SADIE: STRUCTURAL-SEMANTICS FOR ACCESSIBILITY AND DEVICE INDEPENDENCE

A thesis submitted to the University of Manchester for the degree of Master of Science in the Faculty of Engineering and Physical Sciences

2005

By
Darren Richard Lunn
School of Computer Science
Contents

Abstract 6
Declaration 7
Copyright 8
Acknowledgements 9

1 Introduction 10

2 Background and Related Works 15
   2.1 Web Technologies (XHTML and CSS) 16
   2.2 Semantic Web Technologies (OWL) 19
   2.3 Accessibility 20
   2.4 Transcoding 24
   2.5 Annotation 28
   2.6 Semantic Transcoding 29

3 Design 32
   3.1 Capturing Semantics 32
   3.2 Improving Accessibility 36
      3.2.1 Removing Obstacles 36
      3.2.2 Triage 37
      3.2.3 Navigation 39
   3.3 Reasoning Over Web Elements 39
   3.4 SADiE Architecture 41

4 Implementation 45
   4.1 Development Platform (XUL and JavaScript) 45
List of Tables

3.1 SADIe Upper Ontology Properties ........................................ 40
4.1 SADIe Toolbar Keyboard Shortcuts ........................................ 53
5.1 SADIe Evaluation Results ................................................... 68
5.2 SADIe Web Site Evaluation Summary ..................................... 69
A.1 Z-Table For the Normal Distribution .................................... 87
List of Figures

1.1 HiTokyo Blog Entry ........................................... 11
2.1 HiTokyo Blog XHTML Sample ............................... 17
2.2 HiTokyo Blog CSS Sample ................................. 18
2.3 Intercepting Web Communication via a Proxy Server ........ 25
3.1 Ontological Structure of the HiTokyo Blog .............. 35
3.2 SADIE Upper Ontology Class Structure .................. 40
3.3 SADIE Architecture ......................................... 41
3.4 Blogger Ontology with Imported SADIE Upper Ontology ... 42
3.5 Inferred Blogger Ontology After Reasoning ............. 43
4.1 Sample XUL Code ........................................... 46
4.2 Sample JavaScript Code ..................................... 47
4.3 Sample SOAP Web Service Call ............................ 50
4.4 SADIE Firefox Toolbar .................................... 52
4.5 SADIE User Options Window ............................. 55
4.6 HiTokyo Blog After Defluffing ............................ 56
4.7 HiTokyo Blog After Reordering ........................... 57
4.8 The Mac Observer Before SADIE Transcoding .......... 58
4.9 The Mac Observer After SADIE Transcoding .......... 59
5.1 Standard Normal Distribution .............................. 64
Abstract

Currently the World Wide Web is visual-centric with web sites often being designed only with the presentation of data in mind. A consequence of this design perspective is that information contained within the data is only accessible implicitly through the layout of the web page, rather than explicitly through the data itself. While this implicit knowledge is relatively easy to access for sighted users, it is often difficult to access for visually impaired computer users.

This project describes an investigation into a way of allowing visually impaired computer users the same access to information on the World Wide Web as sighted computer users. By using ontologies to capture the semantics of the CSS Stylesheets and XHTML, the implicit information contained within a web page can be reordered and manipulated into an explicit form that better suits the needs of visually impaired users.
Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.
Copyright

Copyright in text of this thesis rests with the Author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the Author and lodged in the John Rylands University Library of Manchester. Details may be obtained from the Librarian. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the Author.

The ownership of any intellectual property rights which may be described in this thesis is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

Further information on the conditions under which disclosures and exploitation may take place is available from the head of School of Computer Science.
Acknowledgements

I would like to thank Dr Simon Harper and Sean Bechhofer for their support and guidance throughout the duration of the project.
Chapter 1

Introduction

Use of the World Wide Web (Web) has grown exponentially over the past two decades. From a simple text based hyperlink environment that easily and conveniently connected academics to resources and works from various research institutions, it has become a graphics based communications media [1] and an essential part of everyday life for most people. It is common for users of this evolved Web to shop, obtain news and weather information, find like-minded people, search for information, plus a whole host of other uses.

One of the popular growth areas of the web has been traditional media going on-line. Print media, such as newspapers and magazines, and live media, such as television and radio, have allowed access to on-line versions of themselves via the Web. In many cases this was a reproduction of the print version of the magazine or newspaper. The same colour and layout was used creating an electronic equivalent of what the customer has bought. These sites were the most popular sites on the web. As web sites designed specifically for Web use were developed, such as Amazon, Blogger, and Slashdot, they copied the same glossy, visual styles of their print based counterparts. This trend has meant that many web designers have moved from the print world to the web design world. At design time they have in their mind a vision of how their creation should look and feel. A static work of art that should not be modified or adapted.

While these glossy and gorgeous looking sites can be accessed by a large number of users, it isn’t suitable for all users. In particular, people with visual impairments have trouble accessing the information that is contained within the page.

Part of the problem is that the web designers, who are trained in graphics and
layout, only think of ways in which to present their information. This creates a visual-centric web page where information is only made available implicitly from how data looks and where it is located.

The sample web site in Figure 1.1 is taken from a blog\(^1\) on the Blogger web site\(^2\) and will be used as a running example throughout this dissertation. Looking at this web page it can be seen that there is a navigation menu on the right hand side of the screen. The user knows that it is a menu because it is contained in its own distinct block of text, separated from the rest of the content by a large amount of white space. The text within the block is a different colour and presented as a series of bullet points, indicating that they are referencing more in depth areas. However, there is nothing stating that this is a menu. The knowledge that it is a menu is implicitly tied to how it is presented on screen.

\(^{1}\)http://hitokyo.blogspot.com/
\(^{2}\)http://www.blogger.com/
For sighted users, this implicit information is accessible, but what about visually impaired users?

Many visually impaired computer users access information on screen through screen readers, such as JAWS [2]. These screen readers operate by reading text from left to right and from top to bottom. They only read text that is on screen, meaning that implicit information can be a hindrance in two ways. The first is that screen readers cannot know what information is being lost as a screen reader has no concept of visual layout. Text can be different colours, different fonts or highlighted in different ways, as in Figure 1.1, but a screen reader will not pick this up, just reading what is displayed on screen as if it was plain text. If sighted users gain knowledge from the layout of the data then this knowledge is lost when being read by a screen reader, making it inaccessible.

Secondly, layout usually creates blocks of text. As mentioned above, navigation menus are often at the side of the screen, with the main content next to that in a separate block of text. These blocks are only seen to be blocks because of their visual layout. As a screen reader has no concept of visual layout, the two blocks merge into one, creating meaningless phrases as the screen is read aloud to the user. Again looking at Figure 1.1, the screen reader would produce:

"Thursday, February 03, 2005 About Me So I had my class and what would you know my idea worked. I also Name:HiTokyo realised that because in the past I was dreading the class, I was also a Location:Tokyo, Japan...”

The sentence is meaningless and requires the presentation and layout to understand it and gain knowledge from the page, making the page inaccessible to those people who cannot see the layout and the presentation.

Even web sites with simple formatting can be difficult to access for visually impaired users. When a sighted user looks at a page, they immediately start to process the information that is on the screen. Using a form of parallel processing they can start breaking the page down into its constituent elements, such as a menu bar and a blog entry. Using this they can quickly draw their attention to the main area of information, scan its contents and ascertain if it contains what they are looking for. If the page does contain information that the user wants, then the page is read in full, as it was intended. If not, then a different site is looked at and the process repeats until the correct information is found.
CHAPTER 1. INTRODUCTION

This is not always possible for a visually impaired user. As already noted, a screen reader reads a page from left to right, top to bottom. There is no way in which it can determine where the area of important information is, so a user has to listen to most of the page before deciding if it is what they are looking for or not. If it is what they are looking for then the page has been read and no time has been lost. However, the typical case is that it takes several attempts at searching until we find what we are looking for, which means for a visually impaired user, having to read several web pages of information that they are not interested in before getting to the information that they really want.

One way to make these visual-centric web pages more accessible to visually impaired users is to capture the implicit information through encoding the meaning of the information. By capturing the semantics of the web page, the implicit information can be made explicit. This explication of the information can be achieved through annotation. By annotating what different parts of the page are, such as a menu or a blog entry, it is possible to develop tools that can use the annotation to manipulate and adapt the page. This can create pages that are better suited to visually impaired users, giving them access to all of the information that is on the page. For example, to aid the screen reader, images and banner advertisements can be removed and blocks of text can be organised so that instead of being aligned horizontally, they appear vertical to each other. It is also possible to annotate the data that the designer considers to be the most important information on the page. That way, relevant information can be put where it is more accessible to the screen reader, saving the user’s time and effort in finding the information that they want.

However, there are some issues when dealing with annotation and semantic mark-up of web pages. Research has shown that designers and web site authors will not adopt technology or techniques that increases their design time significantly. Creating additional annotation of XHTML is seen as an unnecessary overhead [3, 4] which prevents them from creating their beautifully designed web pages. If designers will not adopt techniques then no matter how beneficial they are, they will not be used and are therefore useless.

Web site design can therefore be seen to have three conflicting design goals. Firstly, web pages will always be designed to look appealing in an attempt to draw the attention of web users. As with traditional marketing, the customer’s attention needs to be instantly grabbed in order to get them closer to the product.
Only then is the actual content important because with their attention held, the customer is more likely to look and see what the product has to offer. Secondly, designers will always have tight deadlines and schedules to meet. They are always under pressure to get the website on-line and attracting visitors. Their time is valuable and cannot be taken up by techniques they see as unnecessary or costly. Thirdly, there is a need to make implicit information on web sites available to visually impaired users, through the use of annotation and the capturing of the semantics of the web page.

This project is an attempt to solve these three seemingly intractable criteria by creating an application that allows sighted users to see web sites in all their visually exciting glory, gives visually impaired users the ability to access all the information by making implicit information explicit and to have a low additional design overhead for web site creators.
Chapter 2

Background and Related Works

Currently the web is created with human readers in mind, rather than machines. HTML is predominantly used to create web pages, but the use of HTML combines the content and the style of the web page into one large chunk of code. While this is fine for the person who is viewing the page, it is hard for machines to separate what is style and what is data. This inability for machines to easily get access to the data means that it is difficult for tools to be created that can manipulate, query and modify the data to better suit the user and to automate the user’s tasks.

Take, for example, searching for information using one of the popular search engines such as Google.¹ The user enters a keyword that is related to the information they are interested in finding. Google then searches the web for instances of the keyword, often returning thousands of matches, most of which have no relevance to the user’s original intention. It is then up to the user to either look at all these “hits” until they find the information they are looking for, or refine their search by modifying or adding new keywords.

While SADiE is not directly linked to searching for information, the principal difficulties that the SADiE project tries to solve are the same. Google cannot retrieve the information that a user wants because that information is implicitly bound to the presentation of the document, even if the document has good separation of presentation from information. Similarly, visually impaired users cannot get access to the information they want because it is implicitly bound to the presentation of the document. SADiE attempts to aid visually impaired users by using existing Web and Semantic Web technologies to capture the semantics

¹http://www.google.com/
of the document; then, by using well-established research into accessibility and transcoding techniques, SADIe transforms the page to make the implicit information explicit and allow visually impaired users easier access to the information contained within the document.

2.1 Web Technologies (XHTML and CSS)

Tim Berners-Lee invented the World Wide Web in the 1980’s, with his original vision being that the meaning of information would play an important role on the Web, in order for machines to process both information and data automatically [5]. This vision has become known as the Semantic Web, where automated agents can perform tasks such as information retrieval and search much more effectively than current tools using current web technologies. The Semantic Web is not an alternative to the current Web, but an evolution from the Web as it stands today into the Semantic Web that Tim Berners-Lee envisioned 20 years ago.

For the Semantic Web to operate effectively, machines will need to be able to access data from, and communicate with, hundreds, if not thousands, of different resources. XML\(^2\) is a text based data format that allows different organisations to be able to share information and data by using the same language. XML not only allows the user to structure the data in the document, but it also allows them to say what kind of data it is. This, combined with the fact that the same language is being used universally, gives the user, or the machine, the ability to search and query multiple resources at multiple locations to find and extract more accurate information.

Most web browsers used to access web pages have difficulty displaying XML documents. XHTML\(^3\) is a language that was developed to act as a bridge between XML and HTML. Documents created using XHTML are XML compliant, giving people who use it the power of XML. However, by following guidelines, set out by the W3C, XHTML documents are also compliant with the HTML 4 standard [6]. This allows resources written in XHTML to be accessible by the large number of users who still use traditional HTML based web browsers for accessing web documents.

Figure 2.1 shows some sample XHTML code from the HiTokyo blog. It can

\(^2\)Extensible Markup Language
\(^3\)Extensible HyperText Markup Language
CHAPTER 2. BACKGROUND AND RELATED WORKS

be seen that the code looks similar to that of standard HTML, with the exception that it is possible to give tags an ID or Class value. Unlike formatting attributes, which can occur in standard HTML, these ID or Class values add metadata to the page and help capture the semantics of the document. For example, if we look at the blog entry in Figure 2.1, we can see that the page contains a date-header and a post, and a post contains a post-body and a post-footer. These class elements allow us to build up a semantic representation of the document and its structure.

As noted previously, the problem with HTML is that it combines the presentation and style of a document with its content. This merger of style and content makes it difficult for machines to know which part of a document describes the style and which part of a document is the actual content. CSS⁴ is a way of separating the style and presentation of a document from its content. The CSS file describes how the web browser should display the data found within a web document and defines such things as the font type, the size and colour of text, the positioning and size of images, margins, borders and a large number of other properties [7].

Figure 2.2 shows some sample CSS code from the HiTokyo blog. As the code shows, we can define how certain elements should look. The code in Figure 2.2 shows that all elements that are classified as sidebars should float on the right

---

⁴Cascading Style Sheets
Figure 2.2: HiTokyo Blog CSS Sample

hand side of the page and have a width of 226 pixels. Not only can CSS be used for defining your own styles, but it can also be used for overriding existing styles. For example, web browsers tend to display the body of a document with a white background. CSS allows this to be overridden, so that the body could be green, or have a tiled image. A tag such as <h1>, which indicates a header, can also be overridden from the default large, black font, to have different colours and sizes and even a different font type.

