More than Meets the Eye: A Survey of Screen-Reader Browsing Strategies

Yevgen Borodin* Jeffrey P. Bigham[#] Glenn Dausch[†] I.V. Ramakrishnan*

* Stony Brook University, Comp. Sci. Department, Stony Brook, NY 11794, {borodin, ram}@cs.sunysb.edu
[#] University of Rochester, Dpt. of Comp. Sci. & Eng., Rochester, NY 14618, jbigham@cs.rochester.edu
[†] Stony Brook University, Disability Support Services, Stony Brook, NY 11794, gdausch@notes.cc.sunysb.edu

ABSTRACT

Browsing the Web with screen readers can be difficult and frustrating. Web pages often contain inaccessible content that is expressed only visually or that can be accessed only with the mouse. Screen-reader users must also contend with usability challenges encountered when the reading content is designed with built-in assumptions of how it will be accessed – generally by a sighted person on a standard display. Far from passive consumers of content who simply accept web content as accessible or not, many screen-reader users are adept at developing, discovering, and employing browsing strategies that help them overcome the accessibility and usability problems they encounter. In this paper, we overview the browsing strategies that we have observed screen-reader users employ when faced with challenges, ranging from unfamiliar web sites and complex web pages to dynamic and automatically-refreshing content. A better understanding of existing browsing strategies can inform the design of accessible websites, development of new tools that make experienced users more effective, and help overcome the initial learning curve for users who have not yet acquired effective browsing strategies.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia – *architectures, navigation*

General Terms

Design, Human Factors

Keywords

Accessibility, Blind, Usability, Screen Reader, Browsing Strategy

1. INTRODUCTION

Browsing the Web with screen readers can be a frustrating and challenging experience because of persistent accessibility and usability problems. Some content simply cannot be accessed non-visually with a screen reader. For instance, the content contained within images lacking alternative text cannot be conveyed to screen-reader users; widgets that assume a specific input device (*e.g.*, the mouse) cannot be easily used non-visually; and technology pushing the cutting-edge of web content (*e.g.*, Flash, Silverlight, SVG, HTML 5, etc.) is often incorrectly exposed by the APIs on which screen readers rely.

Many web accessibility problems are better characterized as usability problems that are encountered by screen-reader users. For instance, a web page with an excessive number of links embedded into its content (*e.g.*, Wikipedia) may be difficult to listen to because the screen reader seems to constantly interrupt the flow with an announcement about another link that it has encountered. A web page may be technically possible to access with a screen reader, but the access can be so cumbersome as to be almost unusable (*e.g.*, solving audio CAPTCHAs with common interfaces [7]). When usability problems reach a certain threshold of severity, they may entirely prevent access to content that is technically "accessible" [33] .

Despite the advances in screen-reading software, new web technologies appear on a regular basis, and new kinds of inaccessible content often emerge with them. As an example, screen readers were once unable to work correctly with the content changed by JavaScript, and now it also became important to interpret the semantics of dynamic content. For years the multimedia content in Flash was completely inaccessible to screen-reader users, and just as this is becoming less of an issue, HTML 5 (particularly the canvas) threatens to reintroduce similar problems. Screen readers have gotten better at updating their off-screen buffers with content affected by JavaScript, while ARIA have provided semantics to dynamic content. Some members of the HTML 5 working group are passionately arguing for accessibility to be included from the start. Screen-reader and browser developers, as well as those of us in the accessibility space, are constantly struggling to catch up, but usually remain months or years behind.

In the meantime, advanced end users who do not want to wait for the assistive technology to catch up develop a portfolio of browsing strategies for dealing with content that is inaccessible or unusable. These browsing strategies help mitigate the usability problems and overcome the shortcomings of current screen readers. In this paper, we provide a detailed overview of existing web accessibility problems and describe the coping strategies employed by screen-reader users to overcome these problems. We do not attempt to compare the efficiency of the strategies because the efficiency largely depends on the design of web sites and proficiency of users. Quantitative comparison would also be impractical, given the large scope of the strategy reviews. However, we believe that careful consideration of these strategies

W4A2010, April 26-27, 2010, Raleigh, USA. Co-Located with the 19th International World Wide Web Conference.

will not only expose neglected problems, but also suggest new research directions for improving usability for screen reader users.

We hope that this paper will be of interest to:

- Researchers wishing to expand their knowledge of browsing strategies and to identify open problems in this space;
- Web developers who want to design their web sites in a way that will enable the use of effective browsing strategies;
- Screen-reader users who want to learn new browsing strategies and make their web access more efficient.

We do not pretend to offer a comprehensive list of all possible browsing strategies; we rather start off a list and begin a classification of browsing strategies. Such a categorization is likely to fade over time – new web technologies change the notion what strategies are needed, and new assistive technologies change what strategies are necessary or possible. It is our hope that, in time, only the most effective strategies will persist. *The authors welcome everyone to suggest new strategies via e-mail or by adding them directly to the strategies document available at: http://www.sbhearsay.net/resources.htm.*

2. Related Work

In this section we give a broad overview of related work on Web Accessibility. Work that impacts the usability experienced by screen-reader users can be broadly divided into the following categories overviewed in this section: (i) existing screen readers, (ii) accessibility guidelines and verification, and (iii) research focused on understanding and/or improving accessibility.

2.1 Screen Readers

Most screen readers that are in popular use today are rather similar, and most have common insufficiencies that result in similar problems for users.

Desktop screen readers. Traditionally, screen-reader users have used Windows operating system. Although Microsoft Windows provides a choice of accessibility options such as the magnifying glass, text narration, etc., the majority of screen-reader users do not use the built-in accessibility features, giving preference instead specialized screen-readers[1] [21, 31, 36, 46] and magnification software [23, 47], that limit users mostly to the Windows platform. The recent improvement of Apple's built-in VoiceOver screen reader [40] has boosted the number of Mac users [45].

The name "screen reader" quite accurately describes how these tools first operated and, to a large extent, how they continue to operate today. Screen readers provide keyboard shortcuts that allow navigation of the content to be rendered to the screen in a non-visual way. In Section 3, we give more details on a typical screen-reader interface. Although popular due to their ability to make access to most content possible, they can often be severely limited in the usability they deliver, leading to the popularity of *non-visual web browsers* in the research and open source communities.

