Tools for creating and maintaining text-only web pages

ITBML 3rd Year Project Report

Omid Afzalalghom

Supervised by Dr. A.D.N. Edwards

Department of Computer Science
University of York
Heslington
York
YO1 5DD

Submission Date: 14th March 2002

Total word count including title page, abstract, table of contents, references and bibliography: 19,810 (as calculated by Microsoft Word).

The appendix has not been included in the word count.

Tools for creating and maintaining text-only web pages

Abstract

The World Wide Web is an ever-increasing expanse of information and services. To be excluded access to the Web is becoming a growing disadvantage, yet it is a reality that faces many blind people today. Fortunately, there is a solution for blind web users – text-only web pages.

Text-only web pages do not contain the graphical items often found in web pages, such as images and image maps. The purpose of text-only web pages is to provide the same information as standard web pages but in a more accessible fashion. By providing these pages, web authors can ensure that the blind population are not at a disadvantage to the millions of sighted people using the Web every day.

This project aims to produce a set of tools that will facilitate the creation and maintenance of text-only web pages. Current web authoring products and web design guidelines are discussed as well as the different browsers used by blind people. The proposed tools are hoped to help web authors produce web pages that are more accessible to blind people.

The software developed consists of three complementary tools. The first program creates a parallel directory structure in which both standard and text-only web pages can be stored. The second tool converts standard HTML web pages into text-only format. The third program is a maintenance tool that ensures that each standard web page has a text-only equivalent that is kept up-to-date.

The project concludes that the web authoring tools created are capable of improving the accessibility of web pages. However, the tools have their limits and web authors should endeavour to follow the design guidelines provided by the World Wide Web Consortium. In doing this, web authors can enhance the accessibility of future web pages.

Acknowledgements

I would like to thank Kris Fearon, the YorkWeb author, for her participation during the analysis and design stages. My thanks also go to Dr. Alistair Edwards for his invaluable supervision throughout this project.

Contents

Figures

1 Introduction

1.1 Visually impaired users and the growth of the Web

Visually impaired people are being denied access to one of the most valuable resources of information in the world – the World Wide Web. The Web has become a feature of everyday life in most Western countries, with people using it for many reasons including: education, to find information, and to buy a variety of products and services. In the UK and the USA over 50% of the population use the Internet [Nua, 2001], so it is becoming a growing disadvantage to be excluded access to it.

There are 1.7 million people in the UK who are visually impaired. Impaired vision simply means that someone's sight does not work or does not work as well as it might [RNIB, 2001]. People whose sight has not completely deteriorated often prefer to read the same web pages that fully sighted people do; this might be achieved with the aid of a screen magnifier, for example. Unfortunately, blind people require further assistance if they wish to read the same pages.

For a blind person, a specialist text-based web browser such as Lynx can be used if they wish to access a standard web page with graphical items. Lynx will extract only the text from a web page so that it is in a suitable format for a screen reader to convert it into speech output. However, blind people usually do not want to be seen as being different from sighted people and so prefer to use common web browsers such as Netscape Navigator or Microsoft's Internet Explorer in conjunction with a screen reader. A screen reader converts text on the screen into audio or tactile output, which can then be transmitted to a blind person via speech synthesis or a tactile Braille display respectively. Unfortunately, screen readers cannot convey the meaning behind many of the graphical items that pervade web pages today.

One solution to this problem is for web authors to follow design guidelines provided by organisations such as the World Wide Web Consortium (W3C). These guidelines encourage the design of more universal web pages, accessible to both sighted and visually impaired users. Though with over two billion web pages already in existence it is unrealistic to expect a mass overhaul of design. Also, web designers who focus upon aesthetics are reluctant to compromise presentability for accessibility in order to accommodate a minority.

Another solution to the problem is for web authors to provide text-only web pages, which screen readers can process.

1.2 Text-only web pages

Web authors should ensure that text-only web pages retain as much of the original content as possible so that little or no meaning is lost. This ensures that a blind person

can discuss a web page with a sighted person, confident that they have read the same information.

It should be emphasised that text-only pages are not only of use to blind people. If a user has a slow connection to the Internet then it can be convenient to access the text-only version of a website, thereby avoiding downloading graphical items. Text-only pages can also be of use to people using handheld computers such as Personal Data Assistants (PDAs) where the small screen size means that images are unsuitable.

The current problem for blind people is twofold: there are very few text-only web pages in existence and many of those that do exist are not kept up-to-date. There are two possible reasons for the apparent dearth of text-only pages: ignorance and laziness. Many web authors are simply unaware of the diverse needs of their audience, whilst for others text-only web pages require too much effort. The ignorance that is partly responsible for the scarcity of text-only web pages is quite remarkable considering that disabled people in the UK spend an estimated £40 billion a year on goods and services [DRC, 2002]. It is hoped that the proposed tools will be of use to those web authors who find that text-only web maintenance is too time-consuming.

1.3 Project aims

With a reported 98% of websites being inaccessible to blind people [iCan, 2000] the importance of introducing text-only web pages is clear. The aim of this project was to develop a suite of software tools that would help web authors to both create and maintain text-only HTML web pages in parallel with standard graphical web pages.

The three distinct tools that I planned to implement were hoped to:

- Automate the creation of a parallel directory structure
- Convert a graphical web page to a text-only web page
- Warn a web author when text-only web pages require updating

1.3.1 Parallel directory tool

This tool will set the foundations for better web design by creating a parallel directory structure for the web author. There will be a parent directory with two child directories, one for the storage of graphical web pages and one for the text-only equivalents.

1.3.2 Converting tool

The purpose of this tool is to take a standard graphical web page and convert it into a text-only version. Any HTML code that has been identified during the project analysis stage as unmanageable by screen readers, or specialist web browsers, should

be expunged from the new text-only file. The new text-only web page will then be saved in the second child directory.

1.3.3 Maintenance tool

The maintenance tool should compare the contents of the two child directories at regular specified intervals. Whenever the web author writes and saves a new graphical web page this tool will alert him/her that a text-only version should also be produced. The web author can then use the converting tool to create that text-only version. The maintenance tool will also monitor the graphical web pages for modifications and prompt the web author to modify the corresponding text-only versions.

These tools will be further discussed in Chapter three.

2 Background

Why do websites need to be made more accessible? I feel that this question should be answered before beginning to build the proposed tools in order to justify the need for them.

It could be argued that websites should be designed to be usable by all to the greatest extent possible. Edwards [CHI, 1996] proposes that to provide universal access would require little extra effort if designers were more aware of their audience's abilities. By identifying the needs of blind users, web authors could provide web pages that are fully accessible to people without sight. This idea of a universal design does not mean that one party gains from another's loss; it can very often be of benefit to everybody. For example, the introduction of sloped entrances to buildings for wheelchair users also provides easier access for people using pushchairs. In this way, a well thought-out website designed for ease-of-use is very likely to benefit everyone that uses it and not just blind users.

There are two other reasons for improving the accessibility of websites: economic advantage and legislation.

2.1 Economic advantage

The concept is simple: The more accessible a website is the greater the potential audience. Consequently, an increase in visitors to a website could lead to rising sales revenue in addition to greater advertising power. The visually impaired community comprises over two million people in the UK [RNIB, 2001], which represents a substantial market opportunity to any company. In fact, home shopping services offered by many supermarkets could be exactly what visually impaired people require. However, many of the websites offering such services remain inaccessible to those who, arguably, need them most.

The situation is progressing though with market leaders Tesco showing the way. On 22nd May 2001 Tesco Access become the first website to be awarded the RNIB's "See It Right Accessible Website" logo. After complaints from some visually impaired customers the supermarket chain Tesco designed a new, more accessible, website. Tesco involved twenty people with visual impairments in the design of the new site, which has far fewer images and more links with descriptions. The new website received the RNIB stamp of approval after their usability specialists tested it against their own rigorous standards. The average online spend at Tesco is £90 [Guardian, 2001] so with a further two million people now able to access their services they can expect to see an increase in revenue. If only 1% of the nation's two million visually impaired population were to make the average online spend of £90, Tesco would be reaping an extra £1.8 million in sales revenue.

A more accessible website can also lead to lower overheads. The number of required technical support staff can be reduced as a result of fewer enquiries from customers being denied access. However, the cost of supporting a larger number of customers may sometimes offset this previous advantage gained. An indisputable benefit is that there is less chance of negative publicity if a website is accommodating all users without discrimination. In the case of Tesco Access the company not only avoided negative publicity but also gained free publicity that praised their innovation. Finally, there is an increasing chance of companies incurring considerable legal costs if the law is not adhered to, which will be discussed next.

2.2 Legal requirements

The Disability Discrimination Act 1995 (DDA) came into force, in part, in the UK in December 1996. Part III of the Act gives disabled people access rights to goods and services provided by both public and private sector organisations. Section 19 of the DDA [DDA, 1995] states:

"It is unlawful for a provider of services to discriminate against a disabled person in refusing to provide, or deliberately not providing, to the disabled person any service which he provides, or is prepared to provide, to members of the public."

Whilst there is no explicit mention of website accessibility in the DDA it is possible to see how one might apply the above regulation to a website. The Act gives the following as examples of services: *"access to and use of means of communication"* and *"access to and use of information services"* [DDA, 1995].

The DDA guards against discrimination as well as encouraging positive action towards making services such as websites accessible to the entire population. Legal action against companies exhibiting such discrimination has already been seen in both the USA and Australia. In the USA the National Federation of the Blind (NFB) filed a lawsuit against America Online (AOL). Although this case fell under the jurisdiction of the Americans with Disabilities Act 1990 (ADA), the same situation could arise in the UK. The NFB claimed that the software used by AOL was incompatible with screen reader software used by its members and their services were therefore inaccessible. AOL made an out-of-court settlement with the NFB and subsequently agreed to design future software to be compatible.

Section 302 of the ADA 1990 [ADA, 1990] states that:

"No individual shall be discriminated against on the basis of disability in the full and equal enjoyment of the goods, services, facilities, privileges, advantages, or accommodations of any place of public accommodation by any person who owns, leases (or leases to), or operates a place of public accommodation."

This, along with Section 508 amendments of the Rehabilitation Act 1973, demands the same of US websites as the DDA 1995 does of UK websites. Section 508 requires Federal agencies to ensure that electronic and information technology is accessible to employees and members of the public with disabilities to the extent it does not pose an undue burden [Section 508, 1998].

From 1st October 1999 it has been UK law for service providers to make "reasonable adjustments" for disabled people, such as changing the way they provide their services [DRC, 2002]. The Special Educational Needs and Disability Rights Act, May 2001 [SENDRA, 2001] has made it unlawful for:

"The body responsible for an educational institution to discriminate against a disabled student in the student services it provides, or offers to provide."

In the USA all web pages created or modified after 21st June 2001 must meet the standards drawn up by the Access Board. The Access Board is a Federal agency responsible for developing accessibility standards designed to aid those with disabilities.

All of these Acts mean that companies should be aware of not only the law of their own country, but also the law of any other country where they have customers or websites. If a UK company has a US-based website or American customers then it should meet the requirements of the ADA 1990 as well as the DDA 1995. The W3C guidelines can help to ensure that organisations are in accordance with these Acts.

2.3 Current web design guidelines

The W3C's Web Accessibility Initiative (WAI) is the closest thing to a regulatory body that the Web has. It advises web authors and web developers how to design web pages that are accessible to disabled people. The guidelines have different priorities ranging from one to three:

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."
>
> [WAI, 1999]

There are then three conformance levels as follows: level A where all Priority 1 checkpoints are satisfied; level Double-A where all Priority 1 and 2 checkpoints are satisfied; and level Triple-A where all Priority 1, 2, and 3 checkpoints are satisfied. Clearly web authors should be aiming for at least either level A or Double-A.

The WAI guidelines are rather lengthy so I have summarised the recommendations regarding only visually impaired users, as they are whom the proposed tools are designed to indirectly help. The following guidelines are Priority 1:

Provide text equivalent to visual items. This should be done using the ALT tag for images and image maps. An ALT tag holds a text description of an image; it appears on the screen when a user places the mouse cursor over the image. An image map has two or more parts that when clicked upon will link to different web pages. A list of text links should be provided in addition to an image map. For movies and animations a caption or transcript should be supplied additionally.

The LONGDESC tag can be used if the visual item requires an extended written description. The LONGDESC tag can link to another web page where a fuller narrative can be given. LONGDESC tags and ALT tags are not always necessary though, as some images do not require text descriptions. Nielsen [2000] suggests that if the only purpose of an image is to improve the appearance of a web page then the user does not need to hear a description of it. This does not mean that the ALT tag should be excluded; an empty ALT tag should be used instead. For example, instead of ALT="small square bullet" use "ALT=" "". This would inform a screen reader that the image could be ignored.

Use client-side image maps. Client-side image maps should be used instead of server-side image maps as they allow the user to interact with a preferred input. Text equivalents to client-side image maps can be provided to allow the user to use the keyboard instead of the mouse for example; server-side image maps do not accommodate such a choice.

Do not rely upon colour to convey information. There should be reasonable contrast between background and foreground. Users should be allowed to change the font type and size. Style sheets should be used in place of the FONT tag to maintain a more consistent format. Relative font sizes should be used in place of absolute font sizes to allow the user to increase and decrease the font size as they wish.

Only use tables where necessary. Tables should not be used for layout purposes as they cause problems for users using screen readers. Only truly tabular information should be represented in tables and the TR and TH tags should be used to identify rows and columns.

Provide alternatives to embedded interfaces. If JavaScript is used then an alternative web page should be accessible that does not use JavaScript. A text equivalent using the <NOSCRIPT> tag could be provided for people that do not wish to use JavaScript.

The following guidelines are Priority 2:

Ensure user control of moving text. Screen readers are unable to read moving text so blinking and scrolling text should be avoided. Therefore, the BLINK and MARQUEE tags responsible for such text should not be used. Users should be allowed to freeze any moving text. As Netscape does not support the MARQUEE tag and Microsoft Internet Explorer does not support the BLINK tag, these tags are not conducive to a universal design.

Provide alternative to frames. Frames are distinct areas of the screen that can be navigated independently of each other. Screen readers have particular difficulty handling frames so the NOFRAME tag should be employed to overcome this problem. The NOFRAME tag can hold the HTML code to an alternative web page that does not use frames. A web page should scroll from top to bottom rather than side to side as screen readers read across the page. If a page uses columns in its layout then screen readers will inevitably struggle to relay the information to the user.

Label forms correctly. Input values on forms should have a default value as some browsers have problems with empty fields. Each field should be clearly labelled as some screen readers read lists of consecutive links as one link. The label should immediately precede its field.

I have not included the Priority 3 guidelines in my summary, as they are more concerned with the presentation of the web page rather than its functionality. Web authors can use all of the above checkpoints to design a more accessible website for visually impaired users. I intended to use some of these guidelines myself when designing the proposed web authoring tools. The guidelines helped me to understand which elements of HTML it is that screen readers have problems handling.

