

An Alternative Audio Web Browsing Solution:
Viewing Web Documents Through a Tree
Structural Approach

Esmond Walshe B.Sc.
School of Electronic Engineering
Dublin City University

A thesis submitted for the degree of Ph.D.
September 2006

Supervisor: Dr. Barry McMullin

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Ph.D. is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

ESMOND WALSH

Signed: _____ (Esmond Walshe)

ID Number: 50162071

Date: 21 September 2006

Abstract

This thesis examines methods to aid in the non-visual browsing of web based documents, primarily using synthetic speech output. The serial nature of speech ensures that it is a difficult medium in which to browse documents. Much of the structure implied in the visual appearance of the content is not available through speech. Only a narrow region in the content is perceivable at any given time, and it can be difficult to navigate to the important segments of the document. This is in contrast to visual interaction, where cues such as changes in font or colour establish contextual changes in the content and guide the user accordingly.

A number of browsing/navigation strategies are presented to offset these problems. These are implemented through *WebTree*. This is a highly customisable web browser which renders documents through a dynamically expandable tree structural view. This mirrors the arrangement of mark-up elements in the source file. Contextual information about each element is provided as appropriate. By expanding and collapsing individual tree elements, the user decides how much of the content is rendered at any given time. The user can also determine whether a certain element is automatically expanded in the rendering when encountered, or whether it appears in the rendering at all, effectively allowing for the easy generation of alternative document views. To speed up navigation the user can move through the document based on the element of their choice. Specialised element search functions are also available. Additional navigational functionality is provided to deal with the specific requirements of `<table>` or `<form>` elements.

The thesis reviews different methods previously employed to offset problems with auditory interfaces and compares these with *WebTree*. Initial user tests and evaluations of *WebTree* are presented, which show that the approaches taken provide a viable solution, particularly for the browsing of large or complex web-based documents, by blind users.

Acknowledgements

Though this work bears the name of a single author, I wish to acknowledge all those whose contributions made it possible. First and foremost, I wish to sincerely thank my supervisor, Prof. Barry McMullin, who had to put up with listening to all my ideas, both good and bad, and my missed deadlines. He helped me crystallise many of the ideas found in this thesis and provided many suggestions on how the system could be improved.

I wish to express my gratitude to Dr. Klaus Miesenberger and Dr. Derek Molloy who had the tedious task of examining this thesis. I would like to thank them for being both thorough and fair with their comments and for their suggestions on possible directions in which the research could progress in the future.

I would also like to express my gratitude to the members of the user group who tested the application. The time they spent using the system provided much insight into whether the system was a usable approach to web page navigation. I would also like to thank the following people, Bryan Walshe, Derrick Walshe, Joan O'Malley, Donal Fitzpatrick and Ciaran Kelly for their help in proof reading this thesis and for providing comments on how it could be improved. I would also like to thank the other members of the access group and members of the "Alife" lab for acting as a screen reader when my computer either crashed, or the application I needed to use was inaccessible. Also thank you to my many friends and colleagues for putting up with me for the last while.

The work described here received financial support provided from AIB PLC¹. The work was carried out in the Research Institute for Networks and Communications Engineering (RINCE), established at DCU under the Programme for Research in Third Level Institutions operated by the Irish Higher Education Authority.

¹<http://www.aib.ie/>

Finally I would like to thank my family for all their support throughout the many years I have spent in education. They always strived to ensure I received the best education possible and encouraged me to do the best I could. I owe them a debt that probably cannot be repaid. Throughout the years, my three brothers and one sister read large amounts of material for me that I did not have access to in electronic form. So Mam, Dad, Kenneth, Derrick, Bryan and Sandra — Thanks!

Contents

1	Introduction	1
1.1	Setting Out the Problem	1
1.2	Interaction Modalities	4
1.3	Navigation and Browsing	6
1.4	WebTree	8
1.5	Thesis Structure	11
2	Speech Output and Non-Speech Auditory Cues	14
2.1	Spoken Output	15
2.1.1	Synthetic Speech	21
2.1.2	Comprehension of synthetic speech	25
2.1.3	Voice Intonation and Prosodic Cues	27
2.2	Tactons, Earcons, Auditory Icons and General Non-Speech Sound Cues	32
2.2.1	Tactons	32
2.2.2	Non-speech audio cues	33
2.3	Summary	40
3	Literature Review	41
3.1	The Graphical User Interface (GUI)	42
3.2	Manipulating the Audio Stream to Provide Contextual Infor- mation	49
3.2.1	Using Non-Speech Cues to Denote Structure	54
3.2.2	Using Voice Changes to Denote Structure	57

3.3	Document Modelling	63
3.3.1	Modelling Document Structure	67
3.3.2	Conceptual Graphs	70
3.3.3	The Document Object Model (DOM)	71
3.4	Current Non-Visual Web Browsing Solutions	74
3.4.1	Conventional Browser with a Screen Reading Application	74
3.4.2	Transcoding Proxies	76
3.4.3	Dedicated Browsing Solutions	80
3.5	Web Page Navigation and Summarisation	83
3.6	Tabular Constructs	86
3.6.1	Cells Spanning More than One Row or Column	94
3.6.2	Providing Contextual Information During Navigation	96
3.7	Form Handling	97
3.8	Tree Navigation	104
3.9	Summary	107
4	WebTree Implementation	108
4.1	Emacs Terminology	108
4.2	WebTree Specification and Design	110
4.3	Implementation Environment	114
4.4	Document Modelling	122
4.5	Exposing Document Tree Structure	124
4.5.1	The Widget Tree	126
4.5.2	Screen Overlays	132
4.6	Cascading Style Sheet implementation	133
4.7	Standards Compliance	135
4.7.1	Guidelines	136
4.7.2	Validation	146
4.7.3	Accessibility Levels and Legal Requirements	148
4.8	Summary	152

5	WebTree User Interface	153
5.1	User Interface	154
5.1.1	Primary Document View	156
5.1.2	Customisation	165
5.1.3	Generating Alternative Document Views	167
5.1.4	Keyboard Navigation and Interaction	169
5.1.5	Searching	170
5.1.6	Rendering Tabular Data	173
5.1.7	XHTML Form Handling	178
5.2	Auditory Output	180
5.2.1	Tree Interaction	184
5.2.2	Speaking Form Data	187
5.2.3	Speaking Tabular Data	189
5.3	Reading Strategies	191
5.4	Braille Output	193
5.5	Summary	193
6	User Evaluation	194
6.1	Test User Profile	194
6.2	Methodology	196
6.3	Formal Questionnaire Results	200
6.3.1	Results of the After Scenario Questionnaire	200
6.3.2	Results of the Computer System Usability Questionnaire	200
6.4	Evaluation results summary	201
6.4.1	Interactive Form Data	208
6.4.2	Searching	209
6.4.3	Tabular Data	210
6.4.4	Customisation Issues	212
6.4.5	Usability Versus Page Complexity	213
6.5	Conclusion	215

7	Conclusions and Further Work	217
7.1	Further Work	221
7.2	Contributions of this Research	222
	References	224
A	WebTree Application User Manual	239
B	WebTree Tutorial	248
C	Additional User Evaluation Questionnaires	254
C.1	Preliminary Evaluation Questionnaire	254
C.2	After Evaluation Questionnaire	256
D	Braille	263
D.1	Brief Description	263
D.2	Louis Braille: Biographical note	265
D.3	Braille Standards	266
D.4	Braille Reading and Usage Levels	268
D.5	Issues with the communication of highlighting and other vi- sual cues through Braille	272
D.6	Braille production	274
D.6.1	Refreshable Braille Displays	276

Chapter 1

Introduction

This chapter is intended to outline the basic ideas underlying the design of the WebTree document browser. It points out the origins, primary aims and considerations of the research. It also provides a brief overview of some of the key concepts used within the system. A short description of the modalities in which blind people interact with their computers is presented. The final section provides an outline of the structure of this thesis.

1.1 Setting Out the Problem

The vast quantities of electronic information available through the world wide web has resulted in access to substantial amounts of written material previously inaccessible to blind individuals. Performing transactions online, such as shopping or filling out application forms, has become very popular over the last number of years. In the case of blind people, this has resulted in greater control over aspects of their everyday lives. It can reduce their dependence on sighted assistance to complete daily tasks. Purchasing goods through an ecommerce website is often convenient for both blind and sighted people alike. Nevertheless, it can have greater resonance for blind individuals. Many of the physical barriers associated with navigating large department stores and with the viewing and selection of products can be alleviated through shopping in this manner. However, for a blind person to be

able to take full advantage of these possibilities, websites must be marked up in accordance with accessibility guidelines. For example, the Web Content Accessibility Guidelines (WCAG) W3C (1999*b*), produced by the W3C's Web Accessibility initiative (WAI)¹. However, only a subset of the WCAG guidelines is applicable to web access for blind people. (See section 4.7) for more details. Once a web page meets a certain level of accessibility criteria, blind people can navigate through and gain access to the relevant information. This is achievable using either a dedicated audio browser or with screen-reading software interfacing with a mainstream visual browser.

Many web access solutions for the blind function as add-on auditory interfaces to existing visual browsers. However, there are major differences in the interaction methods used to operate dedicated audio browsers in comparison to their visual counterparts. The visual reader is rather adept at assimilating large amounts of information at the same time. The human eye is expertly capable of quickly scanning through the document to establish those items deemed as being important page content. This is achieved through examining the spatial organisation of elements and through visual cues, e.g., changes in font size, colour and emphasis. For example, many website developers position navigation links to the left of the main content in multi-columned displays. Also, a section heading may have a different colour or font size, making it stand out from the surrounding text. With one quick glance, the visual reader can often isolate the starting point of the required content and re-adjust focus accordingly.

Unfortunately, much of the spatial organisation is lost when using an auditory browser. Due to the serial nature of access technologies for the blind, only a single point in the audio stream is viewable at any given time. Therefore, it is difficult to establish the page structure without examining the entire page. The content is presented in a sequential manner, causing problems for multi-column presentations. Reading a line of text based directly on its screen position would include all of the columns in the order in

¹<http://www.w3.org/wai>

which they appear. For this reason, many access solutions reformat the content and present it linearly, for example, JAWS for Windows², often hiding the spatial organisation from the listener. See section 3.6 for more details on the types of transformations performed on tabular material. A number of web access limitations experienced by blind people are due to the narrow focus imposed by this form of interaction. In the case of data tables, it might be difficult to see how each cell relates to the neighbouring cells in the grid. This is especially the case when linking the cell with any relevant header information, whether it be a row or column header. Thus, increasing comprehension complexity when assimilating such content. When a tabular construct is employed to visually organise the information for purely aesthetic purposes, other problematic issues surface. Presenting the content in a linear format prevents the problem of listening to content from multiple columns when reading a line. However, it does mean that the user must trawl through the content of each column to find the required data. Take for example a two column display. If the left hand column contains navigation links, with the second column containing the main content, the user must navigate passed these links before reaching the required material. Also, if the content in each column is divided into a number of cells, then different fragments of each column would be presented in sequence. This ensures that a continuous read function would read fragments of both columns instead of one continuous column.

Similarly, the use of visual cues, e.g., adjustments in colour or font size to depict contextual changes can pose serious difficulties for auditory interfaces. For each visual cue, the amount of possible variation in text formatting parameters is quite large. Consequently, combining two or more such cues ensures the number of possible format variations is enormous, while the scope for mapping these changes to the audio output is rather limited. Alternative voices or changing voice parameters such as pitch or stress can signal the presence of these cues. However a number of problems exist. See

²http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

section 3.2.2 for a detailed discussion of these problems. The number of available voices can vary greatly from synthesiser to synthesiser.³ Therefore, the voice selected to portray certain information may not be available. An analogous problem does exist for visual formatting: any given user will have only certain fonts installed, and these may not include some fonts specified by a page author. Also, if certain voice parameters are altered by too great an extent, the user can become distracted with the changes, reducing levels of content comprehension (James, 1998). Likewise, if the changes are too subtle the listener may have difficulty recognising adjustments to the output. Therefore mapping these changes to the intended meaning would be extremely difficult. For these reasons, modelling document interaction using a purely audio browser by directly mapping functionality to its visual counterpart is not generally satisfactory.

1.2 Interaction Modalities

Braille and speech output are the two primary media in which blind people can gain access to electronically stored documents. Braille is a tactile method of encoding written material dating back to the early 19th century. No attempt is made to mimic the shape of printed letters. Instead, characters are fashioned using up to six raised dots, positioned in two vertical columns of three. The dots are numbered in sequence from one to six. The top left position is known as Dot 1, while dot 6 is situated on the bottom right. Characters are defined by raising one or more dots in various combinations. For example, **a** is represented by dot one, and **g** contains dots one, two, four and five. In terms of computer interaction, access can be gained through a specialised piece of hardware known as a Braille display, where up to 80 characters are often available. Braille displays and their uses are discussed in section D.6.1. For a more detailed review of Braille and its

³A voice synthesiser is a software program or a mechanical hardware device which generates an approximation of human speech. They are often referred to as “text to speech engines”, for they frequently take a string of text as input and convert it to speech output. For more details see section 2.1.1.

usage, see appendix D.

Alternatively, the user can choose to have information presented through synthetic speech. Although there have been many advances in the quality of synthetic speech, it is still less intelligible than natural speech. Therefore, it can be difficult to listen to lengthy fragments of content, especially if the subject matter is complex. Users can get bored with the presentation due to the monotonous quality of the voice. Also, problems can be caused by poorly timed prosodic boundaries or inaccurate intonation inflections. However, with a little training, a user can read documents at very high speed, using this method. See section 2.1 for a more in-depth discussion of speech.

Both Braille and speech output suffer from similar limitations when presenting computer-based content. This is due to the narrow time-frame in which content is portrayed. Speech output is both sequential and transient, thus only a small segment of the content is in view at any time. Whereas, with Braille, the size of the display is the limiting factor. Although up to 80 characters are sometimes available, screen reading software has to take into account smaller displays. Thus, a physical line of text on the screen does not always equate to what is presented in the display. This narrow view ensures that the reader does not automatically see the page structure. Instead they must examine the entire page to establish such information. Consequently, they cannot easily jump to the important content when viewing pages whose structure is previously unseen.

Both media pose difficulties for the presentation of contextual information provided by visual cues. For example, changes in font, or emphasis. Announcing the presence of such content in the vocal output would increase dramatically the verbosity of the presentation. Therefore, as mentioned above, alternative voices or changes in voice parameters could be used to portray this information. However, with Braille the capacity for presenting such data is extremely limited. There is not much scope for defining new Braille characters to denote contextual cues, discussed in section D.5. For

these reasons, the use of sound cues to complement the primary method of output is often employed. For a description of the types of sound cues used, see section 2.2.

These sound cues can be abstract musical sounds (earcons) (Blattner et al., 1989) or every day sounds whose meaning is taken from their role in the natural environment (Gaver, 1986). There is a learning curve required to associate these cues with the meaning they are intended to portray. However, once learned they ought to be able to unobtrusively signal contextual changes to the listener. In terms of browsing web pages, the presence of a given element, such as a header `<h1>` could be signalled by a sound cue. In chapter 3, the role of sound cues in terms of previous work in the area of browsing marked up documents are discussed.

1.3 Navigation and Browsing

Listening to an entire document through a text to speech engine (TTS) could be described as being akin to listening to a pre-recorded spoken version of the content. As with digital recordings, an ability to navigate to and focus in on small segments of the document for closer scrutiny is imperative (Arons, 1997). Many screen reader applications provide navigation facilities to manoeuvre line by line, or navigate through the content relying on larger chunks of text. For example, paragraphs or virtual pages based on the amount of screen text available. When viewing web-based documents, additional navigational functionality based on a subset of mark-up tags is sometimes provided. For example, manoeuvring to the next table construct or list entity. However, without traversing the entire document, it is difficult to cultivate an accurate impression of the document's structure usually apparent to a sighted user with a single glance.

To circumvent these issues, many screen reader developers equip their applications with a page summarisation mechanism. These summaries are frequently based on the presence of specific named elements such as headers, hyperlinks, or by previously bookmarked page segments (Zajicek and Powell,

1997; Zajicek et al., 1998a). Page summaries based on the first line/sentence within a paragraph, or views containing entire paragraphs comprising specified words or phrases have also been attempted. See the document: Surfing the Internet with JAWS⁴, for more details. An alternative approach would be to examine the sentence structure of the text and generate page summaries of sentences containing the most frequently used word trigrams (Zajicek et al., 1998a).⁵ Unfortunately, this method is rather error-prone, for trigrams less pivotal to comprehending the page contents may feature greatly in the summary, whilst less used more explanatory sentences are excluded.

Other approaches to the presentation of web-based information include Parente's Audio Enriched Links system (Parente, 2004), and the Hearsay system (Ramakrishnan et al., 2004) for browsing hypertext documents through audio. The audio enriched links mechanism provides a spoken preview summary of a linked web page, before the link is followed by the user. The page summary is comprised of its title, its relation to the current page, statistics about its content, and some highlights from its content. The Hearsay system (Ramakrishnan et al., 2004), attempts to automatically partition Web documents through tightly coupled structural and semantic analysis. It transforms raw HTML documents into semantic structures to facilitate audio browsing. Voice XML dialogs are automatically produced from the XML output of partitioning. See section 3.5 for a description of the literature concerning page summarisation.

The major difficulty with many of the current approaches employed to facilitate both internal document navigation and page summarisation, is the tendency to only provide such functionality in relation to a subset of named elements prescribed by the application developers. A better approach would be to encourage the user to perform element searches or create document

⁴<http://tinyurl.com/rqzu4>

⁵Zajicek et al. (1998a) uses the word "trigrams" to denote a three word key phrase. By extracting sentences containing the most frequently found trigrams, a page summary based on the content can be generated.

renderings based on their individual requirements. Under the WebTree system, many of these restrictions are removed.

1.4 WebTree

WebTree has been developed to explore the use of an alternative, highly customisable tree structural approach to the auditory rendering of web based documents (Walshe and McMullin, 2004, 2006). It provides alternative page summaries based on user-selected components of the underlying mark-up. The user dynamically controls how much of the document's tree hierarchy is to be exposed on a (virtual) screen at any given time. Thus, entire element sub-trees may be efficiently traversed with minimal difficulty. The primary goal of this research was to establish any possible advantages that can be associated with this approach. The major concern is the effect this methodology may have on the efficiency at which the user can navigate to and assimilate information.

Although the current version of the system is optimised for XHTML web pages (W3C, 2002*b*), the methods should be applicable to any document organised in a tree-based structure. The system stores the parsed document in a DOM (Document Object Model) tree structure for internal manipulation (W3C, 2004*a*). Both the navigation and display functions interface with the DOM structure to manipulate the document to the user's specifications. Therefore, once issues resulting from the accurate parsing of these documents are offset, the viewing of these pages should not be problematic under WebTree. It should be noted that JavaScript⁶ or Adobe Flash⁷ components when included in web pages are ignored by the current prototype system.

The WebTree system attempts to use the mark-up structural elements employed in the document's make-up to determine how the content is to be portrayed. Decisions on how much content is to be presented and the amount

⁶<http://www.ecma-international.org/publications/standards/Ecma-262.htm>

⁷<http://www.adobe.com/products/flash/flashpro/>

of style information to be associated with this material are bestowed on the user. Through the customisation facility, discussed in section 5.1.2, the user can decide which elements have their content automatically appearing in the display. Thus, acting as a mechanism for generating alternative views of the same document. Similarly, style information to be attributed to the individual elements can be gleaned from an aural style sheet. See section 4.6 for more information.

The main focus of the WebTree system is to afford the user the ability to create alternative audio renderings of a given document. By using the expand/collapse functionality the user is intrinsically creating custom summaries of the content. In addition, the ability to exclude entire element subtrees allows for a powerful method for creating page summaries. The manner in which WebTree's summarisation functions differ from those of current solutions is that they are not limited to a set of elements named by the developer. That is, the user selects the elements on which the rendering is based. Also, summaries can be generated using a group of prescribed elements as opposed to the conventional approach which generates a list containing only a single kind of element, e.g., a list of links or header elements.⁸ JAWS provides a skim read feature that allows the user read a document using the first line or sentence in a paragraph. Functionality to narrow the scope to only those paragraphs satisfying prescribed text rules is also offered.⁹ Users can create a summary of this content, so that they can browse and quickly link to the relevant page segment. Again this appears to be limited to only contain a single element type, i.e., <p> elements. Whereas WebTree also has the ability to show a segment of a paragraph to provide an indication of its content type, and the summaries in WebTree can contain multiple element types.

The user interacts with content rendered by the WebTree application

⁸With JAWS, a list of form elements can be created to include all form control types, e.g., buttons, text entry fields, etc.

⁹Text rules can also be applied to sentences and lines to create summaries, depending on the user's preferences.

through a character-oriented virtual screen/display. By navigating through the portrayed material, the user builds up a mental model of both the type of content being presented and the structure in which the different elements are organised. By expanding and collapsing segments of this structure, the user can gain access to the desired material. There exist three major static points of reference for the user to guide them during navigation. These include the left margin and both the beginning and ending points of the document. All other points are subject to change. The virtual screen width (right margin) is governed by variables stored in the customisation facility. Potentially, a line may contain hundreds of characters, depending on the users' preferences. Navigation within this rendering can be achieved by manoeuvring in either direction character-by-character, word-by-word, line-by-line, or through additional navigational functionality described in chapter 5.

As well as the tree-like arrangement of the displayed mark-up elements, The WebTree application has been augmented with some rather powerful methods to search for content. These alternative methods were included so that a number of different navigational approaches could be experimented with. As well as searching for plain text, the user can search for an individual mark-up element, or limit the text search to only find instances of the search string occurring within specific mark-up elements. Thus if a document is properly marked up with structural elements, such as `<h*>` and `` elements, instead of having their visual appearance created with changes in font, the user should also be able to navigate to the areas of the page that the author deemed important. All elements are allowed as targets for the different search facilities.

The initial user evaluations demonstrate that viewing documents through the WebTree system is a viable alternative to the linear approaches often employed by traditional audio browsers. The advantages of the system were most evident whilst viewing large documents, for entire element subtrees could be bypassed by the reader. This resulted in easy navigation to

the appropriate document segments.

Although the application would depend somewhat on web pages meeting a certain level of accessibility requirements, the WebTree system should increase the user's productivity when viewing such documents. Despite the low numbers of websites in Ireland embracing web accessibility guidelines, such as the W3's Web Content Accessibility Guidelines (WCAG), there may be a reason for a dramatic increase in the number of pages meeting these standards in the near future. The draft code of practice¹⁰ under the "Irish Disability Act 2005" explicitly recommends that public sector websites validate to WCAG level double-a compliance. If adopted, this code of practice will place the requirement for public sector websites to meet WCAG level double-a compliance on a legal footing, therefore the number of pages meeting this requirement should grow substantially.

1.5 Thesis Structure

Chapter 2 presents the different mechanisms for computer interaction used by blind people to gain access to electronically stored documents through a purely auditory modality. The first section of this chapter is dedicated to a discussion of speech output. Issues pertaining to both synthetic speech production and natural speech recordings are illustrated. A review of the literature concerning the intelligibility of synthetic speech in comparison to natural speech is provided. Also included is a brief outline of the effects of prosodic cues in a continuous speech stream. In the second section, the discussion focuses on the different types of non-speech sound cue and the literature concerning research into their usage.

Chapter 3 contains a review of the literature concerning the use of these technologies to portray information to the blind. Research into audio browsers for purely non-visual access is discussed at length. The usage of alternative voices, changes in speech parameters and the use of audio cues to announce contextual information are illustrated. Also presented are the

¹⁰<http://tinyurl.com/o3rdp>

current mechanisms employed to alleviate the sequential nature of the output. These include both additional navigational functionality based on the presence of specific mark-up elements, and methods employed to summarise the page content. A description of the presentation of interactive forms is provided. Also presented is a review of the literature concerning interaction with spatially arranged information i.e., tabular data under a purely audio interface.

Chapter 4 describes the implementation of the WebTree system. First the development environment is discussed, followed by a description of the mechanisms used for document modelling. After which the technologies used to expose the document tree structure are presented. Also, the relevance of the system to the presentation of documents marked up with other mark-up languages, such as XML (W3C, 2004*b*) or HTML (W3C, 1999*a*) is included. Finally, focus is centred on the standards and guidelines on which the system relies, and legislation in the area of web access for disabled users.

A description of the WebTree user interface is presented in chapter 5. The primary document view based on the exposure of the documents' underlying tree structure plus the mechanisms used to customise this view are discussed. Also included is a description of the facility to generate alternative document renderings. Next, the methods of interaction with the system are presented. These include keyboard navigation, and the methods of interacting with complex structures such as tables and interactive forms. Afterwards, the discussion features the spoken output interface. It illustrates how the different components such as tree structure, form data or tabular information are to be presented. Finally, a number of non-visual reading strategies and how they are facilitated under WebTree are described.

In chapter 6, a discussion of the user evaluation process employed to appraise the WebTree system is presented. Initially, a description of the test methodology is offered, followed by a detailed account of the test group's experiences using the application. A number of suggestions provided by members of the evaluation user group on how the interface might be im-

proved are also included. Finally, a summary of the findings of this thesis coupled with some ideas for future work are contained in chapter 7.

Chapter 2

Speech Output and Non-Speech Auditory Cues

This chapter describes a number of alternative mechanisms used to portray content to totally blind computer users through primarily auditory means. The content is organised into two major sections:

Firstly, the discussion focuses on accessing content through spoken output. Initially, a number of limitations with this approach are outlined, with several proposed solutions to reduce these effects. Next a brief history of the production of synthetic speech is presented, including a short overview of the modalities employed to generate this form of output. Subsequently, an analysis of the literature concerning the comprehensibility of synthetic speech is introduced. Finally, the discussion centres on the effects that speech *prosody* and *intonation* have on the comprehension of spoken content.

The second major section details the use of non- speech audio cues e.g., *earcons* and *auditory icons* to convey contextual information to the listener. The design of the different types of audio cues used, and a discussion of the literature concerning their usage is presented.

Although the subject matter dealt with in this dissertation primarily focuses on access to electronic content through auditory means, a discussion of the literature concerning human computer interaction for the blind would

not be complete without the provision of some information on the Braille output modality. For this reason, appendix D contains a detailed discussion concerning Braille and its usage.

2.1 Spoken Output

A number of methods to gain access to written material are available to blind individuals when the spoken medium is employed. These include: the provision of books on Audiocassette; enlisting help from a sighted friend or colleague to read aloud the relevant material; or employing a text to speech engine to read the content of electronically stored documents. Relying on a human reader is not always a practical solution and in many cases is an undesirable one. This is primarily due to issues pertaining to both privacy infringement and the reduction in independence this solution would impose. However, the alternative methods listed have their disadvantages too. There exists a number of problematic issues inherently linked to the serial nature of speech interaction, which must be overcome. This is the case for both pre-recorded speech and that, which is dynamically generated by a text to speech engine.

There exists much previous work in the area of presenting electronically stored content through the use of synthetic speech. The two main bodies of work referenced in this dissertation are those by Raman (1994) and Fitzpatrick (1999). These are referenced continuously throughout this thesis, so it is important to provide a brief overview of some of their main ideas here. However, a more in-depth description of these ideas can be found in section 3.2. The manner in which they internally modelled document structures in their respective systems are described in section 3.3.1. The research in question deals with how to non-visually render documents written in the \LaTeX mark-up language (Lampport, 1985). Although much of this work deals with the communication of mathematical constructs through spoken output, methods for notifying document structural and contextual cues are also offered.

The ASTER project as described in Raman (1994), experimented with assigning different voices to specific screen elements to notify both changes in structure, such as a header element, and to signify a change in context, for example, a piece of emphasised text. In addition, the use of audio cues to signal interface states is used. He describes a specialised *auditory formatting language* (AFL), which can specify how a given element can be rendered non-visually. Many of the ideas offered by Raman (1994) can now be found in the emacspeak¹ application, described in (Raman, 1996a,b).

Instead of using changes in voice to represent contextual changes, in his TechRead project Fitzpatrick (1999) offers alterations in speech prosody, such as changing the rate or pitch of the voice, as a possible solution. Take for example the case of sectional headers. The prosodic aspects of the voice are altered in order to convey the fact that the hierarchical level is changing. The rate is slowed by 10% to yield a slower, more measured utterance. The average pitch of the voice is decreased by 25% to distinguish the sectional title from the remainder of the text. He believed it is preferable to enhance some prosodic elements for the sake of intelligibility, than to achieve natural-sounding, though completely incomprehensible synthetic speech. For more information on the TechRead system, see (Fitzpatrick, 1999; Fitzpatrick and Monaghan, 1998, 1999; Fitzpatrick, 2002; Fitzpatrick and Karshmer, 2004; Fitzpatrick, 2006).

The Assimilation of information through the visual modality, and to a lesser extent through Braille interaction, is acknowledged as being interactive in nature. That is, the reader is reputed to be an active component in the reading process. Operating at a rate comfortable to the reader, the intended intonation that is implied by punctuation markers, grammatical inferences, and/or by highlighting cues is interpreted as of when they are encountered. The reader is able to determine the importance applied to text fragments through the analysis of such cues. Reviewing documents in an interactive manner ensures the user can easily limit the focus to a small

¹<http://emacspeak.sourceforge.net/>

fragment of text for a more in-depth examination. This is especially beneficial in cases where the subject matter is rather complex, e.g., examining scientific or mathematical material.

Fitzpatrick maintains that the printed page plays an important role in the comprehension of written material, by functioning as an external memory store throughout the reading process. He states in (Fitzpatrick, 1999, Chapter 2):

The manner in which the eye can relate to this external memory is a very powerful tool to aid in the retention and comprehension of written information. It can rapidly scan over the printed words, and by virtue of the juxtaposition of characters or symbols on the printed page, semantically interpret those symbols to produce the underlying meaning. Once the information is lost from the short-term memory, it can be easily refreshed by the rapid movements of the eye.

There are a number of steps involved when visually reading a document. A skilled reader will normally read at a rate of 250–300 words per minute. The visual reading task does not normally entail the human eye positioning focus at the beginning of the page, and sequentially moving through the content until the end is reached. Instead, the reading process consists of several distinct movements. Stevens (1996) tells us that the eye performs a number of tasks in order to obtain informational input. The reading process can be broken down into a series of *saccades* (jumps) and *fixations*:

The saccades move the point of fixation in accordance with how much information has been or can be apprehended. Forty nine percent of reading time is taken up with fixations. The rest of the time is taken up with the selection of which portion of the text to next fixate and the move to that location.

In contrast to visual reading, assimilating information through the spoken medium is said to be rather passive in nature. The reader does not ap-

pear to be an active component in the reading process. Instead, the material in question is said to flow sequentially past the listener (Fitzpatrick, 1999). The listener is not afforded full control over the content presented. That is, they are dependant on the reader's interpretation of prosodic boundaries and voice intonation effects, to denote contextual meaning. Poor intonation cues, or wrongly positioned prosodic inflections can render the content difficult to absorb. Although this is sometimes a problem found in human voice recordings, it is much more frequently a feature associated with synthesised speech output. See section 2.1.3, for more details.

The external memory function performed by the printed page is not available when listening to a spoken rendition of the content. Therefore, greater demands are placed on the reader's short-term memory functions to comprehend and assimilate subject matter in this form. This is due to its transient nature, for only a single point in the content can be in focus at any given time. Therefore, additional storage of the material in short-term memory must take place.

According to Shneiderman (2000), the section of the human brain that is responsible for the cognition of speech stimuli is also responsible for problem solving. Solving tough problems is best achieved when attempted in quiet environments, away from speech interaction. The accomplishment of routine physical activities and the interpretation of hand to eye coordination are processed by separate components of the brain, therefore, whilst one is engaged in such activities, e.g., typing on a keyboard, or navigating through the on screen elements with a mouse, the problem solving processes are not adversely affected. In short, humans can successfully walk and think, but find it difficult to think whilst processing speech information at the same time (Shneiderman, 2000). This ought to be a major consideration when presenting complex information through speech output. The user is expected to process speech recognition while memorising the content being presented. This is before any interpretation of the material takes place.

The temporal and transient nature of speech interaction ensures that

it is a difficult medium in which to browse content. Especially when compared to its visual counterpart. Take for example, the possible frustration experienced when searching a lengthy recording for a specific passage of text, e.g., a book stored on an audiocassette, or a digital recording stored electronically. Regardless of the storage mechanism used, the predicament is identical. Fast-forwarding/rewinding sequentially through the recording will eventually find the specified excerpt, however this method can be both slow and cumbersome.

The development of large digital libraries of pre-recorded speech content has resulted in the need for much research into finding powerful and reliable audio search methods. Allowing multiple methods for content indexing negates the need to sequentially search recordings for a given passage. Separating content into segments based on time index i.e., a specified number of seconds/minutes, is not an adequate means for indexing recorded text. It can often bear little resemblance to the contextual breaks within the spoken utterance. Thus, additional methods for audio indexing needed to be established.

The indexing method proposed by Arons (1997) analyzes pause and pitch structural cues in an attempt to establish changes in context. Arons (1997) maintains that both the lengths of the individual pauses, and changes in voice pitch, can be used as reasonably accurate indicators to the natural segmental boundaries in the content. By analysing these prosodic and intonation cues, attempts are made to determine the starting points of structural blocks, such as sections, paragraphs, or the beginning of a new topic, to which focus can be positioned. According to Arons (1997), other researchers have investigated indexing speech recordings through segmenting content based on articulation by different speakers. However, to function, all of these elaborate indexing strategies require a pre-recorded audio stream for analysis. To evaluate contextual pauses, or changes in speaker, an element of look ahead along the temporal cascade is necessary. Thus it is unlikely that such methods could be employed to skim a document using

dynamically generated synthesised speech.

Many issues faced by screen reader users are similar in origin to those of pre-recorded speech. That is, how can the content be best indexed to combat the serial nature of the medium. Due to the transience of speech output, only a small fragment of content is in focus at any given time. Therefore, much work is needed to derive indexing strategies to allow for simple navigation to relevant portions of the content. In Chapter 3 a description of the current literature discussing methods attempting to alleviate this problem is presented. In Chapter 5 the methods in which WebTree tries to address these issues are described.

In many cases, passive listening to the content is sufficient to assimilate the information. For example, many novels and other fictional material can be easily absorbed through sequentially reading large blocks of text. However, a number of situations exist where a greater degree of interaction with the subject matter is necessary to aid comprehension, e.g., the reading of factual material, such as textbooks or technical specifications. To minimise the effects of the transient nature of speech on memory capacity, direct access to a synchronised textual transcription of the spoken output is necessary. The digital talking book specification produced by the “Digital Accessible Information SYstem consortium” (DAISY consortium²) recommend that this facility be introduced. This specification attempts to merge a number of technologies, i.e., naturally recorded speech, synthetic speech and a textual transcript, to improve the reading experience of the user. Thus, if the reader is having difficulty understanding elements of the content, the facility to directly examine the text transcript to verify the intended meaning is available.

The inclusion of text to speech technology in conjunction with a natural speech rendition of the content may seem a little excessive. However, synthetic speech may provide greater control on how the content is presented. For example, it might be necessary to allow the examination of the content

²<http://www.daisy.org/>

on a character by character basis to assist in assimilating complex information. Also, the textual transcript would allow for Braille access, in addition to cut and paste facilities.

2.1.1 Synthetic Speech

For centuries, scientists have aspired to artificially synthesise something approximating human speech. In fact, according to Gold and Morgan (2000), attempts at generating synthetic speech date as far back as the 18th century. Sometime, around the 1780s, Von Kempelen built a mechanical talking machine, which demonstrated that the human speech production system could be modelled. This device was not a direct model of the workings of the human vocal tract. However, von Kempelen did succeed in demonstrating that a mechanical device mimicking human speech production was possible. A bellows was used to provide the air stream; a vibrating reed produced the periodic pressure wave; and a number of various small whistles and levers controlled the articulation of most consonants. The resonator of leather was manipulated by the operator in an attempt to copy the acoustic configuration of the vocal tract during the sonorant sounds (vowels, semivowels, glides, and nasals) (Gold and Morgan, 2000). For additional information on von Kempelen's speaking machine, see the paper by Dudley and Tarnoczy (1950).

Since von Kempelen's time, many attempts at developing an artificial speech production system have taken place. According to Gold and Morgan (2000); Klatt (1987), modern methods of speech processing really began in the U.S. with the development of two different types of apparatus. The "channel Vocoder" (voice coder) and the "Voder" (voice-operated demonstrator) were both pioneered by Homer Dudley in the 1930s. The channel Vocoder analyzed speech into slowly varying acoustic parameters that could then drive a synthesiser to reconstruct an approximation to the original waveform. This idea led to the development of the Voder device, which was an experimental speech synthesiser whose output was controlled by in-

put from a human operator.

The Voder apparatus was composed of a control console in conjunction with an electronic instrument to produce spoken output. The control console consisted of keys for selecting a voicing source or noise source, with a foot pedal to control the fundamental frequency of voicing vibrations. The source signal was routed through ten bandpass electronic filters whose output levels were controlled by an operator's fingers (Klatt, 1987).

It is important to note that the Voder did not speak without a great deal of assistance from the human handler. The interface to the Voder was rather cumbersome and extremely difficult to control. Many prospective candidates for the position of operator were unable to learn the system. For those few who did eventually master the console interface, six to twelve months training was often required (Gold and Morgan, 2000). For a more in-depth discussion of the workings of the Voder device, see the paper by Dudley et al. (1939).

Since the Voder, there has been little or no research to determine whether a real time speech synthesiser can be accurately controlled by a human operator. Modern day speech synthesis is usually performed by digital computer programs or specialised electronic hardware devices. Concentration has focused on devices that derive their information from a stored vocabulary, and those which convert typed or electronically scanned text into spoken output. The latter form of synthesiser is often referred to as a text to speech (TTS) engine. However, many of the underlying principles have remained quite fixed since the time of the Voder. That is, there is often a separation of source and filter followed by the parameterisation of each (Gold and Morgan, 2000).

In his discussion of the source-filter theory of speech generation, Klatt (1987) tells us that in addition to the methods for copying the time-varying spectral patterns of speech, found in the Voder, a critical next step in the history of speech synthesis was the development of an acoustic theory on how speech is produced. Also, the design of formant and articulatory synthesisers

based on this theory were contributing factors. He cites (Fant, 1960) as providing a summary of this theory.

In its simplest form, the acoustic theory of speech production states that it is possible to view speech as the outcome of the excitation of a linear filter by one or more sound sources. The primary sound sources are voice caused by vibration of the vocal folds, and turbulence noise caused by pressure differences across a constriction. The resonance effects of the acoustic tube formed by the pharynx, oral cavity, and lips are simulated by the linear filter. This vocal tract transfer function can be modelled by a set of poles – each complex conjugate pair of poles producing a local peak in the spectrum, known as a formant (Klatt, 1987).

It is generally accepted that there are three distinct classifications of speech synthesiser:

Articulation-based synthesis : This is a synthesis method mostly of academic interest at the moment. It attempts to computationally model the human vocal tract and the articulation processes which occur. These models are currently not sufficiently advanced or computationally efficient to be used in commercial speech synthesis systems.

Concatenative based : Produced through the concatenation of segments of pre-recorded human speech. It is generally recognised that concatenative synthesis delivers the most natural sounding synthetic speech. However, audible glitches in the output, due to natural variations in speech and automated methods used to segment the waveform, are known to sometimes detract from the speech quality.

Formant-based (sometimes referred to as rule based) : No human speech samples are used in this process. Vocal output is generated using an acoustic model. Parameters such as fundamental frequency, voicing, and noise levels are varied over time to create a waveform of artificial speech. Speech synthesised through this method sounds rather artificial and robotic in nature, and would never be mistaken for

a human voice. However, maximising the naturalness of the output is not always the ultimate goal of a synthesiser. Formant synthesis has a number of major advantages over concatenative systems. They are often used by screen reading software for the blind, for speech can be very reliably intelligible, even at high speed. Thus, avoiding the acoustic glitches that can often plague concatenative systems. Formant-based systems have total control over all aspects of the output speech, thus, a wide variety of prosodic or intonation rules can be applied. Conveying not just questions and statements, but a variety of emotions and tones of voice.

Early attempts at producing synthetic speech were achieved using dedicated electronic circuitry. However, modern synthesisers have abandoned these methods in favour of direct computation on a general-purpose digital computer or specially constructed hardware devices (Klatt, 1987). As the personal computer (PC) has become much more powerful and robust, the cost of additional transient memory and permanent storage space has been greatly reduced both in terms of computing resources and monetary value. The distribution of these resources as standard by the PC manufacturers, plus the expense of dedicated hardware solutions, has resulted in the tendency in the last decade to move towards the use of software synthesisers. Once developed, the software synthesiser can be easily and inexpensively transferred from computer to computer. Thus, an unlimited number of copies can be issued without much additional expense to the developer.

Regardless of the type of speech synthesiser selected, a dynamic text to speech engine requires an additional layer of functionality to perform text symbol to phoneme transcription. A phoneme is the smallest phonetic unit in a language that is capable of conveying a distinction in meaning, as the m of mat and the b of bat in English (American Heritage Dictionary, 2000). Writing a list of phonemes for a spoken block of text is generally not a difficult task for a trained linguist. However, it required much research before a text to speech engine could perform this task automatically (Klatt, 1987;

Gold and Morgan, 2000). According to Gold and Morgan, the first fully operational text to speech system was demonstrated by Umeda et al. (1968). This research has been continued by numerous individuals and companies throughout the following decades culminating in the abundance of text to speech engines present today.

For a more in-depth examination of the history of how speech synthesis evolved, and a description of many of the methods experimented with to produce spoken output, see the paper “Review of text-to- speech conversion for English” by Klatt (1987), and the book “Speech And Audio Signal Processing: Processing and Perception of Speech and Music” by Gold and Morgan (2000).

2.1.2 Comprehension of synthetic speech

Research in the area of speech comprehension indicates a definite reduction in the recall of synthetic speech in comparison to that of natural speech (Waterworth and Thomas, 1985; Luce et al., 1983; Lai et al., 2000). Luce et al. concluded that the reason for this is that synthetic speech increases the effort involved in encoding and/or the rehearsal of presented information. They suggest that more processing capacity is required to maintain this information in short-term-memory and then transfer the material into long-term storage. The overheads for processing synthetic speech are analogous to those of noise-degraded speech, due to the additional processing power required. Luce et al. also cited the role of issues relating to prosodic inadequacies, and the often-inadequate specification of acoustic cues to phonetic segments, as major contributors to the problem of synthetic speech intelligibility.

Rabbitt (1966) found that when subjects used degraded speech as the primary mode of interaction, there was a noticeable degradation in efficiency in performing simultaneous secondary tasks. He concluded that this demonstrated greater channel capacity demands on short-term memory, at either the encoding or rehearsal stage, when processing noise-degraded speech. He

also found that if presented with two different sequences of numbers, the first list was recalled much better if the second sequence was presented in clear noiseless speech (Rabbitt, 1968). Thus, it seems that the processing of noise degraded speech and hence synthetic speech takes channel capacity that could otherwise be used for encoding/rehearsal or for performing secondary tasks (Waterworth and Thomas, 1985). Waterworth and Thomas suggest that although more effort is required to encode the synthetic information, it is stored by the listener just as efficiently as natural speech.

Pisoni et al. (1985) report five major factors that influence listener's performance in laboratory situations:

1. Quality of speech signal
2. Size and complexity of message set
3. Short-term memory capacity of listener
4. Complexity of the listening task or other concurrent tasks
5. The listener's previous experience with the system

Many of the previous investigations into the intelligibility of synthetic speech have been carried out using lists of single words, separated by pauses. For example, (Waterworth and Thomas, 1985). It was demonstrated that when the length of the pause was reduced, the retention was degraded far below that of natural speech. Waterworth and Thomas conjectures that the reason for this is that listeners are exhibiting a recency or primacy effect.

Lai et al. (2000) examined the effects of listening to lengthier passages of content using both natural and synthetic speech. Five alternative commercial text to speech engines were used in this study. No definite degradation between the different speech engines was observed; however, levels of comprehension for natural speech were noticeably higher than for synthetic speech. The experiments performed by (Lai et al., 2000, 2001) suggest that although advances in the quality of synthetic speech have occurred, it is still less comprehensible than natural speech.

According to Rosson (1985); Lai et al. (2000), there is evidence of a training effect through exposure to synthetic speech over a period of time.

Lai et al. states:

An important factor with TTS is the rate at which humans adapt their ear to the sound of synthetic speech (or human speech when spoken by a foreigner) and the effect that this adaptation has on levels of comprehension. The amount of training necessary to see a change in the perception of synthetic speech ranges in various studies from a few minutes of exposure (van Bezooijen and van Heuven, 1998) to four hours of training (Greenspan et al., 1988). Also when subjects improve performance during the course of a study, it is hard to differentiate what portion is due to a familiarisation with the process, and what is due to training.

One major factor hindering the comprehension of synthetic speech is the fatigue effect. This is primarily brought about by its often-monotonous quality (Fitzpatrick, 1999). The comprehension of the material can decrease as the listener becomes either tired or bored with the vocal presentation. It often occurs in the course of reading lengthier, more syntactically complex passages of content, or with usage over long periods of time. However, there is conflicting evidence as to how much the comprehension levels of synthetic speech degrade due to the length of the passage being read. Evidence produced by Lai et al. (2000) indicates that comprehension levels degrade linearly with the size of the passage. However, experiments performed by Lai et al. (2001) suggest that the complexity of the subject matter being spoken has a large bearing on the ease of content recognition.

2.1.3 Voice Intonation and Prosodic Cues

Although the intelligibility of speech output is extremely important for content comprehension, a second substantial component influencing this process

is the pleasantness or naturalness of the speaking voice. Voice prosody is one of the major areas under research in the hope of achieving a more natural sounding speech interface. The prosodic component of speech is that set of features which lasts longer than a single speech sound. In linguistics, prosody refers to the intonation, rhythm, and vocal stress in speech. The prosodic features of a unit of speech, whether a syllable, word, phrase, or clause, are called *suprasegmental* features because they affect all the segments of the unit. These suprasegmental features are manifested, among other things, as syllable length, tone (linguistics), and lexical stress—see the Prosody/ (linguistics) page from Wikipedia, the free encyclopaedia³.

According to Klatt (1987), a pure tone can be characterised in physical terms by its intensity, duration, and fundamental frequency. These induce the sensations of loudness, length, and pitch, respectively. In speech, it is the change over time in these prosodic parameters of intensity, duration, and fundamental frequency of vocal cord vibrations that carry linguistically significant prosodic information.

The intensity pattern of the vocal output tends to set off individual syllables, for vowels are usually more intense than consonants. The amplitude of a stressed syllable is generally at a higher decibel than an unstressed phonetic unit. Klatt (1987) tells us that intensity per se is not a very effective perceptual cue to stress, due in part to the confounding variations in syllable intensity associated with vowel height, changes in pitch governed by changes in the fundamental frequency of vocal cord vibration, laryngeal state, and other factors. The duration of the spoken syllable's pronunciation also can have a major influence on the positioning of vocal stress.

The timing of the individual syllables, words and phrases contained in the output is essential in producing natural sounding speech. Both the duration of each syllable, coupled with the accurate insertion of pauses between words or phrases contributes in assigning the desired context to individual speech fragments. For example, consider the word *minute*. When discussing the

³<http://en.wikipedia.org/wiki/Prosody-%28linguistics%29>

time segment, the u is pronounced as a short vowel. However, when used in reference to a small object, the u requires a much longer sound articulation. The amount of stress placed on the syllables in a given word also contribute to its contextual comprehension.

An important component required for the creation of natural sounding speech is the use of voice intonation to provide contextual information. Voice intonation is the variation of tone employed whilst speaking. Intonation and vocal stress are two main elements of (linguistic) prosody. Many languages use pitch syntactically, for instance to convey surprise and/or irony. Or, to change a statement to a question. Such languages are called intonation languages, of which English is a well-known example. In addition, some other languages rely on intonation to convey meaning. Those in which the syllables are contrasted by pitch are known as tonal languages, of which Thai is a prime example. The situation is further complicated due to the existence of an intermediate position occupied by languages with tonal word accent, for instance the Norwegian language, see Wikipedia's web page on Intonation⁴. However, for the purposes of this discussion, only the intonational attributes found in the English language will be considered.

Rising intonation increases the harmonics of the articulation; falling intonation denotes a decrease in voice modulation. The classic example of intonation in an *intonation* language is the question/statement distinction. For example, north-eastern American English language has a rising intonation for echo or declarative questions (He found it on the street?), and a falling intonation for *wh*- questions and statements. Yes/no questions often have a rising end, but not always. Again see the Intonation page from Wikipedia for more details⁵.

However, the meaning attributed to changes in voice intonation can be rather subjective. Much may depend on the location and background of both the listener and speaker. For example, dialects of British and Irish English vary substantially with rises on many statements in urban Belfast,

⁴http://en.wikipedia.org/wiki/Intonation_%28linguistics%29

⁵http://en.wikipedia.org/wiki/Intonation_%28linguistics%29

and falls on most questions in urban Leeds (Grabe, 2004). This subjectivity renders the accurate prediction of intonation cues for the generation of synthetic speech an arduous task. However, without this feature, the synthetic speech would sound extremely monotonous and incomprehensible especially for lengthy passages of content.

To mimic the intonational patterns of natural speech, or to determine the stress applicable to individual speech fragments, an in-depth semantic analysis of the texts' construction must be performed. According to Sproat (1996), the problem of converting text into speech for a given language can be divided into two sub-problems.

The first sub-problem involves the conversion of linguistic parameter specifications (for example, phoneme sequences) into parameters (e.g., formant parameters, concatenative unit indices, pitch pairs) that can drive the actual synthesiser. However, this is considered to be out of scope for this discussion. The second sub-problem involves computing the linguistic parameters from input text. In any language, orthography is an imperfect representation of the underlying linguistic form (Sproat, 1996).

One of the first things an English TTS system needs to do is tokenise the input into words. For the English language this is not generally difficult although for some other languages it is more complicated. A pronunciation then needs to be computed for each word. In English, given the irregularity of the orthography, this process involves a fair amount of lexical lookup though other processes are involved too. Also, some of the words in the sentence should be assigned accents, before breaking the input into prosodic phrases. According to Wikipedia's pages on Intonation⁶, English punctuation only partially correlates with speech prosody. Thus, although various kinds of linguistic information need to be extracted from the text, only in the case of word boundaries can this linguistic information be said to be represented directly in the orthography (Sproat, 1996).

When reading aloud a lengthy sentence, it is quite common for the

⁶http://en.wikipedia.org/wiki/Intonation_%28linguistics%29

speaker to segment the content into smaller phrases to aid comprehension. In situations where punctuation is used liberally through out the text, it is reasonable to expect that prosodic phrase segregation be applied to coincide with these natural boundaries. However, segmenting the content in this manner is not always appropriate. A problem arises when lengthy segments of unpunctuated text are to be transcoded into speech. In such cases, some heuristic algorithms to determine phrase boundaries are introduced. See (Sproat, 1996; Klatt, 1987) for some examples. The major problem with machine generated prosodic segmentation is related to its inability to understand what is being spoken. Thus, accurately concluding the correct prosodic or intonational phrasing for an unknown piece of text is rather difficult (Sproat, 1996).

