MULTLILIGUAL SELF-VOICING BROWSER MODEL FOR ETHIOPC SCRIPTS

Mintesinot Fikre

A THESIS SUBMITTED TO
THE SCHOOL OF GRADUATE STUDIES OF THE ADDIS ABABA UNIVERSITY IN PARTIAL FULFILLMENT FOR THE DEGREE OF MASTERS OF SCIENCE IN COMPUTER SCIENCE

April 17, 2014

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
COLLEGE OF NATURAL SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

MULTLILIGUAL SELF-VOICING BROWSER MODEL FOR ETHIOPC SCRIPTS

Mintesinot Fikre

ADVISOR: Mulugeta Libsie (PhD)

APPROVED BY

EXAMINING BOARD:

1. Dr. Mulugeta Libsie, Advisor _____
2. Dr. Fekade Getahun, Examiner _____
3. Ashenafi Kassahun**,** Chair Person _____

ACKNOWLEDGEMENTS

First and foremost, I would like to thank God for making every step of my way to be smoother and safe. If it has not been from him, I would have not been here.

It is difficult to overstate my deepest gratitude to my advisor Dr. Mulugeta Libsie. He always make me feel easy when I leave his office after having a discussion on my progress. With his enthusiasm, inspiration, and great efforts to explain things clearly and simply, he helped to make thesis fun for me. Throughout my thesis writing period, he provided encouragement, sound advice, good teaching, good company, and lots of good ideas. I would also like to thank Dr. Fekade Getahun for being instrumental in shaping my ideas and helping me articulate the thesis work to come out as a worthy one.

I am indebted to my fellow classmates and friends for making my class time an easy and memorable one. I am thankful to my team work members during class time "ቱዲ" and "ዮኒ", I have learned a lot both academically and from your friendship. ሸሜ (አበባው አበባው አበባው…) you are great, kind, and off course very funny friend, your encouragement and technical advice was a great help.

Lastly, and most importantly, I wish to thank my mom "ጥርዬ" and my sister "ሴሪ". Your prayer, support, and encouragement indeed is fruitful. My dad, I wish you were here today to see your advice and guidance came true you thought me to pursue in academics during my childhood ages, I kept your word! To them I dedicate this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

AmhTTS-Amharic Text to Speech

API-Application Programming Interface

AT-Assistive Technology

ATCB-Adaptive Technology Center for the Blind

BITV-Barrierefreie Informationstechnik-Verordnung

CSS-Cascading Style Sheet

DOM-Document Object Model

DRES-Disability Resources and Educational Services

DSP-Digital Signal Processing

FAE -Functional Accessibility Evaluator

GTP-Grapheme-to-Phoneme

GUI-Graphical User Interface

HPR-Home Page Reader

HTML-Hyper Text Markup Language

IAPB-International Agency for the Prevention of Blindness

IEF-International Eye Foundation

IE-Internet Explorer

ITU-International Telecommunication Union

IPA-International Phonetic Association

LMA-Log Magnitude Approximation

LMA-Log Magnitude Approximation

MSAA-Microsoft Active Accessibility

NLP-Natural Language Processing

PHP-Hypertext Preprocessor

SVG-Scalable Vector Graphics

TTS -Text To Speech

UI-User Interface

UMIST-University of Massachusetts Information System Technology

UNESCO-United Nations Educational, Scientific and Cultural Organization

URL-Uniform Resource Locator

W3C-World Wide Web Consortium

WAI ARIA- Web Accessibility Initiative Accessible Rich Internet Applications

WAI -Web Accessibility Initiative

WCAG -Web Content Accessibility Guidelines

WHO-World Health Organization

WWW-World Wide Web

XHTML-Extensible Hypertext Mark-up Language

XML-Extensible Markup Language

XSS-Cross-Site Scripting

ABSTRACT

This master's thesis describes the design of multilingual self-voicing browser model for Ethiopic scripts. We show that efficient multilingual self-voicing browser model can be constructed for Ethiopic scripts by a purification, parsing, transcoding, and text-to-speech techniques and approaches. Self-voicing browser model research have been an active research for English and other non-Ethiopic languages for quite long time. Though, there has not been any research work done before to design and implement self-voicing browser for websites equipped with Ethiopic scripts. There are various approaches and techniques proposed for English and other languages in designing self-voicing browser. We have adopted the approach introduced by the research work of WebAnywhere for English language and extended the approach to be a multilingual. Multilingual self-voicing browser model is a web based web browsing application that allows visually impaired people access the web from anywhere. The model designed does not require any additional software to be installed on the web browsing machine other than sound card and web enabled machine. In addition, the model is not designed for specific language rather to support multiple language bases. The webpage coded in none accordance with W3C compliance was found to have crucial effect in the performance of our model. Evaluation of the model showed that webpages purified for W3C compliance were not accurately purified to exhaustively parse and generate the required content. We have learned that the webpage purification accuracy has a direct impact on the parser and text transcoder to traverse and generate all the required information. The evaluation of our model, being the first multilingual self-voicing browser model for Ethiopic scripts, has shown promising performance. The HTML purifier library improved the webpage W3C coding compliance of standard by 44% precision accuracy. The parser performance in traversing and generating DOM tree is 96% precision accuracy for webpages their HTML coding compliance improved by HTML purifier. The text transcoder achieves a precision accuracy of 100% by transcoding all parser generated DOM tree. Though, this doesn't mean that all the required content in the webpage has been successfully transcoded as the HTML purifier only cleaned the webpage at accuracy of 44% which we do not able to get 56% of the webpage content that needs to be transcoded. In general, the approaches and libraries, we have used to design our model shows a meaningful performance considering an initial research work.

KEYWORDS: Self-voicing browser, HTML Purifier, Parser, text transcoder, Text to Speech, Website Accessibility, Website Usability

CHAPTER ONE-INTRODUCTION

1.1 General Background

Internet phenomenal growth has changed our life style on the way we perform our day to day activities of sharing information and working collaboratively. It also gives an easy access to information which is known as the power of knowledge. The primary sources of this powerful knowledge are websites which are built on the Internet. The Web is fundamentally designed to work for all people, regardless of hardware, software, language, culture, location, or physical or mental ability. In contrast of old days printed material as a means of information as a resource, the web has changed the means of perceiving and level of access by converting the information automatically in digital format to improve its use. The information content that resides on the web in digital format is able to be represented in audible or refreshable braille so that low vision and visually impaired people can be benefited from the enormous potential it offers.

People with very low vision and visually impaired can only properly utilize information available on the web if the websites are designed to be compatible with the various assistive technologies and such websites are said to be accessible websites. Accessible websites are those that are sufficiently flexible to be used by assistive technologies. An accessible website is very similar to an accessible building which offers curb cuts, ramps, and elevators to allow a person with disabilities to enter and navigate through the building with ease. An accessible website offers similar functionality [28].

Accessibility is not just a high-level theoretical goal. Currently, there are guidelines that web developers can follow so that their websites can be accessible. For instance, the Web Accessibility Initiative (WAI) provides guidelines; called the Web Content Accessibility Guidelines (WCAG) to help developers make their websites accessible [4]. The United States Government offers similar guidelines to web developers, which are included in the Section 508 initiative [7].

All the importance and high potential of the web as the main source of information still has frustrating web browsing experience to those who benefited it most. W3C (World Wide Web Consortium) sets standards and guidelines that web developers need to adhere so that information can be accessed and used effectively for users with diverse abilities. Despite the effort to follow these guidelines, web developers are either ill-equipped or not familiar with the problem that makes web content inaccessible. To fully utilize the web potential, content such as images in the

layout and color is not accessible for low vision and visually impaired web users. Visually impaired people surfing on the web using assistive technologies such as screen reader by searching required content from long linear stream of a complex page has a frustrating experience.

1.2 Statement of the Problem

In the literature, web accessibility generally refers to the application of technical solutions to the design of a website in order to render more accessible to all users, in particular low vision and visually impaired users of assistive technologies. Technical solutions refer to the correct application of properly validated coding such as Hypertext Mark-up Language (HTML) or Extensible Hypertext Mark-up Language (XHTML), which define the structure of the content, together with the use of Cascading Style Sheets (CSS), which define the way the content is displayed [6]. Web accessibility for low vision and visually impaired users is an important goal and at the same time a challenging problem for web content developers and designers. The Web is not built in mind that it will be utilized by visually impaired people. Only a small fraction of major corporate websites are fully accessible to visually impaired users, let alone those of smaller organizations or individuals [1]. However, millions of visually impaired people surf the web every day, and Internet use by those with disabilities grows at twice the rate of the non-disabled [2]. The accessibility problems get much worse when we see contents available on websites of Amharic and other Ethiopian scripts such as Geez and Tigrigna which currently available assistive technology lacks supporting on the making of the information accessible to low vision and visually impaired users. Low vision and visually impaired individuals commonly surf the web using assistive technologies that convert the text of a webpage into synthesized speech. Efficient access to web content remains elusive for individuals accessing the web using assistive technology. Previous efforts to improve web accessibility have focused on developer awareness, technological improvement, and legislation, but these approaches have left remaining concerns as it has only been focused on functionality of websites than usability.

1.3 Motivation

There is enormous amount of information on the World Wide Web (WWW) at the fingertips of anyone with access to the Internet. The vast proportion of employment, education and daily life activities require access to electronic information and mainly through websites. In addition, the

web provides an immense information to low vision and visually impaired despite printed material as most web information can be automatically converted to an “accessible” form.

There are a number of persuasive reasons to develop websites that are technically accessible and usable for people with low vision and visually impaired. According to the to the research conducted by WHO and the International Agency for the Prevention of Blindness (IAPB) on the area of disability in general reveals that there are more than a billion people worldwide, out of this number 80 per cent of the visually impaired population lives in the developing countries of Asia, Africa and Latin America [29]. The 1984 Population and Housing Census of Ethiopia revealed that there were 1,244,881 persons with disabilities in Ethiopia. The census indicated visual impairment as the highest (42.7 percent). According to the 1994 Population and Housing Census in Ethiopia there were 319,194 visually impaired persons where 117,739 (36.89 percent) are fully visionless and 201,455 (63.11 percent) are partially sighted. The census results for Addis Ababa indicate that there were 45,936 persons with disabilities in the city. Of these 12,888(28.0 percent) are persons with visual impairment [31]. Based on the survey conducted by national survey on blindness, low vision and Trachoma in nine regions which are Tigray, Amhara, Oromia, SNNP, Somali, Gambella, Benishangul-Gumuz, Harari, and Afar and two special city administrations which are Addis Ababa and Dire DawaEthiopia in 2006, there are 1.2 million visually impaired people [35].

According to business perspective, the substantial consumer segment isn’t something that should be ignored and the discretionary income these people generate should not be underestimated as well.

One research conducted on USA shows that out of the 54 million Americans with a disability, 4 in 10 are online [5]. These users spend more time logged on and surfing the Internet than nondisabled users. On average, they spend 20 hours per week online. In addition, they report more positive feelings about their interactions. Improving accessibility improves usability for all users and it is morally the right thing to do. The Internet service bandwidth and coverage expansion in our country will play a major role in access to the immense information on the web and those visually impaired people will be benefited in spending their time surfing on the web as we observed same situation in America and other developed countries. In addition, the visually impaired people should be included in the decision-making processes of development planning designed for them so that services could be based on their own needs.

The main purpose of this study is to enable visually impaired people access websites equipped with Ethiopic scripts by designing a multilingual self-voicing browser model that renders interactive speech dialogs from standard HTML webpages of Ethiopic scripts. This research will set a benchmark to information accessibility of Ethiopic scripts to low vision and visually impaired users already available on the web and future development of websites to ensure web fundamental goal is met.

1.4 Objective

General Objective

The general objective of this research is to develop a generic multilingual self-voicing browser model with intent to enable access of websites that have content available in Ethiopic scripts to the low vision and visually impaired community.

Specific Objectives

The following are the specific objectives to achieve the general objective:

- Conducts literature review on web accessibility issues towards low vision and visually impaired users.
- Conduct literature works on Ethiopic Text To Speech (TTS).
- Review related work on self-voicing browser model for English and other languages with the aim to adopt best practices for Ethiopic scripts.
- Review TTS library engine works done for Amharic language from existing works to integrate into the model.
- Review webpage purification, web parsing and transcoding approaches and integrating them with TTS libraries.
- Review IPA (International Phonetic Association) phonetic representation to use it for TTS as a voicing engine in our model.
- Design and develop a generic multilingual self-voicing browser model for Ethiopic scripts.
- Evaluate the performance of each component in the model.

1.5 Scope and Limitations

This research will be undertaken with the aim to model a generic multilingual self-voicing browser model for Ethiopic scripts. The model will set an initial step towards developing a multilingual self-voicing browser model to render an aural output from standard HTML webpages of Ethiopic scripts. The model will comprise of webpage purifier, web parser engine, text transcoder, and multilingual TTS library of Ethiopic scripts. This research will not include TTS library modeling and development.

1.6 Research Methodology

Literature Review

We will make deeper literature review on web development practice and web accessibility standard and methods towards low vision and visually impaired people to have a crisp understanding in developing multilingual self-voicing browser model for Ethiopic scripts.

Tools

In researching and designing multilingual self-voicing browser model, Hypertext Preprocessor (PHP) as server-side scripting programming language and JQuery client-side scripting language will be employed as major development tools for modeling the prototype. We will use Application Programming Interface (API) libraries developed in PHP for designing our model of HTML purifier, parser, and text transcoder, IPA based text conversion by customizing each of them for seamless integration with our parsing algorithm and transcoding to enhance the performance of the model.

Modeling

We will build a prototype to test and evaluate our integrated model of HTML purifier, parser engine, IPA based text transcoder, and TTS library engine.

Testing

We will make objective testing evaluation for our model. Testing will be done to check the performance of our model on Ethiopic script equipped websites. We will test each site with A-Checker and Functional Accessibility Evaluator (FAE) tool to evaluate our model for web accessibility against Web Content Accessibility Guidelines (WCAG 1.0), Section 508, Stanca

Act, and Barrierefreie Informationstechnik-Verordnung (BITV). We will also make human judjment testing as of to date as no website evaluation tool has been developed that can completely replace a human being. This is because with present technology it is difficult to emulate human attributes such as common sense.

1.7 Application of Results

The multilingual self-voicing browser model will be designed to allow visually impaired people to surf the web for websites equipped with Ethiopic scripts. The model offers individuals the ability to compensate for physical or functional limitations, to access knowledge by adapting digital media to the nature of their disabilities, and to enhance their social and economic integration in communities by enlarging the scope of activities available to them. In addition, it will create a platform for visually impaired people to harness the substantial amounts of written material previously inaccessible that has been converted to electronic format available through the web. The result will also allow to read news, perform transactions online, such as shopping or filling out application forms, which have become very popular over the last number of years. Developing self-voicing bowser model can reduce their dependence on sighted assistance to complete daily tasks.

Above all for websites equipped with Ethiopic scripts there has not been any work done yet to make the vast information accessible to the visually impaired people so this work will have meaningful contribution.

1.8 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 discusses the barriers and opportunities in web accessibility and usability, visually impaired people way of interaction with web as a literature review. In this Chapter, we can crisply see techniques, methods, and accessibility guidelines on how we can make a website accessible and usable for visually impaired people. Chapter 3 is devoted to discuss related works done in the area of web accessibility and self-voicing browser model towards visually impaired people. Chapter 4 extensively presents the design of the self-voicing browser model components and architecture. We have discussed the architecture, components functional description and interaction among them in detail. Chapter 5 is devoted to discussing the main implementation of each component in the model. The algorithms, techniques and methods used in how the model has been successfully developed are discussed. Chapter 6 is

devoted to present the evaluation of the model and the results as well as the limitations of the system. Chapter 7 concludes the thesis by outlining the benefits obtained from the research work. It also shows some research directions that can be accomplished in developing a full-fledged self-voicing browser model for Ethiopic scripts.

CHAPTER TWO-LITERATURE OVERVIEW

In this Chapter, we will review low vision and visually impaired users' web accessibility and usability experience. First, we will review the options of web accessibility. Then, we will review studies and researches conducted for low vision and visually impaired people's interaction to a computer and the web for the purpose of learning their needs and problems that are being used as an important input for the design and development of self-voicing browser model. We will also discuss about works done in the areas of self-voicing applications and assistive technologies. Lastly, we will also review works done in the area of Text To Speech (TTS) processing for Amharic and other Ethiopic scripts.

2.1 Introduction

Information sources through web or printed material are at ease access but not for those visually impaired users. Traditionally, visually impaired people have had only limited means of accessing printed material like newspapers and magazines. Braille is the most famous access method, but only a tiny proportion of visually impaired people can read braille. Recent years have seen the wider adoption of audio recordings, but like braille these suffer from lack of immediacy. If we want the news today, not to wait a week for it to be translated, a visually impaired user is usually reliant on sighted people, often volunteers, to produce the material. This reliance and the higher costs of producing alternative format materials such as audiotapes necessarily reduce the material available. This is a poor comparison with what is available to sighted users and their choice of material.

In general, the realization of personal computing becomes affordable and particularly the Internet promises an incredible improvement in access to written materials. With a personal computer, some easily available technology, and a web browser, visually impaired users are no longer restricted to tapes sent through the post or the passive technology to obtain information. When visually impaired users want to access different source of printed material, the material may not be available in an alternative format that they can use it. Relying on what other people choose to translate for their benefit reduces their choice and freedom. Since sighted people use the Internet to get any kind of available information with ease, visually impaired people equally need the same opportunity to surf the web. But accessing information from the web has not been fully achieved

for even languages like English. The problem becomes much deeper for Amharic and other Ethiopic scripts as no work has been tried so far [19].

Accessibility and usability experience of websites by low vision and visually impaired users has been a challenging area for both web and content developers. A number of researches have been conducted to identify and provide solutions for improved web accessibility and usability experience. Based on the research conducted on website usability [19], websites that have been developed in compliance with accessibility laws and guidelines are necessary but not sufficient enough for low vision and visually impaired people to access what they need on the web. It requires more work on studying web usability to make it an integral part of web design. The W3C has published guidelines known as the WAI in addition to the legal requirements.

The area of web accessibility has been a major issue for low vision and visually impaired users to equally participate on the web as of the sighted peers on the full potential of the web. The expanding volume of Amharic and other major Ethiopic script information available on the web is accessible to the sighted users and lacks support of accessibility to the low vision and visually impaired community. A number of researchers and various organizations have been conducting accessibility and usability testing on people with disabilities, specifically low vision and visually impaired users to understand the relationship between accessibility and usability, how low vision and visually impaired users work with websites, develop research guidelines for accessibility and usability, and assess the usability of specific websites. The accessibility of web based information can be improved in two principal ways: through the use of access technology and through adopting good practice in interface design. Both are of equal importance: provision of assistive equipment (adaptive, enabling, or access technology) will enable a visually impaired user to access on screen information receiving output in a way that is appropriate to her or his needs. However, in addition to this, the information provided on screen must be presented in a way that can be interpreted by any kind of access technology. This is what is referred to as “accessible Web design,” “design for all,” or “universal design” [7].

From various researches conducted on extensive usability and by expanding usability a bit more broadly is often referred to as users experience on how visually impaired people conceptualize the web. The result shows that sighted users were six times more successful than users of screen readers at accomplishing given tasks, and three times more successful than users of screen

magnification [6]. These numbers clearly demonstrate how poorly the web is designed for people who are low vision or visually impaired, and how far we have to go. Low vision and visually impaired people have also problems working with the interactivity of sites, particularly when sites deviated too much from standard Internet conventions. Since low vision and visually impaired people rely heavily on memorization to aid in their navigation, it is important to devise a convention that web developers and designers can follow. Deviating from these norms means users have to learn a new routine for each site which results in making their web accessibility and usability experience more challenging. Many people revisit sites because "they know it," because they "know it is accessible." They revisit sites because they are familiar with their layout and therefore can navigate more easily. Thus, once a site has set up a system that works well, it should be kept; however; changing the overall appearance, structure, and navigation will bring issue of usability.

Since June 2001, U. S. federal Web sites must comply with Section 508 of the Rehabilitation Act (29 U.S.C. §794.d). This law requires that agencies provide access to electronic information to people with disabilities. Section 508 identifies 16 specific standards for Web site accessibility. Meeting the required accessibility standards does not, however, necessarily mean that a website is usable for people with disabilities. If a website is not usable, it is not really accessible, even if it has all the elements required by the law [8]. Since HTML has primarily been developed for visual rendering (and it is conventionally used in this way), there will be elements which are not amenable to speech rendering (e.g., image and maps), as well as web design practice which make speech rendering more awkward than visual rendering like the use of frames [9].