2.4 Java and HTML 4.01

The most recent version of HTML is HTML 4.01 and the proposed tools were designed with this in mind. HTML is the publishing language of the World Wide

Web and it defines all elements and attributes that may appear in HTML documents. Elements that determine the structure of a web page are those such as links, tables, applets, text and images. Fonts, colours and style sheets contribute towards the presentation of web pages whilst forms and scripts govern the interaction within an HTML document. Applets, which are written in the Java language, can often exacerbate web page accessibility for blind people. A Java applet is a small program embedded into a web page whose code is transferred to the browser for execution. Due to the fact that the Java code is embedded within the HTML document, this information has, until recently, been inaccessible to software such as screen readers. This made the comprehension of web pages written using applets very difficult for blind people.

However, Sun Microsystems [Sun], have developed a piece of software called the Java Access Bridge. This piece of technology allows applications such as screen readers to gain access to the code behind Java applets. The screen reader interacts with the Java Application Programming Interface via the Java Virtual Machine to access the applets and convey the intended information to the user. JAWS for Windows now provides support for the Java Access Bridge.

2.5 Interview with the YorkWeb author

Before embarking upon the design of the planned tools I decided to ascertain the needs of a web author. I arranged an interview with Kris Fearon, the author of the University of York's text-only web pages, which are also known as YorkWeb. Kris is responsible for maintaining the University's central pages but has the additional responsibility of providing parallel text-only pages for use by, among others, visually impaired people.

Kris expressed her desire to be able to provide one universal set of web pages accessible by all; she finds that maintaining parallel text-only pages is time-consuming and laborious. Her current modus operandi when a new web page is to be created is outlined in (Figure 1).

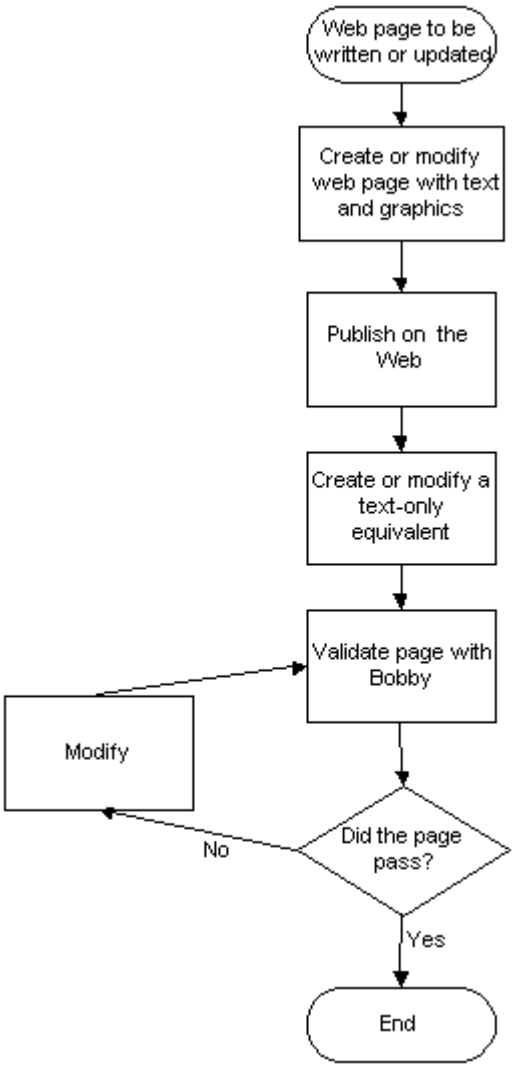

Figure 1. Process used by YorkWeb author to create a text-only web page.

Kris writes the graphical version of web pages first and then immediately creates the text-only version so as to avoid any out-of-date pages. Her method of creating the text-only version is to cut and paste the body text from the original web page into a new file, omitting any images. Kris then uses the online validation tool Bobby provided by the Centre for Applied Special Technology (CAST) to assess the usability of her new text-only web page. If the pages passes validation then the job is done, otherwise further alterations must be made.

Kris identified the main disadvantage of this model to be the time required to create and validate the text-only page. The most time-consuming part of the model is the validation with Bobby. Kris found the results that Bobby produced to be cumbersome and difficult to decipher. The advice offered by Bobby was disorganised and often very technical making it difficult to act upon.

A suitable refinement to the model in Figure 1 would be to validate the graphical web page once it has been written. If the original web page is validated then the text-only

page should pass validation as it is created from the graphical web page. By following web design guidelines such as those issued by the W3C, authors can build solid foundations from which text-only web pages can then be created.

The problem of meeting Bobby's standards could be effectively overcome by letting a piece of software create and modify the text-only web page. Although Kris stated that she would prefer a universal set of web pages, she said that an automated tool that would facilitate the operations shown in Figure 1 would also be of interest to her. Kris had not tried any existing tools that offered such services but there is software already available that can help web authors.

2.6 Existing web authoring tools

Before I could begin to design the intended tools I wished to look at several products within the field of web-authoring software and text-only web pages. I hoped that by exploring the field of web accessibility I could gain an insight into exactly what is required, thus avoiding building something that was unnecessary. Moreover, I would not be wasting time reinventing the wheel.

2.6.1 TOM

TOM (text-only maker) is a joint venture between the National Center for Supercomputing Applications (NCSA) and the University of Illinois [TOM]. Its purpose is to create text-only web pages from graphical web pages with as little human assistance as possible. TOM can also add ALT tags to all images in a document, rendering a page more accessible to blind people. TOM appears to be no longer available through the link advertised on many web pages so I was not able to test the program.

2.6.2 Betsie

The BBC Education Text to Speech Internet Enhancer, or Betsie [Betsie] as it is more commonly known, is a Common Gateway Interface (CGI) script implemented with the Perl interpreter. A CGI script is run from a web server and passes information between a web browser and a web server. Betsie works by receiving a request to view a web page, passing the URL to the web server and then parsing the content of the web page to produce a text-only web page. Betsie will strip out items such as IMG tags leaving only the ALT text description or completing removing the image if there is no ALT tag. Betsie was created in June 1998 in an attempt to make the BBC website accessible to blind people.

2.6.3 W3C HTML validation service

Web authors can use the free validation service provided by the W3C [W3C Validator] to check the conformance levels of their websites, i.e. level "AA", or level "AAA". The program produces a report on a given website highlighting any errors or

discrepancies. If the website meets the requirements of the W3C then the web author may display the W3C logo on the site to show its level of accessibility.

2.6.4 Bobby

Bobby is a free online validation tool provided by the Center for Applied Special Technology (CAST) [Bobby]. A web author can submit a URL to Bobby and receive an online report within seconds highlighting the areas of the web page that do not conform to the W3C WAI guidelines. Bobby seems to have difficulty with JavaScript, often returning jumbled and meaningless text in the report. On the whole though, Bobby will produce a report emphasising the areas in which the website does not conform to the W3C checkpoints. If all Priority 1 objectives are met then the website can display a logo displaying that the W3C Level "A" has been met. The Level "AA" logo can be used if Priority 2 objectives have been met and similarly for Priority 3.

2.6.5 MkDoc.com

MkDoc is a software package that helps web authors to create standards-compliant websites [MkDoc]. Through a web interface, MkDoc uses the W3C checkpoints to guide the user towards designing and creating a more accessible and navigable website. For example, MkDoc will ensure that all the hyperlinks have a meaning so they can be understood when read out of context; it will not allow a link that says, "Click here". Perhaps one the most useful features of MkDoc is the breadcrumb trail-navigation bar. This is a bar that represents the hierarchical structure of a website allowing users to quickly jump between levels. A common problem with web pages is that users can easily lose track of where they are in a document and are left to rely upon the back and forward buttons of their browser to navigate. Unfortunately MkDoc is not free, a single website license costing €800.

2.7 Screen readers and specialist browsers

Blind people have two options for web browsing: they can use a standard browser with a screen reader or use a specialist browser. If a blind person prefers to use a standard web browser such as Microsoft Internet Explorer then they will require the assistance of a screen reader such as JAWS for Windows. Some specialist browsers such as BrookesTalk come equipped with speech engines, others such as Lynx are used in conjunction with a screen reader.

A screen reader gives blind people access to software applications by converting the display of a computer into verbal output. A screen reader intercepts screen data as an application sends it to the screen buffer. This data is then sent to a speech synthesiser for text-to-speech conversion before being allowed to continue on to the screen for display. Neither the screen display nor the application being used is altered during this process.

A typical screen reader has two modes: live mode and review mode. Live mode is usually used during keyboard input and text is spoken as it is entered. Review mode allows the user to highlight different areas of the screen to be read aloud. By using either the mouse or the cursor keys the user can move the cursor around the screen to select different words, lines or windows to be converted to speech.

2.7.1 BrookesTalk

A different kind of specialist browser is BrookesTalk, which uses the Microsoft Speech Engine [BrookesTalk]. BrookesTalk provides additional aid to the user via a menu that provides access to: a document abstract; the headings of the document; the document keywords and all the hyperlinks in the document. BrookesTalk is more suited to users with some residual sight, as it does not handle frames or image maps.

This review has helped me form a better idea of the needs of blind people regarding web accessibility. The next chapter will summarise my findings and discuss the specifications and design of the proposed tools.

2.7.2 Lynx

The specialist text-based browser, Lynx, overcomes the problem of such web pages by removing images and image maps leaving only text. Lynx will replace images with their ALT text, if present, and likewise for image maps. The browser is also capable of handling frames by providing text links to each frame of the web page. In this manner, one frame can be viewed at a time enabling the successful use of a screen reader.

2.7.3 JAWS for Windows

JAWS uses its speech synthesiser and the sound card of a computer to translate the on-screen information into an audio output. JAWS can also output to a refreshable Braille display, which is an electronic device used to read text sent from a computer to a monitor.

Refreshable Braille displays work in conjunction with screen readers and interface with computers through an input/output port. A screen reader sends text to a Braille display as the user guides the cursor to read specific areas of the computer screen. The Braille display then converts the text to tactile output by displaying it using a series of mechanical dots. These dots move up and down to form Braille characters that can be read by the user. Refreshable Braille displays use an eight-dot code rather than the standard six-dot code. The extra two dots are used to convey supplementary information, such as cursor position and text formatting. The main disadvantages of Braille are that it is difficult to learn and that the equipment, such as Braille displays, is expensive.

By using a screen reader a blind person can access the same websites as everybody else without the need for a specialist browser. The disadvantage of this approach is

that many screen readers cannot effectively represent certain elements such as frames and image maps. A web page might be split vertically into two frames with a column of text in each. A screen reader will not read the page a column at a time; it will read across the page thus returning a meaningless jumble of words. A screen reader cannot translate images and image maps into audio output either, as it will only read text.

The topics discussed in this chapter can all influence website design. Web authors have at their disposal various web authoring tools and web design guidelines. The proposed tools were intended to help web authors adhere to these guidelines. The exact requirements and the design of the tools are discussed in the next chapter.

3 Methodology

In the previous chapter the reasons for providing more accessible web pages and the methods for designing them were discussed. Whilst a universal design of web pages seems the ideal solution, it will be some time before the Web is truly accessible to everybody due to its incredible size. In the meantime, text-only web pages can provide a quick and easy middle ground for blind Web users.

The W3C guidelines outlined in Chapter two were written with visually impaired users in mind. However, people that have some degree of vision remaining usually prefer to make use of it and so use the same web pages as fully sighted users. Therefore, the majority of people that use text-only web pages are blind. The term "blind" refers to someone with visual acuity of 3/60 or worse, or 6/60 with a restricted field of vision. 6/60 means that the observer "would be able only to discriminate at six metres what the normally sighted person would see clearly at 60 metres" [Chapman, 1988].

From the W3C guidelines I highlighted the following areas to be addressed: frames image maps, images, and the <SCRIPT> tag. These four aspects of HTML are all barriers to blind people trying to access the Web. I recognised other features of HTML that proved to be obstacles to blind users, forms and tables being two such problems. However, these two particular aspects have given rise to much thought and work involving some complex theories. The non-visual representation of tables, for example, has been the study of an entire student project at the University of York [Kemenade, 2000]. Therefore, given my time constraint I elected to concentrate on the four elements previously mentioned.

I decided to use frames and scripts as examples of document structures impeding accessibility. Both of these HTML elements have accompanying alternative tags that can be used to provide accessibility to a wider audience. The <FRAMES> tag is coupled with the <NOFRAMES> tag whilst the <SCRIPT> tag should be used in conjunction with the <NOSCRIPT> tag. Neither frames nor scripts should affect accessibility when there are complementary tags such as <NOFRAMES> and <NOSCRIPT> to be used.

I focussed on image maps and images due to the high percentage of web pages containing them. According to Beckett's 1997 survey [Beckett, 1997], there are images present in 87.42% of web pages. Also, many images are inherently visual such as photographs. This is a prime example of when the ALT tag should be used to provide the blind user with a text alternative to the image. Unfortunately, web authors often neglect this opportunity for greater accessibility; a mere 21.19% of all <IMG> tags have an ALT text [Beckett, 1997].

I used three web browser solutions available to blind people to demonstrate the way in which these aspects can hinder access to the Web: BrookesTalk, JAWS with the Lynx Viewer [Lynx], and JAWS in co-operation with Microsoft Internet Explorer 6. The choice of these three browser solutions represented the three different approaches that a blind user might take when using the Web. BrookesTalk is the all-in-one browser with a built-in speech engine, Lynx is a text-based browser, and JAWS is a screen reader.

3.1 Development of test web page

To highlight the strengths and weaknesses of each browser I required a web page containing all four of the elements listed above. At this stage I had two choices: find a suitable web page on the Web or create one myself. Trawling the Web for an appropriate web page could have been quite time consuming so I opted for building one myself. This approach ensured that I had the exact test data required.

The web page that I created consisted of a frames version and a non-frames version. If the browser being tested did not support frames then it interpreted the code between the <NOFRAMES> and </NOFRAMES> tags, otherwise it interpreted the code between the <FRAMESET> and </FRAMESET> tags. Figure 2a shows the web page as viewed through a browser that supports frames. Figure 2b depicts the web page viewed as its non-frames version. The links in the left-hand frame of the web page in Figure 2a, when clicked upon, loaded a new page into the right-hand frame, which held the main page. This method facilitates navigation, as the menu is always visible to the user. In Figure 2b the “News”, “Sport” and “Search” links were moved to be part of the main page. In this case the back and forward buttons of the browser were used to navigate between pages.

I included an image map in the web page, which consisted of two rectangles, a circle and a triangle (see Figure 2a). Each shape, when clicked upon, links to another web page. Each shape in the image map had an associated ALT text as did the map itself. The image map is an element often used on the index page of a website to act as a site map.

Three different images were incorporated into the web page to test different attributes of the <IMG> tag. The first image had no ALT tag but did have a <NAME> tag; it was named “poet”. The second image had no ALT tag but was linked to another page with the anchor tag <A>. The third image had an ALT tag giving a brief description of the picture and was also linked to another page using the <A> tag. These three different tags, ALT, <A> and <NAME>, were used in order to provide the browsers and screen reader with as much information as possible to use when trying to represent the images to the user.

