Chapter 3: Web Paradigms and Interactivity

3.1 AJAX: Asynchronous Interactivity in the Web
3.2 Paradigms for Web-Based Communication
3.3 Reverse AJAX and COMET
3.4 Web Sockets and Web Messaging
3.5 Web Workers

Literature:
Asynchronous JavaScript + XML (AJAX)

  - Names an idea which was in use already at the time (e.g. Google Suggest)  
  - Basic idea: Decouple server communication and changes in presentation

- Advantages:
  - User can interact fluidly with the application
  - Information from server is fetched in background
  - Display can always stay up-to-date

- AJAX is not a technology, it is a combination of known technologies
  - XHTML, CSS, DOM, XML, XSLT, JavaScript, XMLHttpRequest

- There are AJAX-like applications which use neither JavaScript nor XML
  - E.g. using Flash and querying servers in the background
  - Many applications nowadays use a different encoding than XML (e.g. JSON)
Classical Synchronous Web Application Model

Jesse James Garrett / adaptivepath.com
Asynchronous Web Application Model
AJAX and Client-Side Scripting

- UI JavaScript
  - Access to loaded/displayed HTML via DOM
  - Flexible input elements (HTML5)
  - Graphics (HTML5 canvas)
- Engine JavaScript
  - Event handling
- jQuery is a good fit for AJAX

Written in JavaScript
**AJAX and Server-Side Scripting**

- Typical examples for asynchronous server interaction:
  - Assistance in form filling (search suggestions, post or bank code decoding)
  - Real-time data (news ticker, stock prices)
  - Event notification (incoming mail, update of presence status)
  - Live chat

Any language for server-side processing (e.g. PHP, also JavaScript)
Example 1 (Last Lecture), Using jQuery

User ➔ Browser ➔ Server ➔ DB

Open URL (php)

HTML+JS/jQuery

AJAX request (jQuery)

write

read
Example 2 (Very Simple Request), Using jQuery

User ↔ Browser
local JS

AJAX request (jQuery)

Server

file

data
Example 2 (Very Simple Request), Using jQuery

The following text is replaced with data retrieved from server (data.txt):

Text to be inserted here

```html
<p id='text'>Text to be inserted here</p>

<script type='text/javascript'>
    $(document).ready(function() {
        $.ajax({
            type: 'GET',
            url: 'http://localhost/~hussmann/data.txt',
            success: function(data, status) {
                alert("Status: "+status);
                $('#text').html(data);
            }
        });
    });
</script>
```

ajaxreq_simple_txt.html
Example 3 (Answered Request), Using jQuery

Browser

User

local JS

update display

AJAX request (jQuery)

Browser → Server

Server

query

result

dbserver.php

DB
Example 3 (Answered Request), DB Server

```php
<?php

$link = mysql_connect('localhost','root','demopw')
or die ('Could not connect: '.mysql_error());
mysql_select_db('music') or die ('Could not select db.');
$title = $_REQUEST['title'];
$query = "SELECT * FROM mysongs WHERE title='$title'";
$result = mysql_query($query) or die (...);
$row = mysql_fetch_array($result, MYSQL_ASSOC);
echo json_encode($row);

mysql_free_result($result);
mysql_close($link);fclose($file);
?>
```

dbserver.php
Example 3 (Answered Request), Request

```html
<input id='inp_title' type='text' size='20'/><br/>
<input id='btn' type='button' value='Search'/></input>
<table id='results' class='result_displ'>
  <thead>...</thead>
  <tbody></tbody>
</table>

<script type='text/javascript'> ...
  $('.btn').click( function() {
    $.ajax({
      type: 'GET',
      url: 'http://localhost/~hussmann/dbserver.php',
      data: {title: $('#inp_title').val()},
      dataType: 'json',
      success: function(data) {
        $('#results tbody').append(
          '<tr><td>'+data.code+'</td>'+ ...</tr>'
        );
      };
    });
```
# Building a List of Search Results

Search for a title name: Lady in Black

<table>
<thead>
<tr>
<th>Code</th>
<th>Title</th>
<th>Artist</th>
<th>Album</th>
<th>Runtime</th>
<th>Remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One</td>
<td>U2</td>
<td>The Complete U2</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Lady in Black</td>
<td>Uriah Heep</td>
<td>Lady in Black</td>
<td>281</td>
<td></td>
</tr>
</tbody>
</table>

- Search result for "Lady in Black".
Example 3 (Answered Request), Asynchronous!