Non-visual web browsers. While enabling computer interfaces to the fullest is a challenging engineering effort, a number of projects have taken an easier approach and targeted only web browsing. The general goal of non-visual web browsers is to make a first-class aural interface to web browsers, drawing from the work of T.V. Raman's emacspeak [34].

Voice browsers are often implemented by augmenting existing web browsers [19]. One of the oldest representatives of voice browsers is IBM's Home Page Reader [3]. The latest IBM's prototype aiBrowser [28] implemented as a plug-in for the Microsoft Internet Explorer has mostly been targeting multimedia web content. FireVox [38] supported by Google is a Firefox browser extension enabling the basic browsing functionalities, as well as the support of ARIA on Windows, OsX, and Linux platforms. More recent work in this space includes the HearSay web browser [13] which integrates into various browsers on multiple platforms and converts web content into a VoiceXML dialog for aural interaction. WebAnywhere [10] is a server-side solution for remote web browsing from any computer terminal. Although voice browsers typically fare better than screen readers when it comes to web browsing, they typically copy the screen-reader interface and offer only minor improvements to the effectiveness of non-visual web browsing.

The state of assistive technologies. This section gives a representative overview of the modern-day assistive technologies that enable non-visual computer access and web access, in particular. Although these technologies empower blind users to browse the Web, most of the assistive tools available today perform minimal content analysis that could enable more intelligent non-visual interfaces; as a result the assistive tools limit their functionalities to navigation over the visual interface. Modern assistive technologies mostly leverage simple heuristics, such as determining labels for form elements by considering the labels located visually to the left. The latest version of Apple's VoiceOver does page-segmentation making it easier to navigate between segments. Overall however, the majority of assistive technology developers have been trying to make the Web more accessible, rather than trying to make non-visual web browsing more *efficient*.

If content were truly designed from the start for non-visual use, then many of the strategies that are common and necessary when using today's screen readers may not be required. Nevertheless, we believe that, for the advantages mentioned previously, screen readers will be here for some time; therefore, it is worthwhile to understand how screen-reader users are coping with the accessibility problems.

2.2 Accessibility Guidelines and Verification

The importance of promoting Web accessibility for individuals who are blind or visually impaired has prompted the World Wide Web Consortium [41] to launch the Web Accessibility Initiative [43] resulting in a number of web accessibility guidelines, *e.g.*, Web Content Accessibility Guidelines [44] addressing information in a web site, Accessible Rich Internet Applications [42] dealing with dynamic web content and web applications, Authoring Tool Accessibility Guidelines [5] attending to web development tools, User Agent Accessibility Guidelines [39] focusing on web browsers and media players, etc.

Although these guidelines are not backed by law, governments around the world are enacting accessibility regulations based on them. In the United States, the rights of people with disabilities are protected by Section 508 of the Rehabilitation Act of 1973,

[1] A comprehensive listing is available at: http://en.wikipedia.org/wiki/Comparison_of_screen_readers

further amended in 1998 to address Web accessibility and mandate that "electronic and information technology developed, procured, used, or maintained by all agencies and departments of the Federal Government be accessible both to Federal employees with disabilities and to members of the public with disabilities, and that these two groups have equal use of such technologies as federal employees and members of the public that do not have disabilities." Similar laws now exist in a number of other countries [32].

A 2006 case of the *National Federation of the Blind [30] vs. Target Corporation* has shown that laws go beyond paper. Based on the NFB's allegations that the inaccessibility of the *target.com* website violated the Americans With Disabilities Act, the California Unruh Civil Rights Act, and the California Disabled Persons Act, the U.S. Federal District Court for the Northern District of California certified the case as a class action lawsuit, which was later settled by Target. Major software companies, in an attempt to comply with these laws, have formed divisions that regulate internal compliance with industry standards and accessibility guidelines, for example, IBM's Human Ability and Accessibility Center [18], Sun Microsystems' [35], and Microsoft Corporation's [27].

Automated accessibility verification. Because understanding accessibility guidelines and verifying compliance of web pages can be difficult [22], a number of automated accessibility checkers were created to help web developers verify the accessibility of their websites, *e.g.,* [1, 2, 11, 29]. While automated verification can check websites' adherence to simple rules, it cannot check the semantics of accessibility metadata, *e.g.,* it cannot check whether the alternative text for images is meaningful, or whether an image used for design purposes should or should not have alternative text, because not all problems can be resolved automatically.

Manual accessibility verification and improvement. Upon encountering multiple accessibility problems, screen-reader users often report these problems to website administrators. Regrettably, it may take months before any changes are made, if at all. In an attempt to overcome this problem, the Social Accessibility (SA) Project [37] has created a social network that joins end-users and supporters, who collaboratively create external accessibility metadata and, in this way, improve the accessibility of web sites. The SA network enables screen-reader users to submit requests describing accessibility problems, and the supporters create accessibility metadata to resolve the problems.

A comparative study of various accessibility verification methods [25] has shown that using a screen reader as part of an evaluation process can help web developers perform better validation. Screen-reader users, however, are not always able to identify accessibility problems unless these problems prevent the users from completing their current task. This suggests the following subtlety of browsing strategies: if an effective strategy exists to overcome an accessibility problem, the effect of the accessibility problem is diminished. Nevertheless, developers should not expect the screen-reader users to know which appropriate strategy to use, because users have vastly different experiences.

Even when a web site is created according to standards, it can still be difficult or impossible to use. As a result, many screen-reader users are left behind as they cannot or will not visit the web sites plagued by numerous accessibility problems. Achieving usable accessibility remains a subjective task, giving rise to browsing strategies overviewed in this paper.

2.3 Improving Web Accessibility

Web accessibility research has attempted to improve accessibility and usability for screen-reader users. Although the approaches and targeted improvements differ, the end goal for many of these systems is either to simulate user browsing strategies or enhance web content so that the browsing strategies already employed by web users are possible, or are easier to use.

