WEB 2.0: Blind to an Accessible New World

Joshua Hailpern
University of Illinois
201 N Goodwin Ave
Urbana, IL 81820, USA
1-217-333-3328
Jhailpe2@cs.uiuc.edu

Loretta Guarino Reid
Google
1600 Amphitheatre Pky
Mtn. View, CA 94043, USA
1-650-253-4380
lorettaguarino@google.com

Richard Boardman
Google
1600 Amphitheatre Pky
Mtn View, CA 94043, USA
1-650-253-6586
rickb@google.com

Srinivas Annam
Google
1600 Amphitheatre Pky
Mtn View, CA 94043, USA
1-650-253-4380
annams@google.com

ABSTRACT
With the advent of Web 2.0 technologies, websites have evolved from static pages to dynamic, interactive Web-based applications with the ability to replicate common desktop functionality. However, for blind and visually impaired individuals who rely upon screen readers, Web 2.0 applications force them to adapt to an inaccessible use model. Many technologies, including WAI-ARIA, AJAX, and improved screen reader support, are rapidly evolving to improve this situation. However, simply combining them does not solve the problems of screen reader users. The main contributions of this paper are two models of interaction for screen reader users, for both traditional websites and Web 2.0 applications. Further contributions are a discussion of accessibility difficulties screen reader users encounter when interacting with Web 2.0 applications, a user workflow design model for improving Web 2.0 accessibility, and a set of design requirements for developers to ease the user's burden and increase accessibility. These models, accessibility difficulties, and design implications are based directly on responses and lessons learned from usability research focusing on Web 2.0 usage and screen reader users. Without the conscious effort of Web engineers and designers, most blind and visually impaired users will shy away from using new Web 2.0 technology in favor of desktop based applications.

Categories and Subject Descriptors
K4.2 [Social Issues]: Assistive technologies for persons with disabilities. H5.4 [Hypertext/Hypermedia]: Navigation, User Issues.

General Terms
Performance, Design, Human Factors.

Keywords
Web 2.0, Screen Reader, Blind, Visually Impaired, Use Models.

1. INTRODUCTION
When screen reader software was created in the 1980's, no one could have predicted how computer technology, and thereby screen readers, would have evolved. However, the latest technological shift has radically altered the visually impaired user's model of interaction.

WWW'09, April 20–24, 2009, Madrid, Spain.

For many years, Web content was relatively static. Aside from the occasional movie, a screen-reader user was able to traverse the textual and image content on Web sites through a series of keyboard-based interactions. However, with the advent of dynamic Web content, the traditional use model has been shattered. While dynamic content that runs in virtual environments, such as Flash, poses many challenges, it exists within its own environment. Hence, its accessibility is directly related to the features made available by that environment. Web 2.0 content (using AJAX and JavaScript) dynamically alters traditional Web page content. It confounds the mental and physical model that screen-reader users have developed.

The implication for Web inaccessibility is becoming a greater concern. Dynamic Web 2.0 [22] applications (e.g., Gmail, Facebook and Apple *.mac* Web Apps) have increasingly blurred the line between desktop applications and the Web. While the added benefit from rich dynamic Web application grows, the divide between functionality and usability is expanding. Specifically, Web 2.0 applications and their use of scripting and dynamic content are actively managing the model of the content presented to the user. Screen readers, with their virtual buffers, are also managing the model of the content presented to their users. It is easy for these two models to get out of sync with one another, leaving the screen reader user disoriented or working with an incorrect model.

Current studies have shown that blind users tend to avoid dynamic pages [3]. Hence large populations are missing out on Web content simply due to its (lack of) accessibility. As researchers, it is incumbent upon us to understand and model the current world, including the use models of our users. Without this understanding, the chasm between new technology and its usability will continue to increase. New technology must be developed with an understanding of the needs and concerns of our target user population. No matter how powerful the technology is, if it is not accessible, over 1.5 million visually impaired individuals in the USA[1] will not be able to use it. The worldwide statistics put the number of visually impaired at 161 million or about 2.6% of the world population [29].

While screen reader implementations that support WAI-ARIA [27, 28] are addressing some of these problems, screen reader users will need to be aware of this more complex model of interaction. Designs should simplify this task as much as possible for the user.

This paper presents two models of screen reader users interacting with the "traditional Web" and Web 2.0 applications. Based on the Web 2.0 model, we present accessibility hurdles directly related to the interaction between screen readers and Web 2.0 applications. We conclude by presenting a model for Web 2.0 accessibility design and user workflow as well as a set of design

requirements for developers to ease the demands on the user and increase accessibility. These findings were grounded in results from iterative usability research, interviews with screen reader users, and an examination of the relevant technology. The main contribution of this work is a series of models for understanding the use model of visually impaired users, an analysis of accessibility for Web 2.0 applications with screen readers, and a series of implications for design of accessible Web 2.0 applications.

2. DEFINITIONS

For those users who are unable to rely upon the visual and textual representation of data on a computer, other audio or tactile based solutions must be employed. We define this user group, those that cannot use vision as their primary mode of interaction, as *visually impaired users*. To help improve the accessibility of computers, companies have created programs called *Screen Readers* that translate textual content on a computer to other modalities for accessing screen content, as well as providing hot-keys for computer interaction (because the utility of a mouse is minimal). While there are tactile displays that translate text to Braille [2], this paper will focus on audio-based screen readers [9, 14]. Specifically, we will focus on the JAWS screen reader, considered the most "popular" solution on the market [10].

In order to describe the process of accessing content via screen readers, we will use the term "reading" not literally, but rather to describe the translation of textual content to computer-generated voice.

3. BACKGROUND AND RELATED WORK

Research on accessibility for the visually impaired community has been conducted for decades, first beginning with SAID and later Screen Reader/2 by Jim Thatcher at IBM [24, 25]. In 1994, Jim Thatcher stated "blind users must have access to the same computing environment as their sighted colleagues" [25]. Solutions have also expanded beyond the scope of purely audio-based systems, such as the Refreshable Braille Display [2]. The ubiquity of screen readers in today's world, in conjunction with an increased awareness for individuals with disabilities [7, 16, 20], has prompted growing attention from a wide range of communities, organizations, and companies [6].

3.1 Web Accessibility

While much work has focused on screen reader development, research has sought to improve accessibility on the Web. Research has examined Web accessibility for new technologies and navigation [18, 21] in order to allow the Web to be as accessible as the desktop application. To lower the overhead for the Web developer, tools have been created to allow the developer to test the accessibility of their existing sites [11, 23]. In addition, software packages have been created [4, 8] to provide developers with a richer set of tools to increase the accessibility of their own site.

