Web Accessibility for Low Bandwidth Input

Jennifer Mankoff[1] Melody Moore[2] Udit Batra[3] Anind Dey[1]
[1]EECS Dept. UC Berkeley {jmankoff,dey}@cs.berkeley.edu
[2]Computer Information Systems Dept. Georgia State University moore@gsu.edu
[3]College of Computing Georgia Tech udit@cc.gatech.edu

ABSTRACT
One of the first, most common, and most useful applications that today's computer users access is the World Wide Web (web). One population of users for whom the web is especially important is those with motor disabilities, because it may enable them to do things that they might not otherwise be able to do: shopping, getting an education, running a business. This is particularly important for low bandwidth users: users with such limited motor and speech that they can only produce one or two signals when communicating with a computer. We present requirements for low bandwidth web accessibility, and two tools that address these requirements. The first is a modified web browser, the second a proxy that modifies HTML. Both work without requiring web page authors to modify their pages.

KEYWORDS: WWW, motor disability, low bandwidth input

INTRODUCTION
The goal of universal access is to make services and information accessible to everyone. One of the first, most common, and most useful of these is the World Wide Web (web). Because of this, much research in accessibility has focused on developing guidelines and tools in support of universal web access. Examples include accessibility guidelines [37, 32] and numerous services for vision-impaired users [27, 31, 18, 2, 38, 9, 15, 34, 14], the people most obviously in need of support when dealing with the graphics and text contained in web pages.

While a few of these systems consider motor impairments [14, 15], none of the guidelines or tools address the needs of low bandwidth users: users who can only produce one or two signals when communicating with a computer. For example, the W3C recommendation for device independence [37, Guideline 9.] states that: "Generally, pages that allow keyboard interaction are also accessible through speech input or a command line interface." Although the number of low bandwidth users is small compared to the number of people with vision impairments, its need is great. A motor-impaired user generally has limited mobility, and access to the services and resources on the web can give him or her increased independence. Our goal is to improve the accessibility of the web to people with motor disabilities who cannot easily use a keyboard, speech, or a mouse or mouse substitute.

More than 85% of elders (people over 65) have some physical impairment [8]. A variety of degenerative disease and other conditions or injuries may impair motor function for people of all ages [22]. Example disabilities include cerebral palsy, late-stage ALS (Lou Gehrig's disease), brain stem strokes, and certain spinal-cord injuries. Many of these disabilities also result in impaired speech, leaving few options for communicating with a computer. People with motor impairments use a variety of creative solutions to control their environment and computers. The particular ones that this work is most applicable to are "switch" interfaces. In addition to the traditional button-activated switches, examples include sip-puff, or pneumatic, switches in which air is used to flip a switch [22]; switches in which a muscle such as the eyebrow is used as a signal [22]; and EEG or other "brain computer interfaces" in which the user creates signals by controlling her thoughts in different ways [26]. A user may have one or more such switches. Unlike a mouse, these input devices produce discrete signals. They may be compared to a very limited keyboard, with as little as one key.

Web browsers and pages are not built to be accessible to such a small number of input signals. They are generally designed with a mouse in mind, and are not very accessible even to users with a standard and keyboard but no mouse. The mouse is normally used to click on links, move the scroll bar, and control navigation forward and back in the history. Although all of these actions have keyboard counterparts (for example: tab, return, page up, page down, alt left and alt right, in Netscape™), they are much easier to do with a mouse. For example, the user must tab through the links in a page in sequence, resulting in a linear access time, often increased by the repeating navigational links present at the top of each page in many web sites. Suppose the user opens a web page, uses the space bar to page down, and identifies a link of interest. She will have to tab through all

of the links starting at the top of the page to get to it, because use of the space bar does not change the current link focus, which defaults to the first link of the page. She may even lose her spot as the browser moves the viewpoint to match each link she tabs through.

Support for low bandwidth users

Given the large number of web pages in existence today, the most promising approach to addressing problems such as these is to automatically add accessibility features to web pages. We have identified requirements for users with severe motor impairments and have created tools that automatically make the adjustments necessary to provide access to the web.

In all, we have identified seven requirements for such a tool, ranging from navigation support to dealing with forms. We present two approaches to automating support for these requirements. One is a browser interface that requires no changes to the HTML of web pages. The other is a proxy server that modifies HTML to be more accessible from any browser. Neither requires the authors of web pages to make changes.