It is when XHTML and CSS are combined together that the power of the two languages becomes apparent. Looking at Figure 2.1, it can be seen that there is no presentation code contained within the document. It only contains the data and the structure of the data. Figure 2.2 has no content. It is code purely for the use of defining styles, either by overriding the styles of existing tags such as <h1> or by defining new styles for new classes such as footer. By combining the content data of the XHTML document and the formatting data of the CSS document together, Figure 1.1 is produced. A web page that, for the human reader, looks visually stunning, but behind the scenes has a distinct separation of style and content. This separation allows automated tools to have easier access to the actual data of the page, without the hindrance of the style and presentation code affecting the processing and manipulation of the data.
CHAPTER 2. BACKGROUND AND RELATED WORKS

2.2 Semantic Web Technologies (OWL)

For machines to be able to access data and information contained within web documents, there needs to be some way for the information and knowledge within the document to be captured. Ontologies are one such way of doing this. An ontology is a description of concepts and the relationships that exist between them [8]. For example, Dog is-a-kind-of Animal shows the relationship between the two concepts of Dog and Animal. Ontologies are often used in the fields of artificial intelligence and knowledge representation because they allow the ability to reason over the concepts and relationships to see if they are logically sound [9] and to discover new relationships and new knowledge from the concepts and relationships that have already been established.

The Semantic Web will require that resources be linked together and related, in a similar fashion to the concepts stored in an ontology. This ontological structure will allow the meaning of the resources to be explicitly defined and therefore more accessible to machines, rather than implicitly defined and accessible only to the human reader [10]. By becoming accessible to machines, automated agents will be able to use the resources on the Semantic Web to discover information and knowledge in a more accurate and precise way than is currently possible. For this reason the W3C has created OWL\(^5\).

OWL is an ontology language for the Web. Its origins derive from two projects to create an ontology language. These were the European OIL project [11] and the United States DAML-ONT project [12]. The work on these two projects was combined and led to the development of DAML+OIL. DAML+OIL was then used by the W3C as the basis for OWL. OWL allows users to write explicit, formal conceptualisations of domain models by using a well defined syntax, formal semantics and allows reasoning support [5]. These combined properties allow the user to check for information such as class membership, consistency of the classes within the ontology and also allows classification of new classes so that they may find new knowledge.

OWL comes in three flavours, or species. The reason for this is that different users require different aspects from an ontology language. Each species of OWL is a subset of the previous. The three species are [5] [13]:

\(^5\)Web Ontology Language
1. **OWL Lite**: This is the smallest OWL language. It supports simple classification hierarchies and simple constraints. It does not support enumerated types or disjoint classes and can be reasoned over using a reasoning tool such as Racer [14] or FACT [15].

2. **OWL DL**: This subset of OWL supports all the features of OWL Lite but allows all OWL constructs to be used. However, there are restrictions on how the classes can be used so that the ontology can be reasoned over, using well-established Description Logic rules, which is what the DL stands for.

3. **OWL Full**: This is the entire OWL language. It has all of the OWL DL features and also includes cardinality restrictions and the ability to treat classes as a collection of individuals. Because of this, it cannot be guaranteed to be decidable in finite time when it is reasoned over.

OWL allows us to describe the data and resources that are on the web and, as mentioned, some species of OWL provide mechanisms for these descriptions to be reasoned over. By reasoning over OWL ontologies we can infer additional knowledge about the resource, such as discovering new classifications of concepts within the resource based on logical assertions. Reasoning can also discover if a resource is logically flawed, as the reasoner can discover contradictions within the model of the data. The contradictions may be due to human error in creating the model or it may be because the model itself is fundamentally flawed. Either way, the reasoner can bring this to our attention and corrections and redesigns carried out as is necessary. Once the OWL model has been reasoned over, tools designed to manipulate and communicate with the resource can use any discovered knowledge to access and manipulate the data held within the web resource automatically, without the need for human intervention.

### 2.3 Accessibility

We are living in a world that revolves around access to information. In a short period of time we have gone from having limited access to information to free and total access to information. Until recently, if a person wanted to read the news they bought a newspaper, available only in the morning, or watched a news bulletin, broadcast at a set time on a limited number of television channels and radio stations that were available. Now, we have access to news content from
around the world, whenever we want it, via the web. There are multiple 24-hour news channels on satellite and cable television. Not only is information available when we want it, but the choice of how we consume it is huge. Streaming audio, video or plain text in multiple languages is readily available.

The Information Society is a revolution, much like the Industrial Revolution of the 18th and 19th centuries. The Industrial Revolution created a distinct class system consisting of general workers, who were poor, professionals, who had a comfortable income and lifestyle, followed by industrialists, who controlled a majority of the wealth. The Information Society we are creating is in danger of establishing its own class system, where there are those people with access to information and those people without [16].

The Graphical User Interface has become the universal way of interacting with computers and gaining access to the vast repository of information that can be found on the Web. While this method of access is good for able bodied people, it is more difficult for people who have a disability [17]. Disabled users are often overlooked when designing these user interfaces, more often than not because designers are not aware of the needs of the disabled user[18]. It is this lack of awareness that is helping to create the two-class system that exists within the Information Society.

Accessibility is the removing of barriers that prevent people with disabilities from participating in life activities. Disabilities can be permanent or temporary. A user who breaks an arm will be unable to use a computer effectively for a few months while the arm heals. However, during that period of healing, they suffer the same difficulties of access as a user who may be missing a limb [18].

Accessibility covers a broad range of issues. There is no such thing as general accessibility, as needs vary from person to person, whether it be a person with total blindness or a person who has a slight hearing difficulty. There is a large range of accessibility measures that software developers can incorporate into the design of their systems. At one end of the scale, it could be something as easy as adding text descriptions to images on web pages using the alt tag [19]. At the other end of the scale, it could be something as complicated as designing applications that can be easily adapted to suit the needs of the individual user, regardless of their disability or ability.

Unfortunately accessibility is not often incorporated into software designs. The most common argument is based around costs, either in terms of economic
costs or costs in time [18]. Designers argue that they spend too much time and money creating accessible designs but don’t get much in return [20]. However, people with disabilities have the same right to access information technologies as other people. By not addressing the issue of accessibility, we are creating a large number of second-class citizens in the Information Society. A recent survey by the Disability Rights Commission reported that 6,860,000 adults in the UK are considered disabled [21] with 1,066,740 people having a visual impairment [22]. By making software more accessible, designers are increasing the potential market for their product by millions of people, and therefore increasing potential revenues, which counteracts any potential increase in design costs.

We live in an Information Society and the Web is the world’s largest information repository, which is why a large proportion of accessibility research is focused on accessing web pages. The W3C has a working group to look at the issues that people with disabilities may face while using the Web. While the guidelines that the group produces try to cover as many accessibility issues for as many disabilities as possible, a large proportion of the guidelines are related to people with visual impairments [23]. The reason for this is because it is visually impaired users who have the most difficulty in accessing the Web due to its visual-centric nature.

Accessibility techniques applied to web pages can vary. A common approach is to add descriptions to the visual elements on the web page. The ALTifier project [24] asserted that there were three categories of diagram:

1. Illustrations that carry information and explain things on the web page.

2. Navigation aids such as buttons, which provide links to other places.

3. Presentation and decoration images whose only purpose is to make the page look nice.

The results of the The ALTifier [24] project was a tool that crawled a web site at design time. The designer was then presented with a list of images and their ALT description. If a description was missing then the ALTifier tool used a combination of heuristics and pattern matching to determine the kind of image it was and provide an appropriate alternative description.

Alt-Server [19] took the research of ALTifier one stage further. Rather than only adding alternative descriptions at design time, ALT-Server analysed the
images of a web page that was being viewed by the user at that time. If there
was no alternative description for an image, then the URL of the image was
sent to a server. The server contained descriptions of thousands of images. If a
match could be found then the description would be sent to the web browser to
display as the ALT for the image. If no description was found one was generated
automatically but marked as “to be described” in the image database ready for
annotation at a further date.

A main feature of the Web is its ability to link together documents and re-
sources through hyperlinks. These hyperlinks allow the user to travel around
the web, visiting one web page followed by another. This allows them to follow
trails of thought, read related items of information and expand the breadth of
information that they were initially seeking.

Due to its visual nature, sighted users can gain a lot of information quickly
by looking at the layout and presentation of a document. They can see what
the page looks like and scan through the document to see if the page is relevant
to what they want to read about. These visual cues are beneficial to travelling
around the Web. A user may click on a link and get taken to somewhere that
they don’t want to be. By quickly scanning the information, they can ascertain if
the page is relevant and either read it or go back to where they came from. Due
to the graph structure of the Web, cycles of links frequently occur. A sighted user
can see if they have visited a page before ether by looking at the layout and style
of the page or by looking at the colour of the hyperlink, which generally change
colour once they have been selected and are cached in memory.

A visually impaired user, however, can find difficulties when travelling around
the Web. The visual aids that a sighted user makes use of when travelling are
not available to a visually impaired user. They cannot see colour hyperlinks or
page layout, so the user can only recognise a visited page once they have read
most of the content through their screen reader. As a screen reader only reads
from left to right, top to bottom in a serial manner, discovering that a page
has been visited in the past, or indeed if a page is relevant to the information
that the user wants, can take some time. To help solve the problem of travel
accessibility, several techniques have been proposed, most of them involving the
notion of transcoding.
2.4 Transcoding

The traditional view of how people access the Web is of a person sat at a desk using a desktop PC [25]. The PC will have a large, multi-coloured display, typically using a screen size in the region of 1024x 768 pixels and having a 32-bit colour resolution. It is this stereotypical image that most web designers use as a starting point when they begin to create the layout and presentation of their web site.

While this access scenario is true for a vast majority of computer users, it is not the only way in which people have access to the Web. We are living in a world where technology is increasingly more mobile and miniature. As an example, take the mobile phone. Not so long ago, a mobile phone could make a call, send a text message and, if your phone was top of the range, it could play games. Now, mobile phones can take photographs, record video, play MP3 music, transfer files using blue tooth technology and access the Web. While the technology has advanced rapidly, the screen size of the phone as remained static, with a typical size being 128 x 128 pixels. Not only are mobile phones more advanced, but we also have more computing technology in our pockets. Personal Digital Assistants (PDA), Smartphones and small laptop computers can now access the Web from any location due to the advent of WiFi. So while communication is easier now than it ever has been, access to information still remains difficult, because the resources that are on the web are not designed for the small screen devices that more and more people are using to access it.

The difficulty in accessing web pages isn’t limited to just small display devices. People with disabilities have access needs other than a large coloured monitor. Severely visually impaired users generally access web resources via screen readers, creating what is in effect a zero-display device. People with motor control disabilities may require larger links in order for them to move the mouse and click the link. Making the links larger means that we need to adapt the display of the web page to be larger than that which was originally intended.

Transcoding is a way of adapting Web content so that it can be viewed on any of the increasing number of diverse devices [26]. Transcoding reformat the content of the web page so that it is better suited for the target device. Because the same page is being adapted, the designer doesn’t have to create multiple pages, with each page targeted at a different device. This benefits the web design team in several ways. Because only one page is required, time spent on constructing
CHAPTER 2. BACKGROUND AND RELATED WORKS

The site is saved. Having only one page eliminates the need for multiple pages, which would create problems in terms of consistency, as trying to manage the content of several documents is difficult. Finally, the target audience of the page is increased, as more users, each with different devices, can access the page.

Figure 2.3: Intercepting Web Communication via a Proxy Server

The most common way of achieving transcoding on a device is via a proxy server. Figure 2.3 shows how the proxy pipeline operates. The web designer designs a web page as per usual and then makes their resource available on the Web. The user accesses the web resource as they would normally access a web resource; typing the URL into a web browser so that they can view the content. Instead of accessing the web page directly, a proxy intercepts the communication between the user’s browser and the web page and adapts the page content before it is rendered on the user’s device.

Several attempts have been made to create transcoding systems with varying degrees of success. One attempt at a PDA transcoding system [27] tried to maintain the original design vision of the web page creator by applying several image transformation techniques to the page as a whole. This included zooming, a fish eye view and different panning techniques.

A second approach is to analyse the web page and use heuristics to look for blocks of content.[28]. These blocks are then separated to form smaller individual pages of content, which are then accessed via a series of links, creating a tree structure.

A third approach, which can be applied to most devices, is to apply rules to the web page [29]. These rules can be as simple as ‘remove all images’ or more complex such as ‘if there is a table and the outer column has a larger proportion of anchor tags than paragraph tags, then it is a menu so move it to another location on screen’.

The problem of travel on web pages has been looked at in several research projects. Each one has offered a different solution, but transcoding techniques
have played a large part in making the web pages more accessible for the user.

WAB\textsuperscript{6} proposed that improving access to links was the key to improving navigation of web pages \cite{30}. The system operated as a proxy server (Figure 2.3), with web pages being analysed for anchor tags as they were accessed by the user. All the links that were found were marked with the word “link” to make them more obvious to the screen reader. A list of links found in the document was also generated and placed at the bottom of the page, so that the user could have easy access to the links, which would aid the navigation of the document.

Richard Chimera and Ben Schneiderman addressed the problems of scanning a document for relevant information. They proposed generating a table of contents for web documents\cite{31} and asserted that there were three kinds of possible layout:

2. Expand/Contract, where an item is clicked and then expanded to display submenus.
3. Multipane, where there are separate panes, each containing different parts of the hierarchy.

The conclusion they reached was that stable tables of contents did not aid the user, as they could not get a feel for the divisions of the text within the document. However, Expand/Contract and Multipane gave the user some feeling of context. They could use trails to give themselves cues as to where they had been in the document and could go back to known states if they became lost or found that they were reading parts of the text that they were not interested in.

TOWEL\cite{32} was a research project that compared the notion of travelling in the real world to travelling in a virtual environment. The research concluded that visually impaired users travel in an egocentric manner, always placing the traveller at the centre of the orientation and not the environment \cite{23}. The visually impaired traveller also tried to look ahead for obstacles by tapping a stick to get auditory feedback of the environment. This method of travelling around the real world could be applied to the virtual world. The act of using a stick to probe the environment was similar to probing hyperlinks to see what the content was at the end. A stick could give feedback to obstacles or a clear path in

\textsuperscript{6}World Wide Web Access for Blind and visually impaired users
the environment. Following a hyperlink either took you to irrelevant information or relevant information.

The TOWEL research, led to the creation of a prototype tool that used annotation of web pages to aid navigation through the web [33]. The designer of the web site added additional annotations to the XHTML of the web page in the form of Class or ID attributes as part of the `<span>` and `<div>` tags. This annotation marked elements of the web page as things such as cue contexts or obs contexts. Cue contexts were items that drew the user’s attention to relevant pieces of information on the page. Obs contexts were obstacles that could cause problems to the traveller. By annotating the web page, TOWEL could use these additional semantics to help navigate the user through the components and structure of the web page including its links to other web pages and resources.