Automatic transcoding approaches. The continuing problems with web accessibility have led to attempts to automatically improve accessibility of web sites. One popular approach is transcoding – automatic modification of the original content before it reaches end-users. Transcoding of web sites was originally developed to adapt them for mobile devices [6] and to personalize pages [24]. To date, this technique has been extensively explored for web accessibility [4]. Many transcoding approaches are based on a proxy server performing the transcoding. Some representative examples include WebInSight [9] that automatically adds alternative text to images, SADIe [20] that uses an ontology and CSS to make pages more accessible, and AxsJAX [15] that transcodes web pages by injecting ARIA metadata.

2.3.1 Handling Dynamic Content

Most Rich Internet Applications (RIA) are currently accessible only to users who visually interact with the dynamic content. If web developers properly exposed states and transitions of their websites, screen-readers would be able to interact with rich content. Unfortunately, interactive web applications are built with a variety of technologies and toolkits, many of which make RIA web sites partially or completely inaccessible. Over the last few years, there have been several efforts to improve the accessibility of dynamic content by manual and automatic authoring of accessibility metadata.

ARIA markup. The use of the W3C standard for Accessible Rich Internet Applications [42] was one of the first attempts to make RIAs accessible. ARIA markup is intended to be used by screen-readers to improve the accessibility of web applications for screen-reader users. ARIA metadata can be embedded into web pages and can be used to describe live areas, roles, and states of dynamic content; *e.g.*, "* Interrupting Message *" instructs screen readers to immediately read the content of the paragraph tag when it changes dynamically. Other possible values include "*off,* " "*polite,* " and "*assertive.*" Regrettably, most of the dynamic content available today does not implement the ARIA standard. Likewise, web developers are unlikely to follow ARIA consistently, for they have not followed other accessibility guidelines. ARIA can also be supplied as part of reusable components or widgets; for example, Dojo Dijit [16] provides ARIA-enabled widgets and a toolkit to build custom accessible widgets. However, Dijit is only one of many available toolkits for building RIAs, and web developers continue creating new inaccessible custom widgets of their own.

Unfortunately, ARIA only helps users navigate content when web developers have included the markup. Screen-reader users have developed strategies for dealing with dynamic content that has not been assigned ARIA markup (Section 5).

Transcoding approaches. ARIA can also be dynamically applied through transcoding. Originally targeting Google's own web pages, the Google AxsJAX project [15] now enables any programmer to write scripts that will automatically inject ARIA markup into existing web applications, eliminating the requirement that the creator of the content provide the markup. The downside of this approach is that it still requires manual effort.

2.3.2 Screen-Reader Support of Dynamic Content

Most voice browsers and screen-readers integrate themselves with regular web browsers such as Internet Explorer or Mozilla Firefox, to obtain web content in the form of HTML DOM trees. Web browsers often give access to extensive APIs through plug-ins, add-ons, or extensions, which provide an easy way to track changes in web pages by exposing DOM mutations (dynamic changes) and page-load events. For example, when a new web page finishes loading, a screen reader can use the page-load event as a clue to start reading the new page. Similarly, when a DOM mutation event occurs, a screen reader could identify and present changed content to the users.

Traditional screen-readers, such as JAWS, maintain a static representation of web content, or off-screen buffer, and allow users to browse within it. Until recently, screen readers did not update their view of a web page except when a new page loaded. Although this buffer can be refreshed manually or automatically at certain time intervals, a survey described in [8] has shown that screen-reader users prefer to suppress updates and often end up browsing and acting upon stale content. On the other hand, the FireVox [38] browser, reading directly from the HTML DOM, announces the content updates as they occur. However, without ARIA attributes, FireVox is unable to filter out irrelevant updates and often interrupts and distracts users from the tasks they are working on. HearSay-Dynamo dynamically updates its reading buffer and can optionally notify its users of dynamic events by playing earcons [14], which are less distracting than reading out changed content.

Part of the reason why dynamic content is still inaccessible to current assistive technologies is the difficulty of distinguishing between important content changes (*e.g.*, dynamic form validation) and irrelevant updates (*e.g.*, dynamic ads). While dynamic web pages can be confusing even to sighted users, screen-reader users have to interact with dynamic content through a serial audio interface; therefore, even important content changes can be distracting to users if the changes are presented as they arrive. ARIA [42] markup, as was previously discussed, allows web developers to specify the importance of updates, and whether they should be ignored, delivered at a convenient time, or announced to users. To the best of our knowledge, no usable interface currently exists for browsing dynamic content without ARIA markup.

Other types of inaccessible content include Flash and Java Applets. Screen-readers currently provide limited support for Java Applets and Flash objects. Flash often remains inaccessible to users because web developers forget to add alternative text to flash controls such as buttons. aiBrowser [28] helps improve the usability of Flash content.

2.4 Remaining Problems

Despite the extensive research in this area, the screen-reader interfaces commonly in use have not changed all that much. Contrary to the advice of usability experts, screen readers still act as an imperfect interface to content that is often fundamentally designed for visual use. This situation seems unlikely to change in the near term because most web content is still produced in a similar way, and it is difficult to adapt it for interfaces targeting audible display.

3. SCREEN-READER INTERFACE

Most state-of-the-art screen readers employ a shortcut-driven interface, which allows users to navigate between webpage elements in a serial manner. For instance, "Up" and "Down" arrow keys are often used to skip between adjacent elements in the order they appear in the HTML source page. To navigate between elements of a specific type, *e.g.*, links, buttons, edit fields, headings, etc., users can press *[Shift+] A | B | E | H* | etc., to go to the [previous] next item of the respective type.

Screen readers typically read out text content, as well as the types and states of complex web page elements, *e.g.*, "textbox blank", "list with 10 items", etc. Optionally, many screen readers can also narrate text formatting and colors. "Left" and "Right" arrow keys allow users to spell out letters and numbers, when necessary, which is especially useful for reading unusual combinations of characters and editing web forms.

