Using Ajax to Track Student Attention

Jan Newmarch
Centre for ICT, Box Hill Institute, Box Hill, Australia
and
Faculty of IT, Monash University, Caulfield, Australia

j.newmarch@boxhill.edu.au

***Abstract*—Tracking the behaviour of users of online learning systems is an important issue, but current techniques have not been able to give deep views on what users do with Web-based learning systems. This paper shows how use of Ajax can provide a richer model of how users interact with Web systems. A case study is discussed.**

***Index Terms*—Web-based learning, Ajax, student models. HTML events**

I. Introduction

Any producer of web-based material is interested in what users do with the pages they visit: what do they visit, how long do they spend there, and what do they do while there? This can be of high commercial value, as information about users can be used to revise pages in order to draw customers into a commercial site, and hopefully to spend money. In the educational domain, knowledge of a user's activities can help to build a better educational experience. The intent is to build up a model of the user and to customise the site to desirable users.

There have been three common techniques used to track user activity: web server logs [13], custom-built browsers [6], [17] or visual observation such as in a usability lab. They are all well-known to have significant drawbacks, as discussed in the next section. Recently a technique called Ajax [9] (for Asynchronous JavaScript and XML) has come to the fore. Primarily this is used to give a more interactive experience with a web site, and has been used by companies such as Google (in Google maps).

HTML 4 compliant browsers support event tracking using languages like JavaScript, such as when a user enters and leaves a page. They also allow "focus" tracking, which can occur when a user switches to, say, an email program without leaving a web page. Combined with the asynchronous aspects of Ajax, we show in this paper how this can be used to give a clearer picture of what a user is doing. We demonstrate the use of this with a formal course for teaching Linux administration.

Consider the scenario:

> Johnny has been instructed to look at some courseware in his browser. He navigates to the page, but after 5 minutes he gets bored. He switches to another tab so he can read his Google mail for 20 minutes. Then he switches back to the courseware page. After another 5 minutes he decides to talk to a friend and starts up Skype. 10 minutes later he returns to the page and finally follows a link in that page to another page of the courseware.

Simple observation of server logs would suggest that Johnny spent 40 minutes on the first page, whereas a closer examination shows that he only spent 10 minutes. This paper shows how to perform some of this closer examination.

The structure of this paper is as follows: the next section discusses current techniques for tracking user behavour. The section after that looks at Ajax and how we use this to generate information. Following this are a number of sections discussing issues arising from this use of Ajax and how to analyse the information gained. Finally an examination of actual server logs is given to show how to give a more accurate picture of a user's browsing habits, and future work is discussed.

The principal contribution of this paper is that it shows how a deeper analysis of student use of web-based courseware can be performed, and illustrates this with a case study. Similar techniques could be used in other situations.

II. Tracking users

HTTP logs are collected by HTTP servers such as Apache. Generally these logs use the Common Logfile Format [3]. These record which pages are accessed, the date, which IP address made the request, and optional other information such as referring page. These logs form a relatively simple way of measuring what users are acessing. However, they only give partial information. They show the requests that actually made it to the server: many organisations now use proxy caches, and if there is a "hit" on a cache, then the request will be handled by the proxy and not make it back to the source server. This can be alleviated by setting the Expires time for each document to zero, but breaks the value of caching.

If the user makes use of the Back button in the browser, then the document will be retrieved from the browser's own cache. This cannot be avoided except by disabling the Back button.

The principal problem is that the server logs can only show that a page is requested from a server. What is done with that page is

unknown. A user may examine it for a long time or simply discard it. Further, it is not clear whether it is a human using a browser or some automated agent such as a spider.

After any one request, if another is made to the same site then another entry is made in that server's log. This provides an upper bound to the time spent on the requested page. But the user may have been somewhere else, or may just never come back.

A second technique is to use a special-purpose browser which logs each user activity. Such browsers can record a great deal of information. There is a minor problem of getting the information back to the server. The major problem is persuading people to use such a browser. Typically this can only be done with a relatively cooperative group of people, as a research experiment.

The third technique is to bring people into a special laboratory and to physically observe their behaviour. This is expensive and time consuming, and can only be done with small groups. However, it does offer the potential for discussion about what it being done, to give a "why" as well as a "what".