DANTE\(^7\) also attempted to address the problem of web page navigation and orientation [35, 36]. The project was based on the idea that due to poor design, the failure to follow accessibility standards and the nature of how screen readers operate, implicit knowledge contained in web pages is inaccessible to visually impaired users. This inaccessibility makes navigation and travel through web pages and their links difficult for the user. The user cannot immediately access the main content of the page, often meeting obstacles and irrelevant material as they try to find the information that they are looking for. Links are often non-descriptive, making it difficult to know whether a link is part of the navigation of the page or if it is a link to a related but complete different web site. This can create orientation problems as the user soon discovers that they don’t know where they are in the web.

The result of the DANTE research was a semi-automatic tool that analysed the DOM\(^8\) of web pages to help with the navigation of the page [37, 38]. First, it looked for travel objects on the page, such as a header, paragraph, etc. When this was done, the objects were classified into roles by using a travel ontology. The roles were elements that are used when a user is travelling through the page. By using annotation, to mark the roles of elements and components of a web page, the page could be described in a way that helped the screen reader to navigate through the content of the page. For example, marking what was information and what was a navigation menu, allowed the screen reader, and therefore the

\(^7\)Named after travelling through hell in Dante’s Inferno [34]
\(^8\)Document Object Model
user, the ability to get to the content more effectively and allowed the user to know what type of content it was.

2.5 Annotation

An Annotation is a remark attached to a portion of a document [39]. These remarks describe the elements within a page, so that it is possible for tools to manipulate the page to meet the end user’s needs. For example, groups of links may be annotated as a navigation menu, images may be annotated as advertisements or paragraphs annotated as the main content of the page. By analysing the annotations, the tool can adapt the page, removing elements that are deemed unnecessary, reordering components to enable better access or breaking a page up into smaller pages so that they better suit small display devices.

Chieko Asakawa classified annotations into two groups [40]. Structural Annotations are annotations that are used to recognise visually fragmented groupings as well as the basic role and importance of each group. Commentary Annotations are used to get a useful description of each group. She then went on to argue that there were two problems with the use of annotations to transcode web pages [41]. The first was that the workload required to annotate web pages was too high. The second problem was that the Web contains many dynamic pages; so applying annotations to them would be difficult.

Asakawa-san’s solution to this problem was an annotation database. The database would contain annotations of web pages based on typical structures that occur on the web. When a page is accessed, its DOM is analysed and annotation pattern matching is used to match the page to an annotation style in the database. The benefit of this method is that the workload in producing annotations is reduced, as there is a central repository of patterns that can be used. Secondly, annotations can be applied to dynamic pages, as it is the structure of the page that determines the annotations, rather than the content.

Annotation can be associated with a document in two ways [39]. Inline annotation allows the annotation to exist inside the document, which is what was used with TOWEL [33]. The annotations sit next to or close by the elements that they are describing. External annotations reside inside a file separate from that of the document that they are describing. These kinds of annotations were used in the work carried out by Katashi Nagao [42]. They use links to point from
the annotation to the element that the annotation is describing. A benefit of this method is that document ownership is not necessary in order to annotate a file. Users can make an annotation that best suits their needs and will produce a more accessible version of the web page when Semantic Transcoding is applied.

2.6 Semantic Transcoding

The Semantic Web separates content and presentation creating two components of a web page: actual content and content display. Even with this separation, it is still possible that knowledge contained within the page is only accessible implicitly via the way in which the page is rendered [43]. To get around this problem, implicit knowledge should be made explicit through capturing the semantics of the page.

As stated above, transcoding is the adaptation of a web page to suit a particular target device on which the page will be displayed. Semantic Transcoding is the adaptation of a web page to suit a particular target device by using the semantics of the page. Whereas transcoding will make educated guesses as to how the page should be adapted, for example by analysing link structure or applying combinations of rules, semantic transcoding will use the metadata that describes the structure of the page in order to make adaptation decisions.

Many semantic transcoding systems use annotation of web pages to capture the structure of the page and the knowledge held within it [42]. Due to annotations providing metadata that better describes the semantics and structure of the web page, semantic transcoding can be more accurate than transcoding. Transcoding, using analysis of the page or the application of rules, can only produce a modified page that is based on the most likely case. For example, the rule that if a column has a large proportion of anchors then it is a navigation menu, is applicable to most web pages, but not all of them. Rather than using probability to deduce what a page should look like, semantic transcoding can accurately create a modified page. If a user does not want banners, then their tool can look for the annotations and metadata that describe a banner and then remove it, rather than guess what a banner element might be. The semantics accurately describe the document, which in turn allows the tools to accurately transcode the page to more accurately suit the users needs.
COHSE\textsuperscript{9} [44][45] was a research project that attempted to improve the quality of the links between documents on the web by exploiting the metadata of the document and creating dynamic links at retrieval. COHSE asserted that most links on the Web are put in place by the authors of the web page. This static link structure is insufficient due to the fact that the web is dynamic, with web pages constantly being created, adapted and destroyed. Due to this dynamic nature, link navigation is often not enough for the user and has to be supported by search, which itself suffers the problems of inaccurancy. The project concluded that links did not have to be embedded within web documents but could be managed and maintained separately. By doing this, it would be possible for the links to be searched and transported in order to make better use of them.

The solution COHSE proposed involved several technologies combining to improve the linking of documents and navigation on the Web. A web document was annotated with metadata so that the data within the document can be categorised more effectively, using a domain specific set of keywords. Separately, ontologies were created. The ontologies used as part of the project were in the form of a thesaurus, so that documents could be categorised under specific keywords, based on their metadata, creating clusters of documents with similar content [46]. For example, a document about clothes would be related to the concept of clothes in an ontology.

To generate relevant links for a document, the user selects a term they are interested in. The link generator contacts the ontology service and gets a list of related concepts for that term. A preferred concept is then selected, which is mapped to the term. This mapping is then used to obtain documents that are relevant to the term that the user was interested in. The mapping is possible because the metadata of the document has been used to decide the document’s category, and it is the category that can be associated with a term. Because more than one document can be relevant, several link destinations can be generated for the document, giving the user a greater choice in the number of target documents that are available. Due to the use of categorisations in the ontology, all the linked documents should be relevant to the user as they are associated to the same term and concept. If there are too many links generated then the ontology service can be called again, asking for a narrowing of the related concepts that it has discovered. Conversely, if the quality of the links is not high enough, then the

\textsuperscript{9}Conceptual Open Hypermedia Services Environment
scope of related terms could be increased, making the system more flexible as it tries to meet the user’s linking needs.

Search for documents can be improved through combining the use of metadata and categorisation and the use of ontologies to relate user terms and concepts. The web page can then be transcoded so that the links are added dynamically and when the user wants them. Because the links are created based on the semantics of the documents, the web experience of the user is more productive in terms of the time saved in finding relevant information and the quality of the information that is found.

XHTML and CSS are technologies that, when combined, can create good-looking web pages that users can enjoy looking at. However, they maintain a separation of the content of a web page from its presentational style. This separation is what allows us to get direct access to the data and semantics of the document. OWL is a web ontology language that allows designers to describe the data and resources that are on the web and provides mechanisms for these descriptions to be reasoned over by using reasoners such as Racer.

Accessibility is the removing of barriers that prevent people with disabilities from participating in life activities. To make web pages accessible, transcoding and semantic transcoding are often used to adapt and modify a web page to better suit the user’s needs. To perform transcoding, access to the semantics of a document is often required.

SADIe will draw upon all these technologies and techniques to create a tool that will dynamically triage a document in order to make it more accessible to visually impaired users. At the same time, it will keep the additional design time of the web page to a minimum, so as to get full support from web site designers.
Chapter 3

Design

Web site design can be seen to have three conflicting goals. Firstly, web pages will always be designed to be beautiful in an attempt to draw the attention of sighted web users. Secondly, designers will not adopt techniques they see as unnecessary or costly. Thirdly, there is a need to make implicit information on web sites available to visually impaired users, so that everybody has the same access to the information.

SADIe attempts to solve these three conflicting criteria by creating an application that allows sighted users to see web sites as they were intended but gives visually impaired users the ability to access all the information by making implicit information explicit. This addresses the designers’ concern that the web page they design is seen as it was originally intended, but it also allows their page to be accessed by a greater audience. SADIe achieves this by capturing the semantics of a web page from the XHTML and CSS to produce an ontology. As designers already use XHTML and CSS technologies to create web pages, the additional design overhead is kept to a minimum. All that is required is an extra ontology file, meaning that SADIe has a higher chance of being accepted by the web design community.

3.1 Capturing Semantics

There has been a substantial amount of research into accessing web content. Access may be required for different devices, such as mobile phones, or it may be to allow different groups of users access to web pages, for example visually impaired users. While lots of research has been conducted, a general consensus
is that content adaptation is crucial for transparent web access under different conditions and client capabilities. To achieve transparent web access we need to annotate documents so as to give hints to the transcoding system as to how to alter the page [24].

Many people have proposed different ways of using annotations and different ways of classifying them. Chieko Asakawa classified annotations into two groups [40]. Structural Annotations and Commentary Annotations. Mari Abe and Masahiro Hori suggested that the distinguishing feature of annotations is that they are either inline or external [39]. There have even been suggestions that annotations are either simple, for example specifying an element’s importance, or complex, such as specifying different images for different devices [47].

However annotations are categorised, they all have similar problems, the main problem being the time it takes to annotate a document. Annotation of one page can take a long time. If there are multiple pages on a web site, it can be tedious for the web page author to do [41]. Solutions have been proposed for this problem, such as CREAM\(^1\) [48], which is an annotation framework. There have also been proposals for the use of WYSIWYG\(^2\) Editors [49] or tools which can automate the annotation process [50].

While the workload of annotation can be reduced, there are still problems with where to put the annotations once they are created. If inline annotations are used, then the size of the file will increase drastically, causing problems for devices that have a low bandwidth web connection. Adding annotations to HTML files also breaks the HTML syntax and web browsers will not be able to render the page properly [47]. As most web content is accessed via web browsers, designers will be unwilling to adopt a technology that will make their sites inaccessible by most of its users. Annotation can also be considered as syntax for displaying content. By putting it inside the HTML document, the principle of separation of content and presentation is violated [51]. External annotations can solve these problems, but creates issues with the maintenance of links between the HTML and the annotation. If a page changes, then the annotation file must be updated too, in order to keep them synchronised [51].

SADIe proposes that additional annotation is not necessary to capture the semantics of a web page in order to transcode it to better suit the needs of a

---

\(^1\)CREAting Metadata for the semantic web

\(^2\)What you See Is What You Get
visually impaired user. For this assertion to be valid, a web designer must have
designed the web site with valid XHTML and CSS, following current guidelines
and standards, such as not using tables for layout and not mixing presentation
with the content.

Looking at Figure 2.1, we can see that the tags contain attributes, such as ID
or Class. In web design these attributes are primarily used in conjunction with
the CSS to determine how to display the elements of the page. The Class and
ID values that occur in the XHTML file will have an equivalent style definition
in the CSS file. While designers use these ID and Class values for determining
the layout of the page, they are also inadvertently capturing the semantics of
the web page. For example, if we look at the blog entry in Figure 2.1, we can
see that the page contains a post and that a post contains a post-body and a
post-footer. These class attributes allow us to build up a semantic representation
of the document and its structure. We do not need to add additional annotation
to know what a post object is, or what a sidebar object is. The values that
determine how objects are displayed can also tell us what the objects are.

Figure 2.1 is a small snapshot of the HiTokyo Blog XHTML. If we were to
go through the entire CSS file and the XHTML file looking for Class and ID
attributes, we would be able to build up the ontological structure that is shown
in Figure 3.1.

This ontology represents the document structure of the HiTokyo Blog. We
can identify objects by their Class and ID attributes, such as post objects, which
contain the text of the blog entry, or sidebar objects, which contain navigation
links. However, these semantics were not added by additional annotation of the
page, but by the designers themselves when they created the page.

Web sites generally consist of many web pages. The Blogger web site has
close to 1,000,000 blogs on it [52], which creates in excess of 1,000,000 web pages,
due to some blogs spanning several pages. When designers create web sites, they
usually create a consistent look and feel. All the pages, for example, will have a
navigation menu in the same location, and the same style of header. Similar fonts
and colours will be used across all the pages of the site. An additional benefit of
using the XHTML Class and ID attributes to capture semantics is that we can
exploit this consistency. If all the pages of the site use the same CSS styles and
the same structure to create the same look and feel, then the elements within the
page will have the same Class and ID attributes. A consequence of this is that
CHAPTER 3. DESIGN

Figure 3.1: Ontological Structure of the HiTokyo Blog
by analysing one page of the site we can build an ontology that can be applied to every page on the site, provided that it maintains the same look and feel. This eliminates the need for time-consuming annotations on every page, creates only a small amount of additional overhead for the designer, yet allows us to capture the semantics of a substantial number of web pages, whether they be legacy pages or new pages.

With this semantic representation of the page captured, SADIe can now use the structure and information to transcode the web pages of the site in order to improve their accessibility to visually impaired users.

### 3.2 Improving Accessibility

Accessibility is the removing of barriers that prevent people with disabilities from participating in life activities [18]. SADIe will attempt to make web pages more accessible to visually impaired users by removing obstacles so that screen readers can read the text without having the flow broken, by improving the time it takes for a user to determine if the page contains information they want to read and by providing easier access to navigation menus.

#### 3.2.1 Removing Obstacles

A big hindrance to visually impaired users when navigating a web site is that they meet obstacles that prevent them from accessing the information that they want. These obstacles can take many forms, including banner advertisements, images to improve the aesthetics of the page or frames.

There have been many proposals for how to deal with these obstacles. As a large proportion of obstacles for visually impaired users are images, ALT-Server [19] suggested adding alternative descriptions to images, so that a description could be read out to the user. While this can be beneficial to some images, it is our view that it would not help with all the images that appear on a web page. As noted previously, images on a web page can be classified as illustrations, navigation aids or decoration images [24]. While an alternative description would be of use for an illustration that helps explain things, descriptions for decorations would be a hindrance. As the Web becomes more visual in nature, the number of decorations increases. A sighted user will ignore these decorations as they read the content of the page, but if a screen reader read a description of a decoration
every time one occurred, then accessing the information would be difficult as the
flow of the text would be constantly broken up with an irrelevant description of
some decoration element.

Silas Brown developed the Web Access Gateway system[53]. His system mod-
ified web pages as the user accessed them by removing objects that he considered
to be inaccessible to users. His list of objects to remove included JavaScript,
embedded media, background images and blinking text, to name but a few.