Once the semantic evaluation of the material has been completed, a set of contextual speech rules are applied to the content to determine any additional stresses and to allocate intonational values to the individual text fragments. According to d'Alessandro and Liénard (1996), the prosodic parameters in natural speech interact in a way that is still unknown, in order to supply the listener with prosodic information while keeping the feeling of fluentness. Understanding the interplay of these parameters is today a very active topic for research on speech synthesis. For prosodic generation, a move from rule-based modelling to statistical modelling is noticeable, as in many areas of speech and language technology. As far as speech naturalness is concerned the problem is still almost untouched. Nobody knows what speech naturalness is or more generally what is expected from a synthesis system once its intelligibility is rated sufficiently highly. In order to explore this domain it will be mandatory to cooperate with psychologists and human factors specialists (d'Alessandro and Liénard, 1996).

2.2 Tactons, Earcons, Auditory Icons and General Non-Speech Sound Cues

In addition to the use of visual, Braille or speech interfaces as principal human computer interaction (HCI) modalities, a number of alternative methods to complement these user interface (UI) paradigms have been proposed. These alternative technologies include, Tactons (Brewster and Brown, 2004), Earcons (Blattner et al., 1989), auditory icons (Gaver, 1986), and generic non-speech sound cues (Buxton et al., 1994). These technologies are not expected to operate as the primary user interaction modality. However, it is suggested that their inclusion as a subordinate UI device can aid in the assimilation of contextual information for non-visual computer usage. Some examples of their usage include: (Raman, 1996*a*; Mynatt and Edwards, 1992; James, 1998; Petrucci et al., 2000).

2.2.1 Tactons

Tactons, or tactile icons, are structured, abstract haptic messages that can be used to communicate information non-visually. A range of different parameters can be used for Tacton construction including: frequency, amplitude and duration of a tactile pulse, plus other parameters such as rhythm and location (Brewster and Brown, 2004). According to Brewster and Brown, tactile/haptic devices have been available for quite a while, however, only recently did they receive much attention from the HCI community. Such devices were often just prototypes existing purely in engineering labs. Now that the technology has developed enough to be reliable and inexpensive, members of the HCI community have started researching areas in which these devices might add to the users overall computer interface experience. It is hoped that such technologies would be beneficial in attracting attention to a change in user interface state where visual or sound cues are not appropriate. This is especially the case where the visual display is quite small, or in situations where it is inconvenient to signal changes through sound. An

example of such a device is a mobile phone. A tactile message in addition to an auditory signal is often produced in order to alert the user to an incoming call or SMS text message.

According to Brewster and Brown (2004), the human sense of touch can be roughly split into two parts: kinaesthetic and cutaneous. "Kinaesthetic" is often used as a catchall term to describe the information arising from forces and positions sensed by the muscles and joints. Force-feedback haptic devices are often used to present information to the kinaesthetic sense. Cutaneous perception refers to the mechanoreceptors contained within the skin, and includes the sensations of vibration, temperature, pain and indentation. Tactile devices are used to present feedback to the cutaneous sense.

Much of the previous experimentation with tactile interaction has focused on stimulating the touch receptacles in the fingers. However, according to Brewster and Brown (2004), some research has been carried out regarding the use of other parts of the body such as the back to receive tactile information. Due to the area of research still being in its infancy, much work is still required to establish the best mechanisms in which this technology might be employed, and the circumstances in which its usage might contribute greatest benefit to the user.

2.2.2 Non-speech audio cues

As stated previously, the temporal nature of speech ensures that the delivery of material through spoken output alone can be sometimes slow and cumbersome. Including additional speech fragments to convey contextual information as a direct component of the speech stream may drastically increase the verbosity of the presentation. This can result in a degradation in the speed of content assimilation. For example, announcing task completion information for a secondary application running in the background may interfere with the cognitive processes for the current task. Thus, alternative mechanisms to alert the user to changes in context were investigated.

One solution to this problem is to use alternative reading voices, or manipulate certain characteristics of the voice to denote such changes. For more information see section 3.2. Another of these solutions suggested using non-speech audio cues, which do not interfere with the current operation to complement the speech output.

According to Buxton et al. (1994, Chapter 1), it isn't just non-visual users who could benefit from using non-speech sounds to communicate contextual information:

As our displays become more visually intensive, the visual channel becomes increasingly overloaded, and we are impaired in our ability to assimilate information through the eyes. In such instances, the same audio cues that help the totally blind user can be applied to help the normally sighted.

Evidence produced by Brown et al. (1989), appears to substantiate this hypothesis.

In general, the type of audio cues used in modern day computing can be classified into two broad ranging categories. The first category includes the use of abstract musical sounds to depict contextual information. These sounds are created by directly manipulating pitch, loudness, duration, and wave shape. Abstract auditory Earcons are a prime example (Blattner et al., 1989). The implied meaning of such cues is not automatically apparent on initial hearing; associations with changes in the user interface must be learned before any advantages of their use can be illustrated. The second classification of audio cue relies on real life associations between familiar every day environmental sounds and the tasks they represent to provide contextual information. For example, specifying cues in terms of events in the natural environment such as a door slamming or people applauding. Thus, the musical sounds focus on the properties of the sound itself, while the everyday sounds focus on the source of the sound (Buxton et al., 1994).

Non-speech audio messages can be thought of as providing one of three general types of information: alarms and warnings, status and monitoring

indicators, and encoded messages. Typically, different types of audio cue are used for each. Humans are capable of monitoring more than one such signal in the background, providing that the sounds are appropriately differentiated. However, it is important to remember that although we can recognise and simultaneously monitor a number of different concurrent audio cues, we can normally only respond to one or two at a time (Buxton et al., 1994).

Auditory Icons

Auditory Icons, proposed by Gaver (1986), are audio cues based on the axiom that natural environmental sounds be employed to provide conceptual computer information. That is, using sound in a manner analogous to the use of visual icons to provide data. They function on the principle that instead of using dimensions of sound to symbolise dimensions of the data; dimensions of the reverberation's source are relied upon to indicate meaning. Gaver (1986) maintained that employing real life natural world sounds for which the user has symbolic associations already mapped ought to enable the listener to quickly establish and memorise their significance when found in a computerised setting. However, he observed that it was not required that auditory icons be realistic representations of the objects they portray but should capture their essential features.

Gaver uses the example of a mailbox to illustrate this point. As an item is deposited, the size of the item and its type can be often ascertained from the sound generated. In computer terms for example, the sound of paper hitting a metal mailbox might denote that the mail received is a text file, whereas the arrival of an executable file might be denoted by a metal on metal reverberation. The weight of the element expressed through the sound of its arrival in the mailbox may indicate the size of the specified item. Naturally an exact mapping of the size through sound alone is rather difficult, however, a reasonable estimate ought to be obtainable through such means. In addition, the positioning of the sound in the audio space might also provide location information for the window in which the interaction

has taken place.

A direct one to one mapping between a sound source and a specific UI concept is not always possible or in fact desirable. The types of sound source can vary drastically across a wide spectrum, whilst still providing similar information. In addition, a single source can often produce many alternate sounds. Accordingly, there exists many varieties of meaning that can be gleaned from a sound source depending on the context in which it appears. Thus, much thought must be invested when selecting the correct sound source to signify the intended meaning.

Gaver (1986) indicates that there exist three classifications of mapping, namely, *symbolic*, *nomic* and *metaphorical*. Symbolic mappings are often allocated on an arbitrary basis, relying on social convention to ensure comprehension. Telephone bells, sirens, and stop signs are examples of symbols. At the other extreme are representations, which have a nomic relation to the information they convey. Their meaning depends on the physics of the situation. The relation between a sound and its source or a photograph and the scene it depicts are examples of nomic mappings: The representations are images of the information. Finally, metaphorical mappings make use of similarities between the item to be represented and the representing system: They are not wholly arbitrary, yet they do not depend on physical causation. Metaphorical mappings include structure mapping, in which similarities between the structures of two things are exploited, e.g., mappings between genealogy and a tree, and metonymic mappings, in which a feature is used to indicate an entire object/event. For example, the use of a hiss to stand for a snake is a metonymic mapping. It was expected that the more a representation's form depends on its meaning, the easier it should be to learn. Thus nomic mappings should be relatively simple to learn, metaphorical mappings somewhat harder, and symbolic mappings the most difficult (Gaver, 1986).

Earcons

Earcons, Proposed by Blattner et al. (1989), are defined as structured non-verbal audio messages used in the user-computer interface to provide information about some computer object, operation, or interaction. Earcons are the aural counterparts of graphical icons. However, there exists a number of major differences between earcons and their visual counterparts. For example, earcons are transient thus require immediate attention whilst being played, whereas icons are often presented simultaneously. The simultaneous presentation of earcons is a complex issue. Blattner et al. discovered a number of issues with earcon recognition when multiple earcons were played concurrently. Research by Brewster et al. (1995*b*) appears to dispel the belief that playing earcons in parallel is an unfeasible solution, once large differences between the earcon components are used. However, McGookin and Brewster (2004) noted a degradation in comprehensibility as additional concurrent earcons are added to the stream. They suggest a number of guidelines to aid the assimilation of data through this mechanism. They recognise that much further work is necessary before presentation through a number of concurrent earcons is practical.

Graphical icons are frequently both selectable and informational, whereas earcons are predominantly just informational (Blattner et al., 1989).

Blattner et al. divide earcons into three classifications: representational, abstract, and semi-abstract. Representational earcons roughly equate to Gaver's class of auditory icons, see section 2.2.2. Abstract Earcons use single pitches or groups of pitches (motives) as the elements or building blocks of earcons. Motives are sequences of pitches that create a short, distinctive audio pattern often characterised by the simplicity of its rhythm and pitch design. Their very brevity and distinctive manner make motives very powerful tools for composing earcons. Compound earcons composed of motives or single pitches can be used to express complex objects. The final category is a combination of the previous two classifications of earcon.

According to Blattner et al. (1989), Bernstein and Picker (1966) tells us

that a motive is a brief succession of pitches arranged to produce a rhythmic and tonal pattern sufficiently distinct to allow it to function as an individual, recognisable entity. These larger structures are used in the generation of abstract earcons. Rhythm and pitch are the fixed (primary) parameters of motives, whilst timbre, register and dynamics are known as the Variable (secondary) characteristics. However, Brewster et al. (1995*b*) appears to contradict this conjecture by implying that timbre and register of the auditory cue play a more important role in earcon recognition than suggested by Blattner et al.. Motives can be combined to create larger more complex earcons. By varying the different components of a motive, a distinct sound can be created.⁷

The three construction principles for compound earcons are combining, inheriting, and transforming. These three methods are used with both representational and abstract elements. Combined earcons are formed by placing two or more audio elements in succession. They can represent computer entities sharing common features and are constructed by using similar audio elements to represent similar classes of information. Inherited Earcons are hierarchical structures, using audio parameters allocated to common multi application features. The more musical features e.g., timbre / pitch/rhythm are used in earcon construction, the more precise the information to be obtained. The timbre component may denote the family/application to which the earcon has been allocated. The second parameter, e.g., rhythm, might signify the type of message being portrayed, such as a system error. A third component such as pitch could provide a precise error message designation. The final type of earcon described by Blattner et al. (1989), is a transformed earcon, where parameters of the audio cue are modified to denote meaning. They suggest that transformed earcons should only be modified in simple ways that clearly retain perceptual equivalences. It was thought that simple changes in timbre, dynamics, and register would pose no perceptual difficulties. However, pitch changes change the contour of the earcon and should

⁷An earcon can be any sound, ranging from a single note, to a motive, to a complex collection of motives.

be administered with care. For detailed guidelines on how earcons ought to be constructed, see (Blattner et al., 1989; Brewster et al., 1995*a*).

The time necessary to learn an earcon based user interface depends on a number of factors. These include the size of the system, the complexity of the audio cues employed, and the amount of shared features across multiple applications. The types of sounds used are also contributing factors. Blattner et al. (1989) suggest exploiting the western tonal scale to produce musical sounds for western listeners. It is thought that a lifetime of listening to western tonal music should aid in abstract audio cue recognition. There is much material available on the recognition of music from work done in the psychology of music and pure sound, however an in-depth discussion of such material is out of scope for this thesis.

It is believed that initially, representative earcons may be easier to learn, due to their relevant mapping to real world situations. However, a large system may require a sizeable number of sounds to represent each feature. In addition, some features may not possess an immediate equivalent sound in the natural environment therefore learning to associate a sound with a unique user interface feature can place a huge additional workload on the user's memory capacity. Thus, although much additional effort is required to map structured musical sounds to user interface states, the reduction in the number of distinct sounds required should aid the user in memorising these associations—that is, once the methodology used to apply structure to the audio is apparent. Experiments by Brewster et al. (1993) demonstrate that structured musical cues are more useful in representing information about the user interface than just using arbitrary unstructured bursts of sound to achieve the same goal. Also, their evaluations as to whether there were significant differences in the recognition abilities of musicians and non-musicians revealed no significant differences in relation to such cues. In addition, they illustrate that the subtle transformations recommended by Blattner et al. (1989) were in many cases too small; thus, large significant changes between components of the earcons were recommended. Also this

work implies that the use of timbre is a powerful tool in recognising the earcon's family origins.

Investigations have demonstrated that the use of earcons to portray hierarchical information is one viable mechanism for imparting such data (Brewster, 1998). Brewster illustrated that four levels of hierarchical information could be presented using inherited earcons. However, combination earcons could potentially be employed to present many more levels of hierarchical abstraction. The main advantage of using this form of compound earcon to portray such data is that the listener must only learn a small set of rules before comprehension of the intended meaning is possible. Take for example a book including chapters, sections and subsections. If the numbers one to nine are allocated alternate sounds, plus an additional sound to separate each number, then many levels of tree abstraction can be portrayed by presenting sounds in sequence. However, the major disadvantage of this approach is that it can lead to rather lengthy sound cues, which may have an effect on the memory capabilities of the listener. It may be difficult to recall the starting motives in a lengthy audio cue.

As part of the same study, Brewster observed a number of different training strategies for learning to operate an earcon enabled user interface. Although describing the rules and displaying the system have some value to the listener, much value can be gleaned by the user actually experimenting with the system. It was also shown that users could recall earcon mappings over a week after previous usage. Brewster conjectured that the reason for this was the simple rules employed for earcon construction were easy to remember and apply.

2.3 Summary

This chapter focuses on the mechanisms used to provide access to electronic information through a purely auditory modality. The next chapter will concentrate on how these technologies have so far been employed to produce non-visual computer interfaces for the blind.

Chapter 3

Literature Review

The previous chapter presented a number of different modalities available for non-visual computer access. A discussion of the literature describing both the advantages and disadvantages of the different technology was presented. The methods in which these technologies can be harnessed by a computerised system to convey information to blind people were also demonstrated. This chapter outlines a number of different approaches in which this technology has been used to interface with computer based information non-visually. These include methods to convey contextual information and navigation of large complex data constructs. The first area under focus is GUI (Graphical User Interface) based systems. An outline of the problems with these systems and a number of attempts to ensure accessibility are presented. The second section is dedicated to the different methods employed to alleviate the problems imposed by the serial nature of speech output. This can be achieved by manipulating characteristics of the speech stream to increase the pace of content absorption. These include using non-speech sounds to provide contextual information and changes in voice, or in voice parameters such as pitch and stress to provide similar effects. In the next section, different methods in which the document can be internally modelled are presented. These range from an **attributed tree structure** to **conceptual graph** representations. The DOM (document object model) used by WebTree

is also discussed. Following this, a brief description of a number of audio web browsing solutions available today is introduced. Finally, a discussion of the different methods employed to both navigate and obtain information about complex constructs is presented. These constructs include tabular data, XHTML forms and tree based structures.

3.1 The Graphical User Interface (GUI)

The adoption of the graphical user interface (GUI) as the primary method for human computer interaction could be perceived as being a major reason for the growth in personal computer ownership in the last two decades. According to Edwards et al. (1994), graphical user interfaces were quickly adopted by the sighted community as a more intuitive interface. Ironically, these interfaces were deemed more accessible by the sighted population because they seemed approachable for novice computer users. Previous to its endorsement screen reading applications were able to provide reasonably good access to computer information. This was due to the use of ASCII text as the main form of output to the screen. With the advance in GUIs, applications rarely provided content in this manner, instead favouring the drawing of content on a pixel by pixel basis. This made interaction for screen reading software difficult, for they now had to develop methods for recognising the characters and deriving associations between elements. To achieve this, many screen reading applications monitor screen drawing requests to glean the information. However, the data provided is often at a very low level, making it difficult to establish how elements relate to one another. Also, the organisation of elements on the screen became important for easy interaction. Related items e.g., controls for a single application, were often segregated to a certain area of the screen. It was now common to have multiple applications open and in view in separate regions of the display (windows) at the same time. In addition, the mode of input changed from predominantly keyboard access to a reliance on a point and click interface facilitated through the use of a pointing device such as a mouse.

The serial nature of speech output insures that it is difficult to obtain an overall picture of the current interface without examining the entire content. Whereas for a sighted user, such impressions can be often gained with a single glance. For this reason, the blind user is dependent on the screen reading application deriving relationships between elements in order to produce a usable system. To allow the application gain a better idea of the intended meaning attributed to each interface element, it makes sense for screen reader applications to intercept content before it is presented visually (Mynatt and Edwards, 1992). Edwards et al. (1994) tells us that this can be problematic if elements are only described at an extremely low level, without a description of the context in which they appear. In this case it can be difficult to establish how the elements relate to one another. Edwards et al. discovered that rather than examining low level elements such as buttons, and their surroundings to work out their implied meaning, a more practical solution would obtain this information from the applications themselves. However, it would not make sense to do this on an application by application basis. Instead they recommended changes to the shared libraries responsible for screen interaction. In this manner, it would be possible for screen reading applications to obtain high level knowledge of the application interface. Nowadays under Microsoft windows¹, the “Microsoft Active Accessibility” (MSAA) architecture provides a mechanism for sharing information between applications (Sinclair, 2000). Many screen reader applications such as JAWS for Windows² from Freedom Scientific³ and GW Micro’s⁴ “window eyes”, and browsing applications such as WebFormator⁵ and Internet Explorer⁶, take advantage of this facility. Similarly, under GNOME: The Free Software Desktop Project⁷, a facility to share data between applications is also included. The “Java Accessibility APT”, from

¹<http://www.microsoft.com>

²http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

³<http://www.freedomscientific.com>

⁴<http://www.gwmicro.com/>

⁵<http://www.webformator.com/englisch/index.php>

⁶<http://www.microsoft.com/windows/ie/ie6/default.msp>

⁷<http://www.gnome.org>

Sun Microsystems⁸, provides similar functionality. Although the provision of this functionality is limited to Java based applications, it is mentioned here for the sake of completeness. Once the application is able to establish information about the interface, the next question is how to best provide access to such information to blind users.

Although the phrase “screen reader” is still in use today, many of these products are much more than just applications that read aloud the on screen content. It is true to say that the original screen reader products from the text-mode era just read the screen. The same is perhaps true for some early products in the GUI era which tried to “read” at the pixel level. However, the products which intercept progressively “higher” levels of interaction between the application and the display are no longer just reading the screen. The development of application level accessibility APIs, such as MSAA, where the application itself may potentially have explicitly encoded accessibility supports, means that these are no longer “screen readers” in a literal sense. But this historical term is still used, even though it’s no longer literally very meaningful.

The primary goal for modern screen reading applications is to allow a blind user to work with an arbitrary graphical application in an efficient and intuitive manner. However, a number of practical constraints must be addressed. First, blind users do not work in isolation, so collaboration with sighted users must be supported. For this reason, their interaction with the computer must closely mimic the experience of sighted users. A second, and sometimes competing, goal is that the blind user’s interaction be intuitive and efficient. There is a careful balance between modelling the visual appearance of the content and providing a representation that is efficient for non-visual access. A prime example of this is in the viewing of tabular data, which is discussed further in section 3.6. If the representation is too visual it might be difficult to comprehend. Whereas if the representation is altered greatly, collaboration with sighted users may become problematic.

⁸<http://www.sun.com>

Edwards et al. (1994) describes the application interface as a collection of objects which are related to each other in different ways, and which allow a variety of operations to be performed by the user. The screen contents are merely a snapshot of the presentation of that interface which has been optimised for a visual, two dimensional display. Providing access to a graphical interface in terms of its screen contents forces the blind user to first understand how the interface has been visually displayed, and then translate that understanding into a mental model of the actual interface. Edwards et al. maintain that the visual display is only one way in which these elements can be displayed. Therefore, they recommended abstracting out the operations of the interface at the semantic level and mapping them to objects in a non-visual medium instead. They maintain that an explicit mapping of the visual interface through non-visual means is inefficient. The blind user doesn't need to know how the on screen elements are structured, once they can gain access to their functionality and establish how these elements are connected. Edwards et al. argue that by constraining the semantic translation so that *similar* objects are produced in the non-visual presentation that the native application produces in its default graphical presentation, they can maintain the user's model of the application interface. Also using the same terminology as used in the graphical interface e.g., (buttons, menus, windows), to describe components should allow collaboration between blind and sighted users. Naming elements in this manner allows blind users to think of the interface in similar terms to their sighted counterparts, therefore, facilitating collaboration.

The model proposed by Mynatt and Edwards (1992); Edwards et al. (1994), presented relationships between elements in a hierarchical form. These include both *parent - child* relationships, e.g., menus and menu buttons, and *cause-effect* relationships as in a push button causes an action. The system is extended to work in a multi-application environment. Essentially the user's desktop is a collection of tree structures. Users can quickly jump between applications while the system stores the focus for each appli-

cation context. Navigation is achieved through the use of the “keypad” or through application specific keystrokes and the use of user defined macros.

Savidis and Stephanidis (1998) presented a system that provided the ability to create *dual user interfaces* for an application. These modes of interaction were to be optimised for both visual and non-visual access. They developed a language called “Homer” in which forms of interaction could be specified. They maintain that whilst there is some cross over in the forms of interaction facilitated by the two modalities, there also exists major differences. By developing interfaces in this manner the modality of interaction should be optimal for the type of usage. This should alleviate problems associated with trying to transform an interface optimised for one modality into another. They demonstrate the effectiveness of this approach with a picture viewing system. The picture is divided into segments navigable by the user. Each segment is provided with a title and description of the portion of the picture. By navigating through the different segments, the user can obtain a better knowledge concerning the layout of the picture. However, this approach has the major draw back of requiring the developer to include large amounts of additional information to ensure that the system is effective.

According to Barnicle (2000) there still exists many obstacles to the efficient access of GUI based interfaces for blind people. These issues include the inefficiency of having to read through large lists of elements, such as menu items to find the correct item. Although sighted people may have to check through the same number of items, they can quickly scan until the required element is located. Whereas the serial nature of speech ensures the user must listen to many if not all of the individual items before making an informed decision. Another major obstacle is the lack of user feedback provided. For example, if a blind user manipulates the size of a font in a word document they are not immediately notified that the change has occurred. Also if non-standard windows controls are used, it can mean that the screen reader has trouble interpreting the intended meaning. For this

reason, screen reader developers spend a good deal of time attempting to customise access to many mainstream applications. Many of the problems cited by Barnicle can also be applied to the web. The large variety of web page structural designs ensure that the user must obtain a general knowledge of how the page is structured before they can become efficient users of any particular website. Barnicle concludes by telling us:

Developers of screen reading technology are continually refining their applications to provide improved access to GUI based software applications. However, mainstream software developers and developers of web-based applications need to gain a better understanding of how users with disabilities are using and attempting to use their products. Including users with disabilities in usability tests is one important step in the process of making products that are usable by individuals with a variety of physical and sensory characteristics.

An alternative approach to traditional screen reading software is the emacspeak⁹ application described in (Raman, 1996*a,b*). According to Raman, a lot of work has been invested into the development of robust off-screen models (internal data structures) of the data to facilitate screen reader interaction. However, audio formatting of the spoken output has not altered a great deal over this time. It is often left up to the listener to establish a mental model of the content by examining different areas of the display. However, much of the *implied* structure appearing in the visual output of the application is lost when provided through a screen reader. Emacspeak uses different voices to convey differences in structural elements. Also, auditory icons are used extensively to enhance the audio experience. Emacspeak integrates speech into the applications themselves in the hope of producing more usable speech output. By using the internal representation of an application's results, a more usable spoken version of the content can be generated, whilst still presenting the content in a satisfactory manner

⁹<http://emacspeak.sourceforge.net/>

for visual users. However, to operate this system efficiently, a customised interaction mode must be generated for each application. Due to the system being open source, individuals are free to write their own custom modes and submit them for inclusion in the main distribution.

GNU Emacs¹⁰ under which emacspeak operates is more than just a text editor. Emacs is, in essence, an applications platform with a primarily text-based user interface. While recent versions of emacs have added some GUI interface mechanisms, these are always secondary, optional, alternatives to primary text based interactions. In this sense, speech enabling emacs applications is somewhat easier than speech enabling applications on a native GUI platform. On the other hand, emacspeak is not *just* a reversion to the earlier text-based screen readers. In using a relatively high level “off screen” model, and integrating tightly with applications, it is actually conceptually very close to modern GUI-based “screen-readers”. However, it does have the serious drawback that it is limited to accessing applications available for this relatively minority platform. Nevertheless, the multitude of applications available that run under this environment ensures a great deal of access through this combination of applications. The available applications range from a shell program providing access to the UNIX shell, the Emacs/W3¹¹ web browser facilitating access to the world wide web, to electronic mail programs such as vm¹²: a lisp-based mail reader for Emacs. The mechanisms used by emacspeak to provide auditory output shall be discussed in section 3.2.

Irrespective of the methods in which the data is obtained from the user interface, the prime component in determining the usability of the system is how the information is presented to the user. For this reason, the remainder of this chapter is dedicated to the mechanisms used to present information in a non-visual modality. Special focus is placed on the portrayal of web page constructs.

¹⁰<http://www.gnu.org/software/emacs/>

¹¹<http://www.cs.indiana.edu/elisp/w3/docs.html>

¹²<http://www.wonderworks.com/vm/>

3.2 Manipulating the Audio Stream to Provide Contextual Information

One major problem with the use of audio output as the primary mode of interaction, is how best to convey contextual information to the listener in an efficient and unobtrusive manner. Augmenting the vocal output with spoken fragments describing the context in which content appears, can significantly increase the verbosity of the output. Although experienced blind users often listen to speech output at very high word per minute rates, presenting information in this manner can impinge on reading efficiency. If the user has to listen to additional information that is superfluous to the main content, they may become distracted with the presentation. This is especially the case if the fragments are in any ways lengthy. Also, the modality is serial in nature, therefore including extra speech fragments will serve to slow the rate of interaction.

In saying this, it sometimes makes sense to have this type of content announced through speech. This can be advantageous in situations where the user is required to directly interact with the document's content—for example, when filling out an online form. Announcing the presence of a form field/control and/or its current value as it is encountered, might be an effective solution. However, speaking the name of each inline element e.g., `` or `` would seriously impinge on the readability of the document. Many of the current web access solutions use this method extensively to convey certain information. For example, when a hyper link is encountered, its presence and status is often announced by JAWS/IE¹³. Other applications actually incorporate an additional text fragment into the content, for example, WebFormator¹⁴. That is, the word “link” is inserted into the content preceding the link text. This ensures that the additional contextual information is navigable by the arrow keys just like any other text. Figure 3.2 shows a simple web page as displayed through WebFor-

¹³http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

¹⁴<http://www.webformator.com/englisch/index.php>

mator. Figure 3.1 shows how the same page might be displayed through Internet Explorer. King et al. (2004a) mainly restricts the use of additional text to convey hyperlinks. Other structural elements such as <h*> elements are signalled with additional line breaks. However, there is no method for determining the type of element currently under point. This method for signalling elements is rather ambiguous and potentially confusing. This is because the user must establish whether the chunk of text is a heading or a short paragraph from the context in which it appears in the text. Also no level information is provided for header elements, for King et al. maintain that the important factor is the meaning implied by marking an element as a heading rather than the level at which it appears. The prime advantage of speaking the contextual cue is that additional methods for gleaning such information do not have to be learned by the user. However, the major disadvantage is the fact that the speech cue is presented serially to the rest of the text, whereas the optimal solution would be to have the context recognised in parallel.

There is an obvious trade off in efficiency when using this method. On the one hand the user knows exactly what the cue means without having to map its occurrence to a learned meaning. That is, once the spoken cue is both concise and explanatory. However, the added verbosity can slow down the user when examining content. Take for example a list of elements containing a number of nested lists. Announcing the beginning and end of each sub list, coupled with information concerning the number of sub elements can be excessive if the user must trawl through this information to get to the required content. This is especially the case where only one or two elements are contained within. Therefore, it is imperative that spoken cues be kept as short as possible. Figure 3.3 shows the default method in which list announcements are made through a combination of JAWS and Internet Explorer.¹⁵

The work by Goose and Möller (1999) suggests that spatial audio (stereo)

¹⁵The announcement of many such elements can be turned off in JAWS if the user prefers.

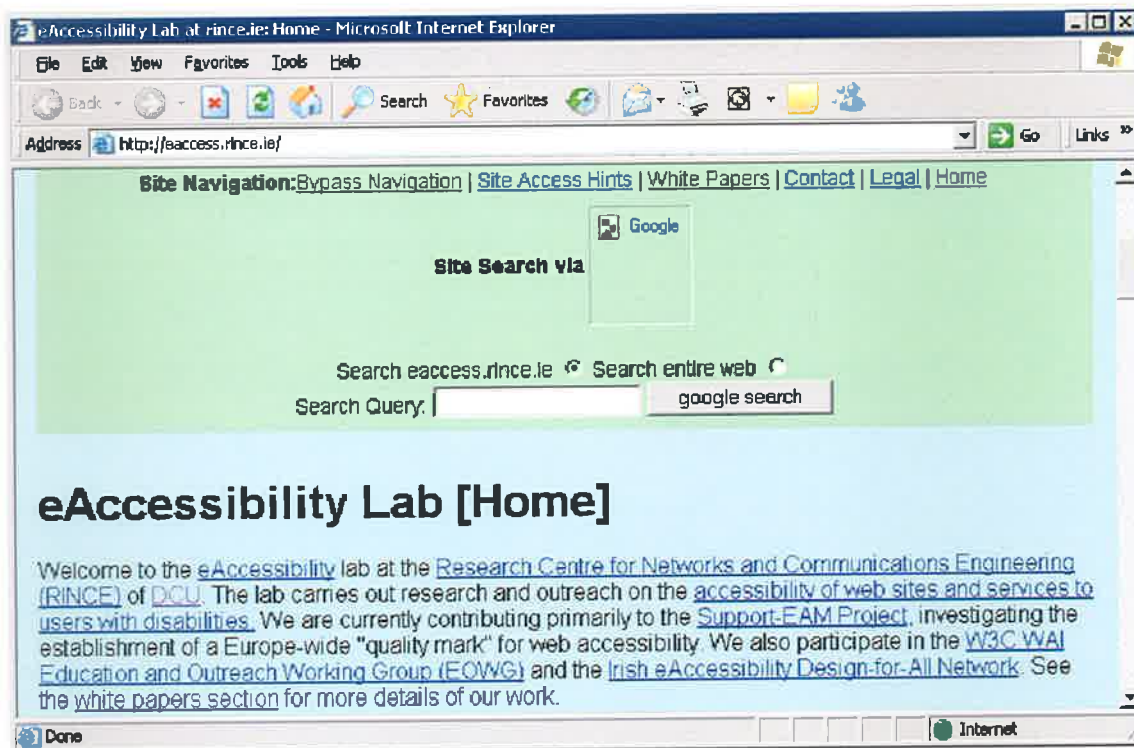
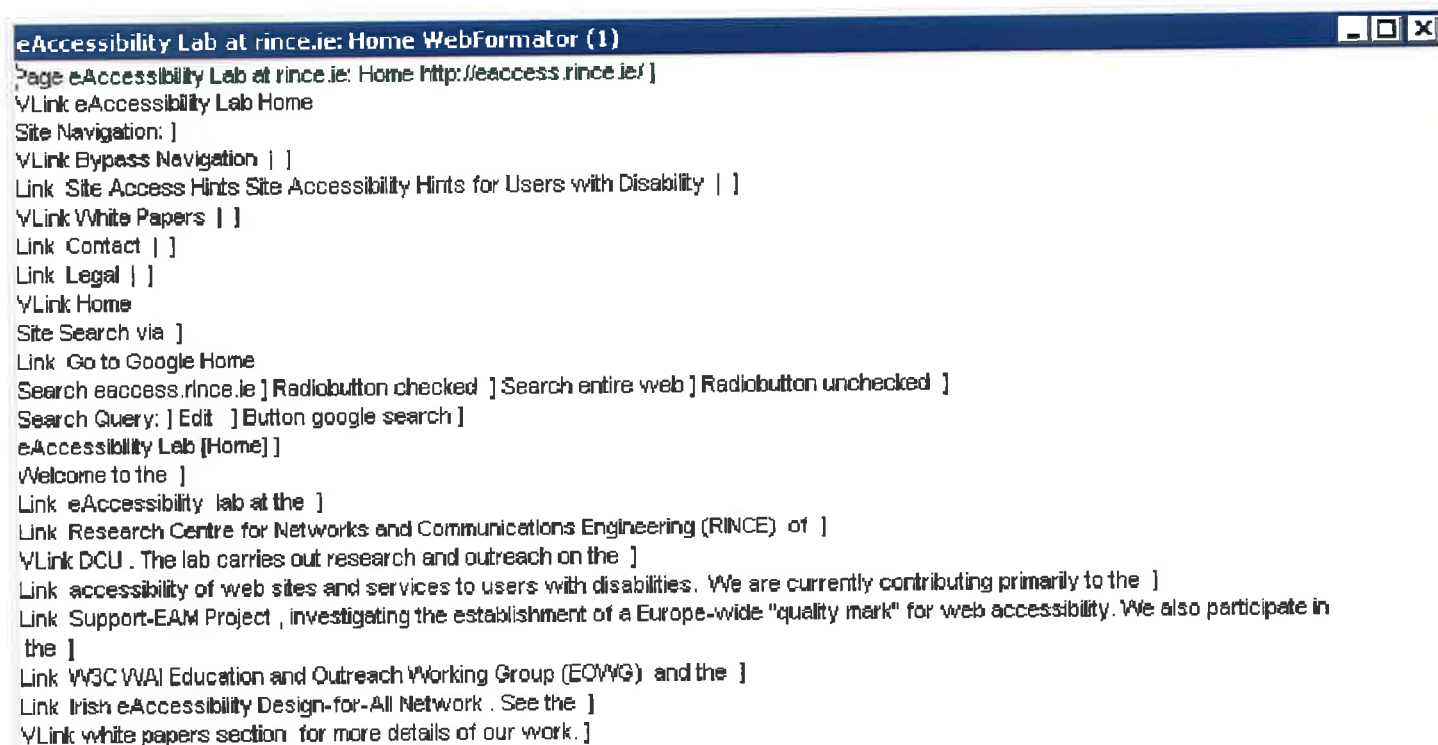


Figure 3.1: A simple web page shown through IE

Figure 3.2: An extract of the EAccess home page conveyed through Web-Formator



```

List of subjects for Computer Science course

List of 3 items (contains 6 nested lists)
• Year 1
List of 2 items nesting level 1 (contains 2 nested lists)
• Semester 1
List of 3 items nesting level 2
• Introduction to Algorithms and Data Structures
• Basic Software Engineering 1
• ...
list end nesting level 2
• Semester 2
List of 3 items nesting level 2
• Algorithms and Data Structures (Continued)
• Basic Software Engineering 2
• ...
list end nesting level 2
list end nesting level 1
• Year 2
List of 2 items nesting level 1 (contains 2 nested lists)
• Semester 1
List of 1 items nesting level 2
• ...
list end nesting level 2
• Semester 2
List of 1 items nesting level 2
• ...
list end nesting level 2
list end nesting level 1
• ...
list end

```

Figure 3.3: How lists are handled by default with JAWS

could be potentially used to signal this information. A second voice positioned in another part of the audio space to the main reading voice could be used to convey certain data *simultaneously*. However literature describing any such experiments showing the effectiveness of this approach was not available at the time of writing.

3.2.1 Using Non-Speech Cues to Denote Structure

The use of spoken cues often makes sense when imparting information concerning the current task. However, when the data is prevalent for a secondary process running in the background, signalling the user in this manner may distract them from the current operation. As is evident from section 2.2, much research has gone into the use of non-speech sound cues to provide contextual information. However, depending on the sound cues employed, there exists a *learning curve* to this approach. Much of the work already described deals with the use of sounds to denote interface changes. For example, task completion or a state change occurring in a secondary application operating in the background. Also methods in which sound can be used to impart positional information in a hierarchy were presented. However, research into the use of non-speech audio cues to signal specific screen elements have also been performed (Edwards et al., 1994; Raman, 1994, 1996*a,b*).

Edwards et al. used non-speech audio in the form of auditory icons to convey an object's type and its attributes. For example, the sound of an old-fashioned typewriter is used to represent a text-entry field, whilst a text field which is not editable (such as an error message bar) is represented by the sound of a printer. They also suggest that auditory icons can be modified to convey aspects of the interface which are presented spatially in the graphical interface such as the size of a menu or list. For example, all menus could be presented as a set of buttons which are evenly distributed along a set pitch range (such as 5 octaves on a piano). As the user moves from one menu button to another, the change in pitch conveys the relative size and current location in the menu. Finally, the labels on buttons, and

any other textual information, can be read by the speech synthesiser.

Emacspeak uses auditory icons to cue common events such as the selecting of items, or the opening and closing of objects. Raman (1996*a*) presents the scenario of retrieving and opening a *www* document as a useful example. Once a link has been activated, an auditory icon is played to notify the user before the name of the retrieved file is announced. Another example of their usage can be found when selecting a file. A sound cue is played when the status of a file is changed. For example, when a file is marked for deletion. Raman (1996*a*) tells us:

Used consistently throughout the interface, these cues speed up user interaction—an experienced user can often continue to the next task when an aural cue is heard without waiting for the spoken confirmation.

Morley et al. (1998) also employed non speech audio cues to aid in the navigation of hypermedia documents. They found that when sound cues were used in conjunction with the navigation commands, users were less likely to get lost when navigating the user interface.

Finally, non-speech sound cues have been used to signal the presence of mark-up elements e.g., <h*> elements, when encountered by the user (James, 1998; Petrucci et al., 2000; Goose and Möller, 1999; Morley et al., 1998), see also: IBM's Home Page Reader¹⁶. James used tonal pitch sequences to denote these elements. By listening to the changes of pitch in sequence, the users successfully determined the correct header levels. However, the problem with this approach is related to the length of the sound cues. If the user determined what the meaning of the element was before it was complete, they still had to listen to the cue in its entirety before continuing. They may be aware that the sound represents a header tag, but might have to listen to the entire cue before accurately determining the level. Therefore James warns against the use of lengthy cues. Also she stresses the importance of choosing sounds that have some relevancy to the element being presented.

¹⁶http://www-3.ibm.com/able/solution_offerings/hpr.html

It should be remembered that a sound that has some significance in one culture may have no corresponding meaning in another. Using snippets of well known musical compositions to convey structural data was not very successful. She attempted to map such cues to the presentation of certain form elements. However, although users could recognise the sound cues often by name, they were not easily able to map their presence to a structural element. James also warns against using numerous sound cues to represent elements of the interface. If their usage is kept low, their effectiveness is greatly enhanced.

James also tells us that different browsing strategies often require different sound mappings. If the user is skimming the page to establish its content, then prominently marking links and headers may be advantageous. However, in this situation list elements may not need to have any marking associated with their appearance. This is especially the case with index pages, where lists are often groupings of hyperlinks. Playing a sound cue for each list item in conjunction with the sound for each link would be both distracting and inefficient. She also cites the example where a user is interested in reading an entire document. In this situation links may not need to be so prominently marked. However, list elements might be important especially if they contain relevant material to aid in content comprehension. Nevertheless, James is not suggesting that each user will require a *tailored* interface to suit their needs. Instead, she recommends that a number of user interface mappings be created so that the user can choose the one they find most appropriate.

Petrucci et al. (2000) created *WebSound* which is a generic Web sonification tool allowing for the easy creation, testing and validation of new sonification models. *WebSound* combines the haptic sense with an audio output. The approach taken attempts to validate the hypothesis that a 3D immersive virtual sound navigable environment combined with haptic manipulation of the audio environment can enable blind users to construct a mental representation of the spatial layout of a document. A possible tech-

nique would be to provide different auditory attributes (earcons/auditory icons) to denote each different tag. Moreover, they believed that the use of a 3D immersive audio environment, which permits a sound to appear from a given position, may give blind users a sense of object location. Using a haptic tablet or a touch-sensitive screen would allow an individual to move his/her finger about while at the same time keeping track of his/her position on the device. The system would then respond with auditory feedback while the user moves his/her finger (device pointer) around the virtual screen.

3.2.2 Using Voice Changes to Denote Structure

Another approach to solving this problem involves the use of different voices to denote changes in context (Raman, 1994, 1996*a,b*; James, 1998; Asakawa and Itoh, 1998; Shajahan and Irani, 2005). Both JAWS/IE and IBM's Home Page reader allow for the content of certain HTML elements to be read with different voices. For example, one voice can be used to read hyperlinks, with different voices assigned to read `<h*>` elements. An alternative solution would be to manipulate voice characteristics such as stress and pitch to convey such changes (Fitzpatrick, 1999). The changes proposed by Fitzpatrick are much less pronounced in comparison to those produced by Raman, in the hope that they will not distract the reader whilst still providing contextual information. However, both approaches have their advantages and disadvantages.

Raman (1994) uses a combination of different voices and audio cues to convey information. He designed an auditory formatting Language called (AFL) which could prescribe how an element was to be presented in a non-visual modality. Using \LaTeX as the source mark-up language, elements in the object tree were assigned presentational instructions through AFL. There were nine predefined voices available with the speech synthesiser used. By manipulating characteristics of these voices such as breathiness or volume, the number of possible voices could be extended. The voice characteristics available are: the speech rate, the volumes of the speaker and the

earphone port, voice-quality parameters, and a number of parameters that deal with voice pitch and intonation. Each dimension has a default step size, which specifies the minimum change needed to be perceptible. Users could specify changes in multiples of the step size for each parameter.

Male and female voices can be thought of as lying in distinct disconnected components of the speech space, since it is not possible to move from a male voice to a female voice simply by changing parameters that affect voice quality (Raman, 1994). The reason for this is because female voices use a different noising source. Therefore, switching from a male to a female voice is analogous to changing fonts. Whereas modifying voice quality parameters is like scaling different features of a specific font.

Raman classifies the use of contextual cues into two categories, *fleeting* and *persistent*. A fleeting cue is one that does not last. Such cues are characterised by their duration being specified by the nature of the cue itself. For example, playing a non-speech sound as a bullet point signifier before an item in a list. Or, announcing the *section* and the relevant number before the *title* of a sectional element. This is sufficient to convey simple constructs; however for more complex elements this method is inadequate. A persistent cue lasts throughout the duration of the item. The duration for such cues is specified by other ongoing events in the audio rendering, rather than by the cue itself. An example of a persistent cue would be the continuous playing of an earcon (Blattner et al., 1989), whilst an abstract is being read. A second example would be changing voice parameters to read aloud each item of a list. If a *nested* list is included the voice is again altered thus the user can determine at which level the current element is present.

Switching to a new voice causes a slight pause in the speech, so it is inadvisable to change the speaking voice in the middle of a sentence, since this ruins the intonation (Raman, 1994). James agrees with the assertion, but for a different reason. She states:

Speaker changes do not seem to be appropriate for marking items when it is expected that the items to be marked will

be found within a coherent text flow. For example, subjects became confused when emphasised text that occurred in the middle of a sentence was marked using a speaker change. This is because people do not expect to hear a different person say one or more of the words in a sentence being spoken by someone else. When users hear examples of this in an interface, their attention is drawn away from the content text and towards the speakers themselves, again trying to understand the relationship between the speakers that would allow them to work together to present a single thought.

However, there exists evidence, which suggests that the number of different speakers used in a vocal presentation should be restricted to a small number. Although James (1998) found that the use of multiple voices to represent different mark-up elements had benefits, there was a marked difference when only one voice change was used, in this case to mark hyperlinks. James reasons that using different voices to mark elements is analogous to marking the same components with changes in colour. When only a small number of items are marked, the user can easily recognise that the element in question possesses an additional contextual meaning. However, in documents where many elements have different colours associated with their representation, the document may just look colourful and unmarked.

Emacspeak uses voice changes extensively to convey structural information (Raman, 1996*a,b*). This is an effective method for presenting structural information. However, it does enforce a high learning curve on the user. A couple of examples where such structural cues are used include: Emacs' "dired mode", which is used to navigate the file system and perform operations such as moving, copying and deleting files, is extended to provide succinct aural feedback. When navigating through the file listing, the user hears the name of the current file or directory spoken; different file types e.g., directories, executables and symbolic links are distinguished by speaking their names in different voices. A second example cited is the assignment

of different voices to presentational cues such as “bold” text and content within quotation marks. In terms of web based documents, many elements such as hyper links and `<h*>` elements are spoken using different voices.

Fitzpatrick (1999) manipulated prosodic characteristics of the vocal stream to convey structural information. He constrained the portrayal of this information to speech cues alone for he believed that the use of audio cues required a large learning curve. He states:

This is not a particularly intuitive mode of presentation, as the user must first learn the meaning of each of the earcons used in the system, and as there is as yet no defined standard for the use of these non-speech sounds, each system employs a different set of noises which they themselves find meaningful.

Research into the presentation of tabular data by Pontelli et al. (2002) appears to backup Fitzpatrick’s assertion that the inclusion of audio cues demands a large learning curve on behalf of the user. They found that when viewing such material, the best results were gained when spatial relationships between the cells were evident. However, when content was shown on a cell by cell basis, adding structural cues such as colour or sound tones to replace the spatial cues were found to reduce effectiveness, even when the meaning of the cues was explained. The evidence so far appears to suggest that the use of auditory cues is useful to convey certain types of information and detracts from the interface for other presentation types.

As synthetic speech does not use all the prosodic features found in natural speech, the approach taken by Fitzpatrick emphasised those prosodic elements found in synthesisers to compensate for those which are lacking. The objective of this work was not to mirror the prosodic patterns of natural speech, but to achieve a close replica which would be intuitively understood by the listener. He believed it is preferable to enhance some prosodic elements for the sake of intelligibility, than to achieve natural-sounding, though completely incomprehensible synthetic speech.

Again using \LaTeX as the base mark-up language, Fitzpatrick presented

structural information through the manipulation of prosodic cues such as pitch and speech rate. Take for example the case of sectional headers. The prosodic aspects of the voice are altered in order to convey the fact that the hierarchical level is changing. The document's title information is seen as being at level zero with the sectional headers at level one. The rate is slowed by 10% to yield a slower, more measured utterance. The average pitch of the voice is decreased by 25% to distinguish the sectional title from the remainder of the text. The pitch range is maintained at 50% of the average pitch; however the various parameters relating to the stress rise, hat rise, and baseline fall are increased. The effect is to produce a slower, though more animated voice. This is akin to increasing the font size in such headings. Ideally the amplitude is also slightly increased if the synthesis device supports it. Each sectional divider is surrounded by pauses to denote its relationship to the surrounding text. The word "section" is announced for elements at this level, however, for lower levels *subsections* and *sub subsection* elements, names are not included. In addition, as the hierarchical level decreases, the changes in prosodic parameters become less pronounced converging towards the default running voice used. Also the length of the surrounding pauses is reduced to reflect the change in hierarchical level.

Similarly different voice characteristics are used to convey emphasised text. However, Fitzpatrick operates on the premise that the important factor is the meaning implied by emphasising a piece of text rather than the type of emphasis used. Therefore no matter whether the content is marked up using bold, underlining or italics the same voice type speaks these elements. In saying this, the option is also available to use different voices to denote each type of emphasis, if this is what the user requires. Similarly, footnotes, marginal notes and parenthesised text are treated with the same voice parameters as one another. He maintains that each of these methods for noting content perform the same semantic function in written text, therefore are handled in the same fashion.¹⁷

¹⁷In this case, "parenthesised text" refers to textual content and not mathematical material, which is treated differently by the system.

Goose and Möller (1999) built a 3D audio only interactive Web browser. Using relative positioning in the sound space, they attempted to convey hypermedia document structural information. To achieve this, a combination of alternative voices and non-speech audio cues were used. They experimented with providing spatial information through the use of stereo and a moving voice source. By analysing the voice position, they conjectured that the users would be able to ascertain their position in the document. However, experiments to prove this assumption were not very successful. Therefore, they decided to deliver the main content from the centre of the audio space, with positional information announced periodically in different locations to reflect its position in the document. This was improved upon by using distinct voice types to provide each type of information. They also extensively use different voices to read the content of specific elements e.g., headers and hyper links.

Goose and Möller also use non-speech sound cues to denote contextual information. For example, spatially positioned sounds are used to determine whether a link is an internal page link or connects to an external document. Three distinct sounds, simulating taking off, being catapulted and landing notify that an internal page link has been traversed. A sound reflecting greater distance, such as a space ship taking off and landing is used to denote an external link. They also provide a facility to examine the sound space to determine the location of elements. Although this could be performed for many different elements, they currently only facilitate the examination of hyper links. Four distinct earcons are employed in the sound survey, thus allowing both position and context information to be conveyed, succinctly and in parallel, without compromising meaning. The different sounds are:

time ticker : to signify time moving by, a “tick” sound can be heard moving along the audio space from left to right. Every fifth tick is emphasised to provide a coarser granularity of time, with which the listener can identify and synchronise.

Link indicators : two earcons representing inter-document links and

internal-document links are sounded at the appropriate positions along the stereo presentation.

Sentence boundary : another sound is used to denote each sentence boundary that is encountered. The listener can use this feature to identify the links and their relative distances as measured in sentences.

If the sound survey was manually selected by the user, as opposed to being triggered as the result of a link traversal, then as the survey sweeps over the first half of the arc this should reaffirm the structural elements recently heard. The second half of the survey introduces what is yet to be heard. As the sound survey sweeps from left to right, the relative volume of the link and sentence carcons is increased and decreased to simulate the relative distance from the current position of the user (Goose and Möller, 1999). Although this method of portrayal can be time consuming, it presents information at a fraction of the time that reading the content would take.

3.3 Document Modelling

An important consideration when creating an efficient document browser is the mechanism in which material is stored internally by the application. This is especially true if the browser is intended to produce alternative views of the content. Direct access to the constituent parts of the document is required to dynamically create alternative views. Often internal data storage constructs reflect the structure of the document in some manner. However, many such mechanisms go beyond this and provide simple methods for linking to other segments of the content. The structure of a document may be expressed using one of two methods. The first method is based on the layout of the document, for example, plain text files. The second relies on the use of mark-up to define how the document is structured and hence displayed. In both cases, the objective is the same: to produce a model which accurately reflects both the content and underlying structure. Due to the WebTree

application working with documents whose structure is defined by mark-up, the discussion here will focus on the latter form of document model.