Accessibility is not just a high-level theoretical goal. Currently, there are guidelines that web developers can follow so that their websites can be accessible. For instance, the WAI provides guidelines, called the Web Content Accessibility Guidelines (WCAG), to help developers make their web sites accessible. The United States government offers similar guidelines to web developers, which are included in the Section 508 initiative [10]. In addition, automated software tools are available to help find accessibility flaws in web sites before the sites are publicly posted. These software tools include Bobby, RAMP, InFocus, and A-Prompt. In fact, many governments make web accessibility a requirement for government information on the web. The United States, England, Canada, Portugal, and Australia require some types of government information to be accessible [11].

Solutions of web accessibility for visually impaired people fall into a number of categories; voice browsers, reliance on a conventional web browser and a screen reader; utilizing the accessibility features of HTML and existing web clients; using transcoding proxy servers to convert webpage HTML into a more accessible format; and using a dedicated web browser. However, it should be considered that it is not an exhaustive list.

Voice Browsers

The concept of voice browser is introduced by W3C as a note for discussion only and the note has been extended and groups are formed like the voice browser interoperation. The voice browser note describes feature needed for realization of voice browsers [42]. A voice browser by a voice browser interoperation group provides the means for people to use their voice to interact with appropriately designed applications. Voice browsers use speech synthesis and prerecorded material to present the contents of webpages. A variety of aural effects can be used to give different emphasis to headings, hypertext links, and list items etc. Users interact with voice browsers by spoken command or by pressing a button on a keypad. Some commands interrupt the browser. For instance to request a list of hypertext links occurring in the current section of the document. Other commands are given when the browser prompts for input, for example, to select an option from a menu in a form [43]. To increase the robustness of speech recognition, voice browsers take advantage of contextual clues provided by the author. This allows the recognition engine to focus on likely utterances, improving the chances of a correct match. Work is needed to specify how such contextual clues are represented. Speech synthesis is driven by dictionaries, falling back for unknown words on rules for regular pronunciation. High quality speech synthesis is possible if the author can extend the dictionary resident in the browser. Speech synthesis is not as good as having an actor read the text. Content providers will inevitably want to provide prerecorded content for some parts of Web pages. Prerecorded content is analogous to the use of images in visual Web pages. The same need for textual fallbacks applies for printing, searching and access by users with disabilities.

Navigation wise alternatives to using a mouse click, for an application using a cellular phone, it would be cumbersome to have to take the handset away from your ear to press a button on the keypad. It therefore makes sense to support voice input for navigation as an alternative to keyboard input. At its simplest the user could speak the word "follow" when she hears a hypertext link she

wishes to follow. The user could also interrupt the browser to request a short list of the relevant links. Voice browsers will allow users to move between form fields and to enter and review field values, using either the keyboard or voice input. Authors must be able to specify what spoken phrases should be used for the selection of links, radio buttons, check boxes, image buttons, submit buttons, and selection lists as key access is already provided by the access key attribute in HTML 4[42].

The other main feature of voice browser is the use of aural style sheets. Authors want control over how the document is rendered. Aural style sheets is part of CSS2 specification provide a basis for controlling a range of features, including volume, rate, pitch, direction, suppressing output for specific elements, spelling out text letter by letter, speech fonts (male/female, adult/child etc.) inserted text before and after element content Sound effects before, during and after elements, including music or prerecorded sounds [42].

Conventional Web Browser and Screen Reader

The web browser market is dominated by Internet Explorer (IE), which holds a 58.21% share [30]. It is therefore the defacto standard for web clients, and web authors frequently write HTML and DTHML code targeted at IE. Using IE and a screen reader or magnifier guarantees that a maximum of websites will work. The problems with this approach are the inaccessibility of content displayed by the browser and the complexity of the user interface. However, progress has been made by screen reader developers, notably Freedom Scientific's JAWS, in supporting IE and by extension in vast majority of web browsers [23].

HTML Accessibility

The second approach takes advantage of the principle of HTML, separating content and presentation, and the native abilities of clients to present content in a way desirable to the user. Web clients permit users to define their own presentation preferences, for example using a particular mix of colors (yellow on black is preferred by many visually-impaired people), fonts (Tiresias is designed to be very legible) and font sizes. Clients can also choose to ignore presentation dictates from webpages, stripping out decorative and confusing background images or preventing text from blinking (harmful to users with epilepsy). The Mozilla browser allows the user to turn on a caret, overcoming the normal web browser canvas problem by providing a means to indicate to a screen reader on current content of interest. The problems with these approaches

are that they fail to address a range of problems related to overly-complex interfaces like tables and page layout which are generally still preserved, so the user must still search over the page for content of interest and the needs of users without any degree of functional vision. The other practical problem is that users are required to specify their user preferences within the client, which is not common user behavior and may not be possible in the user's environment; for example, where the user is on a different computer or where user preferences are locked by network policy [23].

Transcoding Proxy Server

The third approach places the solution between the author and the client by running requested HTML pages through a transcoding proxy server. Requests for webpages from servers are made not to the servers themselves but to an intermediate server, a proxy, which fetches the page, converts it according to a set of rules, and returns the converted page to the requesting client. This process is employed for users of limited browsing devices such as mobile telephones which cannot handle fully featured webpages and rely on proxy "gateways" to reduce the standard webpages into a limited format supported by the telephone. Visually-impaired users can use the same approach; the proxy can be configured to alter the HTML document to provide the font, font size, color and other settings desired by the user in much the same way as the use of the accessibility features of a client. The advantage is that these can be set remotely, so the client itself need not be amended by the user. The disadvantages with this approach relate to the second-hand nature of the HTML document transmission. Page features, such as client redirects, may not be supported by the proxy, and many websites assume the use of a client directly and provide functionality based on this assumption; for example, the use of cookies to track users and provide password-authenticated services to them. Finally, the processing performed by the proxy server requires the server to have full access to the content of the HTML document, which means that secure transmission protocols used in Internet commerce such as HTTPS are unusable [23].

Dedicated Web Browser

The final approach is to use a dedicated web browser specifically designed for visually-impaired people. There are two schemes employed: the first, exemplified by the Home Page Reader from IBM, is a self-voicing application that provides a complete audio interface to webpages. The second is to render the content of a webpage as a text only flat document and permit the user to

access this accessible content using their normal assistive technology, typically a screen reader. This scheme is demonstrated by Webwizard from Baum and WebFormator from Frank Audiodata. Developing a dedicated web browser affords the maximum flexibility in approach, but requires the developer to take more responsibility for the presentation of web content. Although in theory a non-visual web browser is just as standard as a visual one presenting marked-up HTML, in practice the visual bias of the web means that alternative applications have to focus on providing access to resources designed for the sighted. The greater flexibility in approach has led to a number of different products which are worthy of examination [23].

How do visually impaired users access a computer and the web?

The important constraint on the use of computers by low vision and visually impaired users is that they rely on hearing, rather than sight. The main problem of this can be seen from two major constraints. First, they are constrained into examining one thing at a time and in order of not their own making; they do not know the structure of things before they explore them. This includes the problem with unfamiliar, rich, and new interfaces. Second, they have to listen to a large amount of text to give them the same amount of information as a sighted user might be able to gain in a quick glance. Sighted users might be able to glance through a large document, scanning the chapter and paragraph headings for a keyword or phrase, because they can see the headings instantly distinct from the body text and what words they contain. Low vision and visually impaired users, even if they can jump from heading to heading, have to wait for the slower Assistive Technology (AT) like self-voicing applications or screen reader to speak the heading, setting it to read as fast as possible might seem more sensible [19].

These two constraints, fixed order of access and time to obtain information, means that interfaces that rely on hearing must comply with a principle of maximum output in minimum speech. This greatly changes usability as superfluous information is not just a distraction, as a page with lots of links might be for a sighted user, but a real barrier to using the interface. Visually impaired users must not be asked to use a complex interface with many options. If a user misses some output, it will need to be read out again, so an explicit way to repeat things is required. Most importantly, users need control over what is being said; sighted users can move their gaze wherever they want whenever they want, and visually impaired users need some similar control of the focus. Non-

visual interfaces need to provide means to navigate through the document, stop, go back, skip items, repeat and explore the text available [19].

Once a webpage meets a certain level of accessibility criteria, visually impaired people can navigate through and gain access to the relevant information. This is achievable using either a self-voicing browser or with screen-reading software interfacing with a mainstream visual browser [21]. Whether using a screen reader or a self-voicing browser, the use of the sense of hearing rather than vision has great implications for the design of the interface to make the websites usability experience easy. The visual sense, or visual modality, has an enormous capacity for communicating information quickly and easily. If we look at an application on our computer display, we will immediately notice the menus, icons, buttons and other interface controls arrayed on our screen. Each represents a function that is available to us, and a quick glance allows us to locate the function we want and immediately activate it with the mouse. If we see the application like a word processor, we can go straight from reading the text of a document to any one of the functions offered by the interface. If we consider tasks to find the print function, we will have to start at the top left-hand corner of the screen and go through each control in turn and wait until its function is described to us, until we find the function we require. As a result, experienced for low vision and visually impaired computer users will not rely on navigating through menus for every function. They will utilize shortcut keys, such as "CTRL+P" to print a document, develop combinations of keystrokes to complete their most common tasks, and learn the location of commonly-used functions in menus and applications. This requires, however, a consistent user interface, where shortcut keys and keystroke combinations can be relied on to perform the same function each time menu items are always located in the same place.

Interaction Modalities

According to the research work in [21] braille and speech output are the two primary media in which visually impaired people can gain access to electronically stored documents. Braille is a tactile method of encoding written material dating back to the early 19th century. No attempt is made to mimic the shape of printed letters. Instead, characters are fashioned using up to six raised dots, positioned in two vertical columns of three. The dots are numbered in sequence from one to six. The top left position is known as dot 1, while dot 6 is situated on the bottom right. Characters are defined by raising one or more dots in various combinations. For example, ‘a’ is represented

by dot one, and ‘g’ contains dots one, two, four and five. In terms of computer interaction, access can be gained through a specialized piece of hardware known as a braille display, where up to 80 characters are often available.

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

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

In addition media pose brings other difficulties for the presentation of contextual information provided by visual cues. For example, changes in font, or emphasis. Announcing the presence of such content in the vocal output would increase dramatically the verbosity of the presentation. Therefore, alternative voices or changes in voice parameters could be used to portray this information. However, with braille the capacity for presenting such data is extremely limited. There is not much scope for defining new braille characters to denote contextual cues. For these reasons, the use of sound cues to complement the primary method of output is often employed.

Navigation and Browsing

Listening to an entire hypertext document of a web through a TTS could be described as being similar to listening to a pre-recorded spoken version of the content. The approaches applied to the presentation of web based information include “audio enriched links system”, and the “Hearsay

system" for browsing hypertext documents through audio. The audio enriched links mechanism provides a spoken preview summary of a linked webpage, before the link is followed by the user. The page summary is comprised of its title, its relation to the current page, statistics about its content, and some highlights from its content. The Hearsay system attempts to automatically partition web documents through tightly coupled structural and semantic analysis. It transforms raw HTML documents into semantic structures to facilitate audio browsing [21].

2.2 Adaptive Technology

Adaptive technology is a broad term often used to describe both the products and services for people with special needs. It enhances the vocation, recreation, education, and independence of the user. It is any item, piece of equipment, or product system, whether acquired commercially, off the shelf, modified, or customized, that is used to increase, maintain, or improve functional capabilities of individuals with disabilities. The next thing that comes to mind is the question of whom, how, where, and when does one require and should be entitled to adaptive or assistive technology as well as to what extent and at what cost should it be available [2].

Adaptive technology can provide equality between visually impaired individuals and their sighted peers within the emerging information society. With the aid of the appropriate technological devices, visually impaired persons can independently access, process, store, and transmit the same information handled by sighted people. Both use computers to manipulate this information. The only difference lies in the form in which the information is displayed.

There are essentially five methods of output that can render computers and printed materials accessible for individuals who are low vision and visually impaired: s*creen reader, braille printer, reading device, electronic braille displays, text magnification, and self-voicing* [2].

2.2.1 Screen Reader

A Screen Reader is used to convert text, words, or numbers from a computer document (e.g., a word processed document, a spreadsheet, or a webpage) into audible speech spoken through the computer speaker. The person with visual impairment can access computers with the help of speech output to use any word processor application to write letters, school assignments or any other writing. It doesn't require a video monitor and interpretation is represented to the user with TTS, sound icons, or a braille output device. Screen readers are a form of AT potentially useful to

people who are low vision, visually impaired, illiterate or learning disabled, often in combination with other AT, such as screen magnifiers.

There are few screen reader applications on the market that are commercial or open source. The choice of a screen reader is dictated by many factors, including platform, cost as it requires hundreds of U.S. dollars to upgrade a screen reader, and the role of organizations like charities, schools, and employers. Screen reader choice is contentious: differing priorities and strong preferences are common. The leading and widely used operating systems such as Microsoft Windows, Mac OS, and Linux have light-duty screen readers bundled on them.

Microsoft Windows operating systems have included the Microsoft Narrator light-duty screen reader since Windows 2000. Apple Inc. Mac OS X includes VoiceOver, a feature-rich screen reader. The console-based Oralux Linux distribution ships with three screen-reading environments: Emacspeak, Yasr, and Speakup. The open source GNOME desktop environment long included Gnopernicus and now includes Orca.

There are also open source screen readers, such as the Linux Screen Reader for GNOME and Nonvisual Desktop Access for Windows. From the open source screen readers NVDA is gaining popularity.

The most widely used screen readers are separate commercial products: JAWS from Freedom Scientific, Window-Eyes from GW Micro, Dolphin Supernova by Dolphin, System Access from Serotek, and ZoomText Magnifier/Reader from Ai Squared are prominent examples in the English speaking market [3].

Screen readers have been evolved with the operating system; the processing and rendering of the information is dictated by the way information is presented on display or output device. It is classified as follows:

a) CLI (text) screen readers

In early operating systems, such as MS-DOS, which employed command line interfaces (CLIs), the screen display consisted of characters mapping directly to a screen buffer in memory and a cursor position. Input was by keyboard. All this information could therefore be obtained from the system either by hooking the flow of information around the system and reading the screen buffer or by using a standard hardware output socket and communicating the results to the user [3].

b) GUI screen readers

On the arrival of Graphical User Interfaces (GUIs), the situation became more complicated. A GUI has characters and graphics drawn on the screen at particular positions, and therefore there is no purely textual representation of the graphical contents of the display. Screen readers were therefore forced to employ new low-level techniques, gathering messages from the operating system and using these to build up an "off-screen model", a representation of the display in which the required text content is stored [3].

For example, the operating system might send messages to draw a command button and its caption. These messages are intercepted and used to construct the off-screen model. The user can switch between controls (such as buttons) available on the screen and the captions and control contents will be read aloud and/or shown on refreshable braille display [3].

Screen readers can also communicate information on menus, controls, and other visual constructs to permit visually impaired users to interact with these constructs. However, maintaining an off-screen model is a significant technical challenge: hooking the low-level messages and maintaining an accurate model are both difficult tasks [3].

To address these problems, operating system and application designers have attempted by providing ways for screen readers to access the display contents without having to maintain an off-screen model. By using APIs, it is possible for the provision of alternative and accessible representations of what is being displayed on the screen. Existing APIs include: Apple Accessibility API, AT-SPI, IAccessible2, Microsoft Active Accessibility (MSAA), Microsoft UI Automation, and Java Access Bridge. Not only do screen readers differ widely from each other, but most are highly configurable. For example, most screen readers allow the user to select whether punctuation is announced or silently ignored. Some screen readers can be tailored to a particular application through scripting. One advantage of scripting is that it allows customizations to be shared among users, increasing accessibility for all [3].

Screen readers can query the operating system or application for what is currently being displayed and receive updates when the display changes. For example, a screen reader can be told that the current focus is on a button and the button caption is to be communicated to the user. This approach is considerably easier for screen readers, but fails when applications do not comply with the accessibility API: for example, Microsoft Word does not comply with the MSAA API, so screen

readers must still maintain an off-screen model for Word or find another way to access its contents. One approach is to use available operating system messages and application object models to supplement accessibility APIs. Screen readers can be assumed to be able to access all display content that is not intrinsically inaccessible. Web browsers, word processors, icons and windows and email programs are just some of the applications used successfully by screen reader users. However, using a screen reader is, according to some users, considerably more difficult than using a GUI and many applications have specific problems resulting from the nature of the application (e.g., animations in Macromedia Flash) or failure to comply with accessibility standards for the platform (e.g., Microsoft Word and Active Accessibility) and this gets more worse on surfing websites. It is because of this reason that a self-voicing browser comes to importance to alleviate the limitations of screen readers on the web surfing experience [3].

c) Web Based Screen Readers

It is relatively new development technology as web-based application like Spoken-Web that is web portal for managing content like news updates, weather, science, and business articles for visually impaired users. This includes applications such as ReadSpeaker or BrowseAloud that adds text-to-speech functionality to web content. The primary audiences for such applications are those who have difficulty in reading because of learning disabilities or language barriers. Although functionality remains limited compared to equivalent desktop applications, the major benefit is to increase the accessibility of said websites when viewed on public machines where users do not have permission to install custom software, giving people greater 'freedom to roam'.

The development of smartphones, the ability to listen to written documents like textual web content, PDF documents, E-mails etc. while driving or during a similar activity in the same way as listening to music, will benefit to a much broader audience than visually impaired people. The best known examples are Siri for iOS, and Google Now and Iris for Android. The release of the Galaxy S III, Samsung also introduced a similar intelligent personal assistant called S Voice.

The capability and quality of the application also mainly depends on correct structure of the text: heading, punctuation, presence of alternate attributes for images, etc. that are crucial for a good vocalization. A website may also have a nice look because of the use of appropriate two dimensional positioning with Cascade Style Sheet (CSS) but its standard linearization, for example, by suppressing any CSS and JavaScript in the browser, can produce an incoherent

succession of texts to be vocalized [3]. The screen reader provides a user with setting to customize its functionality like verbosity and language as described below.

i) Verbosity

Verbosity is a feature of screen reading software that supports vision-impaired computer users. Speech verbosity controls enable users to choose how much speech feedback they wish to hear. Specifically, verbosity settings allow users to construct a mental model of webpages displayed on their computer screen. Based on verbosity settings, a screen-reading program informs users of certain formatting changes, such as when a frame or table begins and ends, where graphics have been inserted into the text, or when a list appears in the document [3].

ii) Language

Some screen readers can read text in more than one language (e.g., Dutch, German, Chinese), provided that the language of the material is encoded in its metadata. Some screen reading programs also include language verbosity, which automatically detects verbosity settings related to speech output language. For example, if a user navigates a website based in the United Kingdom, the text would be read with a UK English accent [3].

2.2.2 Braille Printer

A Braille printer is a hardware device for "printing" a hard copy of a text document in Braille. Braille translation software is required to translate the text from the computer into Braille. Most Braille translation programs can translate material into several grades or versions of Braille. Computerized Braille embossers definitely have great advantage over the manual Brailing method [2].

2.2.3 Reading Devices/Scanners

The reading devices for the visually impaired allow access to hard copy of ink printed materials into the computer where it becomes accessible. Once the text has been scanned, the user can start listening to the text in a clear voice. In the meantime, the user can save the scanned material for later use. Indeed, with this type of adaptive technology, it no longer presents a barrier to persons who have difficulty to read ink prints [2].

2.2.4 Braille Display

There are also devices that are able to convert ordinary print or the symbols on a computer screen into an exact tactile replica. Braille Display is a vital communication device exceptionally for persons with Deaf-visually impaired. There are also read-write systems, mostly doubling as word processors and computer terminals. Braille text is entered and manipulated by means of a simple six-dot keyboard and a few additional keys or switches. Text is displayed on a small tactile screen. To produce hard copy, the device is interfaced with ordinary standard printers or with Braille embossers [2].

2.2.5 Text Magnification

For persons with partial sight, there is an ever increasing range of useful magnifying lenses. By means of closed circuit television devices, print can be enlarged and brought into focus and small objects observed closely [2].

2.2.6 Self-Voicing Application

A self-voicing application is an application that generates an audible or oral output without requiring a separate screen reader with the help of TTS library. When a program speaks or generates an aural output, it has been termed as self-voicing. One form of self-voicing program is a self-voicing browser which produces a sound output from a standard HTML tag so that it can be used by visually impaired people or people who cannot see the screen. It is a form of AT to enable those who have difficulty reading or seeing to access a website without the use of a screen reader. The idea of it becomes viable on the web because of accessibility and usability experience difficulty using assistive technologies like screen reader by visually impaired users [3].