SADIe proposes that objects that act as obstacles on a web page should be
removed. However, because SADIe uses Semantic Transcoding to modify the
page, blanket rules, such as those used by Silas Brown, will not be applied. For
example, rather than saying that all images should be removed, SADIe will only
remove images that are marked as removable in the ontology for the web page.
This allows SADIe to be more selective. For example, many web pages have a
footer that notes the time the page was last modified. This footer adds little
value to the information of the page and so can be considered as a candidate for
removal. Some footers, however, do carry important information. As transcoding
only takes place based on the semantics of individual web sites, if there is a need
to keep the footer then we can encode into the ontology that the footer should
not be removed.

3.2.2 Triage

When a sighted user looks at a page, they immediately start to process the
information that is on the screen. Using a form of parallel processing, they can
start breaking the page down into its constituent elements, such as a menu bar
and an area of text. Using this they can quickly draw their attention to the main
area of information, scan its contents and ascertain if it contains what they are
looking for. If the page does contain information that the user wants, then the
page is read in full, as it was intended. If not, then a different site is looked at
and the process repeats until the correct information is found.

Many visually impaired computer users access on-screen information through
screen readers, which operate in a serial manner. There is no way for it to
determine where the area of important information is, so a user has to listen to
most of the page before deciding if it is what they are looking for. If it is what
they are looking for then the page has been read and no time has been lost.
However, the typical case is that it takes several attempts at searching until we
find what we are looking for, which means for a visually impaired user having to read several web pages of information that they are not interested in before getting to the information that they really want.

Solutions have been proposed to help with this. Table of Contents can be produced that summarise large text documents [31]. These give users an overview of the document’s sections, allowing them easier access to the sections of text that they want to read. Instead of generating a Table of Contents, SADIe proposes to use a triage system to help users get to the important content of the web page more easily.

The word triage comes from the French word “trier”, which means “to sort” [54]. The origins of triage come from battlefields, where there are a large number of casualties but only a limited number of medical staff. In order that treatment could be given to those who most needed it and who would most benefit from it, a tiered level of assessment was developed where a soldier was assessed and ranked according to their needs [55].

1. **Black** This class of injury was for those so severely injured that they would die of their injuries, such as large-body burns or a lethal radiation dose.

2. **Red** This class of injury was for those that required immediate surgery and were a first priority for surgical teams.

3. **Yellow** This class of injury was for those whose condition was stable for the moment but required hospital care eventually.

4. **Green** This class of injury was for those that required doctor’s care within several hours or days but not immediately, such as broken bones.

5. **White** This class of injury was for those that had minor injuries. They were deemed the “walking wounded” and first aid was sufficient.

SADIe will use this principle of tiered assessment and apply it to information on a web page. With the structure of the page captured in an ontology, elements of the page can be ranked as either High Priority, Medium Priority or Low Priority. When a High Priority element is discovered it is moved to the top of the page. If an element is of less importance, then we can give it a Medium or Low Priority and it will be moved further down the page. The elements will still be on the page and can be accessed, but at a later stage. The High Priority elements are
those which will tell the user if the page is worth reading more thoroughly, so those elements will go where they can be accessed as early as possible by the screen reader.

Looking at the blog in Figure 1.1, it could be argued that the blog post-body is the most fundamental part of the page. While the title and the footer add extra details to the page, such as the time of the blog entry, the post-body carries the information that is the essence of the page. It is this part that will determine if a visitor to the page wants to continue reading or not. Therefore post-body will be marked as High Priority in the ontology and SADIe will move this to the top of the page, where the screen reader will immediately access it.

3.2.3 Navigation

A main feature of the Web is its ability to link together documents and resources through hyperlinks. These hyperlinks allow the user to travel around the web, visiting one web page followed by another. Projects such as WAB have proposed that improving access to links is the key to improving navigation of web pages [30]. Lists of links found in the document can be generated and placed either at the top or the bottom of the page. This gives the user easier access to the links, which will aid the navigation of the web site.

SADIe will use this assertion to make navigation of the web page more accessible to a visually impaired user. Any element that is considered a menu item will be moved to the top of the page. This will give the user easy access to navigation links as it will be in one, easy to access location.

3.3 Reasoning Over Web Elements

Having captured the semantics of a web page, additional information needs to be added in order for SADIe to be able to manipulate the elements of the web page correctly. This has to be done in a uniform manner, so that there is consistency between each web page ontology. If web site designers added information, such as Removable or High Priority etc, in an ad hoc way then it would be difficult for SADIe to reason over the ontology.

For example, a web page designer wants to specify which elements of the web page are removable. There are multiple ways of doing this. A class could be created called “removable” and all removable elements added as subclasses to
it. Another option is for all removable elements to be given a property such as “Removable = true” or “RemovableElement = yes” or any other number of ways to indicate that an element is removable. By using this ad hoc way, when SADIe is asked to remove unnecessary obstacles from the page by the user SADIe will have to be able to deal with all these possible ways of specifying that an element is removable.

![Figure 3.2: SADIe Upper Ontology Class Structure](image)

One way to solve this is to use an upper ontology. An upper ontology contains a set of classes and properties that all ontologies that import it can use. This creates a template with a standard set of rules and one way of doing something, such as adding the property “RemovableComponent = true” to all removable elements. As long as the designer follows the template, SADIe will be able to query the ontology and find all the elements of a web page that it needs to remove.

SADIe supports three ways of making web pages more accessible. It can remove obstacles, rank parts of the page as High, Medium or Low Priority and put menus in a more accessible place. Based on this, the upper ontology should contain a class for each of these types of element, plus properties so that SADIe can infer which class elements belong to by using a reasoner. The SADIe Upper Ontology classes can be seen in Figure 3.2 and the properties of the SADIe Upper Ontology can be seen in Table 3.1.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>isRemovableComponent</td>
<td>true, false</td>
</tr>
<tr>
<td>hasPriority</td>
<td>high, medium, low</td>
</tr>
<tr>
<td>hasElementType</td>
<td>menu</td>
</tr>
</tbody>
</table>

Table 3.1: SADIe Upper Ontology Properties
3.4 SADIm Architecture

Taking into account the above, an architecture, as shown in Figure 3.3, can be created.

The initial stage (Figure 3.3:1) is for the web designer to build an ontology for the web page. This is achieved by going through the CSS and XHTML Class and ID attribute values to build up the structure of the document. When the structure of the page is created, the designer then imports the SADIm Upper Ontology (Figure 3.2). When the SADIm Upper Ontology is imported, the designer can add the properties provided by the Upper Ontology to the classes that represent the components of the web page. For example, in the Blogger Web Page, there is a footer element. If this was considered unnecessary then the property “RemovableCSSComponent = true” would be added to it. Figure 3.4 shows what the Blogger Web Site ontology would look like with the SADIm Upper Ontology Imported.

![Figure 3.3: SADIm Architecture](image)

When a web page is accessed by the user’s web browser, it displays the
Figure 3.4: Blogger Ontology with Imported SADIE Upper Ontology
CHAPTER 3. DESIGN

XHTML and CSS as usual (Figure 3.3:2). This is so that sighted users can view the page as the designer originally intended. If a visually impaired person is using the web browser then they can invoke the SADIe tool. SADIe gets the ontology associated with the web site and sends it to an ontology server. The ontology server then reasons over the ontology, inferring a new class hierarchy by using the properties that the designer added to the web site classes when the ontology was created (Figure 3.3:3).

For example, if the Blogger ontology from Figure 3.4 was reasoned over, the class hierarchy in Figure 3.5 would be created. This is possible because of the properties that were added to the classes that represent the Blogger Web Site. While it is possible for the designer to create this hierarchy manually and without using any properties, we feel that adding properties is a better way to do it. There are two main reasons for this. The first is that a distinct separation between the models is maintained. The ontology consists of a SADIe model, which has been imported, and a model for the web site, which the designer has created. If the models were merged together by the designer then more mistakes would be likely due to the additional complexity. It would also be much more difficult to change the ontology at a later stage if the designer wished to do so.

Figure 3.5: Inferred Blogger Ontology After Reasoning

The second reason is that by using properties, the SADIe Upper ontology can
be modified at a later stage and the changes reflected throughout all the web site ontologies that imported it. This is possible because there is only one central SADIE Upper Ontology that is imported via a URL. For example, we may decide that Low Priority elements carry so little information that they can also be considered as removable components. With properties it is possible to change the rules so that all Low Priority Elements are also classified as RemovableCSSComponents when reasoning takes place. With the property changed the next time somebody uses the SADIE tool all the Low Priority elements will be removed. If a designer manually inserts Low Priority classes from the web site as subclasses of LowPriority, then we loose the power to adapt the properties of the model and have it filter through to all the web site ontologies. An asserted subclass of LowPriority will remain there until it is manually changed.

When the Ontology Service has reasoned over the ontology, SADIE can then start to ask questions. SADIE traverses the DOM of the web page, and for every element it comes across obtains its Class or ID attribute. SADIE then asks questions of the ontology service, depending on what accessibility options the user has chosen. For example, if the user has requested that all obstacles be removed, then for every element in the DOM, SADIE will ask the Ontology Service “is this element removable?” If the Ontology Service replies “yes” then the element is removed, otherwise it is left alone and SADIE moves onto the next element. With all the elements looked at, SADIE then creates the transcoded document and displays it (Figure 3.3:4).

XHTML and CSS can be used to capture the semantics of a web page, which can then be stored in an ontology. SADIE will reason over the ontology in order to determine the meaning of the elements of a web page. By asking questions of the elements, SADIE will transcode the web pages of a web site to improve their accessibility to visually impaired users. Accessibility will be improved by removing obstacles that prevent screen readers from reading text fluently, by improving the time it takes for a user to determine if the page contains information they want to read and by providing easier access to navigation menus.
Chapter 4

Implementation

SADIe uses an ontology created from the CSS and XHTML of a web page and reasons over it in order to determine the meaning of the elements of a web page. By asking questions of the elements, SADIe will transcode the web pages of a web site to improve their accessibility to visually impaired users. For this to happen, the mechanisms of getting the ontology, reasoning over the ontology and transcoding the web page needed to be implemented.

4.1 Development Platform (XUL and JavaScript)

Many previous attempts at developing transcoding systems have relied upon a Proxy Server (Figure 2.3). With a Proxy Server implementation, communication between the client device and the web page are intercepted and the entire web document is transcoded before it reaches the user. SADIe needs to be more flexible than this. We still want sighted users to be able to view the page as it was originally intended, which means that the page must reach the client device before it is transcoded. Secondly, we want to give the visually impaired user flexibility as to which accessibility options they use. If, for example, they do not want to read the important information first, but read the document as it was intended, then the user should be able to turn triaging off. For this reason, SADIe is implemented on the client side as an extension to an existing web browser.
CHAPTER 4. IMPLEMENTATION

<!-- A box which contains the buttons at the bottom of the preferences window -->
<hbox id="preference-buttons" align="center">
  <!-- Adds the OK button and makes it the default button if the user hits return. it calls
       a function to save the user options then closes the window -->
  <button label="OK" default="true" oncommand="saveOptions();window.close();"/>
  <button label="Cancel" oncommand="window.close();"/>
</hbox>

Figure 4.1: Sample XUL Code

There are a large number of web browsers available, including Netscape Navigator ¹, Internet Explorer ² and Mozilla Suite ³, to name but a few. To develop SADIe, a web browser that was customisable was necessary, so it was decided that SADIe should be implemented as an extension to Mozilla Firefox ⁴.

Mozilla Firefox is a web browser based on the Mozilla Web Browser engine. The Mozilla Browser comes as part of the Mozilla Suite, which also includes an email client and HTML editor. While Firefox is based on the Mozilla Suite Browser, it has been redesigned so that it acts as a standalone web browser. There are three main advantages of using Mozilla Firefox. The first is that it is designed to be customisable. Extensions can easily be written, distributed and installed using a language called XUL ⁵. These extensions range from incorporating search engines into the Firefox Toolbar, to giving users a sidebar that can make last minute bids on eBay [56].

XUL is a User Interface Language based on XML. It is the language that Mozilla uses for building the interfaces of its web browsers, email clients and other web-based applications. Because XUL is designed for the rapid building of interfaces, many interface items such as pop-up menus, drop-down lists, text fields, etc are already provided. This saves the developer spending time defining interface items and their behaviour [57].

To use XUL, the designer places the interface elements in the order that they want them to appear on screen and then associates functions to them. The

¹ @Netscape Communications Corporation
² @Microsoft Corporation
³ @Mozilla Foundation
⁴ http://www.mozilla.org/products/firefox/
⁵ XML User Interface Language (pronounced “zool”)
function saveOptions()
{
    var selectedOntologyService;

    // Get the text box value that the user may have entered
    var ontologyLocationTextBox = document.getElementById("user-ontology-location");
    var ontologyLocationTextBoxValue = ontologyLocationTextBox.value;

    // get the ontology option that the user has selected by testing which option was selected
    if(document.getElementById("href").getAttribute("selected"))
    {
        selectedOntologyService = "href";
    }
    else if(document.getElementById("website").getAttribute("selected"))
    {
        selectedOntologyService = "website";
    }
    else if(document.getElementById("central").getAttribute("selected"))
    {
        selectedOntologyService = "central";
    }
    else if(document.getElementById("other").getAttribute("selected"))
    {
        selectedOntologyService = "other";
    }

    // save the user options so we can recall it at a later date
    try
    {
        preferences.setCharPref("sadie.ontology.user.location", ontologyLocationTextBoxValue);
        preferences.setCharPref("sadie.ontology.selected.location", selectedOntologyService);
    }

    // If it fails, then tell the user why
    catch (e)
    {
        alert(e);
    }
}

Figure 4.2: Sample JavaScript Code

functions define the behaviour of what will happen when the interface item is used. Therefore, for example, if the designer wanted a button that opens a new window, all that is required is for XUL to be told where to put the button on the interface and then the name of the function that opens a new window. XUL has already defined the mouse click functionality and the animation of the button being clicked; the designer just has to define what will happen.

Figure 4.1 shows some sample XUL code. This is taken from an options window, which will allow the user to change the preferences of a Firefox Extension. When the user has made the changes, they can either save the changes by clicking the “OK” button or discard them, by clicking the “Cancel” button. The code shown here is the XUL code for defining the two buttons. The designer specifies a box to put the buttons into and its justification (in this case centre). Then, the two buttons are placed inside the box and the functions that are executed when the buttons are clicked are added. In the case of OK, the preferences are saved and then the window is closed. In the case of Cancel, the window is closed.
The functions used in XUL applications are written in JavaScript. To maintain a separation of user interface code and functionality code, the JavaScript is placed in a separate file, with a reference to the file being placed at the top of the XUL file.