3.2 Web 2.0 Applications

In 1999, new techniques were developed to advance the state of Web technologies. The introduction of Dynamic-asynchronous loading [26] converted static HTML Web pages into rich and interactive Web applications. Through the use of JavaScript, AJAX [12] Web sites had the ability to provide desktop-like interactions to users, dynamically updating the content, and retrieving new material to be displayed after page load. Further, Web sites, once limited by images, text and forums, can repurpose keystrokes [13] to facilitate additional interactions, further blurring the line between Web and desktop. Additional development of client-side infrastructure, such as Google Gears, has blurred the lines between desktop and cloud even more. We detail the new forms of interaction and situate them with examples in a fictional Web 2.0 email client, due to email's ubiquity in our society.

3.2.1 AJAX Interactions

Web 2.0 applications can provide user keyboard commands much as standard desktop applications do. This additional functionality allows users to treat Web browsers as application delivery systems. For example, an online Web 2.0 email client could allow users to iterate over emails using keyboard commands (e.g., J and K keys). In addition, the C key can be mapped to open "Compose New Message." By means of keyboard-based interaction, users are able to navigate Web 2.0 applications quickly. Users can access complex functionality without exploring menu systems or finding the appropriate links on a page. This type of functionality also opens the door for Web 2.0 applications to provide features that are only available via keyboards (e.g., online games).

3.2.2 Dynamic Content

One of the most notable features of Web 2.0 applications is the dynamic nature of its content. Web sites can use AJAX to dynamically retrieve new content without page refresh/reloads, and adjust the information, content and layout of a page on the fly. For example, a list of inbox emails can be dynamically updated to reflect new and incoming mail, so a user does not need to repeatedly refresh the page. Larger changes to the page can also be implemented through AJAX. When a user chooses to compose a new email, the entire page appearance can change to reflect the new functionality without refreshing or reloading the page by removing the inbox and replacing it with a new email form. This support for dynamic page content reduces load time and allows for pages to change, reflecting the dynamic needs of users. In short, this additional functionality and seamless interaction increases the deployment of Web-based applications and further blurs the line between desktop and Web applications.

3.2.3 Custom Controls

AJAX facilitates a host of custom control mechanisms for interacting with Web content, such as links, buttons, check boxes, and combo-boxes. The behavior of these custom controls is provided by JavaScript, which can reduce the load time of the Web-application by not requiring all content to be loaded immediately. The appearance of these controls can be further customized for look and feel, allowing the application content to be designed uniquely for each Web application.

As an example of this custom control, a *decorated link* is text that is 'decorated' to appear like a traditional page link. This visual decoration is created using CSS markup. The behavior of the *decorated link* is implemented via JavaScript event detection (e.g., when a user clicks on the link) and JavaScript content changes (dynamically updating page content, or actually redirecting a browser). To the user, these have the same visual appearance as links. When clicked by a mouse, they appear to have the same result (due to the JavaScript).

Additional, application-like controls such as menu systems (e.g., File, Edit, View along the top of an application) can be implemented using AJAX. These custom controls further allow Web 2.0 applications to appear like traditional desktop applications.

3.3 WAI-ARIA

In response to the growing concerns related to Web accessibility in the AJAX world, a new technical specification named WAI-ARIA, or ARIA for short, is in development [27, 28]. The main feature of the ARIA standard is to allow Web pages (or portions of a page) to declare themselves as applications rather than as static documents, by adding role, property, and state information to dynamic Web applications. ARIA is intended for use by developers of Web applications to enhance a Web application's compatibility with screen readers and other assistive technology with the help of an ARIA-supporting Web browser. This provides additional information that will be useful for an assistive technology user to understand the type of the widget and its current state. With this additional role and state information, an AJAX application can be made much more accessible than was previously possible and a screen reader can get useful information such as whether focus is on a tree item, whether the item is expanded or collapsed, or the level of the item in the tree hierarchy.

Consider a dynamic cascading tree on an AJAX page. ARIA markup allows a designer to provide *roles* for *div* elements of the tree in mark up. The outer-most level of the tree structure can be set to *role=tree*, while each item can be labeled *role=treeitem*. As users open and close branches of the tree structure, additional attributes such as *expanded* or *collapsed* can be applied, providing screen readers with appropriate knowledge about changing page structures that can be relayed to users.

In addition to providing roles and states, ARIA also provides mark-up for managing focus and for managing dynamically changing content. The *activedescendent* property gives authors better control over the focus notifications sent to screen readers. Since screen readers track the focus and describe objects that receive focus, it is very important to accessibility that focus be managed properly. The *live* property lets author indicate how important it is to notify users about content that has changed dynamically. Since failure to know when content has changed dynamically is the source of much confusion for visually impaired users, this ARIA property is critical to helping screen readers support web applications

ARIA holds the potential to improve many aspects of Web accessibility. Its main focus is providing additional mark-up, identifiable to screen reader. But, it is not a complete guide to designing more accessible applications. While ARIA is essential, without understanding the constraints and mental models of our users, developers of Web applications cannot fully appreciate the complexity of their situation, nor how to design to meet the needs of this population.

4. USER RESEARCH

During the summer of 2008 researchers developed and tested Web-based solutions to help teach Web 2.0 applications to screen reader users [15]. In particular, [15] examined the use of dynamic and interactive tutorials whose content was embedded within Web 2.0 applications themselves. These solutions were based upon two-hour interviews with five members of the visually impaired community in the Silicon Valley, CA area. Seventeen visually impaired users (20-60 years old) evaluated the resulting tutorial solutions. While this work was specifically targeting interactive tutorial design for screen-reader users, additional lessons, understandings, and models about limitations and accessibility for Web 2.0 applications were revealed.

In feedback from the initial interviews and subsequent software evaluations, users articulated their frustration with dynamic page content. They expressed how difficult it can be to change their mental model of 'how the Web works'. One finding of note was the robust set of interactions that users relied upon to navigate a Web page (via links, tables, headers, and other Web elements), as well as textual content (reading the whole page, by paragraph, by sentence, word-by-word, and even letter-by-letter). Moreover, these complex and varied forms of interaction have become ingrained in their usage of computers. Hence, disruption to these models has severe usability repercussions.

