UAIS manuscript No.
(will be inserted by the editor)

The uptake of Web 2.0 technologies, and its impact on visually disabled users.

Andy Brown · Caroline Jay · Alex Q. Chen · Simon Harper

Received: date / Accepted: date

Abstract World Wide Web (Web) documents, once delivered in a form that remained constant whilst viewed, are now often dynamic with sections of a page able to change independently, either automatically or as a result of user interaction. In order to make these updates, and hence their host pages, accessible, it is necessary to detect when the update occurs and how it has changed the page, before determining how, when and what to present to the user. This can only be achieved with an understanding of *both* the technologies used to achieve dynamic updates *and* the human factors influencing how people use them. After proposing a user-centred classification of dynamic updates, this paper surveys the current state of technology from two perspectives: that of the developer, and those of visually disabled users.

For the former group we introduce some of the technologies that are currently available for implementing dynamic Web pages, before reporting on the results of experiments analysing current and historic Web pages to determine the extent of use of these technologies 'in the wild', and the trends in their uptake. The analysis shows that for the most popular 500 sites, JavaScript is used in 93%, Flash in 27%, and about one-third (30%) use XMLHttpRequest, a technology used to generate dynamic updates. Uptake of XMLHttpRequest is approximately 2.5% per year across a random selection of 500 sites, and is probably higher in the most popular sites.

When examining dynamic updates from the perspective of visually disabled users, we first report on an investigation into which technologies (Web Browser and assistive technologies) are currently used by this group in the UK: Internet Explorer and JAWS are clear favourites. We then describe the results of an experiment, and supporting anecdotal evidence, which suggests that, at best, most users can currently reach updated content, but they must do so manually, and are rarely given any indication that any update has occurred.

With technologies enabling dynamic updating of content currently deployed in about 50% of the most popular sites, and increasing by nearly 5% annually, action is urgently required if visually disabled users are to be able to use the Web. We conclude by discussing some of the issues involved in making these updates accessible.

Keywords Web accessibility · Visually Impaired · Screen readers · Dynamic Content

A. Brown, C. Jay, A.Q. Chen, S. Harper
School of Computer Science
University of Manchester
Kilburn Building, Oxford Road
Manchester, UK
Tel.: +44-161-2757821
Fax: +44-161-2756204
E-mail: andrew.brown-3@manchester.ac.uk

C. Jay
E-mail: caroline.jay@manchester.ac.uk

A. Q. Chen
E-mail: chenqa@cs.manchester.ac.uk

S. Harper
E-mail: simon.harper@manchester.ac.uk

1 Introduction

Accessibility is in a continual state of flux: technologies for enabling disabled users to access Web content are improving, but at the same time the Web is evolving. Technologists and designers are introducing new techniques, or using old techniques in innovative new ways, to create sites that are visually stimulating, and

provide a rich, interactive experience. This is Web 2.0 (O'Reilly, 2007). One example of such a technology is Asynchronous Javascript and XML — AJAX — a script-based technique that allows the content on a page to change dynamically, that is without reloading or changing page. Assistive technologies, such as screen readers (Ramen, 2008; Burks et al, 2006), are not keeping up with these changes, however, and the resultant lag has the potential to create a barrier for disabled users (Zajicek, 2007; Gregor et al, 2005). This paper aims to understand the nature and extent of the problem, and does so by examining it from the following perspectives:

The nature of the updates: What qualities can an update have? This question is explored from the point of view of a user.

Technology evolution: What technologies are available to Web developers who wish to create a site using dynamic updates, and how is the Web evolving with respect to use of these technologies?

Assistive Technologies: Which screen readers are in use, and how do they deal with dynamic updates?

In this paper we develop a taxonomy of updates (section 2), that provides a user-centered view on the range of updates; a summary of the techniques available to Web developers (section 3) provides a complementary developer-centred view. An analysis of the use of these technologies in the Web, and how this is changing (section 4) provides evidence that usage is on the increase, and that handling these updates is going to be an increasingly important requirement for assistive technologies. Unfortunately, however, there is also evidence that these types of updates are currently a real barrier for screen reader users (section 5) — the implications of this are discussed in section 6.

2 A Taxonomy of Updates

Dynamically updating regions on a Web page can be used for a variety of reasons, and with a variety of effects; understanding the range of updates in use is essential for developing accessibility solutions. Here we present a taxonomy of updates. This brief analysis of types of update is intended to be used as a basis for developing an understanding of how users interact with these updates, with some emphasis on building a model of how present them non-visually. Accordingly, this subject is approached from the perspective of the user rather than the developer, examining how the update is seen by the user, not how it is implemented. The taxonomy considers all types of update, irrespective of whether certain groups of user are currently able to perceive or make use of them.

This taxonomy is based purely on the behavioural characteristics of the updates, and not upon presentation or semantics. It is based on a largely theoretical consideration about how updates may behave, and whilst backed up with examples, a comprehensive study of their applications on the Web might highlight areas for revision. Nevertheless, it should serve to inform investigations into how users allocate attention to updates, and thus into making these updates accessible. The categories are as follows:

Timing: What causes the update to occur?
- Automatic: Updates that occur independently from any user activity.
- User-Initiated: Updates that occur as a direct result of user activity.

Page Effects: What effect does the update have on the page?
- Add: Information that the user has not seen before is added to the page.
- Remove: Information is removed from the page and not replaced.
- Replace: Existing content is changed — information that was visible before the user's action is no longer shown, but different information takes its place.
- Rearrange: The information contained in the page does not change, but its structure is modified in a way over which the user has direct control.

2.1 Design Patterns

As an example of how this taxonomy may be applied, we use it to classify the contents of the Yahoo! pattern library[1]. This library contains a set of design patterns, i.e., reusable solutions to common problems, for Web design, many of which involve dynamic page updates. Typically, the patterns are described in terms of some information that needs to be conveyed to the user, and a method for doing this. For example, the *Auto Complete* pattern[2] is described as follows (this is a summary — the library gives much more detail, including advice on when to use, why it works, and accessibility issues):

> Problem Summary: The user needs to enter an item into a text box which could be ambiguous or hard to remember and therefore has the potential to be mis-typed.

[1] `http://developer.yahoo.com/ypatterns/index.php`

[2] `http://developer.yahoo.com/ypatterns/pattern.php?pattern=autocomplete`

Solution: As the user types, display a list of suggested items that most closely match what the user has typed. Continue to narrow or broaden the list of suggested items based on the user's input.

The library contains many examples of how, and why, dynamic updates can be used to make users Web tasks more efficient. The existence of libraries such as this is only likely to increase the usage of these updates. Here, these patterns are used to exemplify application of the taxonomy described above, so those patterns that do not relate to dynamic updates (e.g., methods for presenting a user's status in a community) are not discussed.

The largest class of patterns are the *transition* patterns, which are used to highlight change on a page, to ensure the user is aware of it. Different transition patterns may be used for different types of page change, and may be placed into the page-effects categories from the classification above. These may be used for either automatic or user-initiated updates. Table 1 shows how all relevant patterns fit into this taxonomy.

A second distinct group of patterns contains the four *invitation* patterns, which indicate to the user that a particular item on a page may be interacted with. These include *Cursor invitation*, where the mouse cursor changes to indicate interactivity, and *Tool-tip invitation*, where a tool-tip appears when an area is hovered over, and explains how it may be interacted with — these patterns may be applied to controls for any user-initiated update. The *Drop invitation* indicates suitable targets for drag and drop operations, and thus is applicable to user-initiated rearrange updates. The *Hover invitation* is similar to the tool-tip, but typically gives more information, and that information is added directly onto the web page. Thus it is implemented via an update that is user-initiated (hover) and may add-to or replace content.

The other patterns that use dynamic updates are as follows. *Auto complete* is user-initiated addition (for the first key press), or replacement (for subsequent key-presses); *Drag and drop modules* are user-initiated rearrangement; *Carousel* (a set of pictures from which the user must choose, not unlike a slide show) and *Module tabs* (as in the Yahoo! page in Figure 1) are both user-initiated replacement. In addition to these, which necessarily involve dynamic updates, the *Calendar picker*, *Rating an object*, *Writing a review*, and *Sign-in continuity* patterns may be implemented using dynamic updates.

