchronous?** boolean that indicates whether the request will be processed asynchronously (true) or not (false).
- **user** optional string to authenticate with the server.
- **password** needed if user is defined.

send This is the method that actually connects to the server making the request.
It has a parameter which is used to pass parameters if the request method is "POST".
If no parameters are sent or method is "GET"one can use the special value `null`.

abort Seldom used, aborts the processing of an already sent request.

setRequestHeader defines a request header. It has two string parameters:

key name of the header
value its value

getResponseHeader gets the value of a response header as a string.

getAllResponseHeaders all the headers in a multiline string. It can be splitted using `.split("\r?\n")` and then splitted again with `.split(":")`

XMLHTTP Object Properties

Some properties of the XMLHTTP Object are used to administer the request:

readyState Read-only property that represents the state of the request:

0	Not initialized (XMLHTTP created but not opened)
1	Configured (opened but not sent)
2	Sending (sent or being sent but no response has yet been received)
3	Interactive (we have got the header response but the content is being received)
4	Completed (we have access to the full response)

status HTTP response code (available when readyState is 4)

200	The request has been processed without problems
400	Invalid request sent (e.g. bad request headers or POST data)
403	Access to the resource is not permitted
404	URL not found on the server
405	Method not accepted (some problem processing GET or POST)
414	URL too long
500	Server internal error
503	Server temporally unavailable

statusText read-only string that describes the `status`

responseText read-only string with the contents of the response (its value only has sense when readyState is 4 and status is 200)

responseXML if the response is valid XML, this property is the XML native JavaScript object that can be processes with the DOM API.

onreadystatechange this property allows configuring the callback function that will be called each time the `readyState` property changes its value.
This property can be defined before or after invoking `open` but before invoking `send`.

```
// Option 1
function processRequest() {
  ...
};
xhr.onreadystatechange = processRequest;

// Option 2
xhr.onreadystatechange = function () {
  .....
};
```

Example 1: AJAX Flow

- This example shows the activation of the `onreadystatechange` function
- Extracted from the book *Pragmatic Ajax* (netbeaned in Ajax Netbeans repository)
- **Question: Why does the alert with the message executes twice?**

```
 1 <html>
 2   <head>
 3       <script type="text/javascript">
 4 var xhr;
 5
 6 function modifyPage() { ... }
 7       </script>
 8   </head>
 9   <body>
10       <button onclick="modifyPage()">Click Me</button>
11   </body>
12 </html>
```

Example1: JavaScript Code

```
1  var xhr;
2  function modifyPage() {
3      try {
4          xhr = new ActiveXObject("Msxml2.XMLHTTP");
5          alert("Msxml2.XMLHTTP ActiveXObject created");
6      } catch (e) {
7          try {
8              xhr = new ActiveXObject("Microsoft.XMLHTTP");
9              alert("Microsoft.XMLHTTP ActiveXObject created");
10         } catch (E) {
11         xhr = false;
12         }
13     }
14     if (!xhr && typeof XMLHttpRequest != 'undefined') {
15         xhr = new XMLHttpRequest();
16         alert("XMLHttpRequest Object created");
17     }
18     xhr.open("GET", "./getMessage", "true");
19     alert("Open method called");
20     xhr.setRequestHeader("User-Agent", "my browser");
21     alert("Request header setted");
22     xhr.onreadystatechange = function() {
23         alert("State : " + xhr.readyState + "" );
24         if (xhr.readyState != 4) return;
25         if (xhr.status == 200) {
26             alert("The message is:\n" + xhr.responseText);
27         } else
28             alert("Error processing response. Status=" + xhr.status);
29     }
30     xhr.send(null);
31 }
```

Example1: Java Servlet Code

```
1  package ajaxian.book.explained.servlet;
2
3  import java.io.*;
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6
7  public class MessageServlet extends HttpServlet {
8
9      @Override
10     public  void doGet(HttpServletRequest request,
11                        HttpServletResponse  response)
12         throws IOException, ServletException {
13
14           response.setContentType("text/plain");
15           response.setHeader("Cache-Control", "no-cache");
16           response.getWriter().write("Hello from the server");
17
18     }
19 }
```

Accessing and modifying the page

- In the past example we have only shown alerts but we have not accessed the page content.

- Usually AJAX applications modify the current content of the page with the response obtained from the server.