The first approach allows finer control over modifications and provides a platform on which we can experiment with different mappings between input signals and action based on errors and fatigue. The second approach is platform independent, and available to any user without requiring that any special software be installed.

Automatic approaches are necessary because it is unrealistic to expect that web designers will hand-code the necessary modifications into their websites. There is a rich history of automatic approaches to supporting accessibility [15, 34, 12, 28, 35, 31, 18, 25]. In the future, we hope to impact accessibility guidelines, so that web pages and other applications can incorporate hooks that will make low bandwidth accessibility more feasible.

This work has applications outside of assistive technology. Other users with limited input capabilities include wearable computer and mobile phone users. In summary, the contribution of our work is a set of techniques for making the web more accessible in conditions where input is limited to low bandwidths and errors are likely. In contrast, past work has looked at low bandwidth or alternative output modes, with some related input issues that overlap and inform our work.

Overview

We start with an overview of related work. We then discuss low bandwidth input in more depth, giving examples of current solutions to supporting direct manipulation with limited input. Next, we give a set of requirements for low bandwidth web accessibility and discuss two separate systems we developed to address these requirements. First, we describe a modified web browser. Second, we illustrate how a proxy server can automatically apply them. An ideal interface might combine both of these solutions, but there are trade-offs in their portability. In our future work section, we discuss how user modeling can enhance these tools. We conclude with a discussion of how this work fits into our larger accessibility agenda.

BACKGROUND

A significant body of work in web page accessibility exists, including guidelines for accessibility [37], automatic evaluation tools [3], and transformations that help disabled users to access web pages [27, 31, 18, 2, 38, 9, 15, 34, 14]. However, this work was generally done with vision impairments in mind, and assumes the use of a mouse and keyboard, or at least a keyboard. Those systems that do support motor impaired users generally assume some use of a mouse or keyboard [14, 15]. In contrast, we assume unimpaired vision, but much more limited motor control.

Some of the problems solved by past work influence our own. For example, the Bobby system looks at the order in which text is read by screen readers [3]. Control of order is also important as users tab through links because it impacts the time it will take to get to different links. Kennel *et al.* insert "references" to support navigation within the current document, a technique we also use [18]. This allows a user to skip to portions of the document most interesting to him, and skip over things like long lists of links that take time to navigate through. Navigation work outside the accessibility domain has also informed our research. For example, Zellweger *et al.* and Miura *et al.* have looked at ways to give visual hints about the contents of link targets [39, 25]. This is especially important for a user who must spend significant time and effort performing error recover if he selects a link and visits a page that turns out to be uninteresting. Kaasinen *et al.* have designed tools that transform HTML for small displays [17]. For example, a single document may be split into an index page and multiple content pages (an approach also supported by Huang and Sundaresan for users with disabilities [15]). Our work also involves some similar transformations, but we have the advantage of much more screen space than the palm and cellphone-sized displays this work usually targets. Most important for us is the work in automatic transformations, a difficult problem sidestepped by most commercial systems, which are hand-coded. Systems that automatically modify visual features for low-vision and blind accessibility helped to provide the inspiration for this work [15, 34, 12, 28, 35, 31, 18, 25].

Some existing work has looked at low bandwidth input to user interfaces. For example, Colven and Lysley propose an approach to making Windows more accessible

to switch user interfaces [6]. O'Neill *et al.* present an evaluation of scanning user interfaces [30]. Our own past work proposed using logical control for low bandwidth input [26]. Finally, much work has been done in word-prediction, where limited signals are being mapped onto the very large space of the English language [1, 10, 24], or other text commands [7]. We build off of this past work but focus on the web-browsing task in particular.

LOW BANDWIDTH INPUT

This work is an investigation into low bandwidth input and the associated interface issues for web browsing. User bandwidth is a measure of the information processing rate for a human input or output channel. Bandwidth (in bits/second) is a function of task difficulty and task completion time. Difficulty is a measure of the number of bits of information being processed. If task completion time increases while difficulty remains the same, bandwidth is reduced. In other words, a user who takes longer to complete a task is said to complete it at a lower bandwidth [19]. Bandwidth is influenced by a combination of the user, application and device in use, so interface choices can affect bandwidth and hence performance of a task [5]. Additionally, error rates, which are also impacted by interface design, may increase task completion times, and thus reduce bandwidth. One factor common to very-low bandwidth input is the use of a dwell time for selection. For example, the system may follow a link only after the user selects it and then does nothing for two seconds. This further decreases bandwidth by a function of the dwell time.