There exist many mark-up languages able to represent documents in a manner independently from their layout. Examples of these include \LaTeX (Lamport, 1985) and SGML (Maler and Andaloussi, 1995) (Standard Generalised Mark-up Language), of which HTML/XHTML are derivatives. The reasoning behind such languages is to allow the author to concentrate on writing the content, whilst having the facility to programmatically include structuring rules, e.g., sectional dividers, and parameters governing how content should be displayed. For example, the application of emphasis to a piece of text. Moreover, the notion of Document Type Definitions (DTD) has evolved from (Maler and Andaloussi, 1995). Here, a document template is defined, which encapsulates different classes of documents. For example, the majority of information written for publication on the World Wide Web is written in HTML, which is a DTD of SGML.

The major advantage of producing documents in this fashion is that the use of mark-up tags removes any ambiguity in the intended semantic interpretation. This is the case where mark-up elements are used for their intended purposes and not just to provide a purely presentational effect. Multiple programs that recognise the mark-up language can display the document in a consistent and efficient manner. This allows for the simple sharing of documents. Take for example the World Wide Web: the majority of documents are written in a derivative of the HTML mark-up language, allowing for the easy sharing of content. However, there is a second advantage to this approach. Alternative views can be created from analysing the documents' structure. These views can be based on the presence of specific mark-up elements or element groupings. In addition, navigation facilities can be based on this structural arrangement, see section 3.5 for more details.

Describing the document with mark-up tags allows for different rendering rules to be applied to the content. In terms of HTML/XHTML, this

can be achieved through the use of a cascading style sheet (CSS). Instead of physically including the font size and/or text colour in the mark-up, these attributes can be set through CSS (W3C, 2005). Linkage of mark-up elements to CSS properties can be achieved in a number of ways. The CSS content can be stored inline in the document inside the `<style>` element, or the document can be linked to an external CSS file. CSS styling can also be placed inline in the document by setting the “style” attribute of a given element. The advantage of having the CSS data stored externally is that a style sheet can prescribe the rendering rules for many documents. Thus, if the author decides on a site-wide style alteration, then manipulation of this file can reduce the work load for such changes. However, another advantage of this approach is that the *user* (as opposed to the author) can prescribe their preferred rendering rules for a specific document. For example, the users can set their favoured foreground and background colours, or the default size of text fonts. This can be very advantageous for people with vision impairments. For this reason, the Web Content Accessibility Guidelines (W3C, 1999b) request that presentational properties be set in a style sheet. This separation of presentation and content structure can result in much smaller files, for these attributes are only set in one place instead of every position in the document where they are used.

Similarly, the prescribed audio rendering can also be facilitated through the use of an aural cascading style sheet. An author, or more likely a user in this case, can set the speech preferences to dictate how elements are to be presented. The different methods available for element presentation include different voices, changes in voice parameters and non-speech sounds, which have been already discussed in section 3.2.

Typically, a tree based architecture is used to encapsulate the logical structure of documents. Using this type of structure, the hierarchical content of documents can be modelled both accurately and unambiguously. Using this form of representation, it is possible to replicate the notion of the hierarchical nature of content divisions. Take for example a HTML/XHTML

document. Assuming that the root node of the tree is the topmost level of the document (i.e., the `<html>` element), then any subsequent elements make up the different branches of the tree. Each branch of the tree can lead to other lower level branches, culminating in the lowest level leaf nodes.

However, according to Fitzpatrick (1999):

As the complexity of documents increases, it is becoming increasingly obvious that the nature of the constructs needed to contain both their structure and content need to alter to reflect this. Whereas a plain ASCII document can be modelled using a standard tree-based structure consisting of homogeneous sub-trees, a highly technical document produced by \LaTeX needs more complex heterogeneous structural components. For example, obvious distinctions exist between the textual content found in \LaTeX documents, and the mathematical content also found therein.

This is due to the formatting methods required to structure the document in a manner that is readable by the user. Text can be formatted in linear chunks of letters surrounded by white space to form words. However, this is not always possible for mathematical content where a dependence on the vertical positioning of the text is often found. For this reason Fitzpatrick tells us that two different structures are necessary to represent the different content types. Although, both forms still need to fit into the overall document model. Similar modelling problems relate to documents on the web today. Many web pages contain alternative media constructs. For example, Java applets, graphics, sound files etc. Different approaches are necessary to model these content types (Furuta, 1994). The structuring method appropriate to one type of content may not be effective or efficient for another.

Though the hierarchy of the document and all textual elements are contained in a tree based architecture, it is essential that the capability for the inclusion of alternative structures (such as linked lists) be present within any

system. Some work has been carried out on the transformation of documents prepared using one system of mark-up, into another such language. Two approaches are commonly used to achieve this goal. The first method involves the recognition of the high level structures in the form of an abstract syntax, followed by the conversion of this abstract syntax to any desired concrete syntax. Alternatively, it has been traditional to produce an output form, which comprises the least common denominator of the various input sources, and to then exchange this (Fitzpatrick, 1999).

3.3.1 Modelling Document Structure

ASTER (Audio System for Technical Reading) Raman (1994), aims to produce accurate renderings of documents marked up in the $\text{T}_{\text{E}}\text{X}$ family of languages. Unlike Fitzpatrick's TechRead system, which also provides spoken access to these type of documents, ASTER additionally uses non-speech audio to assist in this process. Although both of these systems can read the textual components of $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ files, they specialise in the reading of mathematical content. Due to WebTree not supporting technical content such as mathematics, an in-depth discussion of the handling of these constructs is out of scope for this thesis.

The component of ASTER responsible for extracting the high-level document structures can cater for varying degrees of mark-up ranging from plain ASCII files to highly complex and technical $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ documents. The document model used in ASTER is the *attributed tree*. In this model, each hierarchical level is modelled as a node in the tree, where each node can have content, children and attributes. For example, take the class *article*, as the root node. It has attributes such as title, author, abstract and date. The children of this object are the various sectional units of the document, while the *prologue* of the object consists of any text which occurs before the beginning of the first sectional unit. The leaves of this model comprise the actual content. According to Fitzpatrick (1999), the model described in the context of the *article* class can be extended to cope with the other classes

of document found in L^AT_EX, such as *report* or *book*.

Some other examples of objects includes the *section* unit. This object type has attributes consisting of title, section number and sectional name. This sectional name could be *section*, *subsection* etc. The children of this node are the subsectional units, while the prologue contains a list of document objects containing the text found before the first subsectional unit. The attributes of the *word* object type are the footnote markers (if any). There are no children. The contents are made up of the string, which is the actual word.¹⁸

ASTER differentiates between the textual and mathematical content of a document; the former being represented by a set of ordered textual objects, while the latter is modelled using the `InlineMath` object. Each node is linked to both its parent and siblings. For a full list of the different objects supported by ASTER see chapter 2 of (Raman, 1994).

The Techread system proposed by Fitzpatrick (1999) models the content using a `cross linked` tree. It differs from Raman's system by using a model more graph like than tree structured. Each document unit is represented by a distinct object. Fitzpatrick (1999) tells us that this idea sprang from an article submitted to the first annual conference on the theory and practice of "digital libraries" in 1994 (*DL94*), by Richard Furuta (Furuta, 1994). Furuta believed that in the case of the digital library, material prepared using diverse composition systems, and using a range of structures to contain their information, would have to be inter-linked to form the whole information space. According to Fitzpatrick, Furuta says "Heterogeneous data structures may be used to describe different elements of an information space. . . . When multiple structures are defined over a set of contents, the general question is whether they are interrelated in any way". This paradigm can be applied to the universe of technical documents, where many diverse elements (textual, mathematical and tabular objects to list but three) are combined to present an author's material (Fitzpatrick, 1999).

¹⁸Aster can be easily extended by defining new objects and adding rendering rules for their presentation.

To account for the need of linkage between heterogeneous data structures Fitzpatrick used a set of distinct objects derived from a base class *section* using the *object-oriented* design paradigm. Using inheritance, objects can be recursively defined, for example, a sectional unit can contain subsections etc. In this way, all objects possess a number of common functions, such as links to parent or sibling elements, in addition to the elements specific to the object in hand. For example, the `TableObject` contains additional links to the cells adjacent to it, be they in the same row or the same column. The internal document model is organised in a hierarchy of these elements. At the top is the root node known as the `global document settings` node. This node contains all definitions and assignments which are deemed to be global to the entire document. This could include such items as the default font used, the default speaking voice used throughout the document etc. At a logical level below this are found the nodes containing sectional units. These are described as non-terminal nodes, for Fitzpatrick believed that there would always be at least one sectional unit below the `global document settings` object. Further down the hierarchy than the *sectional* nodes, are found the actual *terminal* nodes of the model. These nodes contain the content, and the associated formatting; both visual and audio. However, the means in which this system departs from the traditional tree implementation is that the objects can contain additional connections to elements in other parts of the hierarchy. For example, the last paragraph in a section would have a connection to the first paragraph in the next section, or the next element at the same logical level.¹⁹ Due to these additional connections, Fitzpatrick describes the document model as a series of *independent*, though integrated objects, which contains the ability to navigate from any point to any point within the overall hierarchy. These additional connections are similar to internal document hyperlinks (as found in HTML/XHTML), except that they are reflected in the document model as opposed to the document mark-up.

¹⁹Elements can have a connection to any type of element and not just those of similar make up.

3.3.2 Conceptual Graphs

The knowledge representation scheme proposed by Pontelli et al. (2002); Pontelli and Son (2002, 2003) is based on Conceptual Graphs. They cite (Sowa, 1984) as providing a good description of these constructs. Conceptual Graphs and their associated theory were proposed in the 1970s as a way of drawing logical statements in diagrammatic form rather than in a linear text-based calculus which was and is the norm. The basic ontology is very simple as it is in logic. A conceptual graph can have two kinds of node; a concept node that represents types and objects of that type and a relation node that represents a relationship between these objects. The theory allows for a basic expressiveness that is equivalent to first-order logic as well as mechanisms for defining concepts and for representing type hierarchies of concepts and relations. Researchers have extended the formalism to allow representation and manipulation of more advanced ontologies, especially those involving actions and events, and higher level structures such as viewpoints, and nested contexts (Pontelli et al., 2002).

The concept nodes of the graph represent the semantic entities which are described by the document's component. Nodes are commonly organised according to one or more hierarchies. The lower level of the hierarchy commonly includes syntactic elements directly extracted from the document (e.g., cells of a table). The higher levels of the hierarchies provide semantic entities representing general concepts or collections of concepts (e.g., a column is a collection of cells). The edges of the conceptual graphs represent relationships between the conceptual entities identified as nodes of the graphs. A natural class of relationships originates from the presence of a hierarchy between different concepts present in the graph (Pontelli et al., 2002). The conceptual graph representing a document component in this system (e.g., an HTML table) is created by combining three sources of knowledge:

1. Syntactic content of the document (e.g., use of HTML tags and attributes).

2. Direct input from a human annotator (e.g., the teacher, the creator of the document, a third party)
3. History of how the documents components have been used in the past.

3.3.3 The Document Object Model (DOM)

The Document Object Model, or DOM, is a platform independent interface incorporated into many web browsers to allow for programmatic access and manipulation of document content. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term “document” is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data. The DOM is a programming API for documents, which is designed to be used with any programming language.

In the DOM, documents have a logical structure which is very much like a tree; to be more precise, it is more like a “forest” or “grove”, which can contain more than one tree (W3C, 2004a). It provides a structured, object-oriented representation of the individual elements and content in a page with methods for retrieving and setting the properties of those objects. It also provides methods for adding and removing such elements, allowing for the creation of dynamic content. The DOM also provides an interface for dealing with events, enabling the capture and response to user/browser actions. However, the focus of this discussion will be on the DOM representation of a document and the methods it provides to access those objects. The latest version of the DOM specification is the “Document Object Model (DOM) Level 3 Core Specification” (W3C, 2004a). For an introduction to the DOM structure, see the article: Introduction to the Document Object Model²⁰.

²⁰<http://www.brainjar.com/dhtml/intro/>

As a web page is loaded into a DOM compliant browser, a hierarchical representation of its contents which closely resembles its mark-up structure is generated. This results in a tree-like organisation of nodes, each representing an element, an attribute, content or some other object. Each of these different object types will have their own unique methods and properties. However, a common set of methods and properties related to the document tree structure are included as part of each node. The document object serves as the root of this node tree. In addition to the properties and methods common to all nodes, the document object is also responsible for the implementation of the “document interface”. This interface provides methods for accessing and creating other nodes in the document tree. Some example methods are:

- `getElementById()`
- `getElementsByTagName()`
- `createElement()`
- `createAttribute()`

As mentioned previously, the DOM tree reflects the structure of a document’s underlying mark-up hierarchy. Every element is represented by an `element` node with other nodes representing attributes or character data (i.e., text). Technically speaking, the document object has only one child element. In the case of web pages, the root node represents the `<html>` element of the document tree. Its DOM element will contain links to child elements for the `<head>` and `<body>` tags which in turn will have other child elements. By following these element links, and/or by invoking the designated methods of the node interface, you can traverse the document tree to access individual nodes within the structure. However, there are some obvious problems with accessing nodes in this manner. For one, a simple change to the page source, like adding text or formatting elements or images, will change the tree structure. Therefore, the path used before

might no longer point to the intended node. This would mean the method of accessing the content is inefficient because of its unpredictability. A new set of traversal steps must be performed each time to ensure access of the correct content. There are also some issues pertaining to browser compatibility. If some simple line breaks are included in the documents' source to separate elements, some browsers may add nodes to represent this data e.g., Netscape, whereas others such as IE do not. When these nodes are included they are not generally afforded names for they do not represent mark-up elements. However, they do show up when cycling through the child list for a given element. For this reason, powerful navigation methods such as the `getElementById()` method are rather useful. By adding an ID attribute to the paragraph tag (or any tag for that matter), you can reference it directly. This way, you can avoid compatibility issues and update the page contents at will without worrying about where the node for the paragraph tag is in the document tree.²¹ A less direct method to access element nodes is provided by the `getElementsByTagName()` function. This returns an array of nodes representing all of the elements on a page of the specified HTML tag type. For example, all the `<a>` elements.

There are several types of node defined in the DOM, but the ones most common for web page handling are `element`, `text` and `attribute`, see the article: *Introduction to the Document Object Model*²². `Element` nodes correspond to individual tags. They can have child nodes, which may be other elements or text nodes. `Text` nodes represent content, or character data. They will have a parent node and possibly sibling nodes, but they cannot have child nodes. `Attribute` nodes are a special case. They are not considered a part of the document tree - they do not have a parent, children or siblings. Instead, they are used to allow access to an element node's attributes. That is, they represent the attributes defined in an element's HTML tag description, such as the `href` attribute of the `<a>` tag or the `src` attribute on the `` tag.

²¹Each ID needs to be unique to the document.

²²<http://www.brainjar.com/dhtml/intro/>

Webbie (King et al., 2004b,a), and IBM's Home Page Reader (HPR)²³, are two examples of web browsing solutions that use the DOM as their preferred method for internally modelling the content.

3.4 Current Non-Visual Web Browsing Solutions

Solutions to the problem of web accessibility for blind people fall into a number of categories: reliance on a conventional web browser and a screen reader to directly access a website; using transcoding proxy servers to convert web page HTML into a more accessible format²⁴; and using a dedicated web browser to convey the information (King et al., 2004b). King et al. also consider the use of HTML accessibility features, such as changes in colour or font size to enable access in their list of solutions. However, these accessibility features are generally more helpful for people with some vision, and therefore are out of scope for this discussion. The following subsections give a brief description of a number of current web access solutions for the blind. However it should be remembered that this is not an exhaustive list.

3.4.1 Conventional Browser with a Screen Reading Application

Microsoft's Internet Explorer (IE) is the most popular graphical web browser available today for the WINDOWS platform. For this reason, many Web site providers optimise their sites to be viewed through this application. Therefore, using a screen reading application in conjunction with IE should in theory provide the greatest level of access to the largest amount of web pages. However, the problem occurs in the means of data presentation. Reading text as it is presented on the visual display is not practical due to the complexity of constructs involved, e.g., tables and forms. Therefore,

²³http://www-3.ibm.com/able/solution_offerings/hpr.html

²⁴A screen reader/auditory browser is still necessary in this situation, however, the page is reformatted to be optimised for screen reader access. This reduces the dependency on built-in functionality in the screen reader application to aid navigation, allowing less advanced applications to gain access to the content.

other methods of access to the underlying content need to be employed. The two main options under the windows environment are to use MSAA technology (Sinclair, 2000) to query the construction of the screen content. The second method requires direct access to the internal document model (DOM) (W3C, 2004a). Either way, without access to methods to obtain structural data about the elements, web access for blind people would be an arduous task.

JAWS for Windows²⁵ from Freedom Scientific²⁶ in conjunction with Microsoft's Internet Explorer, is a prime example of a solution classed in this category. It is one of the most popular screen reading applications in use today. Although it is designed as a general screen reading application, it contains optimisations to enable it to present web based content. Using MSAA technology (Sinclair, 2000), JAWS can access information presented in Internet Explorer. JAWS version seven can also access the Firefox²⁷ browser from the MOZILLA foundation^{28,29}. Under the JAWS and IE combination, the content is presented in a linear form to the blind reader, whilst still retaining its visual presentation in the browser window. This is the case when reading the document with the regular navigation functions, such as, by line, sentence or with larger blocks of text e.g. paragraphs, (discussed in section 3.5), or when reading the content continuously. However, there are additional methods that aid navigation in a non-linear fashion, e.g. the table navigation functions, discussed in section 3.6, and the ability to tab from link to link, or jump to the next/previous form field/control.³⁰ Depending on the type of functionality requested JAWS either announces the names of specific elements such as links or header elements in the spoken output, or uses different voices to convey this information. See section 3.2 for more de-

²⁵http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

²⁶<http://www.freedomscientific.com>

²⁷<http://www.mozilla.com/firefox/>

²⁸<http://www.mozilla.com/>

²⁹However, in reference to this thesis, when JAWS is mentioned in conjunction with web technology, reference to the JAWS and IE combination is intended, unless explicitly stated otherwise.

³⁰Table navigation functions in JAWS only effectively work with *data* tables.

tails. Hyper links and form fields are presented on separate lines from other content. This is done to aid in the ease of both recognition and selection for interaction. Figure 3.4 shows a simple web page as rendered by JAWS. JAWS typically does not insert additional text concerning the element type so that it is navigable in the audio rendering. The two major exceptions to this rule are the provision of `list` and `<table>` information. However, for the purposes of this discussion, contextual information produced by JAWS that is not navigable in the auditory rendering is shown in figure 3.4. This is done to show how JAWS conveys the different web page elements where voice changes are not used. Figure 3.1 illustrates how the same web page would appear visually in Internet Explorer. Users can also choose to have the presence of `<table>` or list elements announced automatically. Tables used on web pages for layout are presented in a linearised format to the user. The linearisation is done on a row by row basis. Although data tables are also presented in a linear form, as previously mentioned additional navigational functionality is provided to ensure the user has access to the spatial organisation of the construct. See section 3.6 for more details.

3.4.2 Transcoding Proxies

The second approach places the solution between the author and the client by running requested HTML pages through a transcoding proxy server. Requests for web pages from servers are made not to the servers themselves but to an intermediate proxy, which fetches the page, converts it according to a set of rules, and returns the transformed page to the client. Pages can have their structure altered to present a more optimal solution, depending on the circumstances. For example, the *betsie* project (BBC Education Text to Speech Internet Enhancer). This is the filter program currently used by the BBC to create an automatic text-only version of its website. It takes graphical HTML pages as an input and strips out the graphical components to produce a fully functional text only version of the web page. It also linearises tables by removing the `<td>` tags and replaces them with `
` tags

eAccessibility Lab at rince.ie: Home
visited link graphic eAccess
heading level 1 Site Navigation:
 List of 6 items
this page link Bypass Navigation |
link Site Access Hints |
link White Papers |
link Contact |
link Legal |
visited link Home
 list end
heading level 1 Site Search via
heading level 1 link Google
radio button checked Search eaccess.rince.ie *one of two*
radio button not checked Search entire web *two of two*
 Search Query:
edit
 google search *button*
heading level 1 eAccessibility Lab [Home]
 Welcome to the
link eAccessibility
 lab at the
link Research Centre for Networks and Communications Engineering (RINCE)
 of
link DCU.
 The lab carries out research and outreach on the
link accessibility of web sites and services to users with disabilities.
 We are currently contributing primarily to the
link Support-EAM Project,
 investigating the establishment of a Europe-wide "quality mark" for web accessibility. We
 also participate in the
link W3C WAI Education and Outreach Working Group (EOWG)
 and the
link Irish eAccessibility Design-for-All Network.
 See the
link white papers section
 for more details of our work.

Figure 3.4: A simple Web page conveyed through JAWS

to prevent the cells from running into each other. In addition, the BBC specific navigation bar presented on the left of the printed version is relocated to the bottom of the document, for many screen reading applications present such content at the beginning of the web page. However, it does have problems with the use of both Java and JavaScript. More information can be obtained from the Betsie home page³¹.

Brown and Robinson (2001) created the Web Access Gateway³² as another such example. It transcodes already existing Web pages automatically on the fly for use by visually impaired users. They attempt to remove images to create a text only page; all the text information can be serialised by removing tags; they allow for changes in font size and colour settings; alternatively, they also facilitate the enlargement of images and so on. With regard to serialising the text, Home Page Reader and JAWS have already implemented this functionality on the *client-side*. Whereas the Web Access Gateway and Betsie perform content serialisation on the *server-side*.

Another example can be found in the work of (Asakawa and Takagi, 2000; Takagi et al., 2002). Asakawa and Takagi developed an *annotation-based* transcoding proxy system to convert already-existing Web pages to an accessible form. The system consists of two components, one for *structural* annotations and one for *commentary* annotations. Structural annotations are used to recognise visually fragmented groupings as well as to show the importance and basic role of each group. Commentary annotations are used to give users a useful description of each grouping. The objective was to reorder visually fragmented groupings according to the importance of each grouping and give useful descriptions for voice output. The annotations are provided on a "URL" basis by volunteers. However, if pages are similar in construction the same annotation can be used for multiple pages. It achieves this by calculating any similarities in the DOM trees between particular URLs. Hand crafting each annotation file with a text editor would be an arduous task. For this reason, a *wysiwyg* (what you see is what you get)

³¹<http://www.bbc.co.uk/education/betsie/>

³²<http://www.accu.org/cgi-bin/access/access>

editor was developed to assign such annotations to HTML fragments. The operator selects the elements in a group and the application selects the lowest level node incorporating these elements for the annotation assignment. The importance to be attributed to the node can be selected at the same time. An image map is then inserted by the system at the beginning of each page to act as a table of contents for the newly assigned groups. The role of each group is contained in the alternative text for the `<area>` tags in the image map. Manually creating annotation files by volunteers is a time consuming task. For this reason Takagi et al. (2002) attempts to automate the system by creating much of the groupings and assigning annotations automatically. They developed a new algorithm, "Dynamic Annotation Matching", so their transcoding system could automatically determine appropriate annotations based on each page's layout. They also developed a site-wide annotation-authoring tool, "Site Pattern Analyzer." They analysed the feasibility of creating site-wide annotations by using both the algorithm and the tool. By automating much of the annotation process they showed that the time taken to create annotation files for HTML pages could be drastically reduced.

King et al. (2004b) also warns us that the intermediate proxy approach has some disadvantages arising from the second-hand nature of the HTML document transmission:

... the processing performed by the proxy server requires the server to have full access to the content of the HTML document, which means that secure transmission protocols used in Internet commerce such as HTTPS are unusable.

Another approach often used is to offer an alternative accessible version of a website optimised for non-visual usage. However, one major criticism of having alternative views is that these pages are not as regularly updated as those on the main site. Plus, they don't always afford the user with the same level of functionality. Hoffman et al. (2005) tell us that the solution optimal for one user type may render the page less usable for another

group. Nevertheless, they do not suggest separate applications or pages for groups with special needs because they recognise that a legitimate concern exists that such dual source applications or separate interfaces would not be designed or maintained equally. Instead, they believed that emerging technologies that enable dynamic and customised views of the same application page may present an opportunity to improve the overall user experience. They propose some guidelines on how architecture technology can be used to generate accessible pages. An example of such changes includes the addition of internal page links at the beginning to allow quick navigation to the different sections of the document. Alternatively, the often lengthy navigation bar could be re-positioned to feature at the bottom of the page. This would save the screen reader user trawling through such material to find the main content.

3.4.3 Dedicated Browsing Solutions

The final approach is to use a dedicated web browser designed for visually-impaired or blind people. There are two different approaches regularly employed: the first type being a self-voicing application that provides a complete audio interface to web pages, for example, IBM's Home Page Reader (HPR)³³. The second is to render the content of a web page as a text-only flat document and permit the user to access this content using their normal assistive technology, typically a screen reader. This method is demonstrated by WebbIE (King et al., 2004*b,a*) and WebFormator³⁴ from Frank Audio-data. According to King et al., developing a dedicated web browser affords the maximum flexibility in approach, but requires the developer to take more responsibility for the presentation of web content. Although in theory a non-visual web browser is just as standard as a visual one presenting marked-up HTML, in practice the visual bias of the web means that alternative applications have to focus on providing access to resources designed

³³<http://www-3.ibm.com/able/solution.offerings/hpr.html>

³⁴<http://www.webformator.com/englisch/index.php>

for the sighted. What follows is a brief description of a number of these applications.

IBM's Home Page Reader (HPR) is a standalone web browser that segments a web page into a linear array of items which can be traversed by the user and are spoken as they are encountered. The user can select the granularity of the presentation, from single characters to entire elements such as paragraphs. A different voice (female as opposed to male) is used to present links to distinguish them from other content. HPR works directly with the document's DOM structure, and therefore is able to generate views optimised for blind people, as opposed to translating a visual interface. However, HPR does still show the full graphical document view, although it is shown in a different window to the text content being spoken. The default setting presents the page as an array of structural mark-up elements: list items, headers, and paragraphs. This is a good level of resolution for well-constructed web pages, since it allows the user to immediately access the document via a reasonable number of segments which reflect the semantic meaning known to the document author. However, poorly designed web pages where mark-up is used purely for visual presentation are presented less successfully, since there is less scope for inferring the semantic meaning of particular items of content from the mark-up (King et al., 2004b).

BrookesTalk³⁵ (Zajicek et al., 1998a) is another self-voicing web browser. In addition to the regular problem of presenting information relating to elements, it attempts to address the problem of communicating to the end user the semantic content of a page by providing summaries and keywords obtained by analysing the structure of the text. However, blind users did not find the summary information very useful because it was often inaccurate (Zajicek et al., 1998b).

WebFormator from *Frank Audiodata* is a helper application running simultaneously with MSIE. It represents the contents in a text field that can be accessed by a screen reader. This text can be navigated as a normal text

³⁵<http://www.brookes.ac.uk/schools/cms/research/speech/btalk.htm>

based application, and like the other two applications, users can bring up lists of links, frames and other features that can be of use in understanding the content of the web page. Text describing elements such as links are added to the presentation. The user can peruse this text along with the content contained within. WebFormator and HPR also provide different navigation modes for exploring HTML tables. This is useful when navigating from cell to cell within data tables. See section 3.6 for more details on the types of tabular navigation offered. Figure 3.2 shows a simple web page shown through WebFormator.

Webbie (King et al., 2004*b,a*), uses the same tactic as WebFormator, presenting the web page content as accessible text rather than self-voicing an entirely novel interface. It goes a step further in creating a freestanding independent application providing web access. Webbie navigates the DOM, collecting active content components such as hypertext links and form components, and building up a plain-text representation of the content to be presented to the user. As with WebFormator, components are presented on new lines with distinguishing titles, such as, "LINK" for a hypertext link. Functionality is accessed through pressing the return key on a line with a presented component. Webbie supports existing MSIE bookmarks, frames, the great majority of HTML 4, forms, tables, and the display of embedded multimedia. Webbie looks much like Internet Explorer; however, the web page is presented in purely linear text, like a Word document. However this can be problematic where elements depend on spatial layout to denote meaning.

The final application to be mentioned is the emacs-w3 browser in conjunction with emacspeak. No linearisation is performed on the document by default. Tabular content is presented on the basis of its visual representation, with additional commands to isolate and read individual columns. Similarly, links are presented as they would be visually. That is, no additional line breaks are inserted to denote breaks between linked text. A different voice is used to announce hyper link content. Also, headers and

other structural elements are assigned different voices to signify changes in context. However, emacspeak allows for XSLT translations to be performed on the content to render it into a format preferred by the user. A number of style sheets are supplied so the user is afforded control over the document rendering. Also, if these do not meet the needs of the user, they are free to write their own XSLT translations to suit their individual requirements.

3.5 Web Page Navigation and Summarisation

Listening to content through a text to speech engine (TTS) could be described as being akin to listening to a pre-recorded spoken version of the content. As with digital recordings, an ability to navigate to and focus in on small segments of the document for closer scrutiny is imperative (Arons, 1997). However, unlike recorded content, the dynamic nature of TTS engines allows for a fine granularity of access to the material. The user can navigate through the material character by character, or through larger letter groupings e.g., words or entire lines. In addition, the ability to read based on punctuation delimited segments such as entire sentences is often provided. Also, functionality to navigate using larger screen blocks such as paragraphs or virtual pages is offered by many screen readers. The system proposed by Morley et al. (1998) includes a single reading mode based on many of these units. However, it is up to the user to specify the granularity at which the content is to be read. That is, the unit on which the reading functions are based is selected by the user. JAWS for Windows³⁶ also allows navigation based on these units. However, whereas the system proposed by Morley et al. has a single set of keystrokes to navigate units, JAWS assigns such functionality to a number of different keystrokes. This negates the need to actively switch reading modes when a change in navigation is required. However, it does mean the user is required to remember a much larger set of keystrokes.

In terms of web-based documents, sometimes additional navigational

³⁶http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

functionality based on a subset of mark-up tags is included. For example, with JAWS for Windows, pressing the letter 't' manoeuvres focus to the next table construct. Whereas, typing the letter 'f' jumps to the next form field/control. However, the number of elements for which this type of navigation functionality is provided is often restricted to those elements perceived to be beneficial by the developers.

IBM's Home Page Reader³⁷ allows navigation based on the traditional navigation modes e.g., by letter, by word etc. However additional navigation modes based on units linked to HTML constructs is provided. The user can choose to move by element such as paragraph, link, header, etc. The user can also use a number of modes that facilitate movement to the next /previous header element, link, or form control. The same keys are used to navigate to each element, once the user has selected the mode of interaction. The ability to navigate tabular constructs will be discussed in section 3.6.

In addition, many applications allow the user to generate lists of elements, such as hyper links, e.g., WebFormator, Home Page Reader, BrookesTalk³⁸ and JAWS for Windows that can be searched and navigated by the user. This allows for quick access to the hyper link content. This is especially valuable when viewing an index page whilst searching for a specific link. Also the ability to create a list of the header tags on a page is often provided, e.g., JAWS for Windows and BrookesTalk. Finally, a simple jump mechanism to skip past links to the next piece of textual content is sometimes included, e.g., Webbie and JAWS for Windows.

Although these navigation modes allow the user to manoeuvre through the material using text chunks of different sizes, the problem of not being able to determine the page structure without examining the entire web page still exists. Therefore, to circumvent these issues, many screen reader developers equip their applications with a page summarisation mechanism. These summaries are frequently based on the presence of specific named elements such as headers, hyperlinks, or by previously bookmarked page segments

³⁷http://www-3.ibm.com/able/solution_offerings/hpr.html

³⁸<http://www.brookes.ac.uk/schools/cms/research/speech/btalk.htm>

(Zajicek and Powell, 1997; Zajicek et al., 1998a). Others allow navigation based on these elements Goose and Möller (1999). Others have experimented with page summarisations based on the first line/sentence within a paragraph, or views containing entire paragraphs comprising specified words or phrases. For more information see the article: Surfing the Internet with JAWS³⁹. An alternative approach is to examine the sentence structure of the text and generate page summaries of sentences containing the most frequently used word trigrams (Zajicek et al., 1998a). Unfortunately, this method is rather error-prone, for trigrams less pivotal to comprehending the page contents may feature greatly in the summary, whilst less used more explanatory sentences are excluded.

Another approach to the presentation of web-based information is the “Audio Enriched Links system” described in (Parente, 2004). The audio enriched links mechanism provides a spoken preview summary of a linked web page. This is done before the link is followed by the user. The page summary is comprised of its title, its relation to the current page, statistics about its content, and some highlights from its content. Both JAWS for Windows and Home Page Reader perform a more limited summarisation function after a web page has already been opened. The number of certain elements such as links, header elements, frames and tables can be automatically announced to the user. The Hearsay system attempts to automatically partition Web documents through tightly coupled structural and semantic analysis Ramakrishnan et al. (2004). The raw HTML documents are transformed into semantic structures so as to facilitate audio browsing. Voice XML dialogs are automatically produced from the XML output of partitioning. The auditory formatting language proposed by Raman (1994) allowed for the voice setting for a given element to be set to “none”. In this manner, certain elements could be excluded from the auditory rendering.

The system discussed by Goose and Möller (1999) downloads pages linked to the current page in the background and retrieves meta information

³⁹<http://tinyurl.com/rqzu4>

about these pages to provide better information concerning the link's destination. They believed that an automatically generated document summary would have been too verbose and distracting. However, some basic meta-information could be of value to a listener. This might include the title and the time required to listen to the content. An additional synthesised voice was introduced into the audio space to announce this information when a link was encountered.

3.6 Tabular Constructs

Although the mark-up describing the `<table>` construct is tree-like in nature, it poses serious problems for a purely tree-like interface. The optimum methods for presenting and assimilating tabular information relies heavily on its two dimensional (regular graph as opposed to tree) like organisation. Aside from tree related issues, tabular data poses many problems for speech interaction in general. This is due to the limited view imposed by the serial nature of the modality. It is difficult to build a mental model of how the cells relate to one another. Thus, research into the best methods for conveying such material through the spoken medium has resulted in many diverse access strategies.

The visual organisation of the material often takes on a grid like arrangement of horizontal rows and vertical columns of data cells. The data cell is the basic building block for creating these complex structures. In XHTML, the table cell can take on either one of two forms, the data cell `<td>`, and the table header cell `<th>`, which attempts to impose some semantic meaning on the content. Mechanisms to explicitly link the header cells with their associated data cells are available in the XHTML specification (W3C, 2002b). According to W3C (1999b), to make table constructs accessible, the use of `<th>` to mark up header elements, and `<td>` for data cells is mandatory. The use of the `<thead>`, `<tfoot>` and `<tbody>` elements to group rows and the `<col>` and `<colgroup>` to group columns is also required where appropriate to associate data with header cells.

In terms of the World Wide Web, `<table>` element usage can be classified into two different categories. The first category of presentation is *layout* or *formatting* tables, used to organise the information into multi column presentations. Knowledge of the semantic structure is not usually necessary to comprehend the information. Instead, their usage is mainly to achieve a particular visual layout. The W3C's Web Accessibility Guidelines (wcag) discourages the use of `<table>` elements to format content in this way. Instead, it favours the use of style sheets to perform such tasks (W3C, 1999b).

The second type of table is known as a *data* table. These constructs are used to format data which depends on the semantic relationships between the cells to illustrate meaning. An example of a data table can be found in figure 3.5.

Data tables can be further categorised into two types based on their levels of complexity. A *simple* table has at most 2 header cells (`<th>` elements) associated with each data cell (`<td>` element), i.e., one for the row and another for the column in which it appears. Whereas, data cells in a *complex* table can have multiple logical row/column header relationships⁴⁰ Figure 3.5 shows a simple table construct. In addition, a single cell may be expanded to encompass more than one row or column. Thus its semantic significance must be gleaned from its positioning in relation to other cells.

To further complicate the issue, HTML/XHTML allows entire tables to be nested within a single cell, adding to the complexity of the structure. Although this is commonly useful when tables are used for layout, it is not so clear that this is ever useful for data tables. It would certainly cause severe difficulty for any speech based browser to render in a meaningful manner. It is not clear whether it is formalised anywhere that data tables can't or shouldn't contain nested tables. However, at least one automated evaluation tool AccVerify⁴¹ has an option to use nesting as a heuristic indication that

⁴⁰These tables are not categorised based on their number of rows/columns, for both types of table can be quite large in structure.

⁴¹<http://www.cynthiasays.com/About%20Reports/DataTables.htm>

WCAG-A Conformance Failure Rate:	149/159	93.7%
WCAG-AA Conformance Failure Rate:	159/159	100.0%
WCAG-AAA Conformance Failure Rate:	159/159	100.0%

Figure 3.5: Table showing the levels of accessibility in Ireland in 2002. Taken from (McMullin, 2002b).

a table is *not* a data table.

As Yesilada et al. (2004); Spiliotopoulos et al. (2005) have pointed out, tables have no aural equivalent and they can be considered as one of the natural functions of written language. For visual interaction, tables are a rather efficient and effective method to format quantities of data. They easily demonstrate the manner in which items relate to one another. For visually impaired users, in contrast, browsing tables can become an arduous task. The print layout may hinder rather than help access through such an interface. This is because the richness of visual structure is not appropriate or accessible to visually impaired users. The structure, presentation, content and spatial cues such as labelling of tables enable many tasks to be performed and to guide the reader in, for instance, analysing and finding data items. According to Yesilada et al., the retrieval of tabular content is processed by current auditory solutions using three different methodologies:

1. Screen scraping: material is retrieved from the visual rendering of conventional browsers, with the table content often being read line by line across the page in a similar fashion to ordinary text.

2. Applications such as BrookesTalk⁴² and IBM's Home Page Reader⁴³ have direct access to the source HTML mark-up, so are able to provide greater knowledge of specific mark-up element structures such as tables to the user.
3. Special applications are employed to transform the visual presentation of tables into another form, so that screen readers can render them more suitably. Different approaches are taken. Oogane and Asakawa (1998) create an HTML file for each cell, which contains only text. Chisholm and Novak (1999) use a script to cause another Web browser window to open, which contains the text, as well as column and row headers and position information for each cell.

Once the application has gained access to the relevant tabular information, the next question is how best to convey this material in a comprehensive and meaningful manner. One method is to read the elements sequentially row by row. However, this form of interaction can pose a number of problems. When reading horizontally across the line, the user may have to listen to the content of many cells before the required information is found. It may also be difficult to distinguish the contextual breaks between the cells. Also, the content may be organised in a format which requires reading by column to ensure comprehension. Reading content in this manner means the listener is generally a passive participant in the listening process. Therefore, due to the complexity levels possible with such data constructs, a more interactive method of gleaning the information might be preferable.⁴⁴

Raman (1994) proposed exploiting *stereo* (spatial audio) to indicate the location of the cell. The first element of each row is spoken solely on the left speaker; the rendering then progressively moves to the right, with the last element spoken solely on the right speaker. Spiliotopoulos et al. (2005) examined the use of prosodic cues to model the underlying semantic structure

⁴²<http://www.brookes.ac.uk/schools/cms/research/speech/btalk.htm>

⁴³http://www-3.ibm.com/able/solution_offerings/hpr.html

⁴⁴JAWS for windows does allow the user to listen to an entire row at a time. However, this is not the default behaviour of the application.

of tabular constructs. They presented an experimental study concerning the spoken presentation of data tables. Human readers were asked to read aloud both simple and complex sample tables which were then evaluated by blind and sighted listeners alike. They analysed prosodic parameters in terms of boundary tones and pauses in the hope that these could clearly illustrate consistency against cell content and visual structure. They deduced a specification which they believed could form the basis for the auditory scripting of tabular models. They speculated that this would aid in the automatic rendering of such data using synthetic speech. Although this would be useful in demonstrating the breaks between individual cells when reading each row sequentially, it doesn't offset problems relating to the amount of content the user must listen to before reaching the required cell. In saying this, the use of prosodic cues could be very useful when imparting contextual information such as the end of a row, or where a cell's content begins and ends.

Alternatively *transformations* are sometimes performed on the content to aid comprehension. One method often used transforms complex tabular constructs into a linear format (King et al., 2004*b,a*; Yesilada et al., 2004). The linearisation technique is often a reasonable approach to the presentation of tabular data when used for purely layout purposes. However, this approach is insufficient for conveying information in data tables, for much of the structure that the grid-like construct provides is lost. That is, when viewed in their linear form it can be difficult to establish the relationships between the individual cells in the grid (Pontelli et al., 2002; Yesilada et al., 2004).

Table linearisation can be achieved in a number of different ways. One solution would be to base transformations on the order in which elements appear in the structural mark-up, with each cell presented in order from left to right on separate lines to other content, e.g., JAWS for Windows⁴⁵. Figure 3.6 shows the same table as seen in Figure 3.5 when linearised through JAWS. However, row by row linearisation is not always an effective solution.

⁴⁵http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

```
Table with 3 columns and 3 rows
WCAG-A Conformance Failure Rate:
149/159
93.7%
WCAG-AA Conformance Failure Rate:
159/159
100.0%
WCAG-AAA Conformance Failure Rate:
159/159
100.0%
table end
```

Figure 3.6: Table shown in figure 3.5 linearised through JAWS

Sometimes it makes more sense to present the content column by column. A second solution is the approach taken in *Tablin*⁴⁶. *Tablin* is a filter program developed by the WAI Evaluation and Repair group that can linearise HTML tables and render them accordingly to preferences set by the presentation layer (e.g., the screen reader end-user). It provides a lineariser entry form that allows the user to handcraft a table's transformation.

Yesilada et al. (2004) investigated the parameters on which linearisation of tables should be based. Their survey revealed that table structure is not the only aspect that determines the style of reading supported by the table. Therefore, the table can not be linearised solely based on its composition. They suggest performing some semantic analysis on the content to optimise the presentation for audio output, before reformatting. As opposed to the approach taken by *tablin*, which requires the user to specify the parameters on which the linearisation is based, they recommend semi automating the selection process. They supply a number of XSL style sheets that can linearise the content in a number of different ways. The user selects the style sheet that best suits the current table. Emacspeak in conjunction with emacs-w3 also provides a number of XSL style sheets to re-organise the content of

⁴⁶<http://www.w3.org/WAI/References/Tablin/>

HTML based pages.

As previously mentioned, interacting with a data table through a linear approach is insufficient. The grid connections between the cells are important in comprehending the content. For this reason, Yesilada et al. also recommend a system allowing the user to interact with the table content by navigating along the spatial relationships between the nodes, as a viable method for table information perception. There are three levels of navigation functionality provided by the EVITA browser (Yesilada et al., 2004), as follows:

Level-1 Low-level navigation functions : move to right, left, below, above, top, bottom, first and last cell, and get current cell. These functions are based on the “current” cell concept which is necessary due to the transient nature of the modality.

Level-2 High-level navigation functions : they possess the following two dimensions: Actions: move next, last, previous and first, and get current. Targets: row and column. Every action can be applied to every target. For example, move to last row will take the user to the bottom row and move to last column will take the user to the last column.

Level-3 High-level tasks : intersection (a cell can be accessed as an intersection of a row and a column) and a comparison can be made. (More than one row or column can be compared).

A number of auditory web browsing solutions offer table navigation in this manner. Many of the functions of the specialised table reading modes contained in IBM’s Home Page Reader, JAWS for Windows and Window Eyes⁴⁷ reflect this level of interaction. All of the lower level functions are implemented by all three applications, whereas implementation of those higher level functions varies from application to application. Home Page

⁴⁷<http://www.gwmicro.com/>

Reader also allows the user to examine the table without removing focus from the current position. That is, if the user requests it, the focus remains on the current cell whilst the rest of the table is explored.

Pontelli et al. (2000, 2002); Pontelli and Son (2002) have worked on producing a domain specific language framework, which could be applied to the navigation of table constructs. Pontelli et al. propose additional navigation functionality in terms of node clustering, based on tracking the most frequented nodes in previous visitations to that document. They also allow for a volunteer to annotate relationships between nodes and their groupings which are not immediately apparent from the HTML syntax. They first attempt to generate these grouping relationships automatically by examining such items as background colours or visual boundaries. This is then presented to the volunteer who can accept or reject the grouping, before assigning a description. Pontelli and Son (2002, 2003) sketched the design of an action theory to support intelligent non-visual navigation of HTML Tables and other non-linear HTML constructs (e.g., frames). They provide the user with the ability not only to interactively navigate, but also to prescribe queries relating to the content and let a software agent perform the navigation on his/her behalf. Filepp et al. (2002) created the text to prose mark-up language (TTPML) which is used to determine how a table should be rendered aurally. This method defines how cells in a grouping relate to one another and the spoken fragments used to link the reading of these elements to increase readability. Additional prose can be included that doesn't appear in the on screen text. Also, short hand notation such as "Fr" for France could be defined to be spoken in full. Although the development of such languages is a promising approach, the designers or third parties have to learn new languages and new engines have to be introduced to process these special languages.

3.6.1 Cells Spanning More than One Row or Column

Another major problem for strategies facilitating the non-visual access of tabular data is the ability to present cells spanning multiple rows and/or columns. In the visual presentation, the relevant positioning of these cells determines the nature of their relationships with neighbouring cells in the grid. However, due to the serial nature of the modality only one cell can be in focus at a time when viewed through speech output technology. Therefore, the nature of these relationships is not easily apparent when using this type of interface. This is not an easy problem to solve. There must be a trade off in the amount of information that needs to be spoken to make the system usable, whilst not cluttering up the interface with additional contextual fragments. For example, repeatedly announcing each cell's position in the grid can often detract from the usability of the user interface. Whereas failing to signal the presence of these cells could potentially result in the user losing track of their position in the navigational space. Filepp et al. (2002) suggests that defining how a table construct is to be spoken using a specialised mark-up language should offset these problems. The user is no longer the one who must determine how elements relate to one another. This relationship is already defined by the content provider. However, this can be problematic if the user wishes to navigate through the different cells to examine their content at a more in-depth level. It also relies on authors/content providers to create these additional descriptions of how the information is to be presented. If such languages are not widely adopted by content providers, other means of interaction will be required. Therefore, there will be many cases where the cell to cell navigation strategy, proposed in Yesilada et al. (2004) and those implemented by current auditory solutions, such as Home Page Reader and JAWS, will still be required to navigate and absorb the data.

The approaches taken by both JAWS for Windows and IBM's Home Page Reader differ somewhat in how they facilitate the inclusion of these elements. When HPR 3.04 reads a spanned cell after an arrow key is pressed

in Table Navigation reading mode, it speaks the contents of the spanned cell and then tells you the number of columns or rows it spans and/or the current column number. For example, if a cell spans three columns, HPR reads the cells' contents and then says, "width 3, column 1". When you move to the previous or next column, HPR 3.04 announces the column number for the spanned cell and does not repeat the cell's contents. Suppose there is a spanned cell consisting of four columns. The content of the cell is "Save \$500". When HPR encounters the first column of the spanned cell, it says "Save \$500, width 4, column 1." Because you have already heard the entire contents of the cell, if you press the Right Arrow, HPR moves to the second column of the spanned cell and says "column 2". If you press Right Arrow again, HPR says "column 3". If you press Right Arrow one more time, HPR says "column 4". You can also choose to have the cells read without this column information, see Home Page Reader: Online help for end users⁴⁸.

JAWS for Windows on the other hand announces when a change in the number of columns in a row has been detected.⁴⁹ As a new row is visited, the application appears to analyse the number of columns present. If the number of columns is the same as in the previous row visited, then the application refrains from announcing such data. However, if a change does occur, information to this effect is announced. If a cell spanning more than one column is encountered whilst navigating from row to row, JAWS remembers the column position. Then, once navigation is moved away from the colspan cell, focus reverts to the previous column if available. Cells spanning more than one row seem to be treated differently. That is, no announcement is made to this effect. In fact the entire content of that cell appears in the first row in which it is present, with subsequent cells left blank. As one of these blank cells is encountered, the correct contextual data for a cell in that row is spoken, without any content. See section 3.6.2 for details on the type of content read. However, it should be noted that very little information

⁴⁸http://www-3.ibm.com/able/solution_offerings/hpr.html

⁴⁹A cell spanning more than one column is only counted as one column, e.g., a cell spanning 4 columns is still counted as one.

illustrating how JAWS handles such constructs is publicly available. These observations were gleaned from personal experience with the product and from anecdotal evidence provided by other jaws users. JAWS version 5.0 was the software used for experimentation purposes.

3.6.2 Providing Contextual Information During Navigation

A major problem with navigating large two dimensional grids of data through audio is the tendency to lose track of the current position in relation to the overall organisation of the material. This is especially the case for large constructs with many rows and columns of data. As mentioned earlier, Raman (1994) proposed the use of spatial audio to reflect the location of each column in the construct. However, this can be problematic for a number of reasons. The first issue is that the user needs to have the right equipment to make this solution viable. Also, in the case of large constructs the differences in spatial positioning in the audio space may not be substantial enough to accurately communicate the column position without a proper reference location. Finally, tabular constructs can differ in width, i.e., there is no set number of columns, therefore, the position in the audio space for a column in one table may be completely inappropriate to represent the same column number in another table. For this reason, the user must learn the different position of each column in the audio space for every table encountered. Pontelli et al. (2002) showed that using non-speech audio cues to signify column cell relationships where spatial cues were unavailable detracted from the interface. They found that users made more errors when such cues were employed than viewing the same information without any cues. Another approach often taken involves the announcement of some or all of the grid coordinates when a cell moves into focus. This is the approach taken by both JAWS for Windows and IBM's Home Page Reader.

JAWS for Windows announces this information after the content of the cell is read. Home Page Reader affords the user control over which piece of

information is read first.⁵⁰ Aside from knowing which column is currently under focus, it is often necessary to understand the context in which this information is intended to be assimilated. The visual reader can usually infer how each cell relates to both column and row header information. However, this relationship is not obvious when listening through speech. For this reason, mechanisms for programmatically assigning headers to table data cells is included in the XHTML and the later HTML specifications. Once these approaches have been implemented, screen reader applications can accurately associate data cells with the correct header information. The next question is how this information should be used to help the user orientate through the construct.

Home Page Reader affords the user the facility to select which headers are read when a cell is encountered. They can choose *none*, *row* header, *column* header or both.⁵¹ Users can also determine whether the header information is presented before or after the cell's contents. With JAWS for Windows, the header information announced is dependent on the direction in which navigation is taking place. If moving across a row, the column header for a cell is read. Similarly, for column navigation it is the row header information that is spoken. In both cases this is uttered before the cells' contents. In cases where there are no header elements, or at least no explicit mark-up assigning headers to data cells, JAWS assumes the first row of cells to be column headers and the first column to be row headers. The spoken output is then generated accordingly.

3.7 Form Handling

Interactive form constructs can pose serious problems for non-visual access to web based documents. According to Hoffman et al. (2005), prop-

⁵⁰Home Page Reader also allows the announcement of grid information to be turned off altogether.

⁵¹There is little published information on how tables are handled by both Home Page Reader and JAWS for Windows. In fact, the available information is limited to what can be gleaned from their respective user manuals. Neither of which mention how the announcement of header relationships in complex tables are to be handled.

erly semantically marked up form fields are well handled by existing screen readers. Once `<label>` elements have been explicitly assigned to individual controls/fields, screen readers can speak this information when a field is encountered. However, problems can occur where this is not the case. Sometimes, the function of a field/control needs to be deciphered from reading the surrounding text. This is not usually a problem when it is encountered during general reading. However, if the user tabs to the field or moves to the next form field through specialised functionality supplied by the screen reading software, then this contextual information is usually unavailable. The solution proposed by Hoffman et al. suggests placing the element containing the information on the tab-index of the document. In this manner the user will have this information available when tabbing through the content. However, there are disadvantages to this approach. If used extensively, the ability to quickly navigate to the individual form fields afforded by these mechanisms may be drastically reduced. The solution employed by JAWS for Windows⁵² is to use information positioned near the form field as the label. The success rate of this approach is dependent on the design of the site. JAWS works best if the relevant data is positioned to the left or above the current field, see *Surfing the Internet with JAWS*⁵³. Figure 3.7 shows a simple form construct as presented through JAWS/IE.⁵⁴ The same `<form>` construct as displayed by Internet Explorer is shown in figure 3.8.

