Chapter 2: Interactive Web Applications

2.1 Interactivity and Multimedia in the WWW architecture
2.2 Server-Side Scripting
2.3 Interactivity and Multimedia for Web Browsers
2.4 Interactive Server-Side Scripting
2.5 Interactive Client-Side Scripting
2.6 Data Storage in Web Applications

2.7 Asynchronous Interactivity in the Web (Example AJAX)
   – AJAX
   – Reverse AJAX and Comet
   – Web Sockets, Web Messaging
   – Web Workers

Literature:
Asynchronous JavaScript + HTML (AJAX)


• Catchy name for an idea which was in use already at the time:
  – Google Suggest
  – Google Maps

• Basic idea:
  – Loading data from server is decoupled from changes in the presentation

• Advantages:
  – User can interact fluidly with the application
  – Information from server is fetched at regular intervals - 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 HTML
  – E.g. using Flash and querying servers in the background
Classical Synchronous Web Application Model
Asynchronous Web Application Model
AJAX and Client-Side Scripting

• AJAX applications are programs executed in the Web browser
  – Require a runtime environment
  – Usually programmed in JavaScript

• AJAX applications need to modify or construct HTML to be displayed in the browser
  – Requires access to loaded/displayed HTML
  – *Domain Object Model* (DOM) is used for accessing and manipulating page content

• HTML5 Canvas is an interesting candidate for dynamic display of content in the browser
JavaScript Object Tree

- Elements of the displayed document and other information can be accessed and manipulated
- Navigation:
  - Mostly selection by "id"
  - Starting point is often "document" object

```
window, self, parent, top

document

navigator

frames

location

history

forms

images

embeds

anchors

... elements

(button, checkbox, radio, text, ...)
```
DOM Reminder

- DOM is a collection of functions which make it possible to access and manipulate HTML and XML documents in the browser.
- DOM is a standardized API (Application Programming Interface).
  - Usable with several programming languages.
- Examples of DOM object properties and methods:
  - `nodeName`, `nodeValue`, `nodeType`, `attributes`
  - `getElementById()`
  - `parentNode`, `hasChildNodes()`
  - `childNodes`, `firstChild`, `lastChild`, `previousSibling`, `nextSibling`
  - `createElement()`, `createTextNode()`
  - `insertBefore()`, `replaceChild()`, `removeChild()`, `appendChild()`
- Not in DOM but useful and fast:
  - `innerHTML`
AJAX and Server-Side Scripting

• AJAX applications make particular sense when the data loaded from the server changes dynamically
  – PHP scripts or other server-side dynamics
  – Database connectivity
• 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
• For ease of understanding:
  – First examples in the following deal with static content
Request Construction and Handling

• 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 XMLHttpRequest = null;
if (window.XMLHttpRequest) {
    XMLHttpRequest = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    try {
        XMLHttpRequest = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (ex) {
        try {
            XMLHttpRequest = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (ex) {}
    }
}
```
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:
  ```javascript
  XMLHTTP.open("GET", "fibonacci.php?fib=12")
  XMLHTTP.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
  – Usual JavaScript code variants for event handler registration

  » XMLHttpRequest.onreadystatechange = function;
  XMLHttpRequest.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: Very Simple Request

...  
<body>
<script type = "text/javascript">
  var XMLHttpRequest = new XMLHttpRequest();

  function dataOutput() {
    if (XMLHttpRequest.readyState == 4) {
      var d = document.getElementById("data");
      d.innerHTML += XMLHttpRequest.responseText;
    }
  }

  window.onload = function() {
    XMLHttpRequest.open("GET", "data.txt", true);
    XMLHttpRequest.addEventListener("readystatechange", dataOutput);
    XMLHttpRequest.send(null);
  }
</script>

<p id="data">Data from server: </p>
</body>
Local Data and Server Data

- Preceding example:
  - Loads data locally
  - Like URI “file:data.txt”
  - HTTP status code is “0”, status text is empty