With low bandwidth input, there is generally a severe mismatch between the number of input signals and the number of interface elements the user wants to control. A single switch is appropriate to control a single light in a room, however, it is not well suited to controlling an entire house of lights. An interface must multiplex a small number of input signals onto a large number of controls to support low bandwidth input. Unfortunately, most graphical user interfaces (GUIs) are designed to do the opposite: They expect a user to be able to select any of the 800x600 (or more) pixels on the screen, and then narrow this down to a smaller set of functions with the use of menus, buttons, links, and so on. There are several approaches available for addressing the mismatch between input signals and the interface, discussed next.

One powerful tool for reducing the necessary input bandwidth is *logical control*. Logical control refers to accessing the specific functions supported by an application directly. This stands in contrast to direct manipulation, where the user navigates across the screen to a menu, selects the menu, selects an item in it, and so on. A shortcut such as CTRL-S (to save a document) is an example of this. Another example is a hierarchical interface in which the user progressively chooses more specific items. Although there are trade offs here, when mouse motion is slow and difficult, logical control can be an important alternative. We will focus on logical control as a paradigm in this paper. An alternative to logical control is *physical control*, in which the low bandwidth input is being used to simulate a standard physical input device.

Even once logical control is applied, there will generally be far more elements to control than input signals. Another tool that can help handle this mismatch is *prediction*. Prediction refers to interfaces that infer what the user is trying to do in order to reduce the number of choices (and, thus, the number of inputs necessary to select them). An example used by mobile phone users is a "phone keyboard," which multiplexes the entire alphabet onto a few keys and then uses a dictionary, bigrams, *etc.* to disambiguate as the user types [11, 20]. Word prediction, mentioned above, is another example, developed originally for motor-impaired users, and also applied to support those with cognitive disabilities and users of mainstream devices and applications including PDAs and web browsers [23, 29].

Prediction and logical control reduce the number of screen elements that a user must interact with. Two other techniques, scanning and wrapping, increase the number of screen elements that a single switch can select.

Scanning interfaces move the focus of control in a grid *sequentially* and *automatically* from item to item, with a standard timeout between moves. The user needs only one switch, which she triggers to execute the currently selected item [30]. For example, the grid might contain an alphabet, and "execution" might be equivalent to typing a letter (like clicking on it on a soft keyboard). This shows how scanning can be used for logical control. A physical scanning interface might work as follows: the mouse cursor moves horizontally until the user signals a stop; this causes a click and then motion begins again.

Wrapping is used to halve the number of signals needed to control a mouse or traverse a line or table of items. In every case where the user normally needs a bidirectional signal (such as more left and more right), wrapping allows the substitution of a unidirectional signal (move right, for example). With wrapping in place, if the user reaches the edge of a line, the selection wraps to the beginning of that line. She never needs to switch directions because she will eventually reach any point she has passed again. Although this reduces the number of signals needed, it increases the average access time for any given item. Wrapping can be used with either physical or logical control as well. Wrapping differs from scanning in that the user controls all motion. The focus (in logical control), or position of the mouse (in physical

control), changes when the user produces a signal, and only when the user produces a signal.

The discussion thus far has focused on issues of layout and control. A separate topic that has an equally important impact on bandwidth is error. Errors may originate either from user difficulty in action, or from faulty interpretation of user action by the system. An example of the former is repeat switch triggering due to a spasm. An example of the latter is faulty prediction by the system.

In either case, the result is that the user must correct the error. For most users, this is generally handled by switching to a less error-prone modality [21]. However, users with input bandwidth limitations do not have this option. This means that confirmation dialogs and other ways of avoiding errors are critical, especially when the chance of error on the systems side is high. Since errors are often distributed unevenly over the set of available inputs, the mapping of signal to task can impact the number of errors that need to be corrected, and thus the input bandwidth. As a rule of thumb, less error-prone signals should be mapped to more common tasks. An in-depth example of this is given in the next section.

In this work, we focus on logical control for low bandwidth web browsing. We chose this approach because the number of interface actions is far fewer than the number of pixels on the screen and, thus, the necessary bandwidth for selecting an action is lower when the user does not have to deal with all the extra pixels. We make use of wrapping but not scanning or prediction in the prototypes discussed below. We expect to incorporate all four techniques in our work in the future.