To demonstrate the <SCRIPT> tag I included a small piece of JavaScript in the web page source code. The script, if run, simply printed a line of text saying "hello world". I also included a <NOSCRIPT> tag, which provided an alternative for browsers that did not support scripts. If the browser did not support scripts then the message "Your browser does not support scripts" was displayed. As can be seen in Figure 2a the browser did not support scripts. The browser in Figure 2b however did support scripts and so the message "hello world" was displayed. These two script examples are chosen purely for test purposes. For a real web page the script would probably be more extensive and the <NOSCRIPT> alternative would be a satisfactory substitute for the script.

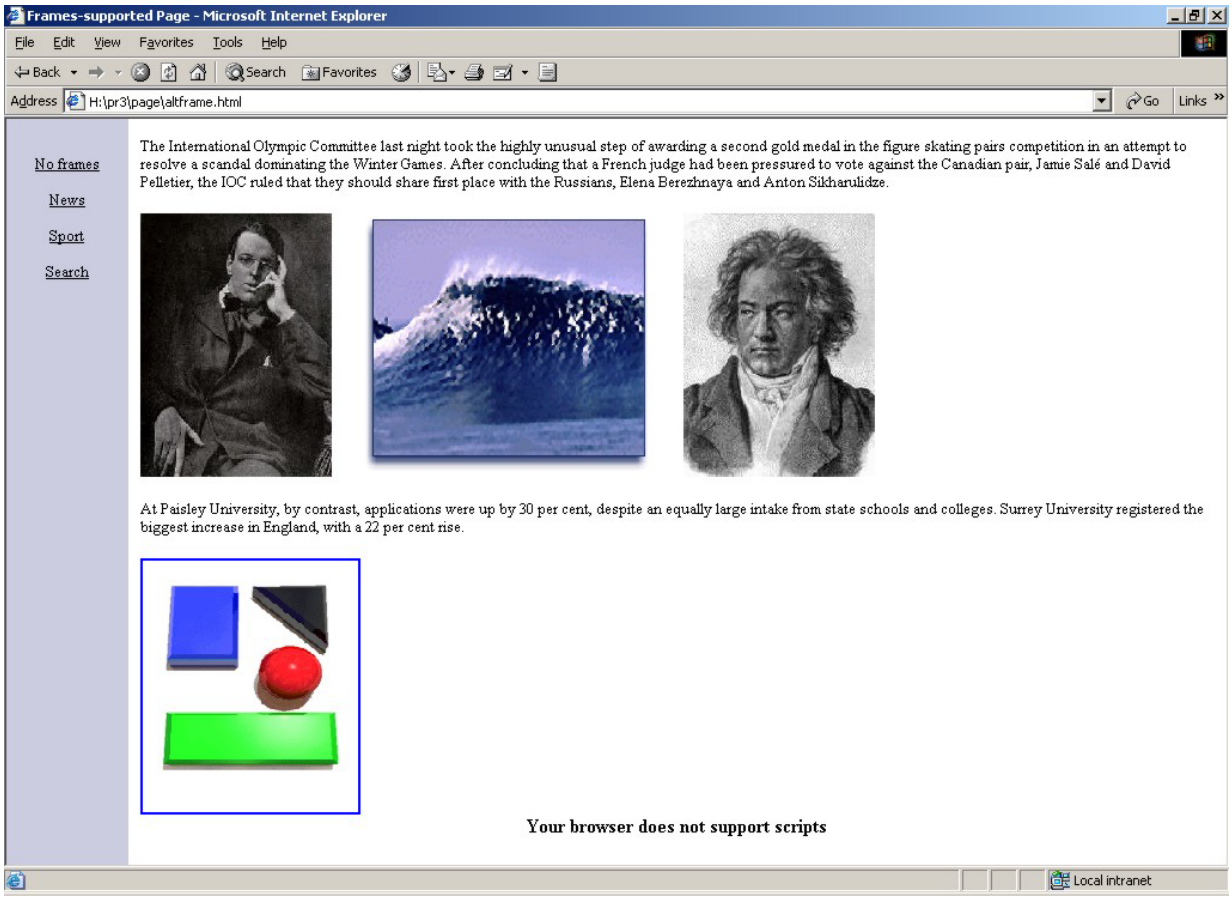

Figure 2a. Test web page supporting frames but not JavaScript.

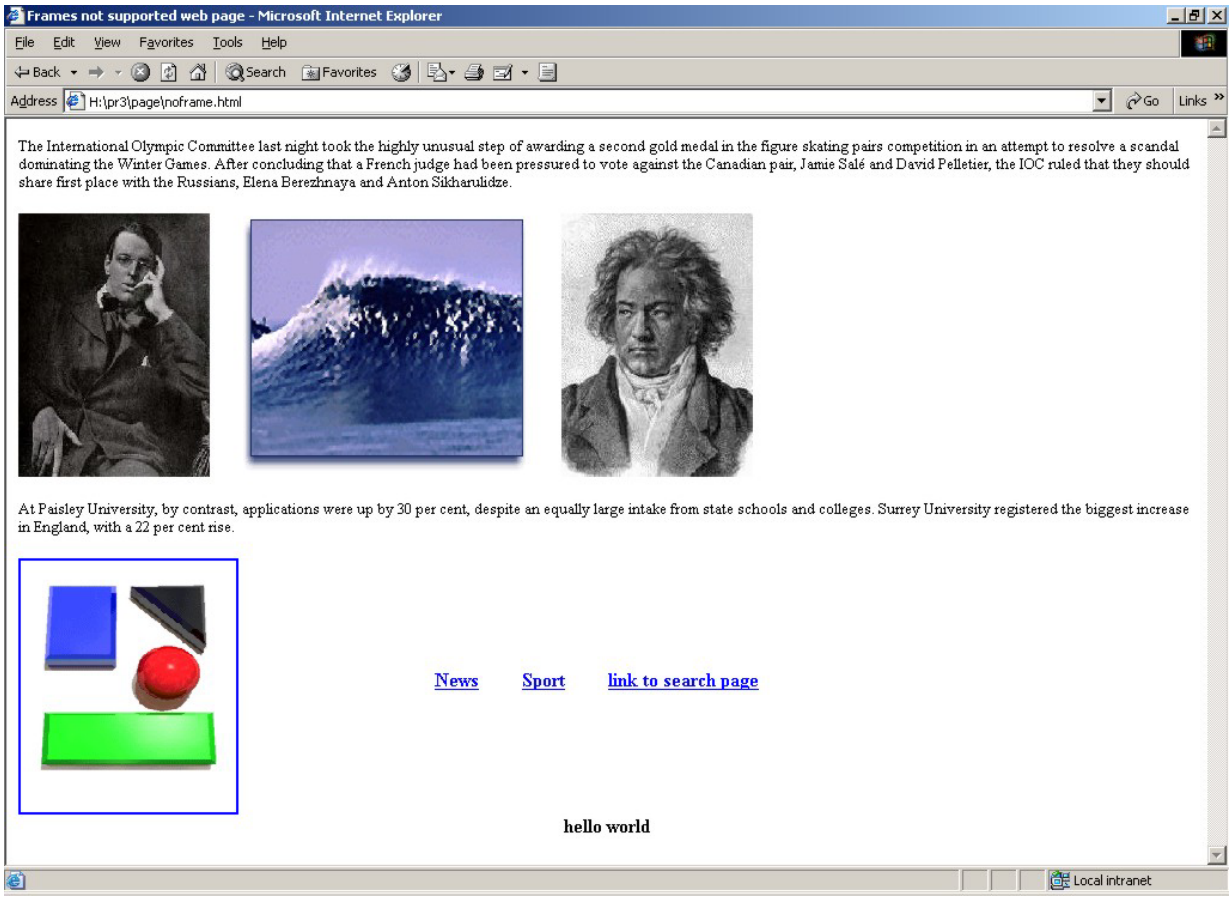

Figure 2b. Test web page supporting JavaScript but not frames.

The next section describes the installation process of the software and the way in which each browser handled the four significant elements of HTML.

3.2 Performance of BrookesTalk

BrookesTalk is a web browser for blind and visually impaired users, which is being developed at Oxford Brookes University. After installing the browser from a CD I familiarised myself with its operations and then used it to view the test web page. The non-frames version was displayed so I deduced that BrookesTalk does not support frames. I commanded the browser to read out the whole document and I observed the way it processed each of the remaining three aspects.

The image map was displayed but not mentioned in the audio output. Moreover, none of the links incorporated in the image map were extracted so a blind user would have been unaware of these further pages and the image map.

BrookesTalk provided the blind user with no alternative representation of the images. BrookesTalk did not use the information in the <NAME> tag, which was attached to the first image. This disregarded information could have been used to convey some representation of the image to the blind user. The second image, which linked to another page, was displayed as an “unknown link” on the links menu. This would

obviously become confusing if there were several unknown links, as the user would not be able to differentiate between them. The ALT tag of the third image was not read out by BrookesTalk meaning that a blind person would be oblivious to any images in a web page. The third image was also displayed on the links menu as "unknown link" despite having an ALT tag.

BrookesTalk did not display the "hello world" text, which indicates that it does not support JavaScript. The browser instead displayed the code within the <NOSCRIPT> tag, "Your browser does not support scripts".

3.3 Performance of JAWS with Lynx

For convenience a Lynx viewer was used rather than Lynx itself, thereby saving on installation and configuration time. The Lynx viewer is an online tool that emulates how a web page would appear through Lynx. The viewer does not provide simulation for image maps however. This did not cause a problem as the W3C website describes how the Lynx browser provides a text menu of the ALT texts from the image map. If ALT texts have been excluded then the URLs are used instead [W3C]. In this way, the Lynx browser would extract the words "circle", "rectangle" and "triangle" from my test web page as these are the ALT texts. The JAWS screen reader was used to convert the text displayed by Lynx into speech output.

I used the Lynx viewer to access the test web page and discovered that Lynx did support frames. The browser represented each of the two frames with a hyperlink to each frame so that they could be viewed individually. I could view either the left-hand frame (the menu) on a full screen or I could view the right-hand frame (the main page) on a full screen.

The image map was represented by the text "USEMAP". However, this is only how the Lynx viewer processes image maps and not how the actual Lynx browser would represent it. As discussed, the Lynx browser would have indicated the presence of the image map by displaying the ALT texts as hyperlinks. In brief, the Lynx browser would show four links represented by the words, "rectangle", "circle", "triangle" and "rectangle".

Lynx handled the images much better than BrookesTalk did from the perspective of a blind person. The first image was represented by the text "[INLINE]", which whilst not intuitive could be remembered by a user and thereafter associated with the occurrence of an image. The Lynx viewer did not use the <NAME> tag of the first image. The second image was displayed as a hyperlink with the text "[LINK]". The third image was represented as a hyperlink using its ALT tag, "Picture of Ludwig Van Beethoven". The text equivalent that Lynx provides for frames, image maps and images can all be conveyed to a blind user with the assistance of a screen reader.

The web page displayed the text “Your browser does not support scripts” indicating that Lynx does not support the <SCRIPT> tag. The issue of scripts causing inaccessibility to blind users is further discussed in the analysis section.

3.4 Performance of JAWS with Internet Explorer

After downloading and installing a demonstration version of JAWS for Windows I used it in conjunction with Microsoft Internet Explorer to view the test web page.

Internet explorer supports frames so the frames version of the web page appeared in the browser window as expected. JAWS correctly began reading the topmost line of text beginning “The International Olympic…”. After reading the second line of text JAWS then moved to the next lowest line. Unfortunately for blind users this was not the third line of the text but the “No frames” option from the menu in the left-hand frame. This demonstrated that frames are not suitable for screen readers and should be avoided.

JAWS did not offer any representation for the image map whatsoever and likewise for the three images. I deemed these outcomes anomalous in view of the fact that there were some ALT texts present. In order to test whether JAWS could read ALT texts I decided to turn the “Show pictures” option off in Internet Explorer. This option results in corresponding ALT texts being displayed in place of images.

With the “Show pictures” option deactivated the image map was replaced by its ALT text, which in turn was read aloud by JAWS. Whilst the ALT texts of the individual shapes within the image map were not displayed, this was still a marked improvement. A blind user would at least be aware that an image map was present even if not able to use it.

The first and second images were still ignored by JAWS as they had no ALT texts but the third image was now replaced by its ALT tag. JAWS read aloud the ALT text thus conveying some meaning of the image to the blind user. This result emphasises the importance of ALT tags and their role in increasing accessibility to web pages for blind people.

Internet explorer supports JavaScript so “hello world” was displayed at the foot of the web page. The message was presented as HTML text so JAWS had no problem in translating it into audio output. Had the JavaScript been more elaborate it may have resulted in something that JAWS could not have translated. JavaScript can be used to achieve many goals and text output is one of the simplest. I will discuss what happens with more complicated scripts in the next section.

3.5 Analysis

By using the different combinations of web browser and screen reader I experienced some of the difficulties that blind web users encounter. I could understand why the four chosen elements of web pages posed particular problems for blind people. I could also see the need for some kind of converting tool that would improve the accessibility of web pages. These issues are discussed in the following subsections.

3.5.1 Frames

Although both the Lynx viewer and Internet explorer handled frames properly, BrookesTalk did not support frames. Also, whilst Internet Explorer supported frames JAWS could not read the information within them correctly. If BrookesTalk and JAWS are not fully compatible with frames then text-only web pages should not contain frames. If a web page does not include the <NOFRAMES> tag the blind user is left with no alternative to frames. A user of BrookesTalk would find a web page without the <NOFRAMES> tag to be completely inaccessible. Based on these findings I recommended that web authors should include the <NOFRAMES> tag whenever the <FRAMESET> tag is used. This measure will ensure that an alternative to frames is always provided for blind users. I therefore proposed that text-only web pages should not contain the <FRAMESET> tag meaning that a blind user would instead see the non-frame version of a web page. Consequently, the converting tool should remove any <FRAMESET> and </FRAMESET> tags and the HTML code within them. The </FRAMESET> tag simply denotes the end of the frames section and should always accompany the <FRAMESET> tag. The <NOFRAMES> tag can remain however, as browsers will process the subsequent code in the absence of a <FRAMESET> tag. This method was hoped to provide an appropriate solution, which would result in text-only web pages that are accessible to blind people.

3.5.2 Image maps

Of the three tested solutions only Lynx could render image maps into a form fully accessible to blind users. JAWS could make the user aware that an image was present but offered no method of accessing the pages that it linked to. BrookesTalk managed even less, not giving any textual representation of the image map at all. Lynx was highly effective in the way that it handled image maps and so I hoped to recreate that efficacy with the converting tool. I proposed that my tool should replace image maps with their hyperlinks and the text “Image Map Link”. To remove the image maps the <MAP>, </MAP> and <AREA> tags were identified as those to be expunged. The <MAP> and </MAP> tags denote an image map and the <AREA> tag denotes each area within an image map.

3.5.3 Images

Images were handled rather ineptly by BrookesTalk not offering the blind user any audio equivalence at all. In contrast, Lynx made full use of whatever ALT tags were included in the web page. Once the “Show pictures” option had been turned off

JAWS also made good use of the ALT texts. From the evidence discussed it was decided that the converting tool should replace all images in web pages by their ALT texts. In order to do this the converting tool should strip out any <IMG> tags leaving only their ALT tags. This measure will increase accessibility to web pages for blind people on the condition that web authors always include ALT tags.