Screen readers often have one or more edit modes that allow users to interact with different types of form elements. For example, in the navigational mode, pressing "E" on the textbox simply moves the screen-reader cursor to the next textbox, without typing "E." However, after pressing "Enter" on the textbox, the user can type in the textbox. Everything the user types is echoed aloud (except passwords), and when the user changes the position of the edit cursor, screen readers read the character that follows the cursor. Pressing "*[Shift+]Tab*" moves the screen-reader cursor to the next editable or focusable element. Some screen readers switch into the edit mode when their cursor is on an editable element.

To access elements of certain types (*e.g.*, links), the user can open an auxiliary window containing a list of all elements of that type (Figure 1). The user can then navigate the list to get an overview of the page, jump to the element selected in the list, or return to the initial position on the page. Some screen readers support landmarks, allowing users to set a landmark and then return to it from anywhere on the page.

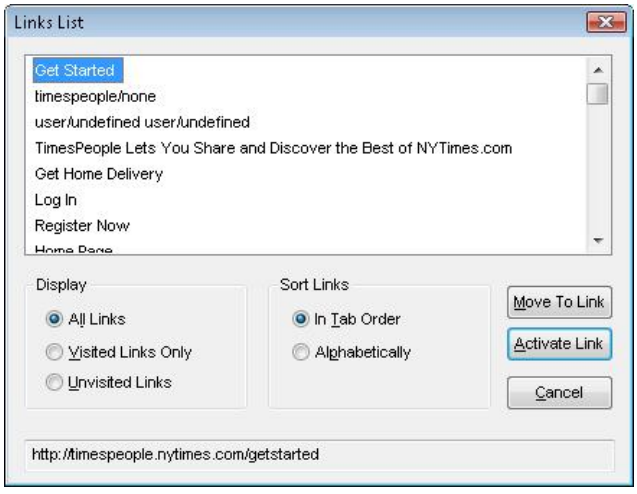

Figure 1 – Dialog Displaying a List of Links in JAWS

4. BROWSING STRATEGIES

The biggest problem in non-visual browsing remains the speed of information processing. To overcome the limitations of screen-reader interfaces, blind users develop browsing strategies that allow them to browse more efficiently. In this section, we describe some general browsing strategies that were observed in the course of several user studies. Many of the basic strategies can be used in complex scenarios (Sec 5-6).

Using Braille displays. Web access with screen readers is in general inefficient due to the limitations of the serial audio interface. Refreshable Braille displays (Figure 2), which are supported by many screen readers, allow users to read web content in Braille. The displays typically have a low resolution of only 40-80 cells, but are still a linear display. Such displays do not offer much improvement, other than providing an alternative modality that is especially important for deaf-blind people. Expert users also find Braille displays more convenient for text editing tasks.

Figure 2 – Braille Display

Increasing the speech rate. To accelerate reading, many users speed up the speech rate with which the screen reader narrates the content. Newly-blind individuals generally prefer naturally sounding voices with normal speech tempo, which are easier to understand. However, expert users speed up the speech rate and choose older formative speech synthesizers whose quality does not degrade as much as newer concatenative speech synthesizers as high speech rates. Congenitally blind users who grew up listening to synthesized speech are able to understand it at an astounding speed of up to 500 words per minute. An important aspect of speed is also the responsiveness of the screen reader to key presses. Expert users expect the delay between a key press and the following speech output to be under 50-100ms.

Individual preferences. Screen-reader users often develop favorite strategies for web browsing. For example, on unfamiliar web pages, many users first try to get a general overview of the page by navigating all headings (using the "H" key with most screen readers). If they cannot locate the information of interest using this strategy, they return to the beginning of the page to read through the entire page. Screen-reader users tend to remember the order of headings and other "landmarks" on frequently visited web pages. For example, the user may remember that the 3rd and 4th textboxes on a page are the user name and password fields. Then, by using the landmarks as points of reference, users can quickly navigate to the part of the page that they want to read. For example, knowing that the product price often precedes the "*add-to-cart*" button, screen-reader users may press "B" to find the button and then easily find the price (Figure 3). It is notable, that users may prefer to navigate the page backwards if they figure out that they can get to the desired content faster this way.

Tasked-based preferences. Browsing strategies also vary based on the task at hand. For example, in form-filling tasks, screen-reader users may choose to press the "Tab" key to iterate over all focusable elements, including form elements and links, or iterate over form fields of specific type, *e.g.*, press "X" to go to the next checkbox, press "E" to navigate to the next editable element. While reading headline news, the users may press "A" to iterate over the headlines links to get an overview of the latest news (headline news are now increasingly labeled as headings). Users can also choose to navigate over visited or unvisited links, which allows them to narrow the scope of their search (Figure 1). If they know about some specific text occurring on the page, users can search directly for the text. If the screen reader can store user-created landmarks on a web page, *e.g.*, the beginning of main content, users can directly jump to the location indicated by the landmark.

Figure 3 – Shopping for Books

Inferring roles of controls. Typically, inaccessible content includes image-links and buttons without alternative text, dynamic content and widgets, Flash, Java Applets, etc. Sometimes users manage to infer the roles of image buttons by exploring other web elements nearby, *e.g.*, search boxes are followed by search buttons in Figure 5. In other cases, users may follow an image-link to read the title of the target page or click a button to determine its function. On subsequent visits, screen-reader users may remember the ordinal position of the links and other web objects they need. This approach is often the only solution for inaccessible Flash and Java Applets. The downside of this approach is that it is very hard to infer roles of controls that have to be clicked in a certain sequence. Trying to discover the relationships between objects by invoking every object is similar to automated AJAX crawling [17, 26], known to be suffering from: state explosion, irreversibility of actions (which requires that transitions be retraced from the start state), adverse actions (*e.g.*, inadvertent deletion of an email), latency between actions and reactions (*e.g.*, AJAX applications).