- To do so, we need:

  1. parts of the page (tipically a `<div>`) must be uniquely identifiable (using its `id` attribute's value).
  2. to modify the content we can use the `innerHTML` property of the element (simpler than pure DOM nodes manipulation)
  3. the AJAX request must be sent when we are sure the `<div>`) is loaded.
     If one wants the request to be done when the page is being loaded, one must use the `onload` method of the body.
  4. the XMLHTTP object must be a global variable of the page, so it can be accessed from different functions.

Example 2: AJAX Modification

index.html

```
1  <html>
2      <head>
3          <script type="text/javascript" src="xhr.js"></script>
4          <script type="text/javascript">\begin{lstlisting}[language=JavaScript,
5  var xhr;
6
7  function modifyPage() {
8      xhr = getXHR();
9      xhr.open("GET", "./getMessage", "true");
10     xhr.setRequestHeader("User-Agent", "my browser");
11     xhr.onreadystatechange= function() {
12         if (xhr.readyState != 4) return;
13         if (xhr.status == 200) {
14             document.getElementById("message").innerHTML = xhr.responseText;
15         } else {
16             alert("Error processing response with status " + xhr.status);
17         }
18     }
19     xhr.send(null);
20 }
21         </script>
22     </head>
23     <body>
24         <div id="message"></div>
25         <button onclick="modifyPage()">Click Me</button>
26     </body>
27 </html>
```

DOM: Document Object Model

- DOM is a W3C standard that allows the processing of XML documents (HTML is not XML but DOM is also ported to it)
- It defines common interfaces to traverse and manipulate a hierarchical representation of the HTML/XML
- Implemented in different languages: Java, JavaScript, Python, PHP, ...
- In JavaScript we can use the DOM to:
  - Manipulate the results obtained from the server (if they are expressed in XML)
  - Manipulate the page by accessing the object `window.document` (or simply `document`)

Example 3: AJAX Modification using DOM

index.html

```
<html>
    <head>
        <script type="text/javascript" src="xhr.js"></script>
        <script type="text/javascript" src="text-utils.js"></script>
        <script type="text/javascript">
var xhr;

function modifyPage() {
    xhr = getXHR();
    xhr.open("GET", "./getMessage", "true");
    xhr.setRequestHeader("User-Agent", "my browser");
    xhr.onreadystatechange= function() {
        if (xhr.readyState != 4) return;
        if (xhr.status == 200) {
            replaceText(document.getElementById("message"), xhr.responseText);
        } else {
            alert("Error processing response with status " + xhr.status);
        }
    }
    xhr.send(null);
}
        </script>
    </head>
    <body>
        <div id="message"></div>
        <button onclick="modifyPage()">Click Me</button>
    </body>
</html>
```

Example 3: AJAX Modification using DOM

text-utils.js

```
1
2
3  function replaceText(el, text) {
4      if (el != null) {
5          clearText(el);
6          var newNode = document.createTextNode(text);
7          el.appendChild(newNode);
8      }
9  }
10
11 function clearText(el) {
12     if (el != null) {
13         if (el.childNodes) {
14             for (var i=0; el.childNodes.length; i++) {
15                 var childNode = el.childNodes[i];
16                 el.removeChild(childNode);
17             }
18         }
19     }
20 }
21
22 function getText(el) {
23     var text = "";
24     if (el != null) {
25         if (el.childNodes) {
26             for (var i = 0; el.childNodes.length; i++) {
27                 var childNode = el.childNodes[i];
28                 if (childNode.nodeValue != null) {
29                     text = text + childNode.nodeValue;
30                 }
31             }
32         }
33     }
34 }
```

Example: Autocomplete

- An example based (minor modifications) on the **J2EE Blueprints Samples** from Sun.
- When filling a text entry, generates a pop-up with possible completions for the already entered characters
- Hands-on-project: modify the example to allow the selection of a name by keyboard and not by mouse.