Similarly, forms which use tabular constructs to arrange the data visually, can pose major problems for blind users. The default rendering by many auditory solutions is to linearise the content. See section 3.6 for more details. If the data does not linearise gracefully with row by row linearisation, then blind people may have serious problems associating form fields with the relevant descriptive data. If the `<label>` element is not used to assign explicit relationships between the field/control and the descriptive

⁵²http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

⁵³<http://tinyurl.com/rqzu4>

⁵⁴Only the currently selected item of a “combo box” appears in the audio rendering when accessing content through JAWS. If no item is selected, the first option in the list is displayed.

FirstName:
edit Second Name:
edit student
radio button checked, one of three PAYE worker
radio button not checked, two of three self Employed
radio button not checked, three of three screen Reader User
checkbox checked What is the primary method you use to interact with your
 computer?
combo box A combination of braille and speech four of four
 Reset
 Submit

Figure 3.7: A simple form presented by JAWS/IE

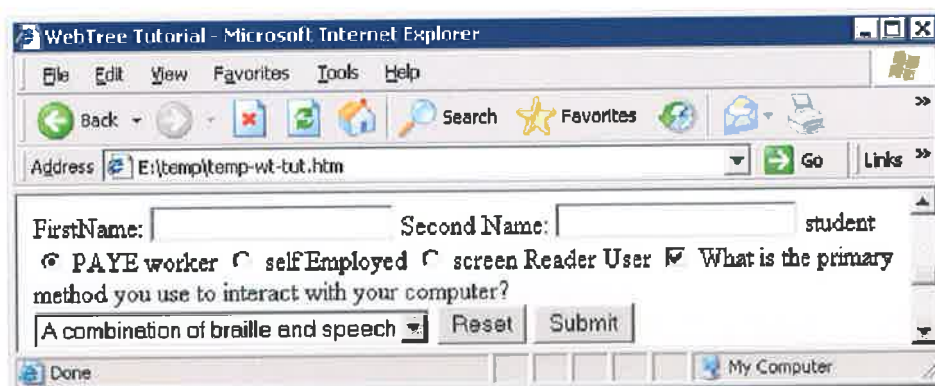


Figure 3.8: A simple form shown through Internet Explorer

```

table Caption: Table 1 Enter course results
Table with 4 columns and 4 rows
Student Number
Exam Grade
Continuous Assessment Grade
Total Grade
001
  Edit0
  Edit0
  Edit0
002
  Edit 0
  Edit 0
  Edit 0
...
  Edit...
  Edit ...
  Edit ...
table end
Reset
Submit

```

Figure 3.9: A tabular form linearised through JAWS/IE

text, then the blind user must rely on an element of guess work to establish the functionality of the different form fields. Figure 3.9 shows a form containing a tabular construct linearised through JAWS. Figure 3.10 shows the same form construct as it might appear through Internet Explorer. If there are many form elements, the user can become disorientated when filling out the content. However, the situation becomes more problematic where blank cells are used to organise the data visually. This requires a greater memory load on behalf of the user to establish relationships between the cells. Thus, rendering this form of interaction more difficult. Although table reading commands can often help in this situation, they do add an additional level of complexity to the problem.

There are situations where the use of a data table to semantically organise the content of form constructs makes perfect sense. Consider a table

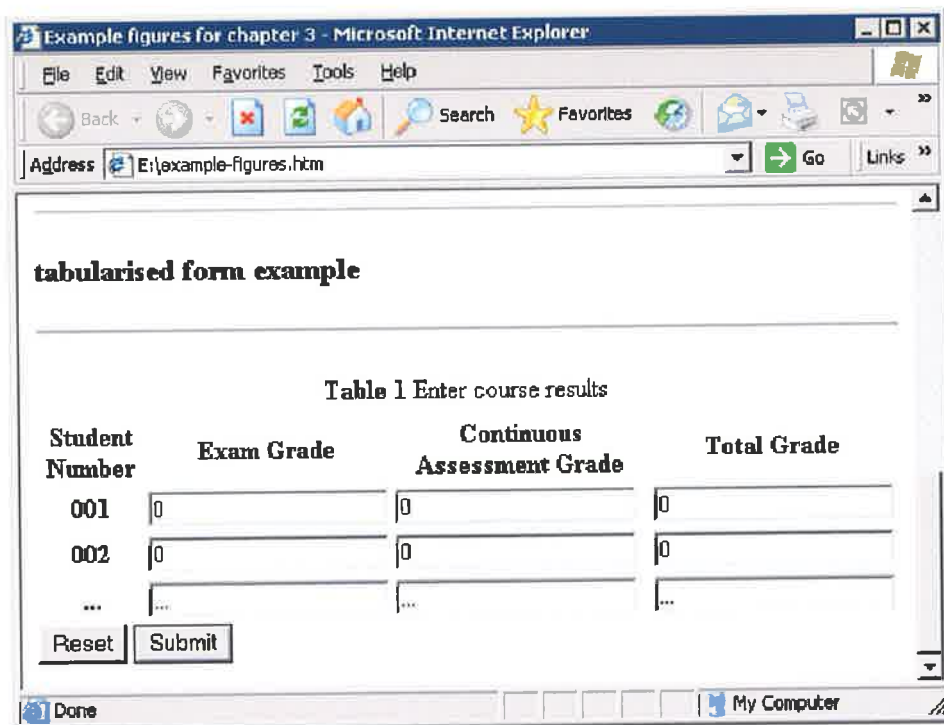


Figure 3.10: A tabular form construct

where the “data” cells contain only form controls; and the correct data to be entered in each case is indicated by the combination of the column and row header for the particular cell. Requiring a `<label>` element for each form field/control would be serious overkill.⁵⁵ Therefore, in an ideal world, it would be logical to have the row/column information act as the contextual text in this case instead.⁵⁶ Jim Thatcher has suggested that using table header mark up instead of `<label>` is perfectly logical and reasonable, but does not “work” with widely deployed screen-readers such as JAWS.⁵⁷ The reason for this may be that the table reading functions in JAWS are incompatible with its specialised forms mode.

Also controls may be grouped in terms of their function, e.g., “first”, “middle” and “last” name. A shared label might be used to group these elements. When tabbing to these fields the announced label may not be enough to clarify the function without the context in which it appears. Adding in full labels for each element might detract from the visual interface. For this reason, Hoffman et al. suggest using additional labels (hidden from view by CSS, but available to screen reader applications). A second solution would be to attach a “title” attribute to the relevant field/control to convey the true nature of the element.

Many auditory web solutions use a special reading mode to read web based documents, e.g., JAWS for Windows and IBM’s Home Page Reader⁵⁸. In this specialised mode, all the navigation keystrokes are available. In JAWS for Windows, form items are presented on separate lines from other content to allow for easy selection and interaction with the item. However, to fill out form fields, the keystrokes must revert to their character input status. For this reason the user must change the mode of interaction to a special forms mode to allow for data entry. Once the content for a given field is filled in, the user can either revert to the reading mode, or move

⁵⁵ `<label>` tags can have only a one to one association with a single control.

⁵⁶ Table headers can be linked to multiple cells.

⁵⁷ <http://lists.w3.org/Archives/Public/w3c-wai-ig/2005AprJun/0014.html>

⁵⁸ http://www-3.ibm.com/able/solution_offerings/hpr.html

on to the next field with special commands often provided. WebFormator⁵⁹ requires the user to switch back to the main IE window to fill out such data. Once the task has been completed, the user can switch focus back to the WebFormator text window. In the case of checkboxes and radio buttons, pressing the selection key when in the specialised reading mode will often suffice to alter the state of the element. That is, once the relevant control is in focus.

Select lists are often displayed in an unexpanded state by default. When encountered, the element type is often announced in conjunction with the first option in the list. To select a given element the user must enter a special mode that expands the list so the user can navigate through the options and choose the one they require. Once the user leaves this mode, the list reverts to its collapsed state, with the selected item displayed in the audio rendering.

Emacspeak in conjunction with the emacs-w3 browser does not require the user to explicitly enter a specialised forms interaction mode for single line text areas. Instead, the mapping for the individual keystrokes reverts to their character input functionality, once focus has been positioned over a text entry field. For multi line text fields e.g., “text areas” a new buffer is opened to take the required content. Once the user has finished entering the content, they can revert to the main document view. The entered data is then presented in the field as required.

Finally, many solutions offer additional short cut commands to aid the navigation of form constructs. For example, in JAWS for Windows, the user can select a view limited to form fields/controls. Many applications offer the ability to quickly jump to the next /previous form field, e.g., JAWS, WebFormator and HPR. However, sometimes the user can also jump to the element of a given type such as the next button or select list, e.g., JAWS for Windows.

⁵⁹<http://www.webformator.com/englisch/index.php>

3.8 Tree Navigation

The final section of this review is dedicated to the mechanisms employed to convey tree information in a non-visual modality. Speaking the content of each item is not where the problem lies. Instead, the issue is concerned with how to unobtrusively convey level information to the listener. Without this data, the listener may have difficulty relating elements to the relevant segment of the tree. This can result in the user becoming disorientated when navigating complex tree hierarchies. As discussed in section 3.2 this can be achieved in a number of ways. The first method involves the insertion of additional speech fragments to denote level information. This method is used by JAWS for Windows⁶⁰ to present list information viewed through Internet Explorer. However it is also used when navigating through the directory tree view in windows explorer. As a change in level takes place, the new level information is announced in the audio stream. In the case of windows explorer, this data includes the tree level and the number of items present at the current hierarchical position. In terms of lists in web based documents, users are notified when entering a nested list and again when leaving this construct. See section 3.2 for an example page containing a list as displayed through JAWS.

The second method by which this can be achieved is through the use of non-speech audio cues. In section 2.2 the use of non-speech sound cues to present hierarchical information was discussed. Brewster (1998) showed how interface hierarchies can be signalled in this manner.

Finally, the third approach to be discussed requires some manipulation of voices and voice characteristics to convey this contextual data (Raman, 1994; Shajahan and Irani, 2005). To represent a change in the level of a sub tree, Raman suggests manipulating dimensions of voice characteristics to create a softer more animated voice. As deeper levels of nesting are entered, the change in voice characteristic produces a sense of falling off into the distance.

⁶⁰http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

Shajahan and Irani experimented with manipulating synthetic voice parameters to convey hierarchical information. In a similar fashion to the experiments with hierarchical earcons performed by Brewster (1998), new voices inherited many characteristics from the parent node. To represent changes in the different tree levels, they used duplication, variation and inclusion of different voice parameters to denote changes in context. Inclusion was defined to include one or more speech parameters to the preceding voice in order to create unique voices. For example, if the child node duplicates all the parameters and the exact values from its parent node (such as speech rate and pitch), then other voice parameters such as *laryngealisation* and *breathiness* can be added (inclusion rule) to the child node, to make the child node sound different from its parent. Similar to the experiments performed by Brewster, Shajahan and Irani (2005) were successful in demonstrating that once the rules concerning the creation of the different voices were understood by the test participants, they were able to recognise and classify voices not previously heard. The results of their experiments show that the manipulation of voice parameters to denote hierarchical information is a viable approach. However, further work is necessary to determine the optimum method in how these voice changes should be used. As part of future work, they suggest designing a study to evaluate the effectiveness of multiple parameter configurations for representing depth and width in hierarchies.

Smith et al. (2003) developed a non-visual tree based navigation interface to the viewing of source code hierarchies. They believe that the approach taken could also be applied to the viewing of other tree based structures such as file systems. The results of these experiments shows that an accessible tree navigation tool provides an effective and efficient strategy for non-visual navigation within hierarchical structures. Working on the basis that the entire tree is fully expanded in view, they derived navigation functions to traverse the construct. Moving left brought the user to the parent element. Navigating up and down visited the different siblings, whilst moving right found the children of the current node. The sibling navigation was arranged

in a circular list. That is, moving up from the first node at a given level would reposition focus on the last node at the same level. Users were able to develop an accurate mental model of the tree and could therefore navigate successfully within both familiar and unfamiliar trees of various sizes. The tool also provides the user with a method for gleaning context information. The "context" features let the user know the current location in the tree, relative to the tree's overall structure. These features include giving the user location information about distance from the root, who the cousins are, and what the density of the sub tree is, as well as providing a means for the user to change the context relative to a new anchor point. The strategy also includes "focus" features that let the user get detailed information about a specific node, including the parent, the number of children and siblings, and the past history of visiting the node.

Chimera and Shneiderman (1994) found that both expand/contract and multi pane interfaces produced significantly faster times than the stable interface for many tasks using a large hierarchical structure, such as a table of contents. The major drawback to a stable interface is that users have difficulty in perceiving the global hierarchical breakdown of the text. Because section and subsection titles are permanently displayed inline, many screens/pages would have to be scrolled to show enough chapter titles to comprehend the major divisions of the text. The expand/contract and multi pane interfaces attempt to overcome the hierarchy breakdown problem by displaying the high-level information contiguously and giving users the choice of viewing specific section and subsection levels on demand. Viewing the document through the stable interface, users must perform a considerable amount of scrolling and can get lost in a large document. The resultant disorientation can be caused by conceptual confusion concerning the logical structure. Whereas, the expand/contract interface preserves more of the logical structure and context than the stable interface since the chapter titles preceding the one chosen for expansion often are visible on the screen. The multipane interface overcomes the conceptual deficiency by constantly

displaying the high-level chapter and section information. Although the experiments performed by Chimera and Shneiderman involved sighted users, it is logical to assume similar problematic issues when viewing these structures non-visually. In fact, the problems might be exaggerated by the limitations on the amount of information viewable at any given time. Although Chimera and Shneiderman favoured the multi pane approach to viewing these constructs, this might be problematic for blind users. The information in one area of the screen alters depending on interactions in another screen area. This means the user must switch to each screen area to analyse the changes and then revert to the previous position to continue navigation. Also, if the hierarchy is quite deep, it may be impractical to display a pane for each level in the display. For these reasons, the expand-contract mechanism should be more appropriate to blind users. As will be evident in the following chapters, WebTree applies a similar approach to the viewing of structurally marked up web based documents.

3.9 Summary

In this chapter an outline of the research to date in relation to making electronic information accessible to blind users was provided. A discussion of the literature in the area of accessibility was presented, coupled with a description on how current auditory solutions render web based documents. In the next chapter, the implementation environment for WebTree is discussed. The methods used for modelling the document are shown, in addition to the methods used to link content in the audio rendering to WebTree's internal model. Also, WebTree's dependencies on third party software are described. The limitations of the system when faced by alternative media types such as flash content are also presented. In section 4.7 the discussion focuses on the web accessibility guidelines on which WebTree depends. Finally, a discussion concerning legislation in the area of web accessibility is presented.

Chapter 4

WebTree Implementation

In the previous two chapters, the different technologies employed by blind people to access electronic information were discussed. A review of the literature concerning how these technologies have so far been used was also included. Thus, much of the remainder of this dissertation is dedicated to a discussion of the WebTree system. In this chapter a number of implementation issues encountered when developing WebTree are described. The first section provides a brief overview of emacs and some emacs terminology. Following this, a description of the implementation environment under which WebTree operates is included. Next, the methods used to model the content and provide access to the tree components are discussed. The final section presents an overview of the standards and guidelines on which WebTree is dependent. This presentation also includes an outline of legislation in the area of accessibility and how it applies to the Web. The relevant legislation is discussed from both an Irish and International stand point.

4.1 Emacs Terminology

To quote the Emacs Manual:

Emacs is the extensible, customizable, self-documenting real-time display editor.

or in simple terms, emacs is a text editor which can be extended to offer much more powerful functionality. A robust customisation facility is provided to allow the user modify settings to better reflect their requirements. However, if this is not sufficient, emacs provides its own interpretive scripting language called `emacs lisp` or `ELisp` so that the application can be easily extended. Much of emacs is written in `ELisp`, therefore, if a specific function does not fulfil a users requirements, they are free to write their own function or modify the existing method for use instead. Many of the different packages supported by emacs are also written in `ELisp`. These range from applications to brows the World Wide Web, e.g., `emacs-w3`¹, to electronic mail programs, such as `vm`², to specialised editing modes for specific programming languages. The different components of emacs have been created in a modular architecture. Thus, the commands available and their keystrokes are dependent on the major/minor mode currently in operation. As `WebTree` has been developed as a major mode under emacs, the rest of this section will explain a number of emacs specific terms that are used throughout the rest of this chapter, and also in chapters 5 and 6.

Buffer : a `buffer` is simply the container for whatever you happen to be currently editing. Rather than making changes directly to a file, you make changes to a `buffer` that holds the contents of that file. The file itself is not changed until you save the `buffer`.

Point : In terms of emacs/emacspeak, the term `point` means the current position in the document. That is, the position in the text of the edit curser.

Mark : Emacs allows the user to save their current position in a buffer by setting a `mark` for that buffer. This is done by storing the position under `point` for later use. A mark is local to a given buffer, therefore, it only affects the one in which it is saved.

¹<http://www.cs.indiana.edu/elisp/w3/docs.html>

²<http://www.wonderworks.com/vm/>

Region : A **region** is an arbitrary chunk of text under emacs. It is defined as the text between **mark** and **point**. Many commands operate on regions, such as those for deleting text.

Widget : In terms of emacs, a **widget** is a component of the interface which the user can interact with. These include, “edit-boxes”, “push-buttons” and “text items”. The default widget classes can be found in `wid-edit.el` in the emacs distribution. By setting the different properties of these objects, the developer can determine the look and feel of the component, and any actions to be taken when the user interacts with the object. For more information on widgets supported by emacs, see the documentation for the The Emacs Widget Library³ package.

Mode : a **mode** defines the way emacs treats the contents of the buffer. Emacs can use both major and minor modes at any given time. Only one major mode may be active in a specific buffer, but several minor modes may be active. A major mode determines the fundamental way emacs should look at the buffer contents. If it’s **text** mode, emacs should treat the contents as plain text. However, if you switch the buffer to **perl** mode, emacs will treat the same contents now as a perl program, and will enforce indentation rules and enable several commands to make entering perl code easier. Minor modes will affect the buffer contents in less fundamental ways, and may be used with many (or all) major modes. In the above example, you may also have **font-lock-mode** enabled, which will make your perl code stand out in different colours for comments, variables, commands, etc.

4.2 WebTree Specification and Design

The first step in developing the WebTree system was to create a high level specification detailing the types of functionality an application of its type

³<http://www.dina.kvl.dk/~abraham/custom/widget.html>

should provide. Establishing the data on which these requirements were to be derived was achieved using two distinct methods. The first required an examination of the problems that many blind users face when browsing/navigating web based documents. Much of this data was obtained through personal experience in accessing electronic material through auditory means and by anecdotal evidence gleaned from discussions with other blind computer users. The second method involved an examination of current solutions to the problem and attempting to establish ways in which these could be improved. It was decided from the outset that the focus of the work would centre on offsetting problems with accessing content caused by the serial nature of speech, as opposed to solving problems emanating from badly composed mark-up and websites not adhering to accessibility standards.

In sections 4.3 and 4.4 the implementation environment and the methods used to model the document for internal manipulation by the application are discussed. Also included in this discussion are many of the design decisions that were made during the creation of the WebTree prototype. In addition, any third party software used as part of WebTree is credited in these sections. However, before describing the details of the prototype implementation, the software engineering approaches used in bringing the system from requirements stage to its current implementation state need to be discussed.

The system design methodology pursued during the creation of the WebTree prototype followed an *agile* approach to software development. The “agile” development approach is a conceptual framework for undertaking software engineering projects. Most agile methods attempt to minimize risk by developing software in short timeboxes, called iterations. Each iteration is like a miniature software project of its own, and includes all of the tasks necessary to release the mini-increment of new functionality: planning, requirements analysis, design, coding, testing, and documentation. While an iteration may not add enough functionality to warrant releasing the product, an agile software project intends to be capable of releasing new software

at the end of every iteration. At the end of each iteration, the project priorities are re-evaluated. Agile methods produce completely developed and tested features (but a very small subset of the whole) every few weeks or months. The emphasis is on obtaining a crude but executable system early, and continually improving it.

The specific “agile” approach pursued in WebTree’s development was the “adaptive software development” (ASD) framework (by Jim Highsmith, 1997). ASD functions on the principle that continuously adapting the development process to the current work is quite a normal feature of the development process. It consists of a repeating series of speculate, collaborate, and learn cycles. This dynamic cycle allows the developers to continuously adapt to changing states of the project based on what they have learned so far. In this manner a working system emerges from the process. The characteristics of an ASD life cycle are that it is mission focused, feature based, iterative, timeboxed, risk driven, and tolerant to change.

As the ASD development cycle is iterative in nature it lends itself well to a form of rapid application development (RAD). Traditional RAD is effective for the evaluation of differing user interfaces (UI), allowing for the selection of the most appropriate UI for each stage of the implementation. ASD also provides some latitude to the developer to solve unanticipated problems as they are encountered. This is achieved through “software spiking”. Software spiking refers to the developer’s ability to break from the current iteration of the implementation, solve an unforeseen problem and then return to the previous state of development. In terms of ASD, the word “speculate” is used instead of “plan” so that any preconceived ideas of the project are not entirely rigid and may be subject to change. These alterations often occur due to changes in available technology and through data learned from problems encountered.

The code base for the WebTree system contains a mixture of object oriented and procedural code. The methods used for content modelling, for example, the “document object model” or DOM, which is used to internally

store the data, and the “widget-tree model” (see section 4.5.1), used to render the content, follow the object oriented design paradigm. However, much of the code is procedural in nature. This includes the methods dictating how the rendered elements interact with one another, and those which govern how/when they are to be rendered. The implementation environment (see section 4.3) did not always lend itself well to the object oriented approach. The language in which the system is implemented is a functional language. Functional languages are geared more towards iterative programming as opposed to the object oriented method. Also much of the third party code on which WebTree depends, although modular, is procedurally based.

The first step in the implementation stage of the project was to integrate the “xml” parser and the methods for modelling the document (DOM, see section 4.4) into the system. Once content could be successfully parsed and stored in the “DOM”, work on creating the dynamically expandable tree rendering functionality commenced. After this objective had been completed, code to integrate the CSS parser and apply CSS “properties” to the content was included. The “url” retrieval functionality was then added to the system. This was followed by code to facilitate elements requiring additional functionality to operate effectively, e.g., hyperlinks, form constructs and tabular data, which were added in that order. Finally, the specialised “search” features were built into the application.

Software documentation was completed as each feature of WebTree was developed. The implementation environment allows for a text string providing information about each function to be entered into the source code. This should allow the application to be easily maintained, for developers can see how the system operates on a function by function basis. However, to simplify matters even further, documentation describing the overall design architecture and how the specific functions interact with one another was also completed.

4.3 Implementation Environment

When developing the WebTree system, it was necessary to determine whether it would be better to create a browsing solution that operated on the client side, or use a server side intermediary proxy application. It may have been possible for the proxy system to generate the dynamically expandable tree rendering from intercepted web pages and passed them to a mainstream browsing application. A Java applet or J2EE traversing proxy solution in conjunction with the user's screen reading application were mooted as possible solutions. However, at the time this project commenced, access to Java applets was not well supported by any of the popular screen reading applications. As for the intermediary proxy solution, it was thought that the additional load on the proxy that expanding/collapsing elements through requests to the server would have had a detrimental effect on the speed and hence the usability of the application. If only a small segment of the page was requested at any given time, much communication between the client browser and the server to coordinate the delivery of the material would be necessary. However, if an entire page was passed by the server with each request, aside from the time delay associated with downloading the newly rendered content, the screen reader application may not leave the user in the same position as before the expansion/collapse request. Performing expansion/collapsing of elements using JavaScript was not a viable alternative either. This is due to the lack in comprehensive support for such technology in screen readers. For further information on the support of JavaScript in current screen reading applications, see the article: *AJAX and Screen readers: When Can it Work?* By James Edwards⁴. For these reasons it was decided to build WebTree as a client side application.

Once it was decided to create WebTree as a client side solution, there were a number of possible ways in which the application could have been developed. The first method is to create a self voicing browser to handle the user's entire experience with the content. In this case, the text to speech

⁴<http://www.sitepoint.com/article/ajax-screenreaders-work>

(TTS) engine is directly controlled by the application. a second approach, as used in the development of WebTree, consists of a browsing application to render the content and allowing the user's own screen reading software to handle the production of spoken output.⁵ There are advantages in providing information through the user's regular assistive technology, as opposed to controlling the text to speech engine directly. For one, it ensures that the risk of clashes in output between the two applications is prevented. This problem can occur when both applications attempt to read aloud the information at the same time. To avoid this, the screen reader must be either turned off or prevented from speaking during the time when the self-voicing browser is in focus. However, when the focus is changed to another application, the screen reader needs to resume speaking⁶. For example, when using IBM's Home Page Reader⁷ in conjunction with JAWS for Windows⁸ as the main screen reader, the user must set up a *sleep* mode for JAWS to stop it reading in HPR. Braille output can remain on so that both a Braille display and speech can be used at the same time. A second advantage is that the content can be provided to the listener using their preferred regular voice characteristics, such as reading speed or voice type. This means that they do not have to create such settings on a per application basis.

One of the major criteria governing the selection of the implementation environment was the requirement to create a low cost solution using "open source" technology, or in some cases "shareware" software. Under the Microsoft WINDOWS⁹ operating system, the major English speaking screen reading applications e.g., Jaws for windows and GW Micro's¹⁰ "window eyes", are proprietary software, hence the source code is not freely available. Although custom control for an application is offered through special script-

⁵Often some customisation of the screen reader is necessary before the application becomes usable and the speech output can be optimised.

⁶Home Page Reader provides a limited form of screen reading functionality known as the Desktop Reader, which can read the windows desktop, plus a number of applications such as WordPad and Adobe Reader 6.0

⁷<http://www-3.ibm.com/able/solution.offerings/hpr.html>

⁸http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

⁹<http://www.microsoft.com>

¹⁰<http://www.gwmicro.com/>

ing languages under these products, the monetary cost of purchasing this software mitigated against their selection. Therefore, focus was switched to the Linux operating system. At the time that development of WebTree commenced, the accessibility work in the GNOME: The Free Software Desktop project¹¹ was still at its infancy. Also, screen reading applications such as Gnopernicus¹² were not available. Therefore, focus shifted from the provision of access through the “GUI” to access through the Linux “text” mode. Although there were a number of auditory solutions available for this environment, many of these were console based and did not furnish a lot of control over the final output. For example, Speakup¹³, which is compiled as a patch into the Linux kernel. At the time that development of WebTree began, only hardware synthesisers were supported by Speakup. `emacspeak`¹⁴ on the other hand is a rather powerful, highly customisable auditory access solution. A major advantage of this system is its extensibility. Due to its modular nature, custom functionality to handle a specific application can be easily incorporated without adversely affecting other programs. Therefore, the WebTree application has been written in ELisp and runs under the GNU emacs-21.3¹⁵ lisp environment on the Linux platform. It is not a stand-alone self-voicing web browser; rather, it relies on hooking into functionality incorporated into `emacspeak-19.0` to produce spoken output.

There is one major disadvantage to be found with the choice of implementation environment. That is, the usage levels of the emacs and `emacspeak` combination as an auditory computer access solution is quite low. This problem did manifest itself when trying to source blind computer users to test the WebTree application.

WebTree could potentially have been written in another language e.g., c/c++. However, to operate under the emacs/`emacspeak` environment, it would have been necessary to write and maintain an additional layer of ELisp

¹¹<http://www.gnome.org>

¹²<http://www.baum.ro/gnopernicus.html>

¹³<http://www.linux-speakup.org/speakup.html>

¹⁴<http://emacspeak.sourceforge.net/>

¹⁵<http://www.gnu.org/software/emacs/emacs.html>

code to handle the interaction between WebTree and emacs/emacspeak. Therefore, it made more sense to write the entire application in the “ELisp” language.

The initial plan was to just implement an alternative interface to the emacs-w3 browser. This would have meant that WebTree would have contained the full capabilities of a working browsing application and the project could just concentrate on designing a usable user interface. However, this was found to be quite problematic. There was very little documentation on major components of emacs-w3. This was especially true in the case of its internal document model and display code. Some information was contained in the “documentation” string for most functions, however there was little information on the overall architecture of the system. Therefore, it was decided to use certain components of emacs-w3 and add additional technologies where necessary.

As already mentioned, WebTree is dependent on a number of components of the emacs-w3¹⁶ web browser to function. Some of the code is directly used by the system without any modifications. For example, the `url` library is used to handle the downloading of documents from the web. Also, the CSS parser included in W3 is used by WebTree to parse Cascading Style Sheets. However, a number of segments of the code base were adapted for use in the system. The form handling code was used as the basis for WebTree’s form handling functionality. Major changes were necessary to produce form widgets compatible with the *widget tree* approach (discussed in more detail below). In addition, it was necessary to alter this code to use the DOM structure as input data, as opposed to w3’s internal representation. As described in section 4.5.1, code to handle the collapsing and expansion of elements was also adapted from third party software. The emacs specific “Isearch” mode was modified to invoke the WebTree specific specialised search functions, once in the WebTree system. This was done so that users could seamlessly use the search methods as they would in any document.

¹⁶<http://www.cs.indiana.edu/elisp/w3/docs.html>

However, bar a few minor lower level display functions adapted from emacs-w3, the rest of the code base for WebTree was written from scratch.

Speech output for the WebTree system is processed by the “emacspeak” application. However, it was necessary to modify some of the “emacspeak” code to optimally render the content of documents presented by WebTree through spoken output. For example, the emacspeak specific `emacspeak-speak-line` function was modified to announce the presence of form fields when encountered whilst reading a document line by line. In addition, when a form element is encountered by other means, the emacspeak specific functions had to be altered to be able to obtain any relevant information about the element from the internal storage mechanism. For example, they need to check if any `<label>` tags are associated with the element and retrieve the subsequent information. Apart from these changes the emacspeak code base was not otherwise altered. A number of WebTree functions do also hook directly into the emacspeak speech output functions, as opposed to using the generalised `emacspeak-speak-line` function. In general, these commands provide some contextual information about the content, for example, the name of the localised element under point, or information concerning table cell coordinates. Due to these direct calls to “emacspeak” speech commands, it is necessary to have emacspeak running when using WebTree.

The system was designed as a general-purpose XML (W3C, 2004*b*) browser. Special emphasis was placed on supporting XHTML (W3C, 2002*b*) web pages. Although the browser is capable of rendering generic XML documents, the current incarnation has been optimised to support documents validating to the XHTML basic standard (W3C, 2000*b*). For the purposes of the prototype application, it was assumed that all documents were written in valid well-formed XHTML basic mark-up. It was also understood that these documents would be marked up in a manner that would meet level Double-A conformance of the *web content accessibility guidelines* (WCAG) version 1.0 (W3C, 1999*b*). It might be argued that in applying these con-

straints, a world far different than the one we live in has been constructed. One journal reviewer has commented:

Their solution depends on pages that validate successfully against XHTML. Indeed, at least one of their tasks involved a translated web site created just for their evaluative experiments. In effect, they have constructed a world far different than the one we live in with pages composed of bits of Flash or recondite Ajax structures. Their experiments remind one of the physics problems that begin, *In a world without friction...* They said nothing unsound, counterfactual, or inaccurate, but they explored a world that, while we would like it to exist, simply doesn't.

Although this research uses web based documents as a basis, the primary focus of the work is to try to increase the usability of document browsing/navigation by offsetting problems imposed by the serial nature of speech. As with any screen reading application, best results are obtained when a web page is marked up in accordance with accessibility guidelines. The less accessible the web page is, the greater the difficulty in browsing its content. Admittedly the numbers of web sites currently meeting WCAG Double-A conformance are still rather low (McMullin, 2002a; Marincu and McMullin, 2004; UK Cabinet Office, 2005). However, there is good reason to believe that this number will increase substantially in the future. This is due in part to the greater awareness of accessibility issues among website developers and also to the recent increase in legislation governing website accessibility. For example, section 508¹⁷ of the Rehabilitation Act in the U.S., makes explicit reference to web accessibility requirements. The draft code of practice¹⁸ under the Irish disability act 2005¹⁹ proposes that all public sector websites should conform to WCAG 1.0 Level Double-A. There also exists some international case law supporting the inclusion of web site

¹⁷<http://www.section508.gov/>

¹⁸<http://tinyurl.com/o3rdp>

¹⁹<http://www.oireachtas.ie/viewdoc.asp?DocID=4338>

accessibility features. See the Reader's Guide to Sydney Olympics Accessibility Complaint²⁰ for more details. It should be noted that not all the guidelines required for WCAG level Double-A conformance are relevant to web access for blind individuals. See section 4.7.3 for more details on the applicable WCAG 1.0 guidelines and a description of the current state of web accessibility, both from an implementation and a legislative point of view.

There is a reason for the second restriction limiting support to XML based documents. The main focus of the project is to design an alternative non-visual web-browsing interface. Therefore, for the sake of the prototype application, it was decided not to centre resources on the development or implementation of heuristic algorithms to solve issues with invalid mark-up, or to decipher the implied meaning in inaccessible web pages. Instead the focus should be concentrated on user interface design issues. However, in saying this, it is believed that the proposed viewing methods would be useful for viewing other HTML document type derivatives. Including support for other HTML DTDs is not a difficult task. Due to the DOM (Document Object Model) being the chosen mechanism for internal document storage, all that is required is additional lisp code to parse such documents and generate a representation in the required format. See section 4.4 for more details.

The prototype system does not currently support additional embedded technologies such as Adobe Flash²¹ content, Java applets²² or JavaScript²³. In many cases the types of presentation produced using these technologies depend greatly on visual interaction. Although Flash content can support accessibility features (Regan, 2005), care must be taken when creating such material.²⁴ For example, when a page is updated and the content has

²⁰<http://www.contenu.nu/socog-PR.html>

²¹<http://www.adobe.com/products/flash/flashpro/>

²²<http://java.sun.com/applets/>

²³<http://www.ecma-international.org/publications/standards/Ecma-262.htm>

²⁴Flash accessibility support is currently implemented only on the windows platform, for it depends on MSAA (Sinclair, 2000) to provide content to a user's specific assistive technology.

changed, many screen reading applications revert to the top of the page and commence reading. This can make it difficult to interact with the content, especially if the material is updated on a regular basis. They offer some practical advice on how to ensure such content is both accessible and useable. Regan (2005) tells us that there are times when flash accessibility is not possible, or at least extremely problematic.

There are some complex forms of Flash content that simply can not be made accessible. For example, many simulations require users to attend to several objects at the same time. Decisions must be made based on multiple factors and relayed back to the simulation quickly. This type of multitasking activity may be easy to do in the real world for someone who is blind, but can pose a real challenge while using a screen reader.

Similar problems can occur when JavaScript is employed to selectively hide certain elements of information. As the user interacts with the script, the content on the page might be updated in accordance with the selection. This can mean that the focus is moved away from the current position. Otherwise, the user may not be aware of the changes due to the narrow focus on the information through the speech output modality. For a detailed description of the problems faced by screen reader users, see the article: AJAX and Screen readers: When Can it Work? By James Edwards²⁵. Therefore, to successfully implement such technologies into WebTree in a useable manner is not a trivial task. WebTree operates under emacs on a Linux platform, whilst Flash technology requires Microsoft windows based MSAA (Sinclair, 2000) to provide information to a screen reader. Therefore, providing support for this content is not currently practical under the current implementation environment. To support JavaScript, a scripting engine must be found/written and incorporated into the system, for emacs-w3 does not support JavaScript. At the moment the system ignores such content.

²⁵<http://www.sitepoint.com/article/ajax-screenreaders-work>

However, as part of future work the best methods concerning how some or all of these technologies could function under the tree viewing approach should be investigated.

4.4 Document Modelling

As implementing an alternative interface to the emacs-w3 browser was not an option, a new method for internally modelling the data was sought. The two major options considered were the “Document Object Model” (DOM) and the “Simple API for XML” (SAX). The DOM is a Tree-based API, which means that an XML document is mapped into an internal tree structure, and an application is then allowed to navigate that tree to find and process the data. SAX on the other hand is an Event-based API. It reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree structure. The application implements handlers to deal with the different events, much like handling events in a graphical user interface. For more information on SAX, see the official SAX²⁶ website. The DOM method was chosen for it lends itself well to the dynamic tree expansion model proposed for the primary document rendering through WebTree. Storing content in a tree-like structure enables simple mapping from elements in the internal document model to what is displayed in the audio rendering. Although it is possible to construct a parse tree using an event-based API, and use an event-based API to traverse an internal memory tree, this would have required much more development work in the implementation of the WebTree system prototype.

For the most part, the internal model used for document storage follows a strict traditional tree approach. At the root of the tree is the `<html>` element. The branches of the tree contain the subordinate elements, with the leaves of the tree containing the textual content. It differs from the model proposed by Fitzpatrick (1999) in the sense that it does not contain links to elements at the same tree level in other sub trees. The only time

²⁶<http://sax.sourceforge.net/>

that the system departs from this model is in the presentation of tabular information. Links are created between `<td>` or `<th>` elements in the same column across `<tr>` elements. However, these links are not reflected in the internal storage model; instead they are reflected in the widget tree for the audio rendering. See section 4.5.1 for more details.

As previously mentioned, the DOM lends itself well to facilitating the types of functionality offered by WebTree. The DOM's hierarchical structure is based on having nodes to represent each mark-up element in the document. Nodes representing the element's attributes are linked directly from the element node. Due to the proposed rendering approach's reliance on the tree-like composition of the document mark-up, a structure reflecting this type of organisation was required. Because of this method of organisation, it is simple to create a direct mapping from the internal representation to the elements exposed in the auditory rendering. See section 4.5 for details on how this was achieved in the system. The aim is to expose only a limited number of document tree elements in the display at any given time. The number of elements, and the type of elements to be included, are determined first by user preferences and then through the dynamic expansion of individual screen elements. See section 5.1.1 for more details on the structure of the primary document rendering.

When expanding/collapsing the content of a given element, the DOM node serves as an easy access point into the document's hierarchical structure. An element's `child`, `parent` and `attribute` nodes are accessible from this position. Exposing the tree structure in this manner facilitates the expansion or contraction of content. The interface can be manipulated to include as much of the DOM tree as required. Similarly, it provides a simple methodology for hiding entire sub trees of XHTML elements from the document rendering.

In addition, using the DOM structure aids the implementation of the proposed alternative types of search functionality. That is, searching for a given element, or limiting the text search to be inside a specific XHTML

element. These specialised search facilities are discussed in more detail in section 5.1.5.

There were two options when implementing the DOM component of the WebTree system. These included developing the DOM component from scratch, or integrating an existing library into the project. The latter option was chosen in this case. The DOM implementation used was written in emacs lisp, however, it is not included as part of the GNU emacs distribution.²⁷ This component does not handle the initial loading or parsing of documents into the DOM data structure. Therefore an additional parser that produced an output in a format that could be accepted and recognised by the functions in `dom.el` needed to be found. The parser selected can be found in `xml.el` in the GNU emacs distribution. To add support for other document types would require a parser that produces a DOM data structure similar to `xml.el`.

4.5 Exposing Document Tree Structure

The next major issue is how the information stored in each DOM node should be presented in the audio rendering. It was clear that just including the textual content of each element would not be sufficient to facilitate the tree viewing approach. Therefore, it was necessary to provide a method to connect text in the audio rendering to its equivalent node position in the DOM tree structure. The expand/collapse element methods depend on this, for they require a starting point in the DOM tree to function. It is also necessary for the application to be able to verify the type of element under point. This is important in the case of a *search by element* or a *restricted incremental search*. See section 5.1.5 for more details on the types of search available. If an element meeting the correct search criteria is found in the DOM structure, the application must then search the exposed elements to find information from its ancestor tree nodes so the newly found item can

²⁷`dom.el`: DOM implementation copyright © 2001 Alex Schroede, maintained by Henrik Motakef: <http://www.emacswiki.org/cgi-bin/wiki.pl?XmlParser>

be expanded. An additional reason is so that the user can programmatically obtain tree positional information about the element under `point`.

It was also important to find a mechanism in which element trees can be dynamically included/removed from the audio rendering by the user in a controlled fashion. This could have been achieved in a number of different ways:

- The first method involves using emacs specific screen **widgets** to draw the text content of an element to the screen. To solve the problem of linking text in the rendering to its position in the DOM tree, a pointer to the corresponding DOM node could be linked to one of the properties of the displayed widget. Aside from this, widgets allow for the easy expansion and contraction of screen elements. The tree-widget found in `tree-widget.el`²⁸ Allows for a generic method to alter the number of elements exposed in the display at any given time. This is achieved through the expansion and contraction of tree widgets easily controlled by the user.
- The second method involves inserting the content contained in each DOM node directly into the buffer text. Associating content with its equivalent DOM node can be achieved by inserting a link from the text properties of the displayed characters to the corresponding position in the DOM tree. The simplest approach to allow for the dynamic expansion and collapsing of elements would be to link the functionality to form control widgets, such as push buttons. Although, managing the expansion and contraction of elements in such a hybrid system of plain text and push button widgets could be quite complex. Expanding the content should not pose too many problems. However, the collapsing of material could be problematic.

In the second approach, the application would have to check for each

²⁸`tree-widget.el`: Tree widget version 1.1 by David Ponce:
<http://tinyurl.com/hw4h3>

```
[ - ] head
WebTree Application User Manual
[ - ] body
[ - ] h1 WebTree Application User Manual
[ - ] h3 Contents
[ + ] ul Navigation
[ - ] h3 Introduction
[ + ] p Many Internet access solutions for the blind are in reality ...
[ + ] p The WebTree application has been written to explore the use ...
[ + ] p As well as the tree like arrangement of the displayed XHTML ...
[Link Back to top]
```

Figure 4.1: A Sample XHTML page as rendered through WebTree

node in the sub tree of the element to be collapsed, and remove them individually from the rendering. Whereas, with the widget tree approach, the widget for the element in question acts as a wrapper for the element sub tree. Thus, when the collapse command is called, the application can simply hide its content. Therefore, the widget tree option was the one implemented.

4.5.1 The Widget Tree

As the name *widget tree* suggests, the widgets are arranged in a hierarchical form reflecting the organisation of elements in the document. For example, the root node represents the `<html>` element. As the user expands each widget, widgets representing the children of the current DOM node are revealed to the user. For example, when the `<html>` widget is expanded, the `<head>` and the `<body>` elements are exposed through the `wt-tree-expand-widget` function. In cases where the child nodes represent text content rather than XHTML elements, the text appears fully expanded in the display once the widget representing the parent node has been expanded.

Figure 4.1 shows a sample web page as it might be displayed through WebTree. It is included here to show how the tree rendering of the document can be represented. The buttons appearing at the beginning of a tree control containing a `+` indicates elements in their collapsed state, whilst `-` signifies expanded elements. For an in-depth discussion of the WebTree user interface and the tree structural view, see chapter 5.

```

(define-widget 'wt-default 'default
  "Default widget from which wt widgets are to be derived"
  :node-dom nil ;; Pointer to DOM node element
  :voice nil ;; contains voice type information for element sub tree
  :before nil ;; content to be placed before each child element
  :after nil ;; content to be included after each child element
  :emacspeak-help nil ;; pointer to emacspeak helper function.
  ;;(Function depends on the widget type)
  :name "" ;; element id or name attribute
)

```

Figure 4.2: Default widget class

To be effective in the WebTree system, all the widgets used had to be augmented with a number of additional properties. A new default widget shown in figure 4.2 was created as a base class from which all other widgets were derived.²⁹ Every node in the DOM tree that is exposed in the display has its own designated screen widget. Most of these are `wt-tree-element-widgets`, which is equivalent to the `tree-element-widget` found in `tree-widget.el`, except that it is now derived from the new base class. It also contains one additional property `:display-element`, (see the next paragraph). However, there are some specialised widgets included to represent particular elements such as hyperlinks `wt-url-link` and form control widgets. Although widgets to represent these elements are available in the file `wid-edit.el` found in emacs-21 they had to be augmented to inherit properties from the `wt-default` class.

Each tree widget has two control components which are represented in the displayed text. The first is a button that when pressed calls the expand/collapse element functions. The second is a text item which contains the name of the XHTML element. In the file `tree-widget.el`, the `tree-widget-value-create` function automatically inserted these controls into the text for every `tree-element-widget` it created. However, this is not always beneficial. There are occasions where it is better to show the

²⁹The properties of the `default` widget found in `widget.el` (part of GNU emacs) from which the `wt-default` widget is derived are excluded here.

content without these controls, as detailed in section 5.1.1. Therefore, the `wt-tree-widget-value-create` function allows for the exclusion of these controls. If the `:display-element` property is set to `nil`, these controls do not appear in the screen text. Similarly, if the `:open` property is set to `t`, the children of the element appear automatically expanded in the audio rendering. The reason why the `:open` and `:display-element` properties are included in the `wt-tree-element-widget` and not in the `default` widget is that these properties are only applicable to elements derived from the `wt-tree-element-widget`. Plus, the `:open` property is an original component of the `tree-widget` found in `tree-widget.el`, so it did not make sense to duplicate this functionality in the `default` widget.

The presentation of table constructs posed a number of different problems for the modelling approaches taken. None of the existing widget types were capable of handling the implementation of tabular constructs in an efficient and reliable manner. This is due to the additional links required to allow navigation between the different cells, discussed in more detail in section 5.1.6. Similarly, many of the relationships to support navigation are not automatically available in the DOM structure.³⁰ The individual cells already have links to sibling elements within the row in which they appear. However, no explicitly derived relationships exist between neighbouring cells in a given column. To achieve this, it is important to note the column number in which the cell appears. This is due to the `colspan` attribute allowing cells to span more than one column, meaning adjacent rows can contain differing numbers of cells. Similarly relying on the sibling links between elements in a row is unwise because of the possibility of cells spanning more than a single row. In addition, there can exist links between the data cells and their associated header information. There are two possible approaches available to create these links. The first involves creating links in the DOM to point to relevant nodes to be positioned above, below, left and right of the current node. The second solution is to reflect these relationships in the

³⁰Inter-cell relationships would need to be explicitly created in the DOM by WebTree.

widget tree, which is the method implemented by WebTree. Although it may have been preferable to have these links created in the DOM structure, the reason for implementing it in this manner is due to the one-directional linkage between individual screen widgets and nodes in the DOM. For the purposes of the prototype application, it was easier to search the on screen rendering for the specific widgets representing the table cells, as opposed to searching the same content for the relevant DOM node. Also, storing the relevant information to support navigation is rather simple with the widget tree approach. However, for it to work successfully, all the cells need to appear in the rendering. That is, all rows, `<tr>` elements should appear automatically expanded, once the `<table>` element is opened. This requirement was not a problem in terms of the prototype application, for it allowed experimentation with the different navigation and table reading facilities. However, for these functions to work effectively with a `<table>` construct with a mixture of rows in their collapsed and expanded states, the code should be altered to reflect the cell-to-cell relationships in the DOM structure. Figure 4.3 shows the widget class for a table cell. Figure 4.4 shows the widget class for higher level table elements. These elements include the `<table>`, `<tbody>`, `<tfoot>`, `<thead>` and the `<tr>` elements.³¹

Due to the tree structural approach to the construction of the rendering, there is no need to have a specific mechanism to memorise which form fields belong to a given `<form>` element. The application only has to examine the `<form>` element sub tree to find the associated fields. For this reason, multiple forms positioned on the same page do not pose any problems for the system. In general the widgets used to manage form interaction are those stored in `wid-edit.el` (part of GNU emacs 21). However, these were edited to inherit from the new `default` widget class.^{32 33}

³¹The WebTree system does not have any special functionality to handle the `<tbody>`, `<thead>` or `<tfoot>` elements. The `wt-table-element-widget` is inserted without any tree controls as a wrapper for the enclosed content, however provides no other functionality. See section 5.1.6 for the reasons why.

³²Once tree controls are included for the `<form>` element, the user can easily establish the start of a new form.

³³The nesting of `<form>` elements is not allowed in XHTML, therefore problems relating

```

(define-widget 'wt-table-cell 'wt-tree-element-widget
  "table cell widget"
  :id nil ;;HTML id attribute string
  :headers nil ;; HTML headers attribute string
  :row-header-widget nil ;;points to the relevant row header widget
  :column-header-widget nil ;; points to the relevant column header widget
  :scope "" ;; content of the HTML scope attribute string
  :left nil ;; points to the widget cell to the left
  :right nil ;; points to Cell on the right
  :up nil ;; widget cell above
  :down nil ;; widget cell below
  :row-number 0 ;; Number of the current row
  :column-number 0) ;; current column number

```

Figure 4.3: wt-table-cell widget class

```

(define-widget 'wt-table-element-widget 'wt-tree-element-widget
  "for table grouping elements e.g., \htag{tbody},
  \htag{tfoot}, \htag{thead}, \htag{tr}"
  :element-number 0)

```

Figure 4.4: wt-table-element-widget class

Gaining access to the position in the widget tree under focus is not a simple operation. Using the `widget-at` function should return a pointer to the widget under point. However, if the widget concerned is not on the tab-order, then it is not recognised as a widget. This became problematic when the element in focus was either an inline text based element or a group of inline text based elements, which apart from a few special cases, are generally not on the documents' tab order, e.g., `` or ``. These do not normally have any form controls associated with them in the display, thus it was difficult to gain access to the DOM node contained in the widgets `:node-dom` property. The list of widgets on the tab-order include:

- `wt-tree-widget-handle-node` widgets (tree control button): Access to the element tree can be gained by obtaining the parent of the widget through its `:parent` property.
- url-link widgets: `<a>` or `<area>` element node.
- Form field widgets: The widgets associated with form controls. Often a `<input>` element.

For the purposes of most user interactive tasks in the WebTree interface, this restriction didn't cause any major problems. The ability to tab from item to item, and expand or contract individual elements was not effected. However, there were a number of areas of development where this lack of functionality started to be quite limiting. Not only was it difficult to be able to inform the user of the type of element in focus at any given time. It had much more serious implications for the development of navigational features that require large jumps from point to point in the DOM structure. These include the implementation of code to handle relative links, the provision of additional table navigation functionality and the development of searching methodology. For example, once one of the table navigation functions is to which `<form>` construct the elements belong to in a multi level structure of forms do not apply.

requested, the application needs to access the widget representing the current table cell to establish the position to which focus should be transferred. Accessing the widget tree through the next widget on the tab-order is not powerful enough to be able to provide the required functionality. Therefore, it was necessary to find another method to gain access to the widget under point. This method is explained in section 4.5.2.

4.5.2 Screen Overlays

Emacs provides a mechanism of *screen overlays* to assign specific properties to text, such as font size or colour changes. In addition to the set of emacs specific properties, the developer is permitted to assign and apply their own properties. These properties are generally ignored by emacs unless they clash with a set list of keywords that emacs recognises. Overlays can be nested, therefore it is possible for each `wt-tree-element-widget` to have a screen overlay associated with it. To aid interaction through WebTree, the following properties were added to each screen overlay:

- `:element-name` : Name of the XHTML element that the overlay represents.
- `:element-widget` : A pointer to the widget in the tree which represents the XHTML element.

