

Improving the Interactive Course Web User Interface

Clinton W. Smullen III
Stephanie A. Smullen
The University of Tennessee at Chattanooga
United States
Clinton-Smullen@utc.edu
Stephanie-Smullen@utc.edu

Abstract: AJAX web applications are proliferating. AJAX is designed to provide smoother interactivity and more responsiveness in web systems, and a web user experience closer to that of a desktop application. Students will naturally come to expect that their courseware applications will respond as quickly. Course site designers and course developers need to be aware of the advantages and disadvantages of the AJAX technology so that they can make effective use of this technology. This report addresses issues in the areas of user interface, web culture, design, software engineering, performance, and security. Awareness of these issues allows course designers and developers to more effectively utilize AJAX in educational courseware. Future developments and areas of research needed to use AJAX technology are indicated.

Introduction

Students in college today and those who will enter soon are very familiar with Web 2.0 applications through the use of popular sites such as Myspace, YouTube, and GoogleMaps. Web 2.0 applications offer, among other things, a richer user experience, one closer to that of a desktop application, than do traditional web applications (now called web 1.0). This experience offers a more sophisticated user interface, with faster and smoother interactivity, and results in a more responsive web application (O'Reilly, 05). This group of students forms the target audience for the educational and training software being developed today. They expect a different user experience than do the students from even three years ago. These students will naturally expect their courseware applications to respond more quickly and provide a better user experience with significantly more interactivity than do most educational and training websites today. Courseware offering such an interface may attract and hold the interest of these students better than the traditional web interface. Course site designers and course developers need to be aware of the advantages and disadvantages of this technology so they can make effective use of it to attract and maintain the interest of the next generation of students, and to best utilize these capabilities to meet the educational goals of the application.

Web 2.0 interfaces are commonly implemented using AJAX. AJAX (Asynchronous JavaScript And XML) is a name applied to a set of technologies (Paulson, 05) designed to improve web application responsiveness, including HTML and CSS, Dynamic HTML, client-side scripting (in either JavaScript or Java), dynamic displays using the DOM model, data exchange using XML, and synchronous/asynchronous data retrieval using XMLHttpRequest. Good references are (Crane, 05) and (McLaughlin, 05). Mature commercial examples of the use of AJAX include Google Maps and Gmail; AjaxPatterns (Ajaxpatterns, 07) categorizes several hundred websites using AJAX applications. The use of AJAX in web programming is becoming more widespread as newer versions of browsers support XMLHttpRequest objects.

An AJAX application can perform behind-the-scenes server interactions, unseen by the user, while the client-side user interface remains active. It then renders the results for the user without replacing the entire page. This differs from the usual web interaction, where the user issues a request, waits for the server to return the requested page, and then the entire visible page is replaced with the response from the server. An AJAX user interface responds dynamically to user actions, rather than waiting for an explicit user request to replace the current page. The user need not move from page to page. Instead, page updates are downloaded in the background in response to user actions at the same time the user may be doing other things.

The technical basics for AJAX are well understood. They (HTML, CSS, etc.) are covered in many typical web programming courses and training sessions. The XMLHttpRequest communication method is a

more recent development, but is implemented in most current browsers. While the technical basics may be understood, the latest research and industry findings suggest many more issues need to be considered to implement effective web applications using AJAX. Although many of these issues are related, we have organized these issues into six broad topic areas: user interface, web culture, design, software engineering, performance, and security.

User Interface Issues

Web 1.0 users are familiar with clicking a link and loading a new page. It is obvious when they have new content, since the page changes; no special feedback is necessary. AJAX gives the application the ability to update a part of a page, rather than replace the entire page. As a result, a user may not be aware that any action has occurred. Some form of interaction symbol or animation is required to inform the user of the change and to bring the change to the attention of the user. At the same time, the designer must be careful not to provide excessive messages that pop up asynchronously and confuse the user. Temporarily changing the background color of the part of the page being updated, or displaying a progress bar in that area are some of the techniques designers have adopted to inform the user of change (Wroblewski, 07).