Example: Autocomplete

index.jsp

```
<html>
    <head>
        <style type="text/css">

            .selected {
                background: #7A8AFF;
                color: #FFFAFA;
            }

            .unselected {
                background: #FFFAFA;
                color: #000000;
            }
        </style>
        <script type="text/javascript">
var isIE;
var req;
var names;
var target;

function getElementX(element){
    var targetLeft = 0;
    if (element.offsetParent) {
```

```
24          while (element.offsetParent) {
25              targetLeft += element.offsetLeft;
26              element = element.offsetParent;
27          }
28      } else if (element.x) {
29          targetLeft += element.x;
30      }
31      return targetLeft;
32  }
33
34  function getElementY(element){
35      var targetTop = 0;
36      if (element.offsetParent) {
37          while (element.offsetParent) {
38              targetTop += element.offsetTop;
39              element = element.offsetParent;
40          }
41      } else if (element.y) {
42          targetTop += element.y;
43      }
44      return targetTop;
45  }
46
47  function init() {
48      target = document.getElementById("complete-field");
49      var popup = document.getElementById("menu-popup");
50      popup.style.top = (getElementY(target) + target.offsetHeight) + "px";
51      popup.style.left = getElementX(target) + "px";
52  }
53
```

```
54 function initRequest(url) {
55     if (window.XMLHttpRequest) {
56         req = new XMLHttpRequest();
57     } else if (window.ActiveXObject) {
58         isIE = true;
59         req = new ActiveXObject("Microsoft.XMLHTTP");
60     }
61 }
62
63 function doCompletion() {
64     var url = "autocomplete?action=complete&id=" + escape(target.value);
65     initRequest(url);
66     req.onreadystatechange = processRequest;
67     req.open("GET", url, true);
68     req.send(null);
69 }
70
71 function processRequest() {
72     if (req.readyState == 4) {
73         if (req.status == 200) {
74             parseMessages();
75         } else if (req.status == 204){
76             clearTable();
77         }
78     }
79 }
80
81 function parseMessages() {
82     if (!names) {
83         names = document.getElementById("names");
```

```
84      }
85      clearTable();
86      var employees = req.responseXML.getElementsByTagName("employees")[0];
87      for (loop = 0; loop < employees.childNodes.length; loop++) {
88          var employee = employees.childNodes[loop];
89          var firstName = employee.getElementsByTagName("firstName")[0];
90          var lastName = employee.getElementsByTagName("lastName")[0];
91          var employeeId = employee.getElementsByTagName("id")[0];
92          appendEmployee(firstName.childNodes[0].nodeValue,
93                         lastName.childNodes[0].nodeValue,
94                         employeeId.childNodes[0].nodeValue);
95      }
96  }
97
98  function clearTable() {
99      if (names) {
100         for (loop = names.childNodes.length -1; loop >= 0 ; loop--) {
101             names.removeChild(names.childNodes[loop]);
102         }
103     }
104 }
105
106 function appendEmployee(firstName,lastName,employeeId) {
107     var firstNameCell;
108     var lastNameCell;
109     if (isIE) {
110         row = names.insertRow(names.rows.length);
111         nameCell = row.insertCell(0);
112     } else {
113         row = document.createElement("tr");
```

```
114             nameCell = document.createElement("td");
115             row.appendChild(nameCell);
116             names.appendChild(row);
117         }
118         row.setAttribute("border", "0");
119         nameCell.setAttribute("onmouseout", "this.className='unselected';");
120         nameCell.setAttribute("onmouseover", "this.className='selected';");
121         nameCell.setAttribute("bgcolor", "#FFFAFA");
122         nameCell.setAttribute("border", "0");
123         var linkElement = document.createElement("a");
124         linkElement.setAttribute("style", "text-decoration: none");
125         linkElement.setAttribute("href", "autocomplete?action=lookup&id=" + employeeId);
126         var nameFontElement = document.createElement("font");
127         nameFontElement.setAttribute("size", "+1");
128         nameFontElement.setAttribute("color", "black");
129         nameFontElement.appendChild(document.createTextNode(firstName + " " + lastName));
130         linkElement.appendChild(nameFontElement);
131         nameCell.appendChild(linkElement);
132     }
133             </script>
134             <title>Auto-Completion using Asynchronous JavaScript and XML (AJAX)</title>
135         </head>
136         <body onload="init()">
137
138             <h1>Auto-Completion using Asynchronous JavaScript and XML (AJAX)</h1>
139             <hr/>
140             <p>
141                 This example shows how you can do real time auto-completion using
142                 AJAX interactions.
143             </p>
```

```
144         <p>
145             In the form below enter a name. Possible names that will be completed
146             are displayed beneath the form.
147             Click on one of the selections to see the employee details.
148             Try typing &amp;quot;Greg&amp;quot;, &amp;quot;Murray&amp;quot;, &amp;quot;Jones&amp;quot;, or
149             &amp;quot;Cindy&amp;quot;.
150         </p>
151
152         <form name="autofillform" action="autocomplete" method="get">
153             <input type="hidden" name="action" value="lookupbyname" />
154             <table border="0" cellpadding="5" cellspacing="0">
155                 <tr>
156                     <td><b>Employee Name:</b></td>
157                     <td>
158                         <input     type="text"
159                                    size="20"
160                                    id="complete-field"
161                                    name="id"
162                                    onkeyup="doCompletion();">
163                     </td>
164                     <td align="left">
165                         <input id="submit_btn" type="Submit" value="Lookup Employee">
166                     </td>
167                 </tr>
168             </table>
169         </form>
170         <div style="position: absolute; top:170px; left:140px" id="menu-popup">
171             <table id="names"
172                    bgcolor="#FFFAFA"
173                    border="1"
```

```
174                         bordercolor="black"
175                         cellspacing="0"
176                         cellpadding="0" />
177             </div>
178         </body>
179 </html>
```