Under WebTree, screen overlays are assigned in the following manner. As each `wt-tree-element-widget` is created in the display by the `wt-tree-widget-value-create` function, a screen overlay is also created. The start and end points of the overlay coincide with the start and end points of the tree widget. The overlay is generated after the widget has been created by adding a piece of `advice` functionality to the `wt-tree-value-create` function. The `advice` function can change how a function operates without altering the specific function to which it is assigned, by specifying code to be run when the assigned method is called. In this case, the `advice` function invokes lisp code to generate an overlay for the widget created

without altering the widget creation function. The `advice` function assigns the XHTML name of the element to the `:element-name` property, and the `:element-widget` property is set to point to the tree widget itself. If the application examines the overlays under point, it can gain access to the current widget in focus by retrieving the `:element-widget` property. Through the examination of the start and end points of each overlay, it is relatively simple to locate the overlay in the list that is most relevant to the current position.

It is important to note that not all screen element widgets have an individual overlay assigned. Those that do are generally widgets that are of type `wt-tree-element-widget`, or have been derived from the `wt-tree-element-widget` type. This is due mainly to the designated `advice` function being only assigned to the `wt-tree-value-create` function.

The form control widgets possess their own generation functions so this piece of advice functionality does not apply. To create an overlay for each form control or field, each specialised creation function would require an association with a piece of advice. Due to the fact that these elements are on the tab-order, this additional step is unnecessary because the `widget-at` function will return the widget under point.

4.6 Cascading Style Sheet implementation

The WCAG 1.0 guidelines (W3C, 1999b) encourage the separation of content from presentation through the use of cascading style sheets. WCAG checkpoint 3.3 states:

Use style sheets to control layout and presentation. [Priority 2] For example, use the CSS 'font' property instead of the HTML FONT element to control font styles.

Cascading Style Sheets are a rather powerful tool used to separate content structure from presentational properties. However, the vast majority

of components in the Cascading Style Sheet level two specification (W3C, 2005) are not directly applicable to the portrayal of material through a purely auditory browser. That is, CSS components to control the visual appearance of elements have little value in a purely audio interface. For example, many of the properties to control visual formatting or to apply presentational aspects such as colour, font size or font weight to an element are not applicable. Nevertheless, there exists a number of CSS constituents that can still have a bearing on this form of interaction. The regular CSS2 properties handled by the WebTree application, include the `display` property, and those properties dealing with the presentation of list items and the inclusion of white space.³⁴

As well as providing instructions to govern the visual presentation of elements, CSS also allows the specification of properties to control the aural presentation of the content. However, so far none of the more popular auditory web browsing solutions for the blind have implemented support for this technology. Only `emacspeak` in conjunction with `emacs-w3` currently supports this facility.

The main focus of the CSS implementation under WebTree deals primarily with the aural style sheet properties to be applied to voice output. At present, only a small subset of these properties has been implemented, being those specifically concerned with controlling voice characteristics. However, others such as those referring to the insertion of pauses, managing speech volume and the playing of non-speech audio cues have not yet been included. As part of future work, it is hoped that many more of the aural style sheet properties will be implemented as part of the WebTree application.

Both user provided style sheets and author provided style information are handled by the WebTree application, although, the use of author prescribed aural styling is rare. The prototype application does not update the DOM with the style information for the different elements. This is due to the use of the `emacs-w3` CSS parser to parse style sheets, which generates

³⁴Code to cope with all possible values for these properties has not been implemented in all cases.

its own internal storage format. W3 also provides functionality to query the parsed representation to find the properties to be applied to a given element. For the purposes of the prototype application, it was thought to be unnecessary to update style information for each element in the DOM tree, for in many cases, the information might not ever appear in the audio rendering. Obviously whether or not this is advantageous is dependent on how much of the tree structure the user chooses to render. Instead, WebTree applies CSS properties to an element as it is expanded into the audio rendering.

The `:voice`, `:before` and `:after` properties of the tree widget for each element are used to store information to be applied to their child elements. For example, the `:voice` property stores the voice settings to be applied to any child elements. If the element in question is a `text` element, then the `:voice` of the parent widget is applied to the data. If no `:voice` element exists for that specific widget, then the ancestor elements are searched. If no `:voice` properties are found, then the default voice is used. The `:before` and `:after` properties are used to store any prescribed text to be placed either side of a child element—for example, a marker to be inserted to denote the start of a new list item.

4.7 Standards Compliance

So far in this chapter the implementation issues faced when building WebTree have been dealt with. Now the external factors that influence the accessibility levels of certain web sites will be highlighted. Thus, the remainder of this chapter will focus on the standards and guidelines which are required to make WebTree, or in fact any purely auditory based browser a workable solution. First the different guidelines to aid web accessibility are discussed. Following this, the benefits of using mark-up that validates against a recommended document type definition are presented. Finally, an overview concerning the different pieces of legislation governing the area of accessibility (both nationally and internationally) and how they are applied to the web is presented.

4.7.1 Guidelines

The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect.

—Tim Berners-Lee, W3C Director and inventor of the World Wide Web³⁵

For most people, the web is a very powerful tool providing immediate access to multiple information resources. However, for people with disabilities it can take on a greater significance. It provides the potential to increase access to many products and services that people not affected by disability can sometimes take for granted. Nevertheless, there can exist many barriers inhibiting usage of the web by such users. In saying this, a number of measures can be implemented to offset many of these problems and ensure that a web page is accessible to such user groups. There currently exists many diverse sets of guidelines proposed in the hope of making the web accessible to all. These include the U.S. governments Section 508³⁶ requirements, applicable under the Americans with disabilities act, which explicitly makes provisions to maintain a certain level of web site accessibility. However, a more comprehensive set of guidelines are produced by the W3C's Web Accessibility Initiative³⁷ (WAI). These guidelines attempt to ensure accessibility for a greater number of user groups.

WAI maintain that it is not just web site developers that have responsibility for accessibility issues. For this reason the Guidelines produced by the W3C's WAI initiative are divided into three separate areas to reflect the responsibilities placed on user agents, web site developers and the makers of web content authoring tools. The list of guidelines is broken down into the following documents:

- User Agent Accessibility Guidelines 1.0 (UAAG) (W3C, 2002a). This document defines criteria for making user agents such as web browsers

³⁵<http://www.w3c.org/wai>

³⁶<http://www.section508.gov/>

³⁷<http://www.w3c.org/wai/>

and media players accessible to the largest possible audience.

- Web Content Accessibility Guidelines 1.0 (WCAG) (W3C, 1999*b*). These are intended to govern how a web site can be coded to meet accessibility requirements. However, some of these requirements are rather subjective or ambiguous, such as making colour contrasts sufficient so that visually impaired users can recognise colour differences. The current WCAG recommendation is WCAG 1.0. However, a revised set of guidelines, WCAG 2.0, is currently being drafted (W3C, 2006).
- Authoring Tool Accessibility Guidelines 1.0 (ATAG) (W3C, 2000*a*). The ATAG guidelines were primarily created to assist developers of authoring tools. These include editing tools specifically designed to produce Web content, for example, HTML and XML editors. In addition, tools that allow the user to save content in a Web based format, for example, word processors or desktop publishing packages. They also guide the creation of applications to convert different document types into Web based formats. For example, filters to transform desktop publishing formats into HTML. Furthermore, they cover the development of applications such as content management systems (CMS), web based multi media applications, and dynamically generated websites from server side databases.

The guidelines developed by WAI³⁸ are widely accepted as being the de facto international standard for Web accessibility. Also, section 508 guidelines have no bearing in an Irish context, thus, in this dissertation focus will be restricted to those guidelines produced by the WAI. However, in terms of the WebTree application, only the UAAG and WCAG guidelines are directly applicable. Although the ATAG guidelines do have a bearing on web content accessibility, they are more concerned with the automated creation of accessible content. Therefore, an in-depth discussion of ATAG is out of

³⁸<http://www.w3c.org/wai>

scope for this dissertation.

Both the UAAG and WCAG guidelines have three conformance levels. The guidelines are structured in the following manner. Each guideline has a number of checkpoints associated with it to ensure conformance. The checkpoints are assigned a priority level ranging from one to three, in order of their importance to accessibility. Thus, to conform to WCAG level Double-A, for example, all the priority one and two checkpoints must be fulfilled.

User Agent Accessibility Guidelines (UAAG)

The UAAG document (W3C, 2002*a*) provides guidelines for designing user agents that lower barriers to Web accessibility for people with disabilities (visual, hearing, physical, cognitive, and neurological). These user agents include HTML browsers and other types of software that retrieve and render Web content. Currently WebTree is a purely audio web browser, with a focus on the presentation of XHTML/XML files. Many of the guidelines recommended by UAAG are not applicable for two reasons. The first being that focus is centred on non-visual usage. The prototype application doesn't attempt to support access through other modalities. There is little current support for interaction through the visual modality. That is, apart from a basic visual display of the audio rendering. The second reason is that documents containing alternative media presentations such as flash content, or material which is dynamically rendered on the client side using JavaScript are not supported by WebTree. See section 4.3 for further details. Similarly, many of the guidelines specifically deal with the presentation of alternative media types such as video and audio files, which are not covered by the WebTree system. The development of the prototype application has so far been focused on creating an alternative mechanism for browsing documents. Access to these alternative media types must be handled by external applications, thus, WebTree is not responsible for the accessibility of these applications. To conform to the guidelines all components used in the user agent must comply with the criteria set out in the UAAG. However, as of

now no conformance claims with the UAAG guidelines are made in relation to WebTree. Although, many features of WebTree do conform. For example, allowing interaction through the keyboard. Other input modalities are facilitated due to the application being implemented under emacs. As part of future work, the system could be altered to meet more of these requirements.

Web Content Accessibility Guidelines (WCAG)

As previously mentioned, the Web Content Accessibility Guidelines (WCAG) W3C (1999b) were designed to enable web developers ensure their web site is accessible to the widest possible audience. WCAG 1.0 consists of 14 separate guidelines, each of which has an associated set of one or more individual checkpoints. There are a total of 65 checkpoints which are classified into three priority levels (1-3), defined in the Guidelines as follows:

Priority 1 A Web content developer must satisfy this checkpoint. Otherwise, one or more groups will find it impossible to access information in the document. Satisfying this checkpoint is a basic requirement for some groups to be able to use Web documents.

Priority 2 A Web content developer should satisfy this checkpoint. Otherwise, one or more groups will find it difficult to access information in the document. Satisfying this checkpoint will remove significant barriers to accessing Web documents.

Priority 3 A Web content developer may address this checkpoint. Otherwise, one or more groups will find it somewhat difficult to access information in the document. Satisfying this checkpoint will improve access to Web documents.

Based on these priority levels, three levels of conformance to the WCAG 1.0 can be achieved, which are described in (Marincu and McMullin, 2004)

as follows:

WCAG-A: All priority 1 checkpoints are satisfied. This is a minimum standard which a site must meet to be considered accessible for any significant disability groups.

WCAG-AA: All priority 1 and two checkpoints are satisfied. This is a "professional practice" standard, which a site should meet to be accessible to a broad range of disability groups.

WCAG-AAA: All checkpoints (at all priorities) are satisfied. This is a "gold standard" of maximum accessibility which some sites may choose to aim for—for example, sites with a particular remit to serve disability communities.

Although developers should be encouraged to meet all of the checkpoints for each guideline, conforming to some guidelines is rather subjective. For example, conforming to guideline 14:

Ensure that documents are clear and simple.

Checkpoint 14.1 states:

Use the clearest and simplest language appropriate for a site's content. [Priority 1]

This guideline aids people of all abilities to access the content. However, what is termed as being clear and simple language by one person may not be for another. It is also difficult to automate testing for such issues.

There exist a number of automated solutions facilitating the checking of a web page for accessibility issues, for example, AccVerify³⁹ and Watchfire WebXACT⁴⁰. However, these can generally only test for certain types of accessibility problems that can be gleaned through examination of the

³⁹<http://www.cynthiasays.com/About%20Reports/DataTables.htm>

⁴⁰<http://webxact.watchfire.com/ScanForm.aspx>

source code. For example, does an image `` tag have an associated “alt” attribute. Nevertheless, other accessibility issues such as, determining whether tabular material linearises gracefully or whether the contrast between foreground and background colours is sufficient are more difficult to test automatically. Thus, although these tools are useful in auditing a web site for accessibility, review by a person with knowledge of accessibility issues is still necessary. Also, to be sure that a web site works correctly through assistive technologies some user testing with members of this user group is recommended. However, this is a contentious issue at the moment. In an ideal, world, if the WCAG guidelines did what they were supposed to, and if evaluation was done accurately, and if assistive technologies worked properly - i.e., in conformance with the UAAG (W3C, 2002a) - then in fact there should be no need for actual testing with any particular Assistive Technology. The problem in the real world is that many of the guidelines are open to interpretation therefore some testing with assistive technology is a useful control on the application and interpretation of the WCAG guidelines. Nevertheless, one must always be very careful not to respond *naively* to bad results from accessibility user testing - problems in using a site, identified in such testing, can sometimes turn out to be problems with the client side technology, or the individual user’s competence or experience; and in that case, changing the server side content is not necessarily an appropriate or effective response.

Not all the checkpoints for each guideline listed in the WCAG guidelines (W3C, 1999b) are directly applicable to accessibility for totally blind people. That is, people who depend on an auditory browser or a screen reader application to interact with a web page. Many of these checkpoints are aimed at ensuring accessibility for other user groups. For example, the use of relative units for setting attributes of mark-up elements e.g., setting font size relevant to the default size, so that the material is scalable for people with low vision. Also, the use of colour contrasts is not usually an issue for blind users. That is, unless there is a dependency on colour to denote contextual

information. For example, using the colour “red” to denote that a form field is required to be filled in. In such cases the WCAG guidelines recommend using alternative means for imparting this information in conjunction with the colour change. Therefore, documents do not need to conform to these guidelines for WebTree to function. When reviewing the WCAG guidelines, each individual checkpoint is not examined to determine whether it is applicable to web access for blind people. However, an overview of the most important relevant areas that the guidelines cover is presented.

The requirements necessary for providing access through auditory solutions are generally those concerning the inclusion of alternative means for obtaining information about visual elements, such as, images `` elements and image maps `<imagemap>` tags. Also content that is dynamically altered by a client side script can be problematic for blind users. For this reason, checkpoint 8.1 states:

Make programmatic elements such as scripts and applets directly accessible or compatible with assistive technologies [Priority 1 if functionality is important and not presented elsewhere, otherwise Priority 2.]

Visually it is easy to make the user aware of a change in content on another part of the screen when a selection is made. However, imparting this information to a blind user is not a simple task. This is due to the narrow focus on the content provided by the mode of interaction. Similarly, those guidelines dealing with the use of new technologies are directly applicable to accessibility for blind people. Guideline 6 states:

Ensure that pages featuring new technologies transform gracefully.

When new technologies are released, they are not always created with the provision of accessibility in mind. Thus it can take a number of years before auditory solutions implement a means of interacting with these technologies. If these technologies depend greatly on visual interaction to perform tasks,

then new methods of how to present the information to a blind user will need to be investigated. For these reasons, it can take some time before such content can become accessible if at all.

It is also important to clarify natural language usage. For example, use mark-up that facilitates pronunciation or interpretation of abbreviated or foreign text. Speech synthesisers and Braille devices can automatically switch to a new language if it is specified in the mark-up. This should ensure the document is more accessible to multilingual users. The natural language of a document's content should be signalled (through mark-up or HTTP headers). Expansions of abbreviations and acronyms should also be provided. When abbreviations and natural language changes are not identified, then the relevant content may be indecipherable when machine-spoken or Brailled.

As demonstrated in section 3.6, it is important to use the correct code to mark-up tabular constructs. In addition, the `<table>` construct should only be used to mark genuinely tabular content and not be used to create a visual multi-column display. That is, once CSS positioning is supported by user agents, which is currently the situation for many of the web browsing solutions available today.

Developers should also ensure that moving, blinking, scrolling, or auto-updating objects or pages may be paused or stopped (WCAG guideline 7). That is, until user agents afford the user control over such items. The reason for this is that some screen readers may have trouble with this type of content. Unless refreshing of the material can be suppressed, the screen reader application will move back to the top of the page and start reading after each content refresh. However, due to WebTree allowing the user to control the presentation of this type of material, this guideline is not applicable in this case.

Each form field should have a `<label>` element explicitly associated with it. WebTree supports this technology. However, the `<label>` information is not automatically announced when a field/control is encountered when

reading through the content line by line. Therefore, the recommendation that labels be placed on the same line or the one previous to the field/control to aid browsing solutions that do not support the `<label>` element can be beneficial in this case. The user can then develop a consistent strategy for associating content with the correct form field whilst reading in this manner. The recommendation that form fields should contain a default value to aid access by older browsers is not applicable in this case. WebTree announces when such a field is encountered, and automatically allows the user enter/edit the content.

Guideline 9 lays down some provisions for interacting with the content in a device independent manner. For blind people to navigate the document, access through the keyboard is necessary. Using a pointing device such as a mouse is impractical if you cannot see the content. This mode of interaction relies heavily on the ability to see the screen elements and how they relate to one another. So that screen reader applications can gain access to alternative text, client side image maps should be used in favour of those stored on the server side. Unfortunately, it is intrinsic to the technology of server side image maps that they cannot be made directly accessible. Whereas, with a client side image map, alternative text can be included by placing an "alt" attribute on the enclosed `<area>` tags. Where it is not practical to replace a server side image map with one on the client side, a fully accessible alternative mechanism equipped with the same functionality must be provided.

The idea of device independence is also extended to applications imbedded within a document. For such elements the relevant accessibility guidelines for that technology should be followed to ensure that the content is accessible to all. Also, the use of `access` key short cuts, plus a `tab-index` specified for the document are recommended. However, these are priority three recommendations.

According to guideline 11, the use of W3C technologies (e.g., HTML, CSS, etc.) are recommended for creating web based documents. The reason

for this is that W3C technologies include “built-in” accessibility features. These specifications undergo early review to ensure that accessibility issues are considered during the design phase. Often non-W3C formats require an external application to be presented. This is also the case for WebTree. Although much work has occurred in improving the accessibility of a number of proprietary formats e.g., PDF (Portable Document Format) and Shock-wave flash content, much care must be taken to ensure these documents are accessible. Avoiding non-W3C and non-standard features (proprietary elements, attributes, properties, and extensions) will tend to make pages more accessible to more people using a wider variety of hardware and software. When inaccessible technologies (proprietary or not) must be used, equivalent accessible pages must be provided. Of course, even when W3C technologies are used, they must be used in accordance with accessibility guidelines.

It is also important to provide context and orientation information to help users understand complex pages or elements (guideline 12). Grouping elements and providing contextual information about the relationships between elements can be useful for all users. Complex relationships between parts of a page may be difficult for people with cognitive disabilities and people with visual disabilities to interpret. For this reason large blocks of text should be sub-divided into more manageable blocks. Header elements should be used where appropriate.

Finally, it is important to provide clear and consistent navigation facilities. If the structure of a page remains constant across a collection of pages, then a blind person can develop methods to navigate to the relevant areas of the page based on this consistency. For example, if the navigation bar is organised consistently from page to page, the user can quickly learn the location of the different navigation segments. Therefore, if the user knows where to look for a specific navigation link, they can easily and efficiently move to that point in the document. However, methods to skip passed navigation bars or blocks of ASCII art should be provided. It is also important to give knowledge of the target page when including link text. Many audi-

tory solutions allow the user move to the link out of context with the main text, by tabbing from link to link, or by examining a list of page links. This can be beneficial where the user is looking for a specific link, or knows that that the information they required is linked to the current page. By viewing the list of available hyperlinks the user can quickly navigate to the relevant link non-linearly. Therefore, making navigation through a serial medium more efficient. However, if the link text is not descriptive, the user may have some trouble deciphering the content of the target page.

4.7.2 Validation

A web page is properly constructed when its mark-up conforms to a standard technical specification. In the case of documents derived from the SGML specification (Maler and Andaloussi, 1995), which includes the HTML family of languages, each standard is specified by a Document Type Definition (DTD) document which contains descriptions of the entities, elements and attributes that can be part of the document, and how these elements relate to one another. Although web pages can be written in a non-SGML derived language, e.g., PDF or RTF, for the purposes of this discussion focus will centre on content produced in a language derived from SGML. The reason for this is that WebTree does not currently support other document types. Because most of the existing Web browsers are able to render—to at least some extent—Web pages which don't conform to a DTD, many of the failures in the HTML code can pass unnoticed by most users. But such code defects can be a real access barrier for users with disability helped by special purpose Web browsers and dedicated assistive technologies. They also complicate, and therefore inhibit, ongoing development of such niche technologies (Marincu and McMullin, 2004).

Although validating to certain coding standards does not mean a web site is automatically accessible, it is widely believed to be a good basis on which accessibility features can be built. For this reason WCAG check point 3.2 states:

Create documents that validate to published formal grammars. [Priority 2]For example, include a document type declaration at the beginning of a document that refers to a published DTD (e.g., the strict HTML 4.0 DTD).

Marincu and McMullin (2004); UK Cabinet Office (2005) investigated how widespread the use of valid mark-up is across a large sample of websites. However, both of these studies have discovered that very few of the websites examined contained valid mark-up. UK Cabinet Office (2005) cites the failure of many web authoring tools and web content management systems to produce valid mark-up as being a major source of this problem. The use of tools that conform to both the ATAG (W3C, 2000*a*) and UAAG (W3C, 2002*a*) guidelines are recommended to be used instead. Currently a number of tools complying with many features of the ATAG are available. They also reason that a lack of training on behalf of web content authors in the use of such tools is also a contributing factor.

Many assistive technologies operate in conjunction with a fully fledged web browser, e.g., JAWS and Internet Explorer. They rely on these browsers to provide them with a document model from which they gain access to the content. These browsers often have very powerful code to both parse documents and correct errors. However, there is no guarantee that the error correction mechanisms will detect and fix every page error. In using valid code an element of ambiguity is removed from the process. Take for example the use of non-unique "id" attributes within a `<table>` element. If a number of cells have the same "id" attribute then screen reader applications might have problems associating cells with the correct header information but this will not be apparent to visual users.

Another important concept in web page construction is to use elements for their intended purpose and not for purely presentational effects. For example, mark up headers with the `<h*>` tags and do not use header mark-up for text that is not logically a header. A second example, would be to use the `<table>` element to mark up tabular constructs as opposed to formatting

such data using the `<pre>` element. By using elements for their intended structural meaning, assistive technologies can generate renderings better suited to the auditory modality. In addition, navigation facilities based on these elements e.g., move to the next/previous header can be introduced. See section 3.5 for more details on the types of navigation facilities available.

In terms of the WebTree system, the use of well-formed structural markup is an important feature of a document's structure. As will be seen in the next chapter, the efficiency of the viewing mechanism is dependent on the arrangement of elements. If little care is taken during the creation of these documents, then any advantages associated with the viewing approach may be lost. For example, if deep nesting of elements containing small amounts of content occurs, having to expand each element of the tree individually to gain access to the content may be an inefficient approach. Similarly, if only a limited structure is imposed on the content, then the ability to gain a quick overview of the material may be problematic.

4.7.3 Accessibility Levels and Legal Requirements

So far, there have been a number of studies investigating the inclusion of accessibility features in Web sites (McMullin, 2002*a*; Marincu and McMullin, 2004; UK Cabinet Office, 2005). Unfortunately, each of these studies show that the current uptake of accessibility features in website construction is still quite low. All of these studies measured a sample set of websites against conformance with the Web Content Accessibility Guidelines. Whereas, UK Cabinet Office (2005) was concerned with accessibility of public sector websites, the warp project McMullin (2002*a*); Marincu and McMullin (2004) examined sites gleaned from both the public and private sector. However, the results of all of these studies are consistent with a slow uptake in accessibility inclusion.

McMullin (2002*a*); Marincu and McMullin (2004) tested the levels of website accessibility exclusively with an automated checker. Whereas, UK Cabinet Office (2005) also included some manual checks on a subset of the

sample site list. Marincu and McMullin (2004) found that only approximately 5% of the sites tested met the automatically testable priority one checkpoints of WCAG Single-A requirements. However, it is not possible to conclude that these sites meet all the criteria for WCAG single-A conformance. The manual checks necessary to claim such conformance levels were not performed. Therefore, it is quite possible that most, if not all, of these sites would actually have failed on one or more of these requirements.

UK Cabinet Office (2005) found that 3% of the 436 online public service websites assessed achieved Level Single-A conformance with the W3C Web Content Accessibility Guidelines (WCAG 1.0) passing the full suite of both automated and manual checks. A further 10% of services fully passed all the automated checks, but showed a material failure on one or more of the manual checks. Another 17% of sites failed one or more of the automated checks, but this failure was limited in extent or scope. Finally, the remaining 70% of sites showed relatively pervasive failure against one or more of the automated checks. No site that achieved Level A conformance was found to achieve the higher standard of Level Double-A conformance. The authors of this report maintain that with a small amount of work, the 10% of sites only failing the manual checks could be made accessible, reaching single-A conformance levels. Similarly, that 17% of sites that had limited failures on the automated tests could also be fixed to meet the grade. Much of the failure to meet Double-A conformance is blamed on authoring tools and web content management systems not producing valid mark-up and not enforcing accessibility requirements at the authoring stage. They also cite a lack of training on behalf of developers to ensure tools that can produce standards compliant mark-up and accessible pages do so, on a regular basis. Finally, some policy recommendations to increase the amount of sites meeting accessibility requirements are presented.

Although WCAG 1.0 accessibility conformance levels are still low, there is good reason to believe that the number of accessible sites will significantly increase in the near future. The first reason for this is the W3C is investing

a lot of resources in creating educational material to assist authors in creating accessible websites. These include tutorials and techniques to ensure the requirements for a given checkpoint are met. A second reason is that the number of tools complying with the features of ATAG (W3C, 2000a) guidelines is on the increase. With some training in the use of these applications, authors will be able to generate content meeting accessibility criteria. Another major reason is the increase of disability access legislation in an international context. In designing this legislation many governments have included explicit reference to the issue of web accessibility. Some countries, such as the U.S. provide their own set of accessibility guidelines. Whereas, other countries have adopted the WCAG 1.0 guidelines as the benchmark for web accessibility requirements. Usually the entire set of guidelines is not required. Instead, provisions are often limited to meeting either WCAG 1.0 Single-A or Double-A conformance levels. According to UK Cabinet Office (2005), those countries featured in their study, which have relevant legislation governing web accessibility, in most cases fared better than those for whom no such requirements exist. One of the exceptions was Denmark who at the time this report was compiled had no such legislation. Instead, the authors of this report believed that a competition for the best public website, of which one of the criteria requires accessibility, is a major factor in ensuring accessibility of such websites.

According to McMullin (2002a), (Waddell and Urban, 2000) tells us that in the U.S., both the Americans with Disabilities Act (ADA) and Section 508 of the Rehabilitation Act are generally regarded as imposing significant obligations on Web site operators to ensure accessibility for users with disabilities. Section 508⁴¹ requirements of the Rehabilitation Act, makes explicit reference to criteria for web accessibility. These roughly equate to the Single-A requirements of WCAG 1.0. In Ireland, the piece of legislation responsible for access to the web for people with disabilities is the Irish

⁴¹<http://www.section508.gov/>

Disability Act 2005⁴². The draft code of practice⁴³ under this act makes explicit reference to conforming to WCAG Double-A requirements. Therefore, if adopted, it will ensure that requiring WCAG Double-A conformance for public sector websites is placed on a legal footing. Although the provisions of the Irish disability act only refer to websites in the public sector, other acts, - particularly the Equal Status Act 2000⁴⁴ and the Employment Equality Act 1998⁴⁵ - have wider scope, including the private sector; but the application of these to web accessibility is still unclear, pending any complaints and case law.

In the U.K., the Disability Discrimination Act, and the more recent Special Educational Needs and Disability Act, create obligations on website providers to ensure pages are accessible to people with disabilities (Sloan, 2001). Similarly, many other countries such as Australia (Disability Discrimination Act 1992), Canada (Canadian Human Rights Act of 1977), have legislation governing web accessibility requirements. For more information on the types of legislation and documents relevant to web accessibility for these and many more countries, see the WAI's policy page⁴⁶. Although many countries do not have direct legislation governing accessibility requirements, some do have action plans to encourage the implementation of such criteria, e.g., Denmark. It should be remembered that the WAI's policy page is not an exhaustive list of legislation for each country. That is, just because relevant legislation for a given country is not listed on this page, it does not mean that such legislation or indeed an action plan for accessibility implementation does not exist.

Although many countries have legislation governing the rights of people with disabilities, there has yet to be much case law to test whether these acts are applicable to the web. According to Sloan (2001), the reason for this is that most complaints are settled before they reach a court

⁴²<http://www.oireachtas.ie/viewdoc.asp?DocID=4338>

⁴³<http://tinyurl.com/o3rdp>

⁴⁴<http://www.oireachtas.ie/viewdoc.asp?DocID=2409>

⁴⁵<http://www.oireachtas.ie/viewdoc.asp?DocID=5663>

⁴⁶<http://www.w3.org/WAI/Policy/Overview.html>

of law. The most famous case in favour of legislation relating to the web is “Maguire verses SOCOG” in Australia, relating to the Sydney Olympics website in 2000, which IBM as the provider of the website lost. For more details on this case see the Reader’s Guide to Sydney Olympics Accessibility Complaint⁴⁷. However, at the time of writing, there is a case pending in the U.S. to determine whether the “Americans with disabilities act (ADA)” is applicable to the web. The case in question is “Sexton verses Target”. Bruce Sexton Jr. has joined the National Federation of the Blind (NFB) as a plaintiff in a lawsuit that charges Target⁴⁸ with violating the federal Americans with Disabilities Act (ADA) and California’s Unruh Civil Rights Act and Disabled Persons Act. The decision in this case could have wide reaching consequences on the accessibility of U.S. based websites. It ought to clear up whether the ADA which was enacted prior to the web can be applied to such technology. If the complaint is up held, then it could force other websites to embrace accessibility for fear of prosecution under these acts. However, failure in this case, may mean companies are not compelled to provide accessible websites. For more information concerning this case, see the article: Accessibility Issue Comes to a Head⁴⁹.

4.8 Summary

In this chapter the implementation issues faced during the creation of the WebTree system were discussed. Also, the standards on which the viewing approach is based coupled with a description of legislation in the area of accessibility were presented. In the next chapter the user interface of WebTree and the viewing approach it facilitates will be looked at in much detail.

⁴⁷<http://www.contenu.nu/socog-PR.html>

⁴⁸<http://www.target.com>

⁴⁹<http://tinyurl.com/q3mxxm>

Chapter 5

WebTree User Interface

In the previous chapter, many of the implementation issues faced by the WebTree system were outlined. The discussion covered both the methods used for document modelling and how the content is rendered in the virtual display. The implementation environment plus the standards and guidelines on which the system is based were also discussed. This chapter provides a description of the WebTree WWW document browser user interface. First, the discussion focuses on the mechanism in which the individual mark-up elements are conveyed. This includes the proposed tree-like arrangement of the displayed elements. Also shown is the way in which information stored in complex data types, e.g., tabular data and/or interactive form constructs, are handled by the system. Following this, a description of the customisation facility governing the method in which individual elements are displayed under the tree modality is discussed. It is shown how alternative views of the same document can be generated using this facility. Next, characteristics of the types of auditory output produced by the system are presented. This discussion includes the use of alternative voices and/or audio cues to denote meaning, and the additional content spoken to signal the type of element encountered. Finally, a number of reading strategies facilitated by the WebTree system are also discussed.

5.1 User Interface

As was discussed in section 3.8, Chimera and Shneiderman (1994) demonstrated that expand/collapse interfaces increased the efficiency of many tasks in comparison to viewing the same content through a static interface. These experiments were performed on a large table of contents. By expanding each chapter, section or subsection element the user gained access to the relevant subordinate elements.¹ The stable/linear interface showed the entire table of contents that needed to be scrolled by the user. All subordinate elements were displayed in conjunction with the chapter headings. This work demonstrated that the approach was a viable solution to working with large quantities of hierarchical information. In WebTree, a similar approach is applied to the viewing of hierarchically marked up documents. However, in addition to the different mark-up elements displayed in the view, the rendering can contain the textual content of the elements already expanded. Also, some additional information is presented concerning the type of content enclosed in an unexpanded element. The work performed by Chimera and Shneiderman (1994) was an experiment to see if the approach provided any advantages for sighted users. The work described here examines whether the approach is viable for a blind user to read documents.

The WebTree system relies on the structural arrangement of the document's underlying mark-up elements to determine how the content is to be portrayed. In addition to the expand/collapse functionality, the user decides how much of the subject matter is to be presented at any given time through the customisation facility. Through this mechanism, the individual elements to appear in the display can be selected. This provides a methodology for generating alternative views of the same document. See section 5.1.2 for more details. Similarly, style information relating to the individual elements can be gleaned from an aural style sheet. See section 4.6 for more information.

¹Only hierarchical divisional elements were shown in the view, and not the textual content enclosed in these elements.

Currently WebTree displays web pages in the serial form that the elements appear in the mark-up. It does not intelligently try to capture the spatial relationships between elements that are not specified directly in the mark-up. Take for an example a `<table>` construct. The application assigns a column and row number to each cell in the table. Cells spanning more than one cell are included, and the column/row number is altered depending on the value of the “colspan” or “rowspan” attributes respectively. In addition header information for a cell is assigned through explicit definition in the HTML mark-up. WebTree is only concerned with data tables. The WCAG guidelines explicitly state that tables should not be used for layout purposes. Although these constructs appear linearly in the rendering, navigation is afforded along the spatial connections between the cells, as described in section 5.1.6. In the case of `<form>` constructs, the application relies on explicit connections between `<label>` elements and the specific form control to provide contextual information. Thus, if the intended arrangement of the content does not mirror its serial presentation in the mark-up, users can still obtain information about the different elements. Without an explicit `<label>` element and form control relationship, it is difficult to determine the context in which it appears if the page uses additional spatial formatting to arrange the content. For more information on how WebTree presents form constructs, see sections 5.1.7 and 5.2.2.

The user interacts with content rendered by the WebTree application through a character-oriented virtual screen/display. By navigating through the rendered material, the user builds up a mental model of both the type of content being presented and the structure in which the different elements are organised. By expanding and collapsing segments of this structure, the user can gain access to the desired material. There exist three major static points of reference for the user to guide them during navigation. These include the left margin and both the beginning and ending points of the document. All other points are subject to change. The virtual screen width (right margin) is governed by variables stored in the customisation facil-

ity. Potentially, a line may contain hundreds of characters, depending on the user's preferences. Navigation within this rendering can be achieved by moving character-by-character, word-by-word or line-by-line in either direction. Alternatively, navigation can be achieved by invoking some WebTree specific navigational commands. These additional commands are described in the following subsections.

It should be noted that where the words *display* and *screen* are used throughout the rest of this chapter, this refers to this virtual display rather than any visual rendering of the content.

5.1.1 Primary Document View

The primary document view is derived from the hierarchical tree-like arrangement of mark-up elements. The view consists of a combination of buttons representing the XHTML mark-up elements, and plain text from elements whose content has already been expanded. These buttons when activated, call functionality to expand or remove their content from the display. Each tree control has two components. The first is the button that controls the expansion/text removal process. The second component provides the user with some contextual information about the element under point. This includes the name of the element e.g., ul, and the content of the element's title attribute if it exists. In the case of paragraphs, <p> elements, the first number of characters contained within are automatically exposed. This is done to provide some indication as to the type of enclosed content. The length of this string is dictated by the user through the customisation facility. Obviously, the benefits of this method of providing contextual information about a paragraph is dependent on how well the document is written. There will be cases where the segment of text shown as part of the tree control bears little resemblance to the actual content. However, for many situations, the string will be an accurate guide to the enclosed material. As each element is expanded into the text, the child components appear in the virtual display beneath the parent element prior to

```

[-] head
WebTree Application User Manual
[-] body
[-] h1 WebTree Application User Manual
[-] h3 Contents
[+] ul Navigation
[-] h3 Introduction
[+] p Many Internet access solutions for the blind are in reality ...
[+] p The WebTree application has been written to explore the use ...
[+] p As well as the tree like arrangement of the displayed XHTML ...
[Link Back to top]

```

Figure 5.1: A Simple XHTML page as viewed through WebTree

the next item.

The reason the element name was used as the content type indicator, e.g., `ul` as opposed to a longer more descriptive text such as `unordered list` was to try to restrict the verbosity of the system. It is important to use short precise cues because they provide as much possible information in the least amount of time. However, as seen in section 6.4, the concern was raised that using element names to signify their presence may be problematic for users unfamiliar with the mark-up language. Therefore, the ability to customise the display to present longer versions of the element name should be included. Alternatively, a context sensitive help function to announce a longer version of the cue may also be practical.

Figure 5.1 shows an emacs buffer image representing a simple XHTML page as it might be rendered by WebTree. To avoid confusion on behalf of the user, the tree control information (including any available contextual information) for each element appears on a line of its own, with its child elements residing immediately beneath. If the content of an element is just plain text or its children are just *inline* elements, the enclosed content is presented on the same line as the tree control once the element has been expanded. The buttons represented by `[+]` denote unexpanded elements, and those with `[-]` indicate elements already expanded. Figure 5.2 shows the same page as rendered through JAWS and IE.

Always having to expand each level of the tree to find relevant informa-

WebTree Application User Manual
Heading level one WebTree Application User Manual
Heading level three Contents
This page link Introduction
This page link User Interface
This page link Customisation
This page link Forms Interface
This page link Table Navigation
This page link Searching
This page link Document Retrieval and Browser History
This page link Index of Keystrokes and commands

Heading level three Introduction

Many Internet access solutions for the blind are in reality just dedicated audio interfaces that serve as add-on applications to out of the box visual browsers. However, it must be noted that there are huge differences in the type of methods of interaction between using a predominantly speech interface and that of visual interaction. The Human eye is expertly capable of scanning through the document to establish what is deemed to be the important page content. This is done by examining the spatial relationships between elements and through the use of visual cues, such as colour and emphasis, included in the text by the author. Unfortunately, due to the cereal nature of speech technology, this is not possible with a purely speech output interface, for it is only possible to examine a single point in the document at any given time. To avoid any confusion on behalf of the user when reading elements that depend on their spatial layout for easy comprehension, many of these applications output the content in the linear format in which it appears in the mark-up. Thus, the user must navigate through all elements that appear in the file before the main content, before the main content is reached.

...

This page link Back to top

Figure 5.2: A Simple XHTML page as viewed through JAWS/IE

```

[-] head
WebTree Application User Manual
[-] body
[-] h1 WebTree Application User Manual
[-] h3 Contents
[-] ul Navigation
o. [Link Introduction]
o. [Link User Interface]
o. [Link Customisation]
o. [Link Forms Interface]
o. [Link Table Navigation]
o. [Link Searching]
o. [Link Document Retrieval and Browser History]
o. [Link Index of Keystrokes and commands]

[-] h3 Introduction
[+] p Many Internet access solutions for the blind are in reality ...
[+] p The WebTree application has been written to explore the use ...
[+] p As well as the tree like arrangement of the displayed XHTML ...
[Link Back to top]

```

Figure 5.3: A Simple XHTML page as viewed through WebTree

tion is thought to be rather cumbersome. Therefore, through the customisation facility provided by WebTree, the user can select whether an element is to be rendered automatically in its expanded or collapsed state. Figure 5.3 shows the same page as in figure 5.1 with the `` element automatically expanded. Furthermore the user may also choose whether it is necessary to assign tree controls to a given element in the virtual display.²

Figure 5.4 shows the same document as in figure 5.3 with a number of tree expansion controls removed to reduce the amount of clutter in the audio rendering.

Assigning tree expansion controls to *every* element would have reduced the overall usability of the system. Therefore, tree controls are reserved for *block* level XHTML elements, as set out in the XHTML basic specification (W3C, 2000b). That is, *inline* elements are automatically expanded

²Enabling both these options—collapsing an element type and hiding its (collapsed) control—would effectively hide the element including its entire element sub tree completely from the audio rendering. While this is not prevented, it would not normally be a useful configuration. Except in cases where a document view limited to contain only specific elements is required.

at the same time as the parent (block level) element. There are, however, two exceptions, namely the `
` and `<hr />` elements. When these are encountered by the display functions, a line break and a horizontal line are inserted into the rendering respectively. To aid the usability of the interface, no tree expansion controls are associated with *inline* elements.

The decision to provide tree controls for only *block* elements and not for all elements found in the document is due to the effect the latter interface state would have on the readability of the content. As speech is serial in nature, the aim is to keep the interface verbosity as low as possible. In general, block elements have the potential to contain large amounts of content so allowing the user control whether they appear in a collapsed state ensures that they can be easily bypassed. As inline elements regularly contain much smaller amounts of information, having to expand such content would seriously impinge on the efficiency of the system. In addition, requiring the reader to listen to the tree control information for each *inline* element could drastically increase the verbosity of the application, hence increasing the time taken to assimilate the material.

As XHTML prohibits the use of *block* elements inside `<h*>` elements, it was decided to treat such elements in a similar manner to *inline* elements. That is, their content is automatically presented in a fully expanded form. If the user selects to have no tree expansion controls associated with these elements, their type may be indicated through speech parameters controlled by an aural CSS. It was decided to expand the `<h*>` tags automatically, for their content often provides a good indicator of the type of material contained in the ensuing document segment. Thus, it can give the user an idea of the subject matter contained in these elements before expansion.³

The rendering of hyperlinks (`<a>` elements) are treated as a special case. The approach taken is similar to that of WebFormator. To aid the user in locating such elements, each link starts on a new line in its fully expanded state. The word *link* is inserted before the link text, to notify the user of its

³This is dependent on authors creating headers that reflect the content of the document section to which they belong.

```

[-] h1    WebTree Application User Manual
o. [Link Introduction]
o. [Link User Interface]
o. [Link Customisation]
o. [Link Forms Interface]
o. [Link Table Navigation]
o. [Link Searching]
o. [Link Document Retrieval and Browser History]
o. [Link Index of Keystrokes and commands]

[-] h3    Contents
[+] ul
Introduction
[+] p    Many Internet access solutions for the blind are in reality ...
[+] p    The WebTree application has been written to explore the use ...
[+] p    As well as the tree like arrangement of the displayed XHTML ...
[Link Back to top]

```

Figure 5.4: A simple web page as viewed through WebTree, with many of the high level tree controls removed.

type.⁴ However, these items could also be styled using speech parameters prescribed by an aural CSS (see section 4.6). Figure 5.5 shows how links are displayed through WebTree. Figure 5.6 shows how the same content appears in WebFormator.⁵

It was decided to place links on separate lines to other content so that the user could quickly move through the text to find the required link. The importance of this is most evident in cases where a number of links are contained within the same block of text. When listening to a constant stream of material, it can be difficult to tell when one link begins and ends. This can also occur even in the case where links are read in an alternative voice to the main text. Consider the example of three links positioned on the same line. The user must listen to the entire content of the first two links before hearing the third link. Whereas moving line by line, the user can quickly scan through the links to find the one they want. This is because the user does not always need to listen to the entire link text before establishing if it

⁴No tree controls are included in the auditory rendering for this element.

⁵In figure 5.5, the content is a mixture of hyperlinks to publications and the dates in which they were published. The date associated with each publication is positioned on the line following the link text.

is the one they require. It could be argued that this additional formatting is unnecessary; for moving from link to link can be achieved by invoking the next or previous link functions. However, when navigating in this manner, any contextual information provided by additional text included between the links is lost.

It is important to note that WebTree has the facility to restrict the insertion of line breaks to only appear before *block* elements in the audio rendering (an exception is made for hyperlinks, see previous paragraph). This facilitates the functionality to read through the content element-by-element. However, this reading strategy is only effective if an element doesn't contain any children that require tree controls, in which case, these elements are treated as starting a new block of text. However, the user can also choose to have lengthy streams of text presented over a number of lines, by setting the `wt-custom-line-length` variable in the customisation buffer. This variable determines the position in which to insert a line break in the content. When this variable is set to a number less than the length of an element's enclosed text, then the content is wrapped accordingly and the reading functionality mimics line-by-line interaction.

In the prototype application there is currently no method for examining the different attribute components of a hyperlink. Not only should the user be able to select what is read when a link element is encountered, e.g., the "link text" or the content of a `title` attribute, a mechanism to examine its list of attributes might also be valuable in many circumstances. Furthermore, this could be beneficial for a number of different elements. Thus, a generic method to examine the attributes for an element ought to be provided. One such method would be to allow access to this information under the tree viewing method by treating attributes as children of the element. The problem with this approach is that the automatic inclusion of these elements in the audio rendering could clutter the interface making it difficult to use. A second approach would be to allow the user have this information conveyed to them on request. This could be done with a

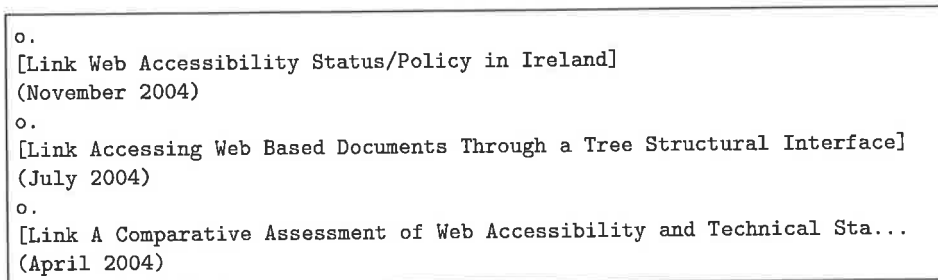


Figure 5.5: How hyperlinks are displayed in WebTree

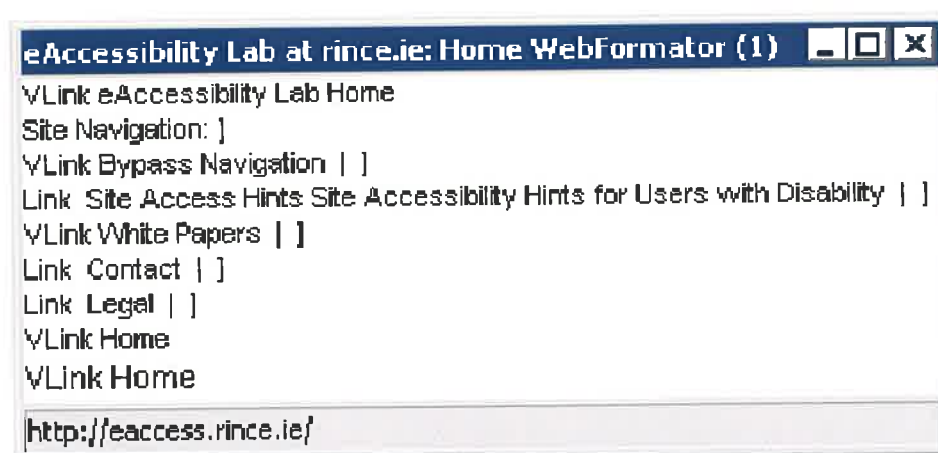


Figure 5.6: How hyperlinks are displayed in WebFormator

function that reads aloud the relevant material, or through adding a second button to the tree control information that if pressed will present the data in the primary tree view rendering.

The portrayal of type information for inline elements is a complex issue. This is due to these elements appearing automatically expanded in the screen text, without any tree controls. To solve this issue, a trade off is necessary between adding to the complexity of the audio stream and increasing its verbosity. These elements can be signalled by introducing additional audio cues, such as changes in voice or adding non-speech sounds which add to the complexity, see section 3.2 for more details. The second option is to increase the verbosity of the auditory stream by announcing the element name through speech cues. Under this system, the classification of inline

element types can be determined through two different methods. WebTree allows the user to assign different voice characteristics to specific elements through the use of aural style sheets. However, as discussed in section 3.2, there is evidence to suggest the number of different speakers used in a spoken presentation should be restricted to a small number. James (1998) reasons that using different voices to mark elements is analogous to marking the same components with changes in colour. When only a small number of items are marked, the user can easily recognise that the element in question has an additional contextual meaning. However, in documents where many elements have different colours, the document may just look colourful and unmarked. Therefore, in the WebTree system, the number of alternate voices and the elements to which they are associated, are allocated by the user.

The second method announces the name of the most localised element under point. However, this information is not spoken automatically. Instead, the user must request this data through the pressing of a single keystroke. Unlike the case of hyperlinks, this information is transiently announced in the audio stream, and not physically added to the text in the virtual screen rendering. Automatically announcing this information when an element is encountered would vastly increase the verbosity of the output. However, during the test process, it was evident that sometimes automatically announcing element names as they were encountered was beneficial. This was the case for header `<h*>` elements, which provide a logical structural division on the content. If no tree controls were included for the element it was difficult to tell that a change in context had occurred. Although the user could check the element under point, there was nothing to make the element stand out from other inline text elements. However, if voice changes or auditory cues were used to signal these elements, adding this spoken cue might be unnecessary. It would depend on whether the user was familiar enough with the system to determine the meaning of the cue.

5.1.2 Customisation

WebTree currently relies on the emacs specific `customize` functionality to take into account user preferences. However, a number of usability issues with this system were observed during the user testing process. See section 6.4.4 for more details.

After taking into account the suggestions made by the user group, it is recommended that subsequent versions of the WebTree application should no longer require this `customize` component. Instead, customisation should be provided through an XHTML form based system. Due to the proposed interface for the customisation facility closely mimicking the functionality of a web based document, it should be possible to view it through the normal document rendering, thus removing the additional learning curve associated with using a separate `customize` feature. Many of the required element display options settings could be handled through the use of multiple select check lists (`<select>` elements). Text edit fields could be used to set custom variables requiring values such as numbers or plain text, e.g., the variable governing line length.

For the most part, the customisation functionality of WebTree deals with how the content is to be displayed/rendered. Using this facility, the user can dynamically generate alternative views of the current document. This is achieved through the exclusion of specific elements from the audio rendering. For example, a view containing only paragraph elements `<p>`, or header elements `<h*>` is simple to generate.⁶ Where users already possess some previous knowledge of the page structure, generating such views should increase the efficiency in locating the required information. See section 5.1.3 for a more detailed description of how WebTree generates alternative document views.

Each of the elements with tree control status, (predominantly XHTML block elements), have two associated entries in the customisation buffer. The

⁶To generate alternative views in this manner, any higher level ancestor elements for the selected elements must be displayed. However, these can appear without their tree controls, to give the illusion that they have been excluded from the rendering.

first determines whether a tree control is inserted into the virtual display once the item corresponding to the element type is detected by the display functions. The second option resolves whether the content of the specific element is to be automatically expanded into the audio rendering. If not set, the tree component remains in its collapsed state; otherwise the tree control is expanded to show the element's children. By altering these settings, the user controls the amount of content to be automatically portrayed when a page is loaded.⁷ In cases where the user elects to always expand a given element type, the user can reduce the amount of clutter on the page by removing the associated tree control.⁸

WebTree does not currently extend similar customisation functionality to the appearance of *inline* XHTML elements. However, to enable the rendering of alternative views, the ability to eliminate specific inline elements from the display is generally necessary. Therefore, this functionality should be included in future versions of the system. There are no plans to include tree controls for each *inline* element. The appearance of these components for such elements would detract from the usability of the display. This is because their presence would affect the readability of the material. For example, if the user must expand each `` element to read the enclosed content, the user may become distracted from their train of thought. Similarly, if the element is already expanded the added verbosity caused by the tree controls may have the same effect. Through the use of aural CSS, the presence of these elements can be conveyed using alternative means. For example, through changes in voice characteristics, such as "pitch" or "stress".