There is much more to a self-voicing browser being usable than devising a speech mark-up language or mimicking a telephone voice response system [20]. Using audio only to browse the Web is much more than adding multimedia capabilities to a visual browser. The mouse and display need to be replaced by a simple keypad and audio feedback. Many page designs were built with the assumption that hand and eye coordination was available so it has to be replaced with only the keypad and audio [1]. The realization of accessing a website without a display with only audio output is the same way as phone audio conversation which two or more person exchange information. If there is no full keyboard, like on a phone, then voice recognition may be necessary

for inputting into forms and search string which is another hot research area that this work will not address.

2.3 Self-Voicing Browsers

A visually impaired user needs to be able to open a URL, read a page, stop, and hear short segments again, continue, hear a link, follow that link, and listen again and fast forward through the unimportant parts. If the display cannot be seen, we need to be able to get some kind of orientation information; how big this page is, how many links, and where are we currently located as sighted users get that from visually scanning the page.

The display is two dimensional with colors and images. So it is sort of two-plus dimensions. Speech is one dimensional. We can add voice changes and pitch changes, just as color is added to the screen. Voice changes make it one-plus dimensions. It is easy to get lost on the Web. It is much easier to get lost with a one dimensional view of a two dimensional universe. So there is a fundamental difference that the self-voicing browser has to accommodate [1].

There have been various works in the area of self-voicing browsers with special requirement by various companies and research groups. A prominent group of self-voicing applications are self-voicing or talking web browsers. Traditionally, talking web browsers have been specially created, as was the case with: pwWebSpeak [36], Home Page Reader (HPR) [37], Connect Outloud from Freedom Scientific [38], and Web Anywhere from University of Washington [24].

A more recent trend has seen the self-voicing capabilities added to mainstream web browsers with free add-ons. In 2004, Opera Software created a self-voicing and speech-recognition extension for the Windows version of their web browser [5]. In 2005, Charles L. Chen devised Fire Vox, an extension that adds speech capabilities to the Mozilla Firefox web browser on Mac, Windows, or Linux. In addition to the basic features that are expected of screen readers, such as being able to identify headings, links, images, etc. and providing navigational assistance, Fire Vox provides support for MathML and CSS speech module properties [6].

Another important category is broader self-voicing applications that function as a complete audio desktop, including editing, browsing, and even gaming capabilities. These include Emacspeak enhancement for Emacs and Karl Dahlke's Edbrowse.

We will see the self-voicing browser applications and extensions that have been developed on the subsequent sections.

i) Fire Vox

Fire Vox is a free and open source extension for the Mozilla Firefox web browser that transforms it into a self-voicing application. Easy to install and operate, it works on Windows, Mac, and Linux. It can work independently, or together with screen readers such as Orca. Since unveiled in 2005, Fire Vox has gained an increasing amount of interest in the accessibility community. It has several features not found in commercial screen readers, such as the ability to handle CSS 3 speech properties to make it the only extension which supports aural style sheets. As of February 2007, Fire Vox supports WAI-ARIA markup for AJAX live regions.

In November 2007, a new version of Fire Vox was released with built-in support for AxsJAX scripts. AxsJAX is a JavaScript framework that injects WAI-ARIA based accessibility into AJAX applications, and these scripts can be used without needing to be added to the web sites directly. These scripts are essentially Greasemonkey scripts for accessibility.

Fire Vox was last updated in August, 2008. The version released by Chen is not currently compatible with Firefox 3.6 and higher. For this reason, Filippo Battaglia at Visilab Research Center of the University of Messina released ML-FireVox that is compatible with any version of the browser and supports both English and Italian voice [3].

ii) IBM Home Page Reader

Home Page Reader (HPR) was a self-voicing web browser designed for people who are visually impaired. It was developed by IBM from the work of Chieko Asakawa at IBM Japan who is a visually impaired person.

It has different set of keys designed for various functionalities. It has a history key, to move backwards and forwards in the history list; a bookmarks key to bring up bookmarks, add them and delete them. And there is a help key, to bring up on line help. It has also the jump functions which allows to jump tables and structures. In addition, it provides a page summary function that announces the title and size of the documents as measured by number of items and number of links [23].

In 2006, it was announced on the HPR mailing list that IBM does not have plans for any further updates of it and the software was subsequently withdrawn from sale by IBM in December 2006. IBM has given code to be used as a Firefox extension. It has been superseded by IBM Easy Web browsing [32].

iii) Brookes Talk

BrookesTalk [23] is another self-voicing web browser that employs a similar approach to HPR. In addition, it attempts to address the problem of communicating the semantic content of a page to the end user by providing summaries and keywords obtained by analyzing the structure of the webpage. It is reported that visually impaired users did not find the summary information of use because it was regarded as inaccurate.

iv) WebFormator and Webwizard

WebFormator from Frank Audio Data and Webwizard from Baum render the content of a webpage as a text only flat document and permit the user to access this accessible content using their normal assistive technology, typically a screen reader. This text can be navigated with a caret as a normal text field, and like the other two applications, users can bring up lists of links, frames and other features that can be of use in understanding the content of the webpage. It also provides different navigation modes for exploring HTML tables, navigating from cell to cell within the table while tables are typically used for layout, rather than structuring data, if a real data table is encountered this may be of use.

v) WebbIE

WebbIE [23] is a freeware web browser designed for screen reader users. It represents webpages as text with a caret, allowing users to use their existing screen reader or assistive technology to read, but it is not self-voicing like Home Page Reader. It was developed as a student project at the department of computation at University of Massachusetts Information System Technology (UMIST). It was first released in 2002 and has been under development. It is often bundled with the LookOUT screen reader and Thunder screen reader [3].

It uses the Microsoft web browser ActiveX control to fetch and parse webpages into W3C Document Object Model (DOM) and MSHTML DOM. It then iterates through the DOM creating a text re-presentation. The implications of this include:

- There is a delay between the web browser control rendering the webpage for sighted people and presenting the DOM to WebbIE to process. WebbIE can only access the DOM when all images and other embedded content have been rendered, which for some slow or media-heavy sites, can take time.
- The text representation is detached from the underlying DOM, so real time updates to the DOM (e.g., Ajax writes) may fail to be represented [3].

2.4 Text To Speech (TTS) for Amharic and Ethiopic Scripts

Our research work does not consider the detailed aspects of TTS implementation approaches and models; rather it focuses on review of the works done in this area with the aim of integrating to our self-voicing browser model of Amharic and Ethiopic scripts. There have been various works done towards TTS for Amharic language but no full-fledged library is produced yet. Speech synthesis and automatic generation of speech waveforms have been under development for several decades. Recent progress in speech synthesis has produced synthesizers with very high intelligibility, but the sound quality and naturalness still remain to be a major problem. In addition to this, most synthesizers currently manipulate a small number of parameters in a highly constrained manner to produce speech and thus lack flexibility. However, the quality of present products has reached to an adequate level for several applications, such as multimedia and telecommunications [12, 13].

TTS system is a system that converts free text into speech [14]. This is a process that a computer reads out the text for people. The TTS system consists of two main phases: the Natural Language Processing (NLP), which does the text analysis and the Digital Signal Processing (DSP) step, which is responsible for speech generation routines [12, 14]. The text analysis routine converts the input text into abstract linguistic description (such as phonemes and stress) via the syntactic structure and semantic focus of the sentence. Input texts are first processed by the "Text Normalization" procedure, which expands concept bearing sequence of character into word sequence of the language. The speech generation routine first uses the linguistic structure to generate the phonetic realization of each phoneme. Then it performs the phonetic-to-acoustic transformation often called as speech synthesis. The phonetic-to-acoustic transformation performs the synthesis based on the speech parameters. These two phases are usually called high-level and low-level synthesis [12, 15].

Speech synthesis is a process of producing an artificial speech from recognized text. The area has been an active research domain for the last decades and various approaches and techniques were introduced but still remain challenging to come up with a solid and viable solution.

Different researchers tried TTS for Amharic and other local languages, the work in [16] tried to develop Amharic vowels synthesizer which comprises of two major parts. The analysis part handles text analysis (transcription of the input word) and extraction of the speech parameters. The synthesis part generates the artificial speech. In the new model, smaller speech units like phonemes are not stored in the database rather the speech parameters are stored. This highly minimizes the memory requirement of the speech synthesizer than other synthesizing methods. The researcher aims to provide a way out for speech synthesis demand that doesn't come to realization due to high utilization of storage and computational power by designing a technique that requires low memory size with greater flexibility to the synthetic speech with formant approach. The researcher was able to develop a system that generates vowels from a given context using formant based approach which is an initial attempt. Unlike the concatenative method that requires speech units to be stored in a database, this work doesn't need to store any of the speech units in the database rather only speech parameters are stored to synthesize the new speech.

Also the work in [17] presented a speech synthesis system for Amharic language. The authors developed Amharic Text-to-Speech system (AmhTTS) in parametric and rule-based that employs a cepstral method by using a source filter model for speech production and a Log Magnitude Approximation (LMA) filter as the vocal tract filter. They evaluated their system based on intelligibility and naturalness on word and sentence listening tests respectively and they achieved 98% correct rates for words and an average mean opinion score of 3.2 (which is categorized as good) for sentences listening tests. The synthesized speech has high intelligibility and moderate naturalness that requires improving the naturalness of the synthesizer. Comparing with previous similar study, their system produced considerably similar quality speech with a fairly good prosody.

The research work in [18] tried to tackle the central problem of TTS synthesis which is Grapheme-to-Phoneme (GTP) conversion in Amharic. Particularly, deriving phonological features which are not shown in orthography is challenging. In the Amharic language, geminates and epenthetic vowels are very crucial for proper pronunciation but neither is shown in orthography. This paper

described an architecture, a preprocessing morphological analyzer integrated into an AmhTTS system, to convert Amharic Unicode text into phonemic specification of pronunciation. The study mainly focused on disambiguating germination and vowel epenthesis which are the significant problems in developing Amharic TTS system. The outcome of the work shows that their proposed automatic geminate assignment method was made by analyzing 666 words and they found 100% correct assignment/restoration of germination. The words were selected from Armbruster (1908) verb collections, where germination is marked manually, representing all of the verb conjugation classes (Type A, B and C).

2.5 Summary

In this Chapter, we have reviewed the challenges and opportunities in website accessibility and usability towards visually impaired people. We have also discussed the website accessibility solutions. Then, we have concisely discussed how visually impaired people access a computer system and surf a web with the aim to learn their interaction and navigational behavior. In addition, we have discussed the adaptive technology capability and methods as a means of website accessibility for visually impaired people. Next, we have discussed how a self-voicing application works and provides web accessibility for visually impaired people. We finally reviewed and discussed types of self-voicing browsers and TTS library for Amharic and Ethiopic scripts.

CHAPTER THREE-RELATED WORK

In this Chapter, we will review research works done in the area of web accessibility and self-voicing browser model towards visually impaired people. To the best of our knowledge, there is no research work done in the area of self-voicing browser model for Ethiopic scripts towards visually impaired people. Hence, our review will mainly focus on works done for English and other languages.

3.1 Web Tree

The research work in [21] tried to examine methods to aid the non-visual browsing of web based documents using TTS approach. The primary focus of the work was to try to increase the usability of document browsing or navigation by offsetting problems imposed by the serial nature of speech. It provides an option for the user to manipulate certain elements to automatically expand and collapse the rendering, or whether it should appear in the rendering at all, to effectively allow for easy generation of alternative document views. To speed up navigation, the user can move through the document based on the element of their choice. The research work introduced a solution to overcome the limitation in internal document navigation and page summarization of hypertext documents by enabling users to perform search and create document rendering based on their preference. It also provides alternative page summaries based on user selected components of the underlying mark-up. The user interacts with content rendered through a character-oriented virtual screen/display.

The manner in which this work summarization functions differs from currently available solutions in that they are not limited to a set of elements named by the developer. Rather, the user selects the elements on which the rendering is based. Also, summaries can be generated using a group of prescribed elements as opposed to the conventional approach which generates a list containing only a single kind of element, like a list of links or header elements. Thus, entire element sub-trees may be efficiently traversed with minimal difficulty.

As with any screen reading application, better results are obtained when a webpage is marked up in accordance with accessibility guidelines. The major concern of this methodology is the effect on the efficiency at which the user can navigate to and assimilate information. The method is applicable to any document organized in a tree-based structure. The system stores the parsed

document in a DOM tree structure for internal manipulation. Both the navigation and display functions interface with the DOM structure to manipulate the document to the user's specifications. Therefore, once issues resulting from the accurate parsing of these documents are offset, the viewing of these pages should not be problematic.

The weakness of this work is that it doesn't consider JavaScript 6 or Adobe Flash 7 components and Extensible Markup Language (XML) based web documents. The evaluation result also demonstrates that users with limited knowledge of HTML/XHTML markup elements had problems to navigate web documents.

3.2 WebbIE

WebbIE [23] was developed with the intent to allow users to access standard browsers like Windows Internet Explorer through an interface that simplifies and represents the content without losing information or being too complicated for non-expert computer users. It is not a separate self-voicing application, but rather provides support for visually impaired people to use in parallel with their screen reader on their familiar environment.

Internally WebbIE uses the IE control object, and this handles the acquisition of webpages and parsing the HTML into the W3C standard DOM which provides rich API for manipulating and querying the webpage (see Figure 3.1). Using IE guarantees maximum compatibility with websites; although another control that handles fetching webpages and parsing them into the DOM could be used with a minimum of alteration and the Mozilla browser has been tested and works successfully.

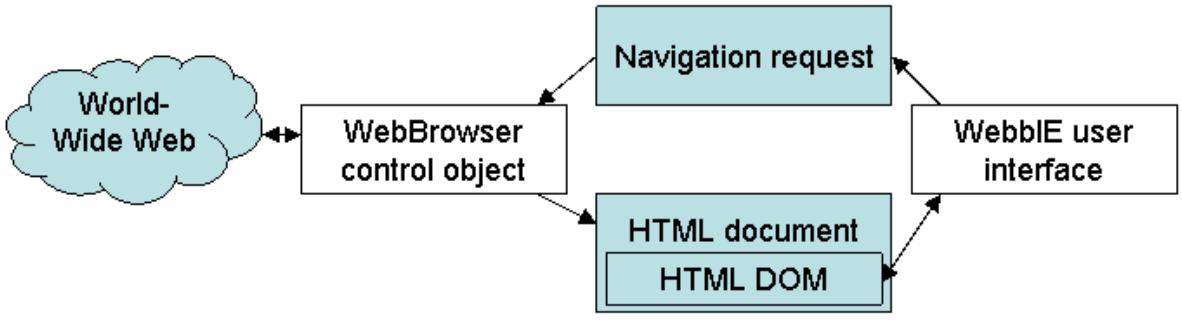

Figure 3.1: The WebbIE Architecture

WebbIE navigates the DOM, collecting active content components such as hypertext links and form components, and building up a plain-text representation of the content and the plain text is

presented to the user. Components are presented on new lines with distinguishing titles, like "LINK" for a hypertext link. Functionality is accessed through pressing the return key on a line with a presented component. Figure 3.2 shows WebbIE in action. WebbIE supports existing IE bookmarks, frames, HTML 4 mostly, forms, tables, and display of embedded multimedia.

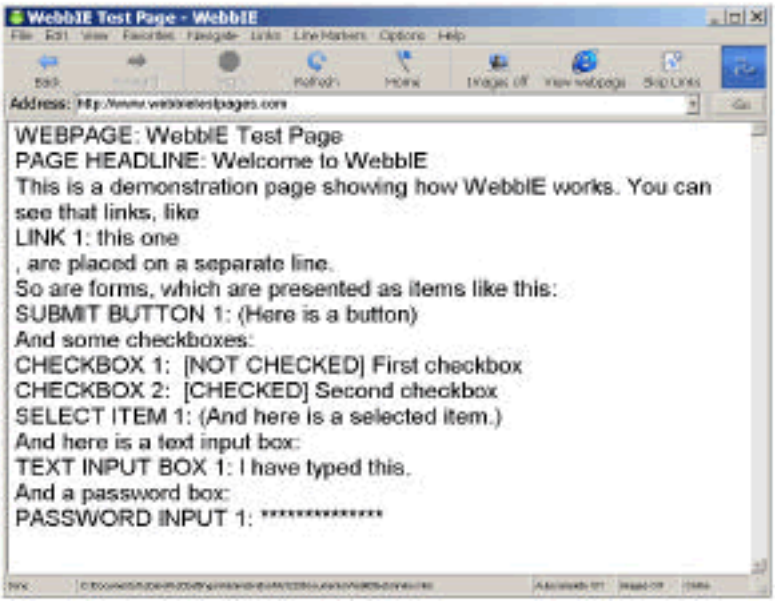

Figure 3.2: Webbie in Action

The work also tried to address partially the main accessibility issues associated with the web such as complex webpages, summarizing webpage content, images, JavaScript, Flash/Java/multimedia embedded content, forms, and frames.

It was evaluated with nine users by means of a questionnaire to learn about their background knowledge on web browsing and WebbIE. The users were all associated with a company that distributes WebbIE and performs training, so the results reflect some common background of training and preference. The users were all screen reader users. The six users that had used the web before used IE in conjunction with their screen reader, although their level of success varied. The users ranged in experience and levels of visual impairment, and the sample size was small. The users expressed confusion over the ignorance of non HTML embedded content which confirms that HTML is the most accessible format for web content. All the users expressed a preference of Google as a search engine suggesting that a tailored Google interface within WebbIE would be important which the work takes as input for their next development. The users also made complaints with WebbIE interface about the many links often encountered at the top of a webpage

before the content of interest. As a consequence the WebbIE function that skips links and moves the cursor to content proved to be an important functionality. Based on the evaluation result, users did not report any general problems with accessibility to websites, but only one user reported them if inaccessible site is encountered while surfing, it will not attempt hard but rather go for another site. This shows companies should make their website developed in accordance with web accessibility guidelines which in turn can give them a potential advantage over their competitors.

The weakness observed on the research work is that it was evaluated with limited range in users experience and levels of visual impairment, and the sample size was small. The other weakness is on the interface which has also many links placed on the top of a webpage before the content; this creates inconvenience for the user to listen same list of links on every page.

3.3 Capti

The research work in [22] focused on providing universally and ubiquitously accessible web browsing application for both visually impaired and sighted people. It is a platform independent web application exclusively for web browsing with its own screen reading functionality on desktop computers to universally and ubiquitously access webpages. The work takes the gap on currently available screen readers that provide very basic functionality and does little to help to find information in webpages faster. Arguably, screen readers can be combined with other applications to provide more usable access to information. The work also tried to reduce the information overload problem for the visually impaired to get the main content from webpages.

The unique benefit provided by Capti is to make web browsing more intuitive and usable for people both with and without vision impairments. Capti can be used to listen to news, blogs, e-books, encyclopedias, and other webpages with the convenience of an audio player.

Capti integrates with existing web browsers, turning them into self-voicing applications. It also enables users to add webpages to its playlist that is synchronized across devices so that a playlist built on a regular computer can be listened to on an iOS device, and then continued from the same place on a laptop online or offline. In addition, it enables users to share their favorite articles with other people through social networks. It also uses intelligent algorithms to extract main content from webpages and identify articles that span multiple webpages. In the screen reader interface, it helps users find differences in webpages as they navigate from page to page in template based websites. It has built in webpage segmentation to facilitate page navigation. It also uses an

algorithm to associate labels with form fields for more accessible form filling, and it has a number of other facilities to make non-visual browsing more usable.

Capti runs on Windows, Mac, Linux, and iOS. It is interoperable with other screen readers, and can be used in parallel with them. Screen reader users can also turn the Capti's self-voicing feature and use Capti with their favorite screen readers.

3.4 HearSay

The HearSay research work [26] focuses on non-visual web browser designed with the goal of improving web accessibility. The research work has two previous versions; on their initial version, they segmented pages according to semantically meaningful parts. On their second version, they used link context to identify and start reading from the segment relevant with the respect to the context.

The third version HearSay 3 has functionalities that include transparent support for multiple languages, collaborative labeling that harnesses the power of the social web, and a usable interface for handling dynamic web content. It detects the languages used in a webpage and chooses both an appropriate text-to-speech engine and a voice to read the page. Many websites specify the language with the XHTML "*lang"* attribute. When this attribute is included, it switches between the supported languages. In pages that include multiple languages, such as Wikipedia, the system seamlessly switches between languages as it reads the content.