The Yahoo! pattern library gives another view on dynamic updates — that of designers (rather than users or developers) who wish to communicate some information to the user. In many cases this is done by dynamically updating the page. This brief introduction to the patterns has shown how the classification system might work, although precise categorisation of many patterns will depend upon the actual implementation.

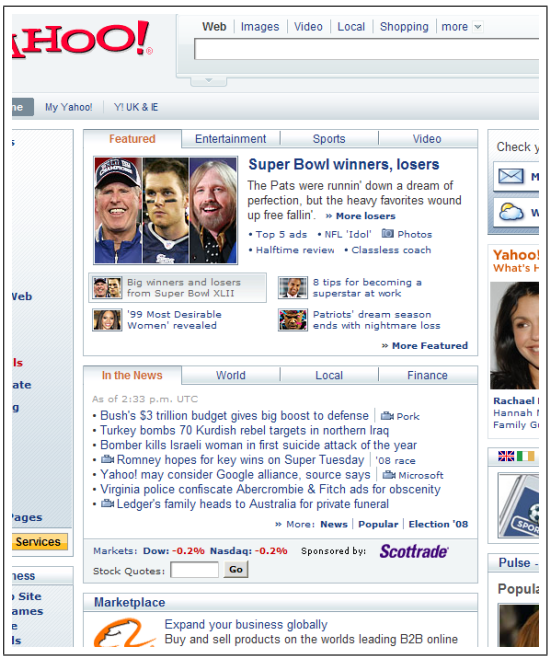

Fig. 1 A portion of the Yahoo home page, early 2008. Two areas of tabs are visible in the central column.

3 Technologies

Although this review is user-centered, developing assistive technology solutions for dynamically updating pages requires a technical understanding of how the updates may be achieved. This section gives a brief overview of the techniques that developers can use to update their pages dynamically.

Broadly speaking, dynamic page changes can be divided into three categories. First, it is possible to change only the style of the content on the page, for example to hide some content from view, while revealing other content; this gives the appearance of a dynamic update, without actually changing the page content. Second, it is possible to interact with an object that is embedded into the page. Examples include Flash objects and Java applets. Finally, it is possible to manipulate the page content directly, via the Document Object Model (DOM)[3]. The DOM is a way of representing documents as a hierarchy, and defines a standard way to manipulate and access HTML documents. It provides a platform and language neutral interface that allows pro-

[3] `http://www.w3.org/DOM/`

Table 1 Classification of Yahoo! patterns. Note that this classification is done according to the update, which the pattern may be loosely associated with (e.g., drop invitation, or which the pattern demands (e.g., auto complete). Patterns that are placed in more than one class, can be used in different ways, the particular class for an instance dependant upon its exact implementation.

	Page Effect				Initiation	
Pattern	Add	Remove	Replace	Rearrange	Automatic	User
Transitions						
Animate				✓	✓	✓
Collapse		✓			✓	✓
Cross-fade			✓		✓	✓
Expand	✓				✓	✓
Fade-in	✓				✓	✓
Fade-out		✓			✓	✓
Self-healing		✓			✓	✓
Slide	✓				✓	✓
Spotlight			✓		✓	✓
Invitations						
Cursor	✓	✓	✓	✓		✓
Tool-tip	✓	✓	✓	✓		✓
Drop				✓		✓
Hover	✓		✓			✓
Other						
Auto complete	✓		✓		✓	
Drag and drop				✓		✓
Carousel			✓			✓
Module tabs			✓			✓

grams or scripts to access and update the content and style of an HTML or XML document dynamically. Various techniques are described below. The first two describe style changes and the use of embedded objects, while the remainder are techniques that may be used to modify the DOM.

Style modification is a common method for giving the appearance of a dynamic update without client-server communication. In these cases, all content is downloaded with the original page, but scripting (typically JavaScript) is used to modify the appearance. The BBC Homepage[4] has several examples of this: there are plus and minus buttons for several of the sections that allow the user to display more or fewer stories. A fixed number of these are part of the page source; the buttons merely control how many are visible using Cascading Style Sheets (CSS) — viewing without CSS displays all eight stories. The functionality is provided by JavaScript (if disabled, a default of three stories is shown, and the buttons have no effect).

Embedded Objects such as Java Applets and Flash may change their content through communication with a server. With applets, this can happen directly through a Common Gateway Interface (CGI) program on the server through Hypertext Transfer Protocol (HTTP), or indeed any protocol. Alternatively, an applet may connect to a Java HTTP Servlet running on the server. Finally, Applets may communicate using the Java techniques of Remote Method Invocation (RMI), Java Database Connectivity (JDBC), or Common Object Request Broker Architecture (CORBA). Adobe's Flash allows relatively sophisticated user interaction, and can also exchange data with the server.

XMLHttpRequest [5] is an application programming interface (API) that can be used by scripting languages (e.g., JavaScript[6]) to transfer XML (extensible markup language) to and from a web server over HTTP. The connection is independent from the connection by which the page was originally loaded, thus allowing data transfer to be asynchronous. The name is misleading: AJAX supports any text-based format of data, not just XML.

AJAX (Asynchronous JavaScript And XML) is a Web development technique based on XMLHttpRequest and JavaScript, allowing asynchronous updates. It uses a client-side scripting language (typically JavaScript) to get new data from the server (via XMLHttpRequest or IFrames) and to interact with the DOM of the page. An example of AJAX can be used to demonstrate the technical aspects of its functioning. Introduced above, Google Suggest is a version of the Google search page that uses AJAX to display search suggestions when a user types in the search box. The suggestions are retrieved dynamically, and update with each keystroke. The essence

[4] `http://bbc.co.uk/`

[5] `http://www.w3.org/TR/XMLHttpRequest/`

[6] `http://www.ecmascript.org/`

of the code is that an XMLHttpRequest object is created, then, on each keystroke, used to pass the contents of the input box to the server. The plain text response is processed by JavaScript to generate an HTML table, which is inserted into the Document Object Model of the page. To the user, this table appears just below the search box.

DOM Load and Save is the W3C recommendation[7] for updating a page without a reload. This is a platform and language neutral interface, and may be implemented in any language. A parser object is created, then passed a URI, returning a document that can be inserted into a page using JavaScript. It is not clear how well supported this technique is by browsers (the above example works in Opera 10.00, but not Internet Explorer 7 or Firefox 3.5.3), nor how well-known this is by developers.

Inline Frames (IFrames) allow a web page to contain another page, and are primarily used for security reasons. The contents of the <iframe> are defined by its src attribute, and may be changed by a script, or by the user activating a link. In the latter case, the link must specify its target as the id of the iframe; clicking the link will then replace the iframe content with whatever the link pointed to.

It is also possible to use a hidden frame to communicate with a server. In this case, the iframe is made invisible (by giving it zero height and width) and made to load a page from the server containing a script (e.g., when a link is clicked). This script passes data to another script on the host page; this then displays the data somehow (JavaScript alert, modification of host DOM, etc). Extra functionality may be gained by passing data to the server page, either from a form, or encoded in its URL.

HTML Objects were introduced in HTML 4, and allow cascading. For example, if the object is a movie, there may be another object embedded within that such as an image, and within that there may be some descriptive text. Then, if the browser can display movies, it does so, otherwise it displays the image; if that is not possible only the text is shown. An object need not be multimedia, but might just be HTML, in which case it can behave in a similar (but not identical[8]) fashion to iframes. As with iframes, the contents of an object may be changed by user action, or automatically.

Plugin scripting allows non-HTML objects on a page to modify the page DOM (in addition to their own content, as above). LiveConnect[9] is an API to allow Java and JavaScript to call each other's methods. LiveConnect appears to have been superseded in Mozilla by an extension to the Netscape Plugin API (plugins include PDF readers, Java applets, Flash players) known as NPRuntime[10]. This allows conforming plugins to access browser objects such as the window object or the DOM element that loaded the plugin, and thereby influence the DOM of the surrounding page. Similarly, plugin methods may be called from a script on the page. If a plugin is able independently to communicate with a server, this technology allows it not just to change itself, but also to dynamically manipulate the rest of the page.

Image Wrappers are an old technique that is much more complicated than the methods above, and appears to be rarely used. It is based on the ability of JavaScript to modify the source of an image. The image used is often a blank single pixel image, so as to remain hidden.

