KeySurf: A Character Controlled Browser for People with Physical Disabilities

Leo Spalteholz
Department of Electrical and Computer Engineering
University of Victoria
Victoria, BC, Canada
V8W 3P6
leos@ece.uvic.ca

Kin Fun Li
Department of Electrical and Computer Engineering
University of Victoria
Victoria, BC, Canada
V8W 3P6
kinli@ece.uvic.ca

Nigel Livingston
CanAssist
University of Victoria
Victoria, BC, Canada
V8W 3P6
njl@uvic.ca

Foad Hamidi
CanAssist
University of Victoria
Victoria, BC, Canada
V8W 3P6
foad@canassist.ca

ABSTRACT

For many users with a physical or motor disability, using a computer mouse or other pointing device to navigate the web is cumbersome or impossible due to problems with pointing accuracy. At the same time, web accessibility using a keyboard in major browsers is rudimentary, requiring many key presses to select links or other elements. We introduce KeySurf, a character controlled web navigation system which addresses this situation by presenting an interface which allows a user to activate any web page element with only two or three keystrokes. Through an implementation of a user-centric incremental search algorithm, elements are matched according to user expectation as characters are entered into the interface. We show how our interface can be integrated with a speech recognition input, as well as with specialized on-screen keyboards for people with disabilities. Using the user's browsing history, we improve the efficiency of the selection process and find potentially interesting page links for the user within the current web page. We present the results from a pilot study evaluating the performance of various components of our system.

Categories and Subject Descriptors

H.5.2 [**Information interfaces and presentation**]: [User Interfaces – Input devices and strategies.]; K.4.2 [**Computers and society**]: [Social issues – Assistive technologies for persons with disabilities.]

General Terms

Human Factors

Keywords

Web Accessibility, Keyboard Access, Web Navigation

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

1. INTRODUCTION

For people with physical or motor disabilities, using a pointing device accurately can be difficult or impossible. As modern graphical user interfaces (GUIs) are generally mouse driven, much work has been done to facilitate accessibility via the keyboard for those users unable or unwilling to use a pointing device. In general, this has resulted in modern GUI applications being efficiently accessible for both keyboard and mouse users. However, with the proliferation of the web, more and more time is spent accessing web pages and web applications via a web browser, instead of using local applications.

For users without the option of using a pointing device, manipulating the interface of web pages with a keyboard is often very cumbersome. Although guidelines for web accessibility – which include best practices for keyboard accessible design – have been developed by the Web Accessibility Initiative [18], adoption rates amongst web authors are still poor [8]. Currently, an alternative web navigation system is only practically useful if it is compatible with an overwhelming majority of popular websites, regardless of their conformance to accessibility standards.

In previous work, we proposed a web navigation system that laid the framework for efficient navigation on the World Wide Web for any users able to use a keyboard or equivalent device [15, 16]. This system, termed KeySurf, is based on the concept of using a layered, incremental search mechanism to select links and other clickable elements on a web page. KeySurf generates a suitable textual description for each clickable element and allows users to select these elements by entering several characters with any keyboard-equivalent input device. In this paper, we refine our incremental search selection method, propose extensions that help users discover potentially interesting content, and make interesting content easier to select.

2. BACKGROUND

In addition to our work, the problem of navigating the web for people with physical disabilities has been approached

from various angles. Schrepp surveyed the accessibility of current websites for both mouse and keyboard users [14], with the conclusion that while the state of website design for mouse users was quite good, the majority of websites are very cumbersome to navigate for keyboard users (using the Tab key to move through links). For popular online resources, even able-bodied keyboard users (simulated with a GOMS model) required between 4 to 10 times as much time to complete navigation tasks than mouse users.

The poor state of keyboard accessibility in major browsers and most websites has prompted several implementations of keyboard navigation systems based on automatically assigned keyboard shortcuts. For Firefox, the extensions Hit-a-Hint [9] and Conkeror [7] are examples of such a technique. With these systems, the user can press a key to bring up a small numbered label beside every clickable element on the page. By typing in the number code of the desired element and pressing Enter, the system "clicks" on that element, thus following the link or activating the button.

While these types of number code systems are quite effective, we believe the design imposes some limits on efficiency and adds an extra cognitive step to the navigation process. If the shortcut labels are not shown on page load, an extra key press is required to show them; while if they are shown on page load, the extra elements cause visual clutter and occlusion problems when there are many labels. In addition, there is no semantic connection between the keyboard shortcut and the clickable element (for instance, the shortcut for a link labeled "Sports News" would be something like 42). A user must bring up the labels, make the connection between the numeric shortcut and the element, and then type in the code. To the best of our knowledge, shortcuts are assigned sequentially with no consideration for user history or the relative importance of clickable elements.

Trewin et al. have approached the problem from a different perspective. Instead of attempting to provide a keyboard navigation system for web users with disabilities, they proposed a method of steadying the mouse cursor such that users are able to more accurately hit targets with the pointer [17]. Bilmes at al. have provided an alternative to mechanical cursor control entirely, by proposing a Vocal Joystick to allow users to control a mouse pointer with continuous voice control [1].