AJAX can improve the perceived speed of an application by performing some operations locally rather than having all operations done at the server. AJAX is particularly well suited to applications that process forms as it can provide faster processing than a traditional HTML application. The performance of applications that use large data sets can be improved by pre-downloading the data sets that the user is expected to use next, as done by the GoogleMaps scrolling tiles system. How to do this in other applications is an area of research. These processing improvements add to the user interface experience by speeding the flow of information to the user and reducing the observed latency for the user. As users become exposed to more AJAX applications their expectation for faster response will grow.

Web culture issues

About 1.2 billion people use the web worldwide. These users are at least experienced with, and most are probably trained in the click-reload user interaction model of traditional web applications. Popular web browsers efficiently support this model, with features such as the “back” button, the “history” list, and the list of “bookmarks” to allow a user to easily return to marked pages. However, since AJAX does not load a new page in response to a user action, none of these browser features work as expected with an AJAX application. The browser “back” button will not return the AJAX application to its previous state. If such an operation is to be supported, the AJAX interface must provide (separately from the browser back button) an “undo” control to unwind the last change made to the page. Similarly, what is the meaning of a “history” list of pages loaded for an AJAX application that does not load new pages? The browser “history” list will not support the AJAX user interface model. The AJAX application user interface can be designed to allow a user to move among views, or to provide a longer list for “undo” operations.

In the AJAX paradigm, to “bookmark a URL” requires storing enough information to recreate the current state of the user’s session, possibly after many incremental updates. A topic of current research is how to efficiently encapsulate such state information in an HTML link, and how to correctly reassemble the view when handed such a link by a user at a later time. Problems related to this include correctly indexing AJAX pages for a sitemap, and allowing search engines to correctly reach the page containing the information desired by a user. This could be a particular problem for educational sites that cross reference topics using links.

Of immediate and practical importance is the issue of how web metrics perform for an AJAX application. Page views (loads) are the most commonly used metric to rank pages for use and popularity, or to determine advertising revenue from ads appearing on the page. In the AJAX model, page loads are not the important issue. This issue has been referred to as “the death of the page view metric” (Dignan, 07). Should every AJAX incremental update be counted the same as a traditional page load? The metrics used by Comscore and Nielsen/NetRatings (Wells, 07) now include measures of not just page views but also the time a user spends on a web page. Is the time spent on an AJAX application page a good measure of use of the page? Or should the AJAX application also provide a measure of the “level of activity” of the user? This would address

the criticism that a browser remaining open on a page provides a long view time, although the user may not be looking at the page at all.

Design issues

An important design issue is the choice of the implementation technologies. The essential elements of AJAX are partial screen updates (rather than a complete page load) in response to user actions and the use of asynchronous communications (rather than the request/response model). The AJAX programming model is not tied to a specific data exchange format (such as XML), a specific programming language (such as JavaScript), or a specific communication mechanism (such as XMLHttpRequest) (Wei, 07). The use of JavaScript, DHTML, and XML fits well with existing web development teams, and most AJAX applications use JavaScript for the clientside coding. However, other languages could be used, such as Java or Flash. The use of Java requires programmers skilled in Java programming, but Java is much more standardized than is JavaScript, and the Java runtime environments on desktops and handhelds provide for a consistent user experience across a wide spectrum of devices. To expand on this theme, Sun recently announced (Sun, 07) JavaFX (JavaFX Script and JavaFX Mobile) designed to run on an even wider variety of devices and to be easier to use than Java/Swing. JavaFX applications are claimed to run on the Java Runtime Environment. Some AJAX applications use a data exchange format other than XML (Simakov, 07), such as JSON, HTML, or plain text. While XMLHttpRequest is not a public standard, it is widely implemented. An application could use more advanced, standardized DOM features rather than the XMLHttpRequest, such as the DOM Level 3 Load and Save functionality. Such usage must wait for the support of browser software suppliers, however.

After implementation technologies, the most important design decisions require the identification of the target client platforms and devices. Traditional computer browsers present one set of design issues, while handheld platforms present different issues. Navigation is difficult for handheld devices; their small screens require zooming in and out or scrolling to display parts of a display, and scrolling interacts poorly with most UI controls used on computer browsers. This area is rapidly changing, however, and devices like the iPhone may drive changes in handheld performance and design.