APPROACH

There are two major approaches to increasing web accessibility. The first is to make web browsers more accessible. The second is to translate HTML into a more accessible layout. The first approach can provide better access to browser-specific functionality while the second is platform and browser independent. We will show examples of each approach, and discuss how they can be automatically supported.

For both approaches, the following requirements must be met in order to allow true web access. The minimal requirements are that:

1. The currently selected link is visible
2. The user can read and navigate text even when it contains no link
3. The user can traverse the history list forward and backward

In addition to these primary concerns, there are a number of important secondary issues:

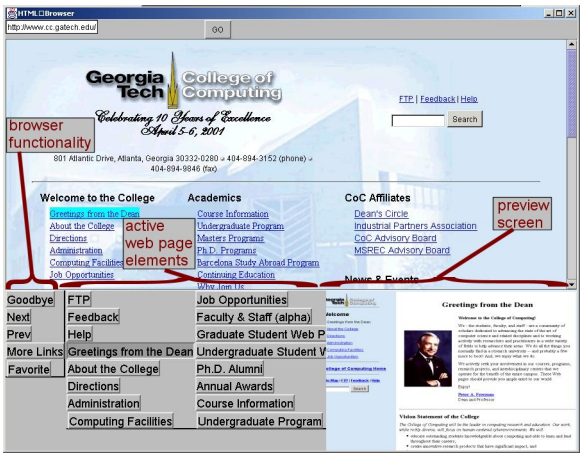

Figure 1: A web browser modified to work with four switch neural-signal users

4. The user can access her bookmarks and add to them
5. The user can go quickly to a point of interest with a minimal number of signals
6. The user is given alternatives for entering text and dealing with form elements
7. The user is given enough information about link targets to make informed decisions about whether to follow them

These requirements are complementary to those recommended by the W3C for supporting users with disabilities. The W3C suggests providing shortcut keys to access important links, ensuring a logical order for tabbing through links, and ways of dealing with image maps and scripts [37, Guideline 9], clearly identifying the target of each link (a priority 2 item that becomes priority 1 for our users), and providing navigation bars and grouping related links [37, Guideline 13].

Browser redesign

Figure 1 shows a web browser that has been designed with error-prone, low bandwidth input in mind. The main screen shows HTML, rendered by a third-party browser [13]. This is important because the user is able to view an identical page to that viewed by able bodied users, a failing of the HTML transcoding approach discussed next. The bottom half of the window has three major control areas: browser functionality (leftmost section); active web page elements (middle section); and a preview screen (thumbnail at bottom right). This browser handles most of the issues mentioned above.

This browser was designed with a particular set of users in mind. These are people with locked-in syndrome who are controlling a computer with neural signals [26]. Neural control users can modify their brain signals by in-

creasing signal characteristics such as frequency or amplitude. Using thresholding, neural control users have produced up to four signals, with a large error in differentiating them, which we will refer to as *nudge A, nudge B, shove A and shove B* [26].

In our design, the two signals with the lowest error rate are used to move the focus forward through the current control area (*e.g.* the list of URLs in the bottom center portion of Figure 1). These are *nudge A*, which moves the focus forward one item inside a control area, and *shove A*, which moves the focus forward one column in a given control area. *Shove B* is used to switch to the next major control area. *Nudge B* moves backward in the current control area. When a link is selected, the browser moves the visual focus so that link is visible in the main HTML window (the top half of the application). The history is accessible through the "Prev" and "Next" buttons, and the bookmarks through the "Favorites" button. Previews are provided for link targets.

Consider Figure 1. Suppose the user wishes to visit the "Job Opportunities" page (top of the second column in the list of URLs), and then wishes to go back to the page shown in the top half of Figure 1. She would need to produce five *nudge A*'s to get from the currently selected link ("Greetings from the Dean") to "Job Opportunities." She would then pause for two seconds, at which point the new page would be loaded in the main view area above. To go back a page, she would need to generate a *shove B*, (moving the focus to the top of the browser functionality area shown in the leftmost column) followed by two *nudge A*'s to select "Prev" After a two second pause, the previous page would appear. Any errors could make this process more complex.