⁷After changes are made to the customisation buffer, the current page must be refreshed before the changes are implemented.

⁸Removing this contextual information results in the user needing to rely on other means of obtaining the element type information.

5.1.3 Generating Alternative Document Views

Many web browsing solutions for the blind offer the ability to create alternative summary views of a document's content. The provision of this functionality is important due to the serial nature of the modality. As previously mentioned, only a narrow focus on the material is provided by this method of interaction. Therefore, it is difficult to quickly scan through the information without some assistance from the application. Section 3.5 discusses many of the different summarisation methods employed by auditory solutions to achieve this goal. These include summaries based on analysis of the textual content in the document. For example, views created to reflect paragraphs containing specified words or phrases. A second approach involves generating web page summaries based on specific mark-up elements. For example, producing lists of header `<h*>` elements, or an index of all the hyperlinks on the current page. However, the mark-up based summarisation features are usually based on a hard coded set of elements chosen by the product developers. Thus, there is usually no generic mechanism in which the user can initiate views based on their own preferred element selections.

In a sense, the intrinsic expand/collapse functionality of WebTree is all about manipulating *alternative* custom document views. In this manner, the user can create alternate renderings of a document to allow them access the required information. However, there is a need for more powerful summarisation functionality to complement the expand/collapse methodology.

The different views generated by the WebTree system rely on the mark-up elements used in the document. However, there are no limits imposed on the type of element used to generate these views. Instead, the user can generate renderings based on any element/groups of elements available in the web page. The generation of these element abstractions is achievable through two distinct methods. First, the user can manually select elements using the *customisation* facility to be included in the view, by only setting the display preferences for the relevant elements.

Although different views may be generated by manually setting variables

in the customisation mechanism, always requiring this facility to dynamically create alternative document views is an unrealistic approach. Nevertheless, this feature plays a major role in the generation of alternative content renderings. For example, consider the situation where the user is interacting with a number of documents structured in a similar fashion. Formulating web page renderings using this method can provide an elegant solution. However, it should be remembered that the optimal document rendering may differ greatly depending on the document mark-up structure and the type of content portrayed. Also, the reading strategy favoured by the reader greatly influences the optimum structure of the audio rendering.

When skimming the document, it may often be sensible to produce views containing paragraph elements unexpanded in the text. That is, once the user chooses to include some of the paragraph text in the audio rendering. It may also be beneficial to have tables and list elements presented in their collapsed state, allowing the user to quickly navigate past these elements. Whereas, for a more in-depth reading of the content, it might be beneficial to have one or more of these element types automatically expanded in the default document rendering.

The *customisation* facility functions by assigning values to a number of display variables. Using this framework, providing functionality to generate views based on a list of user prescribed elements is a simple task. For example, `<h*>` or `htaga` elements. However, due to the iterative nature of the tree construction functions, problems may occur if a specified element is nested within an unlisted element sub tree. Therefore, to ensure a workable solution when rendering alternative views, the established customisation features must be accompanied by two additional settings. The first setting ensures the entire hierarchical tree for the selected element is expanded. The second setting excludes all other content from the named element's ancestor hierarchy. In this manner, the regular tree expansion functionality operates as normal whilst having the capacity to exclude all but the specified elements.

There is a case for allowing the user to *save* named collections of customisation settings, so that they can be quickly reloaded as needed. For example, a set of customisation settings optimum for a specific website, or a list containing the elements the user wishes to view. This is akin to having a number of different local CSS files that might be loaded for different purposes.

5.1.4 Keyboard Navigation and Interaction

As stated previously, the primary mechanism for navigating through the hybrid content rendering of tree structural components and plain text content is through the use of the arrow keys. Moving through the text with the left or right arrows, the reader can examine the content on a character by character basis. Similarly, the up and down arrow keys facilitate navigation to the previous/next line in the virtual view respectively. Using the control key in conjunction with the left and right arrow keys allows for movement word by word. As each of these distinct navigation features is employed, the grouping of characters traversed by the application is automatically spoken.

One of the principal design goals in the WebTree system was to keep the user interface as simple as possible. Thus, where similar interactions were required to perform common tasks across the interface, the same keystroke was used to alter the display. For example, the `enter` key is the main key used to dynamically change the state of the document rendering. This is the key chosen to alter the state of buttons, whether they are tree control buttons, checkboxes or radio buttons. However, the different types of search functionality also have a major affect on the presented content. Once an instance of the search string is found, the element containing the string is automatically expanded into the on screen document view.

In instances where commands are activated by a single keystroke, e.g., performing an element search, or moving to the next tree expansion control, the uppercase version of the same character is used to reverse the action. This is done to reduce the number of keystrokes the user has to learn. For

example, `e` searches forward for an element, and `E` performs the same type of search in reverse. Similarly, `t` searches forward for the next tree control, and `T` reverses the search. The emacspeak specific `emacspeak-learn-mode` command can also be used to check the function bound to individual keys. Once this mode has been entered, pressing a key will result in the name of the function to which it is bound being announced.

For a full list of keystrokes and the commands they invoke, see appendix A.

5.1.5 Searching

Under the WebTree system, the browsing of documents can be performed through a number of distinct methodologies. Moving linearly through each level of the tree, expanding elements when necessary, was one navigation method observed in the user evaluation process. However, an alternative method is to invoke one of the different search functions provided by WebTree to relocate focus to another part of the document. There are three different types of generic search functionality available to the user, as described in the following subsections.

Incremental Text Search

An incremental text search is the primary form of search functionality provided by the system. Once invoked, the application examines the on screen text from point until the next tree control is found. If the defined search string is discovered then focus is positioned on its concluding character in the virtual view. Otherwise, the document's DOM tree is scanned using a recursive depth first search algorithm in an attempt to locate the string. When the search string is found, the tree element containing the text is expanded into the virtual screen rendering and focus is altered accordingly. As each key is pressed by the user, the resulting character is appended to the search string and the process is repeated. If the current length of the search string contains more than one character, the position of point is moved the same

number of letters in the opposite direction before the search begins. This is necessary to ensure that the previously found part of the search string is included in the new search. This search algorithm is used regardless of the search direction. In the case of a backward search the focus is placed on the left most character of the found search string. Conversely, focus is positioned on the right most character when searching forwards.

If the incremental search function is invoked whilst a search is already in operation, a new search is performed using the current search string. In this case the search starts from point without moving the requisite distance in the opposite direction to coincide with the length of the search string, thus effectively searching for the *next* occurrence.⁹

It is necessary to first search the on-screen text for the required string, due to difficulties linking specific characters in the audio rendering with their equivalent positions in the DOM structure. The existing linkages are from character to screen widget to DOM node. However, there is no linkage from individual characters in the audio rendering to individual characters within the linked node. Problems can occur if more than one instance of the search string is contained in the same node. It would be difficult to assess which one is the correct string, due to the lack of knowledge concerning the equivalent position in the DOM relating to the starting point of the search. Thus searching the screen text before the next unexpanded tree node removes any ambiguity. The parent element of the tree control then serves as the starting node for the DOM search. Also, searching plain text takes less time than iteratively searching the DOM, so it is best to check if the required content is already present in the audio rendering before a full DOM search.

Mark-up Element Search

The second form of search allows the user to search for, and jump to, the location of any XHTML element in the document. A number of current

⁹If the incremental search command is called twice without any text being entered, then the previously entered search string is used by default.

web access solutions, such as JAWS for Windows¹⁰ and IBM's home page reader (HPR)¹¹, provide additional navigational functionality based on an ad hoc set of mark-up tags. For example, jumping to the next/previous table construct or list entity. This functionality is dependent on a small list of tags set out by the application developers. There is no leeway to allow the user navigate documents based on elements not on this list. Thus, WebTree affords the user a more powerful generic element search function via which navigational access to all mark-up element structures in a document is available.¹²

The element search functionality can be invoked in two different ways. The first is through an interactive call from the keyboard. In this case, the user is requested to enter the name of the element to search for. The second method requires a call by another function where the element name is passed as a parameter string. To achieve its goals, the function must examine the screen text overlays between the current position and the next tree control to see if the required element is already present. If this is the case, focus is moved to the starting point of the given element. In cases where the element is not found, WebTree performs a recursive depth first search on the DOM tree structure to find this element type. This search starts with the DOM node associated with the next tree control's parent widget. If found the required element sub tree containing the element is expanded into the virtual display and focus is moved accordingly. This procedure is adhered to regardless of the search direction.

The specialised element search forms the basis for much of the additional navigational functionality provided by WebTree. For example, moving to the next/previous hyperlink in the document. Although using the generic element search for these purposes would bring the user to the requested element, additional element specific navigation functions were included to aid power users. These methods call the generic element search function and

¹⁰http://www.freedomscientific.com/fs_products/JAWS.HQ.asp

¹¹http://www-3.ibm.com/able/solution_offerings/hpr.html

¹²The element search is a static string search.

pass the name of the required element as a parameter. Using this framework, additional navigational features based on jumps to specific element constructs can easily be derived.

Combined Element and Incremental Text Search

The third and final search type is an amalgamation of the previous two types of search functionality. That is, WebTree allows the user to specify an element type to which an incremental text search is to be restricted. This is achieved by setting the variable `wt-limit-search-element` through the `wt-set-search-restriction` function with a string containing the relevant element name. After this variable has been set, the regular incremental search functionality may be called as normal. Once this variable has a non-nil value, the search procedure only analyzes content found within the named element and its sub trees. That is, during the screen text search, if the required search string is found, then the application must examine the text overlays under point to establish whether the found instance exists within the named element. If this is the case, then focus is moved accordingly. Otherwise the application continues with the search. If not found, then a recursive depth first search for the relevant element is performed on the DOM tree structure. It starts with the DOM node associated with the next tree control's parent widget. To revert to the regular incremental text search functionality, the restriction must be unset using the `wt-clear-search-restriction` function.

5.1.6 Rendering Tabular Data

Although the mark-up describing the `<table>` construct is tree-like in nature, it poses serious problems for a purely tree-like interface. The optimum methods for presenting and assimilating tabular information relies heavily on its two dimensional (graph as opposed to tree) like organisation. Aside from tree related issues, tabular data poses many problems for speech interaction in general. This is due to the limited view imposed by the serial

nature of the modality. It is difficult to build a mental model of how the cells relate to one another. Thus, research into the best methods for portraying such material through the spoken medium has resulted in many diverse access strategies. For a review of the literature concerning the presentation of tabular structures through purely auditory means, see section 3.6.

In terms of the WebTree application, the individual table cells are physically portrayed in the linearised fashion in which they appear in the mark-up. Each cell is presented automatically expanded with no tree controls included by default.¹³ The main reason for not including the tree controls for such elements is that they would significantly increase the verbosity of the content. If the reader encounters the control each time a new cell is visited, having to always listen to the control information before the cell content, may impinge on the speed of table interaction. This is especially problematic if the content of each cell is quite brief.

The linear presentation of the tabular content should assist in cases where the author decided to use tables to structure the page layout. However, it should be remembered that the work described here is primarily intended to deal with content stored in data tables. The WCAG 1.0 guidelines discourage the use of tables to spatially organise content. Instead they promote the use of cascading style sheets for this (W3C, 1999b):

- 3.3 Use style sheets to control layout and presentation. [Priority 2]
- 5.3 Do not use tables for layout unless the table makes sense when linearised. Otherwise, if the table does not make sense, provide an alternative equivalent. [Priority 2]

Additional navigation functionality is provided to ensure easy movement along the two-dimensional relationships between the individual table cells. These movements include navigating to the cells to the immediate left or

¹³The inclusion of tree controls, and whether the `<th>` and `<td>` elements are automatically expanded can be reconfigured through the `customize` facility.

right of the current cell in a given row, and up or down to the next cell in the relevant column. If the current cell spans more than one row, then navigating to cells to the left or right of the element is only permitted along the first row in which the cell appears. Column navigation is not affected in this situation. Similarly, in the case where the current cell spans more than one column, navigation away from the cell is only permitted along the first column spanned by the cell. In this situation, row navigation behaves as normal. The benefit of navigating such cells in this manner is that the application operates in a predictable fashion. However, there are some disadvantages to this approach. The user may get lost when navigating from cell to cell in a row if the focus is relocated to a cell in a different row, and not returned to the existing row once that cell is exited. A similar problem occurs when reading cells that span more than one column. If navigation does not continue along the same column as before the multiple column cell is encountered, then the user may become disorientated and lost. It is important for the navigation functions to move to the cell when the row /column being navigated is not the first row/column in which the cell appears. The reason for this is the insertion of empty cell place holders in the grid to show the overlap between columns/rows can also be confusing. The ability to enunciate row/column headers and/or numbers, on demand, could be one solution to help users stay oriented when navigating tabular constructs. Of course, the very meaning of row/column numbers is a bit ambiguous when some cells span multiple rows or columns. For this reason, it may be beneficial to have the content of such spanning cells duplicated in each row in which they appear. An automatic cue such as an auditory icon could be used to identify when cells have this spanning property. When you take into account the possibility of having certain `<tr>` elements in a collapsed state, this is a very complex problem, which requires further investigation.

One approach to solving this problem would mean the table navigation functions expanding the different elements when required. However, this may be one situation where it is beneficial to move away from the strict

expandable tree model in favour of a pre-set display format. The expansion collapse functionality adds much complexity to the navigation of these constructs. Therefore, for the purposes of the prototype application, all `<tr>` elements are automatically expanded once the `<table>` element is expanded.

As navigation is performed from cell to cell along a specific row or column, the header information for the current cell is not automatically spoken by the application. Instead, the option is available for the user to simply navigate to the header cell associated with the current cell. Navigation to both the row and column header cells if they are present is possible. Once the user has read the related header information, the user can call a specific table back function to revert focus to the cell prior to the original jump. Although a table header jump could be seen as being equivalent to an internal page link, it was decided to separate the table *back* functionality from the generic *back* function for the following reasons. Consider the situation where the user is browsing a table with multiple logical headers for a given cell. The user may choose to navigate to each header in succession without pressing the *back* button. They might then choose to move away from the header cell to other parts of the table without reverting to the original data cell. In this manner a large list of jumps that can be undone are added to the table back list. If the table *back* function and the Generic *back* function were integrated, the user would have to undo each one of the jumps before undoing a regular internal page link. Therefore, due to the possibility of having multiple internal table header jumps, it was thought to be more beneficial to separate the functionality out into two different methods.

The use of higher-level table grouping elements i.e., `<thead>`, `<tbody>` or `<tfoot>`, available in the full XHTML specification, increases the complexity of the specialised table navigation functionality when viewed under an expandable tree interface. This additional level of abstraction might be beneficial when viewing tabular material in a linear form because it imposes a strict organisation on the data. However, the main benefits concerning the inclusion of such elements occur when applied to the visual rendering of con-

tent. They provide a strict mechanism to handle the displaying of header and footer information. This is advantageous in the case of large tables spanning multiple pages. Displaying both header and footer information on each page may be beneficial when presenting data in the traditional page by page basis. This segregation into pages is important for visual interaction; however, there is no need for such organisation under an auditory interface. Therefore WebTree does not split such content into individual pages. Under this system it would be difficult to consistently determine accurate virtual page segments. This is due to user control over the expansion of elements. Also, as previously mentioned, the size of the document segment viewable at any given time through auditory means is quite small. This ensures the formatting of this type of content as a set of pages including header and footer information is a redundant exercise. This is because the spatial arrangement of content is lost through this type of interaction. It is expected that the ability to jump to and from the relevant header cells should negate any problems with navigating large table constructs.

WebTree considers these elements analogous to any other XHTML block designated element. That is, they appear in the virtual display in the linear form in which they are encountered in the mark-up. However, much additional processing is required by the application to ensure accurate linkage between the header information and the relevant cells in the body of the table. Not only is there a requirement to link the individual cells with cells in both the next/previous row in the table, but there must be an additional link between the first and last rows positioned within these higher-level group elements and those of their counterparts. When WebTree is generating the grid connections between the individual table cells, the first row contained within the `<tbody>` element is connected to the final `<tr>` element of the `<thead>` construct. Similar connections are made between the last row of the `<tbody>` element and the first row of the `<tfoot>` component. The customisation variables governing the display of these elements are set by default to automatically expand their content without the presence of

Keystroke	Navigation Command
1	Go to end of table
2	Column down
3	Move to row header
4	Row left
5	Read content under point
6	Row right
7	go to beginning of table
8	Column up
9	Move to column header
0	go to previous cell from Header jump

Table 5.1: Table of keystrokes and the table navigation commands they invoke

tree-controls, thus, ensuring that the reader experiences a table construct without this additional complexity.

The table navigation keystrokes are linked to the keys representing the numbers zero to nine. The reason for this is so the numeric keypad could be used to provide easy access to the table commands. Also, associating these commands with the number keys allows for the same keymap to be used with a laptop keyboard layout. Table 5.1 contains the list of keystrokes and the table navigation commands they invoke.

5.1.7 XHTML Form Handling

Handling the interaction and presentation of XHTML `<form>` elements poses a number of difficulties when using a tree-based interface. Form controls contained within additional structural elements, e.g., `<p>` tags, can pose problems for this approach. Under WebTree, once a form element has been expanded into the audio rendering, its presentation coincides with the linear organisation of its lower level elements in the mark-up. All field/controls based on the `<input>` tag appear already expanded in the tree structure without any tree controls when presented at the current level of tree abstraction. However, when controls are placed inside additional structural tags, these structural elements must be expanded either manually by the

user, or automatically through the *customisation* facility, to expose the controls in the rendering.

The content of option lists, (based on the `<select>` tag), is not automatically expanded into the screen text by default. Instead, their expansion may be achieved in the same manner as for any other tree control element. These elements, when viewed in their collapsed state differ from regular tree elements only in the type of information displayed as part of their tree control. The regular button and element name components are present, however, a third text item showing the current selected value of the list is also included. Once expanded, the individual options are presented on separate lines to aid the selection process. Each option contains two components. Namely, a checkbox and a text item showing the option value. To select an option, the user must check the checkbox associated with the item. If the element in question is not a *multiple* select list, the application automatically unchecks selected items as a new value is selected.¹⁴

When filling out forms with the WebTree application, there is no additional keystroke to enter a forms mode before access to the fields is granted for editing purposes. Instead, as the focus is placed on a form edit field, the application automatically alters the mapping of keystrokes from their WebTree settings back to the normal character insert function so that editing can take place. However, once focus is moved away from the form field/control through the use of the arrow keys, the key mappings revert to point to the WebTree specific keystroke mappings. Knowledge of the type of keymap currently in use can be obtained by knowing which element type is currently under point.¹⁵

Each form field/control is positioned on a separate line from any text contained within the form construct. It should be remembered that the start of a form field usually coincides with the beginning of the line on which it

¹⁴The state of checkboxes and radio buttons may be altered by placing focus on the button and pressing the `enter` key.

¹⁵The keymap does not change for checkbox, checklists or radio button controls, for as with tree control buttons, the `enter` key is the only key necessary to alter their state.

appears. As focus is placed on the form field/control, the user is notified of its presence through speech output. When tabbing to the individual form fields, any label information to be associated with the element is also read by the application. This is the case whether or not the label element is expanded in the display. Those functions responsible for providing additional information about these elements examine the DOM for any associations with `<label>` elements, as opposed to querying the on screen information.

Figure 5.7 shows a form construct as presented by the WebTree system. Although radio buttons are represented by () and the checkbox by [] in figure 5.7, these are not spoken by the system unless the user is navigating character by character through the content.¹⁶ Figure 5.8 shows the same form as rendered visually in Internet Explorer.

5.2 Auditory Output

In the previous section the different types of functionality making up the WebTree user interface were discussed at length. However, focus will now centre on how the relevant information can be best portrayed to the user. The primary question under analysis in this segment of the document is to determine whether contextual information should always be spoken or whether alternative methods to impart such data should be sought? This material can be presented using three distinct methodologies. These include: adding the contextual information as a spoken fragment of the audio stream; using alternative voice characteristics to signal various contextual cues; playing non-speech sounds to notify users of changes in status. See section 3.2 for more details.

It is not always appropriate to provide additional contextual information in the audio stream. Providing large amounts of extra information may serve only to clutter the presentation, and in doing so, ensure that the

¹⁶Each option in the combo box (`<select>`) element has a checkbox associated with the option. This is to allow the user select the required element in a consistent manner across the entire interface. However, it may be beneficial to rename this component to clearly signify that it is just one component of a list.

```

[-] form
FirstName:
:edit
Second Name:
:edit
student
radio button checked(*)
PAYE worker
radio button not checked( )
self Employed
radio button not checked( )
screen Reader User
checkbox checked[X] What is the primary method you use to interact with
your computer?
[-] Select A combination of braille and speech
checkbox not checked [ ]Through a visual display
checkbox not checked [ ]Braille output
checkbox not checked [ ]Synthetic speech
checkbox checked [X]A combination of braille and speech
[Reset]
[search]

```

Figure 5.7: A Simple XHTML <form> construct as viewed through WebTree

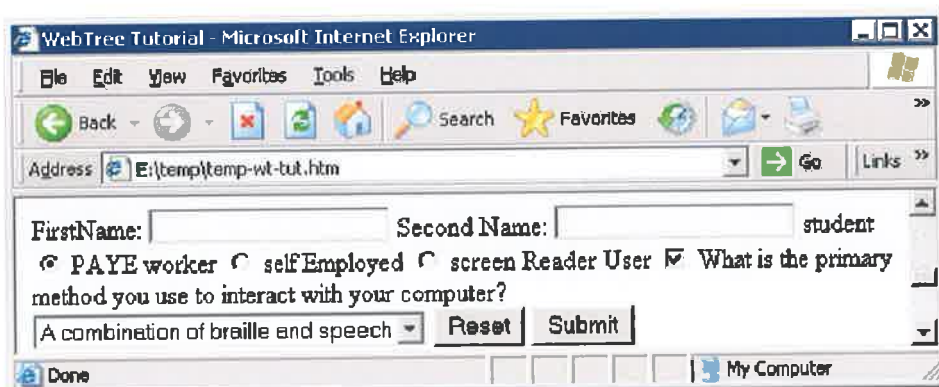


Figure 5.8: A Simple <form> construct viewed through IE

material being presented is difficult to understand. Therefore, a balancing act must be performed to determine whether the extra information should be automatically portrayed. Care must be taken not to overload the user with too much information. Processing this extra data may affect their ability to assimilate the content. Also, the effect that these additional cues have on the user's ability to comprehend the information can be influenced by the users preferences. Some people prefer to have large amounts of information spoken by the application. Others prefer to limit the amount of material to be automatically announced. For this reason it is important to allow the user to customise how much information is automatically presented. In addition, it is often beneficial to allow the user gain access to the information through other means when it is required.

Although the use of a number of sound cues in the WebTree system is proposed, the predominant mechanism used to produce auditory feedback is through synthetic speech. The sound cues used by the application will mainly convey task status and/or task completion information. For example, to signify that a web page has finished loading or that an element search has been completed. However, the main bulk of the information presented through the WebTree system is achieved through speech output. This includes a combination of actual web page content and where relevant, some additional contextual information is also announced.

Under WebTree, contextual information is presented in the spoken output using a number of alternative methods. The first method just includes the relevant contextual data in the audio stream. The prime example where this is advantageous in the WebTree interface is in the viewing of tree control information. The name of the element is included to notify the user of its presence. Whilst a segment of information about the element, e.g., the content of its `title` attribute, or in the case of `<p>` elements, some of the enclosed text, is often also rendered. It is necessary to include this information, so that the user can make an informed decision on whether to expand the element, or whether they will need to look elsewhere for the required

data.

Alternatively, the type of element can be signalled through employing additional voice types to read their content, or by manipulating prosodic characteristics of the vocal output e.g., pitch or stress, which govern how the material is to be spoken. These changes in the spoken output can be used to signify contextual variations in the text. However, this approach requires a certain amount of learning on behalf of the user to dynamically map the voice changes to the type of content they represent. Nevertheless, as discussed in section 3.2, the number of alternative voices should remain low to obtain favourable results in element type recognition. If a large number of voices are used, the user must spend some time attempting to decipher the implied meaning of each voice. The more voices employed by the system the greater the chance of making errors in determining these associations. Also, if the reading voice is constantly changing, it may detract from the information it is attempting to communicate.

In reference to the previous example of the tree control, using the second method for signalling contextual information should remove the need to always include the name of the element in the document rendering. That is, once the user is confident in their ability to distinguish the different voice to element mappings. However, it is not thought that this approach would suit the display of all XHTML elements. Instead, this concept would be best suited to the presentation of elements of high contextual importance, e.g., `<h*>` elements. Or, alternatively those elements necessary to provide navigation, for example, hyperlinks (`<a>` elements). Signalling elements in this manner may mean that additional label information about the element may be unnecessary for automatically expanded elements.

Applying alternative voices to signal the presence of inline elements such as `<emph>` or ``, is not thought to be a viable solution. If an unexpected change in voice type occurs when listening to a constant stream of speech, the listener can become distracted by the actual voice change. This can result in reduced concentration levels when assimilating the content

(James, 1998). For example, consider the case where a `` element appears in the middle of a sentence. The initial segment would be read as normal, the `` component would be announced in an alternative voice, with the final part of the sentence spoken in the original voice. However Fitzpatrick (1999) conjectured that applying changes in prosodic characteristics e.g., changes in the stress levels of the default voice could be used to signal such elements. The prosodic alterations should be powerful enough to notify the user of a change in context, whilst not distracting them from their current task. Although the type of element under point may not be directly evident using this type of spoken cue, the knowledge that a change in context has taken place may just be enough.

The different methods for signalling contextual information through speech may be included in the WebTree user interface through the use of an oral style sheet. However, it should be remembered that its usage is optional. See the section 4.6 for more details.

As stated previously, it is sometimes preferable to have no contextual information spoken by the application; or at least not unless the user specifically requests such data. For example, in the case of table navigation, the reader may not wish to have the current cell's row and column number read automatically. Whereas there may be occasions where knowledge of such data would be beneficial. Thus, the final type of context providing functionality offered by WebTree allows the user to establish their position in the document hierarchy by examining the elements under point.

It should be remembered that different situations require different forms of interaction. Thus, no one form of functionality described above is applicable in every circumstance.

5.2.1 Tree Interaction

To effectively interact with tree structured data, the amount of contextual information required to show how the current position fits into the overall structure must be examined. There is a trade off between providing

enough information so the user can navigate successfully without becoming disorientated, and keeping the verbosity of the speech stream as low as possible. When determining the type of information to be portrayed, a number of additional questions must also be looked at. Should information concerning the current tree level be announced every time a change in level occurs during navigation? Or should alternative mechanisms for obtaining this information be provided? In addition, how much information should be provided about the element being expanded? For example, should the number and/or type of nodes encapsulated within the element be automatically announced? For a review of the literature concerning tree structural presentations, see section 3.8.

Operating the element expansion/collapse functionality needs little spoken assistance. Once the task has been completed, the application proceeds by reading the current line, including the button's status and element name, whilst focus remains on the tree control button. The status of an element can be determined by the value of the button text, positioned at the beginning of each tree control line. That is, [+] for collapsed and [-] for expanded elements.

An expanded element's child components are placed directly beneath its associated tree control and above the content of subsequent elements. The user can often establish the list of child elements once they know which element is *next* at the corresponding tree level. This is achieved by navigating through the content until the higher-level element in question is located. However, this method of recognising and negotiating tree levels can be rather ambiguous. This is due partially to the large number of elements possible at lower levels of the tree, and to the likelihood of multiple levels of element nesting. The facility to allow certain named elements to appear automatically expanded in the text also contributes to this ambiguity. Therefore, an additional mechanism for unobtrusively portraying tree level information was sought.

Automatically speaking the level of each element, or announcing that a

change in the current tree level has taken place, was quickly ruled out. The reason for this is the large increase in the verbosity of the audio stream that this would cause. Presenting contextual information in this manner would offset many of the advantages that the WebTree system has over a linear approach to document presentation. The main aim of the WebTree system is to facilitate easy navigation to important segments of the document. By doing so, saving the reader from having to unnecessarily read through large chunks of material not immediately relevant. Thus, it was thought that always speaking tree levels and/or the number of nodes present at each tree level would reduce the rate at which the user can assimilate the content.

Currently, the user is responsible for requesting tree context information, before it can be spoken by WebTree. However, when announcing this data, the application does not state the actual nested level of the tree element. This is because it is debatable as to how valuable this information would be in a document reading interface. For example, stating that focus is positioned on level seven in the tree does not provide the user with any information about its surrounding elements. Instead, providing contextual information in terms of element relationships should be more beneficial. Therefore, WebTree presents the user with the name of the parent element encapsulating the current content. For this system to operate correctly, access must be also given to the list of ancestors of the current element. Through a combination of examining the element under point, and establishing the relationships between itself and its ancestor elements in the tree hierarchy, it is anticipated that the user will be able to glean an accurate mental model relating to the current position in the document structure.

As part of future work, an experiment to assess whether non-speech sound cues might be exploited to signal tree level changes should be investigated. There are many ways in which sound could be used to achieve this goal. For example, one approach might be to use notes on a musical scale to denote movement through the different levels of the tree. As the user navigates further down the tree hierarchy, the musical sound produced, moves in

sequence along the scale to reflect the corresponding level position. Another possibility would be to rely on two distinct sound cues to provide contextual information. The first cue signifies that the user has moved to a lower rung on the tree structure, while the second indicates that a higher-level element is now in focus. It would be interesting to observe whether such information is enough to accurately navigate through a tree hierarchy, or is knowledge of the current tree level necessary to navigate through the content successfully. Either way, this functionality is not expected to replace the current methods for obtaining tree positional information. In fact, the main goal of the experiment would be to determine whether the additional use of sound complements the current approach. If the user is not sure of the position from the non-speech sounds alone, they can also query the ancestor elements of the element under point.

5.2.2 Speaking Form Data

When interacting with content presented within XHTML `<form>` constructs, the delivery of contextual information is extremely important. This is necessary to help the user orientate through the different fields/controls. It is essential to announce the presence and the current state/value of the field/control when it is encountered. However, in many situations, there is not always enough information to rely on. Sometimes the true function of the element is not obvious without examining the surrounding text. In the XHTML specification there is a facility to programmatically link the field/control to a `<label>` element. In this way, contextual information is explicitly assigned to the form control. Auditory browsers can take advantage of this linkage to provide some contextual information about the element. Otherwise, some heuristics must be employed to determine the context in which the element appears. See section 3.7 for information about form handling and the problems faced by screen reader users.

As a form field/control is encountered when linearly navigating with the arrow keys, its presence plus its current value must be signified to the user.

There is a trade off between announcing contextual information regarding the control's type and/or status, and ensuring the output is as succinct as possible. Speaking the full name of the element type, coupled with its current value is not always necessary. Therefore, in the case of text entry fields, e.g., a single or multi line field, the application simply announces the first line of its current value, followed by : `edit` to show that it can be edited. When more than one line of content appears in an edit field, the word : `edit` is announced after the content of each line is read. The major difference between the two different field types is down to the expected amount of content to be inserted. If the user presses the `enter` key whilst in a multi line text field, a line break is inserted into the field's content, and the user may continue entering data. Whereas, with a single line text box, such an action will result in focus being removed from the form field. In the case of a *password* field, the phrase `password: edit` is announced without any reference to its present value.

In the case of radio buttons and checkboxes, the full name of the element followed by its value, `checked` or `unchecked` is announced whilst navigating in this manner. Due to the `<select>` element being treated as a regular tree control element, no specialised contextual information is included. However, the presentation of its individual `<option>` elements require some additional information. Initially, the checkbox component and its status is spoken followed by the text of the `<option>` element.

There exist many situations where the functionality of a form field/control is obvious from reading the surrounding text. Therefore, when linearly navigating the content, any explicitly associated label information is not automatically spoken when focus is placed on a given element. However, there exists many cases where the control's contextual meaning is not easily deciphered. In these circumstances, an explicit linkage between the `<input>` element and a `<label>` element is necessary to determine its functionality. Users may wish to eliminate any possible ambiguity by having such material read automatically. Thus, the ability to include this additional information

in the spoken output should be configurable in the user's customisation preferences. If an explicit linkage between a `<label>` element and the control is not present, then the content of the `title` attribute for the field/control could be read instead, as suggested by (Hoffman et al., 2005). That is, if a `title` attribute for the element has been included by the page author.

Navigating in a linear fashion using the arrow keys is not the only method provided by WebTree to locate form fields/controls. These elements can be also reached through the specific element search function, or by tabbing through the elements on the document's tab-order. However, using these methods, the ability to establish the context in which a field/control appears is often rather difficult. This is because any contextual information provided by the surrounding text is lost. Thus, once navigating in this manner, the content of any explicitly linked `<label>` element, coupled with the control's type information, is automatically spoken. This is done in conjunction with the information previously described when discussing line by line navigation of these constructs. Where label information is not present, the content of the control's `title` attribute if it exists is read instead.

The optimum order in which contextual information for form fields /controls should be presented is very much dictated by user preferences. Thus, in future iterations of the WebTree system, the presentation of this material should be altered to reflect user choices.¹⁷

5.2.3 Speaking Tabular Data

As the user navigates from cell to cell through the use of the specialised table navigation functionality, the application begins reading the entire cell's contents once it is in focus. The original plan involved only speaking the first line of the portrayed content of the element concerned. The reason for this was to indicate to the listener the type of subject matter contained within. However this proved problematic in situations where the material

¹⁷No experiments to determine the optimum reading order of form field information was performed.

found on the first line was not descriptive enough. As tables should only be used for presenting strictly tabular data and not for page layout, reading the entire cell when under point ought to be a reasonable solution.

It should be remembered that as focus is placed on a data `<td>` element, the content of any associated `<th>` header elements is not automatically announced. Instead, to gain access to this material, a jump to the relevant header cell must be explicitly activated by the user. Once a table cell is in focus, the user can examine its content in more detail using the normal interactive reading functions. That is, reading by character, by word or by line. Thus, when the specialised header jump functions are invoked, the user can either listen to the entire content of the `<th>` tag before returning to the original pre-jump position, or examine its content in a more in-depth level.

There is no definitive answer to the question concerning how much table cell positional information is necessary so that the user doesn't get lost when navigating a complex table construct. The amount of contextual information to be announced is dependent on the users own reading preferences. Automatically announcing the row and column numbers for the current cell should remove any positional ambiguities relating to the specialised table navigation functions. This is especially advantageous in situations where the `rowspan` and/or `colspan` attributes are used to cause the cell to span more than a single grid position. However, many users may prefer to reduce speech verbosity in the interface by only having such information presented on request. Therefore the announcing of cell positional details is configurable in the *customisation* facility.

If the user attempts to navigate in a direction where no cell currently exists, the application informs the reader that movement along the required path is not possible. For example, consider the situation where focus is positioned on the cell with the coordinates row 1 column 1. If the user attempts to move upwards along the column they are informed that they are already in the top cell in the column. Similarly, if the user attempts to initiate a movement to the left, they are notified that they are already at the

beginning of the row. Alternatively, if focus is positioned on the final cell in a particular row or column, the user is informed accordingly if movement is attempted in a prohibited direction.

Although *nesting* of tabular constructs is syntactically allowed by the XHTML DTD (W3C, 2002*b*), it should never be used for data tables. Since the table navigation functionality provided by WebTree is premised on the basis of not having to deal with layout tables, it need not be concerned with this issue.

5.3 Reading Strategies

By nature, the viewing of material through a dynamically expandable tree structured interface requires much interaction on behalf of the reader. Due to the necessity to expand elements as or when they are required, the user has much more control over the amount of content being read in comparison to that of a traditional linear approach. The ability to create alternative views of the document ensures that the user can eliminate fragments of the content, which they deem to be less relevant. For example, during the first reading of a document, the user may prefer to skip over tabular data or interactive forms in the rendering to establish whether the material fulfils their expected criteria.

There are a number of different reading strategies facilitated by the WebTree system. The first method allows the user to employ a skim reading strategy. They can easily glean an overview of the document's content by examining the elements directly enclosed within the `<body>` element. Due to the ability to include additional contextual information as part of an element's tree control, e.g., the `title` attribute, the reader can quickly establish and jump to the most relevant segments of the document structure. Once the required material has been found, the user can easily expand the relevant elements to read the content in greater depth.

The second reading strategy available under the WebTree system is a continuous reading facility. The user navigates to a point in the document

and instructs the application to begin reading the content from that point on. The application continues reading subject matter until either the end of the current rendering is found, or the user explicitly requests a halt to the proceedings. However, it should be remembered that the WebTree system only speaks content rendered on the virtual display and does not automatically expand collapsed elements as they are encountered. Instead, the collapsed elements presence is signalled to the reader and the application continues reading. Therefore, to achieve the effect of continuously reading through the linear presentation of a document, all the elements must be already expanded. As part of future work, this functionality might be altered to expand each element as it is encountered. However, under the current implementation environment this is not a trivial task. This is due to the way emacspeak communicates with the speech synthesiser.

At the moment, the continuous reading function in emacspeak (used by WebTree to perform the continuous read functionality) does not realign focus to the word or phrase that is being spoken by the application. Instead, focus remains at the starting point at which the consecutive reading function was invoked. However, synchronising the word under point to coincide with the word currently being spoken would often be valuable when listening to lengthy texts. The problems with implementing this approach are directly attributable to the methodology that emacspeak exploits to communicate with the different speech servers that drive the individual synthesisers. Synchronising the audio stream to coincide with the actual position in the text requires much communication between the application and the speech synthesiser being used. However, this is difficult to achieve under emacspeak. The emacspeak functions pass the formatted text to the speech servers, which in turn transmit the information to drive the individual synthesisers. Little or no monitoring of the actual content as it is being spoken is performed.

5.4 Braille Output

There is no specific functionality to produce Braille output included in the WebTree system. That is, the content produced is not optimised for this form of interaction. Even so, the WebTree system should also provide advantages to this user group. As with speech, the amount of content viewable at any given time is limited. That is, many of the problems linked to the serial nature of speech are common to Braille interaction as well. Therefore, the tree viewing approach should also be an effective mechanism for reading documents. This is because it allows parts of the content to be bypassed without much difficulty. Also, the generation of alternative views should have a marked affect.

In most situations presenting the information as displayed in the virtual screen should be a usable approach. This is especially true when interacting with the tree content. Tree controls and element names would need to be included in the content, e.g., for `<h*>` elements, so that the user is aware of their contextual significance. Nevertheless, the presentation of contextual information, for example, table cell coordinates, is an interesting problem. With a large display, an area could be cornered off for portraying such content. However, with small displays this approach would be problematic. Therefore, further investigation is necessary before WebTree can produce a rendering optimal to Braille output.

5.5 Summary

In this chapter, the user interface of the WebTree system is described. The methods of rendering the different elements in the virtual screen and how these elements are to be portrayed through speech are discussed. The next chapter will outline the usability evaluations performed on the system and present a discussion of the results.

Chapter 6

User Evaluation

To establish whether providing access to the document's underlying tree structure demonstrates any advantages in comparison to interacting through the traditional linear approach, some evaluation with prospective users is required. A second question that must also be answered is whether such an approach hinders the rate at which content can be assimilated. Thus, the material contained in this chapter discusses the methods engaged to assess the WebTree application's usefulness. Initially, the test methodology is presented followed by any results gleaned from the formal user examinations. Next, an analysis of the user evaluations including issues observed during the test process, and some user impressions of the system are presented. These impressions were either stated during the test process or made on reflection through a questionnaire filled out once the evaluations had been completed. In the final two sections, a number of changes to the interface to increase the usability of the application are recommended.

6.1 Test User Profile

To assess the usability of the WebTree application and the appropriateness of the tree approach to accessing web-based documents, five totally blind individuals in the age group of 21 to 31 were asked to evaluate the software. Four of the subjects had a good knowledge of computing in general, whereas

the fifth user had a more limited knowledge. All were reasonably proficient screen reader users. JAWS for windows¹ was cited as being the predominant screen reader used. Only two subjects reported having recently used any other screen reader application.

All subjects reported that a version of Microsoft Windows² was the main operating system they worked with on a regular basis. Only two users had previously worked with a derivative of the Linux³ operating system. The same two participants stated that they had also used emacs⁴ previously, whereas, only one test subject said that they had ever used emacspeak⁵. Unfortunately, to test the application, blind users who already had a good knowledge of the operating environment were unable to be found.

All the participants use the web on a regular basis for both work/research purposes and leisure activities, such as reading newspapers or booking holidays. However, levels of knowledge concerning HTML/XHTML mark-up elements varied greatly throughout the user group. Two users claimed an advanced knowledge of mark-up elements, whereas one stated an intermediate level of understanding. Another participant had a basic understanding, whilst for the last subject knowledge of such mark-up elements was non-existent. None of the users had ever used the WebTree browser previous to the evaluation process.

Although this user sample is quite small, it should be large enough to glean a reasonable idea of the advantages and/or problems caused by this system. Testing with sighted users was not attempted, as the design of WebTree has been specifically motivated by, and optimised for, blind users.

¹http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

²<http://www.microsoft.com>

³<http://www.linux.org>

⁴<http://www.gnu.org/software/emacs/>

⁵<http://emacspeak.sourceforge.net/>

6.2 Methodology

The user evaluations were performed in the eAccess⁶ lab in Dublin City University. Each experiment was carried out separately, so that participants would not develop preconceived ideas of the system by listening to another test taking place in the background. Before the evaluations took place, each participant was asked to fill out a preliminary questionnaire to determine some user profile information; establish their levels of knowledge of HTML/XHTML elements; and ascertain how competent the user was in interacting with their current assistive technology applications, see section C.1 of appendix C for more details.

After filling out the preliminary questionnaire, the user was introduced to the WebTree software. They were initially given time to familiarise themselves with the user interface. This was first achieved by reading the user manual, appendix A, (read through WebTree). After this they were asked to complete a short tutorial, appendix B, which encouraged the user to experiment with the different types of functionality supplied by WebTree. There was no time limit placed on the familiarisation process, which typically lasted between one and two hours. Instead, the user was requested to indicate when they felt reasonably comfortable with the interface. At this point, participants were asked to perform a number of browsing tasks ranging from simple navigation tasks to more complex table navigation assignments. During the completion of these browsing tasks, the user had access to both the user manual and the tutorial as reference material. Participants were not encouraged to ask for help from the observer. Instead, they were encouraged to refer to the reference documents for help. However, if it was obvious that the user had become disorientated or confused by the audio presentation, then assistance by the observer was permitted.

Each participant was asked to complete seven browsing tasks in all. Four participants successfully completed all seven tasks; whilst the fifth subject's evaluation was curtailed due to unforeseen time constraints. After each

⁶<http://eaccess.rince.ie/>

task was completed, the user filled out a short questionnaire to assess the usability of the application. The questionnaire used for this purpose was the "After Scenario Questionnaire" (ASQ) (Lewis, 1995). The user was asked to rate the system under a number of criteria to establish the application's usability following the completion of each individual task. See section 6.3.1 for more details.

The first four tasks involved navigating through a tree presentation of a snapshot mirror image of the eAccess website. Users were asked to locate and navigate to specified documents. If the required document was not directly linked to by the current page, participants were informed of its location. The reason for including this number of similar tasks was to determine whether with practice the user's navigational efficiency improved. Also to ensure that if any usability issues occurred, they were related to the software and were not just problems concerning the usability of the website.

The fifth task required subjects to search within single pages for different elements using both the specialised element search and the restricted incremental text search functions. To complete task six, the user needed to locate and then fill out an XHTML form. Task seven required the user to navigate through a number of table constructs using the additional navigation functions supplied by WebTree.

Once all the tasks were completed, each participant was asked to fill out two additional questionnaires. The first was the "Computer System Usability Questionnaire" (CSUQ) to evaluate the overall usability of the application (Lewis, 1995). See section 6.3.2 for more details. The final questionnaire used in the WebTree user-testing process, presented in section C.2 of appendix C, tried to establish a better idea of user opinions about and expectations of the types of functionality included. The results of this are discussed in section 6.4.

No quantitative empirical testing in relation to the information location speed, or comprehension testing based on fixed time periods were performed on the application. Although it may be said that the user evaluation pro-

cesses described in this chapter are subjective in nature, it was necessary to proceed in this manner due to the technical background of the user group and the amount of experience that they would have had using the system. The time frame in which the user group members were exposed to the system is thought to have been long enough to provide a good indicator as to whether the approaches taken offer a practical solution for accessing documents. Nevertheless, it is accepted that a much longer period of usage by the test group is necessary before user tests could provide accurate quantitative data in relation to the system in comparison to other auditory solutions. The users were both unfamiliar with the operating environment and the WebTree system, therefore it is estimated that usage over a number of days, maybe even weeks would be necessary before such tests could take place. However, due to the location of the user group being distributed over a large area, providing more long term access to the application was problematic. The members of the test group were primarily Microsoft Windows users, therefore providing the application as a download would not have solved the problem.⁷ They would need to have had access to a machine running Linux, with a functioning emacs/emacspeak setup to run the application. In addition, providing the application as a download for the general public would not have helped the situation either. To perform the quantitative evaluations, a method to control and observe the users interacting with the system would still have been necessary.

In addition, any quantitative testing would have been subject to the starting point of the tree rendering for a given document. It is easy to test access to areas of a document when the starting point for WebTree is either a fully expanded or collapsed tree structure. However, WebTree allows for elements to be automatically expanded by default if requested by the user. Therefore any quantitative test such as those based on information location speed would be subject to the current arrangement of the document elements. WebTree also allows the user to customise websites to display a

⁷Finding emacs/emacspeak users to perform the tests was not possible.

default page rendering. Therefore, the usability of the system would depend on the user's preferences for rendering content.

The structure of web pages tested differed greatly in element hierarchical complexity. Ranging from index pages containing multiple *nested* list elements, e.g., news pages and paper indexes, to large technical reports containing numerous data tables and sectional divisions. In a number of cases the nesting of lists reached three levels of abstraction, however, many of the pages tested had only two levels of list nesting. Pages with *nested* tables were not tested. Similarly, interactive forms that depended on tabular layout were not included in the initial evaluations. However, documents containing predominantly text content were assessed.

The starting point for the tree structural view comprised certain higher level elements already expanded in the audio rendering. The reason for this was the need to expand specific elements common to each page before the content could be accessed. The elements automatically expanded in the text were the `<html>`, `<head>`, `<body>` and `<div>` elements. The `<div>` element adds an additional level of tree abstraction to the content. This can be useful in large documents where content sections are quite lengthy. However, in the case of short documents, having to expand each `<div>` element may reduce the efficiency of this approach. This is especially the case if there is no contextual information available to determine the nature of the enclosed content. A `<div>` element, although not exclusively, is often directly situated within the `<body>` element. If there are no other elements apart from `<div>`s at that level of the tree, it can be difficult to establish the context in which they appear, especially where no title information is included for the element. Due to the test site containing a number of `<div>` elements containing small amounts of content, it was decided to expand these elements without their tree controls in the audio rendering. Similarly the `<html>`, `<head>`, and `<body>` elements also appeared without their tree controls⁸.

⁸if the user wished, they could alter the customisation settings to include / exclude the tree controls for a given element.

	Question 1		Question 2		Question 3	
	Overall, I am satisfied with the ease of completing the tasks in this scenario.		Overall, I am satisfied with the amount of time it took to complete the tasks in this scenario.		Overall, I am satisfied with the support information (online-line help, messages, documentation) when completing the tasks.	
Task	Result	%	Result	%	Result	%
1	6.25	89.3	6	85.7	7	100
2	6	85.7	6.5	92.9	7	100
3	6.75	96.42	6.5	92.9	7	100
4	6.5	92.9	7	100	7	100
5	5.75	82.1	5.25	75	6.75	96.4
6	6	85.7	6.5	92.9	7	100
7	5.25	75	6	85.7	6.75	96.4

Table 6.1: ASQ results

6.3 Formal Questionnaire Results

6.3.1 Results of the After Scenario Questionnaire

Table 6.1 shows the results of the tasks when evaluated under the ASQ. Only results for the four users who completed all assignments have been included. Each question in the ASQ provides a scale from one to seven on which the usability of the task is rated. One represents *strongly disagree* and seven corresponds to *strongly agree*. The results presented for each assignment is the mean mark obtained for each task. The percentage that each mark represents is also included.

6.3.2 Results of the Computer System Usability Questionnaire

The results of the CSUQ are included in table 6.2. These results reflect the evaluations of all five users. Although the fifth participant did not complete all the tasks, it was thought that enough tasks had been completed to be able to give an accurate impression of the system. For each question, the user was asked to rate the system between one and seven, with one

representing *strongly disagree* and seven corresponding to *strongly agree*. The average mark achieved by the system plus the percentage that this mark represents is included for each question. In the case of question 9, only one person answered. All others responded *not applicable*. The user who did answer commented that it was the lack of feedback from searches and element expansions while in progress that caused such a low score.

6.4 Evaluation results summary

Due to participants not having an opportunity to use the WebTree interface previous to the evaluation, it took some time to become reasonably proficient using the system. Their total experience with the application was limited to between one and two hours, immediately before the tests commenced. A limited knowledge of both emacs and emacspeak were large contributing factors in getting to grips with the user interface. This was confirmed by a majority of the trial group users in their comments following the evaluation. This lack in knowledge was also observed during the test process. A number of users attempted to invoke command keystroke combinations available on their regular platform, instead of WebTree specific commands. In addition, a limited understanding of emacs/emacspeak specific short cut commands was quite evident.

Interaction with WebTree can be performed through a small set of generic commands. Nevertheless, many additional commands were included to aid power users of the application. Attempts were made to use where possible keystrokes that were mnemonic to the instructions employed. For example, *e* to perform an element search. However, as with any application there is a learning curve before users become proficient with its usage. The evidence suggested that as the user became more comfortable with the system, their productivity in using the application increased. Most users stated that with a little practice they thought they could become quite proficient users of WebTree.

The user with a limited knowledge of HTML/XHTML mark-up elements

1.	Overall, I am satisfied with how easy it is to use this system	5.8	82.9
2.	It was simple to use this system	6.6	94.3
3.	I can effectively complete my work using this system	6.4	91.4
4.	I am able to complete my work quickly using this system	6	85.8
5.	I am able to efficiently complete my work using this system	6	85.7
6.	I feel comfortable using this system	6.2	88.6
7.	It was easy to learn to use this system	6.6	94.3
8.	I believe I became productive quickly using this system	6.6	94.3
9.	The system gives error messages that clearly tell me how to fix problems	1	14.3
10.	Whenever I make a mistake using the system, I recover easily and quickly	6	85.7
11.	The information (such as online help, on-screen messages, and other documentation) provided with this system is clear	6.8	97.1
12.	It is easy to find the information I needed	6.2	88.6
13.	The information provided for the system is easy to understand	6.4	91.4
14.	The information is effective in helping me complete the tasks and scenarios	6.4	91.4
15.	The organisation of information on the system screens is clear	7	100
16.	The interface of this system is pleasant	6.2	88.6
17.	I like using the interface of this system	6.2	88.6
18.	This system has all the functions and capabilities I expect it to have	6	85.7
19.	Overall, I am satisfied with this system	6.2	88.6

Table 6.2: CSUQ results

initially had some trouble navigating documents. Once the mechanism of how such files are marked up was explained, he found the system quite usable. Another participant didn't like always having to expand elements to get to the required information. However, this problem was partly a symptom of the method he used to browse web pages. Browsing for this user was achieved in a sequential manner expanding each element, as the information was sought. None of the search functionality was used by this subject to locate items. He also had issues with the time it took to navigate to the specified documents using this method. This problem could be partially attributed to a lack of knowledge concerning the layout of the website. The evidence exists to suggest that as users' knowledge of website structure increased, the efficiency of browsing methods were also enhanced. The other three users did not display any major difficulties in navigating with this system.