4 Evolution of the Web

In order to deal effectively with accessibility problems, it is necessary to understand both the *nature* and the *scale* of the problems. The *nature* of this problem — how technologies currently used by disabled users deal with dynamic updates — is covered in section 5; this section seeks to determine the *scale* of the problem: are dynamically updating pages common, and how is the use of this technology changing? It reports on an examination of technology deployment in both historical and current Web pages.

Experiments were performed to analyse current and historic Web pages, to determine the extent of use of these technologies 'in the wild', and the trends in their uptake. A Web robot was used to download the relevant files for the 20 most popular sites, and a random selection of 500 pages, from the last 10 years. In addition, a snapshot was taken of the current situation using the top 500 sites and a random selection of 5000 pages.

4.1 Method

Relevant files (HTML, CSS, scripts) were downloaded for local analysis using a Web robot. Current pages were downloaded directly, older pages were retrieved from

[7] http://www.w3.org/TR/DOM-Level-3-LS/load-save.html

[8] See http://www.w3.org/TR/html401/struct/objects.html#embedded-documents

[9] See http://developer.mozilla.org/en/docs/LiveConnect

[10] http://developer.mozilla.org/en/docs/Gecko_Plugin_API_Reference:Scripting_plugins

the Internet Archive[11]. Four categories of sites were examined, two of which gave information over the last 10 years, and two of which gave a snapshot of the current situation. The selections were designed to compare the most popular sites with a random selection from the Web as a whole. Popularity was determined using the Alexa rankings[12]; the top 20 pages from June 2008 were analysed to gain historical data, while for the snapshot of the current situation the most popular 500 pages were used. A random selection of sites (chosen using the Google Directory[13]) were used to determine the state of the Web as a whole, with 500 pages being analysed each year for historical data, and 5000 for the current situation. For the historical data, the same sites were used over the entire period (i.e., the data does not necessarily represent the most popular sites for a given year). Since these pages were not always archived in a given year, there are some missing data; values are therefore presented as a percentage of the available pages.

For the snapshot data, searches were performed on the page source ((X)HTML) and all associated JavaScript files. For historical data, it was not always the case that JavaScript files were archived (the Internet archive selects URLs by popularity), so it is possible that searches may return fewer results than was actually the case. A further limitation is if a single JavaScript file is required by several pages, it might contain functions that are not used by the particular page being examined but will nevertheless be searched, and might return a false-positive result. More details about the methods used can be found in Chen and Harper (2008).

4.2 Results

JavaScript is, and has been for at least 10 years, very commonly used by Web designers. The analysis reveals that it has been used in at least 90% of the top 20 sites and more than 80% of the random 500 pages for most of the last 10 years, although the data hint at a decline in usage. The data are shown in Figure 2. Flash is becoming increasingly common. Currently 9 (45%) of the Alexa top 20 sites and 20% of the random 5000 sites use Flash; usage is showing a steady increase (Figure 3). Note that Flash is often used for animations and videos, and it is likely that many will not give similar functionality to the dynamic updates under consideration here.

AJAX techniques are also increasing in frequency: XMLHttpRequest is found in around 30% of most popular pages, and in around 7% of the random selection of 5000 pages (Figure 4). In contrast, a search for "createLSParser" found no matches over any of the page selections; this suggests that the W3C recommended DOM 3 Load and Save method is not in common use; unsurprising given the lack of browser support.

Considering the trends, it is apparent that usage of these techniques is increasing. Looking solely at XMLHttpRequest, since mid-2004, when this was used in almost no sites, usage has risen to around 30% of the top sites, and 7% of a random selection. Analysis is not possible on the top 20 data due to the very low numbers, but when applied to the random 500 data, linear regression shows that uptake is approximately 2.5% per year ($R^2 = 0.92$).

5 Assistive Technologies

In addition to exploring the technologies used by Web developers, and examining how their use is evolving, an understanding of the requirements for assistive technologies in this area will also demand an understanding of the capabilities of the technologies currently used by visually impaired users. This section aims to present the current state of assistive technologies, and is divided broadly into two: first are the results of an investigation into which technologies are used by visually impaired users in the United Kingdom; second is experimental and anecdotal evidence about the capabilities of these technologies when used by experienced users.

5.1 Technology Choices

This section examines the use of Web browsers and assistive technologies within the visually impaired community, in particular identifying which products are used by them. This was achieved by analysing the mailing list of an online community, composed largely of blind and visually impaired computer users. The British Computer Association of the Blind (BCAB) is a UK based "organisation of visually impaired people who use Information and Communications Technology". According to their website[14], its members "range from experienced computer professionals to people who are beginning to explore the use of Information and Communications Technology for leisure, study or employment". The BCAB has a mailing list "to provide a lively and informative forum for discussing Information Communication Technologies (ICT) for visually impaired people", which is reasonably active, with a mean of 623

[11] http://www.archive.org/
[12] http://www.alexa.com/
[13] http://directory.google.com/
[14] http://www.bcab.org.uk/

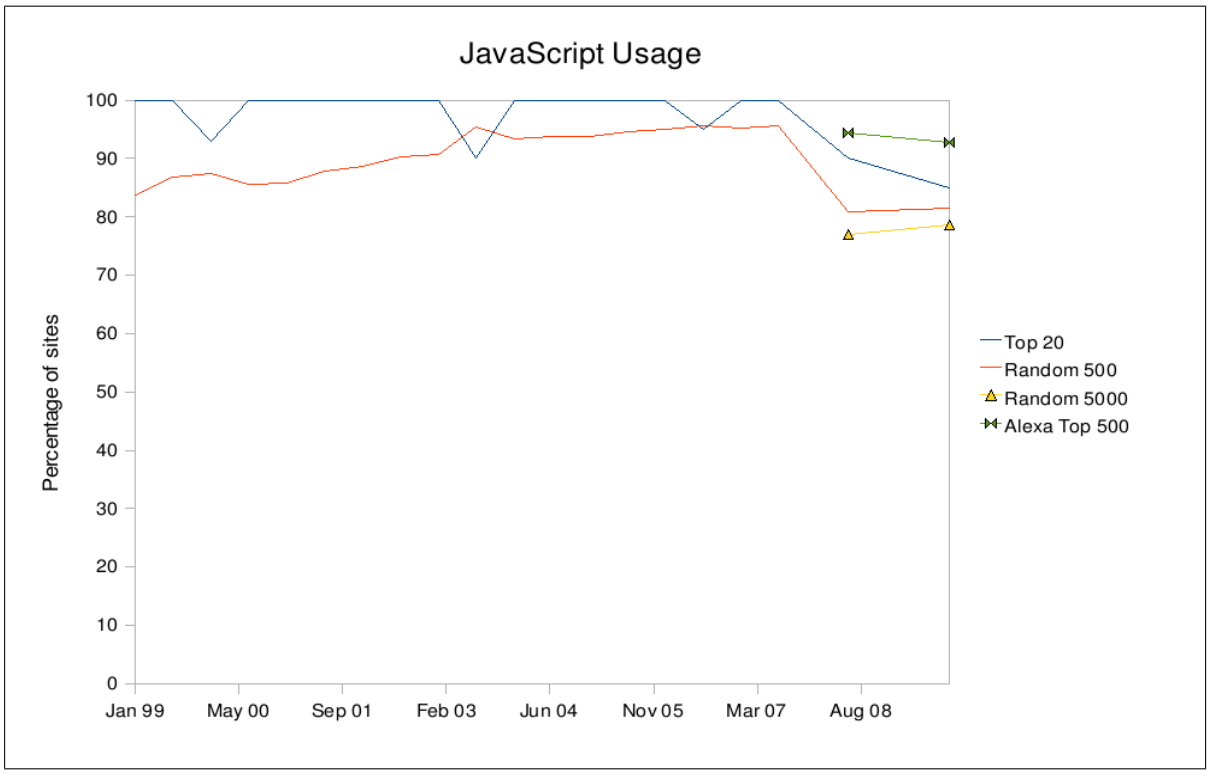

Fig. 2 JavaScript usage 1998 to 2008.