III. Ajax

Ajax is technically very simple: it consists of a JavaScript call that can be made asynchronously to a web server [9]. Typically such a request carries XML data, although this is not prescribed. The browser does not pause, or refresh while such a request is made. If a reply is received, the JavaScript engine may act on this. In the case of Google Maps it caches map data of the sides of the current map for use if the user wishes to view a nearby map. Other Ajax applications may use JavaScript to manipulate the browser's DOM model, to cause apparently interactive responses. The advantage of Ajax is that it can avoid the fetch-wait-refresh cycle usual in following hyperlinks or submitting forms.

Under the Web Consortium's Document Object Model (DOM), user actions in a browser can generate events [4]. These events include loading pages, moving or clicking the mouse, and using the keyboard.

Ajax requests can be called from these Javascript events. We want to track user activity, where the interesting events are load, focus, blur and unload. There are many other events such as mouse motion etc, and it would be possible to track these as well. It is possible that HTML version 5 [5] will extend the set of events, but this is not yet standardised or consistently implemented.

The technique of this paper is to use Ajax to track load, unload, blur and focus events for each page and record them on the originating server. There they can be analysed by the web site's owner.

We include JavaScript in each page that has handlers for load, unload, blur and focus events. When a handler is called, it makes an asychronous GET call back to the server. The browser does not use any returned information, so really all that is needed is to record on the server side the time, the page, the browser and the state change. One simple way is to use the Ajax call to just get a one-pixel image, tagged with the state change as in:

```
GET /dummyimage.png?state=loading
```

This will get recorded in, for example, the ordinary Apache server logs along with the referring page, which is where the state change occurred as

```
192.168.1.11 --[21/Dec/2009:16:47:36
+1100] "GET /dummmyimage.png?state=loading
HTTP/1.1" 200 266
"http://192.168.1.11/boxhill/ict213/test.html"
"Opera/9.80 (Windows NT 6.1; U; en)
Presto/2.2.15 Version/10.10"
```

from which we see that the page `http://192.168.1.11/boxhill/ict213/test.html` was loaded into the Opera browser.

We also note two other systems which use Ajax to track user activity in different ways. The first is Robot Replay [1] which records mouse and keyboard events. This allows tracking of what a single user actually does on a page. Another is a service by Crazy Egg [7] which builds up a record of many users and shows their interaction with heat maps -the hotter a point on the map, the more users interacted with elements there. These two approaches are complementary to the one here which measures how users navigate to and from web pages.

In both of the examples considered later, browser HTML pages are generated dynamically from server-side XML files. The generators were modified to include the required JavaScript. Any Content Management System can probably do the same, to quickly allow a site to be marked up to record user events.

IV. Event generation issues

There are several different ways of attaching Javascript code to DOM objects in order to generate events. One way is to attach JavaScript code code directly to an HTML tag as in <`body onload = "sendGetRequest('http:/dummyimage.png?state=loading')"`> This is an obtrusive method as it requires modification of the HTML of the document. However, it is reliable and produced the most consistent and useful results in event generation.

The second method is to include a Javascript file which attempts to locate the relevant DOM object and assign an event handler to it. There are many tutorials (e.g. [2]) which give example code such as `window.onload = ...` This does not meet the HTML 4 specifications which state that the onload event should be attached to HTML body or frame tags. In practice, this produced inconsistent results, with some browsers failing to generate a focus event after loading, while others loaded and then unloaded, followed by focusing and blurring!

A third technique is to assign a name attribute to all body elements, search for these elements using the JavaScript `getElementsByName()` and then add an event handler to the element. This also turned out to be unreliable as the element needed to be loaded before it could be found and this invalidated the onload event.

The best method of adding event handlers for this project is to add the handler directly to the body or frame element. Where the HTML is generated from a content management system, this will need to be added to the generation mechanism.

V. Browser issues

The HTML 4 event specifications are not very precise. For example, there is no state machine specification of what and when events should be generated during page loading. The forthcoming HTML 5 is more precise, but this is not yet standardised, or consistently implemented by browser vendors. In order to use this technique, the browser must support both JavaScript and the Ajax

functions. This rules out many browsers such as Lynx and the W3C Amaya, but these have negligible market share. Even if we restrict attention to the major browsers of IE, Firefox, Safari, Opera and Chrome, there are still differences.

There are a large number of situations which could generate events. For example, focus events can be generated for a page after loading the document, by switching to it by using the Back or Forward buttons, by selecting a tab containing it, by selecting it from another application, or under most Linux GUIs, by switching from another desktop. Some events should be the result of the window system events such as switching focus between applications, while others belong to a browser such as switching focus between tabs.