In the current paper, we focus on three findings relating to Web 2.0 usage for screen reader dependent users: a model for screen reader usage, a model for Web 2.0 usage, and accessibility concerns for interaction between the two. These findings were directly based upon the feedback from seventeen screen reader users. The following sections, present the main contributions of this paper: a model of screen reader interaction with traditional Web content (Section 5), a model of screen reader interaction with Web 2.0 applications (Section 6), difficulties in accessibility that arise with Web 2.0 applications (Section 6), and a set of design requirements to improve Web 2.0 application accessibility (Section 7).

5. MODEL OF SCREEN READER INTERACTION WITH TRADITIONAL WEB CONTENT

This section presents a model of screen reader interactions with traditional Web content. We first outline the current set of modalities screen reader users have to facilitate the Web-based interactions. To better illustrate the workflow of a screen reader user, we situate our model in a real-world example based upon interviews with and observations of screen-reader users. We conclude by discussing the specific features each modality provides to a screen reader user.

By modeling and understanding the methods screen-reader users rely upon to interact with computers, we are able to gain insight into how new technology should be designed to meet their needs, concerns, and mental models. Further, modeling allows us to compare screen reader support for traditional Web browsing versus Web 2.0 application usage. These models and descriptions were based upon interviews with screen-reader users, development of Web-based technology targeting visually impaired individuals, and use of screen readers themselves.

5.1 Mapping Modalities to Interaction

Our model entails a mapping of user interactions to modes and features to modes. We now probe the modalities and functionality of the screen reader itself. Screen-reader software solutions have multiple modes of interaction. The JAWS screen reader uses these specific mode names, however similar modalities exist in the majority of the commercial solutions:

- **Virtual Cursor Mode** (VC) - Conceptually, **VC** primarily focuses on textual content retrieval and interaction with the static structure of the content. Users rely mostly upon VC for reading text, and navigating the content on a Web page.
- **Forms Mode** (FM) - **FM** facilitates interaction with interactive control objects (text input fields, check boxes, pull down menus, etc.). Users rely on FM to fill out forms and interact with control objects. FM is automatically activated when users hit the enter key while on an interactive control object.

Interaction	Virtual Cursor Mode	Forms Mode	PC Cursor Mode
Navigate between Pages	●	●	●
Navigate within Page	●	○	○
Access to Form Elements (e.g check box, button)	●	◑	○
Read & Re-read Text	●	○	
Text Input		●	

● Full Support ◑ Partial Support ○ Minimal Support

Table 1. High level, user-Web interaction breakdown across screen reader modes for traditional Web usage.

- **PC Cursor Mode** (PCM) - In **PCM**, users can no longer navigate content via a virtual buffer (Section 5.1.1), utilize screen reader navigation, or use screen reader keyboard hot-keys, or read large bodies of static text. This does facilitate keyboard commands to be detected by active applications.

Each of these modes provides the user with a different set of interactions (Table 1). Most users never activate PCM for day-to-day tasks. During our interviews and research, none of the users had utilized PCM before. While the screen reader may automatically switch between VC and PCM, it provides no feedback to the users about this switch. As far as users are concerned, this modes switch never happens (because it occurs transparently by the system), and therefore, they never consciously use PCM; this effectively makes it an unused mode. Throughout this paper, we will treat PCM from the perspective of the user, not the underlying technological structure. Section 5.2 provides an example workflow illustrating screen reader mode switches. See Section 5.3 for a detailed description of functionality in each mode of interaction.

5.1.1 VC and the Virtual Buffer

Popular screen readers use various techniques to improve user experience while they interact with Web pages. Of note is the virtual buffer. This feature, simply put, provides an in-memory copy of the current Web page so a screen reader user can navigate quickly and easily through the page with the help of the screen reader's navigation commands. For example, once a virtual buffer is present, a user can press letter P to move to the next paragraph or use the JAWS command to search in the buffer.

5.2 Illustration of Workflow: Interaction with Traditional Web Content

Section 5 highlights a model of screen readers interacting with traditional Web content. This can be best illustrated by describing the workflow of screen reader users, and how they normally use Web controls and read content. To better demonstrate the use of screen readers and their different modes of interaction, we describe a fictional screen reader user, Alice, interacting with a Web based email client. This scenario is an amalgam of observations, interaction and discussion with users in the experimental context [15].

5.2.1 Loading Web Content (VC Only)

When Alice starts her Web browser, her screen reader is in VC. The Web browser loads her homepage. Her screen reader announces the page load completion and automatically reads all page content from top to bottom. Using the Ctrl hot-key to stop the text to speech, Alice proceeds to press Ctrl+L, and keyboard focus moves to the URL bar in her Web browser. She types in the URL of her favorite Web based email client. Upon page load, the screen reader announces that the content has been completely loaded and begins reading the entire page content.

5.2.2 Reading Page Content & Navigating Text (VC only)

Alice wishes to immediately find out what new emails she has. Because Alice has used this site before, she does not need to read the entire page to understand the current content. Rather, she remembers that the launch page has a list of emails in a table. Still in VC, Alice uses the T key, to cycles through the tables on the page, until she finds the one titled Inbox. Using the Ctrl + Arrow Keys, she iterates row-by-row over the table, listening to the content of the row. To repeat listening, she uses her Control + Arrow Keys to go backwards and forwards word-by-word.

Alice notices that she has a new email from her employee Eve. Using the Arrow Keys, she finds the subject line, which the screen reader informs her is a link. Pressing the Enter key, her Web page reloads, and upon completion, the screen reader begins reading the entire page back to Alice. However, Alice only wants to read Eve's email. To skip to the body of the email, Alice uses the P key, skipping paragraphs until she reaches the body of Eve's email. Alice then invokes the Insert + Page Down key command, which tells the screen reader to read the rest of the page from this location forward. When she wishes to re-read content, she uses her Arrow Keys to re-read each line or Alt + Arrow Keys to re-read each sentence, occasionally using the num-pad 5 key to read each word, a letter at a time. Eve asks Alice to contact their finance director Bob to ask him to send the financial reports from the past year.

5.2.3 Constructing An Email (VC & FM)

Alice presses the Insert + F7 key, and brings up a list of all the links on the page. Using her Arrow Keys, she cycles through the list of links until she locates "Compose New Email." Pressing Enter, the page reloads, and the screen reader begins iterating over all page content.