3. TARGET USERS

The KeySurf system is primarily designed for users with a physical disability that have trouble accurately using a pointing device (such as a mouse, joystick, or trackball) or are not able to use a pointing device at all. More generally, the system is suitable for any user who can type two or three characters (with a keyboard or equivalent input device) faster than they could acquire a small target (such as a hypertext link or form button) with a pointing device. For convenience, we refer to the process of entering characters into the KeySurf interface as *typing* in this paper. However, the interface can be controlled by any input device capable of textual output at some stage, be it a regular computer keyboard, Morse code from a single switch device, any specialized hardware or software for text input, or a speech-based interface as discussed in Section 4.2.1.

Although the KeySurf system is primarily designed for disabled users who are unable to accurately use a pointing device, it can also be beneficial to non-disabled users who are faster or more comfortable with keyboard input than mouse. With the proliferation of mobile devices and public wireless access points, it is often impractical to connect a computer mouse to a laptop or hand-held device, potentially making an efficient keyboard web navigation system very useful.

4. SYSTEM OVERVIEW

The relationship between the major components of the KeySurf system are depicted in Figure 1. KeySurf components can be logically classified as either part of the mechanism underlying link selection, or comprising the navigational support offered by the knowledge of user interests. The visible user interface is minimal, thereby maximizing available space for website display. KeySurf is independent of the input device in the sense that it can be controlled by any input device that is capable of producing character output in some way. If information about which characters are easiest to produce with the given input device is available (the Input Device Efficiency Model), it can be used to generate more efficient labels for unlabeled elements on a page.

The link selection mechanism relies on web page elements from the current page being extracted, assigned a relative priority for the current page, and unlabeled elements being given a label before selection can be commenced. Once characters are entered into the browser, the User Centric Search module (Section 4.1) determines which element is most likely to be that which the user intended, based on the characteristics of the matching elements. Matches are marked by the Match Highlighter module, with a green highlight for the default match, and yellow for other possible matches.

As pages are loaded, the User Browsing History component keeps track of user browsing activity such as page viewing time and key terms from web pages (Section 5.1.1). More explicit indicators of user interest such as a user's search terms or pre-seeded interest keywords are taken into account as well (Section 5.1.3). Web browsing history is used to calculate the interest score corresponding to keywords from extracted pages, which is used in conjunction with the elements on a loaded page to suggest potentially interesting links to users on long pages (Section 5).

4.1 User Centric Search

The concept of using incremental search to find text or select links on a page is not new. For example, this type of search is implemented in the Mozilla family of browsers under the name of "Find As You Type" (FAYT). As the user types a character, the system searches the page starting from the top and focus the first text link that contains that letter. The match is continuously updated as more characters are typed. While this represents a marked improvement over simply iterating through links linearly (which is common in other major browsers), the matched links are often not what a user expects. Since the FAYT algorithm does not take the location of the matched substring into account, the first match is often found somewhere inside a word of link text, resulting in matches that do not necessarily correspond well with the user's expectations. By default, links are also searched starting from the top of the page, resulting in a jarring transition for users as the web page view jumps to off-screen matches.

To improve on this approach, we have implemented a series of algorithmic constraints on the incremental search al-

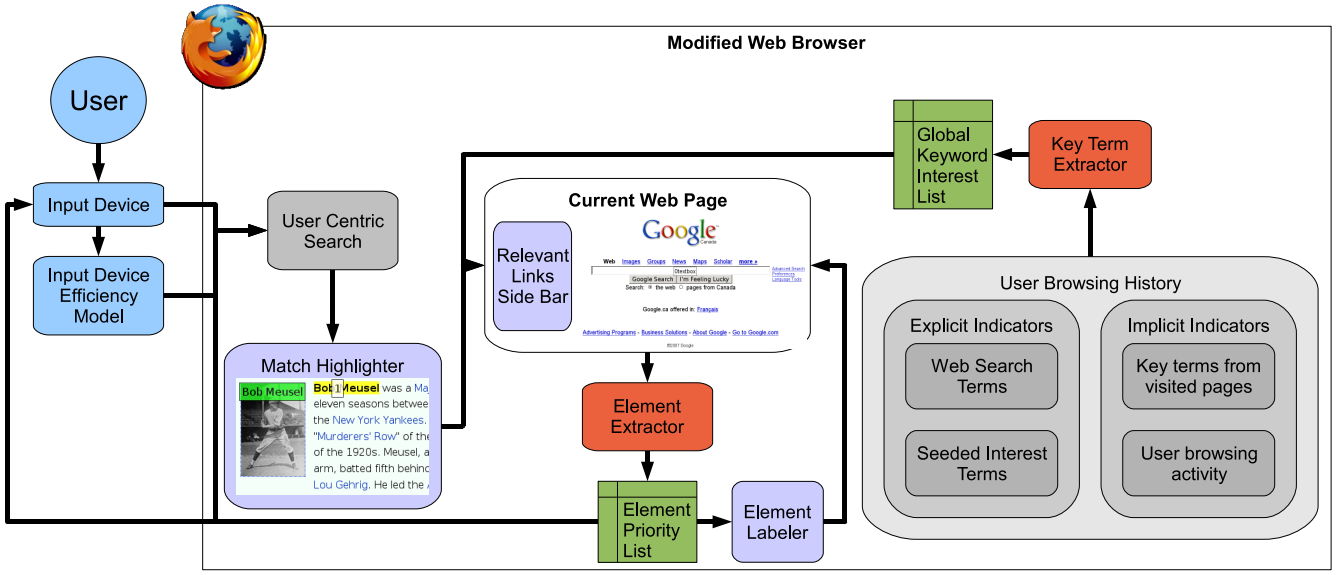

Figure 1: KeySurf system overview.

gorithm to more accurately match the user's expectation of the link that should be selected given a certain query. These constraints are best summarized as a set of relations between which types of matches are prioritized over others. Note that all following examples represent possible matches given that the user has typed an upper case "S".