- Usage of HTTP in the request:
  - `XMLHttpRequest.open`("GET", "http://localhost/~hussmann/data.txt", true);
  - HTTP status code is 200, status text is “OK”
    (if the file exists on the server)
AJAX and XML

• The server response (essentially text):
  – Should have a formal syntax
  – Needs to be analyzed to interpret the response
  – Plain data (strings, numbers) only for small examples

• XML
  – Supports arbitrarily structured information
  – Is fully supported by JavaScript and DOM

• Servers should return data as XML

• Problem (currently):
  – Browser incompatibilities
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 goes here -->
    </table>
</body>
```

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

Adapted from C.Wenz
Transformer Callback Function (1)

```javascript
function dataOutput() {
    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");
        ...
    }
}
```
Transformer Callback Function (2)

... 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<results[i].childNodes.length; j++) {
        var node = results[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;
        }
    }
}
Transformer Callback Function (2)

... 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);
}
“AJAJ?” ;-) Simple Serialization with JSON

- XML Serialization of data
  - Tends to be long
  - Many redundant elements
  - Occupies a lot of bandwidth
- Alternative Serialization: JSON (JavaScript Object Notation)

```
{  
  "ResultSet": 
  {  
    "totalResultsAvailable": "24900000",  
    "totalResultsReturned": 10,  
    "Result": 
    [  
      {  
        "Title": "AJAX - Wikipedia",  
        "Url": "http:\/\/en.wikipedia.org\/wiki\/AJAX"  
      },  
      {  
        "Title": "Ajax: A New Approach to Web Applications",  
        "Url": "http:\/\/www.adaptivepath.com\/p...\/000385.php"  
      }  
    ]  
  }  
}
```
A More Realistic Example

• Using a Web service for post code lookup
  
    – [http://www.geonames.org/
        postalCodeLookupJSON? postalcode=pc & country=cy]  
    – Returns a JSON text object containing descriptions about the location  
      » Administrative region names, place name, latitude, longitude

• Example:
  
    – gives the following result:
    
      ```json
      {"postalcodes": [  
        {"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:

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

• ...continued...
Post Code Example (3)

...continued:

```javascript
function search() {
    var country =
        document.getElementById("country").value;
    var postalCode =
        document.getElementById("postalCode").value;
    XMLHttpRequest.open("GET", "http://www.geonames.org/
        postalCodeLookupJSON?postalcode="+postalCode
        +"&country="+country, true);
    XMLHttpRequest.send(null);
}
var f = document.getElementById("search");
f.addEventListener("click", search, false);
</script>
```

postcode_direct.html
Postcode Lookup “As You Type”

• Using the preceding example, change two lines:
  ```javascript
  var tf = document.getElementById("postalCode");
  tf.addEventListener("input", search, false);
  ```

• Continuously sending requests when a character is typed

• Can evaluate incomplete input
  – Example: UK/LA1 (complete for instance to LA1 4WY)
JavaScript Frameworks to Simplify AJAX Code

- Example: “jQuery” framework
  - Well-known, frequently used
  - Contains many helpful functions for Web applications
  - In particular targeted at simplification of DOM usage