3.5.4 Scripts

Neither BrookesTalk nor Lynx provided support for JavaScript although an alternative was provided via the <NOSCRIPT> tag. If this tag had not been used, however, a blind user would not have been aware that they were being denied access to the full representation of the web page. For example, if I had not included the <NOSCRIPT> tag in the test web page then BrookesTalk and Lynx would not have displayed "Your browser does not support scripts". Instead, there would be no message at all, yet a user of Internet Explorer would have seen the "hello world" caption. So the exclusion of the <NOSCRIPT> tag puts blind users at a clear disadvantage to sighted users. The onus lies on the web author to include an appropriate <NOSCRIPT> tag as its complexity could be beyond an authoring tool.

Internet Explorer was able to process the JavaScript and JAWS subsequently read the output, "hello world". Although the blind user would not have suffered any disadvantage in this case, had the JavaScript been more complex then they might have done. If a JavaScript uses a lot of "document.write" (i.e. text output) scripting as the test page did then information is made visible in the source code and a screen reader can process it. However, if the JavaScript creates a menu and the menu choices are in a separate file, then the screen reader cannot access that code and so cannot convey the information to the user. So despite the fact that Internet Explorer can interpret the <SCRIPT> tag, a screen reader will not always be able to relay the output to a blind user. From this evidence it can be seen how the inclusion of the <SCRIPT> tag in a web page could reduce accessibility for blind people.

I conclude from this that web authors should always use the <NOSCRIPT> tag to provide an alternative to the <SCRIPT> tag for blind users. Text-only web pages should not contain the <SCRIPT> tag but should instead use the code between the <NOSCRIPT> and the </NOSCRIPT> tag. The <NOSCRIPT> tag itself should not be included otherwise the browsers that support scripts will not read the code after the <NOSCRIPT> tag. Therefore, I proposed the converting tool to expunge any <SCRIPT> and </SCRIPT> tags and content therein. The </SCRIPT> tag closes the section of code beginning with the <SCRIPT> tag; the two go hand-in-hand. I also decided that the text-only converter should remove any <NOSCRIPT> and </NOSCRIPT> tags, thereby leaving only plain HTML code that all browsers can interpret. Again, the </NOSCRIPT> tag simply closes the section of code beginning with <NOSCRIPT>.

This section showed that each of the four elements examined can cause problems for blind web users. The next step was to look for a solution that could overcome these problems by creating text-only web pages.

3.5.5 Proposed tools

The roots of some web page inaccessibility problems were highlighted in the last section. From the evidence discussed there was a clear need emerging for some kind of conversion tool. This tool would need to be able to take HTML files and alter them in accordance with the suggestions made during the earlier analysis. This converting tool would produce a text-only version of a web page that did not contain the four elements identified as contributing to inaccessibility.

The proposed converting tool was the key to providing web pages that are more accessible to blind people. By automating an otherwise laborious process the tool facilitates the creation of text-only web pages. This automation allows even web authors with little knowledge of HTML or the needs of blind web users to provide text-only pages. However, in providing web authors with the means to translate a graphical web page into text-only format I created the need for two additional tools. The web author needed a structure in which to store the web pages and also some method of maintenance. As discovered with MkDoc [MkDoc], easily navigable sites begin with a sound directory structure. The converting tool needed a source to read from as well as a destination to write to. To overcome this problem I chose to create a tool that could implement a parallel directory structure. The thought behind this was that graphical web pages would be stored in one directory and text-only web pages would be stored in the other. The text-only converter would then read a graphical web page from one directory, convert it, and then store it into the text-only directory.

The parallel directory structure was also planned to play a role in the maintenance of the text-only web pages. Once the text-only web pages were created it was necessary to keep them up-to-date. To ensure that this was done I settled on a system that would compare the contents of the two directories and look for differences. First the system should check to see that a text-only version of any given graphical web page exists; if it doesn't then the web author should be warned. Second, if a text-only version does exist, the last modification dates of both files should be matched. If the graphical web page has been modified more recently the web author should be alerted that the text-only version might need updating. This maintenance system was intended to run continuously.

3.6 Summary of analysis

To summarise the findings made in the analysis it can be said that the proposed tools must be able to eliminate the following elements from web pages: frames, image maps, images and scripts. These four aspects of HTML have been shown to hinder

access to web pages from the standpoint of a blind person. It will not suffice, however, to simply remove these four barriers altogether. An alternative must be provided so that the blind web user is provided with the same information as a sighted user. In the case of images this would be achieved by replacing the image with its ALT text. The objectives and design of the proposed tools are discussed in the following section.

3.7 Objectives of the tools

The analysis outlined the requirements of the proposed converting tool as being to:

- Remove all <FRAMESET> and </FRAMESET> tags from a given web page and content therein;
- Remove all <MAP>, </MAP> and <AREA> tags;
- Replace all image maps with their hyperlinks represented by the text "Image Map Link";
- Remove all <IMG> tags;
- Replace all images with their ALT texts;
- Remove all <SCRIPT> and </SCRIPT> tags from a given web page and content therein;
- Remove all <NOSCRIPT> and </NOSCRIPT> tags from a given web page;

In addition to the text-only converter were the maintenance tool and the parallel directory structure tool. The maintenance tool was intended to ensure that each graphical web page has an up-to-date text-only equivalent. The directory tool was planned to provide a framework from which the other two tools could operate.

3.8 Design

I chose Perl (Practical Extraction and Reporting Language) to implement the three proposed tools. I decided upon this particular language as Perl has many facilities for string manipulation and data reformatting, both of which I would be employing. I had no previous knowledge of any other programming languages so I was at no particular disadvantage to use Perl over other possibilities. I planned to develop the tools on a Windows platform rather than Unix due to my familiarity with the Windows environment and inexperience of Unix.

3.8.1 Parallel directory structure tool

The tool to create the parallel directory structure was designed to be the first menu option on the main web page of the user interface. When the user made this selection the tool would then create a directory named "webfiles" followed by two subdirectories named "textandimages" and "textonly". Standard web pages were to

be stored in the "textandimages" subdirectory and the newly converted text-only pages were to be stored in the other.

3.8.2 Text-only converter

I planned for the text-only converter to be a menu option on the main web page of the user interface. The user would select the option and then be presented with a text field, a "Browse" button and a "Convert" button. The user would then be faced with the choice of typing in the file name to be converted or selecting it by pressing the "Browse" button and choosing it from the file listing. The user would then click on the "Convert" button and the file name would be passed to the converting program. The program would then translate the contents of that file into a text-only version and store it in its relevant directory.

3.8.3 Maintenance tool

The maintenance tool was intended to run as a demon on the web author's computer. A demon is a process that remains dormant until a particular condition is met. In the case of this program the condition to be met was that a graphical web page did not have an up-to-date text-only equivalent. When this situation arose the demon was expected to spring into action and warn the web author that a text-only version of a web page was required. The demon would then ask the web author whether the text-only version should be automatically created using the converting tool. The demon was set to check for discrepancies every thirty minutes so as not to annoy the web author too often.

3.8.4 The user interface

The next decision to be made was how to present the tools to the user; should the software be command-line driven or should a graphical user interface be employed? I decided that the tools should be designed with accessibility at the forefront of my mind. After all, the software is aiming to improve accessibility so it should lead by example. To achieve the desired level of user-friendliness I opted for a web interface as the front-end of the system. This choice was based on the assumption that all web authors should be comfortable and familiar with a web browser. Using a web browser also meant that a blind web author could access it via their preferred method of browsing.

4 Implementation

In the previous chapter the elements of web pages that pose barriers to blind web users were discussed. The resulting list of objectives outlined what should be done to exclude these obstacles from text-only web pages. The design section then proceeded to describe a software solution that would meet these objectives. With a clear idea of what was required from the proposed tools I then had to consider how to implement them.

4.1 Options

I had two options for implementing the planned solution: take either a client-side or a server-side approach. A client-side solution would mean that the software would run from the web author's computer. The Internet would not be required to use the tools in this case, as the software would be operating in a standalone environment. A server-side solution would operate from a web server and would be accessed from the web author's computer via the Internet.

The advantage of a server-side solution is that the tools could be accessed from any computer with access to the Internet. If a web author travelled around the country using different computers then this method would eliminate the need to keep reinstalling the software. The disadvantage of adopting this approach is that the software would be inaccessible to someone without access to the Internet. Most, if not all, web authors would have access to the Internet but not necessarily twenty-four hours per day. During times when a suitably fast Internet connection was not available (for example, on a train) then the software would be inaccessible. Another disadvantage of a server-side solution is that the maintenance tool would require a permanent connection to the Internet to function correctly. The maintenance tool is designed to compare the contents of two directories every thirty minutes so a fixed connection would be necessary. Accessing the software over the Internet would obviously involve greater delays than if it were running locally from a user's computer.

The advantage of implementing a client-side solution is that a one-off installation would provide permanent access to the software. There would be no need for an Internet connection when a web author wished to create text-only pages. A client-side approach would also afford greater speed and security. All processes would occur locally on the web author's computer enabling operations to be performed quickly. No information or commands would be sent over the Internet so there could be no interference from viruses or hackers. The only disadvantage to a client-side approach is that the software would need to be installed on each computer that a web author used.

In view of the strengths and weaknesses of each approach I decided to opt for a client-based solution.

4.2 Implementing the parallel directory structure tool

Both the text-only converter and the maintenance tool required a directory structure to be in place before they could operate. For this reason I chose to implement the parallel directory structure tool first. I created a web page and placed an input button with value "Create Parallel Directory Structure" in the centre of the page. When the user clicked this button a parent directory named "webfiles" was created in the root of the current drive. The user was not given a choice as to where this directory was created due to the programming complexity and the time constraints involved. Two subdirectories named "textandimages" and "textonly" were then created within the "webfiles" directory. A message then informed the user that the operation was complete and that the directory structure was in place.

4.3 Implementing the text-only converter

With the directory structure in place I could proceed with the remaining two programs. However, as I began to implement the text-only converting tool, I realised that CGI (Common Gateway Interface) would be required. CGI is a process that passes information between a web browser and a web server. I had hoped that the user would be able to click a browse button from a web-based menu and select the file to be converted from a browse window. I planned that the file name would then be passed to the converting program. Unfortunately, CGI was required at this point to pass the file name to the converting tool. Using CGI would have involved a web server, which is exactly what I had chosen to avoid for the reasons discussed in section 4.1.

I was now forced to look for an alternative to the web-based system that I had envisaged implementing. The problem was how to capture the file name of the web page that was to be converted to text-only format. As Perl offers an input interface of its own I decided to make use of it. The line of code "$file=<STDIN>" prompted the user for a file name via an MS-DOS command window and stored it in the variable named "file". This new method forwent the web-based system in favour of a simpler approach. As a result, the cosmetic appearance of the interface was sacrificed for the sake of functionality.

However, there were advantages to the new approach; first, the MS-DOS command window is solely text-based. Therefore, any blind web authors using screen readers to use the tools should not encounter any problems, as there are no graphics. The web-based approach had graphical items such as input buttons, which some screen readers may not have been able to translate. Second, the command window only supports

keyboard input and not mouse input. This meant that the user did not have to switch between inputs allowing for quicker interaction with the program.

4.4 Redesigning the user interface

Now that I had digressed from the proposed web-based approach I had to re-evaluate the situation. I was presenting two tools via two different interfaces, which I thought might create a sense of disjointedness. Therefore, in order to keep a sense of uniformity I concluded that I should implement all three of the proposed tools using the MS-DOS command window as an interface. In discarding the web-based approach I was losing some usability of the tool. For example, the user now had to type in the file name whereas before they could select it from a browse window. This meant that the user would have to know the exact name of the file in order to type it into the system. However, as previously mentioned, the simplicity of the new system afforded greater accessibility to blind users.

Employing the command window interface meant that the programs would now be accessed by the user in a different way. With the web-based approach the user would have accessed the tools by clicking on hyperlinks in a web page. However, with the command window approach the user was faced with two choices: use a Perl engine to run the tool scripts or run an executable version of the tools. If the user wanted to run the scripts directly then they would need to install a Perl engine on their computer. A Perl engine can interpret Perl scripts and can therefore execute them, which Microsoft Windows cannot. The advantage of this method is that the programs operate slightly quicker as they are run directly from the source code. The drawback of using the Perl engine is that the user must take the time to install the Perl software first. The alternative method is to use the executable version of the scripts, which I created using Perl2Exe [Perl2Exe]. Perl2Exe is a free utility that converts Perl scripts to executable files. The Perl script for the parallel directory structure tool was named "mkdir.pl" and "mkdir.exe" was the executable version. The text-only converting tool comprised "parse.pl" and "parse.exe"; the maintenance tool comprised "demon.pl" and "demon.exe". The advantage of an executable file is that it can be run by a simple double-click upon the desktop icon representing it (although it can also be run from a command line). The disadvantage is that the tools will run slightly slower as the Perl scripts are being accessed indirectly via the executable files.

4.5 Implementing the text-only converter (continued)

Now that I had adopted a new approach I recommenced the implementation of the converting tool. Once the user had entered the name of the web file to be converted the program then looked in the "textandimages" subdirectory for that file. If the file was found it was opened for reading and a new file of the same name was created in the "textonly" directory. If the file could not be found then a message was displayed to inform the user accordingly.

The next part of the program scanned each line of HTML in the web page for certain tags. By taking chunks of each line one to eleven characters at a time the program could match them against the tags in question. Eleven was the number of characters in the largest of the tags, </FRAMESET>, including the non-alphanumeric characters. For example, a chunk of five characters might be taken and matched against the <MAP> tag. If the five characters of the code were different then the program would move on one letter and try again. If a match was found then the appropriate action would be taken, which could be to eliminate the whole line of code or just the tag itself.

HTML is not case sensitive so <map> has exactly the same meaning as <MAP>. To account for the possible case differences the converting tool translated all the HTML code in a given file to uppercase before it commenced matching. Perl has an in-built operator, “tr”, that does the case translation automatically. For example, the Perl code: “$word=~tr/a-z/A-Z/;” would translate the contents of the variable “word” into uppercase.

The program copied each line of HTML into the new file unless one of the specific tags was located. As outlined in Chapter three, these tags were <IMG>, <MAP>, </MAP>, <AREA>, <SCRIPT>, </SCRIPT>, <FRAMESET>, </FRAMESET>, <NOSCRIPT> and </NOSCRIPT>. When the converting program located these tags they were dealt with accordingly, some being completely expunged and some being edited.

The following is an example of an <IMG> tag:

```
<IMG NAME="poet" SRC="wby.jpg" width=160 height=220 align="right" >
```

In the case above, no part of the tag would be copied into the new text-only file as there is no ALT tag present. If an <IMG> tag had an ALT tag such as the one below then the program would deal with this differently.

```
<IMG SRC="abc.jpg" ALT="picture" WIDTH=160 HEIGHT=220 BORDER="0">
```

The converting tool would extract the ALT tag from the above line of code. This is because all images should be replaced by their ALT tags, as concluded in section 3.7. So, from the above line of code, “picture” would be the only part copied into the new text-only file.