1. Currently visible links before off-screen ones.
2. Visually prominent links before subtle ones. Web authors often indicate important links with a larger font or emphasized text.

   ex. **Search** before Sports.
3. Same case matches before case insensitive. We assume the upper case letter was deliberately chosen, since upper case letters require additional effort to produce with most input devices.

   ex. Sybase before systematic.
4. Starting characters of link before others. Left to right reading order suggests that the beginning of the link text is the most logical location to start typing letters when starting to select a link.

   ex. Sports News before Download SDK.
5. Starting characters of words before other substrings. Word boundaries present a more logical starting point to begin typing than positions within words.

   ex. Download SDK before Downloads.

Basic selection is accomplished using only these precedent rules. If two or more elements cannot be separated by typing two letters, we provide a shortcut method to uniquely select them with fewer keystrokes. For example, if the page contains the links Download SDK and Download Sudoku, typing "d" will highlight the first in green, while the second link is highlighted yellow and the number "1" is overlaid in a translucent box. Figure 2 shows an example of the three possible types of element highlighting on a web page. With only the rules previously mentioned, a user would have to type "download su" to select the second link, which is clearly not acceptable for users with low bandwidth key input.

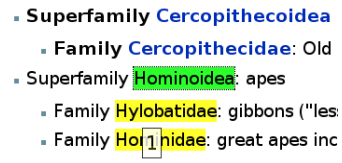

Figure 2: Highlighted elements after typing "h".

To select the second element with fewer inputs, the user may either type the number one (1), or press the down arrow key after they have typed "d". These shortcuts were added primarily for users with low bandwidth keyboard input, as it establishes an upper bound of two keystrokes to select any element (providing that the visible section of the page does not contain more than 11 links with the same first two characters, which is a valid assumption in most cases). Users capable of faster character input retain the option of typing just the letters of their desired link, which requires less visual feedback (no processing of number overlays) at the cost of some extra keystrokes. In the example given above, an experienced user could also type "su" to select the second link (due to constraint number 5).

4.2 Integration With External Input Devices

With prior knowledge of all possible selectable actions on a page, it is possible to improve the performance of certain types of input devices by constraining their set of possible outputs, and providing information about the relative priority of characters. Although the additional information provided by the KeySurf system can be used in any virtual keyboard to improve word completion, this type of integration has the most benefit in ambiguous layout virtual keyboards, which seek to improve typing efficiency by dynamically adjusting their layout based on the probability of

possible upcoming letters. These types of keyboards aim to improve typing performance by making likely letters easy to type. The Dasher project [19] and COGAIN's Gazetalk [10], as well as CanAssist's Dynamic Keyboard [3] are examples of such applications.

To facilitate the integration with input devices, our system exposes the labels for all elements extracted from the current web page, as well as their corresponding relative priority. Any virtual keyboard or other dynamic input device may access these data to improve performance for the task of typing characters to select an element. As a proof of concept, we have begun implementing two interfaces using these data: one using a speech recognition engine (Section 4.2.1), and one using CanAssist's Dynamic Keyboard (depicted in Figure 3).

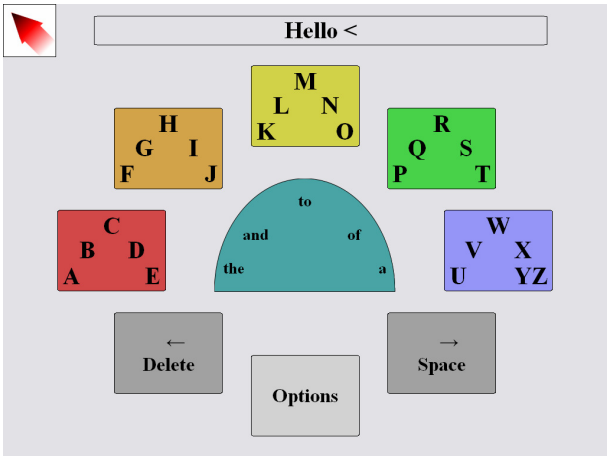

Figure 3: The interface of CanAssist's Dynamic Keyboard.

The CanAssist Dynamic Keyboard presents only five large buttons containing the letters of the English alphabet, which enables users to easily click buttons even with a very inaccurate pointing device (such as a head mouse or eye tracking system). During normal operation, the Dynamic Keyboard uses a statistical language model to filter the available letters, and move the most likely letters into their own buttons (such that they can be selected with one click). This rearrangement can decrease the required pointer movement and number of clicks per character. When the Dynamic Keyboard is used to control KeySurf, the standard language model is replaced by a language model generated from the exposed website text and priorities. As there are far fewer distinct letter sequences on a web page compared to the entire English language, the probability of likely letters having their own button is much higher than with a complete language model, which reduces the required number of inputs to produce characters.