Table 1 shows a confusion matrix describing the common mistakes that occur: an entry at row m, column n indicates how often the signal at row m is mistaken for the signal at column n. This table was generated based on many hours of observation of our users with locked-in syndrome. We used it to help inform our decisions regarding the mapping of signals to actions. Suppose the user is attempting a *shove B* in order to move from the URL window to the browser window. The entries in the *shove B* row indicate that an attempt to generate a shove B will commonly generate a *nudge B* or a *shove B*, but rarely a *nudge A* or *shove A* or timeout. Since *nudge B* simply changes which URL is selected, the most common error associated with *shove B* does not need to be undone, considerably reducing the impact of that error. Conversely, the user will only occasionally switch tasks when a simple URL selection is all that was intended (since a *nudge A* almost never generates an unintended *shove B*). Similarly, since our interface supports wrapping, if a user is having trouble generating the signal for "back" (*nudge B*), she may generate the forward signal instead (*nudge A*), an action that still helps her reach her goal, albeit more slowly.

The preview window also helps with errors by showing the user more information about her intended destination before she makes the choice to follow a link. This is because the cost, in number of steps, of backing out of a wrong choice is very high, requiring four separate actions, one of which is a dwell of two seconds. They are *shove B*, *nudge A*, *nudge A*, and a timeout to select. Because link following is by far the most difficult act to reverse if it is done in error, we chose to use the slower but more reliable timeout signal for this task instead of a *nudge* or a *shove*.

In summary, we redesigned a web browser to be more accessible to a specific type of low bandwidth input, neural control. The browser can be used by anyone who can generate four signals, plus pause for selection. However, considerable thought went into mapping the signals in such a way that the impact of errors would be minimized. The type of errors that occur may be very user-specific, with the result that the browser needs to be redesigned for each user, or automatically adapted to the specific requirements of each user.

This design is a first step in increased accessibility. First, it is not a complete solution because it does not address requirements two, four, and six. Second, it is only useful to a small and unique group of users. Third, it cannot track or adapt to changes that occur as users fatigue, such as the need for a longer timeout. Finally, it cannot be used with less than four available inputs.

HTML Redesign

In addition to the browser redesign presented above, we also developed a set of techniques for modifying HTML. As stated in the introduction, a browser can be controlled by a user who can generate six separate signals. However, many of our target users may not have control of six separate signals. Even for those who do, there remain some usability issues, particularly the linear time to reach URLs at the end of the page, and the lack of visibility of which link is selected.

Figure 2 shows some of the Bobby web pages [3]. Bobby is a program that checks web pages and gives advice about potential accessibility issues. The Bobby pages are designed to be accessible for people with disabilities, in line with mission of the site. On the left is the front page for the Bobby web site, on the right is one of the sub-pages, a page of frequently asked questions (a FAQ).

Figure 3 shows the same pages after they have been modified to be accessible with a two-switch interface. This is the minimal input set that can be supported if modifications are only made to HTML.

	timeout	nudge A	shove A	nudge B	shove B
timeout	common	occasional	rare	occasional	rare
nudge A	never	common	occasional	occasional	never
shove A	never	common	common	rare	occasional
nudge B	never	common	never	common	occasional
shove B	never	rare	occasional	common	common

Table 1: A confusion matrix for the four signals used to control the browser. An entry of p at mn indicates the chance that an attempt to generate the signal in row m will result in the signal in column n with a frequency of p

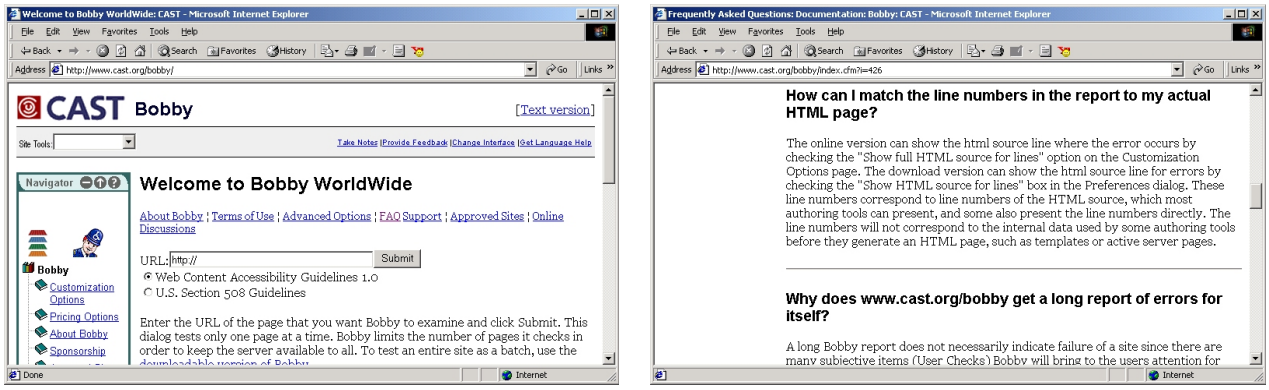

Figure 2: Sample pages from the Bobby web site [3]