JavaScript is an Object-Orientated Scripting Language developed by Netscape. It was originally called LiveScript, and although the name was changed to include Java, JavaScript has very little in common with the Java programming language. JavaScript has all the constructs that other programming languages have, such as variables, logical operations and conditional statements. However, the main use of JavaScript is that it can be used for making web pages more dynamic. JavaScript comes with built-in functions for accessing and manipulating the DOM of the web page [58]. This allows designers to write functions that will adapt a page based on user choices and actions. For example, a function could be written that will change an image on the web page when a user scrolls the mouse over it. Figure 4.2 shows the JavaScript used to save the user preferences when the OK button from Figure 4.1 is pressed.

XUL and JavaScript give Firefox its second advantage as a development platform. XUL is a set of interface APIs that the designer can call and JavaScript is an interpreted language. Because of this, there is no need to compile the source code, as the Firefox Browser will do the execution for us at run time. Mozilla Firefox is available on Microsoft Windows, Apple Mac OS X and Linux, yet we can develop SADIe with one set of source code, and have access to all these platforms.

Finally, Firefox is popular, with 75 million downloads [59] so far. This will give SADIe access to a large number of potential users.

### 4.2 SADIe Ontology Service

To transcode a web page, SADIe requires access to an Ontology Service. The SADIe ontology Service is implemented as a Java application. The reason for this is that there exists a set of well-established Java OWL APIs that can be used. These APIs contain code to create ontologies, establish connections with reasoners and then query ontologies after they have been reasoned over. Rather than SADIe reinventing the wheel and creating its own ontology querying and reasoning code, it is much better to use code that has already been written, has
been debugged, is readily available and works.

The SADIe Ontology Service establishes a connection to a reasoner and creates a new ontology object. The prototypes of its three public methods are:

- public String[] isMenu(String ontologyURL, String[] classNames)
- public String[] isRemovableComponent(String ontologyURL, String[] classNames)
- public String[][] getComponentPriority(String ontologyURL, String[] classNames)

All three methods have the same parameter values. ontologyURL is the URL of the ontology to be reasoned over. classNames is an array of elements from the web page that the method will ask questions about, such as “is it a Removable CSS Component?” isMenu and isRemovableComponent both return arrays of elements that have been established as menu items or removable components respectively. getComponentPriority returns a 2D array that contains matching pairs of an element and its priority.

Arrays are used due to networks being slow and unreliable. By using an array, SADIe minimises the number of calls it has to make to the Ontology Service by sending every element it has found in the DOM in one transaction. SADIe then expects an array to be returned with the Ontology Service results. This is much quicker than, for every element in the DOM, SADIe making a call to the Ontology Service and asking a question, such as “is this removable?”, and then waiting for an answer before moving onto the next element.

Unfortunately, Mozilla prevents Java code from being executed from within JavaScript, so a way of communicating between the Firefox Extension and the OWL Java application was needed. One way to do this was to create a web service from the Java Application.

A web service is a web-based application that can be accessed via the Internet. Web services allow different applications from different sources to communicate with each other. They are not tied to any one operating system or programming language. For example, a Java application running on Windows can talk to a Perl application running on UNIX [60].

