

Group 1

SAJAX: The Road to Secure and Efficient Applications

- Final Project Report -

**Thu Do, Matt Henry, Peter Knolle, Ahmad Yasin
George Mason University, 2006/07/15**

SAJAX: The Road to Secure and Efficient Applications

Thu Do, Matt Henry, Peter Knolle, Ahmad Yasin

*George Mason University Department of Information and Software Engineering
tdo@gmu.edu, mentatx@gmail.com, peter.knolle@gmail.com, ayasin@gmu.edu*

Abstract

The purpose of this paper is to present an overview of Asynchronous JavaScript and XML (AJAX) technology, describe the benefits of its use, and empirically demonstrate its efficient management of client and server resources. We show the efficiency and benefits of using AJAX with benchmarks that compare AJAX versus non-AJAX versions of three web applications. As AJAX does not offer any security in and of itself, we explore the security aspects of AJAX through the use of an instant messaging application that pairs Secure Sockets Layer (SSL) and AJAX. Analysis of the results is performed.

1. Introduction

Asynchronous JavaScript and XML (AJAX) is an approach to web developing that provides a way for client-server web applications to transfer data asynchronously, leading to richer and more responsive web experiences for end-users. The term AJAX was introduced by Jesse James Garrett in [1] in February 2005 to define the combining of the technologies XHTML, CSS, DOM, XML, XSLT, XMLHttpRequest, and JavaScript as seen in such applications as Google Suggest and Google Maps. Since then many different AJAX powered web applications have been produced, but traditional (non-AJAX) web applications are still being produced as well. We provide a comparison of AJAX and non-AJAX applications and a discussion of combining security and AJAX.

First we discuss the basics of AJAX and describe its benefits over non-AJAX web applications. Then we show, through the use of three benchmarks, that AJAX applications transfer significantly fewer bytes between the client and server than non-AJAX applications. Finally, since AJAX provides no security in and of itself we discuss security and AJAX by pairing AJAX with SSL in a secure instant messaging program.

2. Basics of AJAX

The key to AJAX is the asynchronous JavaScript request which is performed by the XMLHttpRequest (XHR) JavaScript object. Almost all modern browsers provide an implementation of the XHR object including the two most used browsers, Microsoft's Internet Explorer [2] and Mozilla's Firefox [3] and in April 2006 the W3C provided a specification [4] for the XHR. While the X in AJAX represents the XML response from the server, it is not necessary for the responses to be XML. As AJAX is becoming more widespread transport formats other than XML are becoming popular, the most notable being JavaScript Object Notation which is more commonly referred to as JSON. The typical AJAX interaction follows.

2.1. A typical AJAX interaction

1. An XHR object is created.

```
// Firefox 1.5
xhr = new XMLHttpRequest()

// Internet Explorer 6
xhr=new
ActiveXObject(Msxml2.XMLHTTP)
```

2. XHR assigns an onreadystatechange event handler to handle the server's response.

```
xhr.onreadystatechange=
function ( ) { // handle it };
```

3. XHR makes an asynchronous HTTP POST or GET request to the server.

```
xhr.open("GET",
"http://xxx.yyy.com/AjaxHandler
?param1=value1", true);
xhr.send(null);
```

4. The server processes the request and sends back a response.

```
// Code varies depending on the
// server.
```

5. XHR receives the response in its onreadystatechange event handler and processes it.

```
// Usually DOM manipulation
```

3. Benefits of AJAX

AJAX applications provide a richer and smoother end-user experience than traditional non-AJAX applications. In a typical non-AJAX application the end-user makes some changes to the page, a request is sent to the server, the server processes the request and a whole new page is written back to the client. In [5] it is noted that this typical request-wait-response-wait pattern is extremely disruptive and when coupled with network latency it can be very irritating to end-users. With every request-response interaction, the entire page gets transferred across the network to the client and the end-user is affected by having to see and wait for the entire page to reload. With AJAX that pattern is no longer necessary. Requests are made asynchronously to the server, the server processes the request and only the information necessary for the client to make changes to the page is sent back to the client. The amount of data transferred across the network from the server to the client is reduced and the end-user does not see or have to wait for the entire page to reload.