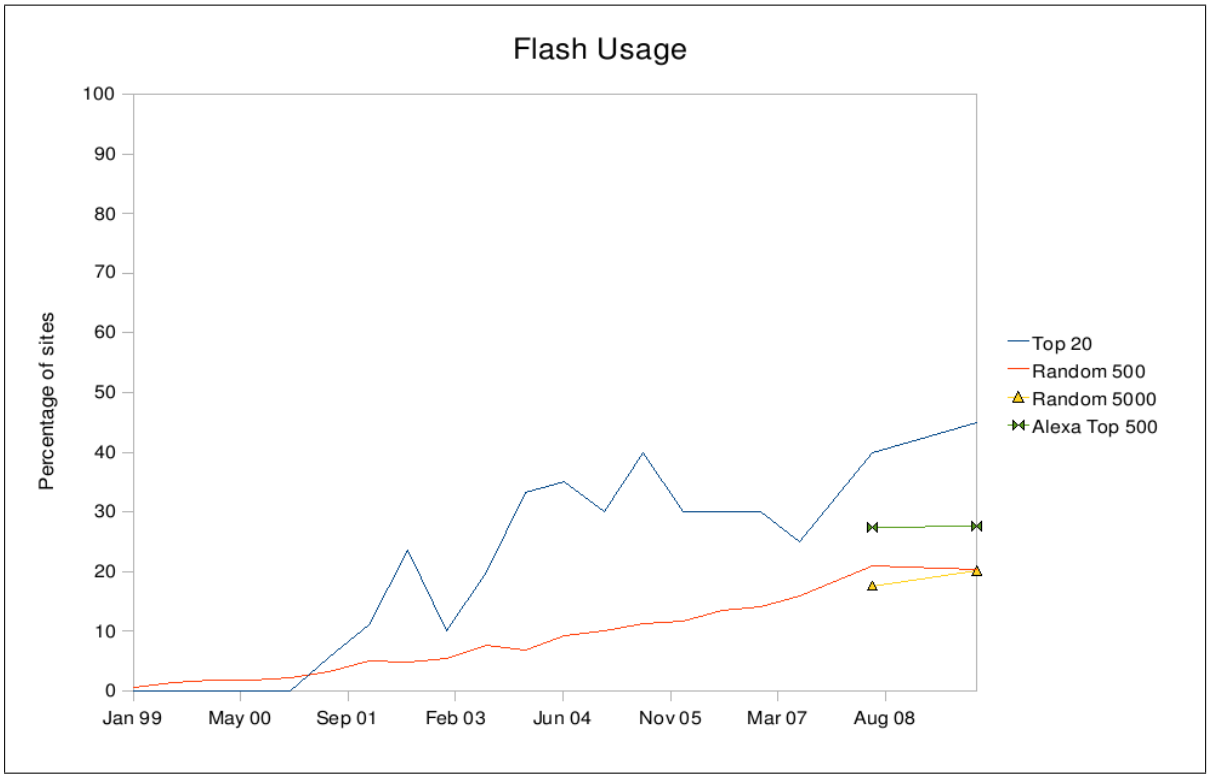

Fig. 3 Flash usage 1998 to 2008.

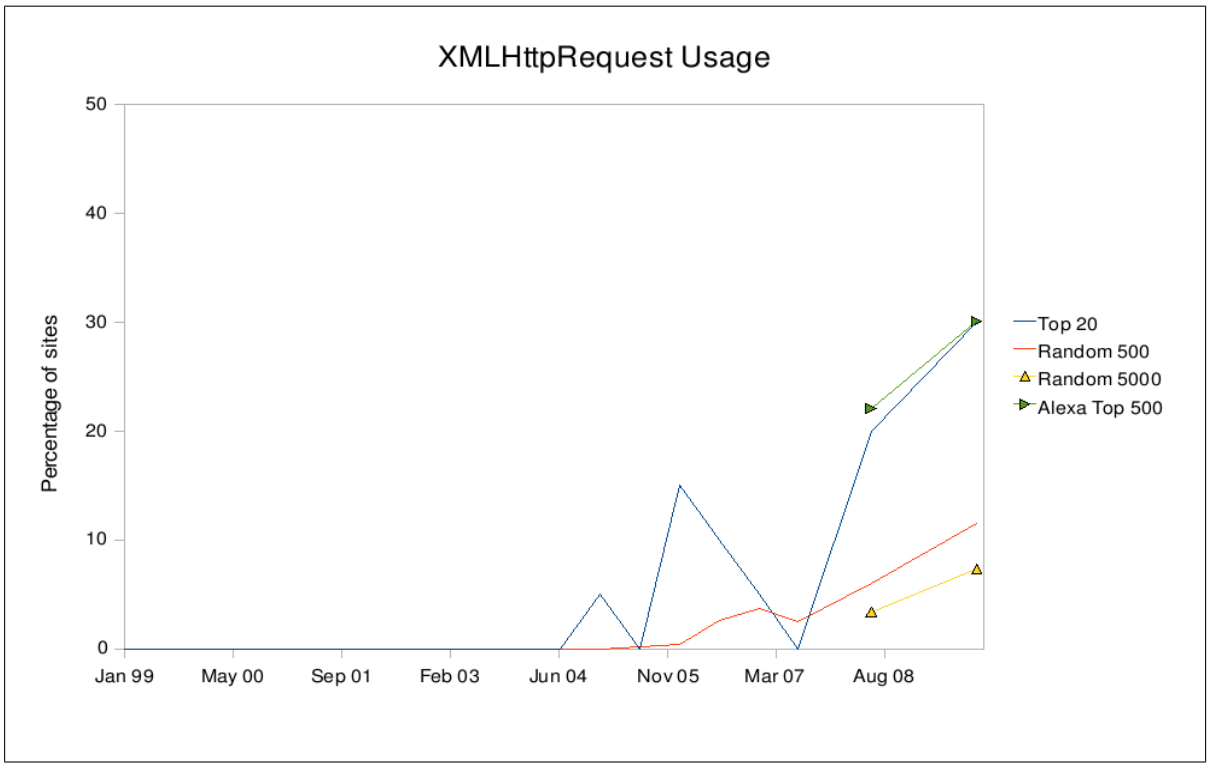

Fig. 4 XMLHttpRequest usage 1998 to 2008. Note that the y-axis only shows to 50%.

posts per month for the year to the end of August 2009. The mailing list archives from June 2006 are publicly available[15].

In order to assess the technologies actually used by blind and partially sighted computer users, a message was posted to the mailing list asking members if they could let us know which assistive technologies and which web browsers they used. Unfortunately no replies were forthcoming, so the information was obtained through an analysis of the archives. Since many of the messages on the list are of the type: 'I'm using screen reader Y on operating system Z, and I'm having difficulty doing X — can anyone help?', this proved a worthwhile technique. It might be argued that the users of the BCAB mailing list may not be representative of visually impaired users more generally, and particularly that they might be more experienced computer users than average. We believe, however, that these users represent precisely the type of people who are likely to be wanting and needing to interact with Web 2.0 technologies.

Analysis of the mailing list was performed by creating a local copy of the entire archive; this allowed more rapid searching to be performed. The first stage of analysis was to identify all unique users of the mailing list. Although this could not be done with complete certainty (since some users might have posted under different user names), there appeared to be some 257 individuals who had posted to the mailing list since June 2006. Searches were then performed on the messages sent by each of these individuals, to determine the type of screen reader and Web browser they were using.

[15] `http://www.freelists.org/archives/bcab/`

5.1.1 Assistive Technologies

There are a variety of assistive technologies available, from relatively simple screen magnification tools, to screen readers and braille output devices; this report concentrates on screen readers. A selection of English language screen readers capable of working with a Web browser is given below.

- Emacspeak, by T.V. Ramen.
- HAL, from Dolphin Computer Access.
- Supernova — HAL with screen magnification and Braille output.
- JAWS for Windows, from Freedom Scientific.
- Microsoft Narrator (bundled with recent versions of Windows).
- NVDA. A free, open source, Windows screen reader.
- Orca for the Gnome desktop, principally on Linux and Solaris.

- System Access from Serotek; a free online version, SA To Go, is also available.
- Thunder from Sensory Software.
- VoiceOver, from Apple (bundled with Mac OS X).
- Window-Eyes, from GW Micro.
- ZoomText, from Ai Squared. Includes a screen magnifier.

In addition to general-purpose screen readers, some more limited systems are available that allow Web browsing. These include dedicated self-voicing Web browsers, such as the Fire Vox extension to the Firefox browser, and WebAnywhere, from the University of Washington, Seattle, and more general systems with some Web capability, such as Guide, a self-voicing set of applications including a browser.