4.2.1 Integration With Speech Recognition

For users unable to use a keyboard or pointing device but able to verbalize their intentions, an alternative is the use of a voice controlled interface. Although consumer speech recognition tools have improved dramatically in recent years, recognition accuracy for dictation is still an issue if the engine is not adequately trained or users have difficulty with clear pronunciation. However, it is clear that speech recognition accuracy and speed can be improved by restricting the recognition vocabulary to small sets and minimizing the word length of commands [2, 12]. Given our objective of selecting page elements with KeySurf, we constrain the recognition set to only those words appearing in the link text and generated labels on the current page. Thus, a user can directly select a link or other element by speaking words in its name. Several similar interfaces have been developed in the past, beginning with speech integration for the Mosaic browser over ten years ago [11].

While the active vocabulary generated from each page is small, any word in the page's language can occur in this set, which dictates that users be able to clearly pronounce a very wide range of words. In a study by Christian et al. comparing the performance of a direct selection voice browser with mouse interaction for non-disabled users, few recognition errors were encountered [4]. However, the tested pages contained less than five visible links at a time, while modern web pages being displayed on high resolution screens can present dozens of selectable elements. In addition, words appearing in link text must be recognizable by the speech engine, which is often not the case for proper nouns or abbreviations. These limitations make direct selection systems unsuitable for users with disabilities who have difficulty with clear pronunciation, or who can only pronounce a small set of words with sufficient clarity to be recognized by speech recognition engines.

An alternate voice interface is simple voice spelling. As selecting any element with the KeySurf system is designed to require only two or three characters, recognizing a whole word is more than is required to uniquely select an element. To represent the letters of the English alphabet, we use a slightly modified version of the NATO Phonetic Alphabet (Alpha, Bravo, etc.) which allows much more robust recognition than using the letters directly. This initial alphabet serves as a good starting set, but letters can be mapped to any word that is distinct from other words in the set and that a user is able to pronounce clearly. The addition of numbers and some common browser commands gives a final vocabulary size of 47 words. We use the CMU Sphinx speech recognition engine [6] to recognize user commands and convert them to the appropriate character. These characters are then transmitted to KeySurf, where they are treated the same as any other text input.

We conducted a small scale user study involving 32 users of varying age (20 to 60 years), gender and ethnic background to assess the accuracy of the untrained Sphinx speech recognition engine on our set of words. Users achieved an accuracy of 94% on the first attempt. We anticipate that even higher recognition rates can be achieved with more advanced speech engines. This shows that the simplified spelling interaction combined with the KeySurf system has the potential to be a very robust yet efficient voice browsing alternative.

4.3 Accessing Bookmarks

For the most part, the major web browsers provide adequate keyboard shortcuts for common web browser actions (Back/Forward, Bookmark page, etc). For these common actions, no additional support is necessary to facilitate use by people with disabilities. Accessing bookmarks, however, is usually quite cumbersome, with users having to manually scroll through each bookmark entry in the menu to locate their desired bookmark (with a bidirectional menu contain-

ing n bookmarks, activating a bookmark requires on average at least $n/4 + 3$ keystrokes, depending on the exact organization of items in the menu). Accessing even a moderate number of bookmarks can be slow.

To address this problem, we have extended our link selection process to also select from bookmarks. Since bookmarks can be represented as simple textual links, the same selection process applies as to that for web pages. If the user activates the "Bookmarks Mode" (via the "." key), the system renders all of the user's bookmarks on a single page, grouped by folder. Individual bookmarks can then be selected by typing letters as normal. This system places an upper limit on the number of required keystrokes to activate a given bookmarked site to four keys (one key to show the bookmarks, and a maximum of three to activate any link, providing that all bookmarks can be rendered on a single page).

5. BRINGING INTERESTING PAGES CLOSER

Scrolling through long pages to find interesting content requires many repeated key activations, which can represent significant effort for users with low bandwidth input. To go beyond a navigation system for visible content, we have developed an addition to the KeySurf system to help the user find interesting content on long pages. After a page loads, we use our estimation of user interest (Section 5.1) to suggest a number of links that may be interesting to the user. If interesting links are found on the page, they are displayed beside the loaded web page in a list. These links serve to highlight sections of pages that may be of interest to users, thus potentially saving time when searching a page for relevant information. In addition, the links are numbered to allow fast access using the same selection process as for regular elements on the main page. Links in the side bar are displayed with a small amount of surrounding text to assist users in assessing the context of the link and making a decision about whether to select it. A screen capture of the browsing interface showing the suggested links bar is depicted in Figure 4.

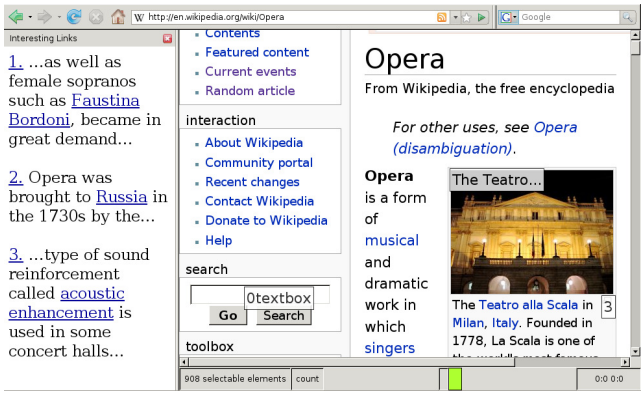

Figure 4: KeySurf interface showing suggestion bar.