Browser

local JS

AJAX request (jQuery)

local JS

Browser

waiting for callback

update display

continued interaction

Server

query

result

DB

AJAX result (JSON)
Demonstrating Asynchronicity of Handling

- Make the database server respond slowly:
  \[ \texttt{sleep(10);} \] before sending the answer

- Make the currently displayed results interactive:
  - Add “remove” button in each row
  - Add event handler for removing the row (in parallel to waiting for server!)

```javascript
$.ajax({
    url: 'ajaxreq_result_json.html',
    success: function(data) {
        $('#results tbody').append(
            '<tr><td>'+data.code+'</td>'+ ... +
            '<td><input type="button" value="Remove"></input></td></tr>
        ).last().find('input').click(
            function() {
                $(this).parents('tr').remove();
            }
        );
    }
}).done();
```
AJAX Functionality (Without Using jQuery)

- Main functionalities required:
  - Construction of a request to be sent to the server
  - Sending a request to the server
  - Waiting (asynchronously) until server responds
  - Calling functions to analyze server response

- All these functionalities are realized in one single object (in the sense of object-orientation):
  - XMLHttpRequest
Basic Control Flow

1. Invoke some javascript
2. Setup request object
3. Request callback
4. Send HTTP request
5. Return control
6. HTTP Response
7. Update page DOM

http://www.ibm.com/developerworks, Dojo framework
XMLHttpRequest (XHR)

- Outlook Web Access for Internet Explorer 5 (end 90s):
  - XMLHttpRequest object invented at Microsoft
  - Realized as ActiveX object
- Mozilla 1.4 (Netscape 7.1) and derivates (including Firefox):
  - Native XMLHttpRequest object for JavaScript
  - Independent of Active X
- Other manufacturers:
  - Followed step by step: Konqueror, Apple Safari, Opera, iCab
- Since Internet Explorer 7 ActiveX no longer required
  - Just JavaScript
- Under W3C standardization (Level 2 Working Draft January 2012)
- Long term situation for creating XMLHttpRequest object will be:
  \[ \text{var XMLHttpRequest = new XMLHttpRequest;} \]
- Currently we have to fight with browser incompatibilities!
  - Frameworks like Prototype or jQuery can help
Platform Independent Creation of XMLHttpRequest

```javascript
var XMLHTTP = null;
if (window.XMLHttpRequest) {
    XMLHTTP = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    try {
        XMLHTTP = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (ex) {
        try {
            XMLHTTP = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (ex) {}
    }
}
```

For older IE versions than 6.0
IE >= 7.0 or standard
Construction of an HTTP Request

- **open()** method of XMLHttpRequest object
  - Note: No interaction with the server yet, despite the name!

- Required parameters:
  - HTTP method: GET, POST or HEAD
  - URL to send the request to

- Optional parameters:
  - Boolean indication whether to use asynchronous or synchronous treatment (default asynchronous = true)
  - Username and password for authentication

- Examples:
  `XMLHttpRequest.open("GET", "fibonacci.php?fib=12")`
  `XMLHttpRequest.open("POST", "/start.html", false, un, pwd);`
Sending a Request

• Before sending: `XMLHttpRequest.setRequestHeader()`
  – Setting headers for the request
  – Recommended: `Content-Type` (MIME type)

• `XMLHttpRequest.send()`
  – Sends request to server

• Parameter:
  – In the simplest case (in particular for GET method): `null`
  – For more complex cases: "Request entity body" is given as parameter
    » Mainly for POST method
States of an XMLHttpRequest Object

