Web development

Client-side programming

Rich Internet Applications and AJAX
Rich Internet Application

- Rich Internet applications (RIA) are web applications that have the features and functionality of traditional desktop applications.
- RIAs typically
  - transfer the processing necessary for the user interface to the web client
  - keep the bulk of the data (i.e., maintaining the state of the program, the data etc) back on the application server.
Main goals of RIAs

- Most sophisticated RIAs exhibit a look and feel approaching a desktop environment.
  - **Richer.** User-interface behaviors not obtainable using only the HTML widgets available to standard browser-based Web applications: drag and drop, using a slider to change data, calculations performed by the client and which do not need to be sent back to the server, ...
  - **More responsive.** The interface behaviors are typically much more responsive than those of a standard Web browser that must always interact with a remote server.
Performance of RIAs

- Client/Server balance. The demand for client and server computing resources is better balanced. This frees server resources, allowing the same server hardware to handle more client sessions concurrently.
Performance of RIAs

- **Asynchronous communication.** The client engine can interact with the server without waiting for the user to perform an interface action such as clicking on a button or link. This allows the user to view and interact with the page asynchronously from the client engine's communication with the server.

  Example: prefetching (an application anticipates a future need for certain data, and downloads it to the client before the user requests it)
Performance or RIAs

- **Network efficiency.** Network traffic may be significantly reduced because an application-specific client engine can be more intelligent than a Web browser when deciding what data needs to be exchanged with servers.
  - Less data is being transferred for each interaction, and overall network load is reduced.
  - However, use of asynchronous prefetching techniques can neutralize or even reverse this potential benefit.
**AJAX definition**

- Asynchronous JavaScript And XML.
- AJAX is a type of programming made popular in 2005 by Google (with Google Suggest).
- AJAX is not a new programming language, but a new way to use existing standards.
- With AJAX you can create better, faster, and more user-friendly web applications.
- AJAX is based on JavaScript and HTTP requests.
Key enabling technology

- With AJAX, your JavaScript can communicate directly with the server, using the JavaScript XMLHttpRequest object.
- By using the XMLHttpRequest object, a web developer can update a page with data from the server -- after the page has loaded!
- The XMLHttpRequest object is supported in Internet Explorer 5.0+, Safari 1.2, Mozilla 1.0 / Firefox, Opera 8+, and Netscape 7.
- http://www.w3.org/TR/XMLHttpRequest/
XMLHttpRequest – the name

- The name of the object is *wrong*, but maintained for historical reasons:
  - May receive any text-based content, not just XML
  - May use also HTTPS, not just HTTP protocol
  - May handle both Requests and Responses, of all HTTP methods
interface XMLHttpRequest {
    // event handler
    attribute EventListener onreadystatechange;

    // state
    const unsigned short UNSENT = 0;
    const unsigned short OPENED = 1;
    const unsigned short HEADERS_RECEIVED = 2;
    const unsigned short LOADING = 3;
    const unsigned short DONE = 4;
    readonly attribute unsigned short readyState;
Standard definition

// request
void open(in DOMString method, in DOMString url);
void open(in DOMString method, in DOMString url, in boolean async);
void open(in DOMString method, in DOMString url, in boolean async, in
DOMString user);
void open(in DOMString method, in DOMString url, in boolean async, in
DOMString user, in DOMString password);
void setRequestHeader(in DOMString header, in DOMString value);
void send();
void send(in DOMString data);
void send(in Document data);
void abort();
// response
DOMString getAllResponseHeaders();
DOMString getResponseHeader(in DOMString header);
readonly attribute DOMString responseText;
readonly attribute Document responseXML;
readonly attribute unsigned short status;
readonly attribute DOMString statusText;
};
Request states

- **UNSENT** = 0
  - The request is not initialized
- **OPENED** = 1
  - The request has been set up
- **HEADERS_RECEIVED** = 2
  - The request has been sent
- **LOADING** = 3
  - The request is in process
- **DONE** = 4
  - The request is complete
**XMLHttpRequest properties**

- **onreadystatechange**
  - stores the function that will process the response from a server
  - `xmlHttp.onreadystatechange = function() { ... }`

- **readyState**
  - holds the status of the server’s response. Each time `readyState` changes, the `onreadystatechange` function will be executed.