Exploring Visual Interface with a Keyboard-Driven Mouse. Some web developers choose to use non-focusable custom widgets that can be edited, clicked, or otherwise acted upon, but are inaccessible to a standard screen reader. The simplest example is an image that looks like buttons and has triggers for mouse-events, *e.g.*, all three buttons in Figure 5 are images, which will be fully accessible if they are implemented as image-buttons and have proper alternative text. To access these elements, users may use their screen readers to simulate the mouse. Many screen readers provide a mechanism to move the current mouse position with the keyboard and speak the content beneath the cursor when possible. Other keyboard shortcuts enable the user to click the mouse buttons. Although this can enable screen-reader users to access content that is not straight-forwardly accessible using only standard keyboard shortcut keys, it is an incredibly inefficient, confusing, and frustrating browsing strategy that many users will only employ as a last resort.

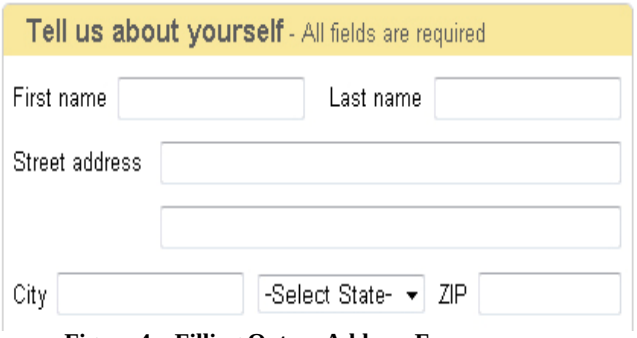

Figure 4 – Filling Out an Address Form

Form filling. Inaccessible content in forms often includes form fields without labels. While editing unlabeled form elements, users can often guess the label by using their prior experience with other forms. For instance, the typical order of address forms is First Name, Last Name, Street Address, City, State, and Zip (Figure 4); so, if there is an unlabeled textbox between the "Street Address" and "City" textboxes, then it is likely to be the second line for the "Street Address." By examining the content of the form element, it is sometimes possible to guess the purpose of the element, *e.g.*, the "State" form element is often represented by a list-box with the choices of the states.

Screen-reader users may choose to press the "Tab" key to iterate over all focusable elements, including form elements and links, or iterate over specific elements by using shortcuts provided by screen readers. One drawback to this approach is that users will miss out on content presented next to the element, explaining how to fill out the form such as the desired format of a phone number. Tabbing is also hindered by pages with poorly labeled form elements, which can disorient the user particularly with large forms.

Falling back to external help. When all else fails, screen-reader users may ask for help from their sighted friends and family. The Social Accessibility [37] network was created to bring together end users and volunteers who can collaboratively create and utilize accessibility metadata. The network allows end users to submit requests for help with inaccessible content and allows volunteers to suggest strategies for overcoming problems, as well as label inaccessible content.

Figure 5 – Variations of Search Boxes

Many users avoid visiting dynamically changing web pages because screen readers still do not adequately support dynamic content. In the cases where this may not be possible, *e.g.*, dynamic form-filling validation, users manually refresh the screen-reader's buffer and look for the changes. A general strategy for looking for statically changed information is to review the list of unvisited links (*e.g.*, look for new headlines on self-refreshing news sites – Figure 8) or to read through the entire page (*e.g.*, looking for error messages in form filling – Figure 7). A more detailed overview of strategies employed in dealing with dynamic and static content changes is given in Section 5.

Figure 6 – "Undo" link appearing at Gmail.com

Users often try to avoid reading through menus and other irrelevant content that appear at the top of the page. One popular strategy for finding the beginning of main content is the use of the "H" key to navigate over headings, which, if used consistently within the web page, can substantially improve the accessibility of a web site. Another common strategy is searching for text that is expected to occur in the main section. Finally, some screen readers such as JAWS, include a keyboard shortcut (the "M" key) that allows users to skip to content that does not contain links. The effect is to skip past navigation menus. A detailed overview of strategies employed in finding desired content is given in Sec. 6.

Although all users develop favorite web browsing strategies, the existence of sophisticated error-handling strategies among screen-reader users indicates that there are still a number of web accessibility problems. It is our hope that this overview of browsing strategies sheds some light on current Web Accessibility problems and will help researchers in the field focus on solving these problems.

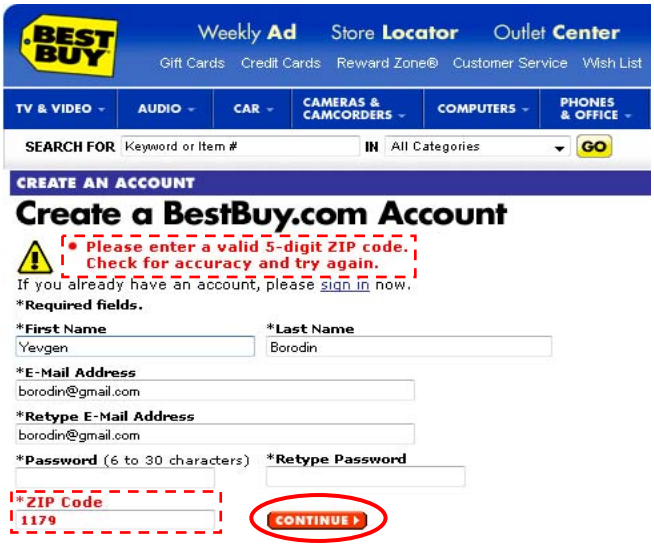

Figure 7 – Server-Side Validation at BestBuy

5. CONTENT CHANGES

Although the most common strategy for dealing with dynamic content is evading it [8], sometimes dealing with dynamic changes is unavoidable. In this section, we discuss several different and yet similar scenarios that demonstrate the problems faced by screen-reader users and motivate a new approach.

Ignoring dynamic content. Using current screen readers, users are not notified of dynamically appearing messages and do not have easy access to updated content, unless it is marked up with ARIA. In many cases, users can simply ignore dynamic messages either because the messages are irrelevant, or because they do not preclude users from accomplishing their tasks. For example, Figure 6 shows a dynamic message (dashed box) appearing on the Gmail.com web page as soon as one deletes an e-mail. In this instance, users can ignore this message because there are other ways of recovering a deleted e-mail, *e.g.*, going to the "Trash" folder and moving the deleted e-mail to the "Inbox." However, it would be considerably easier for users to simply jump to the update message and click the "Undo" link.