Alice knows that an **H2** header immediately precedes the text entry fields, therefore she uses the 2 key, the JAWS command in VC mode to cycle through level two headers on a page, and iterates over all **H2** headers. If she did not know what level header she was looking for, she could have used the H key, the JAWS command in VC mode to cycle through all headers on the page. Upon reaching the appropriate header, she locates the first text area, the *To* field. With focus on the text area, Alice presses the *Enter* key, and switches from VC to FM. The screen reader now places the focus within the text area, and notifies Alice of the change in state/mode. Alice then begins typing Bob's email address. She then presses the TAB key, jumping to the *Subject* line, and fills in the content. She TABs again to the body, and writes the content of her email. She reviews the content by pressing Arrow Keys. Finally, she presses TAB once more and locates the "Send" button. Pressing Enter, the email is sent, and the page reloads to display the content of her sent email. Alice then switches back to VC and she can reread her email or find the inbox link on the page and continue iterating over her inbox.

As illustrated in this example, most reading of textual content and Web page navigation is with VC, while text entry is with FM. Most other Web and computer based interactions utilize VC and FM. Traditional screen reader interaction does not use PCM.

Features	Virtual Cursor Mode	Forms Mode	PC Cursor Mode
Access Check Boxes	●	●	●
Navigate by TAB key	●	●	●
Access to Buttons	●	●	●
Execute Links	●	●	●
Read & Re-read Text *	●	●	
Access Combo Boxes	●	●	
Text Input		●	
JAWS Find	●		
Navigate by Element Type	●		
Navigate by Arrow Keys	●		
Navigate using Headings	●		
Navigate by page links	●		
Easy Tables Navigation	●		
JAWS Goto Line	●		
Read bold, italics, etc.	●		

* in forms mode, only read text in form iteams and only re-read text in edit areas

Table 2. Breakdown of functional interactions across screen reader modes for traditional Web page usage [15].

5.3 Mapping Modalities to Features

While our discussion of our model thus far has described the conceptual segmentation of screen readers and Web usage, we now explore the functionality contained within each screen reader mode. We present a breakdown of functionality across each screen reader mode (Table 2). Due to the difficulty in using mouse-based interactions by visually impaired users, screen readers rely upon custom hot-keys to provide the traditional suite of computer interactions. Because of the wide variety of human-computer interactions, most keys on a keyboard are assigned to interactions. In addition to the general set of keyboard hot-keys, screen readers tend to adjust some keys for application specific functionality. For example, as a user switches between Word and the Windows desktop, certain keys will change to reflect the changes in features/functionality.

Because so many keyboard keys are used for JAWS commands, textual/keyboard input (to type text, or fill out online fields) is not possible within the same modality. To facilitate input, an additional mode (FM) is necessary. In this mode, users enter text into input fields and the keystrokes are not interpreted as navigation commands.

Not all screen readers utilize a separate PCM mode of interaction. For example, WindowEyes [14] allows most standard keyboard based interactions in FM. However, in JAWS, PCM is an additional mode, though not commonly used. Its main purpose is to facilitate interaction that is not text input, and requires use of key activations that are normally captured by the screen reader in VC. One screen reader user described this lack of familiarity by stating:

> Because everything I know about JAWS is not helping me out at all. It sort of like... you have a sufficiently different way of navigating, which is going to require a completely new learning curve to navigate it - PP4

Another user described her frustration with the new modality by stating:

> …this is a command I have never used! …. Nobody – and I have had several different teachers – nobody has told me ... a command for turning the virtual cursor off.. - PP5

6. MODEL OF SCREEN READER INTERACTIONS WITH WEB 2.0 SITES

In this section, we present a model of interaction between screen reader users and Web 2.0 applications. Using this model and interviews with screen reader users [15], we highlight current accessibility issues. Section 6.1 presents a scenario involving a fictional screen reader user Alice, interacting with a dynamic Web-based email client. As more and more modern uses of the Web revolve around these dynamic sources of content, accessibility becomes a growing concern.

6.1 Illustration of Workflow: Interaction with Web 2.0 Content

To better illustrate the use of screen readers and the difficulties inherent with Web 2.0 applications, we illustrate our fictional screen reader user, Alice, interacting with a Web based email client that is using Ajax (without ARIA). We find Alice with her Web browser open, pointed at the Web 2.0 version of her Web based email client. This scenario is an amalgam of observations of interaction, coping techniques and feedback from users in the experimental context [15].

6.1.1 Reading Page Content

When the page loads, the screen reader (in VC) begins reading the entire page content. Alice wishes to immediately find out what new emails she has in her inbox. To iterate over the inbox content, the Web Application uses two hot-keys, J and K. Alice switches her screen reader to PCM, to enable her to interact with these custom hot keys. As she cycles over each element, she gets no audio feedback that the page content has changed or that she is iterating over different emails. Realizing that this custom interaction does not function for screen readers as described, she returns to VC. She then looks for tables. However, the inbox is not created with a table element, but rather with CSS styling of div elements. As a result, her traditional method of interaction does not function. Realizing there must be a link somewhere, she brings up her list of page links. She finds a series of links that appear to follow a pattern: name, a subject, a date/time. This does not appear to be easy to follow, but Alice surmises she can TAB through these links, and get access to the content. After navigating to the series of emails, she TABs through sets of name, subject, date, until she finds one from Eve.

Pressing Enter, she opens the email. However, because the mail client only updated the DOM, the page did not reload. Alice waits for page load confirmation, but receives none. Confused, she presses the Enter key again, and gets an error from the screen reader, informing her that she is not on a link or clickable element. This confuses Alice even more. She instructs the screen reader to re-read the entire page. This is when Alice notices that the page content has changed. Locating the body of the email, Alice reads the content in VC, and realizes that Eve would like Alice to contact their finance director Bob, and ask him to send the financial reports from the past year.

6.1.2 Constructing An Email

Alice presses Insert + F7, and brings up a list of all the links on the page. Using her Arrow Keys, she cycles through the list of

links however, does not find any link called "Compose New Email." What visually appears on the page is not a true link, but a *Decorated Link*, that has the appearance of a link, but does not match the link properties. Therefore the screen reader does not detect it. Alice begins to read through the entire page until she locates the simulated link, however, even though she presses Enter, the page does not change, nor does the screen reader read the text "Compose Mail" as a link. Alice remembers, that she can activate "Compose Mail" via the application C hot key. Alice switches to PCM, and presses the C key. For the sake of simplicity, lets assume that the page *does* reload (does not use a DOM update, though a DOM update would be the most efficient). The screen reader notifies Alice that the page loads, but because she is not in VC, it does not automatically read the entire page. Further, the page uses JavaScript to automatically place Alice in a text field. Alice must then switch back to VC, to uncover what page she is on. She instructs the screen reader to read the page content, and uncovers that she *is* in "Compose Mail." Because this moved her outside of the text area, Alice must switch to FM. She then TABs to her text areas, and fills out her email. TAB'ing to the send button, Alice sends her email.