0 UNSENT
Just created

1 OPENED
Request constructed, sending

2 HEADERS_RECEIVED
Header part of response arrived

3 LOADING
Body is being received

4 DONE
Response has been received completely
Asynchronous Reaction by Event Handler

- In order to react to the received response:
  - Function has to be called when state 4 is reached

- Registering an event handler:
  - Callback function, called when event takes place

- Registering an Ajax event handler:
  - Callback method registered with XMLHttpRequest object
  - Event `readystatechange`, called at any state change

```javascript
XMLHTTP.addEventListener("readystatechange", function);
```

- Testing for the current state by attributes of XMLHttpRequest object:
  - `readystate` gives current state (as number)
  - `status` gives return code, `statusText` gives associated text

- Returned response: `responseText` and `responseXml` attributes
Example 2 (Very Simple Request), Without jQuery

User

Browser

local JS

XMLHttpRequest object

HTTP request

Server

data

file
Example 2 (Very Simple Request)

```html
<body>
  <p>The following text is replaced with data retrieved from server (data.txt):</p>
  <hr/>
  <p id='text'>Text to be inserted here</p>
  <hr/>
...
  <script type = "text/javascript">
    var XMLHttpRequest = new XMLHttpRequest();
    document.addEventListener("DOMContentLoaded", function() {
      XMLHttpRequest.open("GET", "http://localhost/~hussmann/data.txt", true);
      XMLHttpRequest.addEventListener("readystatechange", function() {
        if (XMLHttpRequest.readyState == 4) {
          alert("Status: " + XMLHttpRequest.statusText);
          var d = document.getElementById("text");
          d.innerHTML = XMLHttpRequest.responseText;
        }
      }, false);
      XMLHttpRequest.send(null);
    }, false);
  </script>
</body>
```

ajax/simplerequest.html
Example XML Data

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ResultSet totalResultsAvailable="24900000"
    totalResultsReturned="10">
    <Result>
        <Title>AJAX - Wikipedia</Title>
        <Summary>Background about the web development technique for creating interactive web applications.</Summary>
    </Result>
    <Result>
        <Title>Ajax: A New Approach to Web Applications</Title>
        <Summary>Essay by Jesse James Garrett from Adaptive Path.</Summary>
        <Url>http://www.adaptivepath.com/p...s/000385.php</Url>
    </Result>
    <Result>
        <Title>AFC Ajax</Title>
        <Summary>Official site. Club information, match reports, news, and much more.</Summary>
        <Url>http://www.ajax.nl/</Url>
    </Result>
</ResultSet>
```

From C.Wenz
AJAX Program Creating a HTML Table from XML

- HTML text template (coded in HTML on the result page):

```html
<body>
  <p>
    <span id="number">0</span> of 
    <span id="total">0</span> hits:
  </p>

  <table id="hits">
    <thead>
      <tr><th>Title</th><th>Description</th><th>URL</th></tr>
    </thead>
  </table>
</body>
```

Script has to fill the missing data from XML response. Basic structure of script as above.

Adapted from C.Wenz
AJAX Callback Function for XML Using DOM (1)

function() { //event handler for readystatechange

    if (XMLHTTP.readyState == 4) {
        var xml = XMLHTTP.responseXML;

        var number = document.getElementById("number");
        var total = document.getElementById("total");
        number.innerHTML = xml.documentElement.getAttribute("totalResultsReturned");
        total.innerHTML = xml.documentElement.getAttribute("totalResultsAvailable");

        var hits = document.getElementById("hits");
        var tbody = document.createElement("tbody");

        var results = xml.getElementsByTagName("Result");

        ...
    }
}
... for (var i=0; i<results.length; i++) {
    var line = document.createElement("tr");
    var title = document.createElement("td");
    var description = document.createElement("td");
    var url = document.createElement("td");
    var titletext, descriptiontext, urltext;
    for (var j=0; j<result[i].childNodes.length; j++) {
        var node = result[i].childNodes[j];
        switch (node.nodeName) {
            case "Title":
                titletext = document.createTextNode(
                    node.firstChild.nodeValue);
                break;
            case "Summary":
                descriptiontext = document.createTextNode(
                    node.firstChild.nodeValue);
                break;
            case "Url":
                urltext = document.createTextNode(
                    node.firstChild.nodeValue);
                break;
        }
    }
}
AJAX Callback Function for XML Using DOM (2)