SOAP was originally an acronym for Simple Object Access Protocol but that is now considered to be outdated and is no longer an acronym for anything [61].
It is a message-based protocol based on XML for accessing Web Services on the Web. It uses XML syntax to send text commands across the Internet using HTTP [62]. SOAP is based on a four part structure. The SOAP Envelope contains the entire SOAP message. The Header contains application specific information such as authentication. The Body contains information about the application, such as its name, and the Message Payload contains information that the applications needs, such as parameter values. Figure 4.3 shows a SOAP message to a Web Service that returns the stock value of a given company [63].

```xml
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Figure 4.3: Sample SOAP Web Service Call

The benefits of using a Web Service are that Firefox and SOAP both use HTTP as a communication protocol which allows Firefox to have access to the Java Application through the Web-service. This gets around the restriction of Mozilla not allowing Java code to be executed directly from within the JavaScript.

Secondly, by using a Web Service, there is a distinct separation between the components that make up SADIe. If SADIe accessed the Ontology Service directly, then the user would be required to download the Ontology Service as well as the Firefox Extension. The consequence of this is that SADIe would be tied down to using that one Ontology Service. By using a Web Service, we can change the Ontology Service as is necessary. All the user has to do is change the URL that SADIe uses to access the Ontology Service and a different service could be accessed without breaking the code.

Finally, Mozilla provides packages that allow JavaScript to communicate with SOAP services. The packages do all the hard work in terms of building the SOAP Envelopes and establishing communications with the services. The only thing that needs to be specified in the JavaScript is the location of the Web Service, the name of the method that will be invoked on the Web Service and the
values that should be passed to the method parameters.

To create a Web Service from the Java Application, Apache Tomcat and Apache Axis was used. Apache Tomcat is Web Server that is used as a servlet container for Java Servlet and JavaServer Pages technologies [64]. A developer creates a Java Servlet and hosts it within Apache Tomcat. Tomcat then allows access to the servlet through web browsers. The compiling and execution of the servlet, multi-threading and HTTP is taken care of by Tomcat. The developer only has to concentrate on creating the actual Java Servlet code.

Apache Axis is used in conjunction with Tomcat and is an implementation of the SOAP protocol. However, with Axis, the details of dealing with SOAP and WSDL\(^6\) are hidden from the designer [65]. The designer creates an application, such as the SADIe Ontology Service, like any other standalone Java application. The class files for the application are then placed inside Apache Axis. Axis is then told the location of the service and which methods should be made public. All the Web Service machinery is then dealt with by Axis. The application developer does not have to write code so that their application can send and receive SOAP Envelopes or WSDL files. Axis does all this automatically. It allows the developer to concentrate on the application and allows it to be easily deployed as a Web Service, without the overhead of dealing with the SOAP protocol. Because it is used with Tomcat, all the benefits of web hosting and multi-threading are inherited too.

As the SADIe Ontology Service was written as a Java Application, all that was required for it to become a Web Service and accessible over the Internet was for the class files of the Ontology Service to be placed inside the Axis folder, which is inside the Tomcat Web Server. The Ontology Service was then deployed, that is Axis was told where the class files were located and which methods were to be made public over the Internet, and that was it. No additional code or protocols had to be added to the SADIe Ontology Service, yet it could be accessed from any application that was capable of sending and receiving SOAP messages. As noted, Mozilla provides this mechanism, so very little additional coding was required for a very powerful and flexible way of accessing an Ontology Service from within the SADIe Firefox Extension.

\(^6\)Web Service Definition Language
4.3 SADIe Firefox Toolbar

The user interacts with SADIe via a Firefox Toolbar Extension. When the user installs the extension a new bar is produced underneath the address bar with the SADIe interface components on it. SADIe gives the user three options to coincide with the three different kinds of accessibility features SADIe supports:

1. **Defluff.** This tells SADIe to remove all the elements of the web page that have been deemed as unnecessary and can therefore be removed.

2. **Reorder** This tells SADIe to reorder the web page using triage. The elements that are deemed most important to the web page will be put at the top of the page, followed by those of less importance and finally those elements with least importance.

3. **Toggle** This tells SADIe to find any menu or navigation links and put them at the top of the page for easier access.

SADIe is designed to give users flexibility in the accessibility choices that they make. So, for example, a user may decide that they want to “defluff” the web page combined with “toggle” but no triage. To allow for this mix and match of accessibility features, Defluff, Reorder and Toggle are implemented as a checklist. The user selects the options they want and when the choices have been made, they press the “Make Accessible” button, which starts the SADIe transcoding process. The SADIe Firefox Toolbar can be seen in Figure 4.4.

SADIe is designed for visually impaired users. Accessing GUI components can be difficult for visually impaired users so SADIe has incorporated keyboard...
combinations into the interface to make it more accessible. For example, pressing ALT + L will cause defluff to be either selected or deselected, depending on its state at the time the key combination was pressed. By adding these key combinations, we are removing the need of the user to be able to use a mouse and navigate across a GUI, which aids in the usability and accessibility of the tool. A full list of keyboard short cuts can be seen in Table 4.1.

<table>
<thead>
<tr>
<th>SADIE Toolbar Component</th>
<th>Key Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defluff</td>
<td>ALT+L</td>
</tr>
<tr>
<td>Reorder</td>
<td>ALT+R</td>
</tr>
<tr>
<td>Toggle</td>
<td>ALT+O</td>
</tr>
<tr>
<td>Make Accessible</td>
<td>ALT+M</td>
</tr>
</tbody>
</table>

Table 4.1: SADIE Toolbar Keyboard Shortcuts

When the user clicks “Make Accessible”, SADIE starts to process the web page. First it checks that the user has selected at least one accessibility feature, such as “defluff”. If this test is successful, then a list of chosen accessibility features is created and the CSS for the page is turned off so that any page formatting is removed. This includes layout, colour and font formatting. SADIE then retrieves the URL of the ontology that describes the semantics of the web page. SADIE can check in four places for the ontology.

1. **SADIE Namespace.** The header of an XHTML document can contain namespaces that point to external files, which can help describe the XHTML document. This can include links to JavaScript files or CSS files. If the document contains a namespace for a SADIE ontology, then SADIE will use the ontology URL given when reasoning over the XHTML document. A SADIE namespace for the HiTokyo blog could be:

   \[ <\text{link rel="sadie" type="application/xml" href="http://www.blogspot.com/blogspot.owl"/>} \]

2. **Root Directory.** If no link in the header is given for an ontology, then SADIE will look at the root of the web site URL for an OWL file and use that. In the case of the HiTokyo blog, this could be:

   \[ http://hitokyo.blogspot.com/hitokyo.owl \]
3. **Central Repository.** Several ontologies have been written for the SADIe tool and are stored in a central online location. SADIe can look here for an ontology for the web page that the user is currently reading. As the repository can contain ontologies for several web sites, SADIe uses a simple algorithm to determine which ontology to use. Most web site URLs are in the format www.WEB-SITE-NAME.com or www.WEB-SITE-NAME.co.uk. Typically, it is the WEB-SITE-NAME that distinguishes the web site from any other, so SADIe extracts this and looks for an OWL file with that name. In the case of the HiTokyo blog, this could be:

http://www2.cs.man.ac.uk/lunnd/sadie/ontologies/blogspot.owl

4. **User Repository.** Sometimes an ontology will not be written for a web site that the user wants to look at. If this is the case then there is nothing to stop the user creating their own ontologies for the web site and creating their own local repository. SADIe will use the same algorithm as above to determine which ontologies match which web sites. In the case of the HiTokyo blog, this could be:

file:///C:/home/darren/sadie/ontologies/blogspot.owl

The user can tell SADIe where to look for the web page ontologies. This can be achieved by using the SADIe Options Window. Again, the benefits of the Firefox development environment can be exploited. Firefox tries to maintain a consistent interface for all its extensions. If a Firefox Extension has the ability to let users set preferences, then the same procedure is used to call up the Options Window. The user will select the “Tools” menu in the Firefox Browser followed by “Extensions”. This brings up a list of all the extensions currently installed. The user then selects the extension that they want to set the preferences for, such as SADIe, and clicks the “Options” button. This will causes the Options Window for that extension to appear. All the designer has to do is specify, using XUL and JavaScript, what will appear in the Options Window. The SADIe Options Window can be seen in Figure 4.5.

With the ontology for the web site obtained, SADIe then gets all the ID and Class attribute values. This is achieved by traversing the entire DOM of the web page. Having acquired all the attribute values, SADIe then gets the first
accessibility feature from the list of user selected options it created previously and creates a call to the Ontology Service.

The call made to the Ontology Service is dependent upon which accessibility feature SADIe is currently performing. If, for example, SADIe is Defluffing the page, then the call to the Ontology Service would be isRemovableComponent, to get a list of elements that are removable. With the list of removable elements returned by the Ontology Service, SADIe then traverses the DOM once again. This time, rather than creating a list of all the elements in the DOM, SADIe makes a comparison between the element in the DOM and the list of removable elements. If a match can be found, then the element is removed, otherwise it is left alone and the next element is inspected. The result of Defluffing the HiTokyo Blog in Figure 1.1 can be seen in 4.6. All the formatting has been removed, and the sidebar that was on the right-hand-side of the page has gone as it was deemed to be unnecessary. All that is left are the blog entries, the title of the blog and the ability to search for more blogs.

This process is the same for every accessibility feature the user has asked SADIe to perform. Selecting Reorder will tell SADIe to put the important information at the top of the page. A call to the Ontology Service would request that a list of element priorities is returned followed by SADIe traversing the DOM and rearranging all the elements, putting those with a higher priority near the top of the page. For a blog, the most important information is the blog entry itself, so this is what the user will want immediate access to, in order to be able
CHAPTER 4. IMPLEMENTATION

Figure 4.6: HiTokyo Blog After Defluffing

So I had my class and what would you know my idea worked. I also realised that because in the past I was dreading the class, I was also a bit hyper-sensitive to his behaviour, which meant I was getting more nervous and doing worse. But tonight I was more chilled and not effected by his attitude and so by the end of the class his attitude had become more positive and he was actually smiling at the end of the class. So as Si says Huraah!
to determine if the page contains information that they want to read. The result of Reordering the HiTokyo Blog in Figure 1.1 can be seen in 4.7.

The order in which accessibility features are applied to a web page is not important. A user, for example, could select “menu” and then click the “make accessible” button. Having looked at the transcoded page, they could decide that they want to defluff the page. SADIe will apply defluffing as requested after toggling the menu without any problems. However, it is more efficient for a user to select the options that they want to use and have SADIe apply them at the same time. This is because to improve efficiency when applying multiple accessibility features, SADIe exploits the fact that the Ontology Service is independent of SADIe. When a call has been made to the ontology service, before applying the results to the web page DOM SADIe checks to see if there are any more accessibility features it must apply. If there are then SADIe will make another call to the Ontology Service before transcoding the web page. The Ontology Service can then perform its reasoning and querying of the ontology at the same time SADIe is performing the first transcoding of the page.

Figure 4.7: HiTokyo Blog After Reordering
For example, if the user has selected all three accessibility features, that is Defluff, Reorder and Toggle, then SADIe will first call the Ontology Service for a list of removable elements. When the list returns, SADIe will then check what other accessibility techniques need to be applied. It finds Reorder, so calls the Ontology Service for a list of elements and their priorities. It is after this call that SADIe will start to traverse the DOM and start removing elements. When all the elements have been inspected and removed, SADIe will then check to see if the results of the Ontology Service have returned. As soon as they have returned, SADIe will check to see if there are more accessibility techniques to be applied and in this case calls the Ontology Service for a list of menu elements. The DOM is then traversed and rearranged based on the element priorities and finally the menu elements are placed at the top of the page as soon as the Ontology Service has returned its results for Menu Elements. If a user selects the accessibility features and applies them individually, then this efficiency gain cannot be exploited.

Figure 4.8: The Mac Observer Before SADIe Transcoding
SADIe isn’t restricted to transcoding blog entries. Figure 4.8 shows The Mac Observer web site. This is a web site that has news and features for owners of Apple computers. This web site would be hard to access for a visually impaired user. The screen reader would not be able to deal with three columns of text and would read a line from each column in one continuous sentence. There are lots of images to break up the flow of the screen reader and it would be hard to obtain the main information of the page, which is the latest Apple News. By applying Defluff, Reorder and Toggle, the page is transcoded into that seen in Figure 4.9. Here the menu is at the top of the page where it can be easily accessed as a small list of links to other sections of the site. Immediately after the navigation menu are some of today’s Apple news stories, plus a short blurb to indicate what the full article contains. These are considered to be the highest priority elements of the page as the web site is about news stories related to Apple. The text on the page is in continuous blocks that are not broken up by obstacles or columns of other text and there are no images or advertisements to act as a hindrance to the user when they access the page.

Figure 4.9: The Mac Observer After SADIe Transcoding
SADIe is implemented as a three-stage process. There is an ontology creation stage, where the semantics of a web page are captured and properties added to the elements. This is followed by a reasoning stage, where SADIe asks questions about the different kind of elements that are on the web page, and finally the transcoding stage, where the web page is made more accessible. In order for users to interact with SADIe, a Firefox Toolbar Extension is provided as a mechanism to allow users to choose what accessibility features they want to use and where the ontology for the page they are looking at is located. To see if SADIe’s transcoding is of use to visually impaired users, evaluation was performed on a selection of web pages.
Chapter 5

Evaluation

Users interact with SADIe via a Firefox Toolbar Extension, which provides a mechanism to allow them to choose which accessibility features they want to use and where the ontology for the page they are looking at is located. To see if SADIe’s web page transcoding is of use to visually impaired users we evaluated the tool on several web pages and analysed the results.

5.1 Evaluation Criteria

Evaluation is a way of examining a system to determine the extent to which specified properties are present [66]. SADIe therefore needed to be evaluated to see if it met the specified properties of making web pages more accessible to visually impaired users.

The W3C has a Web Accessibility Initiative (WAI), which provides strategies and guidelines that web designers can use to make the Web accessible to people with disabilities. Part of the WAI is to provide guidelines on evaluating a web site to ensure that it is accessible. These guidelines are targeted at designers using current technology and techniques, such as using CSS and XHTML correctly and ensuring that all images have an alternative description. The main focus of SADIe isn’t web site design, but some of the principles in the guidelines can be applied when evaluating SADIe. The W3C guidelines include steps such as [67]:

1. Select a sample of different kinds of pages from the Web site to be reviewed. This must include all the pages that people are likely to use to enter the site.
2. Use a graphical user interface browser and examine a selection of pages while adjusting the browser settings.

3. Use a voice browser or a text browser and examine the Web site while checking if equivalent information available through the voice or text browser is available through the GUI browser and that the information presented is in a meaningful order if read serially.

The first step of the W3C guidelines is choosing a sample of web pages. The W3C guidelines are targeted at designers of a single web site, which has a limited number of pages, but SADIe is designed for the entire Web. The Web has a large number of web pages on it, with Google using 8,168,684,336 web pages [68] when it performs its searches. A sample of these web pages had to be selected in a way that made the sample small enough so that the evaluation was practical, yet large enough to provide accurate results.

Sampling is a way of estimating error during evaluations. There are two kinds of errors that can occur. There is the sample error, which is the number of failures that occur within the sample. Then there is the true error. This is the error for the entire population that we could potentially have access to, in this case 8,168,684,336 web pages. The true error and the sample error usually give different values. This is because a sample usually does not perfectly reflect the make-up of the entire population. Taking two samples will therefore give two slightly different error rates. Ideally we want to find the true error because this will tell us how accurate SADIe is when it is used on the web, but this can only be found if we test SADIe on the entire population of web pages.

When using sampling, Confidence Intervals can be used to answer the question “How good an estimate of the true error is provided by the sample error?”[69]. By using Confidence Intervals, an upper and lower limit is discovered, within which the true error lies. The upper and lower limits provide a level of confidence, for example 95% certainty, that the range discovered contains the true error [70].

Confidence Intervals are derived from the Central Limit Theorem. This theorem states that if we create a sample from a population with a sufficiently large number \(n\), the distribution of the sample is approximately normal. In principle, there is no way to know how big the sample \(n\) must be, but there is a well-established rule-of-thumb that if \(n \geq 30\), then we can assume that \(n\) is sufficiently large for the sample to be Normally Distributed [71].
Figure 5.1 shows the shape of the Normal Distribution and shows two well-accepted properties. They are, that it is symmetrical about the mean and that 95.45% of the area underneath the curve will lie within two standard deviations of the mean (the shaded area in Figure 5.1).