A problem I encountered whilst parsing <IMG> tags was how to identify the end of an ALT tag. ALT tags are often enclosed by quotation marks in which case the end of the ALT tag can easily be identified by the second mark. However, quotation marks are not compulsory and if they are not used then another method must be employed to

distinguish the end of an ALT tag. To combat this problem I discovered that there is a finite set of possibilities that can follow an ALT tag. If an ALT tag is not followed by an “>” then it is followed by either “WIDTH”, “HEIGHT”, “BORDER”, “ALIGN”, “USEMAP” or “ISMAP”. The converting tool therefore copies a line of code from “ALT” until one of these possibilities occur, at which point copying is suspended once again.

In order to expel image maps from the web page the <MAP> and </MAP> tags had to be removed; this was done by matching five characters at a time to the term “<MAP>”. Once a match was found the program stopped copying to the new file and started to look for the </MAP> tag, which signified the end of the image map code. By taking six characters at a time, the extra one being for the “/”, and matching them against the term “</MAP>” the task could be completed. Once the </MAP> tag was found the converting tool would begin to copy the subsequent code. This process eliminated the <MAP> and </MAP> tags as well as the code within them.

Although image maps themselves were to be expelled, they were to be replaced by their hyperlinks. An image map’s hyperlink resides in the <AREA> tag. The <AREA> tag is found within the <MAP> and </MAP> tags; it defines each area of an image map. Any <AREA> tags are automatically omitted as the code between the <MAP> and </MAP> tags is not copied to the new file anyway. However, the <AREA> tags contain an essential piece of information, namely the URLs to which the image map links. Below is an example of an <AREA> tag:

<AREA shape=rect coords="0,0,50,83" HREF="sport.html" ALT="Sport News">

One of the aims of the converting tool was to replace image maps with their corresponding hyperlinks. The beginning of the hyperlink is signified by “HREF=”, so “sport.html” should be extracted from the above tag and copied into the new file. However, “HREF” is not solely used within image maps, it can be used elsewhere in web pages. For this reason it was necessary to set a flag to a value of one when the program encountered an <AREA> tag. In this way it could be decided whether an HREF tag was to be copied or not. If the AREA flag was set to zero then the HREF tag was not to be copied but if the flag was set to one then the HREF tag should be copied. Just copying the HREF tag to the new web file did not suffice however, as a complete hyperlink is created by the following code:

<A HREF=”sport.html”>Image Map Link</A>

Therefore the additional code was added to the new web file by the converting tool itself. The following code represents a simple image map:

<IMG SRC="abc.jpg” ALT="image map" ISMAP USEMAP="#abc">
<MAP name="abc">

<AREA shape="rect" coords="10,15,74,93" HREF="news.html" alt=news></MAP>

The text-only converter would translate the image map code to:

"image map"
<A HREF="news.html">Image Map Link</A>

As can be seen, the <IMG>, <MAP>, <AREA> and </MAP> tags have been expunged leaving only the image map's ALT tag and the image map hyperlink.

The <SCRIPT>, </SCRIPT>, <FRAMESET> and </FRAMESET> tags were removed in a similar way to the <MAP> and </MAP> tags. Each term was matched to a similar number of characters and the copying process was either suspended or resumed when a match was found. The converting tool would stop copying when the <SCRIPT> tag occurred and resume copying when the </SCRIPT> tag occurred. This process ensured that both the tags themselves and the code within them were expunged from the new text-only web file.

The <NOSCRIPT> and </NOSCRIPT> tags were handled slightly differently from the other tags because the code within them was to be copied to the new file. To achieve this the program would stop copying when either of the tags were found, then jump to the ">" character and resume copying. In this way only the tags themselves were omitted from the new text-only web file and not the HTML code within them.

4.6 Redesigning the parallel directory structure tool

Due to the fact that I had discarded the web-based approach, it was now necessary to redesign the parallel directory structure tool. I had previously implemented the tool via a web page but now I had to present it via the MS-DOS command window. This action only warranted a slight adjustment since it was the interface to the program that required amending and not the program itself. The source code of the parallel directory tool was simply converted to executable format and accessed via a double-click on its icon rather than via a web page. Typing "mkdir.exe" at the command line could also run the tool.

4.7 Implementing the maintenance tool

The third and final part of the implementation was the maintenance tool. The purpose of the maintenance program was to ensure that once created, the text-only web pages did not fall into a state of disrepair. Also, the program was designed to check for new web pages, which might require converting to text-only format.

In order to compare the contents of the two directories storing the web pages, a matching algorithm was required. I began by reading the filenames from the directory

"textandimages" into one array and the filenames from the directory "textonly" into another array. By doing this I created two numbered lists of filenames, which could be easily compared using a while loop. The matching algorithm I used looked at the name of the first file from the "textandimages" directory and then compared it with all the filenames in the "textonly" directory. If a text-only version of the web file was found then the program looked at the last-modification dates of the two files.

Perl has a useful in-built function called "stat", which returns an array of information for any given file. The ninth element of this array gives the number of seconds after a given date that a file has been modified; these seconds are known as epoch seconds. Therefore, if a file has been modified more recently than another it will be represented by a higher number of epoch seconds. The text-only version of a web page was deemed to be up-to-date if it was the most recently updated file, or to be precise: if the number of epoch seconds was greater for that file. If this were the case, the algorithm skipped to the next file in the "textandimages" array and looked for its text-only equivalent. Otherwise, the program informed the user that the text-only version of the file required updating. The user was then asked whether they would like to update the file or not. If the response was positive then the maintenance tool passed the filename to the converting tool to do the updating. If the response was negative then the program moved on to the next file in the array. This process was repeated every thirty minutes to check for recently modified web pages.

If the web file was found not to have a text-only equivalent then the user was asked whether they wished to create one. If the reply was affirmative then the filename was passed to the converting tool and a text-only version was created. If the user did not want to create a text-only version then the program skipped to the next file in the array. The matching algorithm repeated this process for each file that had been read into the array from the "textandimages" directory. This process was repeated every thirty minutes to ensure that recently created web pages also had text-only equivalents.

The two directories used by the maintenance tool should not contain anything other than HTML files. If, for example, image files were stored in the "textandimages" directory, they would also be processed by the matching algorithm. This would result in the user being asked if they want to convert the image files to text-only format. The "textandimages" label refers to web pages that comprise text and images rather than text files and image files. To avoid this situation the web author could store any non-HTML files in separate directories.

4.8 Constraints

The two main constraints that I found myself limited by during implementation were time and my knowledge of Perl. Due to the fact that I was learning Perl whilst I was writing the scripts for the tools I could not devote all my time to just programming.

My inchoate knowledge of Perl sometimes meant that I was forced to take the long way round a problem where a quicker solution might have been available. It is almost certain that problems I thought insoluble could in fact have been unravelled by methods I was simply unaware of. Due to the time limit I was unable to implement fully the user interface design I had hoped to create. I would have preferred a more pleasing-to-the-eye interface than the command window approach adopted. However, I did not have unlimited time and so compromises sometimes had to be made.

4.9 Summary

The three tools that were implemented were designed to aid the creation and maintenance of text-only web pages. It was hoped that web authors could employ these tools to provide a greater number of accessible web pages to the blind Internet population. The tools could be used to both create new text-only web pages and to keep existing pages up-to-date. In order to measure the success of the software an evaluation process was designed. The next chapter will look at the design and results of the software evaluation.

5 Evaluation

The evaluation process was an essential part in the development of the tools. By evaluating the three tools it would become clear as to whether they had accomplished their objectives or whether further work was required. In order to assess the effectiveness of the software it was necessary to develop a method of appraisal.

5.1 Design of evaluation

The tools to be evaluated had three purposes:

- To create a parallel directory structure
- To convert a web page to text-only format
- To ensure the text-only web pages were kept up-to-date

Therefore, the evaluation process had to test these objectives and then measure the success of the tools from the results. The first task was to test whether the parallel directory structure was created correctly. This could be achieved quite simply by running the "mkdir.exe" file from a command line or by double-clicking on the "mkdir.exe" desktop icon. Once the program had been executed I could then use Windows Explorer to verify that the parallel directory structure was in place.

The second task to be evaluated was more complex and required a more rigorous testing procedure. A number of web pages were required for the converting tool to translate into text-only format. These web pages could either be selected from the Web or written by myself. The main requirement of the web pages was that they should contain at least some of the elements identified in Chapter three as being obstacles to blind people, namely: frames, image maps, images and scripts. Once these web pages were decided upon the converting tool could process them. The resulting set of text-only web pages could then be accessed using the same combination of browsers and screen readers as in Chapter three. The three browser solutions used were:

- The BrookesTalk specialist browser
- A Lynx viewer with the JAWS screen reader
- Microsoft Internet Explorer with the JAWS screen reader

These three methods could be used to view the text-only web pages and thus gauge their accessibility. The audio output from each browser or screen reader would then be noted so that it could be compared with the output produced from the standard web pages. In this way it could be seen how much of a difference the text-only format would make for a blind person.

The first website I decided to use was the one I developed during the analysis stage. The results of how this page was interpreted by the three browser solutions had already been discovered in Chapter three. I could therefore compare these results with those of the text-only pages created by the converting tool. However, as the development web page was designed with the tools in mind it would not be sufficient to end the testing procedure here. A more realistic testing environment would be the Web where the tools would, I hope, be used in future. For this reason I chose two further websites for testing: the University of York and the University of Manchester. This might seem like a narrow cross-section but the three websites contained a mixture of the elements that I wished to test. Therefore the test sample was thought to be sufficient to evaluate the text-only converting tool.

The web page that I developed in Chapter three contained frames, images, JavaScript and an image map. These are the four items that were identified as hindering web page accessibility for blind people. The University of York homepage contained both images and an image map. The University of Manchester home page contained images with and without ALT tags as well as JavaScript. The tools I created were designed to work purely with the four elements of HTML code mentioned. Therefore, I tried to evaluate the tools using web pages that only contained these four aspects of HTML. Web pages written using other languages such as Extensible Markup Language (XML) or Active Server Pages (ASP) were deliberately avoided. It should be noted that the University of York does provide a text-only alternative to its website; its standard pages were used to highlight the obstacles within web pages that blind users encounter.

The three combinations of browser and screen reader were to be used in two ways to exploit their inherent differences. BrookesTalk and Internet Explorer do not alter the graphical representation of web pages as Lynx does. Internet Explorer simply displays web pages in their original form, without any rendering towards improving ease-of-use for blind people. BrookesTalk aspires to improve accessibility by extracting information from a web page and presenting it in an accessible format. For example, links from a web page will be grouped together and presented in the links menu at the top of the browser. However, as discovered in Chapter three, BrookesTalk does not offer access to image maps or ALT tags. Therefore it was hoped that web pages, which were previously inaccessible to some degree via BrookesTalk and Internet Explorer, would be rendered more accessible by the converting tool.

The purpose of the Lynx viewer is to render a text-only format of a web page, which is the same objective of the converting tool. Therefore, once the three websites had been converted to text-only by the tool they should be similar to those accessed through the Lynx viewer. The standard web pages and the text-only pages were not expected to differ much when viewed through Lynx. To be precise, the text-only web

pages viewed with BrookesTalk or Internet Explorer should be similar to the standard pages viewed with Lynx.

In addition to this method of evaluation the online validator Bobby (discussed in section 2.5.4) could have been used. However, this validator is designed to improve the overall accessibility of web pages and is not purely focussed upon blind people. Bobby aims to improve the use of colour, promote correct use of tables, enforce the use of ALT tags and more; these are all design issues that web authors themselves should be attending to. The tools I developed were not intended to tackle these design issues; they were intended to automate the removal of HTML code causing inaccessibility for blind people. Whilst Bobby could be used to evaluate the new text-only web pages it would be more suited to analysing those pages designed for universal access.

Bobby produces an online report that highlights each line of code that fails to meet a W3C web accessibility standard. For example, Bobby returned over 200 instances of sub-standard code in the University of Manchester web page. The report demands quite a substantial amount of alterations to the website, all at a different level of importance. These alterations could be of minor importance or of major significance. If an important obstacle was removed from the page then the report might only return 199 instances. However, this reduction would not acknowledge the importance of that one alteration. If the alteration had made only a slight difference to accessibility then this would still have the same effect with the report now citing 199 instances. For this reason, the report produced by Bobby was not suitable for evaluating the web authoring tools.

The next program to evaluate was the maintenance tool, whose task it was to ensure that there was a text-only equivalent of each web page. The maintenance tool compared the contents of two directories and informed the web author when a text-only web page required creating or updating. To test that this tool functioned correctly it would be necessary to have a parallel directory structure in place. By populating the “textandimages” directory with some standard web pages I could begin to see the tool in action. The user should be prompted by the tool to create text-only versions of the standard web pages. If the standard web pages were then modified the tool should prompt the user to update their text-only equivalents. If the user then decides not to update the text-only files, for whatever reason, the maintenance tool should ask the user again thirty minutes later.

The final part of the evaluation was to evaluate the tools using the W3C’s Web Accessibility Initiative guidelines on authoring tool design [WAI tools]. This document has a number of checkpoints with different priorities that authoring tool designers should attempt to satisfy. The W3C deem the following checkpoints to be essential when designing an authoring tool:

- Ensure that the author can produce accessible content in the markup language supported by the tool.

- Ensure that the tool preserves all accessibility information during authoring, transformations, and conversions.

- Ensure that the tool automatically generates valid markup.

- Do not automatically generate equivalent alternatives.

- Document all features that promote the production of accessible content.

- Allow the author to change the presentation within editing views without affecting the document markup.

.

- Allow the author to edit all properties of each element and object in an accessible fashion.

These checkpoints were planned to be used to further evaluate the text-only converter tool.

A more comprehensive evaluation process would have involved the participation of a number of blind people. However, due to time constraints it was not viable to recruit a sample of blind people to test the text-only web pages for accessibility.

5.2 Results

The evaluation method discussed in section 5.1 was applied to the three software tools. The following three sections review the results of each tool.

5.2.1 Parallel directory structure tool

I ran the parallel directory structure tool by double-clicking on its desktop icon. The program performed as expected and created the structure of one parent directory and two child directories.

5.2.2 Text-only converting tool

With the parallel directory structure in place the text-only converting tool could now be tested. The three web pages being used for the evaluation were copied into the "textandimages" directory in preparation for conversion. The first page to be tested was the purpose-built web page. Figure 3a shows the web page as viewed with BrookesTalk. As discovered in Chapter three, BrookesTalk did not convert the ALT tag of the third image to speech output. Furthermore, BrookesTalk did not offer any verbal representation of the image map. In brief, a blind user would be denied access to all three images and the image map.