The page reloads to display a static-text version of her sent email. Alice switches to VC, in order to read the current page content. Realizing her email is correct, she wishes to return to her inbox to continue checking emails. However, the inbox link is also a *decorated link*. In order to begin this cycle again, she must switch to PCM, use an application hot-key to switch to her inbox, then return to VC to cycle through her inbox list.

6.2 AJAX Keyboard Interactions

One central aspect of the screen reader and Web 2.0 model is examining the interaction with AJAX content. With the integration of these new technologically driven features, accessibility and interaction is not readily addressed for screen-reader users. Without providing appropriate means of interacting with content, or understanding the challenges that come from the interaction, Web 2.0 applications will be difficult for screen reader users.

Though the integration of hot-keys into Web applications generally increases the usability of Websites, particularly Web applications, screen-reader users have a hard time taking advantage of this functionality. This is a result of the keyboard capture mechanism used by screen readers. Keyboard actions are captured by the screen reader *before* they reach the Web browser, and therefore before any rich Web application. In order to allow screen readers to interact with Web applications, users must switch to PCM, a mode that is not commonly used. Table 3 is an extended version of Table 2 that includes Web 2.0 interactions separated by screen reader modality. As illustrated in Table 3, traditional Web browsing does not require use of PCM mode, yet it is often the only mode in which users can access application hot keys in keyboard-based Web 2.0 content. In other words, users must switch back-and-forth between VC (for traditional Web interaction and content reading), FM (for input) and PCM (for Web specific features). One screen reader user described this new mode of interaction by stating:

> I would use the analogy of driving on the opposite side in Europe. Because every time they go into this program they are going to have to junk everything that is in their head ... they are going to be doing this every time they go to the Internet – PP10

Features	Virtual Cursor Mode	Forms Mode	PC Cursor Mode
Access Check Boxes	●	●	●
Navigate by TAB key	●	●	●
Access to Buttons	●	●	●
Execute Links	●	●	●
Read & Re-read Text *	●	●	
Access Combo Boxes	●	●	✓
Text Input		●	✓
JAWS Find	●		
Navigate by Element Type	●		
Navigate by Arrow Keys	●		
Navigate using Headings	●		
Navigate by page links	●		
Easy Tables Navigation	●		
JAWS Goto Line	●		
Read bold, italics, etc.	●		
Web Application Hot Keys **			✓
JavaScript caused Text input			✓
Move Focus via JavaScript			✓
JavaScript changes DOM ***			✓
Utilize JavaScript State			✓

● JAWS Web Browsing ✓ Web 2.0 Features

* in forms * mode, only read text in form items and only re-read text in edit areas
** *Virtual Cursor Off* prevents conflict with Screen Reader Keys
*** without audio feedback
**** cannot go word-by-word or line by line

Table 3. Breakdown of features across screen reader modes for Web 2.0 application usage [15] with traditional interaction (for context) is shaded.

In short, AJAX interactions (crucial to Web 2.0) require PCM that most screen reader users do not know about. Given user reaction, and the increased complexity required to switch back-and-forth between screen reader modalities, Web designers must take visually impaired users into consideration from the very beginning of the design cycle.

6.3 Dynamic Content Updates

Highlighting the effects and implications of dynamic content updates in our user model is necessary to understand how to improve technology. When some page content is updated dynamically, screen reader users generally remain uninformed as to page changes. Screen readers alert users to page changes on page refreshes, not on most dynamic content changes. While users can learn to expect content changes based on their actions (e.g., clicking a button), when the content refreshes without their direct action, users can become confused. Further, if users click what appears to be a link (expecting to be redirected to a new page) and pages do not refresh, but dynamically update parts of the content, users can be unaware of page changes or even that they correctly activated the control. To complicate the situation further, traditional Web browsing feedback is not provided in PCM. Therefore, when users invoke AJAX interactions in PCM (Section 6.2), they cannot rely upon any of the feedback they are accustomed to.

As a result, one of the main features of Web 2.0 applications is lost on the visually impaired community. Without feedback, three main situations arise:

1. **NOTIFICATION**: User does not realize content has changed and is therefore unable to locate his/her old position or content they presumed was on the page
2. **LAYOUT & STRUCTURE CHANGES**: The user realizes, somehow, that the page has been updated, but must reread the entire page to discover what has changed.
3. **CONFIRMATION FEEDBACK**: When user performs actions, without confirmation, he/she is unaware if clicking a link or pressing a hot-key actually had an effect

None of these situations is desirable. They reduce the accessibility and usability of the entire Web application.

6.4 Virtual Buffer and Focus

The virtual buffer, and its effects on Web 2.0 applications is another dimension of our user model. While the virtual buffer is an important feature in screen reader technology, it has caused some interaction difficulties for the visually impaired user. Most notably, problems arise when the content of an AJAX application page changes, often without any user intervention. This poses a major issue for the screen readers, since the virtual buffer is no longer in-sync with the Web page shown on the screen. Not only are the unexpected content changes confusing, the user may also lose his/her place in the content. When a screen reader moves around the page, it is actually maintaining the location of the Virtual Cursor (the user's position) in its virtual buffer. When the page dynamically updates, the Virtual Cursor location may also shift, often in an unpredictable fashion. While in the more benign cases, the Virtual Cursor may simply move elsewhere in the virtual buffer, the most extreme cases results in the screen reader getting completely confused. In one such example, we observed the screen reader reading HTML/CSS from the page's source rather than meaningful content. Because the screen reader does not recognize the structure of the Web site, due to the buffer change, hot keys appear to do nothing because, to the screen reader, they have nothing to act upon. This can be mildly confusing to sighted users, but they have the ability to visually scan a page. However, visually impaired users become completely lost as to their position and what page they were on.

6.5 Custom Controls

Though there are benefits to using Custom Controls, there is a major accessibility issue with them. These links and control mechanisms are simply "stylized" to visually appear as their standard Web/form element counterparts, while not utilizing native anchors or controls. As a result, this content is not identifiable by screen readers, and therefore, not identifiable by visually impaired users. We model the accessibility and interaction implications of custom controls.