HearSay 3 supports collaborative labeling by enabling users with choice of either a pre-existing label or defining a new one to any webpage elements using keyboard shortcuts and voice commands. The labels are stored in both local and remote repositories, in this way, maximizing the benefits of collaboration and personalization at the same time. In addition, HearSay 3 detects dynamic updates (DHTML, AJAX, etc.) to the pages and provides a usable interface to assist users in reviewing the updated content. This enables HearSay to provide users with a consistent context when a page is dynamically updated. It lets the user know that content has changed with minimal interruption to the user.

The weakness of this research work is that it doesn't detect language encoding of a webpage if XHTML "*lang"* attribute isn't available and limited language base. The other limitation is the high burden of users to assign labels. The researchers, aware of these gaps, planned to address on their

future work to make their work their next generation self-voicing web browser for non-visual use of web 2.0 applications and leverage its strengths to support collaborative accessibility through labeling.

3.5 WebAnywhere

The main objective of the work in [24] is based on the fact that people should access the web on computers that are not their own. Visually impaired web users lack the ability to access the web from all available computers because their access relies on expensive, specialized software programs called screen readers. It is a web-based, self-voicing web application that enables visually impaired web users to access the web from almost any computer that can produce sound without installing new software (see Figure 3.3).

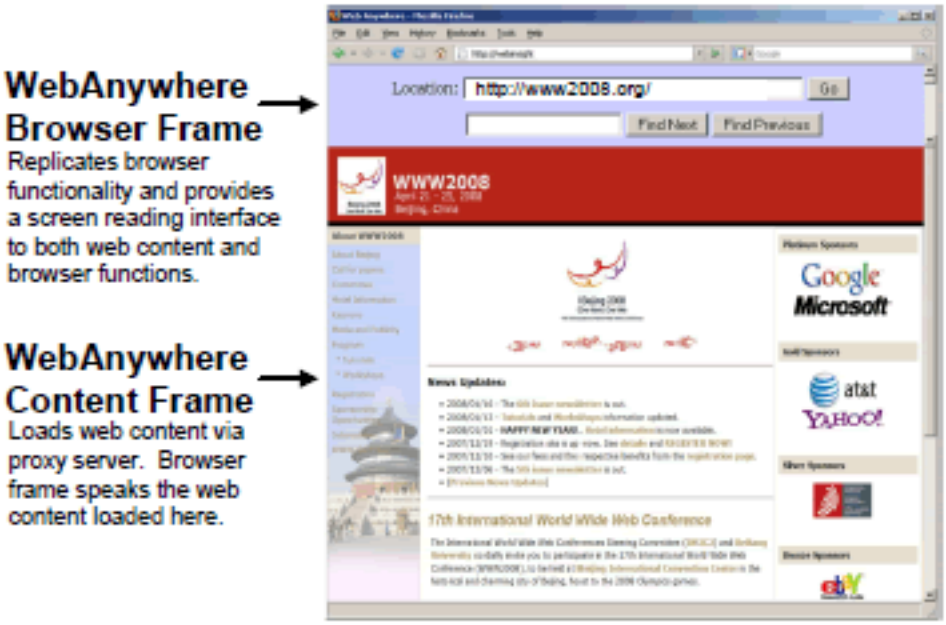

Figure 3.3: WebAnywhere

The WebAnywhere screen reader enables visually impaired users to quickly access web content on any available computer and supports a rich set of user interaction. It also offers a convenient and inexpensive mechanism for web developers to evaluate their site for web accessibility review for those who can't afford to buy a traditional screen reader. Users can browse webpages, skipping paragraph, sentence, word or character. They can quickly navigate between tab-indexed elements, heading elements, form elements, links and table rows and columns using standard keyboard shortcuts. Form input is also supported and does not require a separate forms mode.

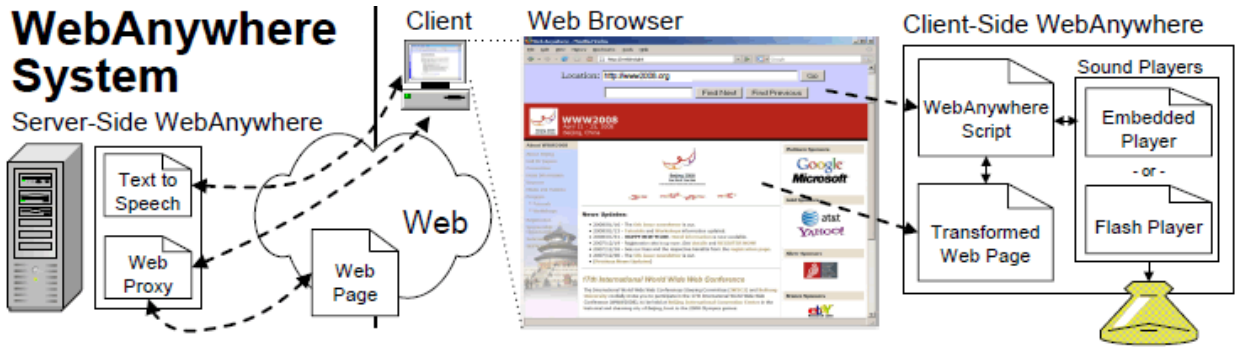

Figure 3.4: Web Anywhere

The Web Anywhere interface is similar to that of traditional screen readers. Support of TTS and web proxy functionality has been moved to a remote server in order to compensate for web application limitations. As shown in Figure 3.4 the system consists of three components which are client-side JavaScript that supports user interaction which determines which sounds to play and coordinates the other subsystems. The second component is server-side text-to-speech generation and caching. The third component is server-side web proxy that makes webpages appear to come from a local server to overcome cross-site scripting restrictions.

The weakness of this work is that it does not respond to a user input if the browser window containing WebAnywhere loses focus. The other main limitation is that it lacks the full screen reader functionality.

3.6 Summary

In this Chapter, we have mainly discussed works related to self-voicing browser application. Since there has not been any related research works done in self-voicing browser application for Ethiopic scripts, we have presented research work done for English and other languages. The research works done in the area of self-voicing browser for webpages of English content, there are still unattained gaps that need to be filled to allow visually impaired people experience web equally with sighted people. Our research work takes the motivation and approaches used by WebAnywhere and HearSay to set ground framework for self-voicing model for webpages in Ethiopic script as a low cost, on-the-go, anywhere web access that can produce sound without requiring additional software installed.

CHAPTER FOUR-Design of Multilingual Self-Voicing Browser Model for Ethiopic Scripts

In this Chapter, we will present the required components to design multilingual self-voicing browser model for Ethiopic scripts. We will in detail present each component's responsibility within the model and the interaction among them. The components are categorized as voicing-server, web-server, and client-side. The client-side components are User Interface and Audio Player. The voicing-server side components are HTML Purifier, Language Identification, Parser Engine, Text Transcoder, TTS, Language selector, and Language Base. The web-server is a server which hosts any given website.

4.1 Model Overview

The model shown in Figure 4.1 is designed to be multilingual capable of providing web accessibility of webpages with Ethiopic scripts. The model has its own webpage with an interface to type in the web site address the user desires to surf and language selection to augment the browsing with what Ethiopic script the website is built with. The model is designed as a web based application that can be used and accessed from any standard web browser. The model does not require any additional software to be installed on the client-side except the machine on the client-side has both an interface to the Internet and audio output. The HTML purifier decomposes the webpage and cleans up in accordance with W3C standards and specifications. The language identification analyzes the given webpage with aim to determine the language encoding. The parser will determine the webpage encoding and declaring language identified by language identifier and then set the default language base to be used by the TTS engine. Then, the parser traverses the Document Object Model (DOM) of each loaded webpage in the order of appearance sequentially as in the visual page. The TTS library generates audible sound for the corresponding IPA based textual representation generated by the IPA text transcoder. The audio from TTS is generated remotely at the voicing-server and played using generic audio players at the users' browser on client-side.

To improve the model usability, it is designed to function on all computers with only two requirements; having Internet connection and audio output. To allow visually impaired users browse the web using any computer other than their own has concerns like permission to install new software, thus the model is designed by moving functionality that would require this to a

voicing-server and thus users can use on any computer that happen to have only interface to the Internet and audio output.

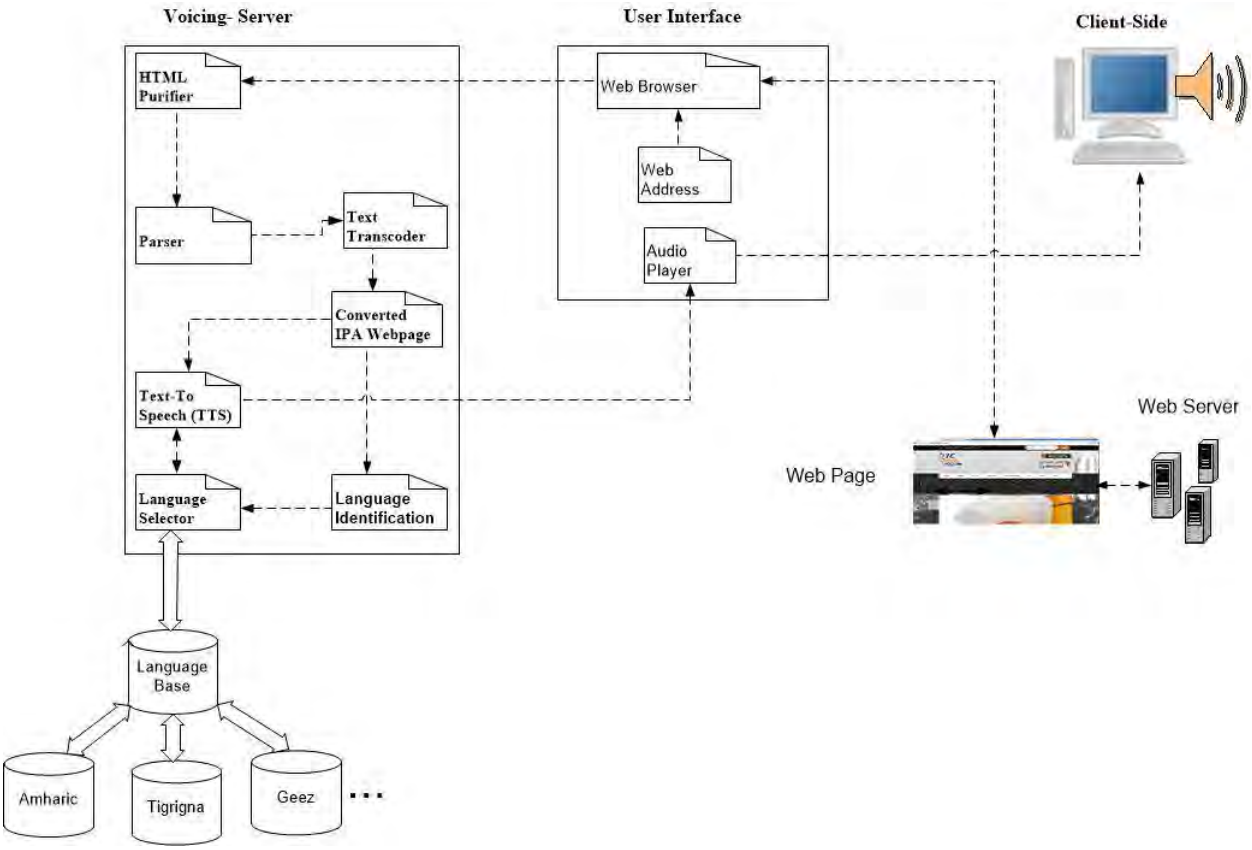

Figure 4.1: Multilingual Self-voicing Browser Model for Ethiopic Scripts

4.2 User Interface

The user interface has one sub-component which is *Web Address* running on web browser as a web based application. The web address component on the client-side is responsible to accept the Uniform Resource Locator (URL) of a given web site through the Hypertext Transfer Protocol (HTTP). The web site source is fetched from the web server and fed into the HTML purifier component. The user interface is provided as a web based application that is simply run with standard permissions. It replicates needed functionality, such as the address field to type in the website address. The user interface is written in HTML and Hypertext Processor (PHP) with a field to type in URL.

4.3 Audio Player

The audio player is responsible for playing sound generated by a TTS engine in the client machine. Because the model is designed as web based, it leverages existing functionality provided by the browser and it uses audio player embedded within the browser. It plays sound on the client machine that is capable of playing MP3 format. In a study conducted over five webpages of English language, the latency of retrieving new multiword sound from server-side was less than 300 milliseconds on a high-bandwidth connection [24]. Based on this fact, our model is designed for the audio player to use IPA based TTS engine of our voicing-server.

4.4 HTML Purifier

The HTML purifier is the first component to be called after the webpage is fetched from the web server as shown in Figure 4.2. In our model, we used HTML purifier library written in PHP that is a standard compliant HTML filtering library. The HTML purifier removes all unnecessary code like XSS with a thoroughly audited, secure yet permissive whitelist; it also makes sure documents are standard compliant, something only achievable with a comprehensive knowledge of W3C's specifications. The HTML purifier decomposes the whole document into tokens and removes non-whitelisted elements, checks the well-formedness and nesting of tags, and validates all attributes according to W3C standards [27].

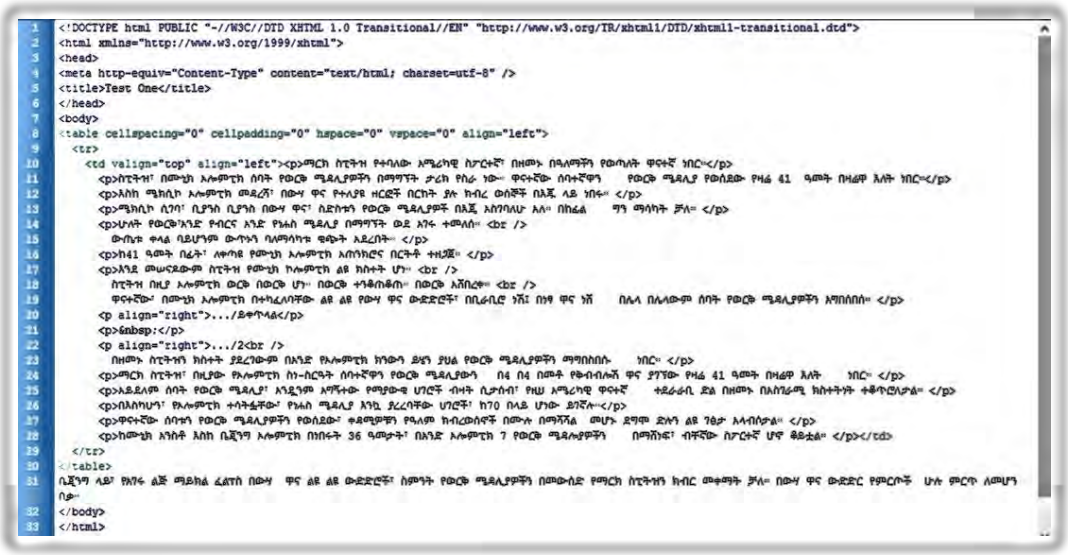

Figure 4.2: HTML File

4.5 Language Identification

The language identification is responsible for automatically recognizing the language encoding of a given webpage that will be used by TTS to select the respective language base. Language identification of a webpage is normally performed based on its content by analyzing coherency of sentences. There have been various algorithm and technique introduced for automatic recognition of language declaration of a given webpage, one of the algorithm is a machine learning algorithm to identify the language based on URLs [40]. The other algorithm is n-gram based algorithm complemented with heuristics and similarity measure [41]. Automatic language identification specifically for Ethiopic scripts is major research area that this research work will not cover.

4.6 Parser Engine

The most important component on designing a self-voicing browser model is a parser engine. A parser engine within our model is responsible for manipulating a stream of HTML tags through a process of tokenization and tree generation according to namespace defined set of rules to generate a *Document* object. It transforms markup tags into semantic structure to facilitate the audio browsing. The parser library accepts URL or file to generate the DOM to be fed into the text transcoder. In our model, we fed the source of the webpage as a file to be parsed and DOM be generated as shown in Figure 4.2. The parser library generates the DOM by keeping the hierarchical structure of the HTML tags and their corresponding values with a unique tag name *"<**text**>"*. The value of each HTML tag is written in the *"<**text**>"* tag that is in turn fed into a text transcoder to generate the text-based version of the parsed DOM.

After the webpage has been purified, the parser with annotations generates a DOM of the original webpage by formatting visual layouts, images, and Java Scripts. If images, audio, and videos are not labeled with appropriate alternative or supportive text, the parser ignores them. Customizations using annotations can include: altering the ALT-attributes of images, adding structural markup (H1, H2, etc.) to content, and reformatting JavaScript navigation elements. The parser with annotations helps to customize a webpage according to specific rule. Webpages often contain clutter (such as pop-up ads, unnecessary images and extraneous links) around the body of an article that distract a user from actual content. Automatic extraction of useful and relevant content from webpages has many applications, ranging from enabling users to access the web more easily over

constrained devices like PDAs and cellular phones to providing better access to the web for the visually impaired through speech rendering, and text summarization. Content extraction is particularly useful for the visually impaired. A common practice for improving webpage accessibility for the visually impaired is to increase font size and decrease screen resolution; however, this also increases the size of the clutter, reducing effectiveness. Screen readers for the visually impaired, like Hal Screen Reader by Dolphin Computer Access or Microsoft's Narrator, don't usually automatically remove such clutter either and often read out full raw HTML. Therefore, both groups benefit from extraction, as less material must be read to obtain the desired results [39].

In the text-based view, we start by analyzing the webpage hierarchical structure that is represented in the DOM as tree structure. The DOM tree structure is represented as a tree of various types of node objects. The Node interface defines properties and methods for traversing and manipulating the DOM tree. Different types of nodes in the DOM tree are represented by specific sub interfaces of Node. Every Node object has a *nodeType* of *Element, Text, Document, comment, document fragment,* and *Attr* that specifies what kind of node it is. The node elements such as H1,H2, DIV, Span, P, Table,UL, LI etc. and target the elements on the page that are most likely to affect a user's ability to access a site. Next, we check the DOM tree for nodes to ensure that images have descriptive ALT tags, and that alternatives are provided for multimedia, Flash, and scripted content. In addition, it helps the webpage to be rendered in accessible and usable way so that low vision and visually impaired people can interact with webpage information [25].

The parser in our model has the following features:

- All script and style elements are deleted,
- All link elements with an "*href*" attribute value that matches the regular expression '\.css\??' are deleted,
- All inline styles are deleted,
- Images are replaced with their alt values, and a link to view the image is inserted after,
- Inputs of type image become inputs of type submit,
- All links on the page become fully qualified,

- Links with a mailto, JavaScript or data scheme links are left alone,
- Only webpages accessed via HTTP are displayed at this time,
- Only webpages that are of type text/html are parsed and displayed.

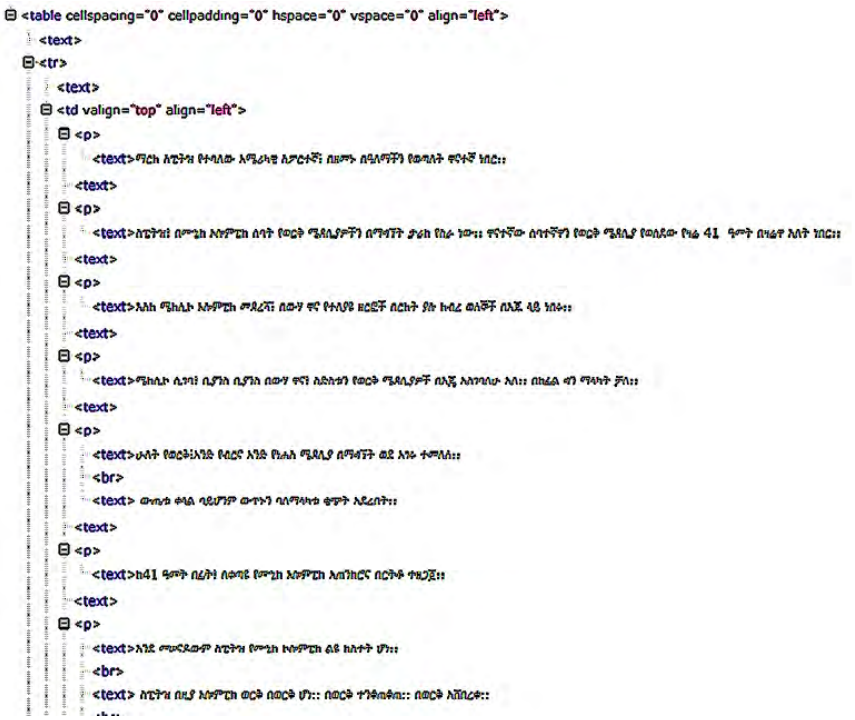

Figure 4.3: Parser Generated DOM

4.7 Text Transcoder

The text transcoder generates the text-based version of the webpage in IPA representation which is called converted IPA webpage that in turn is fed into the TTS engine. The Text Transcoder provides a text-based page that is similar to what a user would hear when using screen-reader technology. The text transcoder generates the text-based version of the webpage from parsed HTML tag values written in the unique tag name *"<**text**>"* by converting each word in IPA equivalent representation. As we can see in Figure 4.3, the parser generated a DOM snippet and the text transcoder traverses looks tag name *"<**text**>"*. The IPA equivalent representation of each Ethiopic script character is stored in a flat file; for instance, the Amharic character "ቀ" IPA equivalent character is "q" and same equivalent representation will be defined for Tigrigna and Geez.