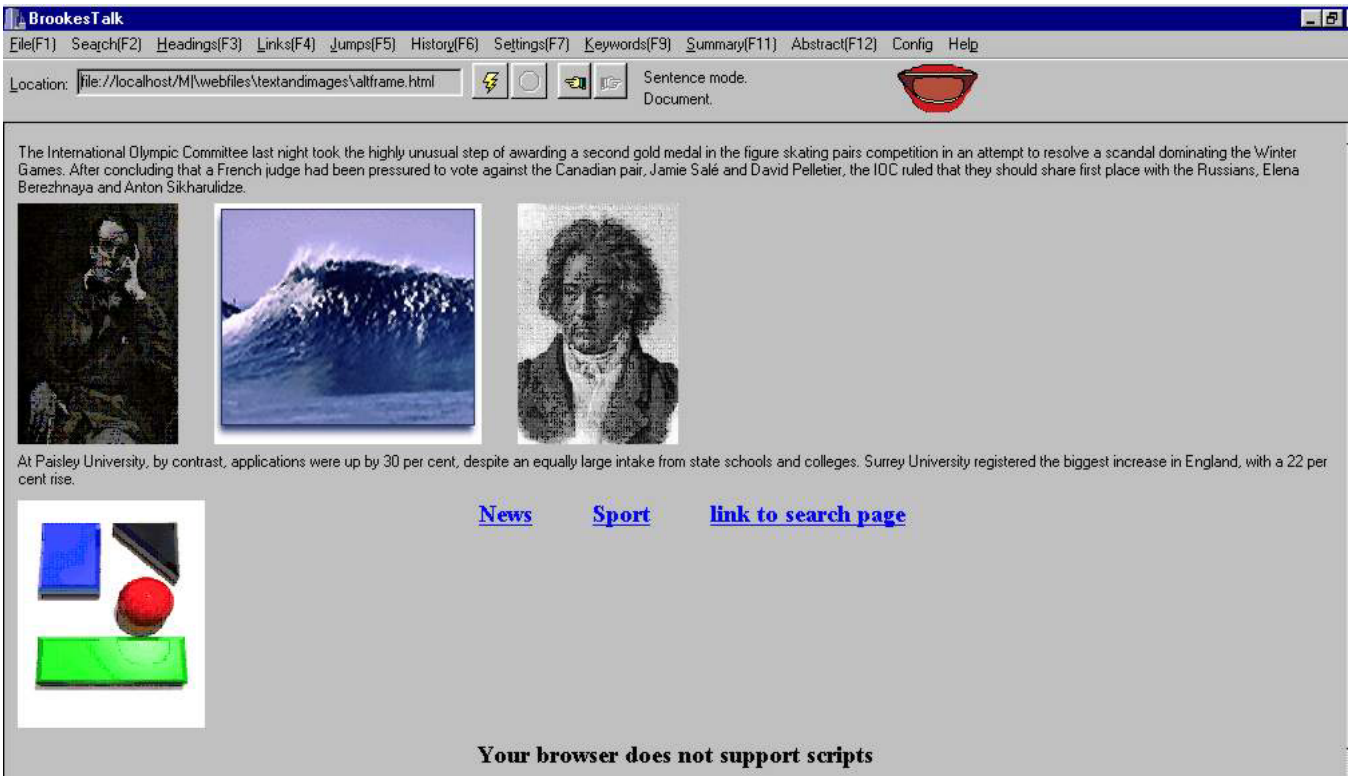

Figure 3a. Purpose-built web page as viewed with the BrookesTalk browser.

The web page was then converted to text-only format by using the converting tool. The page was accessed again using BrookesTalk; figure 3b shows the resulting text-only version. The images and image maps were removed by the converting tool and replaced by their ALT texts. The first two images did not have ALT texts but the third one did and this can be seen in figure 3b as "picture of Ludwig Van Beethoven". BrookesTalk did not offer a verbal representation of this image of Ludwig Van Beethoven when the standard web page was accessed. However, once the text-only converter had replaced the image with its ALT text, BrookesTalk converted the text to speech. As a result, a blind user would now be aware of this previously inaccessible information. Similarly, the image map was replaced by its ALT text and its four hyperlinks. None of this information was available to the blind user when it was in graphical form. However, the converting tool provided BrookesTalk with a textual rendering of the image map, which it could translate to speech output.

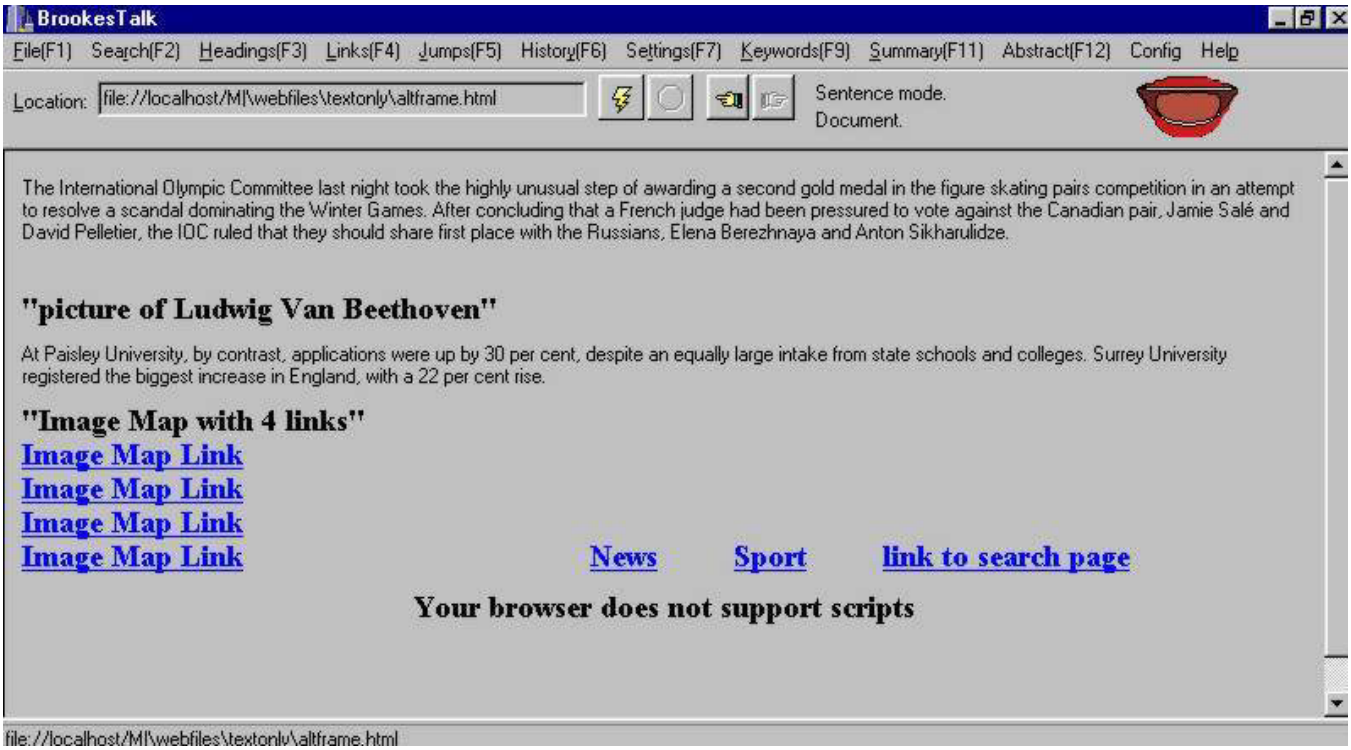

Figure 3b. Text-only version of the purpose-built web page as viewed with the BrookesTalk browser.

The same web page was viewed using Microsoft Internet Explorer to see if the converting tool had a different effect with another browser. Internet Explorer supports both frames and scripts so it was expected that the text-only pages would provide an even greater benefit for the blind user than in the previous case. This hypothesis was made because the tool would now be dealing with frames and scripts in addition to images and image maps. The JAWS screen reader was used in conjunction with Internet Explorer to convert the text into speech output.

As discovered in Chapter three, frames disrupted the order in which JAWS read out the contents of the web page. The main body of text was split up by the screen reader inserting text from the adjacent frame. The images were only verbally represented if they had ALT tags and if the "Show pictures" option was deselected. JAWS conveyed the content created by the JavaScript, as it was a simple text string. If the JavaScript had created a multi-layer menu then JAWS would not have been able to offer a speech representation. The links within the image map remained completely inaccessible to the blind user, as JAWS could not produce a verbal equivalent to the hyperlinks.

The converted text-only page improved the accessibility of the web page considerably. The converting tool removed the frames section from the web page thus enforcing the use of the no-frames version. This action resulted in the information within the page being read aloud by JAWS in the intended order. The images and image map were now removed and replaced by their ALT texts so that

they could be rendered into a verbal form. The links from the image map were also displayed as text, which JAWS was able to convert to speech output. The converting tool removed all <SCRIPT> tags leaving the <NOSCRIPT> section. This meant that the JavaScript alternative was employed, which in this case was just a different line of text to the one that the JavaScript created. Continuing the example of the multi-layer menu, the JavaScript alternative might have comprised a simpler HTML-based menu. In this situation the text-only converter would make a significant difference to the accessibility of the page by providing access to the alternative menu.

The first test web page was accessed with the Lynx viewer to complete its evaluation. Lynx replaced the images and image maps with their ALT tags (where present). Lynx displayed the no-frames and no-script version of the page thereby producing simple text output, which the JAWS screen reader could convert to speech. The Lynx rendering of the standard graphical web page was almost identical to the text-only conversion of the web page, as expected.

The second web page to be tested was the University of York's home page. The graphical version of the web page was opened in BrookesTalk to see how the different elements were handled. The main image of the page had the ALT text, "Welcome to the University of York" yet BrookesTalk did not read this aloud. The remainder of the page consisted of three image maps each containing a menu. BrookesTalk offered no verbal representation of these image maps, thus rendering the page almost inaccessible to a blind user. Fortunately, the University of York offer a text-only version of their website and this link was displayed and converted to speech output by BrookesTalk. However, the thirteen menu options that composed the site map remained inaccessible to the blind user.

The text-only conversion of the second test page substantially improved accessibility. The main image of the page was now replaced by its ALT text and converted to speech output by BrookesTalk. Furthermore, the hub of the page – the thirteen menu options – was now represented by text hyperlinks. These menu options, which were previously embedded within image maps, were extracted by the text-only converter to provide a fully accessible web page.

The University of York web page was now accessed using Internet Explorer and JAWS. A blind person using this combination of browser and screen reader would have found the web page almost completely inaccessible. Once again, the only part of the page that the blind user was presented with was a news link and a link to the text-only version. Indeed, this might be all that a blind user would require of the page. However, if a text-only version did not exist then the user would be left with one news article to read and nothing further.

In contrast, the text-only version of the page created by the converting tool afforded accessibility to all ALT texts and menu options. The JAWS screen reader was able to

convert the menu options into speech output, as they were now represented as hyperlinks and not image maps. The text-only converting tool transformed a near-inaccessible web page into a fully accessible web page.

The web page as seen through the Lynx viewer can be seen in figure 4a. The image has been replaced by its ALT text, "Welcome to the University of York" and the three image maps have been replaced by their ALT texts. The three image map ALT texts are, "See link for text-only version", "Right-menu panel" and "Navigation bar". The Lynx viewer does not emulate a Lynx browser perfectly; a Lynx browser would represent all hyperlinks associated with an image map.

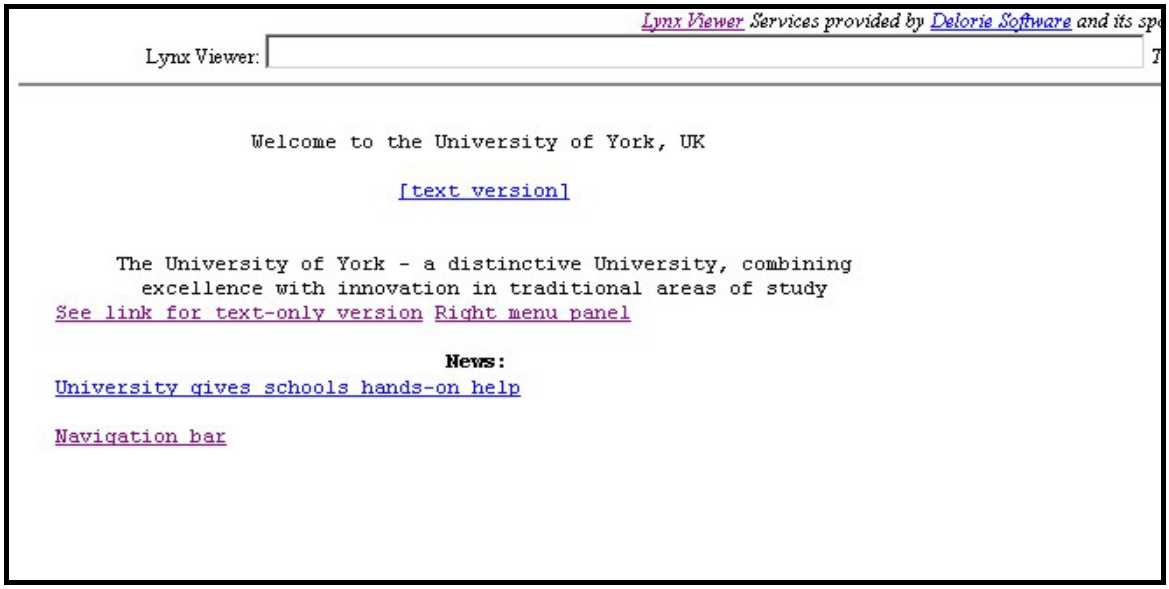

Figure 4a. York University home page as seen with the Lynx viewer.

The text-only version of the web page, as created by the converting tool, is shown in figure 4b. All of the image map links have been extracted and displayed as text in a similar way to that which a Lynx browser would have done. The evident problem with having so many image map links is that the blind user would not be able to distinguish one from another. To provide even greater accessibility to the web page it would be better to represent the image map links with differing text. Either the ALT tag or the URL of an image map link might provide a suitable text representation.