Chrome is still at an early stage of development (at December, 2009) and fails many of the event generation possibilities. Even worse, I was unable to get Safari (under Windows) to generate any events. Firefox v3.0 failed to generate a focus event on tab switching, but this has been fixed in Firefox 3.5. Despite its generally poor reputation with regard to standards, all of Internet Explorer 6, 7, and 8 were consistent and complete with respect to these events.

For the current major browsers, these are summarised in Tables One and Two. I initially carried out experiments with this technology in 2006, with Firefox 1.5. That version would generate a quite extended sequence of focus and blur events until it settled down to a stable state. This made it very difficult to analyse results.

In summary, browsers should not matter, but in practice still do.

VI. Browser-generated events

Many events are generated internally by user interaction with the browser. These include loading URLs, from the menu bar, by following a link or selecting a bookmark. But they also include opening or switching between tabs, closing tabs, or using the Back and Forward buttons. These are summarised in Table One.

IE has correct behaviour for all of these, even IE6. Firefox has a few minor errors. Chrome and Opera have a large number of errors. Safari is omitted as it does not generate any events.

There are some actions which lead to different event sequences between the browsers. For example, loading a URL generates load and focus events for IE and Firefox, but only load events for Opera and Chrome. If it is possible to load a URL without giving it the focus, then Opera and Chrome would have incorrect behaviour, otherwise it may be excusable.

VII. Operating system events

Browsers run within an environment supplied by the operating system or in the case of Linux also supplied by the X Window server and whatever window manager is used. There are many cases where these should signal to applications that a change of state is occurring. The major cases are focus changes between applications' windows, which includes iconification or de-iconification of an application's windows. But these events also include shutdown, hibernation, logging a user off or in the worst case, crashing. Of course, no-one could expect an operating system to generates events in a reliable manner if it is crashing! These are summarised in Table Two, with similar results as before: IE is good, Firefox is almost right and Opera and Chrome have many errors. Safari is omitted as before.

VIII. Virtual machine issues

Virtual machine technologies have been under development for many years. These allow a computer running one operating system to host a guest virtual machine running another operating system. For example, on my Linux laptop running Ubuntu I can run virtual machines under VirtualBox [11] hosting Windows XP, Windows 7 and Fedora Linux guests. While I am within one virtual machine, the task focusing and switching mechanism occurs according to that operating system. For example, when a Windows virtual machine has the focus, then Alt-Tab will switch focus between Windows applications on that virtual machine. However, for each virtual machine there is also an escape mechanism to switch focus from the guest virtual machine back to the host operating system. For example, in VirtualBox, it is the right control key by default.

Most GUIs for the major operating systems will not allow no window to have the focus. However, this will be required for accurate tracking of focus changes. Using VirtualBox, no blur events were generated for any guest operating system applications when the virtual machine lost focus, nor focus events when the virtual machine gained focus. That means that a browser window in a guest system will believe that it still has the focus even though it has been switched to another application in the host system.

Virtual machines are not common on the desktop yet, except for application developers, However, Windows 7 includes a virtual desktop for Windows XP so there may be an increase in virtual desktop utilisation. If that occurs then there will need to be agreement on focus management between virtual machines. At present this is probably ignorable, but needs work for future standardisation.

The effect of events not being generated by the operating system or by a virtual machine will result in some pages being recorded as obtaining focus but with no blur event being recorded even when focus is lost.

IX. Server-side Analysis

Server-side programs can be used to analyse the log files. These programs can be in any language, and run in either batch or interactive modes.

I use the Apache HTTP server. The Common Logfile Format [3] includes date and time of access, referring URL (including host IP and page) and browser accessing the page. (It is possible for one browser such as Konqueror to pretend to be another such as IE, but this is usually to cope with badly designed browser-specific sites, and this practice should be decreasing.). Other Web servers may need to have their log formats adjusted to give appropriate data.

Analysis of logs must distinguish between valid users i.e. those who are searching for or using the courseware, and between

spiders trawling pages for search engines or other uses. This task may be simplified if the site is private or otherwise unknown to spiders.One simple way is to use the file `robots.txt` to exclude spiders. Otherwise, the USERAGENT string (as described in the next section) must be used to exclude spiders.

The raw data needs to be massaged to produce meaningful events, removing non-essential events.

Nevertheless, the Ajax events measure what is generated in the browser. They are not filtered by intermediate proxies nor hidden by pressing the Back button in a browser. In addition, the events are generated in any browser which understands JavaScript events, which is the majority of browsers nowadays. This mechanism does not require custom-built browsers.

X. Browser indentification In an ideal world, it should not matter which browser is used in creating the Ajax-augmented server logs. Unfortunately, Tables One and Two show that browsers still have differences in behaviour. In order to properly interpret the logs, it is still necessary to identify the browser generating the Ajax events.