- **responseText**
  - the data sent back from the server can be retrieved with the `responseText` property
Methods

- open(method, url, async, user, password)
  - method = “GET”, “POST”
  - url = complete URL to request
  - async = true/false (optional, default=true)
  - user, password (optional)
  - Interrupts any on-going send()
- setRequestHeader(header, value)
  - Adds a new header to the HTTP Request
  - Content-Type is one common header to send
    - Examples: text/xml, application/xml
Methods

send(data)

- Initiates the request
- data = HTTP request body (optional)
  - May be a Document or DOMString
- The URL was already given in open()
- send() terminates immediately if async==true, but transfer continues in the background
  - Generates readystatechange events
- send() transfers data synchronously if async==false
Methods

- `getAllResponseHeaders()`
  - Return all response headers as a single string, with headers separated by CR+LF
  - Invalid if UNSENT or OPENED

- `getResponseHeader(header)`
  - Returns the value of a single header
  - Invalid if UNSENT or OPENED
Receiving the response body

- `responseText` of type `DOMString`
  - If LOADING (partial body) or DONE
  - Allow access to a “raw string” of the response body

- `responseXML` of type `Document`
  - Only if DONE
  - For text/xml (or application/xml or *+xml) content types, otherwise null
  - Allows access to the DOM of the XML document
Example

- Create a standard HTML form with two text fields: username and time.
- The username field will be filled in by the user and the time field will be filled in using AJAX.
- No submit button is needed.
Example

<html>
<body> <form name="myForm">
Name: <input type="text" name="username" />
Time: <input type="text" name="time" />
</form> </body>
</html>
Creating an XMLHttpRequest object

```
<script type="text/javascript">
function ajaxFunction()
{
    var xmlHttp;
    xmlHttp=new XMLHttpRequest();

    ...

}
</script>
```
<script type="text/javascript">
function ajaxFunction()
{
var xmlHttp;
try {
    // Firefox, Opera 8.0+, Safari
    xmlHttp=new XMLHttpRequest();
}
catch (e) {
    // Internet Explorer
    try { // Internet Explorer 6.0+
        xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
    }
    catch (e) {
        try { // Internet Explorer 5.5+
            xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
        }
        catch (e) {
            alert("Your browser does not support AJAX!");
            return false;
        }
    }
}
</script>
Calling the server

xmlHttp.open("GET","time.jsp",true);
xmlHttp.send(null);
Processing the response

xmlHttp.onreadystatechange = function () {
    if (xmlHttp.readyState == 4) {
        // Get the data from the server's response
        document.myForm.time.value = xmlHttp.responseText;
    }
}
Attaching to an event

<form name="myForm">
Name: <input type="text" onkeyup="ajaxFunction();" name="username" />

Time: <input type="text" name="time" />

</form>
**Complete example**

```html
<html>
<body>
<script type="text/javascript">
function ajaxFunction()
{
    var xmlHttp=new XMLHttpRequest();

    xmlHttp.onreadystatechange=function()
    {
        if(xmlHttp.readyState==4)
        {
            document.myForm.time.value=xmlHttp.responseText;
        }
    }

    xmlHttp.open("GET","time.asp",true);
    xmlHttp.send(null);
}
</script>
<form name="myForm">
Name: <input type="text" onkeyup="ajaxFunction();" name="username" />
Time: <input type="text" name="time" />
</form> 
</body>
</html>
```
AJAX behavior

Classic web application model (synchronous):

- Client:
  - User activity
  - Data transmission
  - System processing
  - User activity
  - Data transmission
  - System processing

Ajax web application model (asynchronous):

- Client:
  - Browser UI
  - User activity
  - Input to Ajax engine
  - Data transmission
  - Server-side processing

- Ajax engine:
  - Client-side processing
  - Input to Ajax engine
  - Data transmission
  - Server-side processing

- Server:
  - Server-side processing
  - Data transmission
  - Server-side processing

Time flow:

- Data transmission
- Server-side processing

Jesse James Garrett / adaptivepath.com
Exercise 1

- Create an auto-complete feature for entering the name in a FORM
- For every typed letter, an associated text must be updated, reflecting the list of all possible names with those initial(s)
- Once submitted, the name adds up to the list
- Clicking on the suggestion auto-fills the box

Name: Jo

Suggestions: Joe, Joseph, John

SUBMIT
Exercise 2

- Create a FORM for entering the name of a city, based on two drop-down menus (<select> tags).
  - The first <select> contains the list of all provinces (AO, BO, CN, MI, TO, ...) 
  - The second <select> contains the list of all cities in the province
- Every time the user changes the province, then the list of cities MUST be updated
- The form may be submitted only if information is complete
References

- http://www.w3schools.com/ajax/
- http://www.w3.org/TR/XMLHttpRequest/