As noted above, taking two samples and calculating their error rates will give two slightly different results, meaning that we cannot be sure what the true population error is. To help solve this problem the Central Limit Theorem can be applied to estimate the range in which the true population error lies. If the sample is greater than 30, then the Central Limit Theorem tells us that the sample is normally distributed. It is beyond the scope of this dissertation to prove the Central Limit Theorem and Confidence Intervals, but it can be shown that if we take a large number of samples from a population, due to the properties of the Normal Distribution, 95.45% of the time, the sample errors will lie between

\[ r \pm 2 \sqrt{\frac{r(1-r)}{n}} \]

where:

- \( r \) = The sample error rate
- \( n \) = The size of the sample

Therefore we can assume that this is the range within which the true population error lies. We can call this range the 95.45% confidence interval because we are 95.45% certain that the population error will lie within this range.

The \( z \) value in Figure 5.1 is the standard deviation value and determines the confidence level. For example, when \( z = 2 \), the confidence level is 95.45%. We can change the confidence level by changing the value of \( z \) to give us the formula.

\[ r \pm z \sqrt{\frac{r(1-r)}{n}} \]

To find the \( z \) value for different confidence levels, \( Z \)-Tables are used. These are pre-calculated tables that map values of \( z \) to percentage areas of the Normal Distribution curve. A \( Z \)-Table is given in Appendix A.

For example, suppose that a sample \( n \) is taken of 40 web pages and 12 of them fail the SADIe evaluation test, the sample error \( r \) would be \( \frac{12}{40} = 0.30 \). This is only the error for this sample, and a different sample would produce a different error rate, therefore we cannot say that this is the true error. If we wanted to
find the range within which the true error lay with 95% certainty, then looking at
the $z$-table, we can see that the $z$ value we require is 1.96. By using these values
in the formula above, we get $0.30 \pm 1.96 \sqrt{\frac{0.30 \times 0.70}{40}} = 0.30 \pm 0.14$. This means that
we can be 95% certain that the true error rate for the entire population of web
pages would lie in the range 0.16 – 0.44 [69].

A simple way of evaluating SADiE is to analyse the transcoded page and
classify it as either successful or a failure. This will give us a sample error and
allow us to then use $r \pm z\sqrt{\frac{r(1-r)}{n}}$ to determine the true error of SADiE within a
given level of confidence. For this to work, SADiE will need to be tested on at
least 30 pages, because of the properties of the Central Limit Theorem.

Having determined the minimum number of pages to have in the sample, the
next question was “how do we choose the sample of pages so that it fairly and accurately represents existing web pages on the Web?” IBM proposed that while web sites are different visually, if a web site’s role is taken into account, then there are some similarities [72]. By using web site roles, IBM produced eight categories that can be used for classifying web sites. These categories are Corporate Sites, Content and Media, Search Engines, Web Hierarchies and Directories, Portals, E-Stores, Virtual Hosting and Universities. By using these eight categories, we could be sure that the evaluation of SADIe used a fair sample of the kinds of web sites that potential users of SADIe may access.

While it was important to determine how to evaluate SADIe in terms of the web site samples and steps taken during evaluation, it was equally important to determine what to evaluate SADIe for, in terms of making web pages accessible.

A big hindrance to visually impaired users when navigating a web site is that they meet obstacles that prevent them from accessing the information that they want. These obstacles can take many forms, including banner advertisements, images to improve the aesthetics of the page and frames. Many techniques, such as Silas Brown’s Web Access Gateway [53] improve accessibility by removing objects that are considered to be inaccessible to users. Objects that can be removed include JavaScript, embedded media, background images and blinking text, to name but a few. This is the approach SADIe also takes, although the person creating the web page ontology determines what obstacles are removed.

Many visually impaired computer users access on-screen information through screen readers, which operate in a serial manner. Columns of text and tables can break up this serial reading, making the sentences that are read aloud incomprehensible. Many solutions to this problem have been proposed. Several of them involve collapsing tables into individual blocks of text. This can involve moving table cells with lots of links to the bottom of the page or removing columns that contain little text [73]. SADIe reorders web pages so that the important information is placed near the top of the page. It also tries to organise the layout to make sure that the flow of text is not broken.

SADIe also tries to deal with the problems of navigation on web pages. Projects such as WAB have proposed that improving access to links is the key to improving navigation of web pages [30]. Lists of links found in the document can be generated and placed either at the top or the bottom of the page. This gives the user easier access to the links, which will aid the navigation of the web site.
CHAPTER 5. EVALUATION

Asking a group of visually impaired users to use SADIE to access the web would have been an ideal evaluation, but organising such a large-scale test takes time and resources that were not available during the project. However, by using W3C guidelines and taking into account what SADIE was designed to do as well as using the findings of the research indicated above, a series of benchmarks could be produced, against which we could compare the outcome of using SADIE on a web page. The SADIE evaluation procedure was as follows:

1. **Choose Sample Web Pages:** IBM classified web sites into eight categories but to get an accurate true error range using confidence intervals, we needed at least 30 samples. Therefore we decided to select 5 web sites for each IBM category. This gave us a total of 40 web sites in the sample. The W3C guidelines specify that when evaluating a web site the entry point should be tested, as this is the first page the users will access. Therefore, we decided that we should test the entry point (usually index.html) of the web site, plus 4 other randomly chosen pages on the web site. This gave us a total of 5 pages per web site. With 40 web sites, we would be testing 200 web pages in total.

2. **Apply SADIE to Each Page:** To make the evaluation fair, we applied all the accessibility features of SADIE to every page. That is, we applied Defluff, Reorder and Toggle to the page and observed the results.

3. **Evaluate Results of SADIE for Each Page:** Due to the fact that we had no visually impaired users and no screen readers, we had to determine if the transcoding was successful by observing the page. Taking into account what SADIE was designed to do and the research, we asked the following questions of a transcoded page:

   (a) Have all the obstacles that are marked as removable in the ontology been removed?

   (b) Are there multiple columns on the page?

   (c) Has all formatting that uses tables been removed?

   (d) Is there anything that breaks up the flow of text?

   (e) Are all blocks of text aligned vertically as opposed to horizontally?

   (f) Are all navigation links at the top of the page?
If the answer was "yes" to all these questions, then the page was considered to have been successfully transcoded by SADIe. If the answer was "no" to any question, then the page was considered a failure and the reason noted down to help with further studies into the effectiveness of SADIe and what its exact context of use was.

4. **Determine Web Site Success**: In determining a web site’s success or failure, we used the entry point to determine if the site succeeded or failed. The reasoning behind this was that most people would want to read the access point of a web site as that page usually provides links to other sections of the site plus a general overview of the information that can be found on the site. If we can make a page that most people use accessible, then that is more important for the site than providing access to a page that few people will ever read.

Having established a framework for the evaluation, it was then applied to a sample of web pages. To obtain the sample, the first five web sites from each of the eight IBM categories were used. This was to try and add an element of randomness to the testing of the web sites, rather than trying to put more thought into the selection and having bias towards pages that we thought might transcode successfully.

### 5.2 Evaluation Results

Table 5.1 shows the results of the SADIe evaluation. The 40 web sites and their categories are noted as well as how many of the web pages on the site failed the SADIe evaluation and if the entry point was a success or not. We also noted down how the presentation of the site was achieved. If only CSS was used, then this was noted as “Pure” as it used pure CSS formatting. If no CSS was used then this was noted down as “None.” Mixture was for those sites that use CSS for formatting fonts and colours and headings etc, but use tables for layout purposes.

Table 5.2 shows a summary of the results found from the SADIe evaluation. The results are broken down to show the success rate of the pure CSS web sites, those that mixed CSS and table presentation styles and those that used no CSS for the presentation. These three categories are then summarised even further. The Combined CSS Type is the result of all 40 web sites added together. The
### CHAPTER 5. EVALUATION

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>URL</th>
<th>CSS</th>
<th>Failure</th>
<th>Entry Point</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Corporate Sites</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Microsoft Corporation</td>
<td><a href="http://www.microsoft.com/">http://www.microsoft.com/</a></td>
<td>Mixture</td>
<td>2</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Digital Designs</td>
<td><a href="http://www.digitaldesign.us">http://www.digitaldesign.us</a></td>
<td>Pure</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Stagecoach Buses</td>
<td><a href="http://www.stagecoachbus.com/">http://www.stagecoachbus.com/</a></td>
<td>Pure</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>British Nuclear Fuels</td>
<td><a href="http://www.bnfl.com/">http://www.bnfl.com/</a></td>
<td>Pure</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Epson Corporation</td>
<td><a href="http://www.epson.co.jp/s/">http://www.epson.co.jp/s/</a></td>
<td>Mixture</td>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td><strong>Content &amp; Media</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Blogger</td>
<td><a href="http://www.blogger.com/">http://www.blogger.com/</a></td>
<td>Pure</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>The Mac Observer</td>
<td><a href="http://www.macobserver.com/">http://www.macobserver.com/</a></td>
<td>Pure</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>New Musical Express</td>
<td><a href="http://www.nme.com/">http://www.nme.com/</a></td>
<td>Mixture</td>
<td>5</td>
<td>Failure</td>
</tr>
<tr>
<td></td>
<td>BBC News</td>
<td><a href="http://news.bbc.co.uk/">http://news.bbc.co.uk/</a></td>
<td>Mixture</td>
<td>2</td>
<td>Failure</td>
</tr>
<tr>
<td></td>
<td>CNN International</td>
<td><a href="http://edition.cnn.com/">http://edition.cnn.com/</a></td>
<td>Mixture</td>
<td>1</td>
<td>Failure</td>
</tr>
<tr>
<td><strong>Search Engine</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Google</td>
<td><a href="http://www.google.co.uk/">http://www.google.co.uk/</a></td>
<td>None</td>
<td>5</td>
<td>Failure</td>
</tr>
<tr>
<td></td>
<td>Yahoo</td>
<td><a href="http://uk.yahoo.com/">http://uk.yahoo.com/</a></td>
<td>Mixture</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Ask Jeeves</td>
<td><a href="http://www.askjees.co.uk/">http://www.askjees.co.uk/</a></td>
<td>Mixture</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>MSN Search</td>
<td><a href="http://search.msn.com/">http://search.msn.com/</a></td>
<td>Pure</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>HotBot</td>
<td><a href="http://www.hotbot.co.uk/">http://www.hotbot.co.uk/</a></td>
<td>Pure</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td><strong>Web Hierarchies &amp; Directories</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Google Directory</td>
<td><a href="http://directory.google.co.uk/">http://directory.google.co.uk/</a></td>
<td>None</td>
<td>5</td>
<td>Failure</td>
</tr>
<tr>
<td></td>
<td>Yahoo Directory</td>
<td><a href="http://uk.dir.yahoo.com/">http://uk.dir.yahoo.com/</a></td>
<td>None</td>
<td>5</td>
<td>Failure</td>
</tr>
<tr>
<td></td>
<td>This Is Our Year</td>
<td><a href="http://www.thisisouryear.com/">http://www.thisisouryear.com/</a></td>
<td>Mixture</td>
<td>2</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>HotSheet</td>
<td><a href="http://www.hotshot.com/">http://www.hotshot.com/</a></td>
<td>Pure</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>HaaBaa Web Directory</td>
<td><a href="http://www.haabaa.com/">http://www.haabaa.com/</a></td>
<td>Mixture</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td><strong>Portals</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AOL UK</td>
<td><a href="http://www.aol.com/">http://www.aol.com/</a></td>
<td>Mixture</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>MSN UK</td>
<td><a href="http://www.msn.co.uk/">http://www.msn.co.uk/</a></td>
<td>Mixture</td>
<td>2</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Wanadoo</td>
<td><a href="http://www.wanadoo.co.uk/">http://www.wanadoo.co.uk/</a></td>
<td>Mixture</td>
<td>4</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Virgin Net</td>
<td><a href="http://www.virgin.net/">http://www.virgin.net/</a></td>
<td>Mixture</td>
<td>4</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Tiscali UK</td>
<td><a href="http://www.tiscali.co.uk/">http://www.tiscali.co.uk/</a></td>
<td>Pure</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td><strong>E-stores</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Play</td>
<td><a href="http://www.play.com/">http://www.play.com/</a></td>
<td>Mixture</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Amazon UK</td>
<td><a href="http://www.amazon.co.uk/">http://www.amazon.co.uk/</a></td>
<td>None</td>
<td>5</td>
<td>Failure</td>
</tr>
<tr>
<td></td>
<td>Tiny</td>
<td><a href="http://www.tiny.com/">http://www.tiny.com/</a></td>
<td>Mixture</td>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Tesco</td>
<td><a href="http://www.tesco.com/">http://www.tesco.com/</a></td>
<td>Mixture</td>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Red Letter Days</td>
<td><a href="http://www.redletterdays.co.uk/">http://www.redletterdays.co.uk/</a></td>
<td>Mixture</td>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td><strong>Virtual Hosting</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Brazenet</td>
<td><a href="http://www.brazenet.com/">http://www.brazenet.com/</a></td>
<td>Mixture</td>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>InMotion Hosting</td>
<td><a href="http://www.inmotionhosting.com/">http://www.inmotionhosting.com/</a></td>
<td>None</td>
<td>5</td>
<td>Failure</td>
</tr>
<tr>
<td></td>
<td>Path Host</td>
<td><a href="http://www.pathhost.net/">http://www.pathhost.net/</a></td>
<td>Mixture</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>Honest Web Host</td>
<td><a href="http://www.honestwebhost.com/">http://www.honestwebhost.com/</a></td>
<td>Mixture</td>
<td>5</td>
<td>Failure</td>
</tr>
<tr>
<td></td>
<td>Netwalker Internet Services</td>
<td><a href="http://www.netwalker.biz/">http://www.netwalker.biz/</a></td>
<td>Mixture</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td><strong>Universities</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>University of Manchester</td>
<td><a href="http://www.manchester.ac.uk/">http://www.manchester.ac.uk/</a></td>
<td>Mixture</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>University of York</td>
<td><a href="http://www.york.ac.uk/">http://www.york.ac.uk/</a></td>
<td>Mixture</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>University of Sheffield</td>
<td><a href="http://www.sheffield.ac.uk/">http://www.sheffield.ac.uk/</a></td>
<td>Mixture</td>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>University of Oxford</td>
<td><a href="http://www.ox.ac.uk/">http://www.ox.ac.uk/</a></td>
<td>Mixture</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td></td>
<td>University of Southampton</td>
<td><a href="http://www.soton.ac.uk/">http://www.soton.ac.uk/</a></td>
<td>Mixture</td>
<td>1</td>
<td>Success</td>
</tr>
</tbody>
</table>

Table 5.1: SADIE Evaluation Results
Pure + Mix CSS Type is the combined results of only those web sites that used Pure CSS for presentation and those that used a mixture of CSS and tables for presentation. The reason for factoring out the web sites that used no CSS is because part of the design specification for SADIe was that it captured document structure by using CSS and XHTML. If there is no CSS then by design, SADIe will be unable to transcode the page. It is unproductive to evaluate something that we know will fail, so we give a summary of results that do not include those pages that used no CSS for presentation.

<table>
<thead>
<tr>
<th>CSS Type</th>
<th>Site Sample</th>
<th>Site Failures</th>
<th>Sample Error</th>
<th>True Error Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure</td>
<td>9</td>
<td>0</td>
<td>0.00</td>
<td>0 - 0</td>
</tr>
<tr>
<td>Mixture</td>
<td>26</td>
<td>4</td>
<td>0.15</td>
<td>0.02 - 0.28</td>
</tr>
<tr>
<td>None</td>
<td>5</td>
<td>5</td>
<td>1.00</td>
<td>1.00 - 1.00</td>
</tr>
<tr>
<td>Combined</td>
<td>40</td>
<td>9</td>
<td>0.23</td>
<td>0.11 - 0.35</td>
</tr>
<tr>
<td>Pure + Mix</td>
<td>35</td>
<td>4</td>
<td>0.11</td>
<td>0.01 - 0.21</td>
</tr>
</tbody>
</table>

Table 5.2: SADIe Web Site Evaluation Summary

The column “Site Failure” shows how many entry points failed to be correctly transcoded by SADIe from the sample of web sites tested. The sample error is the proportion of web sites from the sample that failed. The True Error Range is the range in which the true error lies for that class of web site, calculated by using the 95% confidence interval.

For example, taking the results of web sites that used a mixture of tables and CSS for presentation, we can see from Table 5.2 that the sample error was 0.15. Using 95% confidence intervals, we require a $z$ value for the $\frac{1}{2}(100 - 95)\% = 2.5\%$ area of the Normal Distribution curve. Looking at the $z$-table in Appendix A, we can see that the $z$ value we require is 1.96. Using the sample size and error rate, we can use the formula from above to get $0.15 \pm 1.96\sqrt{0.15 \times 0.85 \over 26} = 0.15 \pm 0.13$. From this, we can determine that the True Error when SADIe is used with mixed presentation web sites is between 0.02 - 0.28. That is, we are 95% confident that between 2% and 28% of mixed presentation web sites will fail when used by SADIe.

From looking at Table 5.2, we can see that the sites that used no CSS for presentation had a 100% failure rate. This was expected due to the fact that SADIe relies upon the CSS to capture the structure of the web page. If there is no CSS, then the structure cannot be captured, so there is nothing for SADIe to
CHAPTER 5. EVALUATION

reason over or use for transocoding and hence the failure.

Discounting the sites that used no CSS, we can see that SADIe obtained quite
reasonable results. Table 5.2 shows that the error rate for sites that used Pure
CSS for presentation had a 0% failure rate. When the sites that used mixed
presentation are included, the error rate increases. A reason for this is because of
not being able to separate columns of text. During the evaluation it was observed
that a common approach adopted by these mixed sites was to give the entire table
a CSS class value, which SADIe could use, but not give the elements within the
cells of the table a Class or ID value. So while SADIe could remove or reorder
the table as a whole, the contents within the table were inaccessible to SADIe
and so remained in columns. This in turn meant that the screen reader would be
unable to read the text properly and the page was deemed a failure. However,
there were still a large number of web pages that were successful that mixed CSS
and tables for presentation. Table 5.2 shows that the error rate for this category
was 11%, with the true error lying in the range of 1% and 21%.

While these results are encouraging, they must be taken with a pinch of salt as
we are making several assumptions. The first is that we are assuming that web
site design follows the Normal Distribution. While many things in science do
conform to the Normal Distribution, for example IQ rates, biological phenomena
such as the range in weights for members of a species, or social science factors
such as wealth distribution, it is not entirely clear that the Web does follow the
Normal Distribution, or how you could prove it.

Secondly, we are assuming that our sample is an accurate reflection of the
web pages that are available on the Web. IBM’s proposal of eight web site
categories based on the role of a web site [72] provides a good guidance for