Other types of dynamic content, such as pull-down menus, are often implemented by visually hiding the submenus and making them appear on mouse-over. As most screen-readers do not make a distinction between hidden and visible content, users can still access such content in a static fashion. This approach, however, fully depends on the way the dynamic content is implemented. For example, AJAX delivers content on demand; hence, it may not be in the DOM tree unless some event triggering AJAX takes place and the screen reader detects the update. In other cases, there may be too much hidden content causing information overload to screen-reader users, or hidden content can be confusing, as described in form-filling strategies.

Form filling. A more serious problem is captured in Figure 7, where an incorrectly entered zip code results in error messages (dashed boxes) appearing on the page. Only the first message is actually new, the other one just changes the color to red to visually attract users' attention. Form validation can be done both dynamically (on the client-side) or statically (on the server-side). Either way, from the point of view of a novice user, the form did not submit because, at first glance, nothing has changed. A more experienced user would know about form validation and would look for error messages. Error messages usually appear above the form or near the form fields with errors. Users who tab through the form elements will not encounter the error messages, unless the error is somehow indicated (other than with color) in the label of the form field. So, most users can only identify the error by carefully checking every field, which may not be obvious in case of semantic errors (*e.g.*, "Airport code is not recognized"). A more efficient strategy may be to check for error messages above the first form field or near every form field. However, this strategy fails in cases when web designers visually hide all error messages, without realizing that screen readers will often read them anyway. If users had an easy way to review the changes, they would be able to read the error message and then jump to the highlighted field in no time at all [12].

Automatic refreshes. Figure 8 illustrates an automatic page refresh that causes a news item to be inserted into the content of the Los Angeles Times news web site (solid box Figure 8b). So, if the screen-reader's cursor was on the article about the Pope's birthday party before the refresh (dashed box Figure 8a), instead of continuing to read about the same article (dashed box Figure 8b) after the refresh, screen readers start from the beginning.

a) Before **b) After**

Figure 8 – Page Refresh at L.A. Times New

Refreshes happen to nearly 50% of major news websites [12] and often occur at frequent 5-10 minute intervals. Refreshes can be very disruptive to the user because screen readers typically make the user read the page from the beginning. Some screen readers such as JAWS have an option of suppressing automatic refreshes, but sometimes users may choose to refresh the page manually, in which case users either have to read the page from the beginning or employ a more efficient strategy. In case of news websites, users can skip through the headings or look at the list of unvisited links. In other cases, users have to be familiar with the web site and know what they are looking for. For instance, they could employ a landmark-based strategy to jump to the location where they expect changes to occur after a manual refresh; for instance, users could search for "Current Bid" when they watch an item's price on eBay, as illustrated by a solid-line rectangle in Figure 9. The dashed box illustrates the dynamically changed "Time left" field.

Figure 9 – Watching products on E-bay.com

Template-based websites. Because the design of most web sites is based on templates, screen-reader users have to learn how to use a number of strategies to skip repeated template content. Typical user strategies include: skipping to the nearest heading (JAWS shortcut "H"), which works only if the HTML source code implements heading tags; skipping to the next block of contiguous non-link content of a predefined number of words (JAWS shortcut "N"), which may or may not work, depending on the density of links in the template and main content; skipping to the next paragraph (JAWS shortcut "P"), which only works if the HTML source code implements paragraph tags, etc. Overall, the strategies used to avoid repeated template content are similar to those used to locate main page content as described in Section 6. An easy and reliable interface for reviewing changes in web pages could help users skip repetitive template content and proceed to browsing main page content. Finally, the interface has to be the same, regardless of whether the content is updated by statically loading every new page after following a link, or if the AJAX-like approach is used to change the main content dynamically, without actually reloading the template.

6. FINDING DESIRED CONTENT

Screen-reader users tend to develop their own strategies for getting past irrelevant web page content in order to find the content that they want. Based on our observation, we have identified several user strategies for finding desired content in web pages: using skip-to-main-content links; heading-level navigation; keyword searching; skipping by paragraph or non-linked text; and sequential browsing (continuously reading or going line by line through web pages). Advanced users tend to have favorite strategies that they employ in a sequence as strategies fail. Some strategies are based on the user's familiarity with a web page and may evolve as the user becomes more familiar with the page.

Skip-to-main-content links. In an attempt to follow web accessibility guidelines, web developers often place hidden links that allow screen-reader users to skip navigation menus and start reading from the main content of a page. However, since these links are hidden, and developers are often using visual web-development tools, these same developers, or the newly recruited developers, fail afterwards to maintain the skip links. A survey of browsing behaviors [8] has shown that screen-reader users often ignore skip-links because the links are often broken. In a WebAIM survey [45], over 1,000 users responded that they use skip links: whenever they are available 22%, often 16%, sometimes 28%, seldom 19%, and never 10%. Users who do not use skip-links often motivate their choice by a lack of control of where the skip-links take them, combined with prior experience of seeing broken skip-links, and also, for fear that they may skip something important.

Heading navigation. Perhaps the most common technique for skipping to main content is the use of heading-navigation shortcuts that jump to the next or previous HTML heading tag (<H1>-<H6>). In the WebAIM survey, over 1,000 users responded that they used headings: whenever they were available 52%, often 24%, sometimes 14%, seldom 5%, and never 2%.

Keyword search is another technique employed by users to find main page content. Keyword search helps users when they know what they are looking for, *e.g.*, after following a link to an article, one can look for the words occurring in the title of the article to jump directly to the article. Users often search for word combinations, single words, or even letter combinations to find main content.

Skipping to non-linked content in paragraphs. Because the irrelevant content often contains many links, some users employ the "*P*" shortcut to skip to the nearest paragraph of text. This technique works only if <P> tags are used in the HTML source code of the webpage. A more efficient strategy is the use of the "*N*" shortcut that allows users to skip to the next section of contiguous non-linked content, which may not work in web pages abundant in linked content.

Sequential navigation. When all other strategies fail, or if users do not have an efficient strategy, they resort to sequential navigation. In these cases, users either instruct their screen readers to read continuously or arrow down through the entire page. Sequential browsing is the most inefficient and time-consuming browsing strategy.