All users stated that they could navigate through the content of web pages successfully using the WebTree application. When asked if they thought that through examining the document's tree structural arrangement, a general overview of the content could easily be obtained, 4 of the test users answered affirmatively. The same 4 users stated they perceived that they could navigate more easily to the main content of a given web page through this type of view. One participant suggested that with more practice and familiarity with the interface, this would be the case. However, it was argued by another group member that functionality in existing screen readers allows for similar navigation. The functionality referred to by this user is the ability to jump to a number of specific element types, often included by the developers of many auditory web solutions for the blind.

When the question was posed to determine whether viewing a document through the tree structural interface would have any major advantages and/or disadvantages over interacting with a linear rendering of the same document, most users were quite positive in relation to the system. What follows is a list of comments from members of the trial group in reference to the

question.

- “The support for the navigation of tables is good, otherwise I think the approach taken is complex, and doesn’t really offer any significant advantages over current systems. So yes and no.”
- “Absolutely. The reason is that in a visual modality you can see this structure and can navigate quickly to key elements. Linear presentation is serial in nature and places extra cognitive burdens on the user.”
- “I think viewing in the tree fashion allows one to get to the information more quickly.”
- “There is certainly an advantage of being able to navigate by any element in an XHTML document. Of course, this is predicated on the fact that the web page is coded correctly to XHTML levels. Some of the features such as the element search and incremental search are also nice. The way in which forms are accessed and filled in is also more user friendly than the way employed by other screen readers.”
- “It is easier to skip unnecessary or useless information.”
- “Added complexity; Extra time taken to navigate to the desired content”
- “Could be difficult for people with no HTML experience.”

Only one user thought the tree structural view added to the complexity of document interaction. He stated a major disadvantage was the extra time it took to navigate to the desired content by having to always expand each element when encountered. Another member of the user group was concerned that operating such a system might be difficult for people with little or no previous HTML experience. However, overall feedback on the tree structural rendering of content was quite positive.

Most users stated that they liked the outline web page summary that the tree structural view provided. This was especially evident when navigating through large documents. The ability to bypass complex items such as tables and just read the plain text they found appealing. Four of the participants believed that displaying some of the text contained in paragraph elements when in their collapsed state was beneficial in skimming the document. Whilst recognising this functionality to be of benefit to many, the fifth participant would prefer it to be optional. They all agreed that when additional information about the element is available, such as title, or a summary attribute in the case of tables, they should automatically be displayed as part of the tree expansion component.

Two participants were concerned that the use of technical terms in the user interface might be problematic in certain situations. For example, using the name of the mark-up element as part of the tree control information to identify its type. These issues should not regularly arise for users with a reasonable knowledge of HTML / XHTML constructs. However, for people with little or no knowledge of these items, presentation of data in this manner may be quite daunting. Therefore, an option to configure the application to announce the elements' full name was suggested. For example, instead of using the word `ul`, the phrase `unordered list` could be used. An alternative approach would be to provide a simple look-up mechanism to determine the full name of an element when encountered. This would enable the user to determine the genuine meaning of an element name, whilst keeping the verbosity levels low. Both mechanisms have advantages and disadvantages. Therefore user preference would dictate which method would be more beneficial.

Each member of the trial group thought it important to allow the user determine whether or not a tree control appeared in the audio rendering. The main reasons stated were as follows:

- "Showing controls for every single element in a document would make the browsing process more complex from the user's point of view; and

there may be certain elements the user isn't really concerned with in certain circumstances."

- "Because you can control what is displayed and how!"
- "I'd especially think it would be an advantage when navigating a web site you are familiar with. Obviously, when viewing a new site you will want to make sure you can view everything."

The ability to have certain elements expanded automatically in the audio rendering was thought to provide many advantages to the reader. It could reduce the amount of time required to locate a particular piece of content. This is especially the case where the page structural layout is already known. In these situations, a layer of interaction complexity is removed from the interface. Consider, as an example, a web page containing the following elements: `<html>`, `<head>`, `<body>`, a `` element containing a number of navigation links, and finally a number of `<p>` elements. Automatically expanding the `<html>`, `<head>` and `<body>` elements is a rather sensible approach. It removes the additional interactive steps that the manual expansion of these components would require. In addition, consider the case where the user may only be interested in the textual content of the page. Therefore, he may elect to have paragraphs automatically expanded, with the `` element remaining in its collapsed state. The user no longer has to trawl through the content of the list element to reach the main text. However, the list continues to be available for further expansion when necessary.

The application framework can be easily expanded to include additional functionality, therefore, it was suggested that in a future version customisations for specific web site domains or genres should be possible. The optimal customised view may differ greatly from site to site. For example, the elements to be automatically expanded for a newspaper web site are potentially rather different to those required for an internet banking application. However, within many web sites the content structure does not differ a great deal

from page to page. Therefore, setting criteria for the site wide presentation of content should be an achievable goal.

When appearing in the audio rendering without tree controls, users were often unable to recognise the significance of header `<h*>` elements. Even with the ability to establish the type of element currently under point, users were often unaware of the change in context afforded by their presence. Therefore, if such items appear without tree controls in the future, they should be signalled by either the name of the element being announced, or through a non-speech audio cue. Items can also be styled using alternative speech properties assigned by an aural cascading style sheet. However, the effect of styling elements in this manner was not tested significantly. Possibly due to the learning curve required to become proficient with a completely new system, or the fact that all participants were used to reading with a single voice, users chose to have alternative voice cues turned off at all times during the evaluation process.

Although it may be unique to this small sample of users, it was evident that the regular browsing strategies of each user appeared to have an impact on how usable they perceived the system. Those users that articulated that they usually navigated line-by-line or based on screen blocks such as paragraphs or by virtual pages based on the information present on the screen, struggled at first to get to grips with the tree structural interface. However, those users that regularly use advanced navigation commands, such as navigating by element, or searching for required screen text, appeared to adapt to the system quite easily. A direct comparison of browsing with WebTree and the users' regular assistive technology solution was not attempted. The main reason being the user would have had much greater experience using their assistive technology solution. This experience would have been gained over a number of years in many cases. Methods for browsing web-based documents, specific to that assistive technology would have been developed by the user over that time whereas their entire experience using WebTree totalled a number of hours.

6.4.1 Interactive Form Data

All test subjects liked not having to specifically activate a specialised forms mode when entering form data. One user commented:

I particularly like the fact that entering a specialised forms mode is not required. The key mappings changing automatically is a good feature.⁹

However, most test users initially had some difficulty with the form construct. The problem was unrelated to the task of filling out the information or locating individual fields. It was as a direct consequence of the ability to include structural elements such as paragraphs inside form constructs to organise the content. Most users had expected to encounter the form input fields immediately after expansion, and were a little confused when a paragraph element was encountered instead. To solve this problem, a function to automatically expand an element's entire sub tree is proposed. This will complement the regular expansion function. Also, a mechanism to specify that once a specific element has been expanded, certain named constituents if present within its sub tree, appear in their expanded state regardless to how they are treated outside of this construct should be considered.

The standard use of the enter key for setting controls, such as checkboxes and radio buttons was liked by the trial user group. This key is consistently used to interact with all button controls across the WebTree interface. One particular user found that it was sometimes difficult to place focus on a form field/control. Although, he was happy with the way they were presented¹⁰.

There is currently no method to quickly jump to the form submission button and submit the form. The generic element search function would not suffice for it examines the element name, in this case `input` whereas the content of its `type` attribute would be more beneficial. In the case of

⁹The key map changes in question were those from WebTree specific keystrokes back to the character entry functionality, once a form field was in focus.

¹⁰Form fields are generally presented at the start of a line, thus relocating focus to that point allocates focus to a position within the field.

a large form such functionality may be preferable instead of requiring the user to navigate through the fields to locate the submit control. There is no guarantee that the user will attempt to fill out the information in the sequential format in which it appears in the text. In fact there is no way to predict where exactly within the form construct the submission information might occur. Thus, a quick submit function was called for by one member of the trial group.

6.4.2 Searching

The only complaints in relation to the different types of search functionality were concerned with the amount of audio feedback presented. In the case of large documents, searching may take a little time, thus notification that the search was still in operation was requested. Apart from that, users liked the different search technologies and thought them to be of benefit. The designated element search was perceived as being rather convenient when attempting to locate an element not readily available within the virtual screen rendering. One particular trial group member used this technology extensively throughout the evaluation. The ability to limit the incremental text search to specific named elements was also thought to be beneficial across the entire user group.

However, one user requested a static text search facility to complement the existing search alternatives. In many situations this technology would reduce the time necessary to complete a search. This is due to the application no longer requiring the expansion of tree controls to ensure that as each fragment of the search string is located in the uncompressed DOM tree structure it is physically presented in the audio rendering. The fact the search is only performed on one occasion should also reduce its completion time. However, there is one major disadvantage with this type of search functionality. To find the required material, the correct spelling of the word or phrase is necessary. Whereas, with an incremental text search, a small number of letters entered may be enough to find the content.

6.4.3 Tabular Data

The use of the keypad as the basis for providing access to the table navigation functions was perceived as an integral component of its usefulness. This is partially due to the close proximity of the required keystrokes to one another, allowing for extremely quick and simple navigation. It was also suggested by one member of the trial group that the numeric keypad fits the cellular nature of tables perfectly. In general, the different types of navigation methods functioned within users' expectations. However, one user requested that the commands used to jump to the start/end of the table construct place the focus in the first and last cells respectively. This is contrary to the current implementation, which leaves the user at the physical point in the text, where the construct begins and ends.

The table navigation functionality was well received across the entire user group. The ability to jump to a table header element, peruse its content and relocate focus to the previous position was generally thought a good idea. However, it was suggested that an additional function, which announced the relevant header content without manoeuvring away from the current cell would be preferable. Users stated that they would rather not have header information read automatically as focus is placed on a different cell. Instead, they would prefer such information to be available on demand as of when required. In addition, from our observations of the tests, a number of the users found the automatic announcing of row and column positional data quite irritating. A command to read this information at the users request appears to be a more beneficial method of portraying such material. To facilitate users who wish to have such material automatically announced, a configuration feature to control its portrayal should be included as part of the customisation component.

Two members of the trial group in particular liked the linear organisation of the content in the virtual screen presentation. It facilitated the linear reading strategy they regularly employed when assimilating tabular data. Both users rarely used the specialised table navigation functionality

incorporated into their current screen reader applications. However, they both stated they were happy with the grid-like navigational approach pursued by WebTree.

A number of problems with the current WebTree table construct implementation were discovered during the testing process. Once focus was positioned on an end of line character between two different cells, the application announced that the user was not currently in a table cell when attempting to perform a cell to cell jump. To solve this problem, the initial cell could be used as the starting point for the navigational jump where this situation arises. A second problem was due to the original navigational strategy only announcing the initial line of content after navigating along the spatial relationships between cells. However, in many cases the segment of information available was not enough to give an accurate impression of the cells content. Thus, it may be better to automatically read the entire cell's content after a navigational jump.

Although evaluation was performed using large tabular constructs, tables containing cells that span more than a single row or column were not assessed. The reason for this was simple. The time period available for the user to obtain a working knowledge of the application was quite short. The learning curve associated with learning a new system from scratch, coupled with the wide range of different technologies to be evaluated, meant the inclusion of such tests would have considerably added to the complexity of the evaluation. It was thought that this level of complexity would have drastically increased the learning curve for this particular application. Thus, the performance of these tests have not been completed as of yet.

Experiments to determine the best methods for navigating to and from these elements along the grid connections still need to be completed. Would moving along the first column or row in which the cell appears be a viable solution to this problem? Or would navigation along the same row or column from which focus has been originally manoeuvred be a better solution. In addition to this comparative evaluation, the mechanisms for informing the

user of the cell's presence must also be investigated. It is necessary to establish whether the application should announce this information when a cell of this type is encountered, or whether it would be better to use a non-speech sound cue to notify the user of its presence. To answer these questions, an evaluation entirely centred on table navigation under WebTree would be necessary.

6.4.4 Customisation Issues

The one component of the WebTree system that users had great difficulty in using was the customisation facility. The emacs specific `customize` package was used to generate customisation views. However, none of the users had any previous experience of working with emacs customisation buffers, therefore, found them rather difficult to exploit. Emacspeak often employs the use of alternative voices to portray contextual changes in the content. For example, one voice type is reserved to portray text positioned within quotation marks, whilst another is used to announce text contained within a clickable button. However, there is a learning curve to this approach for presenting changes in context. Some practice is required to learn how to decipher the context associated with each type of voice. The user group was not exposed to the WebTree system, or emacspeak for a great deal of time, thus, accurately learning these associations would have been difficult. In addition, the participants found the sudden alterations in voice quite disconcerting, therefore, each one requested that such changes be turned off during the evaluations.

Without voice changes to illustrate variations in context, it can be difficult to decipher the different components of individual elements in the customisation buffer. This is especially the case for inexperienced users. Although the customisation facility is organised as a tree-like structure, the mechanisms for interaction with individual variable options is alien to the rest of the functionality provided under WebTree. Thus users were required to master an additional interface in conjunction to the software being eval-

uated. The arrangement of components comprising each element was not initially obvious to the users in the test group. The methods employed by their regular screen reading applications to portray similar information differ greatly from the approaches taken under emacs/emacspeak. For example, JAWS for windows in conjunction with Internet Explorer displays form edit fields on separate lines to other content. This is not the case under emacs / emacspeak. Whilst tabbing through the different components of a customise element the values of each field/control are announced. However, the context in which they appear is often lost for emacspeak does not automatically speak this material. Therefore, all participants required a great deal of assistance before they could successfully manage to operate the system.

For these reasons, it was suggested that this method for providing customisation should be scrapped in favour of implementing an alternative scheme based on an XHTML form input system. For the most part, in terms of controlling the expansion of elements and the presence of tree controls, this could be achieved by a set of checkboxes, or a multiple select list. The insertion of items, such as line length, could be facilitated by a text edit field. A specialised mechanism designated to the dynamic generation of alternative views should also be considered for power users. See the previous chapter for more details.

6.4.5 Usability Versus Page Complexity

The perceived usability of the system was not directly proportional to the complexity of the page being viewed. Although, as expected, it did have some bearing. Short, yet structurally complex pages fared worst of all under the WebTree system. Having to expand multiple elements to obtain access to a small fragment of text did impinge on the usability of the application. For example, consider the following scenario: a list of publications with a nested list pertaining to each year. If each year contains only one /two publications, requiring the user to expand the list for each year may be overkill. However, in those cases where the number of publications is

quite large, the benefits of having this information hidden when it isn't the target data sought, outweighs the inconvenience of having to expand the element. Therefore, allowing the user select whether a structural element is automatically expanded based on the amount of child elements it contains, might increase the efficiency of the system.

The greatest benefits of the system were observed during the viewing of large complex documents. When faced with large lists and multiple data tables, the WebTree system facilitated an easy mechanism for navigating past these elements to only read the text components of the document. Reading such content often can provide the reader with a good insight into the nature of the documents' subject matter. Then if the reader wishes to peruse the content in more detail, they can simply expand the relevant elements.

When looking for a specific hyperlink on a given page, the additional complexity afforded by the tree system could potentially impinge greatly on browsing efficiency. Having to expand each element to find out whether the required hyperlink is situated within, is extremely problematic. For this reason, additional navigational functionality to manoeuvre link by link is provided. Alternatively, the search functionality could be used to locate the link, or at least navigate focus to the surrounding text. However, the facility to navigate link by link does not offset problems with links requiring contextual information gleaned from the surrounding text to indicate their meaning. This is a problem that is experienced by many applications providing alternative views based on lists of links and is not easily solved. In the case of WebTree, once a user navigates to a link, the link and its ancestor hierarchy are automatically expanded into the text. Therefore, the user can examine the surrounding text to assess the implied meaning.

The tree like view is a powerful mechanism allowing the user to quickly skim the document to obtain an overview of its structural make-up. It also permits the user to easily skip fragments of the content to more quickly reach the required data. However, there are times when linear interaction is a more

beneficial method for perusing written material. When perusing a segment of content in detail, such as continuously reading a section of content, then it is advantageous to have the content of all elements expanded into the rendering. Both interaction mechanisms have advantages / disadvantages depending on the task in hand. Therefore, the ability to linearise a segment of the tree, by expanding an element including its entire sub tree should increase the usability of the system.

6.5 Conclusion

Overall the results of the user evaluations were quite positive. It appears from these results that the approach taken by WebTree is a viable solution for blind people to access the web. Although, ease of use of the application does depend somewhat on their levels of experience with both screen reader applications and with computers in general. It is not thought that vast experience with screen reading technology is necessary to operate the system efficiently; however, a reasonable knowledge of computing would be advantageous. An in-depth knowledge of the HTML family of mark-up languages is not necessary to access the system. The user only needs to be concerned with a number of structural elements to operate WebTree effectively. Therefore, a basic knowledge of the elements and the mechanisms of how they are structured should suffice.

Some additions to the interface coupled with a number of minor changes with existing functionality, were recommended during the evaluations. Four out of the five users tested thought it a viable system for browsing web pages. They also indicated that such a system would provide a number of advantages over a linear interface. However there are some drawbacks. One user stated that he would rather not use the interface due to the complexity that element expansion brings to the browsing of web pages. However, he did not attempt re-configuring the display parameters for elements to automatically portray them in their expanded state. Although initial usability results are quite positive, to obtain a better idea of the usefulness of this sys-

tem for blind users, more familiarisation with the functionality is necessary, especially the customisation facility for controlling the display of elements.

It is difficult to assess a complex system such as WebTree after only a number of hours of usage. To realistically evaluate such an application, it would be necessary for users to attempt to operate the system on a more extended basis. In addition, it is difficult to simulate real time usage of the application, so a number of the tasks may have seemed contrived.

As expected, the benefits of the tree structured approach to document rendering were much more apparent when considering larger documents. Users found it beneficial to bypass unrelated material to quickly locate the required content. The ability to generate views with elements such as `<table>`, `<form>` and list constructs collapsed by default facilitated this navigation. Viewing documents in this manner negates trawling through reams of lengthy information when attempting to locate the required content. However, evaluation using a more comprehensive mission critical website such as an e-banking or e-commerce website should provide a more definite impression of the application's usefulness. The major problem is to locate a suitable website for the evaluation process. A website claiming to be fully accessible, i.e., marked up in accordance with the W3C's web content accessibility guidelines W3C (1999*b*), and validating to the XHTML basic standard W3C (2000*b*) is what is required.

Chapter 7

Conclusions and Further Work

The objective of this research was to examine alternative strategies to browse and navigate web based documents through a non-visual modality. The traditional approach is to present content in a linear form, removing many of the structural cues provided by the visual organisation of the content. Although functionality to supply navigation non-linearly is often provided, there still exists many problems in finding the relevant information on a page through a purely serial approach. The ability to skim through the text guided by visual cues such as layout, colour or changes in font is unavailable in this medium. To alleviate many of these problems, a number of auditory web solutions perform some semantic analysis on the content in an attempt to generate a usable summary of the material. In section 3.5 the literature concerning the different methods for navigation and page summarisation was discussed.

As part of the current work a number of different strategies for interacting with web based documents were devised. The proposed tree viewing strategy, described in chapter 5, allows the user to quickly skim the document to establish whether the content meets their requirements. The user controls how much of the documents content is presented at any given time,

by expanding and collapsing individual elements. When certain elements are left unexpanded in the audio rendering e.g. `<table>` or `<form>` elements, the user can quickly bypass these elements and just read the text. Often the context in which an element appears is obvious from reading the preceding material. However, when contextual information, such as a `title` attribute is available for an element, it is included in the audio rendering. This is done to provide some knowledge of the material enclosed within a collapsed element. By also providing a segment of each paragraph in the virtual screen whilst still in their collapsed state, the user can obtain an idea of the enclosed material. If the user decides they need to read the content in more detail, they can easily expand the relevant elements.

While interfacing with the document through this approach, the user can quickly navigate to the elements of the page they deem important. Through the customise facility, users can select whether to automatically expand a given element, or leave it in its collapsed state. In this way, an optimal view based on the user's requirements can be generated. Obviously, there are many possible variations in web page structure, thus, the rendering optimal for one page may be inefficient for another. However, many websites impose a given structure on all/most of its online content; therefore the generation of site specific settings ought to be a viable solution to this problem.

As demonstrated by the user evaluations discussed in chapter 6, the tree based view is a useful approach to the viewing of web based documents. However, the methods were more effective for large documents as opposed to shorter index pages. Nevertheless, a strict tree view was not an ideal solution for the viewing of all elements. That is, elements that rely on their visual organisation to denote meaning, e.g., the `<table>` and `<form>` elements. To accurately convey these types of content, and the contextual information required to interact with their lower level elements, additional browsing strategies needed to be derived. See section 5.1.6 and section 5.1.7 respectively for more details on how these elements are supported by Web-Tree.

To solve the problem of how best to convey contextual information, a number of different strategies are employed by the system. Firstly, the *name* of the element is included in the tree control information, so the user knows which structural element is currently under point. However, an alternative approach facilitated by the system is to use different voices, changes in voice characteristics or non-speech sound cues to signal this information. These additional methods for signalling material can be provided by an aural CSS style sheet. However, the use of CSS is optional. It is a lot to expect an average computer user to write their own style sheet, therefore a number of default style sheets should be provided, so the user can select one that meets their needs.¹

The user is the only one that knows the type of presentation they personally find optimal. Some users prefer verbose interfaces with little ambiguity in the context in which elements appear. Others rather have much of this information provided only on request. For example, reading table header information, or the grid position in which a cell appears in a table. In general, the approach taken by WebTree is to allow the user customise the type of spoken output provided. In this manner the user can create a browsing solution that caters for their own needs. Although a default setup is offered, the user is free to deviate from this at any time.

Many current solutions provide navigation functionality based on a limited set of mark-up elements. WebTree extends this approach by providing a *generic* element search function. That is, the user can search for any element in the document, and focus will jump to the point in the document where it occurs. Similarly JAWS for Windows² allows the user to search for paragraphs containing specific words, see section 3.5. The functionality is extended to allow the user to limit the text search to any element. For example, the application could be asked to only look within a `<table>` ele-

¹In the case of *inline* elements the only method for obtaining knowledge of the element's type apart from the use of audio cues is by requesting the name of the element currently under point.

²http://www.freedomscientific.com/fs_products/JAWS_HQ.asp

ment or a `<form>` construct for the required information. In this manner it is hoped that the ability to find and navigate to specific material should be improved. This is especially the case where the structure of the web page is previously known. From the initial user evaluations, discussed in chapter 6, the feedback on the different search functionality was quite positive. However, whether the inclusion of these methods is beneficial will only become apparent with extensive usage of the system.

One of the principle goals in the generation of the prototype application was to try to keep the interface as simple as possible. By consistently using the same keystrokes to perform similar tasks, e.g., using the `enter` key to activate a `radio button` or a tree expansion control, the learning curve to become proficient with the application is reduced. Similarly, much of the functionality of WebTree can be used through a small set of generic functions. Additional navigation functions based on these generic methods were added to aid power users.

To operate this system, you do *not* need to be an advanced screen reader user. In fact, the basic user interface is quite simple so it is hoped that novice users of auditory browsing solutions could effectively work with this application. However, a certain amount of knowledge of computing is necessary to use WebTree. That is, the user needs to have some general knowledge of HTML/XHTML constructs to interact with the system effectively.

Finally, WebTree operates on two assumptions. The first being that pages meet WCAG Double-A compliance (W3C, 1999b). Although this could be seen as a major assumption, other auditory web solutions operate most effectively when similar accessibility levels are reached. As with WebTree the less accessible the web page, the less efficient these applications are in conveying the material. The second major assumption is that well formed mark-up is used to generate the documents. Although this is covered by the WCAG guidelines, it is the main premise on which the system is built. If elements are not used for their intended purposes, then the advantages of viewing the tree structure will be reduced. That is, the context often af-

forded by the element type will be lost. For example, if a `<h1>` element is used to mark-up a document heading then the user can establish the context in which the content appears by its tag name. However, if the same element is used to mark-up a paragraph based on how it will look visually, then the advantages of using structural elements for blind users are negated.³

7.1 Further Work

The current prototype was developed so that the proposed browsing and navigation strategies could be assessed. However, much work is still necessary before the system can be described as a workable solution for browsing web based documents. That is, not all of the described functionality has been fully implemented. Therefore the next stage in the development process is to complete the implementation of those elements that are not entirely finished. For example, code should be added to handle more of the aural CSS properties. Also, as mentioned in section 5.1.2 and section 6.4.4, user testing has shown that there are problems with the current implementation of the customisation facility. The solution needs to be redeveloped to be more consistent with other functionality proposed in WebTree. As previously mentioned a facility mimicking an XHTML `<form>` construct could be used to enter the data. Also many additional customisation parameters, such as those to provide greater control over the speech output should be added.

Further testing is required to establish how beneficial the approaches taken actually are. The testing performed so far only included a small number of hours usage with the system. To gain a better idea of the system's usefulness on an individual component basis, further testing over a prolonged period of time is required. One problem due to the development environment is that only a small minority of blind users use emacspeak. This is problematic because it is difficult to find users in Ireland that use

³Although the current prototype is limited to viewing XHTML pages, the approach should be beneficial for any HTML derived document.

emacs/emacspeak on a regular basis. Therefore, it might be a good idea to redevelop the application to function under a more mainstream environment, such as under the Mozilla⁴ project. In this way the application can take advantage of the user's normal setup, to determine problems with the proposed strategies. This is because it was difficult to establish if a number of the problematic issues observed in the user evaluations were symptoms of the approaches taken, or were caused by the user operating in an alien environment.

Finally, a number of recommendations proposed by the user group, such as the ability to expand entire sub trees should be implemented. Also, as mentioned in chapter 5, some further testing to establish the best methods for navigating table cells spanning more than one row/column needs to be performed. Also, as mentioned in section 3.8 there currently exists a number of methods in which tree information can be unobtrusively conveyed to the user. However, the added complexity afforded by these approaches might detract from the usefulness of the system. Also, these methods could potentially clash with the additional signalling methods for contextual information afforded by an aural CSS. Therefore, a simple yet consistent method for portraying tree information in addition to the current solution should be sought.

7.2 Contributions of this Research

The WebTree system is a novel alternative to the traditional linear approach to document navigation for blind users. The primary view is based on exposing the tree structural relationships between elements contained in the document's mark-up. The ability to dynamically hide or expose large sub-trees, coupled with searching both content and structure, allows rapid navigation through large documents. The traditional navigation by element is extended to include all elements as opposed to a specific named set. This is achieved through a generic element search. This work also shows how

⁴<http://www.mozilla.com/>

complex elements such as <form> and <table> elements can be handled under a tree viewing system. To accurately convey tabular data required a departure from the strict tree representation to one that was more graph-like in nature. Interaction with form constructs does not require a special mode that must be invoked by the user. Instead the application manipulates the keystroke functionality so that interaction can be performed directly when the control is encountered.

References

- American Heritage Dictionary (2000), 'The American Heritage Dictionary of the English Language'. Houghton Mifflin Company, fourth edn.
- Arons, B. (1997), 'SpeechSkimmer: a system for interactively skimming recorded speech', *ACM Transactions on Computer Human Interaction* 4(1), 3–38.
<http://portal.acm.org/citation.cfm?doid=244754.244758>
- Asakawa, C. and Itoh, T. (1998), User interface of a Home Page Reader, *in* 'Proceedings of the third international ACM conference on Assistive technologies', ACM Press, pp. 149–156.
- Asakawa, C. and Takagi, H. (2000), Annotation-based transcoding for non-visual web access, *in* 'Proceedings of the fourth international ACM conference on Assistive technologies', ACM Press, pp. 172–179.
- Barnicle, K. (2000), Usability testing with screen reading technology in a Windows environment, *in* 'Proceedings on the 2000 conference on Universal Usability', ACM Press, pp. 102–109.
- BAUK (2004), *British Braille A Restatement of Standard English Braille*, Royal National Institute for the Blind, Peterborough, UK. Compiled and Authorized by the Braille Authority of the United Kingdom.
<http://www.bauk.org.uk/pubs.htm>
- BAUK (2005), *Braille Mathematics Notation*, second edn, Royal National Institute for the Blind, Peterborough, UK. Compiled and Authorized by

the Braille Authority of the United Kingdom.

<http://www.bauk.org.uk/pubs.htm>

BAUK (2006), *Braille Computer Notation*, Royal National Institute for the Blind, Peterborough, UK. Compiled and Authorized by the Braille Authority of the United Kingdom.

<http://www.bauk.org.uk/pubs.htm>

Bell, D. (1962), 'Reading by Touch', *Typographica* 6.

<http://tinyurl.com/qhwyt>

Bernstein, M. and Picker, M. (1966), *An introduction to music*, 3 edn, Prentice-Hall, Englewood Cliffs, NJ.

Blattner, M. M., Sumikawa, D. A. and Greenberg, R. M. (1989), 'Earcons and Icons: Their Structure and Common Design Principles', *HUMAN-COMPUTER INTERACTION* 4(1), 11-44.

Brewster, S. A. (1998), 'Using nonspeech sounds to provide navigation cues', *ACM Transactions on Computer-Human Interaction (TOCHI)* 5(3), 224-259.

Brewster, S. A., Wright, P. C. and Edwards, A. D. N. (1993), An evaluation of earcons for use in auditory human-computer interfaces, in 'Proceedings of the SIGCHI conference on Human factors in computing systems', ACM Press, pp. 222-227.

Brewster, S. A., Wright, P. C. and Edwards, A. D. N. (1995a), Experimentally Derived Guidelines for the Creation of Earcons, in 'In Adjunct Proceedings of HCI'95, Huddersfield, UK', pp. 155-159.

Brewster, S. A., Wright, P. C. and Edwards, A. D. N. (1995b), 'Parallel Earcons: Reducing the Length of Audio Messages', *International Journal Of Human-Computer Studies* 43(2), 153-175.

Brewster, S. and Brown, L. M. (2004), Tactons: structured tactile messages for non-visual information display, in 'Proceedings of the fifth conference

- on Australasian user interface', Vol. 28, Australian Computer Society, Inc., pp. 15 – 23.
- Brown, M. L., Newsome, S. L. and Glinert, E. P. (1989), An experiment into the use of auditory cues to reduce visual workload, *in* 'Proceedings of the SIGCHI conference on Human factors in computing systems: Wings for the mind CHI '89', Vol. 20, ACM Press, pp. 339-346.
- Brown, S. S. and Robinson, P. (2001), 'A World Wide Web Mediator for Users with Low Vision'. Paper presented at the CHI'2001 Conference on Human Factors in Computing Systems Workshop No. 14.
<http://is4all.ics.forth.gr/chi2001/files/brown.pdf>
- Buxton, W., Gaver, W. and Bly, S. (1994), *Auditory Interfaces: The Use of Non-Speech Audio at the Interface*, Unpublished. Incomplete draft manuscript.
<http://www.billbuxton.com/Audio.TOC.html>
- by Jim Highsmith (1997), 'Messy, Exciting, And Anxiety-Ridden: Adaptive Software Development', *American Programmer* .
<http://elj.warwick.ac.uk/jilt/01-2/sloan.html>
- Chimera, R. and Shneiderman, B. (1994), 'An exploratory evaluation of three interfaces for browsing large hierarchical tables of contents', *ACM Transactions on Information Systems (TOIS)* **12**(4), 383-406.
- Chisholm, W. and Novak, M. (1999), Increasing the accessibility of the web through style sheets, scripts and plug-ins, *in* 'CSUN 1999'.
<http://tinyurl.com/fwlmf>
- d'Alessandro, C. and Liénard, J.-S. (1996), *Synthetic Speech Generation*, Cambridge University Press, London/New York,, chapter 5.2.
<http://cslu.cse.ogi.edu/HLTsurvey/ch5node4.html#SECTION52>
- Dudley, H., Riesz, R. R. and Watkins, S. A. (1939), 'A synthetic speaker', *Journal of The Franklin Institute* **227**, 739-764.

- Dudley, H. and Tarnoczy, T. (1950), 'The speaking machine of Wolfgang von Kempelen', *Journal of the Acoustical Society of America*, **22**, 151–166.
- Edwards, W. K., Mynatt, E. D. and Stockton, K. (1994), Providing Access to Graphical User Interfaces - Not Graphical Screens, *in* 'Proceedings of the first annual ACM conference on Assistive technologies, Marina Del Rey, California, United States', ACM Press New York, NY, USA, pp. 47–54.
- Fant, G. (1960), *Acoustic Theory of Speech Production*, Mouton, 'sGravenhage, The Netherlands.
- Filepp, R., Challenger, J. and Rosu, D. (2002), Web accessibility: Improving the accessibility of aurally rendered HTML tables, *in* 'Proceedings of the fifth international ACM conference on Assistive technologies', ACM Press.
- Fitzpatrick, D. (1999), Towards Accessible Technical Documents: Production of Speech and Braille Output from Formatted Documents, PhD thesis, School of computer applications, Dublin City University.
- Fitzpatrick, D. (2002), Speaking Technical Documents: Using Prosody to Convey Textual and Mathematical Material, *in* 'Proceedings of ICCHP 2002', Springer Lecture Notes in Computer Science (LNCS), Springer, pp. 494–501.
- Fitzpatrick, D. (2006), Mathematics: How and What to Speak, *in* 'Proceedings of ICCHP 2006', Springer Lecture Notes in Computer Science (LNCS), Springer.
- Fitzpatrick, D. and Karshmer, A. I. (2004), Multi-modal Mathematics: Conveying Math Using Synthetic Speech and Speech Recognition., *in* 'Proceedings of ICCHP 2004', Springer Lecture Notes in Computer Science (LNCS), Springer, pp. 644–647.
- Fitzpatrick, D. and Monaghan, A. (1998), TechRead: A System for Deriving Braille and Spoken Output from LaTeX Documents, *in* 'Proceedings

- of ICCHP 1998', Springer Lecture Notes in Computer Science (LNCS), Springer, pp. 316–323.
- Fitzpatrick, D. and Monaghan, A. (1999), 'Browsing Technical Documents: Document Modelling and User Interface Design', *Bulletin De Linguistique Appliquee Et Generale* **24**, 5–18.
- Foulke, E. (1964), 'Transfer of a Complex Perceptual Skill', *Perceptual and Motor Skills* **18**, 733–740.
<http://www.braille.org/papers/skills/skills.html>
- Foulke, E. (1979), 'Investigative Approaches to the Study of Reading Braille', *Journal of Visual Impairment and Blindness* **73**(8), 298–308.
<http://www.braille.org/papers/invea/invea.html>
- Foulke, E. (1982), *Reading braille*, Cambridge University Press, pp. 168–208.
- Furuta, R. (1994), 'Defining and using Structure in Digital Documents', *Proceedings of the First Annual Conference on the Theory and Practice of digital libraries* .
<http://www.cSDL.tamu.edu/DL94/>
- Gaver, W. W. (1986), 'Auditory Icons: Using Sound in Computer Interfaces', *Human-Computer Interaction* **2**(2), 167–177.
- Gold, B. and Morgan, N. (2000), *Speech And Audio Signal Processing: Processing and Perception of Speech and Music*, John Wiley & Sons, Inc.
- Goose, S. and Möller, C. (1999), A 3D audio only interactive Web browser: using spatialization to convey hypermedia document structure, in 'Proceedings of the seventh ACM international conference on Multimedia (Part 1)', ACM Press, pp. 363–371.
- Grabe, E. (2004), *Intonational variation in urban dialects of English spoken in the British Isles*, Linguistische Arbeiten, Tuebingen, Niemeyer, pp. 9–31.

- Greenspan, S., Nusbaum, H. and Pisoni, D. (1988), 'Perception of synthetic speech produced by rule: Intelligibility of eight text-to-speech systems', *Behavioral Research Methods, Instruments, and Computers* **18**, 100–107.
- Grunwald, A. (1966), 'A Braille-reading machine', *Science* **154**, 144–146.
- Harris, A. j. (1947), *How to increase reading ability*, New York: Longmans, Green.
- Hoffman, D., Grivel, E. and Battle, L. (2005), 'Designing software architectures to facilitate accessible Web applications', *IBM Systems Journal* **44**(3).
<http://www.research.ibm.com/journal/sj/443/hoffman.html>
- James, F. (1998), Lessons from Developing Audio HTML Interfaces, *in* 'Third Annual ACM Conference on Assistive Technologies', pp. 27–34.
- Johnson, D. G. (2004), 'Fact Sheet on Manual Braille Writing Aids and Labelers'. Document produced by ABLEDATA.
<http://tinyurl.com/f9se7>
- King, A., Evans, G. and Blenkhorn, P. (2004a), 'Blind people and the World Wide Web'. UMIST, Manchester, UK.
<http://www.webbie.org.uk/webbie.htm>
- King, A., Evans, G. and Blenkhorn, P. (2004b), 'WebbIE, a web browser for visually impaired people'. Poster presented at the 2nd Cambridge Workshop on Universal Access and Assistive Technology (CWUAAT).
<http://www.webbie.org.uk/papers/King-CWUAAT2004.htm>
- Klatt, D. H. (1987), 'Review of text-to-speech conversion for English', *Journal of the Acoustical Society of America* **82**(3), 737–793.
http://www.mindspring.com/~ssshp/ssshp_cd/dk_737a.htm
- Knowlton, M. and Wetzell, R. (1996), 'Braille reading rates as a function of reading tasks', *Journal of Visual Impairment and Blindness* **90**, 227–235.
<http://www.braille.org/papers/jvib0696/vb960312.htm>

- Lai, J., Cheng, K., Green, P. and Tsimhoni, O. (2001), On the road and on the Web?: comprehension of synthetic and human speech while driving, *in* 'Proceedings of the SIGCHI conference on Human factors in computing systems', ACM Press.
<http://tinyurl.com/nqjcp>
- Lai, J., Wood, D. and Considine, M. (2000), The effect of task conditions on the comprehensibility of synthetic speech, *in* 'Proceedings of the SIGCHI conference on Human factors in computing systems', ACM Press.
- Lamport, L. (1985), *Latex: A Document Preparation System*, Addison Wesley.
- Larson, K. (2004), 'The Science of Word Recognition'. Microsoft Corporation, Advanced Reading Technology group.
<http://tinyurl.com/4qrnk>
- Legge, G. E., Madison, C. and Mansfield, J. S. (1999), 'Measuring Braille reading speed with the MNREAD test', *Visual Impairment Research* **1**(3), 131-145.
http://gandalf.psych.umn.edu/~legge/braille_reading.pdf
- Lewis, J. R. (1995), 'IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use', *International Journal of Human-Computer Interaction* **7**(1), 57-78.
- Lorimer, J. and Tobin, M. J. (1979), 'Experiments with modified Grade 2 Braille codes to determine their effect on reading speed', *Journal of Visual Impairment and Blindness* **73**(8), 324-328.
- Luce, P., Feustel, T. and Pisoni, D. (1983), 'Capacity Demands in Short-Term Memory for Synthetic and Natural Speech', *Human Factors* **25**, 17-31.
- Lévesque, V., Pasquero, J., Hayward, V. and Legault, M. (2005), 'Display of virtual braille dots by lateral skin deformation: feasibility study', *ACM*

- Transactions on Applied Perception (TAP)* **2**(2), 132–149.
<http://doi.acm.org/10.1145/1060581.1060587>
- Maler, E. and Andaloussi, J. E. (1995), *Developing SGML DTDs : From Text to Model to Markup*, Prentice Hall.
- Marincu, C. and McMullin, B. (2004), ‘A Comparative Assessment of Web Accessibility and Technical Standards Conformance in Four EU States’, *First Monday* **9**(7).
<http://tinyurl.com/q5tcy>
- McGookin, D. K. and Brewster, S. A. (2004), ‘Understanding concurrent earcons: Applying auditory scene analysis principles to concurrent earcon recognition’, *ACM Transactions on Applied Perception (TAP)* **1**(2), 130–155.
- McMullin, B. (2002a), ‘Users with Disability Need Not Apply? Web Accessibility in Ireland’, *First Monday* **7**(12).
<http://tinyurl.com/pq43f>
- McMullin, B. (2002b), ‘WARP: Web Accessibility Reporting Project Ireland 2002 Baseline Study’.
<http://tinyurl.com/gxb86>
- Morley, S., Petrie, H., O’Neill, A.-M. and McNally, P. (1998), Auditory navigation in hyperspace: design and evaluation of a non-visual hypermedia system for blind users, in ‘Proceedings of the third international ACM conference on Assistive technologies’, ACM Press, pp. 100–107.
- Mousty, P. and Bertelson, P. (1985), ‘A study of braille reading: 1. Reading speed as a function of hand usage and context’, *The Quarterly Journal of Experimental Psychology* **37a**, 217–233.
<http://www.braille.org/papers/analys/analys.html>
- Mynatt, E. D. and Edwards, W. K. (1992), Mapping GUIs to Auditory

- Interfaces, *in* 'Proceedings of the 5th annual ACM symposium on User interface software and technology', ACM Press, pp. 61–70.
- Nemeth, A. (1972), *Nemeth code of braille mathematics and scientific notation*, American Printing House for the Blind.
- Oogane, T. and Asakawa, C. (1998), An Interactive Method for Accessing Tables in HTML, *in* 'International ACM Conference on Assistive Technologies', pp. 126–128.
<http://doi.acm.org/10.1145/274497.274521>
- OUP (1998), *The Little Oxford Dictionary*, seventh edn, Oxford University Press, Oxford.
- Parente, P. (2004), Audio enriched links: web page previews for blind users, *in* 'Assets '04: Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility', ACM Press, New York, NY, USA, pp. 2–8.
<http://doi.acm.org/10.1145/1028630.1028633>
- Petrucci, L., Harth, E., Roth, P., Assimacopoulos, A. and Pun, T. (2000), WebSound: a generic Web sonification tool, and its application to an auditory Web browser for blind and visually impaired users, *in* 'In Proceedings of the Sixth International Conference on Auditory Display (ICAD 2000)'.
- Pisoni, D., Nusbaum, H. and Greene, B. (1985), Perception of Synthetic Speech Generated by Rule, *in* 'Proceedings of the IEEE', Vol. 73, pp. 1665–1676.
- Pontelli, E., Gillan, D., Xiong, W., Saad, E., Gupta, G. and Karshmer, A. I. (2002), Navigation of HTML tables, frames, and XML fragments, *in* 'Fifth Annual ACM Conference on Assistive Technologies', Web accessibility, pp. 25–32.
<http://doi.acm.org/10.1145/638249.638256>

- Pontelli, E. and Son, T. C. (2002), Accessible interfaces: Planning, reasoning, and agents for non-visual navigation of tables and frames, *in* 'Proceedings of the fifth international ACM conference on Assistive technologies', ACM Press, pp. 73–80.
- Pontelli, E. and Son, T. C. (2003), 'Designing intelligent agents to support universal accessibility of E-commerce services', *Electronic Commerce Research and Applications: Selected Papers from the International Workshop on Software Agents for Business Automation* **2**(2), 147–161.
- Pontelli, E., Xiong, W., Gupta, G. and Karshmer, A. I. (2000), A domain specific language framework for non-visual browsing of complex HTML structures, *in* 'Assets '00: Proceedings of the fourth international ACM conference on Assistive technologies', ACM Press, New York, NY, USA, pp. 180–187.
- Rabbitt, P. (1966), 'Recognition memory for words correctly heard in noise.', *Psychonomic Science* **6**, 383–384.
- Rabbitt, P. (1968), 'Channel-capacity, intelligibility, and immediate memory', *Quarterly Journal of Experimental Psychology* **20**, 241–248.
- Ramakrishnan, I. V., Stent, A. and Yang, G. (2004), Hearsay: enabling audio browsing on hypertext content, *in* 'Proceedings of the 13th international conference on the World Wide Web', pp. 80–89.
<http://doi.acm.org/10.1145/988684>
- Raman, T. V. (1994), Audio System For Technical Readings, PhD thesis, Cornell University, Computer Science Department.
<http://citeseer.ist.psu.edu/63309.html>
- Raman, T. V. (1996a), Emacspeak-a speech interface, *in* 'Proceedings of the SIGCHI conference on Human factors in computing systems: common ground', ACM Press, pp. 66–71.

- Raman, T. V. (1996*b*), Emacspeak-direct speech access, *in* 'Proceedings of the second annual ACM conference on Assistive technologies', ACM Press, pp. 32–36.
- Regan, B. (2005), 'Best Practices for Accessible Flash Design'. Macromedia, Inc.
<http://www.adobe.com/macromedia/accessibility/whitepapers/>
- Roberts, J., Slattery, O., Kardos, D. and Swope, B. (2000), New technology enables many-fold reduction in the cost of refreshable Braille displays, *in* 'Proc. Conf. ASSETS: ACM SIGACCESS Conference on Assistive Technologies', Association for Computing Machinery (ACM), pp. 42–49.
<http://doi.acm.org/10.1145/354324.354335>
- Rosson, M. B. (1985), Listener training for speech-output applications, *in* 'Proceedings of the SIGCHI conference on Human factors in computing systems', ACM Press.
- Savidis, A. and Stephanidis, C. (1998), 'The HOMER UIMS for dual user interface development: Fusing visual and non-visual interactions', *Interacting with Computers* **11**(2), 173–209.
- Schroeder, F. K. (1996), 'Perceptions of Braille Usage by Legally Blind Adults', *Journal of Visual Impairment and Blindness* **90**, 210–218.
<http://www.braille.org/papers/jvib0696/vb960310.htm>
- Shajahan, P. and Irani, P. (2005), Manipulating Synthetic Voice Parameters For Navigation In Hierarchical Structures, *in* 'Proceedings of ICAD 05-Eleventh Meeting of the International Conference on Auditory Display, Limerick, Ireland,'.
- Shneiderman, B. (2000), 'The limits of speech recognition', *Commun. ACM* **43**(9), 63–65.
<http://doi.acm.org/10.1145/348941.348990>

- Sinclair, R. (2000), 'Microsoft Active Accessibility: Architecture'. Microsoft Corporation.
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnacc/html/actvaccess.asp>
- Sloan, M. (2001), 'Web Accessibility and the Disability Discrimination Act', *Journal of Information, Law and Technology*, **2001**(2).
<http://elj.warwick.ac.uk/jilt/01-2/sloan.html>
- Smith, A. C., Cook, J. S., Francioni, J. M., Hossain, A., Anwar, M. and Rahman, M. F. (2003), Nonvisual tool for navigating hierarchical structures, in 'ACM SIGACCESS Accessibility and Computing', Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility Assets 04', ACM Press, pp. 133–139.
- Sowa, J. F. (1984), *Conceptual structures: information processing in mind and machine*, Addison-Wesley, Reading, Mass.
- Spiliotopoulos, D., Xydias, G., Kouroupetroglou, G. and Argyropoulos, V. (2005), Experimentation on Spoken Format of Tables in Auditory User Interfaces, in 'Proc. of the 3rd Int. Conference on Universal Access in Human-Computer Interaction'.
- Sproat, R. (1996), *Text Interpretation for TTS Synthesis*, Cambridge University Press, London/New York, chapter 5.3.
<http://cslu.cse.ogi.edu/HLTsurvey/ch5node5.html>
- Stevens, R. D. (1996), Principles for the Design of Auditory Interfaces to Present Complex Information to Blind People, PhD thesis, Department of Computer Science.
- Takagi, H., Asakawa, C., Fukuda, K. and Maeda, J. (2002), Site-wide Annotation: Reconstructing Existing Pages to be Accessible, in 'Proceedings of the fifth international ACM conference on Assistive technologies', ACM Press, pp. 81–88.

- Taylor, E. A. (1966), *The Fundamental Reading Skill as Related to Eye-Movement Photography and Visual Anomalies*, second edn, Charles C. Thomas, Springfield, Illinois.
- UK Cabinet Office (2005), 'eAccessibility of public sector services in the European Union'. London.
<http://tinyurl.com/96szo>
- Umeda, N., E., M., T., S. and Omura, H. (1968), Synthesis of fairy tales using an analog vocal tract, *in* 'Proc. 6th Int. Cong. Acoust., Tokyo, Japan'.
- van Bezooijen, R. and van Heuven, V. (1998), *Assessment of Synthesis Systems*, Vol. 111, Walter De Gruyter Inc, pp. 167–249.
- W3C (1999a), 'HTML 4.01 Specification'. World Wide Web Consortium, D. Raggett, A. Le Hors and I. Jacobs, Eds.
<http://www.w3.org/TR/REC-html40/>
- W3C (1999b), 'Web Content Accessibility Guidelines 1.0'. World Wide Web Consortium, W. Chisholm, G. Vanderheiden and I. Jacobs, Eds.
<http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505>
- W3C (2000a), 'Authoring Tool Accessibility Guidelines 1.0'. World Wide Web Consortium, J. Treviranus, C. McCathieNevile, I. Jacobs, Jan Richards, Eds.
<http://www.w3.org/TR/2000/REC-ATAG10-20000203>
- W3C (2000b), 'XHTML Basic'. World Wide Web Consortium, M. A. Baker, M. Ishikawa, S. Matsui, P. Stark, T. Wugofski and T. Yamakami, Eds.
<http://www.w3.org/TR/2000/REC-xhtml-basic-20001219>
- W3C (2002a), 'User Agent Accessibility Guidelines 1.0'. World Wide Web Consortium, I. Jacobs, J. Gunderson and E. Hansen, Eds.
<http://www.w3.org/TR/2002/REC-UAAG10-20021217/>

- W3C (2002*b*), 'XHTML 1.0: The Extensible HyperText Markup Language (Second Edition)'. World Wide Web Consortium, S. Pemberton, Ed.
<http://www.w3.org/TR/2002/REC-xhtml1-20020801>
- W3C (2004*a*), 'Document Object Model (DOM) Level 3 Core Specification'. World Wide Web Consortium, A. Le Hors, P. Le Hégarret, L. Wood, G. Nicol, J. Robie, M. Champion and S. Byrne, Eds.
<http://www.w3.org/TR/DOM-Level-3-Core/>
- W3C (2004*b*), 'Extensible Markup Language (XML) 1.0 (Third Edition)'. World Wide Web Consortium, Cambridge, MA, T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and Francois Yergeau.
<http://www.w3.org/TR/2004/REC-xml-20040204/>
- W3C (2005), 'Cascading Style Sheets, level 2 revision 1: CSS 2.1 Specification'. World Wide Web Consortium, B. Bos, T. Çelik, I. Hickson and H. Wium Lie, Eds.
<http://www.w3.org/TR/2005/WD-CSS21-20050613>
- W3C (2006), 'Web Content Accessibility Guidelines 2.0'. World Wide Web Consortium, B. Caldwell, W. Chisholm, J. Slatin and G. Vanderheiden, Eds.
<http://www.w3.org/WAI/GL/WCAG20/WD-WCAG20-20060427/>
- Waddell, C. D. and Urban, M. (2000), 'An Overview of Law and Policy for IT Accessibility'.
<http://www.icdri.org/CynthiaW/SL508overview.html>
- Walshe, E. and McMullin, B. (2004), Accessing Web Based Documents Through a Tree Structural Interface, *in* 'Proceedings of ICCHP 2004', Springer Lecture Notes in Computer Science (LNCS), Springer.
<http://tinyurl.com/r2muo>
- Walshe, E. and McMullin, B. (2006), Browsing Web Based Documents through an Alternative Tree Interface: The WebTree Browser, *in* 'Pro-

- ceedings of ICCHP 2006', Springer Lecture Notes in Computer Science (LNCS), Springer.
- Waterworth, J. A. and Thomas, C. M. (1985), Why is synthetic speech harder to remember than natural speech?, *in* 'Proceedings of the SIGCHI conference on Human factors in computing systems', ACM Press.
- Yesilada, Y., Stevens, R., Goble, C. A. and Hussein, S. (2004), Rendering tables in audio: the interaction of structure and reading styles, *in* 'The Sixth International ACM SIGCAPH Conference on Assistive Technologies (ASSETS)', pp. 16–23.
<http://doi.acm.org/10.1145/1028635>
- Zajicek, M. and Powell, C. (1997), 'Building a Conceptual Model of the World Wide Web for Visually Impaired Users', *Contemporary Ergonomics* pp. 270–275.
- Zajicek, M., Powell, C. and Reeves, C. (1998*a*), 'Orientation of Blind Users on the World Wide Web', *Contemporary Ergonomics* .
- Zajicek, M., Powell, C. and Reeves, C. (1998*b*), A Web navigation tool for the blind, *in* 'Proceedings of the third international ACM conference on Assistive technologies', ACM Press.