Each browser (or more properly, each HTTP user agent) should send a string in each HTTP request giving the value of the USER-AGENT field [8]. Identifying the browser from this is arcane, largely due to history: Netscape pretended to be Mozilla, Internet Explorer (IE) pretended to be Netscape, and then browsers pretended to be IE [18]. A searchable list is maintained at [14].

XI. Privacy

Capturing and manipulating user activities raises issues of privacy. Just using server legs or these Ajax extensions does not impinge directly on user privacy: no user identification is performed, and since any particular student may use a variety of IP addresses (home address, cafe hot-spot, DHCP-assigned address in the Institution network or logged in to a random computer), there is little opportunity for identifying any particular user through these logs.

It is becoming common, however, to only access to courseware through a Content Management Systems (CMS) such as Blackboard [10] or Moodle [16]. These generally require login to access the courseware, and maintain a session that tracks all activity. Generally this is restricted to navigation within the CMS, and as long as the courseware does not contain internal links invisible to the CMS, is able to track page visits and thus is able to give per user statistics on page visit activity.

There is a reasonable likelihood that this CMS page visit log data can be combined with the Ajax-extended server logs to link the focus activity to particular users. Further, if these techniques were adopted by a CMS, then it could make such activity part of the normal user activity log.

As long as the activity recorded is used for academic purposes only, then this should not be a serious issue. However, if it were to be used for other purposes such as showing whether or not an international student was serious in their study activity, then it may be more contentious.

XII. Courseware in Other Formats

The techniques described in this paper use the Web formats of HTML, XHTML and XML. Many courseware designers make use of other formats such as PowerPoint or Flash. These formats are “web unfriendly” in that they do not directly follow any of the W3C standards. In particular, they do not generate DOM events and thus use of these formats cannot be tracked directly using the techniques of this paper.

Flash files can make use of the Flex programming language [15]. Flex has an event handling model similar to the DOM model: “The Flex event model is based on the Document Object Model (DOM) Level 3 events model. Although Flex does not adhere specifically to the DOM standard, the implementations are very similar.” In addition, a Flex application can communicate with JavaScript within its HTML “wrappper”. Alternatively, it can use the `HTTPService` component to communicate directly back to an HTTP server. Thus it should be possible to adapt Flash pages to the concepts of the technology given here.

An alternative to Flash may be the forthcoming HTML 5, which will allow direct use of these techniques in multimedia pages using the W3C standards.

Powerpoint 2000 supports events as well and can call VBA scripts [12]. Already, HTML 4, the DOM model and JavaScript are adequate to replace any use of PowerPoint, so although it may be possible to duplicate these techniques, it may not be necessary.

XIII. Limitations

In the introduction a scenario was posed whereby a user switched from web browsing to using Skype. Whereas the technology described here can detect loss of focus from the browser, it cannot detect that the application switched to was Skype. To do so would require far more invasive techniques than are currently available (or even desirable?).

XIV. Results

The subject ICT213 *Multi-user Operating Systems Administration* is a second year subject taught in the Bachelor of Computer Systems (Networking) at Box Hill Institute. This is a small class, of only seven students. Logs were kept over a three-week period, and no attempt was made to identify the students. The small size of this group means that results are indicative rather than statistically valid.

The structure of the courseware is that each “lecture” consists of one or two web pages. JavaScript is used so that the lecturer can display the pages in “slide mode” similar to PowerPoint, while the students usually view the pages as single documents. Other structures for courseware are of course possible, such as multiple linked small pages, and such a structure woud produce different results.

The students over the logged period loaded courseware pages 40 times. The average load time was 1014 seconds, or about 16 minutes. By contrast, the students focussed on pages 349 times. Many of these focus times were very short, less than 3 seconds, and may correspond to clicking on a page just to navigate away from it. Excluding these times, the students focussed on the pages

179 times, so that each loaded page was gained and lost the focus on average 4.5 times. The average focus time was a mere 116 seconds, less than 2 minutes!

It is clear that the attention span of this group of students is very low. It should be noted that the logs were kept on the expository material only, and no significant assessment was carried out during this period. While one may expect higher use during assessment periods, these logs will actually provide a means of testing any such assertion.

XV. Conclusion

This paper has demonstrated a technique based on Ajax for gaining more information about student interaction with courseware. While the current implemantation deals only with documents in HTML, XHTML and XML formats, it should be possible to extend it to deal with non-W3C formats such as Flash and PowerPoint. More intereating would be extensions to deal with the expected multimedia components of HTML
5.