Choosing the balance between clientside tasks and serverside tasks (Herrington, 07) can affect the size of the needed download as well as system performance. More complex clientside code requires a larger download, but may offload the server and can provide a more responsive application. A related emerging issue is offline use versus networked use, so called "offline AJAX" (Dojo, 07). An application that is usable disconnected from the network can extend the use of the application during times when network service is unreliable or unavailable. The needed data must be cached, and all needed functionality must be supplied in the client. When reliable network service resumes, the data must be properly synchronized with the data on the server. This capability may be especially needed in handheld devices. The design of an AJAX application to more easily support disconnected use is close to that of designing a desktop application. It requires careful design to ensure the security and integrity of both the local data and the server data.

AJAX raises numerous accessibility issues (WebAIM, 07). The W3C Content Accessibility Guidelines require that web applications function when JavaScript is disabled or not supported. One typical solution is to make a non-JavaScript, alternative interface available to the content. This increases development and maintenance efforts for the site. Another solution is to use AJAX to extend the functionality of the HTML-based content. The HTML content is still available if JavaScript is not. Assistive technologies (such as screen readers) typically have problems with dynamic HTML updates, even though the browser does support JavaScript. The application must explicitly alert the reader that a dynamic change has occurred to allow the reader to process the update. This is difficult to do, and a general solution is an area of active research.

The use of server information push versus client pull is an additional issue for designers. In certain applications it may be more efficient for the server to push updates to a client, rather than waiting for the client to request them. Other types of applications require clientside auditing capabilities. As with all auditing, these must be comprehensive, unavoidable, and tamper resistant. This is another area of research in the AJAX community.

Software engineering issues

It is widely claimed that AJAX is standards-based. However, since the underlying technologies are implemented in browsers from various software sources, the level of cross-browser compatibility depends on the compliance of browsers to web standards. There are many well known cross-browser issues, including JavaScript standards issues, DOM implementation issues, XMLHttpRequest object differences, exception handling differences, and varying support for the different levels of CSS and XML.

Software engineering issues more directly attributed to the use of AJAX include the use of synchronous versus asynchronous connections, and the need to manage possibly many concurrent open asynchronous connections. JavaScript is not known for its structure, and the resulting JavaScript code for the client can be large and complex. Significant software engineering efforts are needed to ensure an efficient, bug-free implementation. The need for extensive multiple browser testing can produce debugging difficulties. Debugging is particularly difficult when using multiple asynchronous connections, which may complete in any order.

JavaScript frameworks are being developed to speed up development (Ort, 06). These include resources such as libraries of functions to manage AJAX connections and XML data, provide cross-browser compatibility, and standardize development (Stephenson, 07). Commercial AJAX frameworks are appearing that provide special “AJAX engines” that handle many of the client-side interactions, can be easily customized for appearance and extended to support new components (Begoli, 06). Integrated development environments are appearing that support AJAX development with a visually rich toolset (Morfik, 07). While these systems promise that developers do not need to write any HTML, CSS, XML, or JavaScript, such promises have not been proven to be effective in the past. With the Google Web Toolkit (Google, 07), the developer uses a Java IDE and development tools to build the client application, and then uses the Google Web Toolkit “compiler” to translate the Java to JavaScript and HTML. A library of special UI “widgets” is available. It is claimed that both the final size of the code and the execution speed in the browser are comparable to that of hand-coded AJAX applications.

Performance issues

The first AJAX performance evaluation case study was White (White, 05), who constructed two sample applications (one using HTML and the other using AJAX), and collected the bytes transferred and the time consumed by the tasks. This AJAX application transferred on average 27% of the bytes that the traditional HTML application transferred, which resulted in a 73% performance increase. Performance increases for this sample application were also observed in the task time taken by the users to accomplish the work. Recent studies comparing the performance of actual HTML and AJAX applications (Smullen, 06; Smullen, 07) have shown that while savings of 40-70% can be achieved, under some circumstances the performance of AJAX may be worse than that of a traditional HTML application.

A typical AJAX application downloads the JavaScript code needed to implement the AJAX interface and information rendering, as well as some HTML and graphics for page layout and branding. This initial download size exceeds the size of a typical HTML application page. However, the AJAX application then only downloads the information requested or needed by the user, typically in XML form, whereas the HTML application downloads a complete HTML page after each user action. Hence the update size for an AJAX application can be significantly smaller than that of an HTML application. This smaller size produces a more responsive system and saves download time. If, however, the client browser does not cache the needed code and supporting elements, then each time the AJAX application is started, it must download these elements again, resulting in poorer performance than a standard HTML application.