6.5.1 Decorated Links & Link-Based Navigation

This lack of screen-reader identification of decorated links has a particularly large negative impact on Web-page navigation. Screen readers provide a mechanism for navigating a Web page via list of links on the page. This link-based view of the Web is a essential tool for Web navigation [6]. A large effort has been made to ensure linked text and alternate text on images are representative of the link destination to aid in this link-based traversal [6]. However, if links fail to appear in the list of links, screen reader users will not be aware of their presence, let alone be able to use them.

6.5.2 Utilizing Custom Controls

To further exacerbate the issue, if a screen-reader user encounters these custom controls or decorated links through Web-page content traversal, traditional keyboard interaction (e.g., activation of button, execution of link, or checking of check-box) may not be supported. Although many implementations of decorated links and custom controls do not support keyboard activation, good JavaScript practice includes support for keyboard events in addition to mouse events. While this additional functionality does not provide methods for screen readers to detect the existence of these control elements, they will have the appropriate functionality attached.

6.6 Access to textual content

A major aspect of Web browsing, computer usage, and our user model is access to textual content. As Table 3 illustrates, PCM provides no ability for users to get access to any textual content. For example, consider a list of inbox emails. A user can adjust his/her position in the queue through an Up key and Down key. If using JavaScript, the user's position would be stored in a JavaScript variable without providing any notification to the screen reader, and the screen reader would not pick up any visual placeholder changes. Moreover the user would receive no feedback that state even changed, let alone what it changed to.

6.7 Existing Web 2.0 Accessibility Solutions

Clearly, solutions can be implemented in the client software to address many of these limitations. The Dynamo Web browser [5] and updates to screen readers themselves are viable solutions to this problem. However, these put the burden of using new software (which may pose additional problems) or updating their existing software (which most likely will incur an additional cost to the user) on the visually impaired user. Further, new software and new update adoption is not ubiquitous. Thus, it is our responsibility, as researchers and developers, to seek out solutions that can be implemented in the back-end server software, so as to increase the impact and accessibility as broadly as possible.

Currently, development of the ARIA specification holds great promise for improving Web accessibility. ARIA allows a developer to specify semantic information for UI widgets with the help of various role values and a set of state attributes that are appropriate for the given role as well as managing focus location and alerting many dynamic content changes. ARIA addresses some of the above problems by providing developers a way to specify roles and states for such controls. Although ARIA is a significant step in making AJAX applications more accessible, it is not a panacea. Obviously, one cannot make a poor UI design better just by introducing ARIA roles and states. Before an AJAX application can be accessible, it is also critical to provide proper keyboard support and proper focus management in the application. While ARIA is a promising first step to addressing the growing complexity of AJAX applications, there are a number of complex UI widgets used in today's Web applications that cannot be described by ARIA markup. However, while ARIA provides critical tools for enabling support of dynamic web applications, it must be used in ways that still present a manageable model for users.

The following section details a set of design requirements for Web designers to create more accessible Web 2.0 applications based directly upon the findings above.

7. DESIGN REQUIREMENTS FOR ACCESSIBLE WEB 2.0 SITES

We present here a set of design implications to influence the development of future Web 2.0 applications. We relate these design implication to sections (highlighted in bold) that discuss usability concerns that are addressed by each design implication. When appropriate, we note when ARIA can provide support. We conclude this section with a brief discussion of how navigating a Web Application that follows these guidelines can ease a visually impaired user's interaction. To make full use of these requirements, we recommend that users integrate these techniques in the early stages of the development. In many cases this will not only produce more accessible code (cleaner with less markup) but can also produce applications that are more useful for the general population.

7.1 Alternate Modes of Audio Feedback

One critical deficit of Web 2.0 applications is the lack of screen reader detection of changes (both state and page content) in a Web page (**6.3, 6.6**). ARIA does provide some resolution to this problem for users in VC. While in PCM, no automatic feedback is provided (without developer assistance). Designers should employ a system (e.g. AxsJAX [8]) to expose author provided audio content to the client's screen reader. This not only provides additional feedback for VCM, but also for PCM. While there is a large benefit to the screen reader user from this additional audio feedback, its inclusion may increase general Website load time. The performance-conscious developer can mitigate this by enabling this additional feedback through an opt-in system.

7.2 User Workflow Design Model

As illustrated in Section 6.1, modern navigation of Web 2.0 applications is not only complex, but requires a multitude of mode switches for screen reader users. Users in our research described this set of complex mode switches as greatly bothersome. One visually impaired user (PP14) stated "the switching between, is a stressful thing for me." Another user summed up his interaction with keeping track of multiple modes by stating:

> … you have it off to read, and off to perform some of the keyboard commands, and you have other commands that you have to use when the virtual cursor is on, and then you have to remember that you have to have the virtual cursor on to turn forms mode on, which confused me... it just seems to be a lot of steps – PP1

Through this feedback from users, we constructed an interaction design that future Web applications can follow. This new model, briefly outlined in [15], breaks up user-based interaction into two conceptual groupings: *Reading Mode* and *Control Mode* (Table 4). In this section, we will provide a complete design and implications of this model. The entire workflow model centers on the premise that users should exist in one of two conceptual modes. This dual modality reduces the strain from an increase in mode switches to access Web 2.0 content, as well as providing a logical model to aid screen reader users learn and remember the mode to access appropriate functionality (**6.2**). *Reading Mode* consists of traditional Web browsing, focusing on retrieving content from the Web site itself. *Control Mode* should contain all the Web application based interaction. Thus performing application commands, accessing features, and manipulating the features of the online program should be presented and be reachable via *Control Mode*. The critical aspect of this model is

	Features	Reading Mode	Control Mode
Btwn Page Navigation	Execute Links	●	
	Navigate by page links	●	
Within Page Navigation	JAWS Find	●	
	Navigate by Element Type	●	
	Navigate by Arrow Keys	●	
	Navigate using Headings	●	
	JAWS Goto Line	✓	
	Easy Tables Navigation	●	
	Navigate by TAB key	●	
Form Content	Access Combo Boxes	●	
	Access to Buttons	●	
	Access Check Boxes	●	
Reading	Read & Re-read Text *	●	
	Read bold, italics, etc.	●	
Textual Input	Text Input	●	
Web 2.0 Features	JavaScript caused Text input		✘
	Web Application Hot Keys **		✘
	Move Focus via JavaScript		✘
	JavaScript changes DOM ***		✘
	Utilize JavaScript State		✘
Feedback	Generate spoken text ****		✘

● Virtual Cursor Mode ✓ Forms Mode ✘ PC Cursor Mode

* in forms mode, only re-read text in edit areas
** *Virtual Cursor Off* prevents conflict with Screen Reader Keys
*** without audio feedback
**** cannot go word-by-word or line by line

Table 4. Illustration of User Interaction model outlined in Section 7.2. Breakdown by feature (Table 3), high-level interaction concept (Table 1), and screen reader modality are inline to provide a robust view of the future of screen reader and Web 2.0 interaction.

that Web developers must provide the appropriate accessible functionality for each mode of interaction as follows:

1) Examine the workflow of the Web app carefully, and divide activities into those that conceptually take place in Control Mode and those that should take place in Reading Mode.