4. Benchmarks

We implemented three benchmark web applications to measure the efficiency, in bytes transferred, of an AJAX version of each versus a non-AJAX version. The server that was used was Apache Tomcat 5.5.17 using Servlets and JSP and the client was Mozilla's Firefox 1.5. Ethereal was used to record the number of bytes being transferred. In all benchmarks the AJAX version transferred significantly fewer bytes than the non-AJAX version. Also of note is that in the non-AJAX versions the end-user experience was much worse, because with each request to the server the entire page was refreshed. However, with the AJAX version only the parts of the page that needed to be updated were refreshed with each request. The results are summarized in table 1.

Table 1. Benchmark Results (average bytes transferred)

	AJAX	Non-AJAX	Percent Difference
1. Auto-populated drop down boxes	14,300	23,783	39.9%
2. Shopping cart	9,765	12,431	21.4%
3. Auto suggest	20,347	43,578	53.3%

4.1. Benchmark One

The first benchmark was based on a typical web form that might be used to collect personal information (see figure 1). Each dropdown box is populated dynamically with data from the server, based on information that has been inputted into previous fields of the form, e.g., as soon as USA is selected as the Country, the State/Prov. box is populated with all fifty states. On average the non-AJAX version transferred 39.9% more bytes than the non-AJAX version.

First Name:

Last Name:

Email:

Age:

Country: State/Prov: Local Area:

Figure 1. Benchmark one's screen

4.2. Benchmark Two

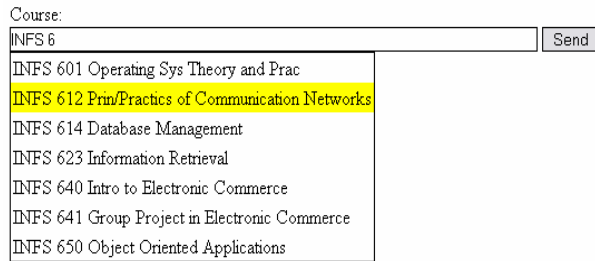
The second benchmark was a simplified version of a shopping cart that might be seen on an e-commerce site (see figure 2). Each time the quantity of an item is changed a request is made to the server and the server updates the session and sends a response back to the client. On average the non-AJAX version transferred 21.4 % more bytes than the AJAX version.

Item #	Description	Unit Price	Quantity
10000100001	White T-Shirt, XL	12.99	<input type="text" value="1"/>
20000100002	Black T-Shirt, L	12.99	<input type="text" value="1"/>

Figure 2. Benchmark two's screen

4.3. Benchmark Three

The third benchmark was an auto suggest text box that displays suggested inputs, in real time, based on what is being typed into the text box (see figure 3). Each time a character is typed into the text box a request is made to the server with the all of the text typed so far. The server then sends back a list of all possible matches for the browser to display. On average the non-AJAX version transferred 53.3% more bytes than the AJAX version.



Course:

- INFS 601 Operating Sys Theory and Prac
- INFS 612 Prin/Practics of Communication Networks
- INFS 614 Database Management
- INFS 623 Information Retrieval
- INFS 640 Intro to Electronic Commerce
- INFS 641 Group Project in Electronic Commerce
- INFS 650 Object Oriented Applications

Figure 3. Benchmark three's screen

5. Secure AJAX (SAJAX) Chat

As AJAX doesn't have any built in security mechanisms, we explored the combination of SSL and AJAX through the implementation of a secure chat application. Initially, there was great concern due to the name of the XMLHttpRequest object. An investigation was launched to determine if it would enjoy the benefits of a client browser's SSL connection, using Ethereal.

The packets intercepted via Ethereal between a test client and server proved to have unreadable payloads (as they were encrypted). This cleared the way to continue researching and developing the Secure AJAX (SAJAX) Chat application. It was decided that SSL (a tried and proven means of encryption) would be utilized for securing transmissions between the server and client.

The next task involved the high-level architecture and organization of the chat clients and server. A database (Microsoft's SQL Server 2005 Express Edition, which is free) on the server (Microsoft Windows Server 2003, Service Pack 1, running IIS 6.0) was used to marshal messages from a sender client to the appropriate recipient client. The server was also charged with the task of keeping track of each client, through the use of sessions (HTTP(S) is, after all, a stateless protocol).