As AJAX user interfaces become more complex and new features are added, the initial download size is becoming even larger. This can be observed by comparing the sizes of versions of web AJAX applications such as gmail. For a given application, a designer must balance the requirement of a significant AJAX download and the savings of smaller incremental updates provided by AJAX against the sizes of traditional server-generated HTML pages. For some applications, AJAX is the wrong choice.

In addition to download size, a complex AJAX application can burden slower clients. The local processor speed may now become the bottleneck rather than network download speed, especially for handheld devices. The efficiency of the implementation of JavaScript in the browser, including the availability of

memory, can also be cited for performance problems. Performance testing is needed, on the variety of client systems targeted by the designer, to ensure that performance is better than that of a traditional server-based HTML application. It may be difficult to generate simulated loads for such testing.

The major savings claimed for AJAX arise from the reduced size of updates, resulting in savings for network download times. Little in the way of savings at the server has been observed in actual applications (Smullen, 07). Actual user task time savings are difficult to compare, since AJAX applications implement a different user interface from an HTML application. While White (White, 05) claimed a 32% performance increase for task times using AJAX, no serious user interface experiments have been published in this area.

Security issues

Security for AJAX applications is an important, but large, topic. A Google query on AJAX security returns over seventy million results. AJAX applications need not be less secure than traditional web applications; as with any code, the security of the application depends on the developer. While many of the potential AJAX security problems are also issues with web 1.0 systems and with other approaches to web 2.0 applications, AJAX has caused attention to focus on malware and exploits using Javascript (Cgsecurity, 07; Hoffman, 06). The greater functionality of an AJAX interface makes the code more complex, and hence more difficult to get correct. However the use of AJAX development software (IDEs, frameworks, etc) may actually result in AJAX applications that are more secure than traditional web applications developed with older (or no) tools.

Javascript is clear text. It is often cached, and is not private or hidden; the browser DOM is also not hidden. Both may be manipulated by hostile code inserted into the browser, or read by a hacker interested in learning about the server(s) used, or even the credentials needed to access that service. Since an AJAX application may issue requests to the server or servers for possibly many services, this can expose more server services to a hacker than would a web 1.0 system (giving a larger "attack surface"), or make denial of service attacks easier (Stamos, 06). It certainly means that more must be secured on the server. Knowing how AJAX accesses a service allows a hacker to more easily generate fake requests. A server may not easily distinguish a real AJAX request from an AJAX request generated by attack code or even an ordinary browser request, since they look identical to the server. Using automated security tools, such as link parsers to find links to services, may not work well with AJAX, where the links are dynamically built. This can make security more difficult to assess and validate.

As with all web applications, validation on the server is still essential. An application should not rely on clientside security. The server code should not trust any data coming from the client, especially state information stored on the client, including information stored in cookies. Sensitive state information should be stored on the server. After the user is authenticated, then the client data should be cleaned and validated. This protects against JavaScript injections, SQL injections, and XML injections on both the clientside and the serverside.

Security begins with careful coding. Poorly coded and managed asynchronous service requests can cause an unintended denial of service attack on your own server. Care should be taken not to allow cross-domain service requests, as this may allow cross-site scripting (XSS) attacks. A XSS attack is where code is sent to the browser that is then executed in the browser with the user's credentials. XSS attacks can be used to steal data, take control of a user's session, run malicious code, or launch phishing scams. Traditional web applications may also be vulnerable to XSS, but Ajax allows an attacker to exploit XSS vulnerabilities in a covert manner, by making multiple requests in the background while the user is unaware that anything is happening. Allowing the server to pass requests to other services hidden inside the server's domain may simply open other systems to attack. Recently it was demonstrated (Fisher, 07) that a tool that uses XSS can install malicious code on the user's computer, and then use this code to attack other computers. As Mike Cobb states, "Ajax is not that new and it hasn't introduced new vulnerabilities, just variations of old ones. The problem is that Ajax applications tend to be very complex." (Cobb, 06)

Conclusions

With the proliferation and popularity of AJAX applications, students will expect more rapid responses and better user experiences from course sites. Educational courseware designers and developers should consider exploiting the capabilities of AJAX in their applications, particularly the use of better user interfaces, enhanced user interactivity, and faster and more controlled modifications to the browser page. Courseware offering such an interface may attract and hold the interest of these students better than the traditional web interface. While problem areas do exist, particularly with URLs and bookmarks, support for the AJAX technology is widely implemented in browsers, and can be made secure with careful design and implementation.