One signal is used to move the focus from link to link, the other to follow a link. If a user has only one switch available, and wishes to use a timeout for selection, our modifications would have to include strategically placed "null" links where the user can safely pause to read.

Figure 3(a&b) is a case study to illustrate how the seven requirements given could be solved without modifications to a browser. Figure 3(a) shows a modified version of the Bobby front page *coded by hand* to illustrate solutions to almost all of the requirements. Figure 3(b) shows the FAQ page after it has been *automatically* modified by a proxy to include solutions to requirements one, two and three. We give details below.

1. **The currently selected link is visible** We modified the HTML to highlight each link as it is selected. This required the use of javascript.
2. **The user can able to read and navigate text that contains no links** This was only an issue for the FAQ page. We added links that support navigational control (up page and down page) at each paragraph break.
3. **The user can traverse the history list** Our solution was to add "Back" and "Forward" links to the page that call corresponding javascript functions. In Figure 3(a), we replace the less common "Forward" with "Control", which links to a page with a variety of control features including forward, bookmarks, and so on. Only forward and back are implemented in the proxy.
4. **The user can access bookmarks and add to them** This requires proxy support. Web browsers already save bookmarks in HTML form which makes access fairly simple. Adding a bookmark may be supported by putting a *file as bookmark* link at the top of every page. Selecting this sends the user to a page that allows him to select a folder, in which to place the bookmark. This is not fully implemented in the hand-coded version and not supported at all by the proxy.
5. **The user can quickly access text and links of interest** We use several techniques here. First, we add links for skipping unwanted text. Navigation bars are a good example of text that should be skippable. By adding one link, we allow the user to skip four or more links on various portions of the Bobby website. Many other sites have significantly larger navigation bars, making this approach even more beneficial. The hand-coded page (Figure 3(a)) illustrates this (the Skip button in the navigation bar).
6. **The user is given alternatives for form elements** Forms may be handled with link-based widgets if necessary. Unlike many of the previous changes, this could negatively impact usability for able bodied users. For example, our redesigned front page includes a pointer to a soft keyboard for URL entry. This shows why a proxy may be a better solution here

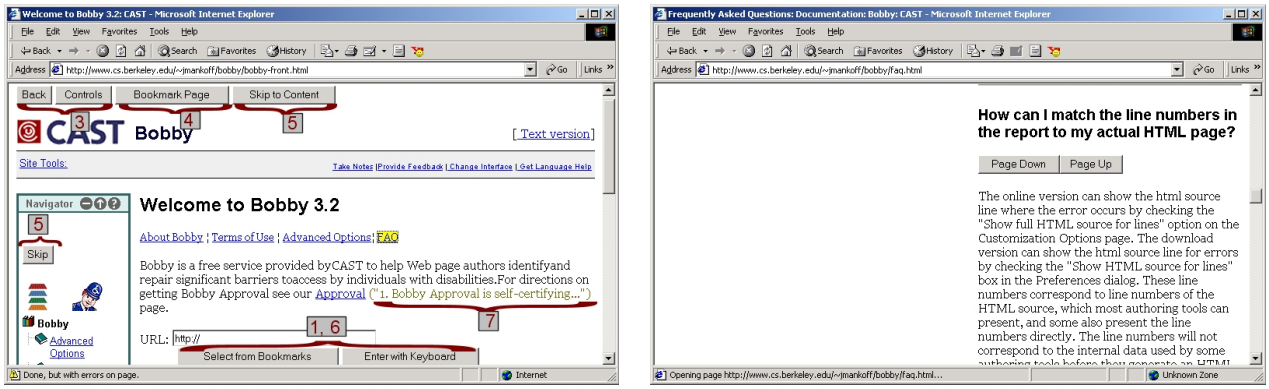

Figure 3: The same web pages modified to be accessible to low bandwidth users **(a)** The front page, hand-coded for accessibility. **(b)** The FAQ page, accessed through our proxy.

than a universal redesign. Some other possible form substitutions include a list of links instead of a menu and a reduction in free form text entry where possible. Where text is constrained (*e.g.* numeric) this can be used to reduce the number of keys presented. We have not implemented this yet.