2) Make sure that the user does not have to switch modes frequently, e.g., when interacting with the app, the user should not have to switch to Reading Mode to get oriented or to navigate to a different part of the application. When in Reading Mode, the user should not need to switch into Control Mode to navigate or perform some action *in the middle of reading*.

3) Clearly there will be times when it is necessary to change modes. Try to design the application so this can happen automatically and naturally. (e.g., use of application role can cause parts of the content to be processed in Control Mode). At a minimum, ensure that the switch follows the conceptual model of needing to *control* the application or needing to *read* content.

4) In PC mode, content is only read when it takes focus. If it is important to have the screen reader speak content on certain operations, it may not happen automatically. It is important that the author recognize what info the user may need and arrange to have it spoken (if possible). ARIA live regions or ARIA *activedescendents* may be one approach. Making content focusable via tabindex = 0 is another way. AxsJAX provides convenient utilities to assist the author in this task.

Though there is an ability to facilitate content to be read in *Control Mode* without using a screen reader [8, 19], users would lose many aspects of textual reading (slowing it down, going backwards and forwards, and going word-by-word, or letter-by-letter). For example, in an email client, switching folders, iterating over emails, opening emails, and switching to compose email should be accessed via control mode while *reading* an email should be accessed in *Reading Mode*. During *Control Mode* systems such as AxsJAX[8] about current actions and location in the content. We illustrate our breakdown in Table 4 by presenting an amalgam of high-level interaction concept (Table 1) along side specific functionality (Table 2, Table 3) associated with each screen reader mode (Section 5.1).

7.3 Synchronizing with Virtual Buffers

While the screen reader's virtual buffer is necessary for natural Web content navigation via screen reader keyboard commands, resolving the problems that have arisen due to dynamic Web content is a large problem (**6.3 & 6.4**). Much of this burden lies with advancing the screen reader itself. A common workaround is to use keyboard commands to force the screen reader to update its virtual buffer. Additional research has also examined dynamic content change detection systems [5]. Regardless, this is a serious problem, one that screen reader developers are working on (and been fixed for several AJAX situations in their newer versions). Additional supports for dynamic changes within widgets are provided through ARIA, however many dynamic updates cannot be fully described and relayed given the current standard.

Nevertheless, mitigating the virtual buffer remains a thorny issue in the support of AJAX applications. Until most deployed screen readers address this concern, Web developers must strive to at least alert screen reader users to content changes, and possible synchronicity issues. Workarounds to force buffer updates [17], present temporary solutions for developers.

7.4 Use of Custom Controls

While use of custom controls is beneficial to the look-and-feel and performance of Web pages, designers need to take screen reader users into consideration (**6.5**). ARIA provides additional support for complex UI widgets and controls. As this markup becomes prevalent and screen readers integrate support, existing custom controls will become accessible. However, as new forms of interactions are developed, standards like ARIA will also need to evolve. Even today, some UI widgets involve structures or interactions that cannot be described by existing ARIA language.

Until appropriate mark-up is provided, designers must explore other forms of accessibility support. As a rule of thumb, if a Web 2.0 designer is going to make content that visually looks/ functions like a standard Web element (e.g., link, button), then additional features must be provided for screen readers. With many of these features, keyboard based interaction is easily available. For example, decorated links can be made accessible in three steps:

1. Ensure that links have a keyboard listener and provide the same functionality on pressing Enter as for a mouse click.
2. Provide an ARIA role="link" on each of the decorated links for browsers and screen readers that supports ARIA.
3. Finally, add them to tab order by providing the attribute tabindex="0". This attribute is supported for non-interactive elements in Internet Explorer, Firefox, Opera and newer versions of WebKit.

Sadly, these small solutions are often not applied by web developers. For more complex custom content, providing mechanisms for navigation and feedback is essential. The PowerKey support in AxsJAX [8] provides a mechanism for defining and managing command line interfaces to application functionality. Designers can use this approach to enable easy and fast keyboard access for visually impaired users custom controls. PowerKey has been used to improve keyboard access on pages such as Google Health and Craigslist.

7.5 Illustration of Workflow: Interaction with Web 2.0 Content

We return to our fictional screen reader user, Alice, who is interacting with a Web based email client that follows the design requirements outlined in Section 7. Table 4 provides a complement to the narrative by illustrating her "mode" of use. We find Alice with her Web browser open, pointed at the Web 2.0 version of her Web based email client. This scenario is based upon technological solutions implemented in, and an amalgam of observations of users in the experimental context [15].

7.5.1 Reading Page Content (Navigating Text)

When the page loads, the screen reader in *Reading Mode* (VC) begins reading the entire page content. Alice wishes to immediately find out what new emails she has in her inbox. To iterate over the inbox content, the Web Application uses two hot-keys, J and K. Alice switches her screen reader to *Control Mode* (PCM), and cycles through her inbox. As she iterates, the Web app provides feedback to the user via AxsJAX, informing Alice of each sender, subject, and date. When she finds the emails she wants, Alice presses the Enter key, and she is provided audio confirmation of the dynamic content refresh (by AxsJax, ARIA support, or other solution). Alice switches to *Reading Mode* (VC) and reads the content of Eve's email asking Alice to contact their finance director Bob, and ask him for the financial reports.

7.5.2 Constructing An Email

To contact Bob, Alice returns to *Control Mode* and presses the C hot key. As the DOM gets updated, Alice is provided with Audio confirmation that the page loaded, and that she is a field of the form for composing a message. Alice fills out her email. She TABs to the send button and sends her email. When the page reloads, she is alerted to the DOM change. Should she wish to read the page, she can switch to VC. However, she is already in *Control Mode*, so she is easily able to press the application hot key I to jump to the inbox.