The technique presented was essentially stand-alone. However, i should be straightforward to use this within existing Content Management Systems such as Blackoard and Moodle to give them more sophisticated reporting capabilities on user activities.

The use of virtual machines is not yet very widespread among the user community, although it is likely to spread to some extent. There is not yet a defined model of event interaction between virtual machines and their guest operating systems, and this gap needs to be filled.

References

[1] Dion Almaer and Ben Galbraith. Robot replay: Watch your users via ajax. http://ajaxian.com/archives/ robot-replay-watch-your-users-via-ajax.

[2] Stephen Chapman. Using window.onload. http://javascript.about.com/ library/blonload.htm.

[3] WWW Consortium. Common logfile format. http://www.w3.org/ Daemon/User/Config/Logging.html#common-logfile-format.

[4] WWW Consortium. Document object model (dom) level 2 events specification. http://www.w3.org/TR/2000/ REC-DOM-Level-2-Events-20001113/.

[5] WWW Consortium. A vocabulary and associated apis for html and xhtml. http://dev.w3.org/html5/spec/Overview.html.

[6] Andy Edmonds. Uzilla: A new tool for web usability testing. *Behavior Research Methods, Instruments and Computers*, 2003.

[7] Crazy Egg. See where people click: Visualise the user experience on your website. http://crazyegg.com/.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Rfc2068 -hypertext transfer protocol – http/1.1. http://www.faqs.org/rfcs/rfc2068. html.

[9] Anthony T Holdener. *Ajax: The Definitive Guide*. O'Reilly, 2008.

[10] Blackboard Inc. Blackboard. http://www.blackboard.com/.

[11] Sun MicroSystems. Virtualbox. http://www.virtualbox.org/.

[12] Stephen Rindsberg. Make your vba code in powerpoint respond to events. http://www.pptfaq.com/FAQ00004.htm.

[13] Charlie Schluting. Analyzing web server logs. http://www.serverwatch. com/tutorials/article.php/3518061/Analyzing-Web-Server-Logs.htm.

[14] Andreas Staeding. List of user-agents (spiders, robots, crawler, browser). http://user-agents.org.

[15] Adobe Systems. Adobe flex 3 developer guide. http://www.faqs.org/rfcs/ rfc2068.html.

[16] Moodle Trust. Moodle. http://moodle.org/.

[17] Ganesan Velayathan and Seiji Yamada. Behavior-based web page evaluation. *Web Intelligence and Intelligent Agent Technology Workshops,*, 2006.

[18] Nicholas C. Zakas. History of the user-agent string. http://www. nczonline.net/blog/2010/01/12/history-of-the-user-agent-string/.

Table 1: Browser behaviour for browser events

User action	Firefox 3.0	IE6/IE7/IE8	Opera 10	Chrome beta
New page loaded (from link, url, bookmark)	load focus	load focus	load	load
Focus away to another tab pane	none (error) (Firefox 3.5: blur)	blur (IE6: N/A)	blur	blur
Focus from another tab pane	focus	focus	focus	focus
Open link in same pane	unload	unload	unload	none (error)
Open link in new window	none (error)	unload	blur	blur
Open link in new tab pane	none, but no focus change	blur but no focus change (error)	blur	none, but no focus change
Browser is closed	unload	unload	none (error)	unload
Tab pane is closed	unload	unload	none (error)	unload
Back Button to another page	none	unload	none (error)	none (error)

Back Button to this page	load focus	load focus	none (errr)	none (error)
Forward button to another page	unload	unload	none (error)	none (error)
Forward button to this page	load focus	load focus	none (error)	none (error)

Table 2: Browser behaviour for system events

User action	**Firefox 3.0**	**IE6/IE7/IE8**	**Opera 10**	**Chrome beta**
Focus away to another application	blur	blur	blur	blur
Focus away to another browser window	blur	blur	blur	none (error)
Focus away to another desktop (Linux)	blur	N/A	none (error)	none (error)
Focus from another application	focus	focus	focus	focus (Linux) none (error) (Windows 7)
Focus from another browser window	focus	focus	focus	focus
Focus from another desktop (Linux)	focus	N/A	focus	none (error)
Window is closed	blur unload	unload	none (error)	unload
Iconify	blur	blur	blur	blur
De-iconify	focus	focus	focus	focus
Computer hibernates	blur focus blur	blur	none (error)	?
Restore from hibernation	focus blur focus	?	?	?