Example: Autocomplete

AutocompleteServlet.jsp

```
1  package com.sun.j2ee.blueprints.bpcatalog.ajax;
2
3  import java.io.*;
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6  import java.util.*;
7
8  public class AutocompleteServlet extends HttpServlet {
9
10     private ServletContext context;
11     private HashMap employees = new HashMap();
12
13     public void init(ServletConfig config) throws ServletException {
14         this.context = config.getServletContext();
15         employees.put("1", new EmployeeBean("1", "Greg", "Murray"));
16         employees.put("2", new EmployeeBean("2", "Greg", "Murphy"));
17         employees.put("3", new EmployeeBean("3", "George", "Murphy"));
18         employees.put("4", new EmployeeBean("4", "George", "Murray"));
19         employees.put("5", new EmployeeBean("5", "Peter", "Jones"));
20         employees.put("6", new EmployeeBean("6", "Amber", "Jones"));
21         employees.put("7", new EmployeeBean("7", "Amy", "Jones"));
22         employees.put("8", new EmployeeBean("8", "Bee", "Jones"));
23         employees.put("9", new EmployeeBean("9", "Beth", "Johnson"));
```

```
24          employees.put("10", new EmployeeBean("10", "Cindy", "Johnson"));
25          employees.put("11", new EmployeeBean("11", "Cindy", "Murphy"));
26          employees.put("12", new EmployeeBean("12", "Duke", "Hazerd"));
27      }
28
29      public void doGet(HttpServletRequest request, HttpServletResponse response)
30              throws IOException, ServletException {
31
32          String action = request.getParameter("action");
33          String targetId = request.getParameter("id");
34          StringBuffer sb = new StringBuffer();
35          if (targetId != null) {
36              targetId = targetId.trim().toLowerCase();
37          }
38          boolean namesAdded = false;
39          if ("complete".equals(action)) {
40              Iterator it = employees.keySet().iterator();
41              while (it.hasNext()) {
42                  String id = (String) it.next();
43                  EmployeeBean e = (EmployeeBean) employees.get(id);
44                  // simple matching only for start of first or last name
45                  if ((e.getFirstName().toLowerCase().startsWith(targetId) ||
46                          e.getLastName().toLowerCase().startsWith(targetId)) &&
47                          !targetId.equals("")) {
48                      sb.append("<employee>");
49                      sb.append("<id>" + e.getId() + "</id>");
50                      sb.append("<firstName>" + e.getFirstName() + "</firstName>");
51                      sb.append("<lastName>" + e.getLastName() + "</lastName>");
52                      sb.append("</employee>");
53                      namesAdded = true;
```

```
54                  }
55              }
56              if (namesAdded) {
57                  response.setContentType("text/xml");
58                  response.setHeader("Cache-Control", "no-cache");
59                  response.getWriter().write("<employees>" + sb.toString() + "</employees>");
60              } else {
61                  //nothing to show
62                  response.setStatus(HttpServletResponse.SC_NO_CONTENT);
63              }
64          }
65          if ("lookupbyname".equals(action)) {
66              Iterator it = employees.keySet().iterator();
67              ArrayList names = new ArrayList();
68              while (it.hasNext()) {
69                  String id = (String) it.next();
70                  EmployeeBean e = (EmployeeBean) employees.get(id);
71                  // simple matching only for start of first or last name
72                  if (e.getFirstName().toLowerCase().startsWith(targetId) ||
73                          e.getLastName().toLowerCase().startsWith(targetId)) {
74                      names.add(e);
75                  }
76              }
77              if (names.size() > 0) {
78                  request.setAttribute("employees", names);
79              }
80              context.getRequestDispatcher("/employees.jsp").forward(request, response);
81          } else if ("lookup".equals(action)) {
82              // put the target employee in the reqeust scope to display
83              if ((targetId != null) && employees.containsKey(targetId.trim())) {
```

```
84                  request.setAttribute("employee", employees.get(targetId));
85                  context.getRequestDispatcher("/employee.jsp").forward(request, response);
86              } else {
87                  context.getRequestDispatcher("/error.jsp").forward(request, response);
88              }
89          }
90      }
91  }
```