As this shows, there are a large number of options for visually impaired users to choose from. Analysis of the BCAB mailing list archives revealed which screen readers were used by 130 users in the first period, and 116 in the second. These are given below. It should be noted, however, that these figures are gathered from statements made over the last 2 years, and thus may not represent exactly what is being used now. Also, it was clear that many members had experience of more than one screen reader, indeed some used more than one on a regular basis, and it is not necessarily the case that all the screen readers being used were mentioned. Furthermore, there is a possibility that the clear popularity of JAWS among members of the BCAB list might deter people from asking about other screen readers, thereby amplifying the popularity of JAWS. Nevertheless, the results are consistent with our expectations, based on our experience with organisations helping visually impaired users learn IT skills (Henshaws and Access Summit).

The first analysis was performed on the data for the period between June 2006 and June 2008. Of the 130 users for whom information was available, some 87 used JAWS for Windows. An additional 16 users posted messages on the list suggesting that they either used JAWS, or had experience with it; this total suggests that 79% of members were JAWS users. 10 used HAL, and a further 3 Supernova, while 9 used Window Eyes. 13 people used other screen readers, including Zoomtext, Thunder, NVDA, Guide, Voiceover, and Orca. These last two, for Mac OS and Linux respectively, are notable exceptions to the predominance of the Windows operating system. It should be emphasised again that these numbers are not exclusive: several members of the list used more than one screen reader. When the analysis was repeated on the messages posted between June 2008 and June 2009, the results were broadly similar. Of the 111 users for whom it was possible to identify at least one screen reader with which they were familiar, 87 used JAWS (78%), 12 Window Eyes, 3 used HAL and 6 used Supernova. Zoomtext was used by 2 users. NVDA and Orca appeared to have been the subject of greater discussion, and to have increased in popularity, with 5 NVDA users and 6 users of Orca (including 4 using the Vinux[16], a 'customized version of Ubuntu for visually impaired users' developed by Tony Sales of the RNIB — a user of the BCAB mailing list), although with such small numbers it is not possible to identify trends. There is no denying, however, the interest in low-cost or free alternatives to the often expensive commercial offerings. These data are summarised in Figure 5.

There were also discussions about System Access, in particular SA To Go, a web service that provides screen reader access to various applications, including web browsing. In particular it claims to make 'Web 2.0 accessible to the blind and visually impaired'[17]. About 6 or 7 users appeared to use, or have used, this product. In a similar vein, there was interest in (although little to no evidence for use of) the WebAnywhere accessible browser from the University of Washington (Bigham and Prince, 2007).

As a final note, it is possible to use the data to identify how up-to-date people are with their screen readers. Figure 6 shows the proportions of people using the different versions of the most popular screen reader (JAWS for Windows) over three periods. This shows that users do upgrade over time, but that there is a significant number not using the most recent version. This is in broad agreement with a survey (Web Accessibility in Mind, 2009a) that showed that 16.4% of users had not updated their screen reader in the last year.

5.1.2 Browsers

For the general population, the market share of browsers, as measured by Net Applications[18], is given in Table 2. Analysis of the BCAB mailing list archives indicates that the situation is not radically different for blind users. Although information on browsers for the period up to June 2008 could only be obtained for 27 users of the mailing list, some 24 of these used Internet Explorer (IE), 6 used Firefox, and 1 used Lynx. Note that some users claimed to use both Internet Explorer and Firefox; this mirrors the situation with screen readers, where multiple tools were used. The reasons can only be speculated upon, but may include one browser not being

[16] `http://vinux.org.uk/`

[17] `http://www.serotek.com/cas.html`

[18] From `http://marketshare.hitslink.com/report.aspx?qprid=0`, June 2008 and July 2009

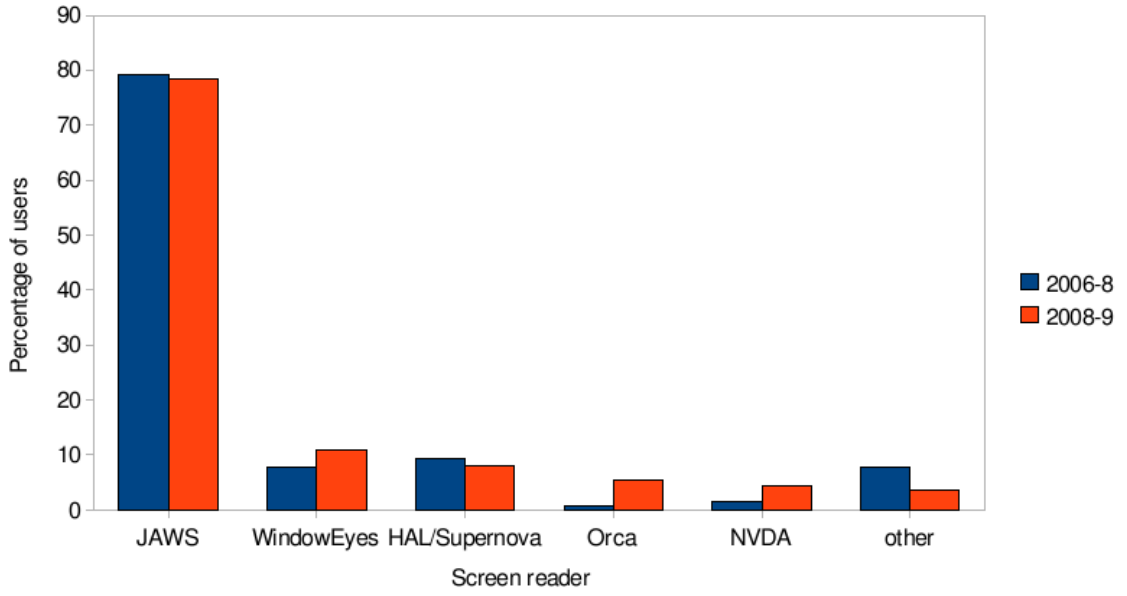

Fig. 5 The popularity of different screen readers among posters on the BCAB mailing list. HAL and Supernova are grouped, since these differ only by the presence of a screen magnification tool.

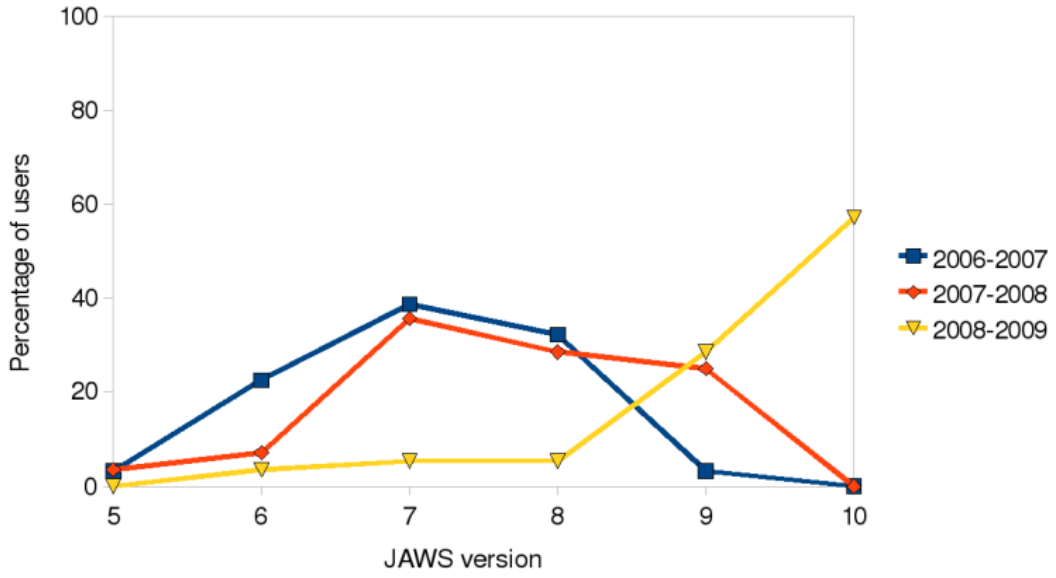

Fig. 6 The changing distribution of JAWS version use: the proportion of people using different versions over three periods (approximately June to June).

available on all computers used, or preference for one browser over another when using a certain site. If the the assumption is made that those using both browsers spend equal time on each, these figures give approximately 81% to Internet Explorer and 15% to Firefox; these are of the same order as for the figures for all users. For the period of July 2008 to July 2009, information was obtained for 28 users, 20 of whom used Internet Explorer and 11 Firefox. Thus the market share amongst BCAB users is approximately 72% to Internet Explorer, and 28% for Firefox. While these figures are based on information from a very small group, they are consistent with the figures for the wider population, and reflect the increasing use of Firefox.