For each word in each link on the page, the system queries the keyword list to find a match (Section 5.1.2). If a word cannot be located in the keyword list, it is stemmed (using a JavaScript implementation of the Porter stemming algorithm [13]) and the lookup is repeated. The score of each word in the set defined by the intersection of the current link text (L) and the keyword list (K) is summed to produce the link score $LS = \Sigma KS(k_i)$ where $KS(k_i)$ is the keyword score for each keyword $k_i \in L \cap K$. For performance reasons, we only check the highest priority elements in the keyword list when calculating the keyword score for link words on the page. The link interest scores are used to determine interesting links in the suggestion bar.

5.1 Inferring User Interest

The browsing enhancement features of the KeySurf system rely heavily on having a good estimate of the user's interests. To determine what may be of interest to the user, we look at various aspects of a user's browsing history, such as pages visited, page activity, and topics searched. These interest indicators are discussed in detail in the following sections. Logging these interest indicators and computing the estimated user interest scores has minimal impact on browsing performance, as processing is performed after a page has loaded and the user is not actively interacting with KeySurf.

5.1.1 Page Interest Score

As other authors have noted, while it may be more accurate to explicitly query the user about their degree of interest in visited pages or general topics, the rating process is too intrusive and time consuming for most users. Especially when considering the target population, where any selection requires significant effort, explicit ratings are not feasible. To address this problem, there has been significant prior work in the area of implicitly determining user interest based on a user's browsing history. Previous approaches have attempted to find a correlation between a user's interests and various factors such as page viewing time, scrolling time, mouse clicks, and related activity (bookmarking, printing). In a user study comparing explicit ratings with observed factors, Claypool et al. concluded that time spent viewing a page and the total time spent scrolling provide the best indicators of interest in that page [5].

Since one of the goals of our system is to reduce scrolling time by bringing interesting links to the top of a page, measurements of scrolling time may no longer accurately reflect user interest. Thus, page interest scores are calculated based on the maximum amount of time spent viewing a page in the user's web history. Web page viewing times do not accumulate if a user revisits the same page at a later date. This prevents often loaded pages (such as the user's home page) from dominating the page viewing times, even though each session may only last a few seconds. Additionally, pages that are visited repeatedly often contain very dynamic content (news sites present a prime example), such that key terms may change at every visit, and the page viewing time for the previous visit cannot be meaningfully added to the current viewing time.

We define page viewing time as the time elapsed between when a page is rendered by the browser and the time just before the page is hidden from view (either by navigating to a new page or closing the browser). By only measuring time after a page is rendered, we exclude the variable loading time due to network congestion and page size. To detect when a user is actively viewing a page (versus being distracted by external events), we keep track of input activity (mouse movements, key presses, and mouse clicks) while a web page

is open. If there are no input events for 2 minutes, the page timer is paused until another input event occurs. To calculate page score PS, we normalize page viewing times by defining a maximum viewing time of 20 minutes as maximum interest, and scaling times to this maximum to obtain page scores between 0 and 1.

5.1.2 Web Page Keyword Ranking

The keyword extraction and scoring algorithm is executed on every web page visited by the user, starting from the time the page is initially displayed by the browser. The algorithm steps are as follows:

1. Web page loads and is displayed to the user.
2. System begins recording page viewing time (See Section 5.1.1).
3. Web page is converted to plain text format, and processed by a keyword extraction algorithm. To extract keywords from the document, we make use of the Term Extraction Service, part of Yahoo! Search Web Services [20]. This allows us to send plain text documents to the web service using XMLHttpRequest and retrieve a set of key terms. Although details of the algorithm are not available, the use of the Term Extraction Service has the benefit of not requiring complex term extraction algorithms and supporting data to be implemented on the user's computer. Users in need of assistive technology often do not have the resources for high performance computing hardware, and minimizing the computational requirements of our system is an important factor in its utility. In the future we may implement a keyword extraction component on our own servers to realize more control over the extraction process.
4. Web page is closed (by navigating to another page).
5. Page score PS is calculated from the recorded user activity on the page.
6. Keywords are assigned a normalized weight based on their relative importance in the page ($W_1..W_n$ for the n keywords on the page). Relative importance is determined by the keyword extraction process. Currently we weight keywords from highest to lowest in the order they are returned from the Term Extraction Service.
7. The keyword scores in the keyword list are updated as follows:
   - For each keyword that appeared on the current page, a keyword score KS_i is calculated from the current page score PS and the keyword's relative importance W_i as follows:

     $$KS_i = PS * W_i$$

     Keyword scores are added to the keyword list using the incremental mean formula:

     $$MKS(n+1) = KS_i + \frac{n * MKS(n)}{n+1} \quad (1)$$

     Where $MKS(n)$ (Mean Keyword Score) is the value previously stored in the keyword list and n is the number of pages where this keyword has occurred. Note that if the keyword does not exist in the list, the second term will be zero and the keyword will be added with an initial score of KS_i. The value of n is incremented in the list for each of these keywords.
   - If a keyword on the list is not present on the current page, its score in the list is reduced by a constant aging factor A, where $0.9 < A < 1.0$. The aging factor is introduced to bias the list of keywords to more recent user interests, and to ensure that isolated high interest keywords do not dominate the top of the list. This factor is chosen empirically to balance prioritizing newer interests with maintaining long term interests in the list.