Example: Autocomplete

employees.jsp

```
1  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2
3  <html>
4      <head>
5      </head>
6      <body>
7          <h1>Search Results</h1>
8          <hr/>
9
10         <c:choose>
11             <c:when test="${requestScope.employees == null}">
12                 <p><font size="+2" color="red">Unable to locate any employees.</font></p>
13             </c:when>
14             <c:otherwise>
15                 <c:forEach var="employee" begin="0" items="${requestScope.employees}">
16                     <p><a href="autocomplete?action=lookup&id=${employee.id}">
17                         ${employee.firstName} ${employee.lastName}
18                         </a>
19                     </p>
20                 </c:forEach>
21             </c:otherwise>
22         </c:choose>
23         <br/>
```

```
24          <p>
25              <a href="index.jsp">Go back to the application home</a>.
26          </p>
27      </body>
28  </html>
```

Prototype

- Trying to program in JavaScript without any library is almost suicidal
  - There are subtle differences among the implementations of JS in the different browsers
  - Programming in JavaScript is tricky (e.g. it has objects but no classes nor inheritance)
  - (the language has some flaws but it was a *"hack that got shipped"*)

- Prototype is a fairly broad library that üpgrades"the language
  - makes common tasks easier (mainly Ajax oriented)
  - provides ways to implement Java-style inheritance
  - extends HTML DOM elements with new properties and methods
  - provides utilities for working with JSON (Java Script Object Notation)

Example: The MyTunes Application

index.html

```
1  <html>
2      <head>
3          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
4          <title>MyTunes Library</title>
5          <script type="text/javascript" src="js/prototype.js"></script>
6          <script type="text/javascript">
7              function loadTunes() {
8                  new Ajax.Updater('tunesBox', 'list.jsp', {method: 'get'});
9              }
10         </script>
11         <link rel="stylesheet" href="css/tunes.css" type="text/css" />
12     </head>
13     <body onload="loadTunes();">
14         <div id="pageTitle">MyTunes</div>
15         <div id="tunesBox">
16             <img id="spinner" src="images/wait_spinner.gif" height="33" width="33" />
17         </div>
18     </body>
19 </html>
```

Example: The MyTunes Application

list.jsp

```
1  <%@ page contentType="text/html" pageEncoding="UTF-8" %>
2  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
3  <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
4
5  <sql:query var="songs" dataSource="jdbc/MyTunesDS">
6      select * from Songs
7  </sql:query>
8  <table border="1" width="100\%" cellpadding="8">
9      <thead>
10         <tr>
11             <td>Name</td>
12             <td>Artist</td>
13             <td>Album</td>
14             <td>Genre</td>
15             <td>Year</td>
16         </tr>
17     </thead>
18     <tbody>
19         <c:forEach var="song" items="${songs.rows}">
20             <tr>
21                 <td><a href="edit.jsp?id=${song.id}">${song.title}</a></td>
22                 <td>${song.artist}</td>
23                 <td>${song.album}</td>
24                 <td>${song.genre}</td>
25                 <td>${song.year_}</td>
26             </tr>
27         </c:forEach>
28     </tbody>
29 </table>
```