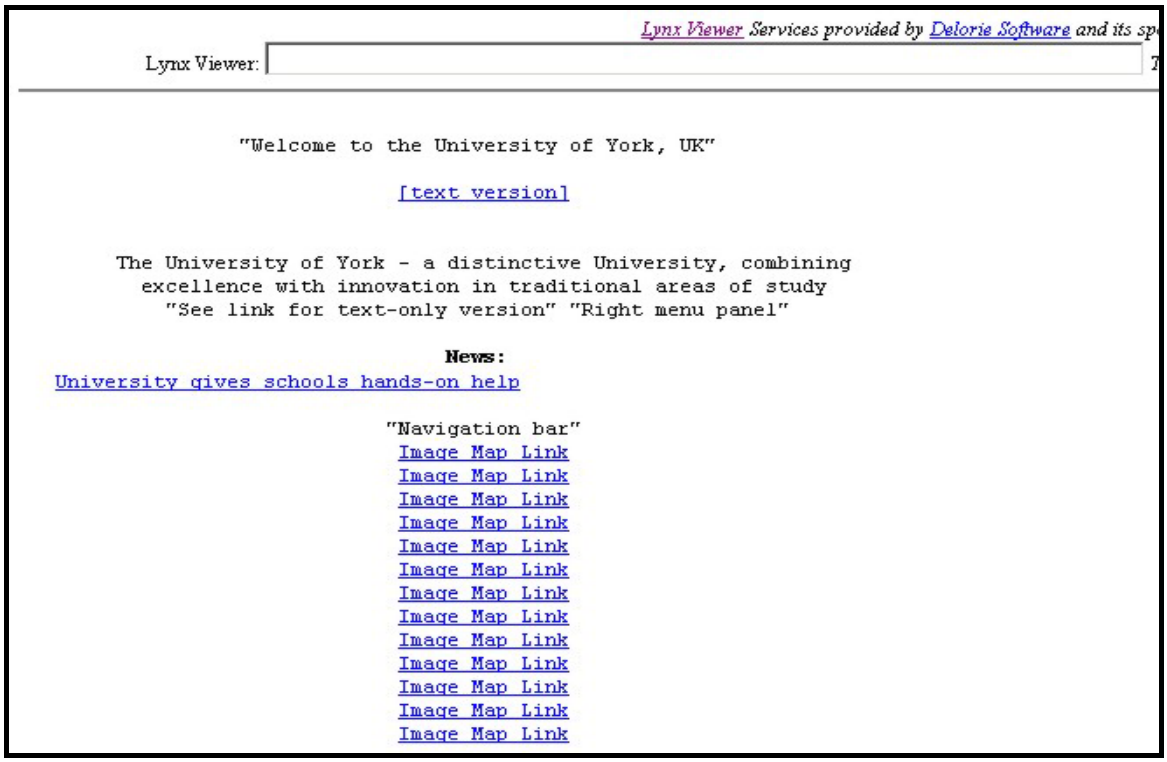

Figure 4b. York University home page after text-only conversion as seen with the Lynx viewer.

The third and final website used in the evaluation was the home page of the University of Manchester. This web page had a number of images, some of which did not have ALT tags. Several of the images contained text and were menu options; these can be seen on the left-hand side of the page in figure 5a beginning with "Welcome". JavaScript was also used in the page to operate the drop-down menu on the right-hand side of the screen.

When viewed through the BrookesTalk browser this web page was only partly accessible. The University of Manchester logo did not have an ALT tag so a blind user would have been unaware of the image. BrookesTalk did not convert the eight menu options on the left-hand side of the page to speech output. This prevented a blind user from being able to explore a large part of the website. Nonetheless, the body text in the centre of the page and the links to the right of the page were all accessible.

The text-only conversion of the web page produced a more accessible format. The eight menu options were replaced by their ALT tags and converted to speech output by BrookesTalk. The images without ALT tags were removed, as they were not conveying any information to the user, they were more for cosmetic appearance. The drop-down menu remained inaccessible in both versions of the web page, as there was no alternative to JavaScript provided by the web author.

Figure 5a shows the University of Manchester web page as displayed by Microsoft Internet Explorer. The JAWS screen reader did not produce a very accurate verbal representation of this web page. The main reason for this was that the web author had used tables to layout the information. As highlighted in section 2.3, the W3C advise against using tables for layout purposes as this causes problems for screen readers. Instead of reading down the columns, JAWS read across the page thereby mixing text strings to create nonsensical sentences. Also, JAWS did not convert the menu options on the left-hand side of the page to speech output, so a blind user would have been precluded access to a large part of the site once again.

Figure 5a. Manchester University home page as viewed with Internet Explorer.

The text-only conversion of the web page offered slightly improved accessibility. The resulting page is shown in figure 5b. The menu options on the left-hand side of the page were converted from images to their ALT texts, which JAWS subsequently converted to speech output. However, the general layout of the document was not altered due to the fact that the converting tool was not designed to handle tables. With the tables still in place JAWS could still not render a sensible verbal representation of the page.

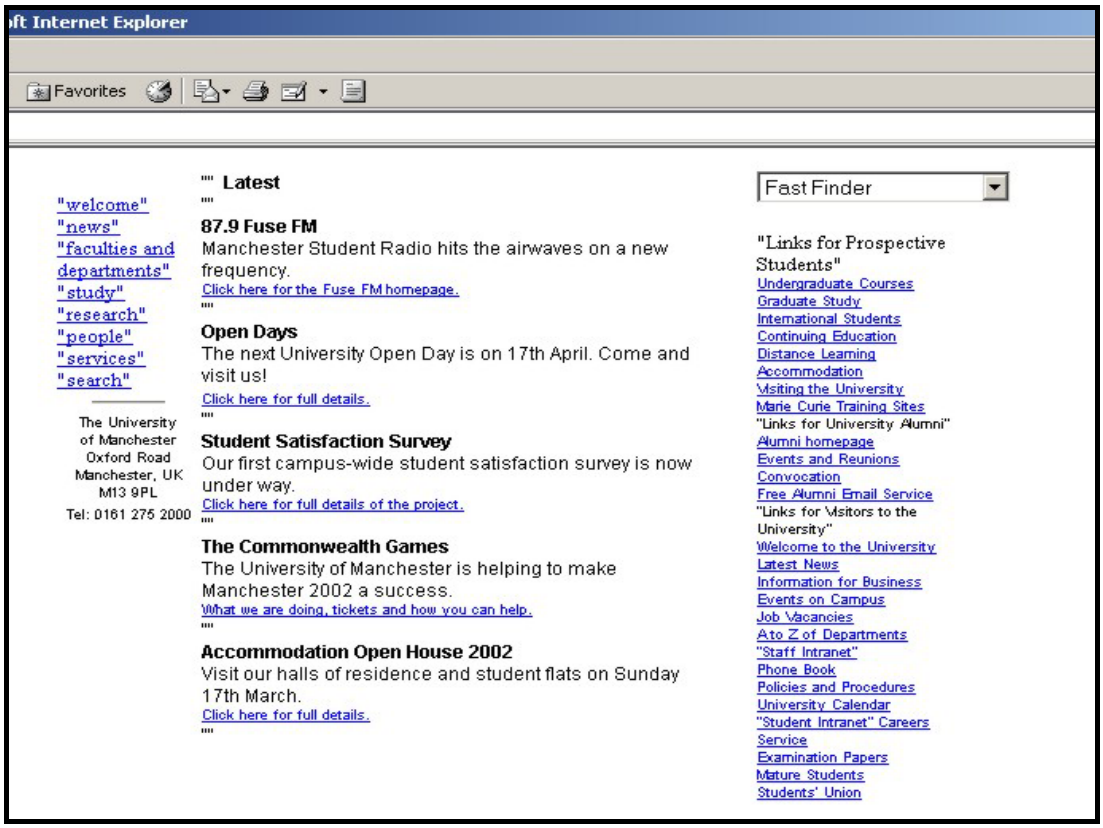

Figure 5b. Manchester University home page after text-only conversion as viewed with Internet Explorer.

The Lynx viewer produced a different representation of this web page compared with the text-only conversion in figure 5b. Lynx is designed to handle tables and it rearranged the page so that the information was displayed vertically. The first column with the eight menu options was displayed first followed by the text in the centre of the page followed by the long list of links from the right-hand side of the page. This meant that JAWS could read the page one line at a time from left to right and still make sense. Although the converting tool was not designed to accommodate tables it is clearly an important issue, which might be considered in future work.

The drop-down menu remained inaccessible even through the Lynx viewer. The text-only conversion of the web page could not improve the matter as there was no <NOSCRIPT> tag to provide a substitute. If the web author had provided an alternative to the menu then the text-only converter would have used it to make the menu accessible to blind users. This was a good example of when an alternative to JavaScript should be used to avoid putting blind people at a disadvantage.

5.2.3 Maintenance tool

The three web pages that were used to evaluate the converting tool were placed in the "textandimages" directory in order to test the maintenance tool. Once these files were in place the maintenance tool was designed to check whether they had a text-only equivalent in the "textonly" directory.

The maintenance tool was launched by running the "demon.exe" file from the command line. As expected, the tool informed the user that the file "altframe.html" did not have a text-only equivalent (this was the purpose-built web page) and asked whether one should be created. The program was requested to convert the file to text-only and the corresponding file appeared in the "textonly" directory as planned. The converting program then displayed a message to say that the file "york.html" did not have a text-only equivalent. The maintenance tool was asked not to create a text-only version of the University of York web page for the moment. The program then moved on to the third file in the directory, "manchester.html" and prompted the user for conversion. The program was asked to create a text-only format of the University of Manchester web page and it did do so correctly.

There were now three web files in the "textandimages" directory but only two of them had text-only equivalents. Therefore, after thirty minutes, the maintenance tool prompted the user to convert "york.html" again. Thirty minutes was the time decided in section 3.9.3 that would cause minimal disruption to the web author. This time the file was converted to text-only and subsequently appeared in the corresponding directory.

The next feature of the maintenance tool to test was whether it would warn the web author when a text-only page was not kept up-to-date. In order to assess this aspect of the program a slight adjustment was made to "altframe.html" in the "textandimages" directory. Within thirty minutes a message appeared informing the user that the file had been altered. The user was then asked whether they wished to update the text-only equivalent of the web page. The program was requested not to update the text-only version of the file at that moment. The maintenance tool prompted the user again thirty minutes later and this time the program was asked to update the text-only file.

5.3 Usability of the software

The final part of the evaluation was to measure the usability of the software itself. The results hitherto have shown that the software tools function correctly but how easy are they to use? To help answer this question the W3C's Web Accessibility Initiative guidelines were used. Figure 6 summarises the guidelines and looks at how the web authoring tools performed in relation to them.

Guideline	Passed?	Comment
Ensure that the author can produce accessible content in the markup language supported by the tool.	✗	Tool should allow user to include accessibility information.
Ensure that the tool preserves all accessibility information during authoring, transformations, and conversions.	✓	Proven by results.
Ensure that the tool automatically generates valid markup.	✓	Proven by results.
Do not automatically generate equivalent alternatives.	✗	Prompt user for ALT texts for images.
Document all features that promote the production of accessible content.	✗	Provide a help system.
Allow the author to change the presentation within editing views without affecting the document markup.	✗	First need to provide graphical user interface.
Allow the author to edit all properties of each element and object in an accessible fashion.	✗	As above.

Figure 6. Usability of software in accordance with W3C's WAI guidelines.

The results show that the three authoring tools did not pass all the criteria set by the W3C. However, the software still has a lot of scope for expansion and would be able to meet the full set of guidelines with more development. The first guideline requires an authoring tool to allow the web author to include accessibility information when adding an object. For example, if the user added a video clip to a web page then the tool could allow the user to add an accompanying transcript. This guideline and the final two guidelines could be met once a more sophisticated user interface design was in place.

The converting tool has proved that it not only preserves the HTML code in web pages but can also automatically generate it. The test results showed that some HTML code was modified, such as images being replaced by their ALT tags. Also, new HTML code was introduced in the instances where image maps were replaced by hyperlinks represented by "Image map link".

The text-only converter does not prompt the web author for ALT texts so the fourth criterion was not satisfied. A modification to the program might cause image tags without ALT texts to be brought to the web author's attention. This would certainly ensure greater accessibility for blind people and would add to the overall design of the web page.

The software did not meet the fifth guideline, which suggested the provision of a help system. The current set of tools is quite simple and easy to use so a help system was

not deemed necessary. However, a user manual or online help guide might be required should the tools be further developed.

5.4 Summary

The evaluation procedure has shown both the parallel directory structure tool and the maintenance tool to perform exactly as expected without error. The performance measurement of the text-only converter tool was, however, more complex and more subjective. Images were successfully replaced by their ALT texts and image maps were replaced by their hyperlinks. It was recognised that the image maps would have been more accurately represented by either their ALT tags or their URLs rather than the text, "Image map link". The converting tool some times relied upon the web author to include certain tags such as the <NOSCRIPT> tag and the <NOFRAMES> tag. This meant that in the absence of these tags the text-only equivalent could not always improve accessibility without the intervention of a web author. Figure 7 summarises the results from the testing of the three web pages.

Browser	Web format	Frames	Image Maps	Images	Scripts
BrookesTalk	Standard	✓	✓	✓	✓
	Text-only	✗	✗	✗	✓
Internet Explorer	Standard	✓	✓	✓	✓
	Text-only	✗	✗	✗	✓
Lynx	Standard	✗	✗	✗	✓
	Text-only	✗	✗	✗	✓

Figure 7. HTML elements that hindered accessibility for blind users.

As the Lynx browser has a similar job to the converting tool the results were the same for both the standard and text-only web pages. The results in Figure 7 show that the converting tool greatly increased accessibility for users of BrookesTalk and Internet Explorer. However, due to the absence of script alternatives the JavaScripts still caused problems in the text-only web pages.

6 Conclusion

The testing of the three web authoring tools produced encouraging results, as can be seen in Chapter five. The text-only converting tool was successful in eliminating those elements of web pages that were identified as hindering accessibility. However, it was found that there was a limit to how far the converting tool could improve accessibility. Instead of endeavouring to adapt the tool to correct an ever-increasing number of design faults, web authors should be looking to avoid these design flaws in the first place. Alternative tags such as the <NOFRAMES> and the <NOSCRIPT> tags are essential if blind web users are to gain access to the same information as sighted users. The ALT tag is also crucial in order to convey graphical information to the blind user.

6.1 Criticism

Despite the goals of the tools being achieved there were certain aspects that could be improved upon. Although the converting tool rendered image maps accessible by extracting their hyperlinks there was still room for improvement. The hyperlinks were represented by the text “Image Map Link” and so did not give the blind user any idea of what lay behind the link. A better solution might have been for the hyperlinks to be represented by their ALT tags or by their URLs. This would offer some differentiation between links rather than several occurrences of “Image Map Link”.

The converting tool did not attempt to give the text-only web pages any particular layout. This could result in some pages appearing rather untidy, although a screen reader would still be able to read them. However, as text-only pages are not only used by blind people, the layout of the pages should have perhaps been taken into account. The onus must lie with the web author to create a good first-time design so that web pages still look neat after text-only conversion.

The maintenance tool did its job of checking every thirty minutes for text-only files that required updating. However, the warning to the web author was displayed in a command window, which would not always be maximised. Therefore, either the command window should be always on top of other windows or a warning message should be flashed up for the web author. The frequency that the tool checked for updates was fixed at thirty minutes; depending upon the web author, this might have been too short or too long a time delay. A better design might have enabled the web author to decide upon the time delay him or herself.

The parallel directory structure tool could have been more flexible in its operation. The user was not given a choice as to where the directory structure was set up. A more user-friendly approach could have prompted the user to enter a directory location in which they wished the structure to be placed.

6.2 Further work

Although the objectives set out in section 3.8 were met there remains a large scope for further work. The text-only converting tool has tackled the issue of images, image maps, frames and scripts in web pages but other obstacles remain. Tables and forms are but two of the other barriers that hinder web accessibility for blind people. Further work might look to expand upon the capabilities of the converting tool and include these two further elements of HTML.