Table 2 Market share of Web browsers for June 2008 and July 2009. The data for all users is from Net Applications, while the data for BCAB members represents a small group of users of the mailing lists.

	Browser	Market share (%)	
		June 08	July 09
All users	Internet Explorer	75.4	67.7
	Firefox	18.9	22.5
	Safari	2.8	4.1
	Chrome	—	2.6
	Opera	2.1	2.0
	Netscape	0.5	0.7
	Other	0.2	0.5
BCAB	Internet Explorer	81	72
	Firefox	15	28

In terms of how up-to-date users are with their browser versions, only a little information is available, and that only for Internet Explorer (IE), but this suggests that the typical user is using the version before the latest. For those users for whom the version can be deduced, in 2008, 6 users were using IE7 and 13 users version IE6; in 2009, 6 users were using version IE8 (released March 2009), 10 version IE7, and 2 version IE6. While these figures are of little significance (e.g., while the version reported reflects the latest version mentioned by the user, this may still be several months out of date), they fit with observations made when browsing the lists. Here, discussions suggested that the biggest factor influencing choice was compatibility with the user's assistive technology; typically only the latest screen reader versions support the most up-to-date browsers.

The messages on the mailing list suggested little use of other browsers. For example, no users claim to use Opera on a regular basis. Similarly, there is little mention of the Apple browser, Safari, perhaps reflecting the apparently overwhelming predominance of Windows as the operating system of choice.

5.1.3 Specialist systems

While the majority appear to browse the Web using a standard browser in conjunction with a screen reader, this is not necessarily the case for all. Some specialist programs are available, including WebbIE, Guide and Fire Vox. These are introduced here for completeness, but few BCAB mailing list users appeared to use any of them.

WebbIE is "a web browser for blind and visually-impaired people, especially those using screen readers"[19]. It also enables other Web activities such as listening to Internet radio, downloading podcasts and reading RSS feeds. WebbIE is an Internet Explorer-based browser which is stand-alone, but is required for web browsing with the Thunder screen reader (and comes installed with it). For this reason, and taking into account the results of the survey of screen readers, we can conclude that it is used by at least 2 or 3 members of the BCAB mailing list.

Guide [20] Is essentially a self-voicing set of applications, including email, document editing and Web access. It is also possible to change the visual characteristics of the display, e.g., to increase contrast or size. As noted above, this was used by very few — evidence could only be found for one person using this software.

Fire Vox [21] is an extension to the Firefox Web browser. It gives screen reader-like access to this browser, and is developed for Web 2.0, supporting ARIA (Accessible Rich Internet Applications (Gibson, 2007)) markup. It did not, however, appear to be used by members of the BCAB list, and, although there were discussions about it, these were relatively negative (e.g., 'what's the point?'[22]).

5.2 Data Validation

The use of passive techniques to mine information about the choices made by visually impaired users appears to be novel. The standard technique is to ask direct questions, for example through a survey. The most comprehensive and relevant of these is the WebAIM screen reader survey, which has been carried out twice, over the period December 2008 to January 2009 (Web Accessibility in Mind, 2009b), and again in October 2009 (Web Accessibility in Mind, 2009a).

While these can reach a wide audience (particularly if available online), there may also be problems. The most significant is sampling, as it can be difficult to get proportionate representation from all communities. For example, it was noted in the results of the first WebAIM screen reader survey that an unusually high proportion of respondents used the Safari Web browser; the authors speculated that:

> 'this may be due in part to the fact that some in the Mac community actively solicited survey participation and encouraged respondents to indicate their Safari use, perhaps partially due to feeling snubbed because we didn't list them with IE and Firefox as direct choices.'

[19] `http://www.webbie.org.uk/`

[20] `http://www.softwareexpress.co.uk/read_more_about_guide.asp?P=GBP`

[21] `http://www.firevox.clcworld.net/about.html`

[22] For example, see the thread 'Firefox accessibility' at `http://www.freelists.org/archives/bcab/04-2007/threads.html#00036`

The approach taken here does not suffer this type of problem — by mining the mailing list post-hoc, deliberate over-representation of a community is unlikely. That is not to say that sampling questions do not arise, however. First, the sample is only ever going to represent the type of users that post on the mailing list. Second, the questions people ask are likely to be influenced by previous content; users with a question about their screen reader may be less likely to ask about it if they perceive the majority of users are not familiar with it.

To validate, to a certain extent, the data collected, it can be compared to the results of the WebAIM survey. A comparison of the results of the second WebAIM survey (October 2009) with the second analysis period for the BCAB lists (July 2008 to July 2009) is given in table 3. Note that the screen reader figures from the WebAIM survey are for those chosen as respondents' primary screen reader. While there are differences, notably the much higher use of Safari and Voiceover from the WebAIM survey, the figures are in broad agreement.

Table 3 Comparison of WebAIM survey results with analysis of the BCAB mailing lists. The percentage of people using different Web browsers and screen readers.

Technology	% of users	
	WebAIM	BCAB
Internet Explorer	71	72
Firefox	28	23
Safari	8	0
JAWS	66	78
WindowEyes	10	11
NVDA	3	5
Hal/Supernova	1	8
Voiceover	9	0

5.3 Current Technologies and Dynamic Updates

In order to determine what users experience when viewing and interacting with dynamic Web content with current technology, volunteers were asked to view a set of test pages. These pages contained content that updated in different ways, according to the taxonomy (Section 2). Participants were asked to view them using their normal software (Web browser and assistive technology) and report what they found. In support of this, some anecdotal evidence is also given, in the form of quotes from mailing list discussions about people's experiences with dynamic Web sites.

5.3.1 Method

Ten Web pages were written in HTML (HTML 4.0 Transitional). Eight of these contained dynamically updating content (using JavaScript, XMLHttpRequest, and, in some cases, PHP), one gave an introduction to the experiment, and a final page indicated the end of the experiment. WAI-ARIA markup was not used. The pages were kept as simple as possible, generally comprising a heading, a few paragraphs of text, and a link to the next test page. The first four test pages also had a link that initiated the update. The final page asked for comments on, and gave links to, external sites with dynamic content, and gave email addresses for participants to send their experiences to. Note that the content removal and rearrangement pages (pages 3,4,7 and 8) did not use XMLHttpRequest — these pages used plain JavaScript to modify the DOM.

Each of the eight test pages updated according to one class in the taxonomy of updates. That is four pages updated automatically, while the other four updated as a result of the user following a link. Each set of four pages had one page where content was added, one where content was removed, one where it was replaced, and one where it was rearranged[23]. In order to make them interesting and, to a certain extent, realistic, content was taken from the Wikipedia, with pages selected using the random page generator.

Participants were asked to give feedback about what happened when viewing the pages, and encouraged to give as much detail as possible. The instructions also gave some indication as to what the users should expect, and the type of information desired from them:

> "The first four pages update when a link is followed. In this case, please read the page to the end so you are familiar with its content, then go back to follow the link (these pages contain only two links, one to initiate the update, the other to move to the next page in the test). Note what happens directly after the link is clicked: for example, the page is read from the beginning and new text is found; new text is read straight away; nothing happens. If nothing happens, are you able to detect any changes to the page?
> The second four pages should update automatically within 10 seconds of loading (and possibly again after that) - please read the page and note what happens. Does anything appear to change after approximately 10 seconds?"

[23] The pages used in the experiment can be viewed online at `http://hcw.cs.manchester.ac.uk/research/saswat/experiments/atreview.html`

Since this experiment was purely investigatory, there were no hypotheses to be tested. After viewing the eight test pages, participants were asked to interact with three external pages: Google Suggest[24], iGoogle[25], and Yahoo![26]. The following instructions were given:

> "The previous 8 pages have used Web 2.0 technology to modify the content. The content has been made deliberately simple, to make it easier to determine if the changes are detected. It would also be useful if we could have feedback on how assistive technologies work with the same technologies on 'real' sites. Feedback on the following sites would be appreciated:
> Google Suggest: We are particularly interested in what happens when text is entered into the search box
> iGoogle: Are you able to access any preview information about the CNN.com stories. Would you find this useful?
> Yahoo: Can you read the information in the 'World' news section of this page?"