```javascript
... for (var i=0; i<ergebnisse.length; i++) {
    ...
    for (var j=0; j<ergebnisse[i].childNodes.length; j++) {
        ...
        title.appendChild(titletext);
        description.appendChild(descriptiontext);
        url.appendChild(urltext);
        line.appendChild(title);
        line.appendChild(description);
        line.appendChild(url);
        tbody.appendChild(line);
    }
    hits.appendChild(tbody);
}
```
A More Realistic Example

• Using a Web service for post code lookup
  
  - http://api.geonames.org/postalCodeLookupJSON?
    postalcode=pc & country=cy?
    username=registered_user

  - Returns a JSON text object containing descriptions about the location
    » Administrative region names, place name, latitude, longitude

• Example:
  - http://api.geonames.org/postalCodeLookupJSON?
    postalcode=80333&country=DE

  - gives the following result:

    - {"postalcode": [
      "adminCode3":"09162","adminName2":"Oberbayern",
      "adminName3":"München","adminCode2":"091",
      "postalcode":"80333","adminCode1":"BY",
      "countryCode":"DE",
      "lng":11.5668,"placeName":"München",
      "lat":48.1452,"adminName1":"Bayern"]}

Post Code Example (1)

- HTML:

```html
<!html>...
<body>
  <label for="country">Country</label>
  <select id="country">
    <option value="DE" selected>Germany</option>
    <option value="UK">UK</option>
  </select>
  <br/>
  <label for="postalCode">Postal Code</label>
  <input type="text" id="postalCode" value="82327"><br/>
  <input type="button" id="search" value="Search place name"><br/>
  <label for="placeName">Name</label>
  <input type="text" id="placeName" size=28>
</body>
```

- followed by JavaScript...:
Post Code Example (2)

• ... followed by JavaScript:

```javascript
<script type = "text/javascript">
  var XMLHTTP = new XMLHttpRequest();
  XMLHTTP.addEventListener("readystatechange", function() {
    if (XMLHTTP.readyState == 4) {
      var p = document.getElementById("placeName");
      var resultobj = JSON.parse(XMLHTTP.responseText);
      p.value = resultobj.postalcodes[0].placeName;
    }
  }, false);