The converting tool was designed to eliminate certain elements of HTML code in order to improve accessibility. A complementary approach could look at not only eliminating problematic code but also inserting advantageous code. Code to be introduced into a web page would be tags such as the ALT, <NOFRAMES> and <NOSCRIPT> tags. This would remind web authors of good practice and thus improve the design and accessibility of text-only web pages.

There are a plethora of web design guidelines and web authoring tools available on the Web today. If web authors use these existing means to full effect then web page accessibility should be on the rise. Unfortunately, the content of the Web is created and maintained by a wide cross-section of people with varying abilities and conformity cannot be guaranteed.

7 References

[ADA, 1990] The Americans with Disabilities Act of 1990:
http://www.usdoj.gov/crt/ada/statute.html

[Beckett, 1997] D. Beckett, *30% Accessible - A Survey of The UK Wide Web*, Sixth International World Wide Web Conference Proceedings, 1997

[Betsie] Betsie Home Page:
http://www.bbc.co.uk/education/betsie/

[Bobby] Bobby Home Page:
http://www.cast.org/bobby/

[BrookesTalk] BrookesTalk Home Page:
http://www.brookes.ac.uk/schools/cms/research/speech/btalk.htm

[Chapman, 1988] E.K. Chapman, J.M. Stone, *The Visually Handicapped Child in your Classroom*, Cassell Education Limited, 1988

[CHI, 1996] E. Bergman, A.D.N. Edwards, *Universal Design: Everyone has Special Needs*, Computer-Human Interaction Conference Proceedings, 1996:
http://www.acm.org/sigchi/chi96/proceedings/panels/Bergmann/edb_txt.htm

[DDA, 1995] The Disability Discrimination Act 1995:
http://www.legislation.hmso.gov.uk/acts/acts1995/1995050.htm

[Disability Rights Commission, 2002] The Disability Rights Commission:
http://www.drc-gb.org/drc/RightsAndRequirements/Page132.asp

[Guardian, 2001] The Guardian Newspaper:
http://www.guardian.co.uk/internetnews/story/0,7369,521725,00.html

[iCan, 2000] Digital Divide and the New Economy:
http://www.ican.com/news/fullpage.cfm/articleid/63D65B3F-41AF-44A3-A4E02D616B924F41/article.cfm

[Kemenade, 2000] H. V. Kemenade, *Application of a methodology for the design of non-visual tables*, 3rd year project, Computer Science Department, University of York 2000

[Lynx] Lynx Viewer:
http://www.delorie.com/web/lynxview.html

[MkDoc] MkDoc Web Authoring Tool:
http://mkdoc.com/

[Nielsen] J. Nielsen, *Designing Web Usability*, New Riders Publishing, 2000

[Nua, 2001] Nua Internet Surveys:
http://www.nua.ie/surveys/how_many_online/index.html

[Perl2Exe] Perl2Exe Script Converting Tool:
http://www.indigostar.com/perl2exe.htm

[RNIB, 2001] The Royal National Institute For the Blind:
http://www.rnib.org.uk/

[Section 508, 1998] Section 508 of the Rehabilitation Act of 1973:
http://www.access-board.gov/sec508/guide/act.htm

[SENDRA, 2001] The Special Educational Needs and Disability Act 2001:
http://www.hmso.gov.uk/acts/acts2001/20010010.htm

[Sun] Sun MicroSystems:
http://java.sun.com/

[TOM] Text-Only Maker Web Authoring Tool:
http://archive.ncsa.uiuc.edu/Indices/Outreach/IntroducingTOM.html

[WAI, 1999] Web Content Accessibility Guidelines 1.0:
http://www.w3.org/TR/WAI-WEBCONTENT/

[WAI tools] Checklist of Checkpoints for Authoring Tool Accessibility Guidelines 1.0:
http://www.w3.org/TR/ATAG10/atag10-chktable.html

[W3C] Review of Lynx Viewer:
http://lists.w3.org/Archives/Public/w3c-wai-er-ig/2000Feb/0047.html

[W3C Validator] W3C Validator:
http://validator.w3.org/

8 Bibliography

The following are URLs and literature used by the author but not quoted:

T. Christiansen, R.L. Schwartz, *Learning Perl*, O'Reilly 1997

T. Christiansen, N. Torkington, *Perl Cookbook*, O'Reilly 1999

T. Christiansen, L. Wall, R.L. Schwartz, *Programming Perl*, O'Reilly 1996

A.D.N. Edwards, *Speech Synthesis – Technology for Disabled People*, Paul Chapman Publishing Ltd, 1991

G. Gardiner, E. Lowe, *Disability Legislation and Its Impact on Websites,* Buchanan Ingersoll Law Practice, 2001:
http://www.buchananingersoll.com/euro_law/articles/disabilityimpact.html

L. Harrison, J. Richards, J. Treviranus, *Authoring Tool Support: "The Best Place to Improve the Web"*, Adaptive Technology Research Centre, University of Toronto, 2001:
http://www.utoronto.ca/atrc/rd/library/papers/richar_j.html

R. Hayward, *WWW browsers for blind people*, 3rd year project, Computer Science Department, University of York 1997

P. Hoffman, *Perl for Dummies*, IDG Books, 1998

J.J. Lazzaro, *Adaptive Technologies for Learning & Work Environments*, American Library Association, 2001

S. McManus, *Tesco launches visionary website*, 2001:
http://www.sean.co.uk/a/webdesign/accessibility.shtm

M. Sloan, *Web Accessibility and the DDA*, Journal of Information, Law and Technology, 2001:
http://elj.warwick.ac.uk/jilt/01-2/sloan.html

S. Thompson, *A comparison of two approaches to Web access for blind users*, 3rd year project, Computer Science Department, University of York 2000

G. Weber, *Itd Technotes: Braille Displays*, University of Stuttgart, 1994:
http://www.rit.edu/~easi/itd/itdv01n3/weber.html

P. Woodcock, *Guidelines for ALT texts for auditory browsing*, MSc project, Computer Science Department, University of York 2001

The Comprehensive Perl Archive Network:
http://www.cpan.org

Freedom Scientific:
http://www.freedomscientific.com/

The Making Connections Unit:
http://www.mcu.org.uk/

The Royal National Institute for the Blind, *First RNIB Web Access Award goes to Tesco*, 2001:
http://www.rnib.org.uk/whatsnew/pressrel/may2001/tesco.htm

The World Wide Web Consortium:
http://www.w3.org

Appendix

Source code for the web authoring tools

9 Appendix

This chapter comprises the source code of the three web authoring tools.

9.1 Parallel directory structure tool

Program Name: mkdir.pl
Author: Omid Afzalalghom
Date: December 2001

```
#Code begins here.

#Define variable to hold system message.

$message = "The parallel directory structure has been created \n" ;

#Define variables to hold directory names.

$dir1="webfiles";
$dir2="textandimages";
$dir3="textonly";

# Moves to the root of the directory and creates the parallel directory structure.

chdir ("/");
mkdir ($dir1);
chdir ($dir1);
mkdir ($dir2);
chdir ("/");
chdir ($dir1);
mkdir ($dir3);
chdir ("/");

#Print message informing user of task completion.


&PrintMessage;

 sub PrintMessage {
 print $message ;

 exit 0 ;
 return 1 ;

 }
```

9.2 Text-only converting tool

Program Name: parse.pl
Author: Omid Afzalalghom
Date: December 2001

```
#Code begins here

print "Please type the name of the file that you would like to convert to text-only \n";

$file=<STDIN>;        #keyboard input is stored in $file variable.

$dir= "\\webfiles\\textandimages\\";
chdir ("/");
chdir ($dir);

#Opens standard web page for reading.
open (HTML, $file) or die "The file could not be found. \n" ;

$dir2= "\\webfiles\\textonly\\";
chdir ("/");
chdir ($dir2);

#Opens new text-only web page to write to.
open (OUTPUT, ">$file");

#Defines variables used in parsing routine.
$string2='';
$link='';
$copy=1;
$imgcopy=1;
$mapalt='';

while (<HTML>){                  #Reads original html file line by line.
  $string1=$_;
  $img = 0;

  $n= length $string1;

  for ($i=0; $i<=$n; $i++)        #Reads one character at a time from html file.
    {
        $letter = substr $string1,$i,1;

        $char4 = substr $string1,$i,4;
        $char4=~tr/a-z/A-Z/;
        $char4end = substr $string1,$i,5;
        $char4end=~tr/a-z/A-Z/;

        $char5 = substr $string1,$i,5;
```

```
$char5=~tr/a-z/A-Z/;
$char5end = substr $string1,$i,6;
$char5end=~tr/a-z/A-Z/;

$char6 = substr $string1,$i,6;
$char6=~tr/a-z/A-Z/;
$char6end = substr $string1,$i,7;
$char6end=~tr/a-z/A-Z/;

$char7 = substr $string1,$i,7;
$char7=~tr/a-z/A-Z/;
$char7end = substr $string1,$i,9;
$char7end=~tr/a-z/A-Z/;

$char9 = substr $string1,$i,9;
$char9=~tr/a-z/A-Z/;

$char10 = substr $string1,$i,10;
$char10=~tr/a-z/A-Z/;
$char11end = substr $string1,$i,11;
$char11end=~tr/a-z/A-Z/;

if ($copy == 4) {$copy=1;}
if ($copy == 5) {$copy=1;}
if ($copy == 6) {$copy=1;}
if ($copy == 7) {$copy=1;}
if ($copy == 10) {$copy=1;}
if ($copy == 11) {$copy=1;}

if ($alt == 2) {$alt=1;}
if ($img == 1) {$copy=1;}
if ($href == 1) {$href=2;}
if ($href == 3) {$href=4;}
if ($href == 5) {$href=6;}
if ($href == 7) {$href=8;}
if (($href == 2) and ($letter ne "")) {$href=1;}

if (($letter eq '>') and ($imgcopy==0)) {$img=1; $imgcopy=1;}
if ($char4 eq '<IMG') {$copy=0; $imgcopy=0;}
elsif ($char4 eq '<MAP') {$copy=0;}
elsif ($char4 eq 'ALT=') {$alt=2;}

if ($char5end eq '</MAP>') {$copy=5;}
if ($char5 eq 'WIDTH') {$alt=0;}
elsif ($char5 eq 'ALIGN') {$alt=0;}
elsif ($char5 eq 'ISMAP') {$alt=0;}
if ($char6 eq 'BORDER') {$alt=0;}
elsif ($char6 eq 'HEIGHT') {$alt=0;}
elsif ($char6 eq 'USEMAP') {$alt=0;}

if ($char5 eq '<AREA') {$copy=0; $area=1;}
```

```
        elsif (($char5 eq 'HREF=') and ($copy==0)) {$href=1;}
        elsif (($char5 eq '.HTML') and ($copy==0)) {$href=3;}
        if (($char4 eq '.HTM') and ($copy==0)) {$href=5;}
        if (($char4 eq '.COM') and ($copy==0)) {$href=7;}

        if ($letter eq '>') {$href=0;}
        if ($letter eq '>') {$alt=0;}

        if ($char7 eq '<SCRIPT') {$copy=0;}
        if ($char7end eq '</SCRIPT>') {$copy=7;}

        if ($char9 eq '<FRAMESET') {$copy=0;}
        if ($char11end eq '</FRAMESET>') {$copy=11;}

        if ($char10 eq '<NOSCRIPT>') {$copy=10;}
        if ($char11end eq '</NOSCRIPT>') {$copy=11;}

        if ($copy == 11) {$i = $i+12;}
        if ($copy == 10) {$i = $i+10;}
        if ($copy == 7) {$i = $i+8;}
        if ($copy == 6) {$i = $i+7;}
        if ($copy == 5) {$i = $i+6;}
        if ($copy == 4) {$i = $i+5;}

        if ($alt == 2) {$i = $i+3;}
        if ($copy == 1) {$string2=$string2.$letter;}
        if (($alt == 1) and ($imgcopy == 0)) {$string2=$string2.$letter;}
        if ($href == 5) {$i = $i+3};
        if ($href == 7) {$i = $i+3};
        if ($href == 3) {$i = $i+4};
        if ($href == 1) {$link=$link.$letter;}
        if ($href == 4) {$link= "<br>" . "<a ". $link . ".html" . ">" .
        "Image Map Link" . "</a>" ;}
        if ($href == 6) {$link= "<br>" . "<a ". $link . ".htm" . ">" .
        "Image Map Link" . "</a>" ;}
        if ($href == 8) {$link= "<br>" . "<a ". $link . ".com" . ">" .
       "Image Map Link" . "</a>" ;}

        if (($href == 4) and ($area==1)) {$string2=$string2.$link;}
        if (($href == 6) and ($area==1)) {$string2=$string2.$link;}
        if (($href == 8) and ($area==1)) {$string2=$string2.$link;}
        if ($href == 4) {$link=''; $href=0; $area=0;}
        if ($href == 6) {$link=''; $href=0; $area=0;}
        if ($href == 8) {$link=''; $href=0; $area=0;}
  }
}

print OUTPUT "\n $string2 \n";

close(HTML);
close(OUTPUT);
```

9.3 Maintenance tool

Program Name: demon.pl
Author: Omid Afzalalghom
Date: December 2001

```
#Code begins here

while(1) {
  sleep(1800);
  $dir1="\\webfiles\\textonly";
  $dir2="\\webfiles\\textandimages";

  chdir ("/");
  opendir(TEXTONLY, $dir1);
  @textnames=readdir(TEXTONLY);  #Reads text-only filenames into array.

  chdir("/");
  chdir($dir2);
  opendir(NORMAL, $dir2);              #Reads standard filenames into array.
  @imagenames=readdir(NORMAL);

  $i=0;
  $listpos=0;

  #While all files have not been checked, search for text-only equivalent.
  while($listpos<=$#imagenames){
      $file=@imagenames[$listpos];
        while($i<=$#textnames){
              $tfile=@textnames[$i];
               last if $file=~/^\.\.?$/;                #Ignore root and parent directories.
            #If filenames match then compare dates.
               if($file eq $tfile){
                   &date();$i=0; last}
               elsif(($file ne $tfile) and ($i==$#textnames)){
                    print "$file requires converting \n";
                    &convert();                        #Convert file if no equivalent exist.
                    if ($answer){
                      &parser($file); $i=0; last}
                    else{$i=0; last}
               }
               else{$i++}

        }
              $listpos++;
  }

}
```

```
sub date{

     chdir ("/");
     chdir ($dir2);

     @filedata=stat($file);
     $date=$filedata[9];

     chdir ("/");
     chdir ($dir1);

     @textfiledata=stat($tfile);
     $date2=$textfiledata[9];

     if ($date>$date2)
        {&update();}
}

sub update{
    print "$file has been modified.\n";
    print "Do you wish to update the text-only version? \n";
    print "Press 'y' if you wish to convert or 'n' if you do not \n";
    $reply=<STDIN>;
    $reply= substr $reply,0,1;
    if ($reply eq "y"){&parser($file);}
    else {return 0;}
}



sub convert{
    print "Press 'y' if you wish to convert or 'n' if you do not \n";
    $answer=<STDIN>;
    $answer= substr $answer,0,1;
    if ($answer eq "y"){$answer = 1;}
    else{$answer = 0;}
    return ($answer)
```