Participants were recruited by posting a request for volunteers on the BCAB mailing list, the dolphin users Yahoo! group, and through the Mozilla accessibility developers group. 13 participants responded, having tested 16 systems.

5.3.2 Results

Although the participants who completed the experiment only gave feedback on a selection of the possible combinations of browser and screen reader, it is possible to discern some trends, and to gain an impression about the current state of technology. In general, it was noticed that older versions of the screen readers (which were more likely to be used with older browsers) coped relatively poorly with dynamic information. Newer versions have changed, first by actually making all the updated information available, then by making changes more obvious to the user. Table 4 summarises the results.

Clearly the most important attribute for the combination is that the user can actually get to the updated information. From this experiment, it would appear that all combinations tested (apart from the oldest version of JAWS — version 6) are capable of coping, to this extent, with dynamic updates. The fact that the new information is available, if the user searches for it, suggests that the 'model' the user interacts with is refreshed to cope with new information (although in some cases this needed to be done manually).

Although participants in this experiment could identify the changes in many cases, it was rare that they were given any information (implicit or explicit) about changes. It will be remembered that they were aware not only of the fact that the test pages were dynamic, but also how the update was triggered: it is likely that without this information many of them would not have been aware that the pages had changed.

The functionality of the screen readers, in terms of notifying users about updates, is clearly improving. The latest Linux system — Orca with Firefox 3 — offers some implicit notification, in that the new information was spoken immediately. Content rearrangement was not announced, and could only be inferred, and removal of content seemed to cause some confusion. When page content was replaced, however, the user was able to understand the changes and access the new information easily. Version 7 of Supernova / HAL has some implicit notification for updates triggered by clicking a link, again by taking the user to the new information. Unfortunately no users of the latest version (version 9) participated in the experiment, but listed in the new features for this version is "New DOM support"[27]. Window-Eyes only seems to notify by speaking 'downloading page', and when used with IE 7, it would appear that it is necessary to refresh the buffer to even read the updated information. Using Firefox 3 and the Webvisum add-on seemed to remove the need to do this. Those versions of JAWS tested, including JAWS 10, appear to offer no notification. JAWS 11 has significantly enhanced features with respect to dynamic content, including support for ARIA. Unfortunately it has not been possible to test how it handles content which has not been enhanced with ARIA markup.

5.4 Discussion

These evaluations have highlighted the difference between making information technically accessible, meaning that the information is there if the user searches for it, and making it accessible in a way that allows the user to interact with it in an efficient manner. While recent technology manages the former (older technology does not even appear to provide the most basic level of accessibility), it certainly does not manage the latter.

[24] `http://www.google.com/webhp?complete=1&hl=en`
[25] `http://www.google.com/ig`
[26] `http://www.yahoo.com/`
[27] `http://www.dolphinuk.co.uk/productdetail.asp?id=5&z=3`

Table 4 Summary of screen reader / browser performance.

Screen Reader	Browser	Results
JAWS 6	IE 6	Only link-initiated updates seen. No notification.
JAWS 9	IE 6	Updates all seen. No notification.
	IE 7	Updates all seen. No notification.
	FF 3	Updates all seen. No notification.
JAWS 10	IE 6	Updates all seen. No notification.
	FF 3	"More flaky, not properly supported"
Orca 2.23.3	FF 3	Updates all seen. New content spoken immediately.
Supernova 6.53	IE 6	Updates all seen, but user needed to return to start to see new content.
Supernova 7	IE 7	Updates all seen. User taken to new content for link-initiated updates.
NVDA	FF	Updates all seen. No notification.
Window-Eyes 6.1	IE 6	Updates available after refreshing page buffer.
Window-Eyes 7	IE 7	Updates all seen after reloading browse mode. "Downloading page" spoken during AJAX update.
	FF 3	Updates all seen.

While it is possible to get to the information provided by updates, comments from visually impaired users demonstrate that this is merely bare necessity, and is not sufficient for true accessibility. What is also important is that users are aware that an update has occurred: considering the ability of the visual system to detect changes, lack of notification to screen reader users would mean they must interact with these sites in a significantly different way. Two quotes from visually impaired users illustrate the problem. The first is from a participant:

> "As I have said before. The most crucial thing is that nobody tells us anything has occurred. Its going to be a hard job to actually second guess which lines are updating if they are doing so like an ongoing cricket score. I mean simple scrolling messaging programs do my head in!"

The second is from one of the BCAB members; it is taken from a discussion about problems finding contacts on the Friends Reunited Web site[28]:

> "Actually, the Friends Reunited website is an interesting case study for us on this list at the moment as it is now highly dynamic. In fact, if you work out how to use it, it is perfectly accessible, but Jaws, as usual, is rather rude and doesn't tell you when things have been updated. You just have to presume they have been as if you're telepathic, and go and find the updates. But you can get to absolutely everything if you're prepared to use the hover keystroke and the links list."

It was noted above, however, that users are often left unaware that any change has taken place, and must rely on contextual clues around the link, memory, and experience to deduce that the page has altered. As noted above, the ability of the participants to notice and describe changes will probably have been improved by their expectations — they knew in advance that the pages would be changing.

This may be contrasted with the situation for visual browsers. The visual system is very efficient at detecting relevant changes: Jingling and Yeh (2007) noted how, if relevant, new objects appearing abruptly in the visual scene were attended quickly. Assuming, therefore, that a change occurs on a visible part of the page, is pertinent to the user's current task, and is a sufficiently large change, it might be expected that sighted users become aware that an update has occurred almost instantly. Indeed, this is supported by the eye-tracking studies reported by Jay and Brown (2008). This study used eye-tracking to monitor users attention while they performed tasks using dynamically updating Web pages, and concluded that users attended some types of update rapidly. The report offers some suggestions for factors which influence whether an update is attended to by the user: the main ones are how the update was initiated and how important it is to their task (these factors are, of course, related).

If users noticed a change (or expected one, as was the case in all of these pages), they were often able to deduce correctly how the page had changed. Certainly, new information was generally recognised as such, and replacement of chunks of text was also noticed. The pages used in this experiment were designed to be simple and, in the case of the user-initiated updates, the page changed in close proximity to the link that initiated it. It is not clear how the users would have fared (both in identifying that a change has occurred and identifying how the page had changed) if the effect was more distant from the cause — potentially a problem given the differences between visual layout and the or-

[28] `http://www.freelists.org/archives/bcab/05-2008/msg00355.html`

der of the source code and/or the order in which the screen reader presents the page. The external memory afforded by a visual display is likely to make change detection significantly easier for sighted users.

Having examined how screen reader users are able to identify when a change has happened, and what has changed, the next consideration is how users can determine whether the information provided in the update is useful to the user. This is beyond the remit of this review of assistive technologies, but it should be noted that the ability to overview information is generally greatly restricted for non-visual users, compared to those able to make a visual glance.

Perhaps one of the more significant problems faced by users was when updates occurred too often to allow the content to be read. For example, participant 5 commented:

> "I agree that deletions were a problem, but so were focus issues on the pages that received time-based updates. From my point of view, the latter problem was the more serious, since it interfered with reading the text."

This is a general problem as it is not specific to any particular screen reader or browser, but results from the fundamental differences between visual and non-visual interaction with information. It is at least partly caused by the difficulties screen reader users have in glancing at new content (to assess whether or not it needs proper reading), as well as the actual reading speed. This is a known problem, and the potential for ARIA to deal with situations like this has been examined by (Thiessen and Chen, 2007). They found that ARIA had limited ability to make frequently changing pages (their example was a chat room) accessible, although further developments in ARIA and its application are improving this (Thiessen and Chen, 2009; Thiessen and Russell, 2009).

In conclusion, it can be seen that modern screen readers and browsers are evolving to allow non-visual access to the information on dynamically updating pages. Unfortunately, however, the ways in which visually impaired users must currently recognise, access and understand that information, do not allow the level of efficiency necessary to allow these users to significantly benefit from dynamic Web pages. Although these results present a snapshot of rapidly changing technologies, the data shown in Figure 6 shows that they will continue to be relevant to many users for some time.