Example: The MyTunes Application

edit.jsp

```
1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2
3  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4  "http://www.w3.org/TR/html4/loose.dtd">
5
6  <html>
7      <head>
8          <title>Edit a Song</title>
9          <script type="text/javascript" src="js/prototype.js"></script>
10         <script type="text/javascript" src="js/editor.js"></script>
11         <link rel="stylesheet" href="css/tunes.css" />
12     </head>
13     <body onload="loadSong();">
14         <div id="pageTitle">Edit Song</div>
15         <div id="tunesBox">
16             <span id="spinner">
17                 <img src="images/wait_spinner.gif" height="33" width="33" />
18             </span>
19             <form id="songForm" onsubmit="catchSubmit();">
20                 <input type="hidden" name="id" id="id" />
21                 <div id="name">
22                     <span id="nameLbl">Name:</span>
23                     <span id="title" onclick="edit(this)">?</span>
24                 </div>
25                 <div id="artistDiv">
26                     <span id="artistLbl">Artist:</span>
27                     <span id="artist" onclick="edit(this)">?</span>
28                 </div>
29                 <div id="albumDiv">
30                     <span id="albumLbl">Album:</span>
31                     <span id="album" onclick="edit(this)">?</span>
32                 </div>
33                 <div id="genreDiv">
34                     <span id="genreLbl">Genre:</span>
35                     <span id="genre" onclick="edit(this)">?</span>
36                 </div>
37                 <div id="yearDiv">
```

```
38                      <span id="yearLbl">Year:</span>
39                      <span id="year_" onclick="edit(this)">?</span>
40                  </div>
41              </form>
42          </div>
43          <div class="backLink">
44              <a href="index.html">Back to MyTunes Library</a>
45          </div>
46      </body>
47  </html>
```

Example: The MyTunes Application

editor.js (lines 1-39)

```
1  // global used to keep track of what's currently being edited
2  var currentElement = null;
3
4  function loadSong(){
5          params = window.location.search.parseQuery();
6          var id = params["id"];
7
8          // create handler that will be invoked
9          // when response is received from server
10         var handler = function(xhr){
11                 // use responseJSON property added by Prototype
12                 var json = xhr.responseJSON;
13                 // check for error
14                 if (json.error){
15                         // display the error
16                 }
17                 var song = json.song;
18                 // clear the spinner
19                 // use Prototype's $() shortcut notation
20                 $("spinner").innerHTML = "";
21                 // set the display data
22                 $("id").value = song.id;
23                 $("title").innerHTML = song.title;
24                 $("artist").innerHTML = song.artist;
25                 $("album").innerHTML = song.album;
26                 $("genre").innerHTML = song.genre;
27                 $("year_").innerHTML = song.year_;
28         };
29
30         // create options for
31         var options = {
32                 method : "get",
33                 onSuccess : handler,
34                 parameters : { "id" : id }
35         };
36
37         // send the request
38         new Ajax.Request("SongServlet", options);
39 }
```

Example: The MyTunes Application

SongServlet.java

```
1  public class SongServlet extends HttpServlet {
2
3      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
4      throws ServletException, IOException {
5          response.setContentType("application/json");
6          PrintWriter out = response.getWriter();
7          try {
8              String message = "";
9              JSONObject resp = new JSONObject();
10             try {
11                 Context initContext = new InitialContext();
12                 Context envContext = (Context) initContext.lookup("java:/comp/env");
13                 DataSource ds = (DataSource) envContext.lookup("jdbc/MyTunesDS");
14                 Connection conn = ds.getConnection();
15                 Statement stmt = conn.createStatement();
16                 ResultSet rs = stmt.executeQuery("SELECT * FROM Songs WHERE id=" +
17                                                  request.getParameter("id"));
18                 if (rs.next()) {
19                     JSONObject song = new JSONObject();
20                     String[] attrs = {"id", "title", "artist", "album", "genre", "year"};
21                     for (String attr : attrs) {
22                         song.put(attr, rs.getString(attr));
23                     }
```

```
24                      resp.put("song", song);
25                  }
26              } catch (JSONException ex) {
27                  Logger.getLogger(SongServlet.class.getName()).log(Level.SEVERE, null, ex);
28                  message= "Problema amb el JSON";
29              } catch (SQLException ex) {
30                  Logger.getLogger(SongServlet.class.getName()).log(Level.SEVERE, null, ex);
31                  message = "Problema amb la BD";
32              } catch (NamingException ex) {
33                  Logger.getLogger(SongServlet.class.getName()).log(Level.SEVERE, null, ex);
34                  message = "Problema amb JNDI";
35              }
36              resp.put("error", message);
37              out.write(resp.toString());
38          } catch (JSONException ex) {
39              Logger.getLogger(SongServlet.class.getName()).log(Level.SEVERE, null, ex);
40              // Desesperated with cached exceptions ;-)
41          } finally {
42              out.close();
43          }
44      }
45      // ....
46  };
```