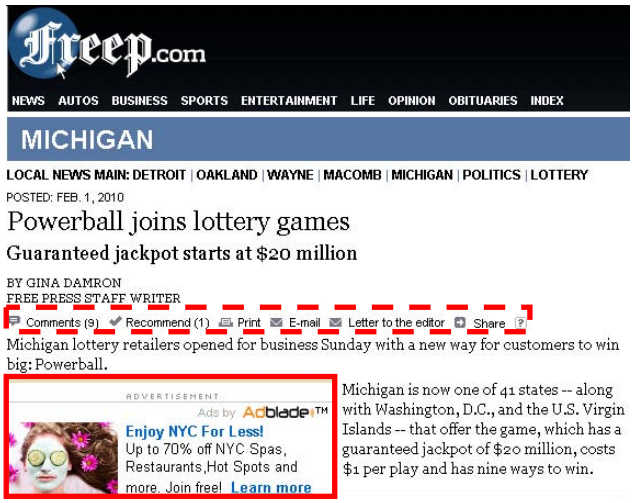

Figure 10 – Adds at the Free Press news web site

Strategies for skipping irrelevant information. Very often the beginning of main page content coincides with the title of an article, product description, etc. However, having found the title with any of the described strategies, users still have to skip through irrelevant links, ads, and lists of related articles or products that web developers tend to place in between the title and the rest of the content, *e.g.*, the dotted rectangle in Figure 10 encompasses the controls that separate the title of the article from its body. Even worse, irrelevant content is sometimes placed right in the middle of main content, forcing users to guess if it represents the end of the content or just some annoying interruption, *e.g.*, the solid red rectangle in Figure 10 encompasses an ad that is inserted after the first paragraph of the article. While many screen-reader users end up reading through the ads for fear of skipping relevant information, others try to skip ads.

Strategies for skipping irrelevant information usually include paragraph and non-linked-text navigation. Since ads often appear in inline frames (<iframes>), some users try to skip them using

frame navigation or turning off iframes altogether, thus making frames invisible. However, in the latter case, users may inadvertently turn off other important frames; for instance, Gmail has several frames for the list of emails, chat, etc. When reading news, screen-reader users often switch to print view, which removes the ads and minimizes the number of steps needed to read the article.

7. IMPLICATIONS

A better understanding of the browsing strategies employed by current screen-reader users can help inform the tools that we create to both help less experienced users and to identify the research directions that we choose to pursue in the future. One of the implications of the existing strategies is that, if a clear and effective browsing strategy lets users bypass an apparent accessibility or usability problem, then that problem may not be as important as the ones that do not have such strategies.

Far from being a comprehensive list, the browsing strategies outlined in this paper are meant merely as a starting point. We hope that other users and researchers will start to add their own browsing strategies – a repository that would be quite useful for both users looking to become more proficient with their tools and researchers seeking to create the next generation of assistive technology.

Although we have listed a large number of strategies that we have observed and the utility of which we occasionally commented on, we have not attempted a quantitative evaluation of their effectiveness. A ripe area for future research is to investigate the browsing strategies that we have identified here to formally gauge their utility. Finally, we hope that our effort to overview existing browsing strategies will help future assistive technology developers and researchers create a more usable experience for screen-reader users.

8. ACKNOWLEDGMENTS

We would like to thank NSF (Awards: IIS-0534419, IIS-0808678, CNS-0751083) and Sigma Xi (Award: G20071013028487872).

9. REFERENCES

[1] ACChecker. *Web Accessibility Checker*. 2009 [cited 2009; Available from: http://achecker.ca/checker/index.php.

[2] aDesigner. *IBM Alphaworks project donated to Accessibility Tools Framework (ACTF)*. 2004 [cited 2009; Available from: http://www.alphaworks.ibm.com/tech/adesigner.

[3] Asakawa, C. and T. Itoh, *User interface of a Home Page Reader*, in *Proceedings of the 3rd International ACM Conference on Assistive Technologies*. 1998, ACM: Marina del Rey, California, United States.

[4] Asakawa, C. and H. Takagi, *Transcoding*, in *Web accessibility: a foundation for research*, S. Harper and Y. Yesilada, Editors. 2008, Springer Publishing Company, Incorporated. p. 388.

[5] ATAG. *W3C Authoring Tool Accessibility Guidelines*. 2009 [cited 2009.

[6] Bickmore, T.W. and B.N. Schilit, *Digestor: device-independent access to the World Wide Web*, in *Selected Papers from the 6th International Conference on World Wide Web*. 1997, Elsevier Science Publishers Ltd.: Santa Clara, California, United States.

[7] Bigham, J.P. and A.C. Cavender, *Evaluating existing audio CAPTCHAs and an interface optimized for non-visual use*, in *Proceedings of the 27th international conference on Human factors in computing systems*. 2009, ACM: Boston, MA, USA.

[8] Bigham, J.P., A.C. Cavender, J.T. Brudvik, J.O. Wobbrock, and R.E. Lander, *WebinSitu: a comparative analysis of blind and sighted browsing behavior*, in *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility*. 2007, ACM: Tempe, Arizona, USA.

[9] Bigham, J.P., R.S. Kaminsky, R.E. Ladner, O.M. Danielsson, and G.L. Hempton, *WebInSight: making web images accessible*, in *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*. 2006, ACM: Portland, Oregon, USA.

[10] Bigham, J.P., C.M. Prince, and R.E. Ladner, *WebAnywhere: a screen reader on-the-go*, in *Proceedings of the 2008 International Cross-Disciplinary Conference on Web Accessibility (W4A)*. 2008, ACM: Beijing, China.

[11] Bobby. *Accessibility Checker*. 1995 [cited 2009; Available from: http://www.cast.org/products/Bobby.

[12] Borodin, Y., J.P. Bigham, R. Raman, and I.V. Ramakrishnan, *What's new?: making web page updates accessible*, in *Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility*. 2008, ACM: Halifax, Nova Scotia, Canada.