Text transcoding have different types: *Chunking, Filtering, Reordering, Scaling*, and *Text-only*.

a) Chunking involves decomposing a webpage into a series of smaller pages or chunks that can be more easily accessed on small screen devices.

b) Filtering does not restructure a page like other transcoding methods, but merely removes content from the page that is deemed unnecessary.

c) Reordering involves rearranging a webpage so that content considered to have more important information is placed in a position that is easier to access.

d) Scaling attempts to reproduce a desktop version of the Webpage on small screen devices.

e) Text-Only transcoding results in a text only version of the Webpage.

Since a webpage is above all an informational document, converting a given webpage representation to text-based will give us the required information. Structuring a webpage properly without considering any style sheet applied will enable the webpage to be accessible when converted to text-based. Our model uses Text-Only version in IPA equivalent representation text transcoding to convert a given webpage to text-based.

4.8 Converted IPA Webpage

The converted IPA webpage is an IPA equivalent text-based version of a webpage generated by the text transcoder. The converted IPA webpage is an IPA textual representation of the given webpage after purification and parsing. The converted IPA webpage is a collection of words generated from the DOM tree element by traversing using a pre-order Depth First Search (DFS), which is approximately top-to-bottom, left-to-right scheme. The resulting converted IPA webpage might not be optimal for those pages that rely on graphics to display navigation and text may not be automatically transcoded to show navigation and text as the visual rendering. The generated list of IPA equivalent words in the converted IPA webpage will be used by the IPA based TTS engine for the selected language base.

4.9 Text-To-Speech

The TTS engine is responsible to generate an audio for transcoded IPA based textual content of a converted IPA webpage by fetching the respective languages from the language base. The TTS component is not part of our thesis work and we transcoded the webpage in IPA representation to

use a TTS engine that can generate audio from IPA equivalent text. The appropriate TTS engine is fetched based on the selected language from the language base and voice to read the transcoded markup contents. Many websites specify the language encoding using the XHTML "*lang"* attribute; though, we have only standard XHTML "*lang"* attribute definition for Amharic language which prevents us from automatically detecting and seamlessly switching for other local languages of Ethiopic script. Because of this reason, we need a component that can automatically identify the language declaration of a webpage.

Retrieving sounds involves both caching and prefetching on both the server-side and the client-side. As text-to-speech conversion takes place in the voicing-server, the latency of requesting, generating and retrieving speech could potentially reduce the quality of service. Latency of retrieving speech is an important factor because it directly determines the user's perceived latency.

TTS conversion is a relatively processor intensive task. As most browsers maintain their own caches of files retrieved from the web, the speech that is retrieved from the voicing-server is cached on the client machine by the browser. Retrieving sounds from the browser cache to play is a very low-latency operation relative to retrieving sounds from the voicing-server.

4.10 Language Selection

The language selector is responsible for selecting the respective language library from the language base based on language identification component result. The TTS fetches the respective library from the language base and plays through audio player enabled on the web browser.

4.11 Language Base

The language base is a multilingual component which stores audio files for Ethiopic scripts that are fetched and used by the TTS engine. The user selects the language option for the webpage encoding language when browsing a given webpage and the TTS will invoke the corresponding language base for each transcoded text. The Language base is what it makes our model multilingual by having TTS for Ethiopic scripts.

4.12 Summary

In this Chapter, we have discussed the architecture of Multilingual Self-Voicing Browser Model for Ethiopic-Scripts and components. We have also discussed each component's responsibility within the architecture and interaction among them. The model is organized in client-server

architecture comprising eight main components categorized as client-side, web-server, and voicing-server. The user interface component, with its sub components web address, fetches the webpage from the web-server. The user interface provides a field to add the URL of the webpage. After the webpage is fetched from a web-server it is redirected to the voicing-server. The HTML purifier is responsible for cleaning the webpage document from unnecessary code and HTML tags not coded in accordance with W3C's specifications. Then, the parser with annotations accepts the webpage cleaned by the HTML purifier to generate DOM by formatting visual layouts, images, and Java Scripts. The result of the parser is fed into the text transcoder to generate an IPA text-based version of the webpage from parsed HTML tag values written in the unique tag name *"<**text**>"*. The result of the text transcoder is a converted IPA webpage which is a text-version consisting of collection of words generated from each DOM element in IPA equivalent representation. The converted IPA webpage is fed into the TTS engine to generate and play using an audio player by fetching the corresponding language from the language base. The language base is a collection of audio files for Ethiopic-Scripts that are to be used by the TTS engine.

CHAPTER FIVE-Implementation of Multilingual Self-Voicing Browser Model for Ethiopic Scripts

In this Chapter, we will present the implementation detail of self-voicing browser model for Ethiopic scripts in detail. We discuss only the core components implementation detail of our thesis work scope. The client-side web based application that interacts with the user is implemented using HTML, PHP, CSS, and JQuery that provides option to type in address of a webpage and selection of language for the webpage encoding. We have used a programming language PHP to implement the voicing-server side components HTML purifier, parser, and text transcoder.

First, we will see in detail how the user interface is implemented. Secondly, we will see in detail the algorithm and techniques used for purifying a webpage after a webpage is retrieved from the web server. Thirdly, we will explain the parsing implementation detail which traverses the DOM of a purified webpage to generate in the form that can be used by text transcoder. The last section focuses on text transcoder specific methods to generate textual representation of a webpage.

5.1 Implementation Environment

For this research work, we have used three main API for HTML purification, HTML parser, and text transcoder. We have modified each API library to make them fit in the model.

The algorithms we have developed are implemented using the PHP programming language. We have installed additional PHP core libraries to enhance HTML purifier capability.

We have used Windows 8 64-bit x-64 based processor as an operating system. The hardware component comprises of Intel(R) Core(TM) i5-3230M CPU of 2.60 GHz, 6GB memory, and 1TB hard disk.

5.2 User Interface

The user interface shown in Figure 5.1 is the component that interacts with the user to type in the webpage address. The user interface is designed and implemented in responsive scheme with Bootstrap CSS framework and JQuery library that automatically detects the platform and render the webpage to fit the targeting device. The application webpage is defined on HTML5 *doctype* as Bootstrap uses CSS and HTML elements from HTML5. The transcoded webpage will be shown on transcoded webpage panel while the webpage is being played in audio format.

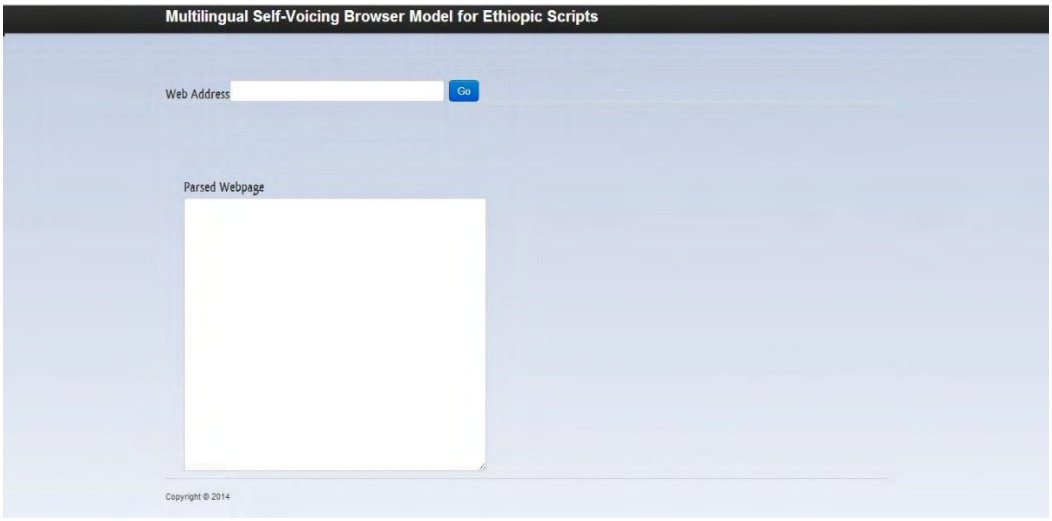

Figure 5.1: User Interface

We have defined different sets of bootstrap framework CSS definitions that will be rendered based on targeted platform and minified them for faster loading time. When the application opens in platform of resolution 350px, it will render the appropriate CSS definition and makes it fit as shown in Figure 5.3.

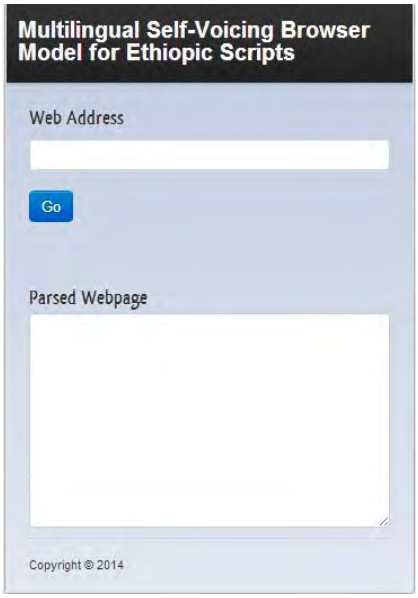

Figure 5.2: Device resolution of 350px

5.3 HTML Purifier

The HTML purifier component is responsible for cleaning the webpage HTML tags for improper coding. We need this component to clean up the webpage so that the parser traverses the DOM to generate a page that can be transcoded. As shown in Figure 5.4, the HTML purifier has different sets of parameters to configure how the purifier should behave. We have used a library built in PHP 5 which is a standard compliant HTML filter library capable of removing all unnecessary code better known as Cross-Site Scripting (XSS). It also ensures webpage is coded in accordance with W3C's specifications standards. We have selected HTML purifier library built in PHP 5 for its wide range of filtering capability, easily customizable feature, and above all it is open source. It provides sets of attributes for purifying a webpage like auto format of HTML tags, CSS manipulation, error collection, and HTML document type filtering. We have enhanced the library capability by enabling or adding optional extensions of PHP such as:

- iconv: Converts text to and from non-UTF-8 encodings to UTF-8 encodings
- bcmath: Used for unit conversion and image crash protection
- tidy: Used for pretty printing HTML
- CSSTidy: Clean CSS stylesheets using %Core.ExtractStyleBlocks
- Net_IDNA2 (PEAR): IRI support using %Core.EnableIDNA

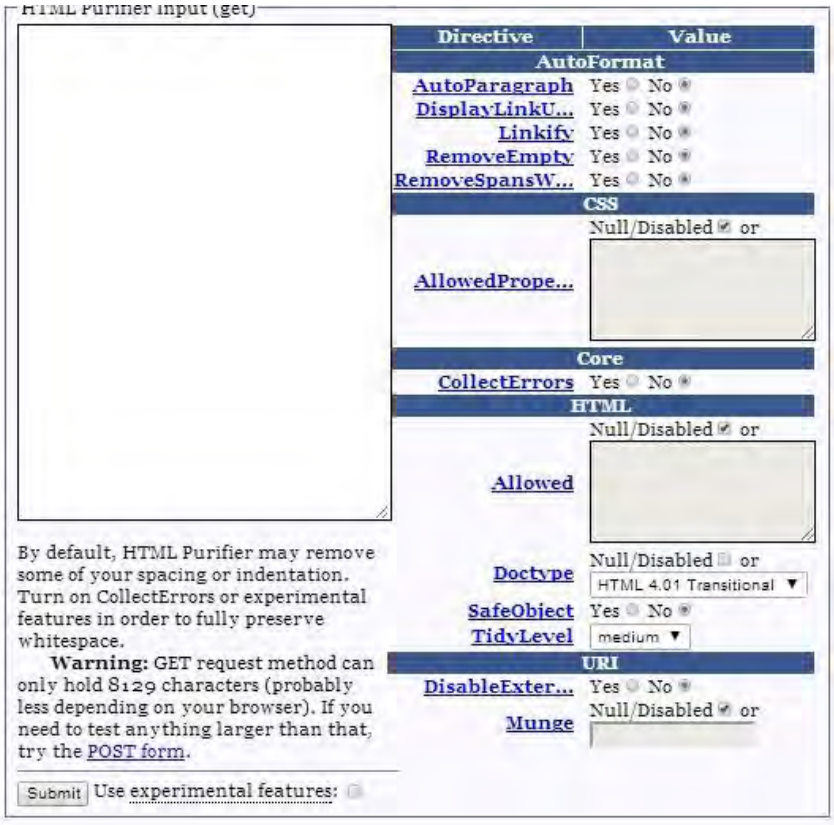

Figure 5.3: HTML Purifier

HTML Purifier can process the following doctypes:

- XHTML 1.0 Transitional (default)
- XHTML 1.0 Strict
- HTML 4.01 Transitional
- HTML 4.01 Strict
- XHTML 1.1

It supports character encodings:

- UTF-8 (default)
- Any encoding iconv supports (with crippled internationalization support)

We have configured the API library to be HTML Tidy which is another library that cleans up poorly written HTML webpages. HTML Tidy corrects and cleans up HTML content by fixing

markup errors. More recently, Tidy has been extended to support HTML5. The HTML Tidy is not required to be configured separately; instead we have embedded to the HTML purifier library. The Tidy library performs HTML cleanup tasks which includes mismatched end tags, miscue tags, missing end tags, mixed up tags, misplaced tags, missing "/" in end tags, amending markup for missing tags, appending missing quotation marks around attribute values, removing unknown/proprietary attributes, and tags lacking a terminating ">". The Tidy performs the cleanup as shown in Table 5.1.

Table 5.1: Tidy HTML

Unpurified HTML	Tidy Purified HTML
Mismatched end tags የእናት አገሩን ገንዘብ	የእናት አገሩን ገንዘብ
Miss-nested tags ኢቲቪ በግድ አስታውሱ በግድ ዳግሞ አልቅሱ ብሎ ሙት አመት ምናምን እያለ ሆዳችንን ሊያባባ ይሞክራል እንጂ እኛማ ከሳቸው ሞት በኋላ ስንት የሚያስለቅስ ገጥሞናል::ሆድ ይፍጀው ብለን እንጂ!በጣም እርር...አንጀታችን ቁስል...ወሽመጣችን ቁርጥ ብሎ...እንደው የት ሀገር ልሂድ? ብለን አልተማረርንም?	ኢቲቪ በግድ አስታውሱ በግድ ዳግሞ አልቅሱ ብሎ ሙት አመት ምናምን እያለ ሆዳችንን ሊያባባ ይሞክራል እንጂ እኛማ ከሳቸው ሞት በኋላ ስንት የሚያስለቅስ ገጥሞናል:: ሆድ ይፍጀው ብለን እንጂ!በጣም እርር...አንጀታችን ቁስል...ወሽመጣችን ቁርጥ ብሎ... እንደው የት ሀገር ልሂድ? ብለን አልተማረርንም?
Missing end tags <h1>የእናት አገሩን ገንዘብ <h2>የእናት አገሩን ገንዘብ</h2> <h1>የእናት አገሩን ገንዘብ</h1>	የእናት አገሩን ገንዘብ የእናት አገሩን ገንዘብ የእናት አገሩን ገንዘብ
Mixed-up tags የቻይና መንገድ በጋ በጋ እንደ አስፓልት ከረምቱን ደግሞ የዋና ገንዳ ይሆናል::ቻይና	የቻይና መንገድ በጋ በጋ እንደ አስፓልት ከረምቱን ደግሞ የዋና ገንዳ ይሆናል::ቻይና ወዳጅነታችን ውሃ ለመሻገር እስከመተዛዘል መድረሱን በዚህም የፈጠርነው የስራ እድል መንግስት በኮብልስቶን

ወዳጅነታችን ውሃ ለመሻገር እስከመተዛዘል መድረሱን በዚህም የፈጠርነው የስራ እድል መንግስት በኮብልስቶን ለተደራጁት ከሰጠው እንደማይተናነስ የምናውቀው እኛ ነን::	ለተደራጁት ከሰጠው እንደማይተናነስ የምናውቀው እኛ ነን::
Missing "/" in end tags <a href="http://www.shegerfm.com">ሸገር<a>	ሸገር
List markup with missing tags: <body> አንድ ሁለት	<body> - አንድ - ሁለት </body>
Missing quotation marks around attribute values ሸገር	ሸገር
Unknown/proprietary attributes <a hrex="http://www.shegerfm.com>ሸገር<a>	ሸገር
Tags lacking a terminating ">" <p በጋ በጋ እንደ አስፓልት ክረምቱን ደግሞ የዋና ገንዳ ይሆናል::</p	በጋ በጋ እንደ አስፓልት ክረምቱን ደግሞ የዋና ገንዳ ይሆናል::

In addition, we have configured the behavior of the Tidy by setting up the parameters as shown in Figure 5.5. There are three levels of cleaning in the purification which are *light, medium, and heavy*. Light is the lenient level. If a tag or attribute is about to be removed because it is not supported by the doctype, Tidy will change it into an alternative that is supported. Medium is the correctional level. At this level, all the functions of light are performed, as well as some extra, non-essential best practices enforcement. Changes made on this level are very benign and are unlikely to cause problems. Heavy is the aggressive level. If a tag or attribute is deprecated, it will

be converted into a non-deprecated version. We have used the medium level [27]. The level can be changed by setting the "*%HTML.TidyLevel"* configuration directive.

The main reason we have set the purifier at the medium level is that it is tasked with converting deprecated tags and attributes to standards-compliant alternatives, which usually need ample amounts of CSS. It is also not failsafe, sometimes things do get lost in the translation. This is why when HTML purifier can get away without cleaning, it will not; this is why we set the default value medium and not heavy [27]. There are only a few attributes that have problems across browsers as shown in Table 5.2.

Table 5.2: Attributes incompatibility across browsers

Element Attributes	Changes
Caption:align	Firefox supports stuffing the caption on the left and right side of the table, a feature that IE does not have. When align equals right or left, the text will simply be aligned on the left or right side.
Img:align	The implementation for align bottom is good, but not perfect. There are a few pixel differences.
Br:clear	Clear both gets a little unsteady in Internet Explorer.
Hr:noshade	All browsers implement this slightly differently, we have chosen to make no shade horizontal rules gray.

We have configured the purifier to clean presentational markup such as <font>, <nobr>, and to be replaced by CSS properties. Besides, we have adjusted the Tidy cleaning method not to delete "<*p*>" and heading tags "<h1>, <h2> etc." that are not followed by any content. In the case of instruction set by PHP, ASP, or JSP within the HTML element, the purifier is

configured to ignore and delete any unmatched tag. For example, if we use PHP, ASP, or JSP to create a start tag and the end tag written in HTML markup, the purifier will automatically delete the end tag.

config-tidy.txt - Notepad

File Edit Format View Help

```
// config file for HTML Tidy
indent: auto
indent-spaces: 2
wrap: 52
markup: yes
output-xml: no
input-xml: no
show-warnings: yes
numeric-entities: yes
quote-marks: yes
quote-nbsp: yes
quote-ampersand: no
break-before-br: no
uppercase-tags: no
uppercase-attributes: no
char-encoding: latin1
new-inline-tags: cfif, cfelse, math, mroot,
  mrow, mi, mn, mo, msqrt, mfrac, msubsup, munderover,
  munder, mover, mmultiscripts, msup, msub, mtext,
  mprescripts, mtable, mtr, mtd, mth
new-blocklevel-tags: cfoutput, cfquery
new-empty-tags: cfelse
drop-empty-elements:yes
```

Figure 5.4: HTML Purifier Code Snippet

We have shown some of the configuration parameters options in the Table 5.3.