References

- AJAX Patterns (2007) "Mature AJAX Platform", <http://ajaxpatterns.org/Examples>.
- Begoli, E. (2006) "An Open Source AJAX Comparison Matrix", <http://www.devx.com/AJAXRoundup/Article/33209/1954>
- Cgisecurity (2007) "AJAX Security", <http://www.cgisecurity.com/ajax/>
- Cobb, M. (2006) "MySpace, YouTube successes open door to Web 2.0 dangers", http://searchsecurity.techtarget.com/columnItem/0,294698,sid14_gci1233256,00.html
- Crane, Pascarello, James (2005). Ajax in Action, Manning Publishing Company, Greenwich, CT.
- Dignan, L. (2007) "Comscore Files For IPO: Ajax Is A 'Risk Factor'", <http://internet.seekingalpha.com/article/31383>
- Dojo (2007). Dojo Offline, <http://dojotoolkit.org/offline>
- Fisher, D. (2007) "Hackers broaden reach of cross-site scripting attacks", http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci1248127,00.html
- Google (2007) Google Web Toolkit <http://code.google.com/webtoolkit/>
- Herrington, J. (2007) "AJAX and XML: Five AJAX Anti-patterns", <http://www-128.ibm.com/developerworks/xml/library/x-ajaxxml3/>
- Hoffman, B. (2006), "AJAX Security Dangers", <http://www.spidynamics.com/assets/documents/AJAXdangers.pdf>
- McLaughlin, B. (2005) "Mastering AJAX", <http://www.ibm.com/developerworks/web/library/wa-ajaxintro1.html>
- Morfik (2007) <http://www.morfik.com/>
- O'Reilly, T. (2005) "What is Web 2.0: Design patterns and business models for the next generation of software", <http://www.oreillynet.com/lpt/a/6228>
- Ort, Basier (2006) "AJAX Design Strategies", <http://java.sun.com/developer/technicalArticles/J2EE/AJAX/DesignStrategies/>
- Paulson, L. (2005) Building rich web applications with AJAX, IEEE Computer, 38(10), 2005, 14-17.

- Simakov, P. (2007) "AJAX Without XML", <http://www.softwaresecretweapons.com/jspwiki/ajaxwithoutxml>
- Smullen, C. & Smullen, S. (2006) Modeling AJAX Application Performance , 524-074, Proceedings of Web Technologies, Applications, and Services, WTAS 2006, 7/17/2006 - 7/19/2006, Calgary, Alberta, Canada, ed. J. T. Yao, http://www.actapress.com/Content_of_Proceeding.aspx?proceedingID=391
- Smullen, C. & Smullen, S. (2007) AJAX Application Server Performance, Proceedings of the IEEE SoutheastCon 2007 (CH37882), March 22-25, 2007, Richmond, Virginia, pp. 154-158.
- Stamos, Lackey (2006) "Attacking AJAX Web Applications", http://www.isecpartners.com/files/iSEC-Attacking_AJAX_Applications.BH2006.pdf
- Stephenson et. al. (2007) Prototype JavaScript Framework, <http://www.prototypejs.org/>
- Sun Microsystems (2007) JavaFX, <http://www.sun.com/software/javafx/index.jsp>
- WebAIM (2007) "Accessibility of AJAX Applications", <http://www.webaim.org/techniques/ajax/>
- Wei, C. (2007) "AJAX: Asynchronous Java + XML?", <http://www.developer.com/design/article.php/3526681>
- Wells, T. (2007) "Nielsen/NetRatings Focuses on Total Minutes", <http://www.seoachat.com/c/a/Search-Engine-News/Nielsen-NetRatings-Focuses-on-Total-Minutes/>
- White, A. (2005) "Measuring the Benefits of Ajax", <http://www.developer.com/xml/article.php/3554271>
- Wroblewski, L. (2007) "AJAX & Interface Design", http://www.lukew.com/resources/articles/ajax_design.asp