Appendix A

WebTree Application User Manual

Introduction

Many Internet access solutions for the blind are in reality just dedicated audio interfaces that serve as add-on applications to out of the box visual browsers. However, it must be noted that there are huge differences in the type of methods of interaction between using a predominantly speech interface and that of visual interaction. The human eye is expertly capable of scanning through the document to establish what is deemed to be the important page content. This is done by examining the spatial relationships between elements and through the use of visual cues, such as colour and emphasis, included in the text by the author. Unfortunately, due to the serial nature of speech technology, this is not possible with a purely speech output interface, for it is only possible to examine a single point in the document at any given time. To avoid any confusion on behalf of the user when reading elements that depend on their spatial layout for easy comprehension, many of these applications output the content in the linear format in which it appears in the mark-up. Thus, the user must navigate through all elements that appear in the file prior to the main content, before it can be read.

The WebTree application has been written to explore the use of an alternative, tree structural approach to the displaying of content, when applied to the portrayal of Internet based documents. Through this application, we wish to discover any possible advantages to the user that can be associated with this system. We are especially concerned with the effect this approach may have on the efficiency at which the user can navigate to and assimilate information.

As well as the tree like arrangement of the displayed XHTML elements, The WebTree application has been augmented with some rather powerful methods to search for content. These alternative methods were included, so that we could also experiment with alternative approaches used to navigate through these documents to find the required information. As well as searching for plain text, the user can search for an individual XHTML element, or limit the text search to only find instances that occur inside specific XHTML elements. Thus if a document is properly marked up with structural elements, such as `<h*>` and `` elements, instead of having their visual appearance created with changes in font, the user should also be able to navigate to the areas of the page that the author deemed important.

User Interface

The main document view of the application is based on the idea of exposing the tree structural arrangement of elements contained in the XHTML file. The view consists of a combination of buttons representing the XHTML mark-up elements, which when pressed, expand or remove their content from the display, and plain text content from elements that have already been expanded. The expansion controls are made up of two components. The first being the button that controls the expansion or text removal process. The second component is necessary to provide the user with some contextual information about the element under point. This includes the name of the element e.g., "ul" and the content of the elements title attribute if it exists. In the case of paragraph elements, the first number of characters contained

in the element is exposed automatically. The amount of characters to be displayed can be customised in the `wt-display` group. To avoid confusion on the part of the user, the tree controls for a specific XHTML element are placed on a line on their own.

Not all elements have tree controls associated with them. This functionality is just limited to those elements that are designated as *block* elements in the XHTML basic specification (W3C, 2000b). All designated *inline* elements are automatically expanded in the display by default, without any tree expansion controls.

In designing the WebTree application, we have tried to keep the user interface as simple as possible. The `enter` key is the main key used to dynamically change the state of the display, for this is the key that performs operations on buttons, whether they are tree control buttons, checkboxes or radio buttons. However, the search functionality does also have a major effect on the display, for, once an instance of the search string is found, the element containing the string is automatically expanded into the display.

In the instances where commands require just one keystroke for activation, such as performing an element search, we just use the upper case version of the same character to reverse the action, to avoid having a huge number of keys for the user to learn. E.g. `e` searches forward for an element, and `E` performs the same type of search backwards.

Customisation

All document views generated by the WebTree application are controlled by a set of customisable variables stored in the file `wt-custom.el`. The state of all of these variables can be altered through the emacs supplied `customization` package. The custom groups, `wt-element-expansion` and `wt-display` are the two groups responsible for the types of alternative view generated. The `wt-display` group variables govern such things as line length and the amount of text to be displayed when an element has not already been expanded. But with the `wt-element-expansion` group we can govern

the generation of the alternative document views. In this group we can specify which elements possess tree controls in the screen text, and those elements that we wish to have automatically expanded when encountered by the application. In addition, we can remove entire sub trees of elements from the display, by setting both the `display` variable and the `expansion` variable for a specific element to nil.

To access the `wt-tree` group, which contains links to all the sub groups of customisation variables for the WebTree application, from within the application itself, you must press the `c` key.

To access the `wt-element-expansion` group automatically, just press the `C` key.

Forms Interface

Unlike some other Internet solutions for the blind, there is no specialised forms mode that the user must enter, so as to be able to fill out online form information. Instead, once focus has been placed on a form field / control, the application automatically alters the key mappings to allow for the correct information to be inputted. For example, when focus is placed on a `text` field, *any* letter pressed will result in that character appearing in the space allocated for the specific field. In the case where the character pressed invokes a specific WebTree related command, the command call is overwritten and the character is placed in the text field as normal.

Each form field / control appears on a separate line from any text contained within the form construct. Also, it should be noted that the start of a form field usually coincides with the beginning of the line. As focus is placed on the form field / control, its presence is announced. When *tabbing* to the individual form fields, any `<label>` information associated with the element is also read by the application.

To change the state of both checkbox and radio button elements, move focus to the element and press the `enter` key.

To submit the form information, focus must be moved to the submit

button, and then the `enter` key must be pressed.

Table Navigation

In terms of the WebTree application, the table construct is physically portrayed in the display in a linearised fashion, with some additional navigational functionality provided to ensure easy navigation from cell to cell along the spatial relationships between individual table cells. Navigation is provided to the cells to the immediate left or right of the current cell in a given row, or up or down to the next cell in the column under focus. If the current cell spans more than one row, then navigating to cells to the left or right of the element is only permitted along the first row in which the cell appears. Column navigation is not affected in this situation. Similarly, in the case where the current cell spans more than one column, navigation away from the cell is only permitted along the first column spanned by the cell. In this situation, row navigation behaves as normal.

As we navigate from cell to cell along a specific row or column, the header information for the current cell does not automatically appear in the spoken output of the application. Instead, the option is available for the user to simply navigate to the header cell associated with the current cell. Navigation to both the row header and column header cells, if they are present, is possible from the current cell. Once the user has read the related header information, there is a specific table *back* function that reverts focus back to the cell from which the original jump took place.

A list of table navigation commands and their related keystrokes are as follows:

- 8: Column up
- 2: Column down
- 4: Row left
- 6: Row right

- 9: Move to column header
- 3: Move to row header
- 7: Go to beginning of table
- 1: Go to end of table
- 0: Go to previous cell from Header jump

These keys were chosen to allow for the keypad to be used for simple table navigation.

Searching

There are three different forms of search functionality provided by the Web-Tree application. The first method to be included is a regular text search facility, which incrementally searches the content for a user supplied text string. As each character is inserted by the user, the application just searches for the next instance of the current string. The search can be ended by pressing the `enter` key, or by changing the focus, through the use of the arrow keys or keys such as `home` or `end`.

The second type of search functionality restricts the regular incremental search method to only look for instances of a specific text string within the contents of a particular XHTML element sub tree. For example, we can limit the search to only search elements for a specific word or phrase. In this case, all other instances of the search string that are not contained within an element sub tree are ignored.

To achieve this form of search functionality, we must first notify the application as to which element we wish to restrict the search to. This is done by calling the `wt-set-search-restriction` function. When this method is invoked, the user is prompted to insert the element name. Alternatively, the user can just hit the `UpArrow` key, to view the previous restriction element selections. Once the restriction has been set, the incremental search functionality can be invoked as normal.

The final piece of search functionality allows the user find and navigate to any individual XHTML element in a given document. However, to achieve this, alternative search methods to those used in the incremental text search are invoked. Once these functions are called the user is prompted to type in the name of the element to search for, or alternatively, using the arrow keys, the user can select an element previously searched for by the application. To activate the search the `enter` key must then be pressed.

A list of the search related keystrokes and their actions are as follows:

- `Control-s`: Incrementally search forward for text.
- `Control-r`: Incrementally search backward for text.
- `s`: Set search restriction.
- `S`: clear search restriction.
- `e`: Search forward for element.
- `E` Search backwards for element.

Document Retrieval and Browser History

When the `wt` start up command is invoked, the browser automatically loads the web page pointed to by the `wt-default-homepage` variable, which can be set through the emacs specific `customization` system. If this variable is not set, then the application uses the value of the `www_home` environment variable to act as the home page for the current user. As each subsequent document is retrieved, its URL is added to the browser's history list for later perusal by the user.

Navigation through the browser history can be achieved in two different ways. The first method involves the user invoking specialised history navigation functions, through the use of application specific keystrokes. A list of these keystrokes and their associated actions can be found below. The second method for navigation only occurs when either the `wt-fetch`, or the

`wt-open-local-file` functions are invoked. The user is prompted to type in a URL, or alternatively, using the arrow keys, the user is able to navigate back through the previous URLs visited in the current browser session. Pressing the `enter` key on one of these URLs / `file` path names will result in the retrieval and displaying of the document concerned.

List of keystrokes and their associated actions:

- `o`: `wt-open-local-file`. Opens an XHTML file stored locally.
- `Control-o`: `wt-fetch`. Retrieve a URL from the web.
- `backspace`: General back function
- `control -b`: Back to previous page
- `control-f`: Page forward
- `b`: Undo relative link

Index of WebTree Keystrokes and Commands

Keystroke	Command
b	Undo relative link
c	Go to element display customise group
C	Main WebTree customise group
e	Search forward for XHTML element
E	Search backward for XHTML element
l	Move to next hyperlink
L	Move to previous hyperlink
o	Open locally stored XHTML file
p	Read XHTML element under point
r	Redraw document
s	Set search restriction for restricted incremental text search
S	Clear search restriction for restricted incremental text search
t	Move to next tree control element
T	Move to previous tree control element
backspace	General back function
control-b	Page back
control-f	Page forward
Control-o	Retrieve URL
control-s	Incrementally searches forward for text
control-r	Incrementally searches backward for text
control-leftarrow	Back word
control-rightarrow	Forward word
control-home	Beginning of buffer
control-end	End of Buffer
tab	Forward to next element on the tab order
shift-tab	back to previous element on the tab order
0	previous cell from Header jump
1	End of table
2	Column down
3	Jump to row header
4	Row left
5	Speak current cell under point
6	Row right
7	Beginning of table
8	Column up
9	Jump to column header

Appendix B

WebTree Tutorial

This is a short tutorial to help the user gain a basic understanding of how to operate the WebTree application. However, for a more in-depth description of the WebTree application's functionality, see the user manual in appendix A.

Tree Element Expansion

The initial display can contain a mixture of some tree controls, representing XHTML *block* elements, and some plain text content from elements that have already been expanded. The designated *inline* XHTML elements are always expanded automatically by default in the screen text, without the presence of any tree expansion controls.

To expand an element of the tree, the user must activate the button that is placed at the beginning of the line representing the element's tree control. This button can appear in two different forms. If the button is represented by [+], then the element still exists in its collapsed state. To expand this element, move focus to the button and press the **enter** key. The amount of information to be exposed by this action can vary greatly depending on the setup and the type of element to be expanded. That is, how much of the sub tree is to be shown automatically due to the key press, depends greatly on the custom variables. See the customisation section for more details.

If the button is represented by [-], then the element already exists in its expanded state. Thus, hitting the **enter** key when the button is under point shall remove the elements entire contents from the displayed text.

Before moving on to the rest of the tutorial, it is advisable for the user to familiarise themselves with the methods for both the expansion and the collapsing of elements.

Customisation

The list of designated XHTML block elements to be automatically expanded when encountered by the WebTree application is dictated by the users own preferences. In the customisation group, `wt-element-expansion`, the user can select whether they would prefer to have an individual element automatically expanded by default. The naming convention for the expansion variables is the string “wt-custom-display-” followed by the name of the XHTML element. To automatically expand an element, just ensure its expansion variable value is set to `t`. Set it to `nil` otherwise. To go directly to the `wt-element-expansion` group, press the letter **C**.

In addition, using the same customise buffer, you can set the display-element variables for each XHTML block element. These variables determine whether the tree controls for a given element actually appear in the screen text at the point where the element is presented. This option is available for those cases where the user prefers to always automatically expand a specific element, and would rather reduce the amount of clutter on the screen, by eliminating its tree expansion control from the display.

To have these changes take effect on the browser settings, the user must also select either the **save for current session**, or **save for future sessions**, options that appear near the top of the customise buffer.

We recommend that you spend some time experimenting with these different settings, until you have a good idea of how to generate an optimal view for your requirements. For example, try changing the settings for the `<p>` element and the `` element.

To get back to the document that you were using after making alterations to the customisation buffer, either activate the `finish` button or close the customize buffer using the emacs `kill-buffer` command bound to the key sequence `control-x` followed by `k`. Alternatively, you can use the emacs `switch-buffer` bound to `Control-xb` key combination.

Once back in the WebTree related document, press the letter `r` to redraw using the new settings.

Setting both the display related variable and the expansion related variable for the same element to a value of `nil`, results in the element and its entire sub tree not appearing at all in the display.

Using XHTML Forms

When filling out forms with the WebTree application, there is no additional specialised forms mode that the user must enter before access to the fields is granted for editing purposes. Instead, when the focus is placed on a form field, the application automatically alters the mapping of keystrokes from their prescribed WebTree settings back to the normal character insert function. However, once focus has been removed from the form field / control, the key mappings revert to point to the WebTree specific keystroke mappings.

Try filling out the following form. This is just a sample form and is not intended to be submitted to anywhere.

Remember, to change the state of a radio button or a checkbox, just move the focus to it and press the return key.

Table Navigation

Table navigation is quite simple Under the WebTree application interface, once you use the keypad keys to manoeuvre around through the different cell-to-cell relationships. Remember you need to have the num lock key set to allow numbers to be entered otherwise the keys are bound to the same functions as the arrow keys and the keys in the page-up and page-down

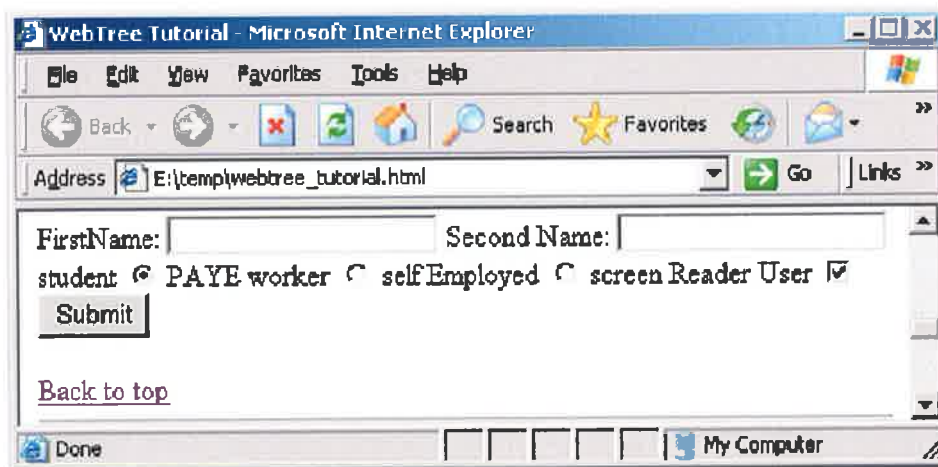


Figure B.1: A simple form construct

group.

A full list of the table navigation keys and their functions is available in the user manual. However, if you do not have access to the user manual, you can use the emacspeak speak specific `emacspeak-learn-mode` function bound to the `control-e control-h` key combination to discover which function is bound to each key. To leave this mode, type the emacs `cancel` keystroke, which is bound to `control-g`.

Here is a simple table construct in which you can try out the different table navigation functions.

Searching

The regular incremental text search functionality bound to the keystrokes `control-s` and `control-r`, does not require any elaborate description of its usage. As each character is typed in to the search string by the user, the application tries to find the next instance of the string in its current state. Each time an instance of the search string is found, the containing tree elements are automatically expanded into the audio rendering, and focus jumps to that point.

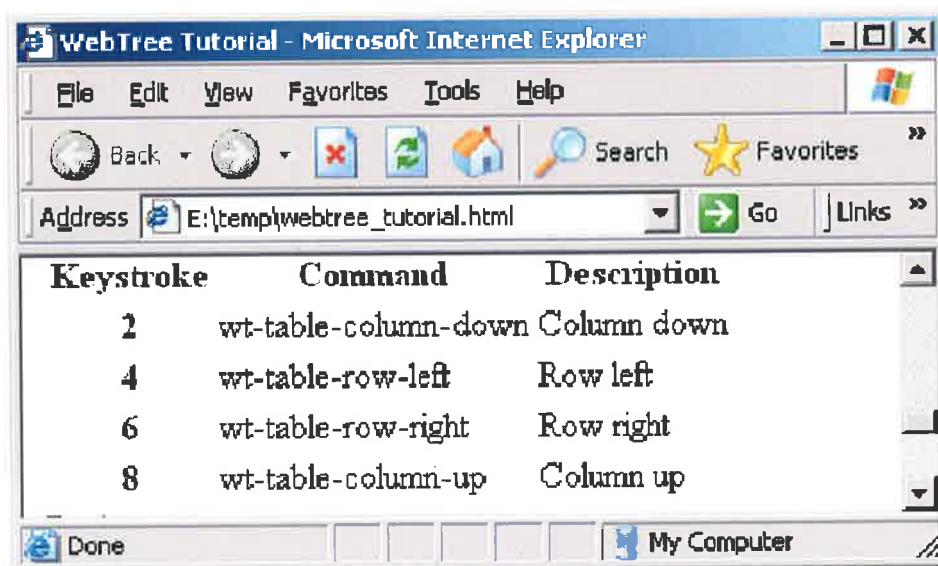


Figure B.2: A simple table construct

The types of search that require a much greater description of their workings are the XHTML element search functions, and the restricted incremental text search functionality.

The element specific search allows the user to jump to the position of any XHTML element, once it has been included in the document's mark-up structure. A number of web browsing solutions for the blind have implemented a limited form of this type of search functionality, i.e., the user is allowed to jump directly to such elements as `<h*>` elements or to `<table>` constructs. However, under the system implemented in the WebTree application, all elements contained in the document are searchable. For example, try searching this document for the next paragraph `<p>` element, or search for a `` element. To perform a search of this kind, press the letter `e`. If searching backwards use `E`. Once this is done, you will be prompted for the name of an element. Type in the name of the element to search for, excluding the less-than and greater-than signs. Then hit the `enter` key. If the element is present in the direction in which you are searching, the element is automatically expanded into the display and focus is placed on the found

instance.

The final type of search is the restricted incremental search. This works by only searching inside the sub trees of a named element for the relevant text. For example, try restricting the search to be inside a <p> element and search for a word or phrase you know to be in the document. If the search string is found but outside a paragraph sub tree, then it is completely ignored by the application.

To restrict the search, press the **s** key. You will be then prompted for the name of the element to limit the search to. Once this has been entered, press **enter**. To invoke the search use the same keystrokes as the normal incremental search, **control-s** or **control-r**. Remember that if you wish to go back to using the normal incremental search, you must unset the search restriction using the **S** key.

Appendix C

Additional User Evaluation Questionnaires

C.1 Preliminary Evaluation Questionnaire

It is necessary to gather some profile information about the users testing the WebTree application, so we can establish a general profile covering the entire group of test participants. Thus, we request that you fill out the following form information. Note: No identifiable information about a single user shall be included in the final report.

Thank you for your participation.

Please place answers after the question in the text. Where there are multiple choice answers provided for a question, mark the one most suitable.

1. User profile information (all fields are optional):

- First Name:
- Last Name:
- Age?
- Job Title:

2. How would you describe your current level of eyesight?

- Blind:
- Partially sighted:
- Fully Sighted:

Screen Reader Usage

3. Are you a screen reader user?

- Yes:
- No:

4. If so, which screen reader do you use on a regular basis?

5. Have you used any other screen readers recently?

- Yes:
- No:

If so, which ones?

6. What would be your level of proficiency with screen reader usage?

- Advanced:
- Intermediate:
- Novice:

Technical Knowledge

7. Which operating system do you use mostly?

8. Have you ever used the Linux operating system?

- Yes:
- No:

9. Have you ever used emacs?

- Yes:

- No:

10. Have you ever used emacspeak?

- Yes:

- No:

Internet Usage

11. Do you use the Internet on a regular basis?

12. If so, for what functions? E.g. do you need it for research purposes at work?

13. Which web browser -screen reader combination do you normally use?

14. How would you describe your knowledge level of HTML?

- Advanced:

- Intermediate:

- Basic:

- Non-existent:

15. Have you ever used the WebTree application before today?

- Yes:

- No:

C.2 After Evaluation Questionnaire

This questionnaire is used in the WebTree user testing process to try to establish a better idea of the user opinions about, and expectations of the types of functionality included in the application.

Thank you for participating.

Please answer the following questions. Feel free to add any comments you might have relating to a question after the question in the text.

Please place answers after the question in the text. Where there are multiple choice answers provided for a question, mark the one most suitable.

1. Do you think that through the examination of the tree structural view of the document, you can easily get a general idea of the subject matter of the content being presented?

- Yes
- No

2. Do you think that through this type of document view, you can navigate more easily to the main content of a given web page?

- Yes
- No

3. When using this application, can you navigate through the content of web pages successfully?

- Yes
- No

4. In general, do you think that viewing the document through the tree view interface would have any major advantages over viewing the same document in a linearised form? Please give a reason for your answer.

5. In general, do you think that viewing the document through the tree view interface would have any major disadvantages over viewing the same document in a linearised form? Please give a reason for your answer.

6. Is the ability to configure the XHTML elements that appear in the document view without any tree controls, an advantage?

- Yes

- No

Why?

7. Does the ability to have certain elements expanded automatically have any advantages for the user?

- Yes
- No

Why?

8. When displaying the tree controls for an element, should the name of the HTML element be used, or would it be better to use the full name for the element? For example, in the case of a `` element, should the user be told that an "UL" element or an "unordered list" is present. Please give a reason for your answer.

9. For elements not expanded, is it important to display in the tree control information, any additional data about the element if it is present. For example, the content of a `title` attribute or in the case of a table, the content of the `summary` attribute.

- Yes
- No

10. In the case of unexpanded paragraph elements, the `<p>` element, is it beneficial to show a small portion of its contained text in the display, so as to alert the user of the type of subject matter it contains.

- Yes
- No

Please give a reason for your answer.

Forms

11. Did you like the way in which HTML forms were presented in the WebTree application?

- Yes
- No

Please give a reason for your answer.

12. Is the lack of a specific forms mode an advantage or disadvantage? Why?

13. Do you like the way in which the key mappings change so that character input is received by the application once a form field is under focus, and how it reverts back to using the general application key mappings once focus has been removed from the field?

Tables

14. Do you like the way in which table constructs are presented under the WebTree application?

15. Is the choice of using the keypad keys to provide table navigation functions a good one?

- Yes
- No

Why?

16. Would you prefer if the header information for each table cell were to be read automatically every time you navigated to that cell?

- Yes
- No

17. Is the ability to jump to the header cell for any given table cell and back again of any benefit?

18. As a preference, would you prefer to have the header information read automatically, or have the ability to jump to the header information when it is required?

19. Did the table navigation commands operate in a manner that you would have expected?

- Yes
- No

Searching

20. Did you try to use the “search for an element” functionality?

- Yes
- No

21. Do you like the ability to search the document for any specific element?

- Yes
- No

22. Do you believe this type of search to be beneficial?

- Yes
- No

If you have any comments relating to the specific element search facility, whether positive or negative, please list them here:

1.

2.

3.

23. Did you try to limit the normal text search to only search for text inside a specific XHTML element?

- Yes
- No

24. Do you like the ability to limit the search in this manner?

- Yes
- No

25. Do you think this type of functionality to be of benefit?

- Yes
- No

Comments

If you have any additional comments about the WebTree application, please add them here.

Negative Comments

1.

2.

3.

4.

5.

Positive Comments

- 1.
- 2.
- 3.
- 4.
- 5.

Appendix D

Braille

This appendix considers the use of Braille as the primary mode of written interaction and how technology might employ this medium to provide access to electronically stored information. Initially, a description of Braille and its advantages/disadvantages compared to alternative interaction modalities is presented. Some statistics relating to the current usage levels of Braille both in Ireland and from an international perspective are included. Also featured are the results of a number of studies concerning the rate of content assimilation through Braille. A number of distinct reading strategies to increase reading rates are also described. Finally, a number of mechanisms used to generate Braille content are discussed.

D.1 Brief Description

Braille is a tactile method of encoding information used by the blind to perform written communication, dating back to the early 19th century. Braille characters do not try to mimic the shape of their print equivalents, for coding in this manner would have required much additional space for the written material. Instead, each character is fashioned from a combination of up to six raised dots, positioned in two adjacent vertical columns of three. The dots inhabiting the first column are recognised as dot 1, dot 2 and dot 3, whereas dots 4, 5 and 6 complete the second column. Each letter definition

is indicated by the positioning of different numbers of dots in alternate combinations. For example, the letter 'e' is represented by dots 1 and 5 and the 'g' character includes dots 1,2,4 and 5.

Books transcribed into Braille, are usually quite large and rather bulky in nature, often requiring the division of content into numerous volumes. This is partly due to the space required to represent each Braille character. It is only possible to place 40 or so characters on a given line, whereas with printed text, the inclusion of up to 75 characters is frequently attainable. In addition, it is necessary to insert a limited quantity of white space as a divider between lines of text, to ensure lines may be deciphered and read with ease. Therefore, there often only exists approximately 27 lines of actual content on a standard Braille page. A second reason for this bulkiness is the thickness of the paper recommended for Braille production; to sustain the dots over a reasonable time frame, a card varying between 100gsm and 130gsm in weight is required. To illustrate this point even further, the content of an average printed desktop dictionary such as *The Little Oxford Dictionary* (OUP, 1998) encompasses 38 volumes of Braille.

The presentation of Braille content is often subject to alternative formatting rules to the visual structuring of text. These rules serve to both increase the readability of Braille, and ensure its presentation is further compacted. An example of such a rule is evident when signifying the beginning of a new paragraph. In printed text, such blocks of content are generally separated by at least one line of white space. Whereas in Braille, the white space line separator is removed and the initial word is indented by two spaces. In general, the insertion of blank lines is reserved to signal larger content dividers, such as the beginning of a new section, or chapter.

To aid the readability of Braille documents, additional spaces that would normally appear in printed text after punctuation marks are removed and replaced by a single space character. When large segments of white space appear on a line, difficulties can arise in determining whether the next block of text encountered by the reader's fingers belongs to the current row. This

can impact on the speed at which content can be assimilated. Both in Ireland and Britain, English language Braille formatting is governed by a set of rules produced by the Braille Authority of the United Kingdom (BAUK, 2004).

D.2 Louis Braille: Biographical note

Louis Braille, the inventor of the Braille reading system for the blind, was born in Coupvray, a small town not far from Paris in 1809. He was the son of a harness maker, and his blindness was directly attributable to his father's occupation. It occurred as the result of an accident in his father's workshop when he was four years old, in which he managed to pierce his eye whilst playing with an awl, destroying the eye completely. Some time later, infection set in as a result of the accident and transferred to the remaining eye damaging it irrevocably.¹

For a number of years, he received a limited education whilst attending a local school, however, it was obvious that without the ability to read and write, the pace of learning was rather constrained. At the age of 10, Braille received a scholarship to leave Coupvray for Paris to be educated at the Royal Institution for Blind Youth, where he was taught practical skills such as chair caning and slipper making. Skills to provide him with an income so as he wouldn't have to spend his life begging on the street. This was a common fate for many blind people of the time. Whilst attending the Royal Institution for Blind Youth, Braille learned to read using a system of large embossed print letters. However, books were bulky and rather difficult to produce using this system, for it was necessary to press large copper letters into the underside of the paper resulting in raised lettering.

Braille's method for representing the printed character set was not entirely an original idea. Whilst he was studying in Paris, a soldier named Charles Barbier visited the school he was attending and presented a reading system he had invented called "night writing". It consisted of an arrange-

¹Information in this section is summarised from *The life of Louis Braille* at: <http://tinyurl.com/879kn>

ment of up to 12 raised dots, each combination representing an individual sonorant sound as opposed to individual letters. It had already been rejected by the army as a viable communications mechanism. They attributed their rejection to comprehension difficulties with learning such a complex system. However, a young Louis Braille was not so quick to dismiss this methodology. He realised that with much simplification, it could provide a viable method for enabling blind people to read and write. After much experimentation, a 6-dot combination was settled on, and the first book using Braille's coding system was produced in 1827.

The Braille coding system was not initially accepted as a functional communications mechanism. Many sighted individuals were unable to appreciate the benefits of such a system, for the raised dot combinations didn't attempt to mimic the shapes of the printed character set. One particular head teacher went as far as to enforce a ban on the use of Braille in the school altogether. However, many blind people recognised the advantages of the system and learned to master it in secret. Braille himself didn't live long enough to witness his coding system widely accepted as the premier method of reading and writing for the blind. He died from tuberculosis in 1852, at the age of 43, after struggling with illness for a long time.

D.3 Braille Standards

There are 64 possible dot patterns available when the regular six dot Braille cell is used. These range from zero dots representing a space character to a full complement of six (Foulke, 1982). There are two main recognised forms of Braille output used to encode material written in the English language:

Grade 1 Braille consists of the character set representing the letters of the alphabet, the numbers zero to nine, and a number of punctuation characters. The numbers one to nine are represented by the letters 'a-i', with the letter 'j' used to denote zero. To signify that a number is being presented, a special number sign precedes the characters. Where letters and numbers are mixed, a special letter sign must be inserted between the last

digit and the first letter, except on those occasions where they are separated by white space (BAUK, 2004).

Grade 2 Braille includes the entire grade 1 character set, plus numerous shorthand abbreviations to represent letter groupings and in many cases entire words. These consist of both single character contractions, and a large amount of multi character abbreviations. These serve to reduce the space required to encode the material. They also facilitate an increase in Braille reading speeds. The inclusion of many of these shorthand characters is often subject to a set of contextual rules. For example, the Braille character consisting of dots 2 and 3 provides a number of different meanings depending on its position in a letter grouping. When viewed as the initial character of the assemblage, it denotes the characters **be**. When located as the final character, it means a ;, otherwise it signals the letters **bb**. It is impossible for this character to appear at the end of a line, unless a semi colon punctuation mark is required (BAUK, 2004).

In a number of languages, there is a *Grade 3* level of Braille notation, which is highly abridged. It comes quite close to shorthand, however Grade 3 is too complex for all but a small minority of readers who have a good command of language and a good memory (Bell, 1962).

Throughout the previous two centuries, transcription into Braille has occurred for many different written worldwide languages. However, different language transcriptions may enforce variations in the method for producing specific character sets. This is most evident when a language contains specialised accented lettering, such as those found in Irish and French. These accented letter representations often clash with accepted shorthand abbreviations corresponding to grade 2 Braille for English. Thus, it is important to adhere to the correct writing standards designated for a given language.

Coding systems in Braille have also been devised to provide access to both mathematical and written music material. Due to the abundance of additional printed symbols required to provide clarity under these specialised coding systems, it is frequently necessary to use contextual character group-

ings to represent discrete symbols. These character groupings, coupled with the rules governing their usage ensures a high learning curve to the reading of such complex material. Unfortunately there exists no definitive standard determining how these symbols are to be reproduced.

There are many different standards available for producing *mathematical* symbols. That is, many countries have developed their own standards. For example, Braille mathematical and scientific notation in Ireland and the UK is governed by BAUK (2005), guidelines set down by the Braille authority of the United Kingdom (BAUK), which adapts a linear approach to producing mathematical symbols. Whereas the *Nemeth Code* is the preferred method of representation for North America and Canada (Nemeth, 1972). Using the Nemeth code, attempts are made to fashion a Braille symbol corresponding to the shape of its printed counterpart. The major disadvantage of this approach is as a result of the sheer number of symbols possible. Difficulties can occur in illustrating significant differences in the shapes of mathematical constructs, so that high levels of accurate symbol recognition are attainable. As new symbols possessing only subtle differences to existing constructs are included in printed mathematics, the most relevant Braille translation code may have already been mapped to represent an alternative symbol (Fitzpatrick, 1999). However, due to the WebTree application not attempting to support the display of such complex mathematical data in its current form, an in-depth discussion of the production of such material is out of scope for this thesis.

D.4 Braille Reading and Usage Levels

Although a reasonably high word reading rate for Braille can be gained through reading a passage of text using only one hand, the *most* efficient methods in assimilating Braille material requires the user to use both hands operating in tandem (Mousty and Bertelson, 1985). Mousty and Bertelson suggested that both hands play a part in the assimilation of the information, thus, to a certain extent, limitations are imposed on the reading speed by

the pace of the subordinate (slower) hand. The greatest speeds of reading demonstrated by this study were reached by those individuals for whom a large variance in reading speed between the hands was not evident.

In his review of the literature concerning the research into Braille reading methods available at the time, Foulke (1979) states that a number of different investigations discovered that the optimum reading method involved the two index fingers working in tandem to perform much of the reading task. The initial segment of a line is usually read with both hands side by side, with the remainder of the line read with just the right hand, whilst the left hand attempts to locate the starting position of the next line. Based on the available evidence, Foulke suggests that faster reading is possible when two index fingers are employed independently, because the reader can use the time spent in reading more efficiently. He hypothesised that those Braille readers who use two index fingers read faster because they have learned to involve the two index fingers cooperatively in the same perceptual process (Foulke, 1979).

Many blind people employ more than two fingers when reading Braille. Depending on personal preference, up to eight fingers might in theory be employed. However, much of the recognition process is encapsulated in the index and middle fingers. Foulke (1982) cites (Foulke, 1964) as an experiment determining the sensory capacity of normally unused fingers. He discovered that reading ability diminishes rapidly with progression from the index finger to the little finger for both hands.

Foulke (1982) tells us that evidence exists, (Harris, 1947; Taylor, 1966) which demonstrates that the average silent reading rate for high school students in the United States of America ranges from 250 to 300 words per minute, and reading rates two or three times as fast are sometimes observed. Producing accurate figures relating to Braille reading speed is a controversial topic. The conventional estimate of the mean reading speed of adults is about 100 words per minute (wpm) (Foulke, 1982; Lorimer and Tobin, 1979), but others have claimed that experienced Braille readers achieve rates be-

tween 200 and 400 words/minute (Grunwald, 1966). Legge et al. (1999) tells us that a recent study by Knowlton and Wetzel (1996), measured speeds for experienced Braille readers who were asked to read as quickly as they could. The mean speed was 136 words/min with a range of 65 to 185 words/min. Some of the discrepancy in estimated speeds across studies may be due to differences in text materials, measurement methods and definitions of reading speed (Legge et al., 1999). Legge et al. proposes a system of quantifying reading speed through the summation of the number of characters, verses the time elapsed. This process yielded an average reading speed of approximately 124 wpm, with a minority of subjects reporting reading speeds of between 154 to 232 wpm. However, the reading rates for the majority of test subjects ranged between 82 and 144 words per minute.² It is logical to assume that slower reading speeds would be found if grade 1 Braille were used due to the number of additional characters presented.

Historically, Braille functioned as the principle method of written communication for the blind. However, with the advent of talking computers and books stored on audiocassette, the Braille usage levels amongst the blind community have fallen drastically over the last number of decades. According to data from the American Printing House for the Blind (APH), in 1963, 51 percent of legally blind school children in graded programs, in both public and residential schools in the United States of America, used Braille as the primary reading medium. Another four percent read both Braille and print. APH data demonstrates that the percentage of Braille users has declined steadily since the 1960s, reaching a low of 9.45 percent in 1994. The 1995 data recorded the percentage of Braille users at 9.62 percent (Schroeder, 1996). However, the realisation has occurred that these new technologies cannot completely replace Braille, which has led to a renewed interest in Braille literacy among both educators and consumers. Since the early 1990s, more than half of the individual states in the U.S. have passed Braille literacy laws requiring that Braille instruction be provided to all stu-

²All of these experiments were performed using grade 2 Braille.

dents who can benefit from it. This includes those who are currently able to read print but whose eyesight is medically expected to deteriorate with time. Much of this Braille literacy legislation also requires publishers to provide all textbooks sold under its jurisdiction in electronic formats that can be reproduced as Braille (Johnson, 2004).

The National Council for the Blind of Ireland (NCBI)³ provides a library service to approximately 4,000 blind and partially sighted readers, issuing material in both the Braille and audiocassette media. Of this estimated 4,000 library patrons, approximately 400/10 percent of readers favour receiving subject matter in Braille.

Assimilating information through the use of the Braille medium is not as efficient as employing a visual means of interaction. There are a number of factors that determine this:

1. As in the case of Braille readers reading a block of text, sighted readers assimilate information by moving from word to word using a serial methodology. However, it is thought that a parallel method of pattern matching recognition is employed by visual readers to identify the individual letters comprising each word, resulting in an increase in speed at which information can be assimilated (Larson, 2004). This parallel method for pattern matching the character groupings is difficult for Braille interaction due to the serial nature of the modality.
2. Visual cues such as large fonts or changes in colour signify variations in context to the visual reader, alerting them to the important segments of a given page. This allows the reader to skim through the document, only reading the sections of text they deem important. Much of this multi modal information is lost when reading the same document through Braille. See section D.5, for more information.
3. In the printed medium, extensive usage of spatial formatting is regularly applied to documents to demonstrate contextual changes,

³<http://www.ncbi.ie/>

whereas in Braille, much of this formatting is either non-existent or much less pronounced.

D.5 Issues with the communication of highlighting and other visual cues through Braille

In the printed medium, numerous visual cues are often employed to denote changes in context, or to signal the importance attributed to a piece of text. These cues include variations in the typographical font size, changes in both foreground and background colour, and the use of additional highlighting strategies such as bold and italics. Many of the advantages provided to the visual user by the inclusion of these contextual prompts are lost when viewed through a serial medium. That is, the skimming ability these cues can provide is not available under these conditions. However, it may still be valuable to signal the presence of much of this information in the Braille text to ensure the reader recognises the significance of certain textual fragments in the document.

Accessing content through tactile interaction is much less precise than that of visual communication. Therefore, alternating the size of the Braille dots, to denote changes in the dimension of typographical fonts, is not a viable solution. Rather large changes in the dots' dimensions would be necessary to signal font size alterations to the reader. Problems may also arise in determining the correct line to which these elements belong. This is especially the case where a number of alternative font sizes are positioned adjacent to one another. Handling the large variances in text size could impinge on reading speed, rendering it both slow and laborious. Plus, the implementation of such a system, using conventional methods for Braille production, may not be practical.

There exists only a finite number of different combinations that can be derived from the six possible Braille dots, and the majority of these are already assigned to either a letter, punctuation sign, a digit or a shorthand

abbreviation in grade 2 Braille. Thus, the scope to derive additional Braille characters to infer the highlighting of text is rather limited. One viable solution is to insert additional contextual characters into the written representation to perform this task. In fact, these limitations are so stringent that even character capitalisation must be achieved by inserting additional contextual characters. A single dot 6 preceding a word denotes that the initial letter is capitalised. A second dot 6 before the word indicates that the entire letter grouping is in uppercase (BAUK, 2004).

Another solution would be to define Braille characters in terms of cells containing more than six dots. BAUK (2006) specifies characters in terms of eight dots, i.e., two columns of four. However, characters which fit within the regular six dot constraint are still presented as described previously. Foulke (1982) mentions previous work in which Braille cells with varying numbers of rows and columns were experimented with. It was discovered that the recognition of cells with six rows and six columns, or five rows and five columns was rather slow and error prone. However, cells possessing three/four rows or columns of dots were easily recognisable. It was suggested that cells containing three columns with 'N' rows could be used to denote additional Braille characters. No changes to the representation of characters in the current Braille standard were recommended. That is, additional columns or rows would not be included in their presentation. However, extra dot positions would be used where additional characters currently not specified were required. A standard single column space was recommended to separate each character to avoid confusion due to the variation in numbers of columns. Nevertheless, it should be noted that the use of additional columns of dots has not been incorporated into standard Braille (BAUK, 2004).

Alternatively, some specialised mark-up tags to denote the beginning and end positions of emphasised text could be inserted into the content (Fitzpatrick, 1999). Each tag could contain some identifiable information to inform the user of its type. Standard Braille does not currently include

a system to facilitate the mark-up of items in this manner. However, one major consideration to be taken into account is the additional verbosity it might impose on an inherently serial method of interaction. If a document contains numerous emphasised segments, the verbosity of the content could grow quite considerably. This is due to the amount of characters necessary to reliably identify the mark-up tags, both as mark-up tags, and the type of emphasis to be indicated. However, in the case of lengthy highlighted segments, this may be a viable method for signalling typographical semantics.

Currently, the only highlighting method supported by Braille is italicised text. It is not slanted as with its visual formatting, instead, additional contextual characters are included to signify the change in emphasis (BAUK, 2004).

D.6 Braille production

During the early days of Braille production, a pointed object known as a stylus was used to punch tiny indentations into a piece of thick paper, resulting in groups of raised dot patterns representing individual Braille characters. Although the stylus is still used today, there are a number of additional mechanical mechanisms used to produce content in Braille. When using a stylus, it is necessary to imprint the text dot by dot into the underside of the page in reverse order, working from right to left. The modern Braille stylus has a rounded steel point set in a wooden, metal, or plastic handle. As the point is pressed into one side of the paper, it results in the appearance of a raised dot. The point of the stylus is rounded so it will not puncture through the paper entirely, but only displace a dot corresponding to the size of the point.

In modern times, the stylus is frequently used in conjunction with a Braille slate, whose purpose is to hold the paper in place while guiding the stylus to create Braille cells that are well formed and arranged in straight lines with proper spacing. The slate generally consists of a front and back plate joined with hinges, with additional pins to hold the paper in place.

The back plate has shallow holes arranged in three- hole by two-hole cells representing the six dots of the Braille cell. The front plate possesses a series of rectangular openings, each with six indentations along its sides to guide the stylus into the corresponding shallow holes on the back plate. The shallowness of the holes on the back plate is intended to prevent the rounded point of the stylus from puncturing through the paper (Johnson, 2004).

Braille may also be produced manually with a Perkins Braille typewriting machine, or electronically through a computer, operating either some refreshable Braille display technology, or a Braille embosser.

The Perkins Braille writing machine is a manual typewriter like device, which employs the use of needles to punch the required Braille dot combinations on to the underside of a page, resulting in raised Braille characters. Unlike a conventional typewriter, a large volume of keys is not required to represent individual letters. Instead, only six keys, each of which is connected to a needle, plus one additional key to function as a space bar, are available. The space bar is positioned in the centre of the keyboard, with a row of three keys positioned on either side. The left hand side set of keys are responsible for the generation of dots in the first vertical column. The key adjacent to the space bar produces dot 1, whilst the one furthest to the left generates dot 3. Similarly, the keys to the right of the space bar are responsible for the second vertical column of dots. The immediate key to the right is charged with producing dot 4, and the furthest key generates dot 6. Thus, by pressing different combinations of these keys, Braille characters are produced.

A Braille embosser is an electronically controlled, mechanical device used to produce pages of highly intelligible Braille output. It functions by imprinting an adjustable row of needles into the underside of the paper. The levels of the individual needles is adapted by the appliance to create the array of dots necessary to produce the line of text, transmitted by the controlling software. The advent of the embosser, coupled with character recognition scanning devices has resulted in a major increase in the pace of printed text

to Braille translation in recent years. Much manipulation of the printed document by the user, and in turn by the controlling software, is required, before the embossing takes place. This work is necessary to ensure the formatting of the target material is correct. Even whilst taking the additional editorial labour into account, Braille books are now produced at much greater speeds than when using previous manual methods of production.

D.6.1 Refreshable Braille Displays

A refreshable Braille display is an electronic device, which dynamically synthesises the production of Braille through the automatic raising and lowering of individual pins. The pins are often arranged horizontally across the display region in groups of eight, with each character cell divided into two vertical columns of 4, forming a single line of text. Although traditional Braille encoding requires just six dots to construct characters, many refreshable Braille display devices allow for character patterns of up to eight dots. Eight dot combinations offer a much greater scope in the variety of characters available. However, so far their usage has been limited to the encoding of computer notation (BAUK, 2006). The number of character cells available for display purposes can vary greatly, depending on the size of the device. In the case of larger appliances, up to 80 Braille characters may be available.

The chunk of text presented in the display region of the device normally coincides with a segment of the line currently under focus in the on screen text. The raw data to be exposed is not generally transmitted directly to the display apparatus. Instead, an element of processing by a piece of specialised software e.g., a screen reading application, is routinely undertaken before the data is transmitted to the device. The types of manipulation performed on the text may include the translation of text to accommodate grade 2 Braille shorthand abbreviations, or the insertion of contextual symbols to denote numerical data. Although many advantages to using these appliances exist, costs relating to the technology used in the manufacture of conventional

refreshable Braille display devices ensures that they remain rather expensive to purchase and out of reach for many blind computer users.

The large cost associated with conventional refreshable Braille displays is related to the price of the individual actuators necessary to alter the state of the pins. If you consider that each pin/dot requires its own actuator, and the number of dots possible, the reasons for the high cost should be quite apparent (Roberts et al., 2000). For example, an 80-character display, with eight dots available in each cell, would require 640 individual actuators. Automated production methods and mechanical miniaturisation have advanced a lot in recent years, so, in theory, Braille displays “could” be much cheaper. You only have to look at the dramatic decline in costs of conventional printers - also electromechanical or “mechatronic” devices - to see this. But the impact still depends a lot on economies of scale. As long as these particular actuators are specific to the one small, and perhaps even declining, niche of Braille displays, then production scale remains comparatively small, and costs correspondingly high.

One attempt to reduce the costs of Braille computer interfaces, examined the viewing of text through a single cell character display. The reader’s finger remains on the cell throughout the reading process, whilst at designated time intervals, the arrangement of the pins is altered to reflect the composition of the next letter. However, according to Roberts et al. (2000), this system was not very effective for the finger has a low sensitivity to this type of stimulus. Braille is easier to read when there appears to be a horizontal motion of interaction between the fingers and the Braille material. By scanning back and forth with the fingers, the user can simulate the horizontal reading movement with a single cell display; however, after some time the additional scanning may become quite tiresome.

The system proposed by Roberts et al. simulates the viewing of a continuous line of Braille. This facade is achieved by placing the letters on a rotating wheel, which slides under the reader’s fingers sequentially from right to left. Multiple characters may appear in the display at any given

time, thus, removing any issues resulting from the lack of horizontal motion in the reading process. The cost savings attributed to this device are expected to be rather dramatic for instead of requiring hundreds of actuators to display the characters, only three actuators, four in the case of eight dot Braille, are required. As each cell rotates across the actuators, which are situated to the right of the reading area, the intended Braille patterns are generated. In turn, as the cell rotates out of the reading area, a ramp exerts pressure on the pins forcing them into a lowered state. The ability to freeze the display to facilitate a more in-depth examination of the content is also proposed by the authors. From the initial user evaluations with the prototype apparatus, it was evident that with a little practice, blind individuals were successfully able to read using this system. However, such a device is not yet commercially available.

To date, much of the work in the development of refreshable Braille technology has centred on producing Braille using physical pins to simulate the dots. However, a study investigating the feasibility of displaying virtual Braille dots through lateral skin deformation was performed by Lévesque et al. (2005), who maintain that when the fingertip is locally deformed in the manner of a progressive wave, one typically experiences the illusion of objects sliding on the skin, even if the deformation contains no normal deflection. They developed an electromechanical transducer, designed to create skin deformation patterns with a view to investigating the possibility of displaying Braille dots. Although, the test subjects were able in many cases to distinguish correctly the intended patterns, the process in which they did so was often rather slow and error prone in comparison to conventional Braille reading. They also noted that a reduction in tactile sensitivity in the fingers occurred, when using their prototype system for a prolonged period of time. They concluded that before a commercial system would be viable, much further research in the area is necessary.

The major advantages in using a Braille display device are often observed when examining complex data types. For example, mathematical notation

for which comprehension is sometimes difficult. Greater benefits may be achievable through direct examination of the individual characters through the fingertips as opposed to having to deal with the additional memory load necessitated by speech output. Braille requires the reader to interface with the content in an interactive manner, whereas interfacing with spoken output tends to be more passive in nature. See section 2.1 for further details. Tactile methods of viewing such complex information may furnish a greater knowledge of the spatial relationships between mathematical terms. Also, interacting in this manner often provides a greater ability to handle the association between a closing bracket with its corresponding opening bracket. However, limitations relating to the size of the display region on the device can impinge on the readability of such information. This may be especially true in cases where the length of the equation is much greater than the amount of available character cells.

Another major area of content presentation where the use of a refreshable Braille display could be advantageous is in the viewing of tabular data. In theory an entire horizontal row of the table could potentially be presented in the display region of the device. However, in practice, the implementation of this concept is rather dependant on the length of the material and, the number of character cells available. Presenting tabular data in this manner could negate any problematic issues with comprehending the spatial relationships between the individual elements in the row. However, screen reading software manufacturers often prefer to present tabular data one cell at a time in a linear format, due to the differences in character cell capacity provided by the different refreshable Braille display solutions. Therefore, they must ensure that additional contextual information concerning the current grid position is provided to aid comprehension. Also, in many cases, additional navigational functionality to move along the spatial relationships between the cells has been made available.

A number of disadvantages can also be attributed to the use of this technology. Some of which are as a direct consequence of the expense relating

to such devices, see section D.6.1. This is especially the case when compared to the costs of synthetic speech production. However, many of the other problems arise as a direct result of the serial nature of the medium. Only a small segment of the page content can appear in the display region at any given time. Although the user is usually afforded navigational control inside the document, it is difficult to obtain an accurate impression of the page structure, without exploring the entire document. The issues relating to the portrayal of visual cues such as large fonts and typographical changes to emphasise important content, see section D.5, means that the skimming effect employed by visual readers to locate changes in contextual information is extremely difficult to mimic.