Example: The MyTunes Application

editor.js (lines 40-103)

```
40 function edit(elem){
41         var id = elem.id;
42     // Disable onclick when editing
43     elem.onclick = "";
44         var str = createEditor(id, elem.innerHTML);
45     // Re-enabling it
46     elem.onclick = "edit(this)";
47         elem.innerHTML = str;
48         currentElement = $("song_"+id);
49         $("song_"+id).focus();
50 }
51
52 function createEditor(name, value){
53         var str = "<input onkeypress=\"catchSubmit(event)\"
54                         onBlur=\"makeText(this)\"
55                         type=\"text\"
56                         name=\"";
57         str += name;
58         str += "\" value=\"";
59         str += value;
60         str += "\" id=\"song_";
61         str += name;
62         str += "\"/>";
63         return str;
64 }
65
66 function createSpan(name, value){
67         var str = "<span onClick=\"edit(this)\" id=\"";
68         str += name;
69         str += "\">";
70         str += value;
71         str += "</span>";
72         return str;
73 }
74
```

```
75 function makeText(input){
76         // save record
77         var formData = $("songForm").serialize(true);
78         saveData(formData);
79         // go back to display
80         input.parentNode.innerHTML = input.value;
81 }
82
83 function saveData(song){
84         var handler = function(xhr){
85                 var json = xhr.responseJSON;
86                 if (json.error){
87                         // display the error
88                 }
89         };
90         var options = {
91                 method : "post",
92                 onSuccess : handler,
93                 parameters : song
94         };
95         new Ajax.Request("UpdateServlet", options);
96 }
97
98 function catchSubmit(event){
99         // KEY_RETURN is a field added to the Event class by Prototype
100        if (event.keyCode == Event.KEY_RETURN){
101                makeText(currentElement);
102        }
103 }
```

Example: The MyTunes Application

SongServlet.java

```
1  public class UpdateServlet extends HttpServlet {
2
3      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
4              throws ServletException, IOException {
5          response.setContentType("application/json");
6          PrintWriter out = response.getWriter();
7          String clause = "";
8          String message = "";
9          try {
10             Context initContext = new InitialContext();
11             Context envContext = (Context) initContext.lookup("java:/comp/env");
12             DataSource ds = (DataSource) envContext.lookup("jdbc/MyTunesDS");
13             Connection conn = ds.getConnection();
14             Statement stmt = conn.createStatement();
15             for (Enumeration keys = request.getParameterNames(); keys.hasMoreElements();) {
16                 String key = (String) keys.nextElement();
17                 if (!key.equals("id")) {
18                     String value = request.getParameter(key);
19                     clause = clause + key + "='" + value + "' ";
20                 }
21             }
22             stmt.executeUpdate("UPDATE Songs SET " + clause + "WHERE id=" + request.getParameter("id"));
23         } catch (SQLException ex) {
```

```
24              Logger.getLogger(UpdateServlet.class.getName()).log(Level.SEVERE, null, ex);
25              message = ex.toString();
26          } catch (NamingException ex) {
27              Logger.getLogger(UpdateServlet.class.getName()).log(Level.SEVERE, null, ex);
28              message = ex.toString();
29          } finally {
30              out.write("{\"error\" : " + message + "}");
31          }
32
33      }
34      // ...
35  };
```

As to conclude

- This has been a very rushy introductuon to Ajax
- Like it or not, JavaScript is one of the languages of the future^HHHHHHpresent
- Raw JavaScript is not enought: use at least prototype
- (X)HTML + CSS + JS + ¿Prototype? is a **MUST**
- To explore: GWT (Google Web Toolkit) does Ajax from Java