Due to the expedient nature of this research, and the fact that a three-tiered development model (data-access layer, business-logic layer, and presentation layer) was decided upon for use in development on the server-side, CodeSmith (a template-based code generating tool) was used to generate the data-access and business-logic tiers. Not only did this save a great deal of time and effort, it also ensured that the code (written in ASP.NET 2.0, C#) would be consistent and follows best practices.

The client-side development provided an interesting challenge. Since the server was unable to contact a client browser without a request being made, most of the SAJAX Chat logic was implemented on the client-side. It was designed with a base window containing the client's buddy list, and several satellite windows (opened from and controlled by the base window) to manage distinct chats.

Polling was used in the base window to retrieve data from the server (all data from the server was returned in XML format). Every two seconds, an AJAX call would be made to request an updated buddy list, as well as to receive any new messages. These messages would be routed to the appropriate satellite window via the base window. If the correct satellite window was not open, the base window would spawn it. The satellite windows communicated directly with the server through AJAX calls to send messages, and were architected to handle error messages. For example, if a message was sent to a buddy who was not signed-in to the system, a message indicating this would be displayed in the sending satellite window.

Once constructed, the SAJAX Chat application worked very well. It was secure, and built using AJAX – thus achieving both of our primary goals. It was also very intuitive and easy to use. Although certainly not commercial-grade at this point, we plan to add more specialized buddy lists and chat room functionality in the next iteration of research. After such a successful initial period of development, we believe that SAJAX Chat is a fantastic base from which to continue efforts in experimenting with the capabilities of AJAX in security-oriented environments.

6. Summary

We have shown that AJAX is a valuable web development technique that can be used to make more efficient web applications. We feel that as a technology, AJAX is only beginning to be realized for

its potential applications, particularly in areas regarding security. We feel that as a technology, AJAX is only beginning to be realized for its potential applications, particularly in areas regarding security.

Through benchmarks, we have shown how dramatically AJAX can reduce the bandwidth load on a server by sending and receiving only necessary data. Further, the strain on a server's processor(s) can be reduced by allowing clients to update themselves, given the necessary information. The benchmarks also hint at a myriad of problems in web development that AJAX is much better suited to deal with than standard client/server interactions. A perfect example being the auto suggestion benchmark (Benchmark Three), which is choppy at best without using AJAX.

When this research began, it was believed that securing an AJAX application would be the most difficult portion of our research. In the end, we were able to rely on SSL, and free to concentrate efforts on the application itself. Despite some minor architectural issues in developing the client-side of SAJAX Chat (which any project goes through), development moved along rapidly.

Though SAJAX Chat is not a revolutionary discovery, it a powerful indicator of how inventive the web development community has become. With minimal time, no budget, and armed only with an idea and a few development tools, we were able to introduce a practical and feasible alternative to installed, non-secure chat clients.

Future plans call for chat room functionality and specialized buddy lists. While no timetable has been produced for the newly proposed features, it is likely that it would be accomplished relatively quickly, given the solid base of development from which we would begin. We submit our findings in the hope that it inspires new and creative applications in the realm of secure AJAX development.

7. References

[1] Garrett, J.J, "Ajax: A New Approach to Web Applications," February 2005. [Online]. Available: <http://www.adaptivepath.com/publications/essays/archives/000385.php> [Accessed June 13, 2006].

[2] Microsoft Corporation, "XMLHttpRequest Object", microsoft.com, 2006. [Online]. Available: http://msdn.microsoft.com/workshop/author/dhtml/reference/objects/obj_xmlhttprequest.asp. [Accessed June 13, 2006].

[3] XULPlanet.com, "XMLHttpRequest", xulplanet.com, 2006. [Online]. Available: <http://www.xulplanet.com/references/objref/XMLHttpRequest.html> [Accessed June 13, 2006].

[4] W3C, "The XMLHttpRequest Object", www.w3.org, 2006. [Online]. Available: <http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/>. [Accessed June 13, 2006].

[5] Smith, K. "Simplifying Ajax-Style Web Development", *Computer*, vol. 39, no. 5, pp. 98-101, May 2006.