5.1.3 User Searches

A more direct indicator of user interest is search terms entered into search engines or the search fields of other websites. As we detect these search terms, they are processed by the same keyword extraction algorithm as discussed in Section 5.1.1. We treat these keywords similarly to those extracted from visited pages. As there is no equivalent to the page score for terms extracted from search fields, we add these terms to the list with an empirically determined initial score (close to 1).

The key to making effective use of data entered into web forms is to differentiate between search fields and other text input fields, such as those used for web based email interfaces or other personal information fields commonly found on web pages. To detect search fields, we compare the text entered into text fields with the GET parameters of the next loaded page. If a match is found, we assume that the text was a search term and process it as an indicator of user interest. Although this simple method cannot detect all search fields (some custom search fields use POST to submit terms), all major search engines are supported.

6. EVALUATION

To evaluate the performance and usability of our web navigation system, we conducted testing to evaluate the quantitative navigation efficiency of the selection mechanism, as well as the usability for users with disabilities.

6.1 Average Navigation Efficiency

Using the selection shortcut of the user centric search method (Section 4.1), we determined the upper limit of the keys required to uniquely select any element to be two (three to activate). However, the actual number of keys necessary on typical web pages depends on the distribution and text of visible elements. As the number of visible elements increases, the probability of the first character of a desired link being unique amongst the starting characters of all visible links on the page decreases. Due to the default selection that allows the activation of the (green highlighted) link even if there are other matches, if there are n visible links starting with the same character, then one link will be uniquely selectable with one keystroke, while $n-1$ elements will require two keystrokes (assuming that not more than 11 links start with the same two characters). To find the average number of required keystrokes, we implemented a modified version of the KeySurf system that automatically loads randomly selected web pages, and calculates the average keystroke cost for each visible element on that page.

The system analyzed a set of 726 unique web pages, chosen randomly by the system by following links from several root sites (various popular web portals). Results are shown in Figure 5.

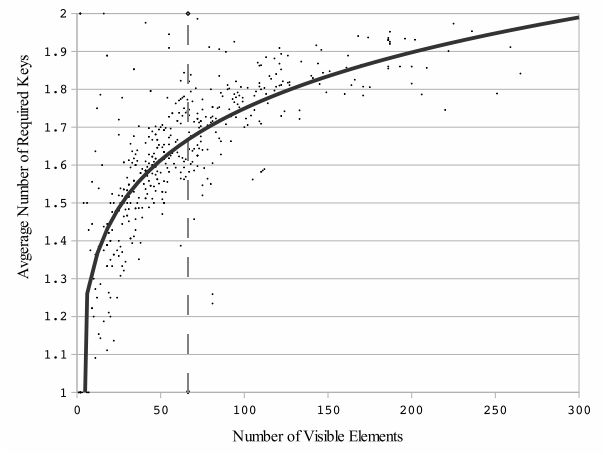

Figure 5: Average required keys per element selection.

The average number of visible elements (elements visible on page load) was 66.4, with each element requiring on average 1.63 keys to select. Activating an element after it has been selected adds one key, giving an average of 2.63 keys required to follow a link or activate another web page element. Performance with page sequences from real user sessions is expected to improve further over these results, as KeySurf attempts to prioritize elements that the user is likely to select.

6.2 Usability for People With Disabilities

To determine the usability of the system for those with disabilities, we conducted preliminary testing to gather evidence on how KeySurf performs for some representatives of the target audience. This initial testing is aimed to compare our system to the browsing method that study participants are most familiar with. At this point we do not aim to compare KeySurf to other keyboard navigation systems, as none of the participants were using such systems, and we wanted to initially determine if KeySurf would be an improvement over their usual browsing method. Participants are described in Section 6.2.1, the experimental setup is detailed in Section 6.2.2, and results are presented and discussed in Section 6.2.3.

6.2.1 Participants

We conducted our test on four persons with Cerebral Palsy. Participant age varied from 17 to 35, with all participants having experience using computers and navigating on the web. Three participants used pointer based input devices to navigate the web at home, while one used a regular keyboard. One participant was deaf and non-verbal but was able to communicate by using sign language and reading lips. One other participant was also non-verbal and communicated with a text to speech assistive device.

The test consisted of participants performing a web navigation task with a pointer device using a regular browser, and a keyboard device using KeySurf. Table 1 shows the input devices used during the test by each user. Both the pointer and keyboard devices were chosen individually for each user to match the device they used at home as closely as possible.

Subject	User's Accustomed Device	Keyboard Device
A	Penny Giles Trackball	Regular keyboard
B	InfoGrip Joystick Plus	Intellikeys USB
C	Penny Giles Trackball	Regular keyboard
D	Regular keyboard	Regular keyboard

Table 1: Input devices used by test subjects.