selection. However, it is difficult to say that choosing only 5 web sites for each
category, which we did, could accurately reflect that category when the number
is so small and the selection was not purely random.

Finally, we are basing success and failure on how the page looks after transcod-
ing. While we can make a valued judgement that the transcoded page will be
more accessible, based on research in the field, we can never be truly sure of
SADIe’s success until we test it with some real users and get their feedback and
opinions.

While these assumptions need to be addressed, the initial results of SADIe
look promising. As Table 5.2 shows, the combined error rate when we tested web
pages that used pure and mixed presentation was only 11%. While we are not claiming that SADIe can successfully transcode 89% of all web sites and make them accessible, this initial result does provide a good basis for future evaluations and tests to see how useful SADIe is.
Chapter 6

Conclusion

Use of the World Wide Web has grown exponentially over the past two decades and has become an essential part of everyday life for most people. It is common for users of the Web to shop, obtain news and weather information or search for information from the large number of web pages that are available.

Many of the people who design these pages for the Web have moved from the print world to the web design world. This has meant that they have brought their print world skills with them so at design time, they have in their mind a vision of how their creation should look and feel. This in turn leads to a web page that is a static work of art that should not be modified or adapted.

While these visually attractive sites can be accessed by a large number of users, it is not suitable for all users. In particular, people with visual impairments have trouble accessing the information that is contained within the page. One reason for this is that the web designers only think of ways in which to present their information because of their background in graphics and design. This creates a visual-centric web page where information is only made available implicitly from how data looks and where it is located.

Implicit information can cause problems for visually impaired users in several ways. The first problem is that implicit information is lost to screen readers because they have no concept of visual layout. Text can be different colours, different fonts or highlighted in different ways to indicate its meaning, such as a navigation menu, but a screen reader will not pick this up. It will just read what is displayed on screen as if it was plain text. If sighted users gain knowledge from the layout of the data, then this knowledge is lost when being read by a screen reader, making it inaccessible.
Secondly, layout usually creates blocks of text, such as navigation menus often being placed at the side of the screen, with the main content next to that in a separate block of text. These blocks are only seen to be blocks because of their visual layout. As a screen reader has no concept of visual layout, it is difficult for the user to get access to the separate blocks of text via a screen reader.

Finally, a screen reader reads a page from left to right, top to bottom. There is no way in which it can determine where the area of important information is, so a user has to listen to most of the page before deciding if it is what they are looking for. This means having to read a lot of information that a user is not interested in before getting to the information that they really want.

One way to make these visual-centric web pages more accessible to visually impaired users is to capture the implicit information through encoding the meaning of the information. By capturing the semantics of the web page, the implicit information can be made explicit. This can be achieved through annotation. By annotating what different parts of the page are, it is possible to develop tools that can use the annotation to manipulate and adapt the page to make it more accessible.

Unfortunately, annotation can be expensive in terms of time and many web designers will not adopt techniques that are seen to create too many overheads and give little in return.

Web site design can therefore be seen to have three conflicting goals. Firstly, web pages will always be designed to be beautiful in an attempt to draw the attention of sighted web users. Secondly, designers will not adopt techniques they see as unnecessary or costly. Thirdly, there is a need to make implicit information on web sites available to visually impaired users so that everybody has the same access to the information.

SADIe attempts to solve these three conflicting criteria by creating an application that allows sighted users to see web sites as they were intended but allows the page to be transcoded so that it is more accessible to visually impaired users. At the same time, SADIe tries to keep the additional design overhead to a minimum. This is achieved by making implicit information explicit through the use of a web page’s CSS and XHTML to create an ontology.

The ontology can be created because XHTML tags can contain attributes, such as ID or Class. In web design, these attributes are used with CSS to determine how to display the elements of the page. While designers use these ID and
Class values for determining the layout of the page, they are also inadvertently capturing the semantics of the web page. This allows us to build up a semantic representation of the document and its structure. We do not need to add additional annotation to know what a navigation object is, or what a sidebar object is. The values that determine how objects are displayed can also tell us what the objects are. This eliminates the need for time-consuming annotations on every page, creates only a small amount of additional overhead for the designer, yet allows us to capture the semantics of a substantial number of web pages, whether they be legacy pages or new pages.

Accessibility is the removing of barriers that prevent people with disabilities from participating in life activities [18]. SADIe attempts to make web pages more accessible to visually impaired users in three ways.

Firstly, SADIe Defluffs the page by removing obstacles from the page. A big hindrance to visually impaired users when navigating a web site is that they meet obstacles that prevent them from accessing the information that they want. These obstacles can take many forms, including banner advertisements, images to improve the aesthetics of the page or frames. By removing the obstacles screen readers can read the text without having the flow broken.

Secondly, SADIe Reorders the page and allows quicker access to the most important information on the page. When a sighted user looks at a page, they can start breaking the page down into its constituent elements, such as a menu bar and an area of text. Using this they can quickly draw their attention to the main area of information, scan its contents and ascertain if it contains what they are looking for. Many visually impaired computer users access on-screen information through screen readers, which operate in a serial manner. There is no way for it to determine where the area of important information is so a user has to listen to most of the page before deciding if it is what they are looking for.

SADIe uses the principle of triage to rank the information on a web page. With the structure of the page captured in an ontology, elements of the page can be ranked as either High Priority, Medium Priority or Low Priority. When a High Priority element is discovered it is moved to the top of the page. If an element is of less importance, then we can give it a Medium or Low Priority, and it will be moved further down the page. The High Priority elements are those which will tell the user if the page is worth reading more thoroughly, so those elements will go where they can be accessed as early as possible by the screen reader.
Finally, SADIe Toggles Menus on the page to allow quicker access to navigation menus. A main feature of the Web is its ability to link together documents and resources through hyperlinks. These hyperlinks allow the user to travel around the web, visiting one web page followed by another. SADIe makes navigation of the web page more accessible to a visually impaired user by moving any element that is considered a menu item to the top of the page. This gives the user easier access to navigation links as they are in one, easy to access, location.

To achieve these three accessibility functions of Defluff, Reorder and Toggle, SADIe is implemented as a three-stage process. There is an ontology creation stage, where the semantics of a web page are captured from the XHTML and CSS and properties added to the elements. This is followed by a reasoning stage, where SADIe asks questions about the different kinds of elements that are on the web page, and finally the transcoding stage, where the web page is made more accessible.

In order for users to interact with SADIe, a Firefox Toolbar Extension is provided as a mechanism to allow users to choose what accessibility features they want to use and where the ontology for the page they are looking at is located. It was the implementation in Firefox that caused most of the problems when developing SADIe. Because SADIe was developed as an extension to Firefox, the SADIe toolbar had to follow the Firefox style. This is so that any new extension for Firefox is consistent with all the other features and extensions of the Firefox Browser. SADIe was designed for visually impaired users. This does not necessarily mean that the user has no vision, but maybe a slight vision impairment. One way to help people with slight vision impairments is to have highly contrasting text, for example large yellow fonts on a black background [29]. Because SADIe was bound by the design style of Firefox, it was not possible to do this and so the text used by SADIe was a small black font on a grey background, which is the same style that Firefox uses. This is not very easy to read and the ability to use a high contrasting style would have been nice.

A further problem arose when we tried to implement keyboard combinations. Accessing GUI components can be difficult for visually impaired users so keyboard combinations can be added to the interface to make it more accessible. For example, it would have been logical to use the combination ALT + D to select the Defluff option for SADIe. Unfortunately this was already reserved by Firefox, with the ALT + D combination highlighting the address bar. This meant that we
had to find alternative unused combinations for the SADIE toolbar features. ALT + L was used for Defluff and other less logical choices for the other accessibility features. A full list of keyboard short cuts can be seen in Table 4.1.

While visually impaired users did not evaluate SADIE, research into accessibility was used as a guide to see if SADIE met accepted accessibility techniques. The evaluation involved choosing a sample of 40 web sites. To get a good sample of different types of web pages, IBM’s eight web site categories of Corporate Sites, Content and Media, Search Engines, Web Hierarchies and Directories, Portals, E-Stores, Virtual Hosting and Universities was used, with 5 web sites being chosen from each category. Five pages for each web site, including the entry point (usually index.html) of the web site were tested with the SADIE tool. To make the evaluation fair Defluff, Reorder and Toggle was applied to every page and the results observed.

The sites that were tested were split into three groups consisting of web sites that used Pure CSS for presentation, no CSS for presentation or a mixture of CSS and Tables for presentation. The sites that used no CSS were not included in the results. The reason for this was because by design, SADIE relies upon the CSS to capture the structure of the web page. If there is no CSS, then the structure cannot be captured, so there is nothing for SADIE to reason over or use for transcoding and hence all sites that used no CSS for presentation failed.

Discounting the sites that used no CSS for presentation, the results for those that used Pure CSS and a mixture of CSS and tables for presentation gave encouraging results. The error rate for the sample was only 11%. While we are not claiming that SADIE can successfully transcode 89% of all web sites and make them accessible, this initial result does provide a good basis for future work on SADIE to see how useful it is.

One area that can be researched is what to do with those sites that use a mixture of tables and CSS for presentation. While many of the mixed presentation sites could be transcoded successfully, several of them failed. A reason for this is because of SADIE not being able to separate columns of text. During the evaluation, it was observed that a common approach adopted by these mixed sites was to give the entire table a CSS class value, which SADIE could use, but not give the elements within the cells of the table a Class or ID value. So while SADIE could remove or reorder the table as a whole, the contents within the table were inaccessible to SADIE and so remained in columns. This in turn meant that
the screen reader would be unable to read the text properly and the page was deemed a failure. There are several possibilities to solving this problem. One is for SADIe to reason over the table Class or ID value and apply the results of this to every cell element. For example, if SADIe determines that a table is removable, then remove the entire table. If SADIe determines that the table is High Priority, then treat every cell as High Priority. The contents of the cell could then be removed and arranged in a more accessible way at the top of the page where High Priority elements are placed.

A second approach is for SADIe to reason over the table Class or ID value and apply the results to the columns instead of individual cells. Single column tables could be aligned horizontally, making the content much easier to access than if they were aligned horizontally, like a multi-column table.

Studies would have to be carried out to see if it is possible to determine the best order to place these cells or columns if they were removed from the table. Tables are used in presentation to make sure information is grouped and presented in a certain way. This grouping would have to be maintained in order to make the information that was within the table comprehensible, even if it was displayed in a different, more accessible way.

Further work would also need to be carried out with the evaluation of SADIe. Due to time constraints, no visually impaired users were asked to test SADIe to see if it improved their web experience. While our evaluation criteria was based on well established research into improving accessibility, the results were ultimately just a value judgement that we made as to whether a web page was successfully made more accessible or not. The W3C Web Accessibility Initiative guidelines [67] recommend that visually impaired users should be involved in testing web sites for accessibility. Without their assessment and feedback of SADIe, it is difficult to truly assess the usefulness of SADIe for visually impaired users.

SADIe is a prototype tool to help make web pages more accessible to visually impaired users. While work still needs to be carried out on SADIe, in terms of functionality and evaluation and feedback, the initial results do look promising and indicate that SADIe does have the potential to be of use. People with disabilities have the same right to access information technologies as everybody else. SADIe goes some way to improving access for visually impaired Web users.
Bibliography


[29] BBC. About Betsie.


[34] DANTE. http://dante.man.ac.uk.


[57] *XUL Tutorial*.


[61] Mark McKeown. The joy of web services.


[63] *SOAP Example*. http://www.w3schools.com/soap/soap_example.asp.


[74] Anita W. Huang and Neel Sundaresan. A semantic transcoding system to adapt web services for users with disabilities. ASSETS’00, 2000.


[94] Peter G. Fairweather, Vicki L. Hanson, Sam R. Detweiler, and Ruchard S. Schwerdtfegar. From assistive technology to a web accessibility service. ASSETS’02, 2002.
BIBLIOGRAPHY


[100] Chieko Asakawa and Hironobu Takagi. *An Automatic Web Content Accessibility Compliance Tool for Section 508*.


# Appendix A

## Z-Table

<table>
<thead>
<tr>
<th>z</th>
<th>0.0003</th>
<th>0.0005</th>
<th>0.0007</th>
<th>0.0009</th>
<th>0.0011</th>
<th>0.0013</th>
<th>0.0015</th>
<th>0.0017</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3.9</td>
<td>0.0003</td>
<td>0.0003</td>
<td>0.0003</td>
<td>0.0003</td>
<td>0.0003</td>
<td>0.0003</td>
<td>0.0003</td>
<td>0.0003</td>
</tr>
<tr>
<td>-3.8</td>
<td>0.0007</td>
<td>0.0007</td>
<td>0.0007</td>
<td>0.0007</td>
<td>0.0007</td>
<td>0.0007</td>
<td>0.0007</td>
<td>0.0007</td>
</tr>
<tr>
<td>-3.7</td>
<td>0.0011</td>
<td>0.0011</td>
<td>0.0011</td>
<td>0.0011</td>
<td>0.0011</td>
<td>0.0011</td>
<td>0.0011</td>
<td>0.0011</td>
</tr>
<tr>
<td>-3.6</td>
<td>0.0015</td>
<td>0.0015</td>
<td>0.0015</td>
<td>0.0015</td>
<td>0.0015</td>
<td>0.0015</td>
<td>0.0015</td>
<td>0.0015</td>
</tr>
<tr>
<td>-3.5</td>
<td>0.0023</td>
<td>0.0023</td>
<td>0.0023</td>
<td>0.0023</td>
<td>0.0023</td>
<td>0.0023</td>
<td>0.0023</td>
<td>0.0023</td>
</tr>
<tr>
<td>-3.4</td>
<td>0.0032</td>
<td>0.0032</td>
<td>0.0032</td>
<td>0.0032</td>
<td>0.0032</td>
<td>0.0032</td>
<td>0.0032</td>
<td>0.0032</td>
</tr>
<tr>
<td>-3.3</td>
<td>0.0042</td>
<td>0.0042</td>
<td>0.0042</td>
<td>0.0042</td>
<td>0.0042</td>
<td>0.0042</td>
<td>0.0042</td>
<td>0.0042</td>
</tr>
<tr>
<td>-3.2</td>
<td>0.0053</td>
<td>0.0053</td>
<td>0.0053</td>
<td>0.0053</td>
<td>0.0053</td>
<td>0.0053</td>
<td>0.0053</td>
<td>0.0053</td>
</tr>
<tr>
<td>-3.1</td>
<td>0.0064</td>
<td>0.0064</td>
<td>0.0064</td>
<td>0.0064</td>
<td>0.0064</td>
<td>0.0064</td>
<td>0.0064</td>
<td>0.0064</td>
</tr>
<tr>
<td>-3</td>
<td>0.0077</td>
<td>0.0077</td>
<td>0.0077</td>
<td>0.0077</td>
<td>0.0077</td>
<td>0.0077</td>
<td>0.0077</td>
<td>0.0077</td>
</tr>
<tr>
<td>-2.9</td>
<td>0.0087</td>
<td>0.0087</td>
<td>0.0087</td>
<td>0.0087</td>
<td>0.0087</td>
<td>0.0087</td>
<td>0.0087</td>
<td>0.0087</td>
</tr>
<tr>
<td>-2.8</td>
<td>0.0097</td>
<td>0.0097</td>
<td>0.0097</td>
<td>0.0097</td>
<td>0.0097</td>
<td>0.0097</td>
<td>0.0097</td>
<td>0.0097</td>
</tr>
<tr>
<td>-2.7</td>
<td>0.0108</td>
<td>0.0108</td>
<td>0.0108</td>
<td>0.0108</td>
<td>0.0108</td>
<td>0.0108</td>
<td>0.0108</td>
<td>0.0108</td>
</tr>
<tr>
<td>-2.6</td>
<td>0.0118</td>
<td>0.0118</td>
<td>0.0118</td>
<td>0.0118</td>
<td>0.0118</td>
<td>0.0118</td>
<td>0.0118</td>
<td>0.0118</td>
</tr>
<tr>
<td>-2.5</td>
<td>0.0129</td>
<td>0.0129</td>
<td>0.0129</td>
<td>0.0129</td>
<td>0.0129</td>
<td>0.0129</td>
<td>0.0129</td>
<td>0.0129</td>
</tr>
<tr>
<td>-2.4</td>
<td>0.0140</td>
<td>0.0140</td>
<td>0.0140</td>
<td>0.0140</td>
<td>0.0140</td>
<td>0.0140</td>
<td>0.0140</td>
<td>0.0140</td>
</tr>
<tr>
<td>-2.3</td>
<td>0.0151</td>
<td>0.0151</td>
<td>0.0151</td>
<td>0.0151</td>
<td>0.0151</td>
<td>0.0151</td>
<td>0.0151</td>
<td>0.0151</td>
</tr>
<tr>
<td>-2.2</td>
<td>0.0162</td>
<td>0.0162</td>
<td>0.0162</td>
<td>0.0162</td>
<td>0.0162</td>
<td>0.0162</td>
<td>0.0162</td>
<td>0.0162</td>
</tr>
<tr>
<td>-2.1</td>
<td>0.0173</td>
<td>0.0173</td>
<td>0.0173</td>
<td>0.0173</td>
<td>0.0173</td>
<td>0.0173</td>
<td>0.0173</td>
<td>0.0173</td>
</tr>
<tr>
<td>-2</td>
<td>0.0184</td>
<td>0.0184</td>
<td>0.0184</td>
<td>0.0184</td>
<td>0.0184</td>
<td>0.0184</td>
<td>0.0184</td>
<td>0.0184</td>
</tr>
<tr>
<td>-1.9</td>
<td>0.0205</td>
<td>0.0205</td>
<td>0.0205</td>
<td>0.0205</td>
<td>0.0205</td>
<td>0.0205</td>
<td>0.0205</td>
<td>0.0205</td>
</tr>
<tr>
<td>-1.8</td>
<td>0.0226</td>
<td>0.0226</td>
<td>0.0226</td>
<td>0.0226</td>
<td>0.0226</td>
<td>0.0226</td>
<td>0.0226</td>
<td>0.0226</td>
</tr>
<tr>
<td>-1.7</td>
<td>0.0247</td>
<td>0.0247</td>
<td>0.0247</td>
<td>0.0247</td>
<td>0.0247</td>
<td>0.0247</td>
<td>0.0247</td>
<td>0.0247</td>
</tr>
<tr>
<td>-1.6</td>
<td>0.0267</td>
<td>0.0267</td>
<td>0.0267</td>
<td>0.0267</td>
<td>0.0267</td>
<td>0.0267</td>
<td>0.0267</td>
<td>0.0267</td>
</tr>
<tr>
<td>-1.5</td>
<td>0.0288</td>
<td>0.0288</td>
<td>0.0288</td>
<td>0.0288</td>
<td>0.0288</td>
<td>0.0288</td>
<td>0.0288</td>
<td>0.0288</td>
</tr>
<tr>
<td>-1.4</td>
<td>0.0309</td>
<td>0.0309</td>
<td>0.0309</td>
<td>0.0309</td>
<td>0.0309</td>
<td>0.0309</td>
<td>0.0309</td>
<td>0.0309</td>
</tr>
<tr>
<td>-1.3</td>
<td>0.0330</td>
<td>0.0330</td>
<td>0.0330</td>
<td>0.0330</td>
<td>0.0330</td>
<td>0.0330</td>
<td>0.0330</td>
<td>0.0330</td>
</tr>
<tr>
<td>-1.2</td>
<td>0.0351</td>
<td>0.0351</td>
<td>0.0351</td>
<td>0.0351</td>
<td>0.0351</td>
<td>0.0351</td>
<td>0.0351</td>
<td>0.0351</td>
</tr>
<tr>
<td>-1.1</td>
<td>0.0372</td>
<td>0.0372</td>
<td>0.0372</td>
<td>0.0372</td>
<td>0.0372</td>
<td>0.0372</td>
<td>0.0372</td>
<td>0.0372</td>
</tr>
<tr>
<td>-1</td>
<td>0.0393</td>
<td>0.0393</td>
<td>0.0393</td>
<td>0.0393</td>
<td>0.0393</td>
<td>0.0393</td>
<td>0.0393</td>
<td>0.0393</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0414</td>
<td>0.0414</td>
<td>0.0414</td>
<td>0.0414</td>
<td>0.0414</td>
<td>0.0414</td>
<td>0.0414</td>
<td>0.0414</td>
</tr>
</tbody>
</table>

Table A.1: Z-Table For the Normal Distribution

87