7. **The user is given information about link targets** Links may be annotated with brief descriptions of their targets. Other researchers are investigating techniques for solving this automatically [39, 25]. It could also be facilitated by something akin to the author-provided alternate tag for images. Figure 3 (a) illustrates our solution to this, that of displaying the first few words from a target link in parentheses after the link.

FUTURE WORK AND CONCLUSIONS

Our larger goal is to develop a user model that incorporates information about fatigue, cognition, and a wider variety of input devices. The question of how to translate this model into interface modifications is nontrivial. We will approach it iteratively by designing user interfaces for specific instances and looking for useful patterns. We plan to use this model to develop a general tool in support of automatic modification of a variety of interfaces beyond just web browsers. Examples include drawing programs, spreadsheets, email, and other commonly used applications.

In order to address fatigue and error rates, we plan to model the user's capabilities as a state machine that encodes the number of available signals and the difficulty of moving between them. State machines have been used in the past to describe an input device [4] and to model errors [16]. We are combining the two approaches. This is functionally equivalent to a confusion matrix. We propose to develop a set of rules that can map this user model into interface modifications. One approach would be to model the interface as a state machine as well, and then compute the potential bandwidth of different mappings. Rudnicky and Hauptmann used a similar technique to develop visualizations that could help in identifying optional interfaces in a more constrained domain (mediation techniques) [33]. Trewin and Pain show that statistics about users (such as those in our confusion matrix) can be used to successfully adapt an interface to be more accessible [35, 36].

The resulting system will automatically adapt to different users' capabilities, including number of signals and error rates. We also plan to investigate techniques for dynamically updating our model as users fatigue.

Our plan is to develop a system that can be used by anyone with one or more signals at his or her disposal. This system will be expanded to make use of scanning and prediction, and will meet all seven requirements. Scanning will be turned on at user request. Prediction will be used to help the user when filling out forms. The expanded browser will also handle frames and other common HTML usages robustly.

In conclusion, we have shown two approaches to increase web accessibility in the face of low bandwidth input and demonstrated how they can address a host of problems. In the future, we hope to combine these fairly synergistic approaches (browser and HTML redesign).

Acknowledgements

Thanks to Susanne Jul for her characterization of direct manipulation as navigating through individual pixels on the screen. This work was supported by NSF award IIS-0118917.

REFERENCES

1. J. L. Arnott et al. Prediction and conversational momentum in an augmentative communication system. *CACM*, 35(5):46–57, May 1992.
2. C. Asakaw and A. Itoh. User interface of a home page reader. In *Proc. of ASSETS'98*, pp. 149–156, 1998.
3. Center for applied special technology (CAST), Bobby

service. Web Page. Available at: http://www.cast.org/bobby/.