Subjects A and C used a large Penny Giles track ball designed specifically for people with disabilities to control the mouse cursor. Although pointing accuracy was a problem for subject A due to difficulty with fine motor control, subject C was quite accurate at the expense of movement speed. Subject B used a joystick to control the mouse cursor with good accuracy and speed. Subject D was not able to use any pointing device and used a keyboard for both tasks.

For interfacing with KeySurf, subjects A and C used a regular computer keyboard, on which they typed letters with one finger, while subject B used a larger, pressure sensitive keyboard designed for people with disabilities (Intellikeys USB). The plastic key guard for this custom keyboard which subject B was accustomed to was not available during testing, which slightly impaired his/her typing performance. Subject D used a regular computer keyboard for both tests, using his/her mouth to press keys. With an unmodified browser, subject D presses the Tab key to advance the element focus on web pages.

6.2.2 Experimental Design

We designed tests to measure the time necessary to select a visible link using each subject's accustomed input device and compared it to the time required using KeySurf with a character based input device (such as a keyboard). We picked two sets of 12 Wikipedia articles with similar layout such that one can navigate from page 1 to page 12 in each set by following links. The two sets were chosen such that the spatial location of each link leading to the next page was similar between sets, but different in successive pages. In other words, if L_{Ai} represents the location of the target link in the i^{th} page of set A, then $L_{Ai} \approx L_{Bi}$ and $L_{Ai} \neq L_{Ai+1}$. Page content was not considered in our choice of articles.

Prior to commencing the test, operation of our system was verbally explained to users. As some subjects have communication difficulties, the time required for this was variable, but did not exceed five minutes. The steps required to select a link with KeySurf were explained as follows:

1. Type the first letter of your link.
2. If your link turns green, press "Enter" to activate it.
3. If your link turns yellow with a number beside it, type the number and press "Enter".
4. If your link turns yellow without a number, type the next letter in the link and press "Enter".

Users were given time to practice selecting links on several Wikipedia pages of their choice with both input methods

until they felt comfortable with both. Two random subjects were assigned to start with KeySurf, while the other two started with their usual method. Times for the selection process on each page were recorded separately.

To test selection performance, the first page in a set was loaded and the desired link was pointed out to the user (by physically pointing at the link on the screen). At the same time, a timer was started which measured the time the user required to follow the link. The procedure was repeated on all pages in the set and again using the other selection method on the other set of pages.

6.2.3 Results and Discussion

Experimental results for each user are presented in Figure 6.

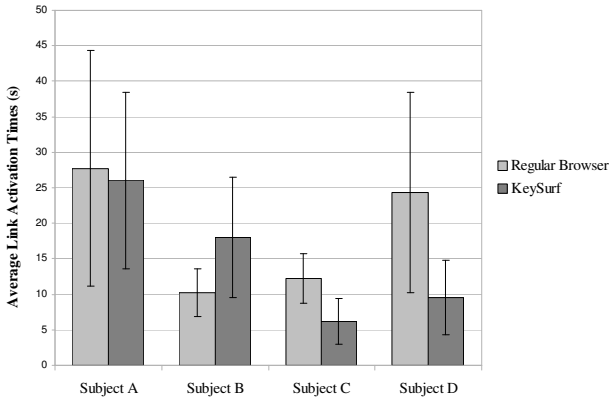

Figure 6: Mean link acquisition times with KeySurf and unmodified web browser.

For users accustomed to using a pointing device to navigate on the web, the resulting times show that the relative performance of KeySurf depends on the user's typing rate versus their ability to accurately control the on-screen pointer. To determine the significance of the differences in mean selection times, we performed independent T tests assuming unequal variances and a null hypothesis of no difference in means. P values for subjects A through D were 0.79, 0.014, 0.0042, and 0.0060, indicating that the null hypothesis could be rejected with high confidence for all subjects except A.

While the mean selection times for subject A were not significantly different between methods, the range of values is more constrained using KeySurf than with the pointing device. This is due to the fact that subject A frequently had problems with involuntary movement when controlling the track ball. This led to highly variable selection times, as some links could be selected very quickly, while others required several attempts. In contrast to this, subject A's typing performance was quite constant, making selection times with KeySurf much more predictable. However, more study is necessary to determine if the use of KeySurf would be beneficial to subject A.

Although subject C was accurate with the trackball device, performance with the KeySurf was significantly better, with a mean time of 6 seconds versus 12 for the trackball selection. In an informally extended testing session with subject C, the advantage of KeySurf was found to increase further for selection tasks that required scrolling, since with the trackball the subject has to move to the scroll bars to scroll down, and then move back to the web page content to make a selection.

Subject B did not benefit from the keyboard selection process used in KeySurf. In fact, mean link activation time with the joystick was significantly lower than with the keyboard. This result stems from the fact that subject B was very accurate with the joystick, but had trouble accurately pressing keys without the help of the plastic key guard. Although KeySurf performance for subject B may increase with the use of a better keyboard, we anticipate that for this subject the regular pointing interaction is more suitable.