```javascript
$(function() {
    $('#search').click(function() {
        var url = 'http://www.geonames.org/postalCodeLookupJSON?' + 'postalcode=' + $('#postalCode').val() + '&country=' + $('#country').val() + '&callback=?' + '
        $.getJSON(url, function(data) {
            $('#placeName').val(data.postalcodes[0].placeName);
        });
    });
});
```
Sajax: PHP Framework for AJAX

• Example for a framework supporting Ajax
• Sajax (Simple Ajax)
  – [http://sajax.info](http://sajax.info)
  – Open Source
  – Framework (library) for several scripting languages, including PHP
• Abstracts from technical details of AJAX
  – Write AJAX applications without knowing about XMLHttpRequest
• Basic idea:
  – Create a server-side dynamic function (in PHP)
  – "Export" this function with Sajax (sajax_export('functionname'))
  – In the JavaScript section of the page, call sajax_show_javascript()
    (a PHP function generating JavaScript)
  – Corresponding to the server-side function, now a JavaScript function exists
    (x_functionname) which calls the server-side function asynchronously (i.e.
    a callback function is given as parameter)
Problems with AJAX

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

• Polling
  – Browsers send more requests at a more regular pace;
    i.e the base assumptions for Internet traffic engineering change

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

• Indexing by search engines
Chapter 2: Interactive Web Applications

2.1 Interactivity and Multimedia in the WWW architecture
2.2 Server-Side Scripting
2.3 Interactivity and Multimedia for Web Browsers
2.4 Interactive Server-Side Scripting
2.5 Interactive Client-Side Scripting
2.6 Data Storage in Web Applications

2.7 Asynchronous Interactivity in the Web (Example AJAX)
   – AJAX
   – Reverse AJAX and Comet
     – Web Sockets, Web Messaging
     – Web Workers

Literature:
Mathieu Carbou: Reverse Ajax, Part 1: Introduction to Comet,
Client- and Server-Side Events

- Cause for asynchronous handling of information: Unpredictable events
  - Dynamic changes of information base independently of browser reloads
- Client-side asynchronous events:
  - Caused by user interaction (other than submitting a request)
  - Examples: Typing into a text field, pointing to a location
  - Asynchronous reaction: Load associated information
  - Client *pulls* information from server
- Server-side asynchronous events:
  - Caused by external party
  - Examples: Incoming mail, user going offline
  - Asynchronous reaction: Update local information
  - Server pushes information to client
- Traditional Ajax is adequate for client-side events
- For server-side events, we have to look for *Reverse Ajax*
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
Reverse Ajax with Comet

- Client request is suspended at server
- Server responds to the request each time a new server-side event happens
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
Chapter 2: Interactive Web Applications

2.1 Interactivity and Multimedia in the WWW architecture
2.2 Server-Side Scripting
2.3 Interactivity and Multimedia for Web Browsers
2.4 Interactive Server-Side Scripting
2.5 Interactive Client-Side Scripting
2.6 Data Storage in Web Applications

2.7 Asynchronous Interactivity in the Web (Example AJAX)
   – AJAX
   – Reverse AJAX and Comet
   – Web Sockets, Web Messaging
   – 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
  – One connection for bi-directional communication, very small latency
    » “sub 500 millisecond” latency
  – Able to traverse firewalls and proxies
  – Secure connection can be used (HTTP/S)

• Has been separated out of HTML5
  – API developed by W3C, protocol standardized as IETF RFC 6455
  – Browser support growing recently
    » Earlier unsecure version disabled
    » Secure Websockets: Firefox, Chrome, 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);
  -- or myWebSocket.onopen = ...
  myWebSocket.addEventListener("message", function);
  -- or myWebSocket.onmessage = ...
  myWebSocket.addEventListener("close", function);
  -- or myWebSocket.onclose = ...
  ```
- 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: Example Socket.IO

• Abstraction APIs abstract away from the concrete transport mechanism
• Socket.IO API (JavaScript)
  – Chooses between
    Web Sockets
    Flash Sockets
    Comet Long Polling,
    Comet HTTP Streaming
    forever iFrames
    JSONP polling
    depending on circumstances
  – Originally developed for Node JS (JavaScript engine for servers)
  – Symmetrical usage of JavaScript on client and server side
    » or usage of Java servlets on server side
Chapter 2: Interactive Web Applications

2.1 Interactivity and Multimedia in the WWW architecture
2.2 Server-Side Scripting
2.3 Interactivity and Multimedia for Web Browsers
2.4 Interactive Server-Side Scripting
2.5 Interactive Client-Side Scripting
2.6 Data Storage in Web Applications

2.7 Asynchronous Interactivity in the Web (Example AJAX)
   – AJAX
   – Reverse AJAX and Comet
   – Web Sockets, Web Messaging
   – 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