4. W. A. S. Buxton. A three-state model of graphical input. In *Proc. of INTERACT'90*, pp. 449–456, 1990.
5. S. K. Card et al. A morphological analysis of the design space of input devices. *ACM TOIS*, 9(2):99, 1991.
6. D. Colven and A. Lysley. Designing and using efficient interfaces for switch accessibility. In *Proc. of the 6th ERCIM Workshop on 'User Interfaces for All'*, p. 2, 2000.
7. J. Darragh and I. Witten. The reactive keyboard. *International Journal of Man-Machine Studies*, 39(3):521–528, 1993.
8. Developments in aging: 1981. U.S. Printing Office, U.S. Senate Special Committee on Aging, 1982.
9. T. Ebina et al. Fast web by using updated content extraction and a bookmark facility. In *Proc. of ASSETS'00*, pp. 64–71. ACM, 2000.
10. N. Garay-Vitoria and J. Gonzalez-Abascal. Intelligent word-prediction to enhance text input rate. In *Proc. of IUI'97*, pp. 241–244, 1997.
11. D. L. Grover et al. Reduced keyboard disambiguating computer. Patent No. US5818437, 1998. Tegic Communications, Inc., Seattle, WA.
12. V. L. Hanson et al. Transcoding web pages for users with vision disabilities. In *Conference and Workshop on Assistive Technologies for Vision and Hearing Impairment: Support Technologies for Independent Living and Work*, August 2001.
13. H. Heistermann. The webwindow home page. Product Web Page. Available at: http://home.earthlink.net/~hheister.
14. D. Hermsdorf. Webadapter: A prototype of a WWW browser with new special needs adaptations. In *Proc. of ICCHP 98*, pp. 151–160, 1998.
15. A. W. Huang and N. Sundaresan. A semantic transcoding system to adapt web services for users with disabilities. In *Proc. of ASSETS'00*, pp. 156–163, 2000.
16. S. E. Hudson and G. L. Newell. Probabilistic state machines: Dialog management for inputs with uncertainty. In *Proc. of UIST'92*, pp. 199–208, 1992.
17. E. Kaasinen et al. Two approaches to bringing internet services to WAP devices. In *Proc. of the WWW9*, 2000.
18. A. Kennel et al. WAB: W3-access for blind and visually impaired computer users. *SIGCAPH Bulletin*, June 1996.
19. I. S. MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7(1):91–139, 1992.
20. I. S. MacKenzie et al. Letterwise: Prefix-based disambiguation for mobile text input. In *Proc. of UIST'01*, 2001.
21. J. Mankoff et al. Interaction techniques for ambiguity resolution in recognition-based interfaces. In *Proc. of UIST'00*, pp. 11–20. ACM Press, November 2000.
22. W. C. Mann and J. P. Lane. *Assistive Technology for Persons with Disabilities*. The American Occupational Therapy Association, Inc., 2nd edition, 1995.
23. T. Masui. An efficient text input method for pen-based computers. In *Proc. of CHI'98*, pp. 328–335, 1998.
24. A. McKinlay et al. Augmentative and alternative communication: The role of broadband telecommunications. *IEEE Transactions on Rehabilitation Engineering*, 3(3), September 1995.
25. M. Miura et al. inlineLink: Inline expansion link methods in hypertext browsing. In *Proc. of the International Conference on Internet Computing (IC 2001)*, pp. 653–659, 2001.
26. M. Moore et al. Nudge and shove: Frequency thresholding for navigation in direct brain-computer interfaces. In *Proc. of CHI'01*, pp. 114–120, April 2001.
27. S. Morley et al. Auditory navigation in hyperspace: Design and evaluation of a non-visual hypermedia system for blind users. In *Proc. of ASSETS'98*, pp. 100–107, 1998.
28. E. D. Mynatt and W. K. Edwards. Mapping GUIs to auditory interfaces. In *Proc. of UIST'92*, pp. 61–70, 1992.
29. Netscape communications corporation. Product Web Page. Available at: http://www.netscape.com.
30. P. O'Neill et al. Evaluation of scanning user interfaces using real-time-data usage logs. In *Proc. of ASSETS'00*, pp. 137–141, 2000.
31. T. Oogane and C. Asakawa. An interactive method for accessing tables in HTML. In *Proc. of ASSETS'98*, pp. 126–128, 1998.
32. M. Rowan et al. Evaluating web resources for disability access. In *Proc. of ASSETS'00*, pp. 80–84, 2000.
33. A. I. Rudnicky and A. G. Hauptmann. Models for evaluating interaction protocols in speech recognition. In *Proc. of CHI'91*, pp. 285–291, 1991.
34. H. Takagi and C. Asakawa. Transcoding proxy for nonvisual web access. In *Proc. of ASSETS'00*, pp. 164–171, 2000.
35. S. Trewin and H. Pain. Dynamic modelling of keyboard skills: Supporting users with motor disabilities. In *Proc. of the 6th International Conference on User Modeling (UM-97)*, pp. 135–146, 1997.
36. S. Trewin and H. Pain. A model of keyboard configuration requirements. *Behaviour and Information Technology*, 18(1):27–35, 1999.
37. Web access initiative (WAI): World wide web consortium (W3C). Consortium Web Page. Available at http://www.w3.org/WAI/ and http://www.w3.org/TR/WAI-WEBCONTENT.
38. M. Zajicek et al. A web navigation tool for the blind. In *Proc. of ASSETS'98*, pp. 204–206, 1998.
39. P. T. Zellweger et al. Fluid links for informed and incremental link transitions. In *Proc. of Hypertext'98*, pp. 50–57, 1998.