8. CONCLUSION & FUTURE WORK

A critical aspect of increasing accessibility is a better understanding of screen reader user needs. As Web technology continues to evolve, content designers must remain vigilant and keep accessibility in mind. This paper presents three models which provide a new understanding of how technology should be designed: 1) *interaction with traditional Web content* shows the mental and use models current screen reader users, 2) *interaction with Web 2.0 applications* illustrates difficulties inherent in screen readers interacting with AJAX technology, 3) *user workflow design* is a model for Web application designers to follow that minimizes the strain on model switching for screen reader users. In addition to these three models, we present a set of design guidelines to facilitate improved accessibility in a Web 2.0 world, based on model analysis and feedback from user research. While much work is being conducted on improving screen reader

technology to address the changing needs of users, this paper focuses on changes that can be made by designers. By focusing on authoring design implications, improvements in accessibility can be made for all screen reader users today.

While any new application disturbs the mental model of users, the disruption to screen reader users is even greater than the standard disruption, because of the role that the screen reader plays as an intermediary between the user and the content as rendered in the browser. Thus this paper articulates what form that additional disruption takes, and suggestions to alleviate it.

In the future, we look to continue to update these design guidelines to reflect the evolving state of Web technology. Further work can also improve Web development tool kits and design new authoring solutions (e.g., ARIA [27, 28], AxsJAX [8]) to provide developers with easy-to-integrate solutions. In addition, a rigorous study of the application of these design implications is being investigated in order to evaluate the result.

By basing our findings directly on feedback from users, lessons from designing Web 2.0 applications, and an analysis of screen reader technology, we believe that our conclusions are grounded in a holistic understanding of both the technology and the target users. There are many technologies rapidly coming together, and simply pulling them together does not solve the problems of screen reader users. Designers must also consider the user's work model and the limitations that the technology imposes. Through the models and implications for designed outlined here, we believe Web 2.0 applications can be updated to meet the needs of over 161 million visually impaired users.

9. ACKNOWLEDGMENTS

We would like to thank all of our participants, Sensory Access Foundation, VISTA Center, and Lighthouse for the Blind.

10. REFERENCES

[1] American Foundation for the Blind. Facts and Figures on Americans with Vision Loss. http://www.afb.org/Section.asp?SectionID=15&DocumentID=4398

[2] Becker, J. V., Becker, D. A., Hinton, D. E. and Hugh G. Anderson, J.Braille computer monitor(6700553). Unites States Patent Office, USA 2004.

[3] Bigham, J., Cavender, A., Brudvik, J. and Wobbrock, J. WebinSitu: A Comparative Analysis of Blind and Sighted Browsing Behavior. Proceedings of ASSETS 2007 (Tempe, AZ), 2007.

[4] Bigham, J. P. and Ladner, R. E. Accessmonkey: a collaborative scripting framework for Web users and developers. Proceedings of W4A. Bamf, Canada). ACM, 2007.

[5] Borodin, Y., Bigham, J. P., Raman, R. and Ramakrishnan, I. V. What's New? - Making Web Page Updates Accessible. In Proceedings of ASSETS 2008 (Halifax, Nova Scotia), 2008.

[6] Boyden, C. and Greco, L. C21. Web Usability for Assistive Technology. Procedings of CHI 2007 (San Jose, CA), 2007.

[7] Caldwell, B., Cooper, M., Reid, L. G. and Vanderheiden, G. Web Content Accessibility Guidelines 2.0. 2008.

[8] Chen, C. L. and Raman, T. V. AxsJAX: a talking translation bot using google IM: bringing Web-2.0 applications to life. In Proceedings of W4A (Beijing, China). ACM, 2008.

[9] Freedom Scientific. JAWS. 2007

[10] Freedom Scientific. JAWS for Windows ® Screen Reading Software. http://www.freedomscientific.com/products/fs/jaws-product-page.asp

[11] Fukuda, K., Saito, S., Takagi, H. and Asakawa, C. Proposing new metrics to evaluate Web usability for the blind. In CHI '05 extended abstracts (Portland, OR). ACM, 2005.

[12] Garrett, J. J. Ajax: A New Approach to Web Applications. adaptive path, February 18, 2005. http://www.adaptivepath.com/ideas/essays/archives/000385.php, 2005.

[13] Google. Keyboard shortcuts - Gmail Help Center. http://mail.google.com/support/bin/answer.py?hl=en&answer=6594

[14] GW Micro. Window-Eyes. 2007

[15] Hailpern, J., Guarino Reid, L. and Boardman, R. DTorial: An interactive tutorial framework for blind users in a Web 2.0 world. UIUC Technical Report #UIUCDCS-R-2009-3029.

[16] Harkin, T. AMERICANS WITH DISABILITIES ACT OF 1990. U. S. Senate, 1990.

[17] Juicy Studio. Improving Ajax applications for JAWS users. http://juicystudio.com/article/improving-ajax-applications-for-jaws-users.php

[18] Kim, J. W., Candan, K. S. and Mehmet E. Dnderler. Topic segmentation of message hierarchies for indexing and navigation support. Proceedings of WWW 2005 (Chiba, Japan). ACM, 2005.

[19] Lemon, G. WAI-ARIA Live Regions. http://juicystudio.com/article/wai-aria-live-regions.php

[20] McKeon, H. P. The Rehabilitation Act Amendments (Section 508). U. S. Congress, 1998.

[21] Miyashita, H., Sato, D., Takagi, H. and Asakawa, C. Making multimedia content accessible for screen reader users. In Proceedings of the W4A Conference (Banff, Canada, 2007). ACM, 2007.

[22] O'Reilly, T. What is Web 2.0 (2005). http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-Web-20.html

[23] Takagi, H., Asakawa, C., Fukuda, K. and Maeda, J. Accessibility designer: visualizing usability for the blind. In Proceedings of the ACM SIGACCESS conference (Atlanta, GA). ACM, 2004.

[24] Thatcher, J. SAID (Synthetic Audio Interface Driver). 1984

[25] Thatcher, J. Screen reader/2: access to OS/2 and the graphical user interface. Proceedings of Assets '94 (Marina Del Rey, CA). ACM, 1994.

[26] W3C. HTML 4.01 Specification - IFRAME. http://www.w3.org/TR/html4/present/frames.html#h-16.5

[27] W3C and Editor: Schwerdtfeger, R. Roadmap for Accessible Rich Internet Applications (WAI-ARIA Roadmap). http://www.w3.org/TR/wai-aria-roadmap/

[28] W3C - Web Accessibility Initiative and Editor: Henry, S. L. WAI-ARIA Overview. http://www.w3.org/WAI/intro/aria

[29] World Health Organization. WHO | Magnitude and causes of visual impairment. wttp://www.who.int/mediacentre/factsheets/fs282/en/