AJAX: Rich Internet Applications

Web Programming

Uta Priss
ZELL, Ostfalia University

2013

Outline

Rich Internet Applications

- A combination of desktop and web applications.
- Easy to install on the client (just needs browser).
- Client engine; "fat" client.
- Operating system neutral.
- Asynchronous communication (reload without refresh).
- Reduced server-client communication.
- Might even work off-line.

(The term "Rich Internet Applications" was coined by Macromedia in 2002.)

Examples

- Google Maps: adjacent maps are "pre-fetched".
- Microsoft Silverlight: programmable plugin for graphics, audio, video.
- Curl: web programming language uses Curl runtime environment plugin (not to be confused with cURL).
- Adobe Flex: based on Flash and J2EE.
- JavaFX (Sun Microsystems).
- Google Web Toolkit: produces AJAX applications from Java code.
- Browser-based "operating systems" (EyeOS, G.ho.st,, etc).

Older similar technologies

- Remote Scripting (Microsoft, since 1998)
- Rich web applications/clients
- Java Applets
- DHTML: HTML + JavaScript + CSS + DOM;
  changes content after the page has loaded;
  eg. rollover buttons, drop-down menus
- Adobe Flash

Shortcomings of Rich Internet Applications

- Network traffic can increase if too much data is pre-fetched.
- Initial download time can be long.
- Requires JavaScript (might be disabled, might be slow).
- "Sandboxing": incomplete access to system resources.
- Traditional http-based network monitoring techniques don't work, because of the asynchronous requests.

What is AJAX

AJAX: Asynchronous JavaScript And XML

- Used for creating interactive web applications or rich Internet applications.
- Update pages "on the fly".
- Retrieve data from the server asynchronously in the background.

The term AJAX was coined by Jesse J. Garrett in 2005.

Possible since 1997 (IFrame in IE and Layers in Netscape).

What is AJAX NOT

AJAX is not ...

- a programming language;
- a server technology.

Because it uses standard programming languages and browser technologies.

Technologies used

- XHTML and CSS for presentation
- DOM for dynamic display of and interaction with data
- XML and XSLT or JSON etc for interchange and manipulation of data
- XMLHttpRequest object or IFrames etc for asynchronous communication
- JavaScript or VBScript etc to bring these technologies together

How does it work:

- User triggers an HTML event, such as onClick or onMouseOver.
- JavaScript sends HTTP request to server.
- Server supplies a file (XML, HTML or text) as normal.
- JavaScript in the current page selects part of the file for display.
- JavaScript statement displays the selected part.

Pros and cons of AJAX

Pros:

- Increase speed, reduce bandwidth.
- More interactivity.
- More complex applications (e.g. email clients)

Cons:

- Browser's "back" button and bookmarking may not work.
- Not indexed by search engines.
- Accessibility issues: people who use screen readers and other specialist browsers may not be able to access the content.
- Will not work if JavaScript disabled.
- Even more possibilities for errors, browser incompatibility etc.

The XMLHttpRequest (XHR) API

For example:

```
var request = new XMLHttpRequest()
request.open("GET",url,true)
request.send(null)
request.onreadystatechange=stateChanged()
function stateChanged() {
      if (request.readyState == 4) {
         alert(request.responseText)
      }
}
```

The XMLHttpRequest Methods

- request = new XMLHttpRequest() : defines new request object
- request.open("GET",url,true) : "true" means asynchronous, don't wait for "send"
- request.send(null) : send the request
- request.onreadystatechange : defines the event handler
- request.readyState : "4" means finished
- request.responseText : the response as a string
- request.responseXML : the response as XML

Combining with DOM

If the response is returned as XML, DOM can be used to extract content:

```
xmldoc = request.responseXML;
node = xmldoc.getElementsByTagName('...').item(0);
alert(node.firstChild.data);
```

What can JavaScript do with the content?

This code shows how “replace this” is replaced by “value” when the user clicks (without refreshing the page):

```
<span id='n'>replace this</span>
onclick="document.getElementById('n').innerHTML='value'"
```

For AJAX, the “document.getElementById ...” can be placed into the onreadystatechange Event Handler.

Other means of sending content

Apart from sending content as XML, it can also be sent using JSON (JavaScript Object Notation):

```
{
"Name": "John Doe",
"Age": "25",
"address": {
      "streetAddress": "10 Colinton Road",
      "postalCode": "EH10 5DT",
      "city": "Edinburgh"
      }
}
```

XML versus JSON

XML has more features (validation, many tools and predefined formats).

JSON is simpler, smaller, easier to parse. It is supported by other languages (for example, PHP) as well.

Security: Server-side

Transmitting data between client and server always implies security risks!

On the server-side: this is mostly the normal security problem. The usual checks for POST/GET data and Query_Strings must be performed. If databases are involved, then the requests need to be checked for SQL injection and so on.

Security: Client-side

On the client-side: this depends on the security of JavaScript. Using "eval()" for parsing JSON is dangerous. There is a danger of cross-site request forgery, etc. If third party advertisements are inserted in a page, then no single developer is in charge of the code.

Unfortunately, the user is not in control of the code. The user only has a choice of turning JavaScript on or off and making sure that the latest browser version is installed.

Summary:

- JavaScript allows replacing content without refreshing the page (for example, using the tag) .
- The XMLHttpRequest API facilitates retrieving content from server-side files or remote websites.
- The XMLHttpRequest methods can run in separate threads, without stopping the script $\Rightarrow$ asynchronous requests.
- The requested website can be a server-side script which might access databases or other complex functionality.
- Complex content can be sent as XML (processed by DOM) or as JSON.

Javascript development tools

In order to develop and debug complicated Javascript (including remote scripts, AJAX, etc), one should use tools:

- Firefox: Firebug, Web Developer Extension
- Internet Explorer Developer Toolbar,
  Visual Web Developer 2008 Express Edition, Visual Studio
- Safari: Web Inspector
- Opera: Dragonfly

Remote content

The XMLHttpRequest API allows to include remote content in Javascript. This uses the HTTP protocol and is used for data.

Other possibilities: include remote Javascript code (see next slide) or use IFrame.

Including a remote script

```
<script src='http://servername/username/scriptinclude'
type='text/javascript'></script>
```

The file 'scriptinclude' contains Javascript functions etc, but no html and no <script> tags.

Security: cross-site scripting attacks

- In PHP, htmlentities() should be applied to all requested content.
- Including content via XMLHttpRequest: remote site could potentially request local page, which is outside sandbox (especially for older IE versions).
- If malicious URL is embedded in a genuine page and the user clicks on the malicious URL, a script from the malicious page could run as if it came from the genuine page.

Defence: same origin policy (SOP)

Scripts from one IP prevented from accessing data/properties of documents from different IP.

But, this does not apply to scripts loaded via <script src= ...> which are treated as same origin as the loading page.

(Mashups often use <script src= ...>.)

Search engine APIs

- In order to issue search engine queries from within a program, an API is needed.
- The program can then display the results in a custom manner or it can use the results to issue more queries.
- Google's API used to be based on SOAP. But the current version is using AJAX.
- Other search engines have similar APIs. For example, Yahoo has APIs mostly using REST.

Google's API

```
<script src='http://www.google.com/jsapi' type='text/javas
...
google.load('search', '1.0');
function OnLoad() {
var searchControl = new google.search.SearchControl();
searchControl.addSearcher(new google.search.WebSearch());
searchControl.draw(document.getElementById('searchcontrol'
searchControl.execute('VW GTI');
}
google.setOnLoadCallback(OnLoad, true);
...
<body><div id='searchcontrol'>Loading</div></body>
```