Table 5.3: Tidy Configuration Parameters Example

Parameters	Description
doctype	Type: **DocType** Default: **auto** Example: **html5, omit, auto, strict, transitional, user**
drop-empty-elements	Type: **Boolean** Default: **yes** Example: **y/n, yes/no, t/f, true/false, 1/0**
drop-font-tags	Type: **Boolean** Default: **no** Example: **y/n, yes/no, t/f, true/false, 1/0**
fix-uri	Type: **Boolean** Default: **yes** Example: **y/n, yes/no, t/f, true/false, 1/0**
merge-emphasis	Type: **Boolean** Default: **yes** Example: **y/n, yes/no, t/f, true/false, 1/0**

We have also optimized the purifier not to filter certain HTML coding so that it can be escaped in the cleanup. As shown in Figure 5.6 code fragment we have tuned the purifier on and off for certain tags in the cleanup process. The *br@clear* fix, ensuring that *<br clear="both" />* will pass through unharmed in the purification; whereas, in the case of *p@align* all purifications are turned off, except for the *p@align* one.

The HTML purifier takes an arbitrary snippet of HTML and rigorously test, validate and filter it into a version that can be easily parsed. The steps are Lexing (parsing into tokens) the document, executing various strategies on the tokens, removing all elements not in the whitelist, making the tokens well-formed, fixing the nesting of the nodes, validating attributes of the nodes, and generating HTML from the purified tokens. We invoked the purifier library with our optimized configuration as shown in Figure 5.6 and additional code fragment is attached in Appendix C and Appendix E.

```
// For tests/multitest.php, which versions to test?
$versions_to_test    = array();

// Stable PHP binary to use when invoking maintenance scripts.
$php = 'php';

// For tests/multitest.php, what is the multi-version executable? It must
// accept an extra parameter (version number) before all other arguments
$phpv = false;

// Should PEAR tests be run? If you've got a valid PEAR installation, set this
// to true (or, if it's not in the include path, to its install directory).
$GLOBALS['HTMLPurifierTest']['PEAR'] = false;

// If PEAR is enabled, what PEAR tests should be run? (Note: you will
// need to ensure these libraries are installed)
$GLOBALS['HTMLPurifierTest']['Net_IDNA2'] = true;

// vim: et sw=4 sts=4

$config->set('HTML.Doctype', 'XHTML 1.0 Transitional');
$config->set('HTML.TidyLevel', 'medium'); // all changes, minus...
$config->set('HTML.TidyRemove', 'br@clear');

$config->set('HTML.Doctype', 'XHTML 1.0 Transitional');
$config->set('HTML.TidyLevel', 'none'); // no changes, plus...
$config->set('HTML.TidyAdd', 'p@align');
```

Figure 5.5: Purifier optimization code fragment

5.4 Parser

The parser component is the core part of the self-voicing browser model which is responsible for traversing a cleaned webpage purified by HTML purifier and generating text transcode ready format. It is responsible for reading a webpage by keeping the content organization in order of appearance. It preserves the organization of the webpage content when formatting does not work, as in the case of visually-impaired users. Apart from the ability to arrange webpage content in a meaningful organization, it is also desirable to control which text elements are to be read or opted out. This control over what to be read is also another benefit of the mapping from normal webpage content to the hierarchical structured one.

It works by traversing through the markup tags and generating information contained within the markup tags. The DOM tree represents an HTML document as a tree of node objects. With any tree structure, one of the most common things to do is traverse the tree, examining each node of the tree in turn. However, there are major differences between visually impaired users visualizing the webpage and their sighted counterparts.

To create the same level of visualization of the webpage, we adhere the parser engine as much as possible to the spatial organization of information. Due to the serial nature of access technologies for the visually-impaired, only a single point in the audio stream is viewable at any given time. Therefore, it is difficult to establish the page structure without examining the entire page. The content is presented in a sequential manner, causing problems for multi-column presentations. A visual reader is rather adept at assimilating large amounts of information at the same time. The human eye is expertly capable of quickly scanning through the document to establish those items deemed as being important. This is achieved through examining the spatial organization of elements and through visual cues, e.g., changes in font size, color and emphasis. For example, many websites position navigation links to the left of the main content in multi-columned displays. Also, a section heading may have a different color or font size, making it stand out from the surrounding text. With one quick glance, the visual reader can often identify the starting point of the required content and re-adjust the focus accordingly. Reading a line of text based directly on its screen position would include all of the columns in the order in which they appear [21].

We have used Simple HTML DOM parser built in PHP 5. It supports invalid HTML, find specific tags on HTML page with selectors, extracts content from HTML in a single line, and creates DOM from a URL or file, dumping content from DOM tree. The DOM extension allows us to operate on XML documents through the DOM API in PHP 5. The DOM extension uses UTF-8 encoding. We have used *utf8_encode()* and *utf8_decode()* to work with texts in ISO-8859-1 encoding or *Iconv* for other encodings. The parser generates DOM tree of each HTML tag content on a unique tag *"<text>* and snippet of code is attached in Appendix E.

A Web page is parsed into a tree of nodes called the DOM. The base class for all nodes in the tree is node. The algorithm to generate DOM tree is shown in Figure 5.7. The nodes are further broken down into node types that are relevant to the rendering which are *document, element, and text*.

- **Document**-The root of the tree is always the document. There are three document classes, document, HTML document and Scalable Vector Graphics (SVG) document. The first is used for all XML documents other than SVG documents. HTML documents and SVG inherits from document class.
- **Elements** are the tags that occur in HTML or XML source converted into elements. From a rendering perspective, an element is a node with a tag name that can be used to cast to a specific subclass that can be queried for data that the renderer needs.
- **Text** is a raw of text that occurs in between elements converted into text nodes. Text nodes store raw text, and the render tree can query the node for its character data.

```
For a webpage
        Generate DOM tree
                Check node type
                        If node is document
                                Assign node type as document
                        Else If node is element
                                Assign node type as element
                        Else node is text
                                Assign node type text
End for
```

Figure 5.6: Algorithm for DOM tree

The Renderer Tree

The render tree is very similar to the DOM in that it is a tree of objects, where each object can correspond to the document, element or text nodes. The render tree can also contain additional objects that have no corresponding DOM node. Renderers are created through a process on the DOM called *attachment* and the algorithm for the renderer tree is shown in Figure 5.8. As a document is parsed and DOM nodes are added, a method called attach gets called on the DOM nodes to create the renderers.

```
For DOM tree
        Generate Render tree
                Check node type
                        If node is document
                                Attach document to the tree
                        Else If node is element
                                Attach element to the tree
                        Else node is text
                                Attach text to the tree by amending <text><text> tag
End for
```

Figure 5.7: Algorithm for Renderer

Accessing Style Information

During attachment of the DOM to the renderer tree CSS is queried to obtain style information for an element. The resultant information is stored in an object called a render style. Every single CSS property can be queried via this object. If the *display* CSS property for the element is set to *none* or if the node is a descendant of an element with *display: none* set, then no renderer will be created. The subclass of the node and the CSS display property value are used together to determine what kind of renderer to make for the node. Attach is a top down recursive operation. A parent node will always have its renderer created before any of its descendants will have their renderers created. Then, we have detached all styling from the renderer generated tree. The algorithm to extract the textual content from renderer tree and CSS removal from render style is shown in Figure 5.9.

```
For each Renderer tree
        Find tag with <text><text>
                Read text and assign to text collection array
End for
For each Render Style Object
        Remove CSS
End for
```

Figure 5.8: Algorithm for accessing and detaching CSS

5.5 Text Transcoder

Text transcoder converts a web page into its IPA equivalent text-only version by eliminating all page layout that is present in the original page. Each character of the text is converted to its equivalent IPA representation by looking up from IPA defined list of Ethiopic scripts. The IPA equivalent representation of each Ethiopic script characters is stored in a flat file. It can also be used by the web developer to determine if the reading order of the information presented in the page makes sense when read in the order that would be followed by a screen reader or speech browser. In this section we discuss text transcoder responsibility in generating a readable IPA representation file to be used by the TTS engine.

It accepts the renderer tree generated by the parser and generates IPA text-based version of the webpage by reading the content from the "<text>" tag. The IPA text transcoder look up and convert each character of Ethiopic scripts to its equivalent IPA representation. We have implemented our text transcoder with *JQuery.parseHTML()* API to traverse and transcode a string of text from an array of DOM nodes. The JQuery library does not run script in the parsed HTML as we turned off "*keepScripts"* explicitly false. The pseudo code that traverses recursively through the parsed DOM and search for a tag name <text><text> and convert each character of text to its IPA equivalent representation as shown in Figure 5.9. The IPA text transcoder takes the parsed webpage as an input and generate an IPA equivalent representation of each character for the respective language. The output of the IPA text transcoder is fed into TTS engine capable of synthesizing speech from IPA generated list.

Input:
The IPA text transcoder accepts a parsed webpage.
Output:
List of IPA equivalent characters tagged with <phoneme alphabet="ipa" ph=""></phoneme>

1. Accepts a parsed webpage.
2. Traverse each DOM tree for a tag name <text><text>
3. Retrieve the word and split each character to convert it to IPA equivalent.
4. Read the IPA flat file which defines each character IPA equivalent
5. Compare each character from the IPA list and replace it with its IPA equivalent when matches occur.
6. After each character word is converted to IPA equivalent, assign the converted IPA word in a "ph" value of <phoneme alphabet="ipa" ph=""></phoneme> tag.

Figure 5.9: Algorithm for IPA text transcoder

The resultant textual content is the converted IPA webpage that is to be used by the TTS engine. The converted IPA webpage as shown in Figure 5.10 is a stream of IPA equivalent texts generated by the IPA text transcoder. The IPA representation of each word is defined with a tag name *"<phoneme alphabet="" ph=""></phoneme>"* comprises of two attributes value. The *"ph"* value stores the equivalent IPA representation of the word extracted by the text transcoder. The idea of transcoding each word of the Ethiopic script with the tag name *"<phoneme alphabet="" ph=""></phoneme>"* in IPA equivalent is inherited from the work of AT&T Labs and Research which they demonstrate an IPA based TTS [45].

```
<phoneme alphabet="ipa" ph="marəkə"></phoneme>
<phoneme alphabet="ipa" ph="səpitəzə"></phoneme>
<phoneme alphabet="ipa" ph="yätäbaläwə"></phoneme>
<phoneme alphabet="ipa" ph="'ämerikawi"></phoneme>
<phoneme alphabet="ipa" ph="səporətäña"></phoneme>
<phoneme alphabet="ipa" ph="bäzämänu"></phoneme>
<phoneme alphabet="ipa" ph="səpitəzə"></phoneme>
<phoneme alphabet="ipa" ph="bä'alämačənə"></phoneme>
<phoneme alphabet="ipa" ph="yäwät'alätə"></phoneme>
<phoneme alphabet="ipa" ph="wanatäña"></phoneme>
<phoneme alphabet="ipa" ph="näbärə"></phoneme>
<phoneme alphabet="ipa" ph="səpitəzə"></phoneme>
<phoneme alphabet="ipa" ph="səpitəzə"></phoneme>
<phoneme alphabet="ipa" ph="bämunikə"></phoneme>
<phoneme alphabet="ipa" ph="'oloməpikə"></phoneme>
<phoneme alphabet="ipa" ph="säbatə"></phoneme>
<phoneme alphabet="ipa" ph="yäwärəqə"></phoneme>
<phoneme alphabet="ipa" ph="medaliyawočənə"></phoneme>
<phoneme alphabet="ipa" ph="bämagəñätə"></phoneme>
<phoneme alphabet="ipa" ph="tarikə"></phoneme>
<phoneme alphabet="ipa" ph="yäsəra"></phoneme>
<phoneme alphabet="ipa" ph="näwə"></phoneme>
<phoneme alphabet="ipa" ph="wanatäñawə"></phoneme>
<phoneme alphabet="ipa" ph="säbatäñawanə"></phoneme>
<phoneme alphabet="ipa" ph="yäwärəqə"></phoneme>
<phoneme alphabet="ipa" ph="medaliya"></phoneme>
<phoneme alphabet="ipa" ph="yäwäsädäwə"></phoneme>
<phoneme alphabet="ipa" ph="yäzare"></phoneme>
<phoneme alphabet="ipa" ph="41"></phoneme>
<phoneme alphabet="ipa" ph="'amätə"></phoneme>
<phoneme alphabet="ipa" ph="bäzarewa"></phoneme>
<phoneme alphabet="ipa" ph="'ələtə"></phoneme>
<phoneme alphabet="ipa" ph="näbärə"></phoneme>
<phoneme alphabet="ipa" ph="'əsəkä"></phoneme>
<phoneme alphabet="ipa" ph="mekəsiko"></phoneme>
<phoneme alphabet="ipa" ph="'oloməpikə"></phoneme>
<phoneme alphabet="ipa" ph="mädaräša"></phoneme>
<phoneme alphabet="ipa" ph="bäwəha"></phoneme>
<phoneme alphabet="ipa" ph="wana"></phoneme>
<phoneme alphabet="ipa" ph="yätäläyayu"></phoneme>
```

Figure 5.10: Converted IPA Webpage

5.6 Summary

In this Chapter, we have discussed the implementation techniques and methods applied to build the design of Multilingual Self-Voicing Browser Model for Ethiopic-Scripts. We have discussed each component implementation detail and libraries used with the reason why we have chosen them. The model is implemented with five core components. The user interface is implemented in a responsive mode to automatically detect the targeting device and render on the best way it fits. The user interface provides a field to type in the webpage address. The HTML purifier is

responsible for cleaning the webpage document from any unnecessary code and HTML tags not coded in accordance with W3C's specifications. The parser is responsible for generating DOM tree of the webpage by formatting visual layouts, images, and Java Scripts. IPA text transcoder is responsible to generate an IPA text-based version of the webpage from parsed HTML tag values written in the unique tag name *"<**text**>"*. The result of the IPA text transcoder is a converted IPA webpage which is a collection of words generated from each DOM elements.

CHAPTER SIX-EVALUATION AND TESTING

This Chapter focuses on the evaluation of our model, performance of each component in our model, reliability and usability. Self-voicing browser will be evaluated towards its effectiveness, mainly accessibility and usability with precision computations. The first task we have performed was preparation of webpages from news portal website equipped with Amharic language. We have evaluated each pages for accessibility by A-Checker and Functional Accessibility Evaluator (FAE) tool. The websites selected are Ethiopianreporter.com, Shegerfm.com, and Waltainfo.com. We have analyzed the accessibility result by giving it accessible and usability level percentage.

Initially, we have run the accessibility check on A-Checker and FAE for 30 pages attached in Appendix A collected from Amharic equipped websites. Based on the result we have given accuracy score for proper HTML coding of the webpages. Then we evaluated each webpage accuracy level by each component of the model. The performance level of our parser component in the model is believed to be affected by the webpage proper HTML encoding even after HTML purification. After we perform each webpage accessibility test in A-Checker and FAE, we cleaned it with HTML purifier component. We have found meaningful improvement in the webpage accessibility score as we rerun each webpage test in the A-Checker again after purification. Then, we have evaluated the webpage with the parser component to generate DOM of the webpage content. The result shows that the webpage content is well parsed by Simple HTML DOM parser library. Finally, the transcoded webpage is evaluated against parsed DOM tree if there is any opted out DOM element.

The performance of the model is measured by HTML content accuracy (precision), i.e., the fraction of HTML tag content correctly parsed or transcoded by the model.

6.1 Webpage Preparation

Webpages are the main sources in evaluating our model. For this purpose we have collected over 30 webpages URL from selected Amharic equipped news portal websites. We have manually picked 10 URL of webpages from each of the three websites (Ethiopianreporter.com, Shegerfm.com, and Waltainfo.com). We have attached the sample website URL in Appendix B.

6.2 Evaluation Criteria

Evaluation of self-voicing browser model mainly focuses on the accuracy level of properly parsing and transcoding of the webpage to fetch all content that resides in the webpage. We mainly evaluated our model for accuracy of parsing and transcoding. The accuracy level is calculated by counting the number of HTML tag elements of the original webpage and number of corresponding generated parsed tag elements. The precision for parser is calculated as the number of correctly parsed HTML tags over the total number of HTML tags. For the case of text transcoder, the precision is calculated as the number of correctly transcoded HTML tag elements over the total number of parsed HTML tag elements.

$$\text{Precision (HTML Purifier)} = \frac{correctly\ purified\ HTML\ tags}{properly\ coded\ tags + Improperly\ coded\ tags}$$

$$\text{Precision (parser)} = \frac{correctly\ parsed\ HTML\ tags}{properly\ purified\ tags + Improperly\ purified\ tags}$$

$$\text{Precision (transcoder)} = \frac{correctly\ \ transcoded\ HTML\ tags}{properly\ parsed\ tags + Improperly\ parsed\ tags}$$

6.3 A-Checker and FAE Test

We have performed accessibility review of each webpage on A-Checker and FAE. A-Checker is an online accessibility checker that tests webpages for conformance to various accessibility guidelines. It provides options to test against HTML validator, CSS validator, Stanca Act, WCAG 1.0, Section 508 etc. The FAE analyzes web resources for markup that is consistent with the use of DRES/CITES HTML best practices for development of functionally accessible web resources and resources that support interoperability. The HTML best practices are not a new standard, but rather a statement of techniques for implementation of the W3C WCAG and United States Federal Government Section 508 standards that not only improve accessibility for people with disabilities, but also the interoperability of web resources for everyone so all people benefit by having more options to access and use web resources [34].

Results show that overall, 85% of the webpages placed for test requires big room for improvement on all rated dimensions of accessibility. Very few webpages assessed on any criterion received a "pass" rating. As it can be seen in Figure 6.1, one webpage evaluation result show that it has a potential improper HTML coding of 111 problems with 2 known problems of not giving alternative text for images and title for link.

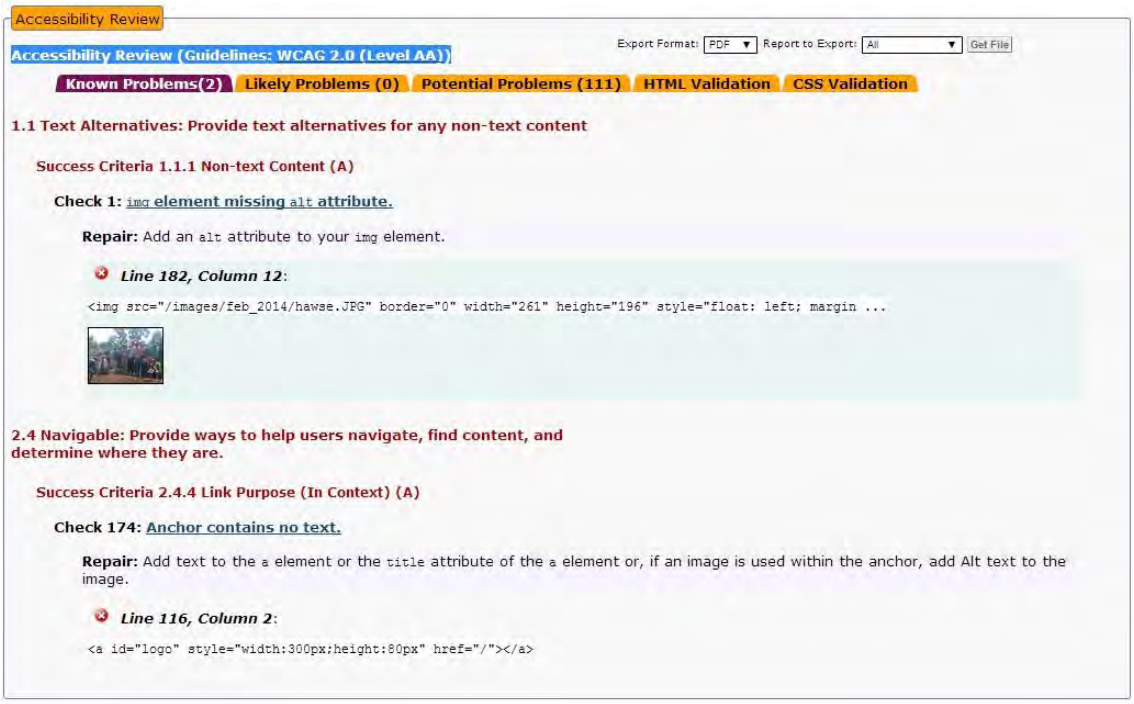

Figure 6.1: A-Checker before HTML Purification

The FAE evaluates a given webpage functional accessibility. It organizes the analysis of webpages based on *navigation and orientation, text equivalents, scripting, and styling* categories. Based on the FAE evaluation shown on Table 6.1, it shows evaluation result of pass, warn, and fail for each category.