  ...continued...
</script>
```
...continued:

    document.getElementById("search").
    addEventListener("click", function() {
        var country =
            document.getElementById("country").value;

        var postalCode =
            document.getElementById("postalCode").value;

        if (XMLHTTP.readyState != 0)
            XMLHttpRequest.abort();

        XMLHttpRequest.open("GET",
            "http://api.geonames.org/postalCodeLookupJSON?"
            + "postalcode=\"+postalCode+\"&country=\"+country+"
            + "&username=XXXX\", true);

        XMLHttpRequest.send(null);
    }, false);

    </script>
Postcode Lookup “As You Type”

• Using the preceding example, change one line:
  ```javascript
document.getElementById("postalCode").addEventListener("input", function() {...}, false);
```

• Continuously sending requests when a character is typed

• Can evaluate incomplete input
  – Example: UK/LA1 (complete for instance to LA1 4WY)
Problems with AJAX

• Back button
  – Browsers do not store dynamically modified pages in history

• Bookmarks
  – It is difficult to set a bookmark at a specific state of a dynamically created flow of pages
  – Option: use document-internal anchors (#)

• Indexing by search engines
Chapter 3: Web Paradigms and Interactivity

3.1 AJAX: Asynchronous Interactivity in the Web
3.2 Paradigms for Web-Based Communication
3.3 Reverse AJAX and COMET
3.4 Web Sockets and Web Messaging
3.5 Web Workers
Basic Web Paradigms: Documents

- **HTML:**
  - Originally intended for scientific papers: Limited structure
  - Purely static
  - Not object-oriented

- **HTML5:**
  - More flexible structure, graphics output, input elements, media playback

- **DOM:**
  - Dynamic changes of documents

- **CSS:**
  - Separation content/presentation, presentation classes

- **JavaScript:**
  - Dynamic changes, object-orientation
Basic Web Paradigms: Communication

• HTTP:
  – Request-response architecture:
    » Requests have to be initiated by client
  – Restricted parameter syntax (keyword-value pairs)
  – Synchronicity: Client has to wait for response

• AJAX:
  – Enables asynchronous handling of requests in client

• Basic restriction to request $\rightarrow$ response remains!
  – “Client-driven” architecture
Types of Client-Server Interaction

Client-driven

- **Send data to server**
  - Example 1: Sending shopping cart contents
  - Other examples: Location update, logging

- **Pull data from server**
  - Example 2: Very simple request

- **Pull selected data from server**
  - Example 3: Database query
  - Manifold other examples

Server-driven

- **Push data from server to client**
  - Examples: New mail, breaking news, chat

- **Request data from client**
  - Examples: Status inquiry, security check
Server-Driven Applications in the Web

• Frequent and easy solution: **Polling**
  – Client sends requests to server in regular intervals

• Disadvantages:
  – Redundant load to client, server, network
  – Changes traffic characteristics
  – Limited time resolution for real-time events

• Alternatives:
  – (a) “Reverse AJAX”/”COMET” – Tricking the Web architecture
  – (b) Going beyond traditional HTTP
Chapter 3: Web Paradigms and Interactivity

3.1 AJAX: Asynchronous Interactivity in the Web
3.2 Paradigms for Web-Based Communication
3.3 Reverse AJAX and COMET
3.4 Web Sockets and Web Messaging
3.5 Web Workers

Literature:
Mathieu Carbou: Reverse Ajax, Part 1: Introduction to Comet,
Reverse Ajax with HTTP Polling

- Server event information pulled by client through regular polling
- Easily realizable in JavaScript using “setInterval()”
- High network load, imprecise timing
Reverse Ajax with Piggyback Polling

- Assuming different needs for information exchange between client and server
- Whenever a client-triggered request is processed, additional information about latest server-side events is added to the response
Reverse Ajax with the Comet Model

• Proper support for asynchronous server-side events:
  – Requires availability of a channel for the server to push new information to the client
  – Server-client connections needs to be maintained over a long period of time

• Alex Russell 2006 (Blog)
  http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/
  – Web Applications exist which use server-side events and long-lived client-server connections (Gmail GTalk, Meebo)
  – “Lacking a better term, I’ve taken to calling this style of event-driven, server-push data streaming “Comet”. It doesn’t stand for anything, and I’m not sure that it should.”
    (Both Ajax and Comet are brands for household cleaners.)
  – Other terms for the same idea: Ajax Push, HTTP Streaming, HTTP server push
    » Sometimes also Reverse Ajax...
Comet Web Application Model
Connection Management in Comet

• Comet based on *HTTP Streaming*:
  – A single TCP/IP connection is kept open between client and server
  – For instance using the “multipart response” supported by many browsers
    » Going back to the “server push” feature implemented by Netscape in 1995, e.g. to send new versions of an image by the server
    » Response is “stretched over time”

• Comet based on *Long Polling*:
  – Standard XMLHttpRequest sent by client
  – Server suspends response until event happens
    » Specific programming techniques on server required
    » Storing the request context
  – As soon as client receives response (and processes it), client sends new request (which is suspended again)
  – Relatively easy to realize with current browsers and XMLHttpRequest
Reverse Ajax with Comet

- Client request is suspended at server
- Server responds to the request each time a new server-side event happens
Chapter 3: Web Paradigms and Interactivity

3.1 AJAX: Asynchronous Interactivity in the Web
3.2 Paradigms for Web-Based Communication
3.3 Reverse AJAX and COMET
3.4 Web Sockets and Web Messaging
3.5 Web Workers

Literature:
Mathieu Carbou: Reverse Ajax, Part 2: Web Sockets,
http://websocket.org
General Idea and General Problem

• Idea:
  – Web client (browser) communicates at the same time and in the same data space with several different hosts
  – See “post code” example

• Security problem: “Cross-site scripting”
  – Web application A gets access to data from Web application B
  – In the worst case including authentication data

• Current principle in browsers:
  – Only one Web application at a time communicates with a browser instance
  – Being relaxed in new approaches (under security precautions)
Web Messaging

• HTML5 Web Messaging
  – Draft by W3C, driven by Google
  – Most recent version October 25, 2011

• Document A, if knowing about another document B, can send a (text)
  message to document B (on a different domain)

• Specific iframe in document A calls `postMessage()` referring to domain
  and window of document B.

• Document B can handle the event in event handler
  – Gets information about origin, **which needs to be checked**
  – Document B checks format of message and takes additional precautions

• Simple to use, high security risks
WebSockets

• Originated in HTML5 (WHAT Working Group)
  – HTML5 Web Sockets specification
  – Full-duplex communication channel between client and server
  – Establishment (“handshake”) client-initiated, over HTTP
  – One connection for bi-directional communication, very small latency
    » “sub 500 millisecond” latency
  – Able to traverse firewalls and proxies (port 80)
  – Secure connection can be used (HTTP/S)

• Has been separated out of HTML5
  – API developed by W3C, protocol (“ws:”) standardized as IETF RFC 6455
  – Browser support growing recently
    » Earlier unsecure version disabled
    » Secure Websockets: Firefox 6, Chrome 14, Safari 6, Opera 12, IE10
  – Server support growing
    » e.g. Java servers: Tomcat 7, GlassFish 3.1, JBoss 7, IIS 8, ASP.NET 4.5
WebSocket Client API (JavaScript)

• Connect to an endpoint (WebSocket handshake):
  
  ```javascript
  var myWebSocket =
  new WebSocket("ws://www.websockets.org");
  ```

• Associate event handlers to established connection:
  
  ```javascript
  myWebSocket.addEventListener("open", function);
  myWebSocket.addEventListener("message", function);
  myWebSocket.addEventListener("close", function);
  ```

• Send message to server over established connection:
  
  ```javascript
  myWebSocket.send("hello");
  ```

• Disconnect from endpoint:
  
  ```javascript
  myWebSocket.close();
  ```
Reverse Ajax with WebSockets

- Simple, low-latency solution
- New standard, not yet widely used – probably the way to go in future
- *Abstraction APIs* help to keep programs independent of transport
  - See e.g. socket.IO
Chapter 3: Web Paradigms and Interactivity

3.1 AJAX: Asynchronous Interactivity in the Web
3.2 Paradigms for Web-Based Communication
3.3 Reverse AJAX and COMET
3.4 Web Sockets and Web Messaging
3.5 Web Workers

Literature:
Threading in Web Browsers

• Thread = Sequence of instructions to be executed
  – May be in parallel to other threads
  – May be part of a larger process (together with other threads)

• Traditionally, Web browsing is single-threaded

• Complex Web applications (and multimedia) require multi-threading
  – Example: Asynchronous interaction in Ajax and Reverse Ajax
  – Example: Playing back a movie/sound, being still able to control it
  – Example: Synchronizing a movie with subtitles or animations
  – Example: Long loading time for multimedia document
    – user has decided to do something else
  – Example: Independent animations on a single page
    (content and advertisement)

• Web Worker:
  – Specification for light-weight JavaScript threads in browsers
  – Originated by WHATWG, now separated from HTML5
  – Supported e.g. in Safari, Chrome, Opera and Firefox
Principles for Using Web Workers

• Creating a new worker:
  – var worker = new Worker(“my_worker.js”);

• Sending a message to the worker:
  – worker.postMessage(“hello worker”);

• Receiving a message from the worker:
  – worker.addEventListener(“Message”, function, false);
  – function (event) { … event.data … }

• What a worker can do:
  – Communicate, including Web Messaging and Web Sockets
  – Send and process Ajax requests
  – Establish timers
  – Basic JavaScript (but no DOM access)
  – Web SQL databases
  – Web Workers (!)

• Shared Worker: Working with multiple documents