6 Conclusions

The Web is evolving - that is certain. Perhaps the greatest challenge currently posed by Web 2.0 for users of assistive technologies arises from the technological advances that enable content to update dynamically. Traditionally, Web pages did not change once they had been served. In this context, they have many of the characteristics of a document. Despite the potential for multimedia, with sound and video components, the page acted as a page: it could be read (or watched, or listened to), and not (except in a very limited sense) interacted with. Web pages did not contain controls and did not change over time: they were static. The rise of scripting technologies, and particularly AJAX, has changed this. Now, pages are often dynamic and interactive — the whole model of a page has changed quite dramatically. Not only do readers have the challenge of navigating the Web, and understanding and relating different sections of a page, but now they may also need to understand whether the page has changed, which parts have changed, and whether the changes are of interest. As the review of assistive technologies (Section 5.3) showed, these pages are currently dealt with poorly, but with this technology used in a large and increasing number of sites (Section 4), action is urgently required if visually disabled users are to be able to use the Web.

The implicit understanding that (experienced) sighted users have of what an interface component is (i.e., their ability to recognise it as, for example, a slide show control) allows them to guess what its effect will be. At the same time, their ability to notice change in parafoveal vision, then either ignore it, glance at it, or consider it carefully, enables them to deal with an update appropriately, with minimum disruption to their primary task. Only by recreating these benefits through audio can screen reader users operate these controls, and their host pages, efficiently.

Given this, a significant part of the problem relates to understanding what causes an update, and what information it provides. Since the controls and interfaces used by designers are not natively supported in HTML, they are generated using combinations of simpler technologies, an approach which obscures the true nature of the component: its function only becomes explicit when seen visually. To make the function explicit to the user of an assistive technology, however, the function must be determined from the code. Solving the problem of making this content accessible requires coordinated work on three fronts: content, user agent, and assistive technology.

Currently, the focus on content improvement is through WAI-ARIA: the Accessible Rich Internet Applications

Suite (Gibson, 2007) from the W3C's Web Accessibility Initiative. The essence of the ARIA solution (see, for example, Buzzi and Leporini, 2009; Thiessen and Russell, 2009; Thiessen and Chen, 2009) is to address some of the accessibility barriers by adding semantic metadata to a page[29]. In addition to enabling keyboard accessibility and making the role and state of controls explicit, ARIA provides a mechanism for noting live regions — those parts of a page that may be updated. These can have further attributes to denote what types of update to announce (e.g., just additions), how important it is to announce the update, and how much of a region should be read when part is updated.

The role of user agents in this context is relatively straightforward: it is simply necessary to pass the content to the assistive technology. Full support for the DOM, including events, and DOM Mutation Events in particular, gives the assistive technology the information it needs: what has changed, and when. The final decisions: what to present, when, and how, need to be made by the AT. User agents should also support WAI-ARIA markup, and pass this information to the AT so that it may be used to support these decisions.

We are in a situation where update handling is poor, except for those sites using ARIA, when rendered on user agents and assistive technologies that also support it. Despite implementation of ARIA by many large corporations there are still many sites where it is not implemented, and this is likely to remain the case for some time. An additional complication is the lag between AT development and user-uptake (see Figure 6). There is even an argument that ATs should not use ARIA markup uncritically, but should be able to override it. For example, it is not difficult to envisage inappropriate use of ARIA to make advertisements more salient, and even well-intentioned developers may find it difficult to choose the appropriate settings, as many are unlikely to do the user testing that is really necessary (the rewards for creating an accessible site are not always obvious to budget holders (Richards and Hanson, 2004)). In other cases, it may just be very difficult to find settings that work, even with user testing. For example, Thiessen and Chen (2007) have explored the difficulties associated with using ARIA to provide an usable interface to a Web-based chat client for screen reader users. While developments have improved matters (Thiessen and Russell, 2009; Thiessen and Chen, 2009), using ARIA in sophisticated rich user interfaces is a complex and demanding task, requiring considerable time and money, and a good understanding of users. It is also likely that further development of ARIA is necessary, for example, it appears that the benefits of an auto-suggest list could be difficult to achieve using ARIA (Brown et al, 2009).

While adding accessibility-enhancing semantics at the authoring stage can provide useful information to assistive technologies, there is also a strong argument for more refinement in their default handling of updates. Many pages with dynamic content, indeed probably the vast majority, do not have ARIA markup. The review of dynamic content, and resulting taxonomy (section 2) shows that there is considerable variety, both between and within classes. Presenting these all to the user in the same way is not appropriate. Yet, on the whole, this is what screen-readers currently do: most give users no indication of updates, while the most sophisticated at the time of review announced updates triggered by clicking a link and not those that occurred automatically. While even this level of sophistication is welcome, it is doubtful that it is optimal. Dealing with updates on a case-by-case basis, according to its attributes and content is likely to be necessary. Only then can people using assistive technologies to browse the Web benefit from the benefits dynamic content brings to sighted users.

Based on these results we recommend that:

- Web developers make use of WAI-ARIA markup, or use libraries that include it.
- User-agent and assistive technology developers continue to improve support for ARIA.
- Assistive technologies detect and present even those updates that are not ARIA-enhanced. With access to a live DOM, detection is not difficult, and notification or presentation based on even a couple of simple rules could save users considerable confusion.
- A set of rules should be developed to understand how to present different types of update. These can inform both usage of ARIA by developers and its handling by ATs, and can show how to present updates not marked up with ARIA semantics.

Acknowledgements This work is part of the Single Structured Accessibility Stream for Web 2.0 Access Technologies (SASWAT) project and is funded by the UK Engineering and Physical Sciences Research Council (EP/E062954/1). As such the authors would like to thank them for their continued support.

References

Bigham JP, Prince CM (2007) Webanywhere: a screen reader on-the-go. In: Assets '07: Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility, ACM, New York, NY, USA, pp 225–226

[29] The WAI has a primer on ARIA: `http://www.w3.org/TR/wai-aria-primer/`

Brown A, Jay C, Harper S (2009) Audio representation of auto-suggest lists. In: W4A'09: Proceedings of the 2009 Cross-Disciplinary Conference on Web Accessibility (W4A), pp 58–61

Burks MR, Lauke PH, Thatcher J, Rutter R, Waddell C (2006) Web Accessibility: Web Standards and Regulatory Compliance. Friends Of Ed

Buzzi M, Leporini B (2009) Editing wikipedia content by screen reader: Easier interaction with the accessible rich internet applications suite. Disability & Rehabilitation: Assistive Technology 4(4):264–275

Chen AQ, Harper S (2008) Web evolution: Method and materials. Technical Report, University of Manchester, URL `http://hcw-eprints.cs.man.ac.uk/74/`

Gibson B (2007) Enabling an accessible Web 2.0. In: W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A), ACM, New York, NY, USA, pp 1–6

Gregor P, Sloan D, Newell AF (2005) Disability and technology: building barriers or creating opportunities? Advances in Computers 4:283–346

Jay C, Brown A (2008) User review document: Results of initial sighted user investigations. Technical Report, University of Manchester

Jingling L, Yeh SL (2007) New objects do not capture attention without a top-down setting: Evidence from an inattentional blindness task. Visual Cognition 15(6):661–684

O'Reilly T (2007) What is Web 2.0: Design patterns and business models for the next generation of software. Communications and Strategies 1:17–37

Ramen TV (2008) Specialized Browsers. In: Harper S, Yesilada Y (eds) Web Accessibility: A Foundation for Research, Springer-Verlag, chap 12, pp 195–213

Richards JT, Hanson VL (2004) Web accessibility: a broader view. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, ACM, New York, NY, USA, pp 72–79

Thiessen P, Chen C (2007) Ajax live regions: chat as a case example. In: W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A), ACM, New York, NY, USA, pp 7–14

Thiessen P, Chen C (2009) ARIA Live Regions: An introduction to channels. Journal of Access Services 6(1):215–230

Thiessen P, Russell E (2009) Wai-aria live regions and channels: Reefchat as a case example. Disability & Rehabilitation: Assistive Technology 4(4):276–287

Web Accessibility in Mind (2009a) Screen reader user survey results. `http://webaim.org/projects/screenreadersurvey2/`

Web Accessibility in Mind (2009b) Survey of preferences of screen readers users. `http://webaim.org/projects/screenreadersurvey/`

Zajicek M (2007) Web 2.0: hype or happiness? In: W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A), ACM, New York, NY, USA, pp 35–39