- **Navigation & Orientation:** Inclusion of structural markup that facilitates navigation and contextual orientation;
- **Text Equivalents:** Proper use of images for interoperability and the provision of text descriptions for non-text content;
- **Scripting:** Avoidance of scripting techniques that compromise accessibility and interoperability;

- **Styling:** Use of CSS styling techniques to separate content and structural information from styling and presentation;

Table 6.1: FAE Webpage Run Test Summary Result

Category	Status	% Pass	%Warn	%Fail
Navigation & Orientation	Partially Implemented	70	11	11
Text Equivalents	Partially Implemented	75	0	25
Scripting	Complete	100	0	0
Styling	Almost Complete	83	16	0
HTML Standards	Complete	100	0	0

Then, we have passed each webpages to the HTML purifier to improve the accessibility. We run the test for the second time after purification of the webpages and the result shows that it made significant improvement. The accessibility rate improves by reducing the potential error in HTML from 111 to 22 and the known problems such as giving an alternative text for image and title for a link from 2 to 0 as shown in Figure 6.2. Based on the HTML purification we are able to achieve 19.81% of HTML cleanness than the original one. This enables webpages to be well formatted for the parser to generate DOM tree more accurately.

Figure 6.2: A-Checker after HTML Purification

We can see from Table 6.2 that webpages precision has made meaningful improvement after cleaning the webpage for improper coding of tags, removing non-whitelisted elements, checking

the well-formedness and nesting of tags, and validating all attributes. The webpages HTML coding standards has been improved by 0.72 or 72% after HTML purification.

Table 6.2: HTML Purification Evaluation

Website	**Before HTML Purification Precision**	**After HTML Purification Precision**
http://www.Waltainfo.com	0.24	0.64
http://www.Ethiopianreporter.com	0.35	0.76
http://www.Shegerfm.com	0.26	0.76
Average	**0.28**	**0.72**

6.4 Parser Test

The next evaluation we performed is parser test by passing each webpage purified to examine the generated DOM tree. The parsed HTML page is bounded by each textual content in a unique tag *"<text><text>"* as shown in Figure 6.3 shows the result of the parsed DOM tree in our model.

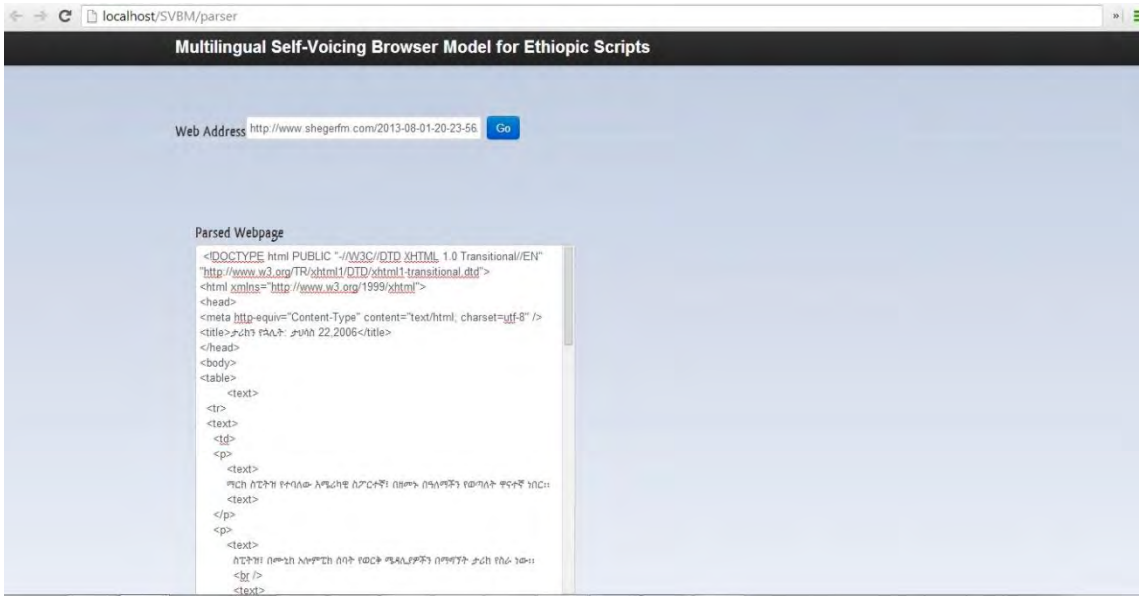

Figure 6.3: Screenshot of parsed webpage

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>ታሪክን የጎሊት: ታህሳስ 22,2006</title>
</head>
<body>
<table>
        <text>
 <tr>
 <text>
  <td>
  <p>
    <text>
    ማርክ ስፒትዝ የተባለው አሜሪካዊ ስፖርተኛ፣ በዘመኑ በዓለማቹን የመጣለት ዋናተኛ ነበር።።
    <text>
  </p>
  <p>
    <text>
     ስፒትዝ፣ በሙኒክ አሉምፒክ ሰባት የወርቅ ሜዳሊያዎቹን በማግኘት ታሪክ የስራ ነው።።
      <br />
      <text>
     ዋናተኛው ሰባቱንዋን     የወርቅ ሜዳሊያ የወሰደው የዛሬ 41  ዓመት በዛሬዋ እለት ነበር።።
      <text>
    <text>
  </p>
   <p>
     <text>
     እስከ ሜክሲኮ አሉምፒክ መዳረሻ፣ በውሃ ዋና የተሳያዩ ዘርፎች በርከት ያሉ ክብረ ወሰኖች በእጁ ላይ ነበሩ።።
      <text>
   </p>
   <p>
     <text>
     ሜክሲኮ ሲገባ፣ ቢያንስ ቢያንስ በውሃ ዋና፣ ስድስቱን የወርቅ ሜዳሊያዎች በእጄ እስገባለሁ አለ።። በከፊል     ግን ማሳካት ቻለ።።
     <text>
   </p>
   <p>
     <text>
     ሁለት የወርቅ፣አንድ የብርና አንድ የነሐስ ሜዳሊያ በማግኘት ወደ አገሩ ተመለሰ።።
     <br />
     <text>
     ውጤቱ ቀላል ባይሆንም ውጥኑን ባለማሳካቱ ቁጭት አደረበት።።
     <text>
     <text>
    </p>
   <p>
     <text>
     ከ41 ዓመት በፊት፣ ለቀጣዩ የሙኒክ አሉምፒክ አጠንክሮና በርትቶ ተዘጋጀ።።
     <text>
   </p>
   <p>
     <text>
     እንደ መሠናዶውም ስፒትዝ የሙኒክ ኣሉምፒክ ልዩ ክስተት ሆነ።።
     <br />
     <text>
     ስፒትዝ በዚያ አሉምፒክ ወርቅ በወርቅ ሆነ።። በወርቅ ተንቆጠቆጠ።። በወርቅ አሸበረቀ።።
     <text>
     <br />
     <text>
     ዋናተኛው፣ በሙኒክ አሉምፒክ በተካፈለባቸው ልዩ ልዩ የውሃ ዋና ውድድሮች፣ በቢራቢሮ ነሽ፣ በነፃ ዋና ነሽ     በሌላ በሌላውም ሰባት የወርቅ ሜዳሊያዎችን አግበሰበሰ።።
     <text>
     <text>
    </p>
   <p>
     <text>
     .../ይቀጥላል
     <text>
   </p>
   <text>
 </tr>
 <text>
</table>
</body>
</html>
```

Figure 6.4: Parsed HTML Source

The result reveals that average of 90% of all webpages content is properly parsed and DOM tree generated. Based on the parsing accuracy precision result shown in Table 6.3, the HTML

purification improves the parser accuracy of generating DOM tree. The result shows that we are able to parse and generate DOM tree at parsing accuracy of 0.9 or 90%.

Table 6.3: Parser Evaluation

Website	**Parser Precision**
http://www.Waltainfo.com	0.93
http://www.Ethiopianreporter.com	0.87
http://www.Shegerfm.com	0.91
Average	**0.90**

6.5 Text Transcoder

After parsing we have performed text transcoder evaluation. We evaluated the accuracy of the transcoder for text transcoding from the generated DOM tree. The text transcoder generates text only version of the webpage from parsed "<text></text>" tag as shown in Figure 6.5.

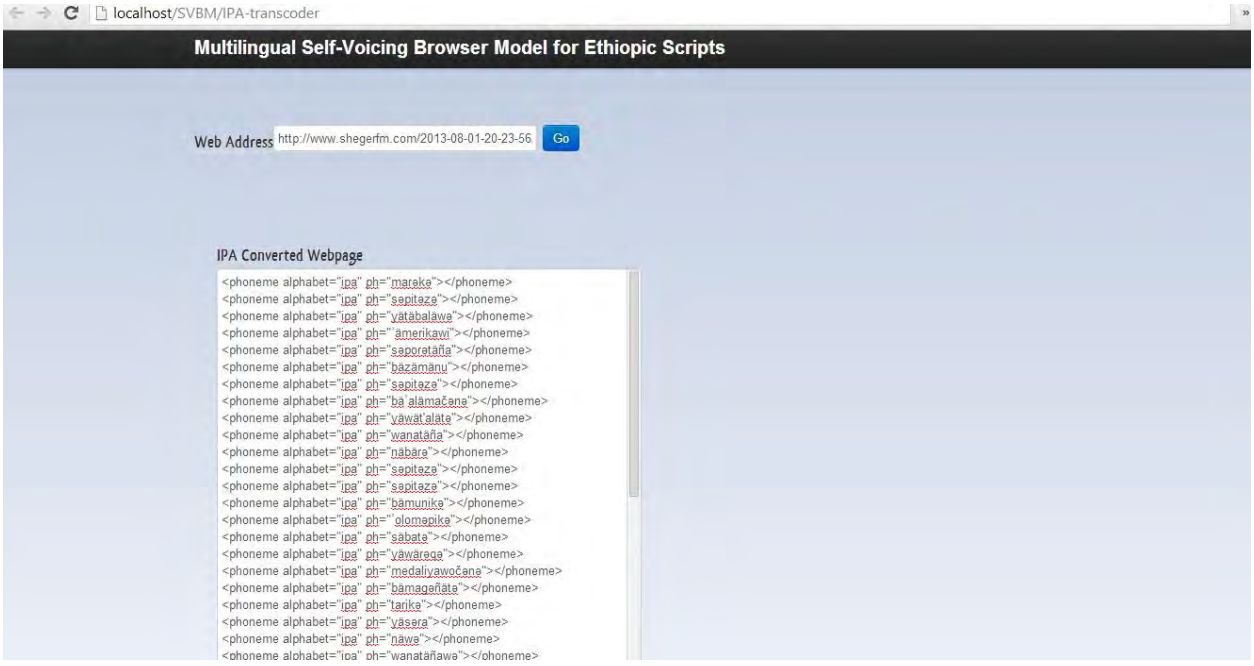

Figure 6.5: Screenshot of IPA transcoded webpage

The result shows that the text transcoder successfully generates all textual content from the DOM tree without missing any tag. The text transcoder accuracy of generating the plain text from parser generated DOM tree hits its objective by properly transcoding all parsed DOM elements. Based on Table 6.4 precision accuracy result, we are able to attain a 100% transcoding accuracy in all the webpages.

Table 6.4: Text Transcoder Evaluation

Website	**Text Transcoder Precision**
http://www.Waltainfo.com	1
http://www.Ethiopianreporter.com	1
http://www.Shegerfm.com	1
Average	**1**

6.6 Discussion

Due to unavailability of TTS library engine to integrate it to our model, we have not performed evaluation on TTS integrated model, though we have evaluated the main components of the model to measure their performance. The main problem we encountered in our evaluation experiment is the webpage HTML purification which counts 85% of total webpages that are not properly HTML coded that takes 15 to 30 seconds to clean up each webpage depending on the page HTML complexity and size. The most expensive and time consuming task of all tasks in the model is the HTML purification. The HTML purifier library filters out all the possible permutations of HTML tags and their attributes to clean up the webpage and provide cleaner version of the webpage for the parser. The parser generates from purified version of the webpage the DOM tree in maximum of 1 seconds from a 500 lines of HTML source code. The text transcoder converts the webpage to a converted IPA webpage by transcoding each “<text>” tag in a maximum of 3 seconds from 500 lines of HTML tags. We also realized the time to return the result by both parser and text transcoder depends on the number of HTML source code lines.

CHAPTER SEVEN-CONCLUSION AND FUTURE WORK

The web offers individuals the ability to compensate for physical or functional limitations, to access knowledge by adapting digital media to the nature of their disabilities, and to enhance their social and economic integration in communities by enlarging the scope of activities available to them. Webpages are documents written in HTML defined by the international web standards body, the W3C. HTML offers feature to represent information or knowledge by combining text information with meaningful semantic mark-up of the text, for example “This is a header” or “this is a list of information”. Having written a document in HTML, the author can provide formatting information for a particular medium using the related CSS. These contain information on how the author wishes the information to be presented that may be used by the client, so a visual style sheet might instruct the client to use a particular font or color combination, and an audio style sheet to use a particular voice or volume [23].

In practice, however, the web is primarily a visual medium, and the principle of separating content and presentation has not been honored. Presentation information has been included in HTML mark-up. This has repercussions for access by visually impaired users. The other main obstacle on the use of webpages by visually impaired people is the embedding of non-text content into HTML documents. HTML does provide mechanisms for annotating this embedded content: the specification states that an author can set the ALT attribute of an embedded content element to describe the image as text or indicate that the content is of no value to a visually impaired people, for instance, an image used as a background spacing element.

As a result surfing the web for visually impaired people, providing equal platform on using the web becomes an active research area. Solutions to the problem of web accessibility for the visually impaired people fall into one of four categories: reliance on a conventional web browser and a screen reader; utilizing the accessibility features of HTML and existing web clients; using transcoding proxy servers to convert webpage HTML into a more accessible format; and self-voicing browser. Self-voicing browser is web-based application that enables visually impaired people to access the web from almost any computer that can produce sound without installing additional software.

7.1 Conclusions

In this thesis we have designed and modeled a multilingual self-voicing browser model for Ethiopic scripts that is universally and ubiquitously accessible web browsing application. It can also make web browsing more efficient and make visually impaired people more mobile. As there has not been any work done before on self-voicing browser model for Ethiopic scripts, we have reviewed works on other languages mainly for the English language. In addition, there has not been any multilingual base self-voicing browser model done for any other languages to the best of our knowledge. The model is designed as a web based application which accepts the webpage URL. After accepting the webpage URL, the HTML purifier library component cleans up the webpage in accordance with W3C standard compliant. The parser takes a purified webpage and generates a DOM tree representation of the webpage by removing all presentation elements and placing each content to be transcoded in the unique tag name "<text>". The parser removes all inline and embedded style sheets, scripts, link converted to the title attribute, and images replaced with alternative text. Then, the IPA text transcoder traverses through the parsed tree and generates a DOM tree IPA equivalent representation of the words to be used by IPA based TTS engine. The TTS library engine comprises of multiple language bases of the respective Ethiopic scripts.

The evaluation of our model, being the first multilingual self-voicing browser model for Ethiopic scripts, shows meaningful performance. We have evaluated our model with webpages for HTML validation, CSS validation, WCAG 1.0, Section 508 W3C standard complaint after webpage has been purified by HTML purification library which we are able to attain 19.81% improved standard complaint score. Based on the precision score evaluation, webpages has scored a performance of W3C standard compliant of 28% before HTML purification and 72% after HTML purification. The result tells us that webpage HTML purification has made a significant difference on validating and correcting webpages. When we see the evaluation result of the parser, it has a precision score of 90%. Regarding, the IPA text transcoder precision score of accurately transcoding each DOM tree elements is the hit with a score of 100%.

7.2 Contribution

The main contribution of this thesis work are abridged as follows:

- The study has adopted the work in self-voicing web based application system from English language to the Ethiopic scripts.
- The study has cemented in setting up a ground model on building a full-fledged multilingual self-voicing browser model for Ethiopic scripts.
- The study showed the strategy, algorithms, and techniques in developing multilingual self-voicing browser model.
- The study has designed multilingual model for self-voicing browser that is language independent.
- The study identified and showed the major components to construct a self-voicing browser model and IPA based TTS library engine and how they can be seamlessly integrated and function as a unit.
- The study identified basic challenges in developing and implementing multilingual self-voicing browser model for Ethiopic scripts and the possible strategies to solve those challenges.

7.3 Future Work

Designing and developing multilingual self-voicing browser model is a difficult task, which demands more time and expertize of human as this is one big task for one person to deal. We have identified numerous rooms of improvement on making the model full-fledged and accessible for visually impaired people on surfing the web by themselves without any help. The following are some of the recommendations that we propose for future work.

- Developing HTML purifier library with a better HTML validator performance. The HTML purifier we have used in the model only improved by 44% and we need to develop a library which can give a better performance.
- Extending the model and designing speech interactive component that accepts URL and page encoding by spoken command so that visually impaired people can give the input in speech than typing through keyboard.

- Incorporating combinational keyboard shortcuts similarly provided by popular screen readers.
- Developing webpage summarizer that sums up to provide summary and keywords obtained by analyzing the webpage structure. This will give high level descriptive summarization of the webpage to help them decide to continue listen the webpage content or looking for other webpage.
- Developing hierarchical parsing tool that can traverse by keeping the page structure hierarchy.
- Developing an automatic identifier of webpage encoding language specifically webpages equipped with Ethiopic scripts.
- Developing Flash/Java/multimedia embedded content parsing and transcoding tool.
- Developing a component which handles online forms.
- Creating a playlist of webpages to allow them listen at a later time from anywhere. It means that whenever they found a webpage that caches their interest, they can add it to a playlist of webpages and can access the playlist from anywhere.
- Setting up user interaction tracking log to learn user navigation and usage.

References

[1] Jim Thatcher, Phill Jenkins, and Cathy Laws, “IBM Special Needs Self Voicing Browser.”

[2] Adaptive Technology Center for Blind (ATCB): http://www3.sympatico.ca/, last accessed on February 17, 2013.

[3] “Types of screen reader”, http://www.tutorgigpedia.com/ed/Screen_reader, last accessed on February 17, 2013.

[4] American Foundation for the Blind web site, http://www.afb.org, last accessed on February 17, 2013.

[5] John Borland, “Opera releases talking Web browser”, http://news.cnet.com/Opera-releases-talking-Web-browser/2100-1032_3-5502431.html, last accessed on February 17, 2013.

[6] Jakob Nielsen, "Beyond Accessibility: Treating Users with Disabilities as People", 2001, http://www.nngroup.com/articles/beyond-accessibility-treating-users-with-disabilities-aspeople/, last accessed on October 13, 2012.

[7] Peter Brophy and Jenny Craven,” Web Accessibility”, Library and Information Services for Visually Impaired People, pp. 950–972, 2007.

[8] Tim Berners-Lee, “Accessibility”, http://www.w3.org/standards/webdesign/accessibility, last accessed on October 13, 2012.

[9] “Web Accessibility Initiative”, http://www.w3.org/wai, last accessed on October 13, 2012.

[10] “Section 508 Standards”, http://www.section508.gov, last accessed on October 13, 2012.

[11] John M. Slatin and Sahrron Rush, “Maximum Accessibility: Making Your Web Site More Usable for Everyone”, 2002, Addison-Wesley Professional.

[12] Rolf Carlson and James Glass, “Vowel Classification Based on Analysis-by-Synthesis”, International Conference on Spoken Language Processing, Banff, Canada, pp 33-46, 2004.

[13] Sami Lemmetty, “Review of Speech Synthesis Technology”, Unpublished Master’s Thesis, Helsinki University of Technology, 1999.

[14] Minghui Dong, Kim-Teng Lua, and Haizhou Li, “A Unit Selection-Based Speech Synthesis Approach for Mandarin Chinese”, Journal of Chinese Language and Computing 16 (3): pp 135-144, 2006.

[15] Maria Moutran Assaf, “A Prototype of an Arabic Diphone Speech Synthesizer in Festival”, Unpublished Master’s Thesis, Uppsala University, 2005.

[16] Nadew Tademe, “Formant Based Speech Synthesis for Amharic Vowels”, Unpublished Master’s Thesis, Addis Ababa University, 2008.

[17] Tadesse Anberbir and Tomio Takara, “Development of an Amharic Text-to-Speech System Using Cepstral Method”, Proceedings of the EACL 2009 Workshop on Language Technologies for African Languages AfLaT2009, pp 46–52, Athens, Greece, 31 March 2009.

[18] Tadesse Anberbir, Michael Gasser, Tomio Takara, and Kim Dong Yoon “Grapheme-to-Phoneme Conversion for Amharic Text-to-Speech System”, Proceedings of Conference on Human Language Technology for Development, pp 68-73, Alexandria, Egypt, 2011.

[19] Alasdair King, Gareth Evans, and Paul Blenkhorn“Blind people and the World Wide Web”, http://www.webbie.org.uk/webbie.htm, last accessed on February 24, 2013.

[20] Thomas C. Weiss, “Homer-Voicing Web Browser for the Blind”, http://www.disabled-world.com/assistivedevices/computer/homer-web-browser.php, last accessed on February 24, 2013.

[21] Esmond Walshe,”An Alternative Audio Web Browsing Solution: Viewing Web Documents Through a Tree Structural Approach”, 2006, Dublin City University.

[22] Yevgen Borodin, Andrii Sovyak, Alexander Dimitriyadi, Yury Puzis,Valentyn Melnyk Faisal Ahmed, Glenn Dausch, and I.V. Ramakrishnan, ”Universal and Ubiquitous Web Access with Capti”, Co-Located with the 21st International World Wide Web Conference, Lyon, France, April 16-17, 2011.

[23] A. King, G. Evans and P. Blenkhorn,”WebbIE: A Web Browser for Visually Impaired People.” The 2nd Cambridge Workshop on Universal Access and Assistive Technology (CWUAAT), pp 35-44, United Kingdom, 2004.

[24] Jeffrey P. Bigham, Craig M. Prince, and Richard E. Ladner, “WebAnywhere: Enabling a Screen Reading Interface for the Web on Any Computer”, ACM, Beijing, China, 2008.

[25] “Text Transcoder”,http://www.htctu.net/divisions/webaccess/transcoder/transcodermain.htm, last accessed on October 17, 2013.

[26] Yevgen Borodin, Jeffrey P. Bigham, Amanda Stent, and I.V. Ramakrishnan, “Towards One World Web with HearSay3”, W4A '08 Proceedings of the 2008 international cross-disciplinary conference on Web accessibility (W4A), pp 130-131, ACM, New York, USA, 2008.

[27] “HTML Purifier”, http://www.htmlpurifier.org/, last accessed on October 28, 2013.

[28] Helen Petrie, Omar Kehir "The Relationship between Accessibility and Usability of Websites", CHI 2007 Proceedings, San Jose, CA, USA, April 28-May 03, 2007.

[29] "World Report on Disability", http://www.who.int/disabilities/world_report/2011/en/, last accessed on February 17, 2013.

[30] "Market Share Statistics for Internet Technologies", http://www.netmarketshare.com/browser-market-share.aspx?qprid=0&qpcustomd=0, last accessed on March 23, 2013.

[31] "Central Statistical Agency of Ethiopia", http://www.csa.gov.et, last accessed on January 12, 2014.

[32] "IBM Home Page Reader", http://www.tutorgigpedia.com/ed/IBM_Home_Page_Reader, last accessed on February 17, 2013.

[33] "HTML TIDY", http://www.w3.org/People/Raggett/tidy/, last accessed on November 16, 2013.

[34] "HTML Best Practices", http://html.cita.illinois.edu/, last accessed on December 03, 2013.

[35] "National Survey on Blindness, Low Vision and Trachoma in Ethiopia", http://pbunion.org/Countriessurveyresults/Ethiopia/Ethiopian_National_Blindness_and_trachoma_survey.pdf, last accessed on March 09, 2014.

[36] "pwWebSpeak", http://www.talkinginterfaces.org/artifacts/pwwebspeak/, last accessed on March 09, 2014.

[37] Jim Thatcher, Phill Jenkins, and Cathy Laws, "IBM Special Needs Self Voicing Browser", http://www.w3.org/Voice/1998/Workshop/PhilJenkins.html, last accessed on March 09, 2014.

[38] "Freedom Scientific Blind Low Vision Group: Connect Outloud Quick Reference Guide", http://www.freedomscientific.com/fs_support/documentation/ConnectOutloudDocs/ConnectOutloud_QSG.pdf, last accessed on March 09, 2014.

[39] Suhit Gupta, Gail Kaiser, David Neistadt, Peter Grimm "DOM-based Content Extraction of HTML Documents", last accessed on March 13, 2014.

[40] Eda Baykan, Monika Henzinger, Ingmar Weber "Web Page Language Identification Based on URLs", last accessed on April 08,2014.

[41] Bruno Martins and Mário J. Silva "Language Identification in Web Pages", last accessed on April 08,2014.

[42] "Voice Browsers", http://www.w3.org/TR/1998/NOTE-voice-0128, last accessed on April 07, 2014.

[43] “Voice Browser Interoperation: Requirements”, http://www.w3.org/TR/vbi-reqs/, last accessed on April 07, 2014.

[44] “The Voice Browser Working Group”, http://www.w3.org/Voice/, last accessed on April 07, 2014.

[45] “AT&T Natural Voices Text-to-Speech Demo”, http://www2.research.att.com/ttsweb/tts, last accessed on April 07, 2014.

Appendices

Appendix A: Amharic Phonetic List IPA Equivalence and ASCII Transliteration

IPA Equivalent	Transcription	Amharic Scripts
Consonants		
[p]	[p]	ፕ
[t]	[t]	ት
[k]	[k]	ክ
[ʔ]	[ax]	አ
[b]	[b]	ብ
[d]	[d]	ድ
[g]	[g]	ግ
[pʼ]	[px]	ጵ
[tʼ]	[tx]	ጥ
[cʼ]	[cx]	ጭ
[q]	[q]	ቅ
[f]	[f]	ፍ
[s]	[s]	ስ
[ʃ]	[sh]	ሽ
[h]	[h]	ህ
[sʼ]	[sx]	ጽ
[tʃ]	[c]	ች
[gʼ]	[j]	ጅ
[m]	[m]	ም
[n]	[n]	ን
[nʼ]	[nx]	ኝ
[l]	[l]	ል
[r]	[r]	ር
[j]	[y]	ይ
[w]	[w]	ው
[v]	[v]	ቭ
[z]	[z]	ዝ
[zʼ]	[zx]	ዥ
Vowels		
[ə]	[e]	ኧ
[ʊ]	[u]	ኡ
[ɪ]	[ii]	ኢ
[a]	[a]	ኣ
[e]	[ie]	ኤ
[i]	[ix]	እ
[o]	[o]	ኦ

Appendix B: Webpage Address collected for model evaluation

Webpage URL
http://www.waltainfo.com/index.php/2013-07-12-15-28-45/12270-2014-02-10-12-53-29
http://www.waltainfo.com/index.php/2013-07-12-15-28-45/12269--121
http://www.waltainfo.com/index.php/2011-09-07-11-53-43/12282-2014-02-12-06-51-31
http://www.waltainfo.com/index.php/2011-09-07-11-53-43/12272-2014-02-11-04-42-22
http://www.waltainfo.com/index.php/2011-09-07-11-53-43/12261-2014-02-10-06-52-54
http://www.waltainfo.com/index.php/2011-09-07-11-53-43/12254-2014-02-08-19-01-54
http://www.waltainfo.com/index.php/2011-09-07-11-53-43/12219-2014-02-06-05-22-17
http://www.waltainfo.com/index.php/2011-09-07-11-53-43/12168-2014-02-01-07-23-26
http://www.waltainfo.com/index.php/2011-09-07-11-53-43/12155-2014-01-31-11-00-18
http://www.waltainfo.com/index.php/2011-09-07-11-53-43/12143-2014-01-30-08-46-15
http://www.shegerfm.com/2013-08-01-20-23-56/152-22-2006-22-2006-22-2006-22-2006
http://www.shegerfm.com/2013-08-01-20-23-56/153-23-2006-23-2006-23-2006
http://www.shegerfm.com/2013-08-01-20-23-56/154-24-2006-24-2006-24-2006
http://www.shegerfm.com/2013-08-01-20-23-56/155-30-2006-30-2006-30-2006
http://www.shegerfm.com/2013-08-01-20-23-56/156-01-2006-01-2006-01-2006
http://www.shegerfm.com/2013-08-01-20-23-56/145-30-2006-30-2006
http://www.shegerfm.com/2013-08-01-20-23-56/144-24-2006-24-2006
http://www.shegerfm.com/2013-08-01-20-23-56/142-18-2006

http://www.shegerfm.com/2013-08-01-20-23-56/176-03-06-2006
http://www.shegerfm.com/2013-08-01-20-23-56/175-01-06-2006
http://www.ethiopianreporter.com/index.php/news/item/4938
http://www.ethiopianreporter.com/index.php/news/item/4937
http://www.ethiopianreporter.com/index.php/news/item/4936
http://www.ethiopianreporter.com/index.php/news/item/4935
http://www.ethiopianreporter.com/index.php/news/item/4884
http://www.ethiopianreporter.com/index.php/news/item/4883
http://www.ethiopianreporter.com/index.php/news/item/4882
http://www.ethiopianreporter.com/index.php/news/item/4879
http://www.ethiopianreporter.com/index.php/news/item/4878
http://www.ethiopianreporter.com/index.php/news/item/4877

Appendix C: Fragment of the HTMLPurifier.php