[13] Borodin, Y., J.P. Bigham, A. Stent, and I.V. Ramakrishnan, *Towards one world web with HearSay3*, in *Proceedings of the 2008 International Cross-Disciplinary Conference on Web Accessibility (W4A)*. 2008, ACM: Beijing, China.

[14] Brewster, S.A., P.C. Wright, and A.D.N. Edwards, *An evaluation of earcons for use in auditory human-computer interfaces*, in *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. 1993, ACM: Amsterdam, The Netherlands.

[15] Chen, C.L. and T.V. Raman, *AxsJAX: a talking translation bot using Google IM: bringing Web-2.0 applications to life*, in *Proceedings of the 2008 International Cross-Disciplinary Conference on Web Accessibility (W4A)*. 2008, ACM: Beijing, China.

[16] Dijit. *Dojo Dijit JavaScript toolkit*. 2009; Available from: http://dojotoolkit.org/projects/dijit.

[17] Duda, C., G. Frey, D. Kossmann, R. Matter, and C. Zhou, *AJAX crawl: making AJAX applications searchable*, in *Proceedings of the 2009 IEEE International Conference on Data Engineering*. 2009, IEEE Computer Society.

[18] HAAC. *IBM Human Ability and Accessibility Center*. 2009 [cited 2009; Available from: http://www-03.ibm.com/able/.

[19] Hanson, V.L., J.T. Richards, and C. Swart, *Browser augmentation*, in *Web accessibility: a foundation for research*, S. Harper and Y. Yesilada, Editors. 2008, Springer Publishing Company, Incorporated. p. 215-230.

[20] Harper, S. and S. Bechhofer, *SADIe: structural semantics for accessibility and device independence*. ACM Transactions on Computer-Human Interaction, 2007. **14**(2): p. 10.

[21] JAWS. *Screen reader from Freedom Scientific*. 2009 [cited 2009; Available from: http://www.freedomscientific.com/products/fs/jaws-product-page.asp.

[22] Kelly, B., D. Sloan, L. Phipps, H. Petrie, and F. Hamilton, *Forcing standardization or accommodating diversity?: a framework for applying the WCAG in the real world*, in *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*. 2005, ACM: Chiba, Japan.

[23] MAGic. *Magnification software from Freedom Scientific*. 2009 [cited 2009; Available from: http://www.freedomscientific.com/products/lv/magic-bl-product-page.asp.

[24] Maglio, P. and R. Barrett, *Intermediaries personalize information streams.* Communications of the ACM, 2000. **43**(8): p. 96-101.

[25] Mankoff, J., H. Fait, and T. Tran, *Is your web page accessible?: a comparative study of methods for assessing web page accessibility for the blind*, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2005, ACM: Portland, Oregon, USA.

[26] Mesbah, A., E. Bozdag, and A.v. Deursen, *Crawling AJAX by inferring user interface state changes*, in *Proceedings of the 2008 8th International Conference on Web Engineering - Volume 00*. 2008, IEEE Computer Society.

[27] Microsoft. *Microsoft Accessibility*. 2009 [cited 2009; Available from: http://www.microsoft.com/enable/.

[28] Miyashita, H., D. Sato, H. Takagi, and C. Asakawa, *Aibrowser for multimedia: introducing multimedia content accessibility for visually impaired users*, in *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility*. 2007, ACM: Tempe, Arizona, USA.

[29] MVS. *W3C Markup Validation Service*. 2009 [cited 2009; Available from: http://validator.w3.org/.

[30] NFB. *National Federation of the Blind*. 2009 [cited 2009; Available from: http://www.nfb.org/nfb/Default.asp.

[31] NVDA. *NonVisual Desktop Access*. 2009 [cited 2009; Available from: http://www.nvda-project.org/.

[32] Paciello, M.G., *Web accessibility for people with disabilities*. 2000: C M P Books. 392.

[33] Petrie, H. and O. Kheir, *The relationship between accessibility and usability of websites*, in *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2007, ACM: San Jose, California, USA.

[34] Raman, T.V., *Emacspeak - direct speech access*, in *Proceedings of the second annual ACM conference on Assistive technologies*. 1996, ACM: Vancouver, British Columbia, Canada.

[35] Sun. *Sun Microsystems Accessibility*. 2009 [cited 2009; Available from: http://www.sun.com/accessibility/index.jsp.

[36] SuperNova. *Screen reader from Dolphin*. 2009 [cited 2009; Available from: http://www.dolphincomputeraccess.com.

[37] Takagi, H., S. Kawanaka, and M. Kobayashi, *Social Accessibility: achieving accessibility through collaborative metadata authoring*, in *ASSETS*. 2008: Halifax, Canada.

[38] Thiessen, P. and C. Chen, *Ajax live regions: ReefChat using the Fire Vox screen reader as a case example*, in *Proceedings of the 2007 International Cross-Disciplinary Conference on Web Accessibility (W4A)*. 2007, ACM: Banff, Canada.

[39] UAAG. *W3C User Agent Accessibility Guidelines*. 2009 [cited 2009.

[40] VoiceOver. *Screen reader from Apple*. 2009 [cited 2009; Available from: http://www.apple.com/accessibility/voiceover.

[41] W3C. *World Wide Web Consortium*. 2009 [cited 2009; Available from: http://www.w3.org.

[42] WAI-ARIA. *W3C Accessible Rich Internet Applications*. 2009 [cited 2009; Available from: http://www.w3.org/TR/wai-aria.

[43] WAI. *W3C Web Accessibility Initiative*. 1997; Available from: http://www.w3.org/WAI/.

[44] WCAG. *W3C Web Content Accessibility Guidelines*. 2009 [cited 2009; Available from: http://www.w3.org/TR/WCAG10/.

[45] WebAIM. *Web accessibility in mind survey*. 2009 [cited 2009; Available from: http://www.webaim.org/projects/screenreadersurvey/.

[46] Window-Eyes. *Screen Rrader GW Micro*. 2009 [cited 2009; Available from: http://www.gwmicro.com/Window-Eyes.

[47] ZoomText. *Magnification tool from aisquared*. 2009 [cited 2009.