Since subject D already used a keyboard to navigate the web, it was expected that any improved keyboard navigation system would have a large effect on element activation times. This was verified by our testing, where mean selection time using KeySurf decreased by 60% over this subject's accustomed method. Subject D was very pleased with KeySurf and has begun using the system at home.

7. FUTURE WORK

Although we hypothesize that the sidebar with suggested links will be useful to find interesting content and to reduce scrolling time, this component of our system remains to be tested. Effective user testing of this component is difficult, as the performance of the recommendation system depends on an accurately populated interest keyword list for a given user, which can only be obtained through regular use. We are currently planning extended user testing, where users will be asked to use the KeySurf system as part of their home web browsing routine for a period of several weeks. During this time, the system will record usage statistics such as sidebar use, page activity, and element selection cost to build a more accurate estimate of the system's performance for real world use.

An additional important factor determining the viability of the KeySurf system is its perceived usability by users. As KeySurf presents an interface that users are likely not familiar with, qualitative feedback will be very important to judge its value as a web navigation tool.

8. CONCLUSIONS

We have presented KeySurf, a novel character controlled web navigation system designed for users with disabilities that prevent them from accurately controlling a pointing device. By implementing several constraints on the matching algorithm that prioritize likely elements, we improve on the incremental search selection process and allow the user to select any element on a web page with very few keystrokes. Taking into account indicators of user interest, we provide a link suggestion system to aid in finding interesting content, and reduce the need for scrolling on long pages. Results from a pilot study indicate that the KeySurf web navigation system can significantly decrease selection time for people who have difficulty with accurate pointer control, even if their typing speed is slow.

9. REFERENCES

[1] J. A. Bilmes, X. Li, J. Malkin, K. Kilanski, R. Wright, K. Kirchhoff, A. Subramanya, S. Harada, J. A. Landay, P. Dowden, and H. Chizeck. The vocal joystick: A voice-based human-computer interface for individuals with motor impairments. In *Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, Vancouver, October 2005.

[2] R. V. Buskirk and M. LaLomia. A comparison of speech and mouse/keyboard GUI navigation. In *CHI '95: Conference Companion on Human Factors in Computing Systems*, page 96, New York, NY, USA, 1995. ACM.

[3] canassist.ca. CanAssist Dynamic Keyboard. http://canassist.ca/dynamickeyboard/, Oct 2007.

[4] K. Christian, B. Kules, B. Shneiderman, and A. Youssef. A comparison of voice controlled and mouse controlled web browsing. In *Assets '00: Proceedings of the 4th International ACM Conference on Assistive Technologies*, pages 72–79, New York, NY, USA, 2000. ACM.

[5] M. Claypool, D. Brown, P. Le, and M. Waseda. Inferring user interest. *IEEE Internet Computing*, 5(6):32–39, 2001.

[6] cmusphinx.org. CMU Sphinx: The Carnegie Mellon Sphinx Project. http://cmusphinx.org, Oct 2007.

[7] conkeror.mozdev.org. Conkeror. http://conkeror.mozdev.org, Feb 2008.

[8] S. Hackett, B. Parmanto, and X. Zeng. Accessibility of internet websites through time. In *Assets '04: Proceedings of the 6th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 32–39, New York, NY, USA, 2004. ACM Press.

[9] hah.mozdev.org. Hit-a-Hint. http://hah.mozdev.org/, Feb 2008.

[10] J. P. Hansen, A. S. Johansen, D. W. Hansen, K. Itoh, and S. Mashino. Language technology in a predictive, restricted on-screen keyboard with ambiguous layout for severely disabled people. In *EACL 2003*, 2003.

[11] C. T. Hemphill and P. R. Thrift. Surfing the web by voice. In *MULTIMEDIA '95: Proceedings of the 3rd ACM International Conference on Multimedia*, pages 215–222, New York, NY, USA, 1995. ACM.

[12] S. Oviatt. Interface techniques for minimizing disfluent input to spoken language systems. In *CHI '94: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 205–210, New York, NY, USA, 1994. ACM.

[13] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[14] M. Schrepp. On the efficiency of keyboard navigation in web sites. *Universal Access in the Information Society*, 5(2):180–188, 2006.

[15] L. Spalteholz, K. F. Li, and N. Livingston. Generating efficient labels to facilitate web accessibility. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 1319–1320, New York, NY, USA, 2007. ACM.

[16] L. Spalteholz, K. F. Li, and N. Livingston. Efficient navigation on the world wide web for the physically disabled. In *Proceedings of the 3rd International Conference on Web Information Systems and Technologies*, pages 321–326, Mar. 3-6, 2007.

[17] S. Trewin, S. Keates, and K. Moffatt. Developing steady clicks - a method of cursor assistance for people with motor impairments. In *Assets '06: Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 26–33, New York, NY, USA, 2006. ACM.

[18] w3.org. Web Accessibility Initiative (WAI) Home Page. http://www.w3.org/WAI/, Oct 2007.

[19] D. J. Ward, A. F. Blackwell, and D. J. C. MacKay. Dasher - a data entry interface using continuous gestures and language models. In *UIST '00: Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, pages 129–137, New York, NY, USA, 2000. ACM Press.

[20] yahoo.com. Content Analysis Web Services: Term Extraction. http://developer.yahoo.com /search/content/V1/termExtraction.html, Oct 2007.