```
/**
    * @param HTMLPurifier|HTMLPurifier_Config $prototype Optional prototype
    *             HTMLPurifier instance to overload singleton with,or HTMLPurifier_Config
    *             instance to configure the generated version with.
    *
    * @return HTMLPurifier
    */
   public static function instance($prototype = null)
   {
      if (!self::$instance || $prototype) {
         if ($prototype instanceof HTMLPurifier) {
            self::$instance = $prototype;
         } elseif ($prototype) {
            self::$instance = new HTMLPurifier($prototype);
         } else {
            self::$instance = new HTMLPurifier();
         }
      }
      return self::$instance;
   }

   /**
    * Singleton for enforcing just one HTML Purifier in your system
    *
    * @param HTMLPurifier|HTMLPurifier_Config $prototype Optional prototype
    *             HTMLPurifier instance to overload singleton with,
    *             or HTMLPurifier_Config instance to configure the
    *             generated version with.
    *
    * @return HTMLPurifier
```

Appendix D: Fragment of the simple_html_dom.php

```
// read HTML tag info
    protected function read_tag()
    {
        if ($this->char!=='<')
        {
            $this->root->_[HDOM_INFO_END] = $this->cursor;
            return false;
        }
        $begin_tag_pos = $this->pos;
        $this->char = (++$this->pos<$this->size) ? $this->doc[$this->pos] : null; // next
        // end tag
        if ($this->char==='/')
        {
            $this->char = (++$this->pos<$this->size) ? $this->doc[$this->pos] : null; // next
            // This represents the change in the simple_html_dom trunk from revision 180 to 181.
            // $this->skip($this->token_blank_t);
            $this->skip($this->token_blank);
            $tag = $this->copy_until_char('>');
            // skip attributes in end tag
            if (($pos = strpos($tag, ' '))!==false)
                $tag = substr($tag, 0, $pos);
            $parent_lower = strtolower($this->parent->tag);
            $tag_lower = strtolower($tag);
            if ($parent_lower!==$tag_lower)
            {
                if (isset($this->optional_closing_tags[$parent_lower]) && isset($this-
>block_tags[$tag_lower]))
                {
                    $this->parent->_[HDOM_INFO_END] = 0;
```

```
            $org_parent = $this->parent;
            while (($this->parent->parent) && strtolower($this->parent->tag)!==$tag_lower)
                $this->parent = $this->parent->parent;
            if (strtolower($this->parent->tag)!==$tag_lower) {
                $this->parent = $org_parent; // restore origonal parent
                if ($this->parent->parent) $this->parent = $this->parent->parent;
                $this->parent->_[HDOM_INFO_END] = $this->cursor;
                return $this->as_text_node($tag);
            }
        }
        else if (($this->parent->parent) && isset($this->block_tags[$tag_lower]))
        {
            $this->parent->_[HDOM_INFO_END] = 0;
            $org_parent = $this->parent;

            while (($this->parent->parent) && strtolower($this->parent->tag)!==$tag_lower)
                $this->parent = $this->parent->parent;
            if (strtolower($this->parent->tag)!==$tag_lower)
            {
                $this->parent = $org_parent; // restore origonal parent
                $this->parent->_[HDOM_INFO_END] = $this->cursor;
                return $this->as_text_node($tag);
            }
        }
        else if (($this->parent->parent) && strtolower($this->parent->parent-
>tag)===$tag_lower)
        {
            $this->parent->_[HDOM_INFO_END] = 0;
            $this->parent = $this->parent->parent;
        }
```

```
        else
            return $this->as_text_node($tag);
    }
    $this->parent->_[HDOM_INFO_END] = $this->cursor;
    if ($this->parent->parent) $this->parent = $this->parent->parent;
    $this->char = (++$this->pos<$this->size) ? $this->doc[$this->pos] : null; // next
    return true;
}

$node = new simple_html_dom_node($this);
$node->_[HDOM_INFO_BEGIN] = $this->cursor;
++$this->cursor;
$tag = $this->copy_until($this->token_slash);
$node->tag_start = $begin_tag_pos;

// doctype, cdata & comments...
if (isset($tag[0]) && $tag[0]==='!') {
    $node->_[HDOM_INFO_TEXT] = '<' . $tag . $this->copy_until_char('>');

    if (isset($tag[2]) && $tag[1]==='-' && $tag[2]==='-') {
        $node->nodetype = HDOM_TYPE_COMMENT;
        $node->tag = 'comment';
    } else {
        $node->nodetype = HDOM_TYPE_UNKNOWN;
        $node->tag = 'unknown';
    }
    if ($this->char==='>') $node->_[HDOM_INFO_TEXT].='>';
    $this->link_nodes($node, true);
    $this->char = (++$this->pos<$this->size) ? $this->doc[$this->pos] : null; // next
    return true;
```

```
}
// text
if ($pos=strpos($tag, '<')!==false) {
    $tag = '<' . substr($tag, 0, -1);
    $node->_[HDOM_INFO_TEXT] = $tag;
    $node->append('<text><text>');
    $this->link_nodes($node, false);
    $this->char = $this->doc[--$this->pos]; // prev
    return true;
}
if (!preg_match("/^[\w-:]+$/", $tag)) {
    $node->_[HDOM_INFO_TEXT] = '<' . $tag . $this->copy_until('<>');
    if ($this->char==='<') {
        $this->link_nodes($node, false);
        return true;
    }
    if ($this->char==='>') $node->_[HDOM_INFO_TEXT].='>';
    $this->link_nodes($node, false);
    $this->char = (++$this->pos<$this->size) ? $this->doc[$this->pos] : null; // next
    return true;
}
// begin tag
$node->nodetype = HDOM_TYPE_ELEMENT;
$tag_lower = strtolower($tag);
$node->tag = ($this->lowercase) ? $tag_lower : $tag;
// handle optional closing tags
if (isset($this->optional_closing_tags[$tag_lower]) )
{
    while (isset($this->optional_closing_tags[$tag_lower][strtolower($this->parent->tag)]))
    {
```

```
            $this->parent->_[HDOM_INFO_END] = 0;
            $this->parent = $this->parent->parent;
        }
        $node->parent = $this->parent;
    }
    $guard = 0; // prevent infinity loop
    $space = array($this->copy_skip($this->token_blank), '', '');
    // attributes
    do
    {
        if ($this->char!==null && $space[0]==='')
        {
            break;
        }
        $name = $this->copy_until($this->token_equal);
        if ($guard===$this->pos)
        {
            $this->char = (++$this->pos<$this->size) ? $this->doc[$this->pos] : null; // next
            continue;
        }
        $guard = $this->pos;
        // handle endless '<'
        if ($this->pos>=$this->size-1 && $this->char!=='>') {
            $node->nodetype = HDOM_TYPE_TEXT;
            $node->_[HDOM_INFO_END] = 0;
            $node->_[HDOM_INFO_TEXT] = '<'.$tag . $space[0] . $name;
            $node->tag = 'text';
            $this->link_nodes($node, false);
            return true;
        }
```

```
        // handle mismatch '<'
        if ($this->doc[$this->pos-1]=='<') {
            $node->nodetype = HDOM_TYPE_TEXT;
            $node->tag = 'text';
            $node->attr = array();
            $node->_[HDOM_INFO_END] = 0;
            $node->_[HDOM_INFO_TEXT] = substr($this->doc, $begin_tag_pos, $this->pos-
$begin_tag_pos-1);
            $this->pos -= 2;
            $this->char = (++$this->pos<$this->size) ? $this->doc[$this->pos] : null; // next
            $this->link_nodes($node, false);
            return true;
        }
        if ($name!=='/' && $name!=='') {
            $space[1] = $this->copy_skip($this->token_blank);
            $name = $this->restore_noise($name);
            if ($this->lowercase) $name = strtolower($name);
            if ($this->char==='=') {
                $this->char = (++$this->pos<$this->size) ? $this->doc[$this->pos] : null; // next
                $this->parse_attr($node, $name, $space);
            }
            else {
                //no value attr: nowrap, checked selected...
                $node->_[HDOM_INFO_QUOTE][] = HDOM_QUOTE_NO;
                $node->attr[$name] = true;
                if ($this->char!='>') $this->char = $this->doc[--$this->pos]; // prev
            }
            $node->_[HDOM_INFO_SPACE][] = $space;
            $space = array($this->copy_skip($this->token_blank), '', '');
```

```
            }
            else
                break;
        } while ($this->char!=='>' && $this->char!=='/');

        $this->link_nodes($node, true);
        $node->_[HDOM_INFO_ENDSPACE] = $space[0];

        // check self closing
        if ($this->copy_until_char_escape('>')==='/')
        {
            $node->_[HDOM_INFO_ENDSPACE] .= '/';
            $node->_[HDOM_INFO_END] = 0;
        }
        else
        {
            // reset parent
            if (!isset($this->self_closing_tags[strtolower($node->tag)])) $this->parent = $node;
        }
        $this->char = (++$this->pos<$this->size) ? $this->doc[$this->pos] : null; // next
        // If it's a BR tag, we need to set it's text to the default text.
        // This way when we see it in plaintext, we can generate formatting that the user wants.
        // since a br tag never has sub nodes, this works well.
        if ($node->tag == "br")
        {
            $node->_[HDOM_INFO_INNER] = $this->default_br_text;
        }

        return true;
    }
```

Appendix E: Fragment of Code for removing CSS, Comments, and Scripts.

```
function load($str, $lowercase=true, $stripRN=true, $defaultBRText=DEFAULT_BR_TEXT,
$defaultSpanText=DEFAULT_SPAN_TEXT)
   {
      global $debugObject;
      // prepare
      $this->prepare($str, $lowercase, $stripRN, $defaultBRText, $defaultSpanText);
      // strip out comments
      $this->remove_noise("'<!--(.*?)-->'is");
      // strip out cdata
      $this->remove_noise("'<!\[CDATA\[(.*?)\]\]>'is", true);
      // Per sourceforge
http://sourceforge.net/tracker/?func=detail&aid=2949097&group_id=218559&atid=1044037
      // Script tags removal now preceeds style tag removal.
      // strip out <script> tags
      $this->remove_noise("'<\s*script[^>]*[^/]>(.*?)<\s*/\s*script\s*>'is");
      $this->remove_noise("'<\s*script\s*>(.*?)<\s*/\s*script\s*>'is");
      // strip out <style> tags
      $this->remove_noise("'<\s*style[^>]*[^/]>(.*?)<\s*/\s*style\s*>'is");
      $this->remove_noise("'<\s*style\s*>(.*?)<\s*/\s*style\s*>'is");
      // strip out preformatted tags
      $this->remove_noise("'<\s*(?:code)[^>]*>(.*?)<\s*/\s*(?:code)\s*>'is");
      // strip out server side scripts
      $this->remove_noise("'(<\?)(.*?)(\?>)'s", true);
      // strip smarty scripts
      $this->remove_noise("'(\{\w)(.*?)(\})'s", true);
      // parsing
      while ($this->parse());
      // end
      $this->root->_[HDOM_INFO_END] = $this->cursor;
      $this->parse_charset();

      // make load function chainable
      return $this;

   }
```

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: _____

Signature: _____

Date: _____

Confirmed by advisor:

Name: _____

Signature: _____

Date: _____