Bridging the Web Accessibility Divide

A Dissertation Presented

by

Yevgen Borodin

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Stony Brook University

December 2009

Stony Brook University
The Graduate School

Yevgen Borodin

We, the dissertation committee for the above candidate for the
Doctor of Philosophy degree, hereby recommend
Acceptance of this dissertation.

Dr. I.V. Ramakrishnan – Dissertation Advisor
Computer Science Department

Dr. Steven Skiena – Chairperson of Defense
Computer Science Department

Dr. C.R. Ramakrishnan
Computer Science Department

Dr. Richard Ladner
Computer Science Department
University of Washington

Dr. Amanda Stent – Dissertation co-Advisor
AT&T Labs – Research

This dissertation is accepted by the Graduate School

Lawrence Martin

Dean of the Graduate School

Abstract of the Dissertation

Bridging the Web Accessibility Divide

by

Yevgen Borodin
Doctor of Philosophy

in

Computer Science
Stony Brook University
2009

The World Wide Web has come to take an important part in our lives as a medium of obtaining and exchanging information, performing financial transactions, getting education and employment, applying for social services, etc. However, in its evolution from text-based web pages to interactive graphical web applications, it left behind millions of people with impaired vision and blindness, for whom web access with screen-reading assistive technologies has become slow and inefficient. Modern screen readers perform almost no content analysis to facilitate access to relevant information; as a result, users are forced to process information sequentially through simplistic serial interfaces, which causes information overload. The growing popularity of Web 2.0 and dynamic content technologies is further widening the Web Accessibility divide between the ways blind and sighted people browse the Web.

In this dissertation, I present the HearSay3 web browser, which I envision as a next-generation assistive technology that will alleviate many Web Accessibility problems. HearSay3 is designed to help screen-reader users find relevant information in web pages more efficiently. The browser unifies access to information in web pages and provides a level of abstraction from the underlying Web technologies. Further in this dissertation, I explore several context-aware algorithms that empower HearSay3 to automatically analyze and identify relevant information in web pages. I report on several experiments to demonstrate that the proposed algorithms and interfaces can substantially improve the user experience and make non-visual web browsing more efficient and usable. I conclude this dissertation by outlining ample opportunities for applications of this work and further directions of research.

My specific contributions include: 1) a usable prototype of the HearSay3 aural web browser; 2) a layered non-visual interface for accessing various types of information; 3) a unifying approach to accessing changes in web pages with the Dynamo interface; 4) experiments evaluating the effectiveness of the Dynamo approach; 5) a novel context-directed browsing approach to finding the beginning of main content with CSurf; 6) experiments evaluating the effectiveness of the CSurf approach; and 7) an overview of strategies used by screen-reader users to overcome common Web Accessibility problems.

Dedication

…to my Dearest Parents, William Kinley, and my beloved Grandparents whose unwavering love and support have kept me reaching for the stars…

Table of Contents

List of Figures

List of Tables

Preface

Chapter 1 provides the motivation for this dissertation, overviews the contributions of the dissertation, and discusses its broader impact.

Chapter 2 gives a broad overview of related work on assistive technologies, accessibility guidelines, and web accessibility research. Topic-specific literature reviews are provided in the relevant chapters throughout this dissertation.

Chapter 3 introduces the HearSay project, presents the HearSay3 web browser, and describes its architecture and essential components, including the vxmlSurfer VoiceXML interpreter, automation of repetitive web browsing tasks with macros, and remote telephony access with TeleWeb, and others.

Chapter 4 describes general browsing strategies employed by screen-reader users, overviews the standard non-visual browsing interface, presents novel features of the HearSay browser interface, and reports on qualitative evaluation of HearSay.

Chapter 5 provides motivation for the Dynamo project, describes user strategies for coping with changes in web pages, presents the Dynamo approach with the underlying Dynamo-Diff page comparison algorithm, and presents the results of automated experiments with Dynamo-Diff. The chapter concludes with a description of two separate user studies conducted to evaluate the Dynamo approach.

Chapter 6 provides the motivation for the CSurf project, describes user strategies for finding main content in web pages, and overviews my prior and current work on finding relevant content in web pages. The chapter concludes with a report on automated evaluation and a description of human-subjects experiments with CSurf.

Chapter 7 concludes the dissertation by summarizing the contributions of this dissertation, suggesting possible further directions of research, and offering a broader message on web accessibility research.

Acknowledgements

I am very grateful to my undergraduate professors and advisors Mark Hardy and Carol Poucher whose support and encouragement led me to graduate school. I am infinitely grateful to I.V. Ramakrishnan, Amanda Stent, and Susan Brennan who made me fall in love with the exciting world of research. Dr. Ramakrishnan ignited my passion and excitement for research and invention; Dr. Stent taught me how to be meticulous and thorough in everything I do, and Dr. Brennan helped me realize the implications of my research to Psychology. I am very indebted to Chieko Asakawa, Terri Hedgpeth, and T.V. Raman – the amazing people who helped me realize that I am in the right field!

I would like to extend my gratitude to my collaborator Jeffrey Bigham who always challenged my ideas and, somewhere along the way, has become a good friend. I would like to thank my former colleagues and friends Jalal Mahmud, Yury Puzis, Faisal Ahmed, Asiful Islam, Glenn Dausch, as well as over a hundred Master's students for contributing to the HearSay project. My special thanks go to Nilesh Mahajan, Valentyn Melnyk, Elizabeth Cohen, and RaLynn McGuire for their help in running the automated and human-subjects experiments described in this dissertation. Also, I am thankful to the hundreds of researchers, professors, students, and friends who have influenced my work.

I would like to thank the T.J. Watson, Almaden, and Tokyo IBM Research Centers for giving me a great industrial research experience. Furthermore, I would like to express my appreciation to Hironobu Takagi, Chieko Asakawa, Shinya Kawanaka, Shimei Pan, Tessa Lau, Jeffrey Nichols, Allen Cypher, Roger Pollak, Michelle Zhou, from whom I learned a great deal while working at one of the greatest industrial research labs.

Finally, I would like to thank the National Science Foundation (Awards IIS-0534419, CNS-0751083, IIS-0808678) and Sigma Xi Grant-in-Aid of Research (Award G20071013028487872) for supporting the HearSay project.

Curriculum Vitae (2009)

Research Interests

Web accessibility, web content analysis, context in human-computer interactions, multimodal user interfaces, applications of machine learning, information retrieval, natural language processing

Education

2005-2009 *Ph.D. in Computer Science, Stony Brook University (SUNY), NY*

Advisors: Dr. I.V. Ramakrishnan (ALL Lab) and Dr. Amanda Stent (HCI/NLP Lab)
Projects (Java/JavaScript):
HearSay Context-Directed Non-Visual Web Browser
TeleWeb: A System for Web Browsing via the Phone
Voice Macros for Automating Repetitive Browsing Tasks
VxmlSurfer (Flexible VXML Interpreter)
CMo (Mobile Web Browser)

2003-2005 *M.S. in Computer Science (GPA 3.6), Stony Brook University (SUNY)*

Advisor: Dr. Annie Liu (Design and Analysis Research Lab)
M.S. Projects (Python):
Constraint-based pseudo-random relation generator
Web data-acquisition and validation tool based on RelaxNG XML schema
Web site generator based on XML data

2002-2003 *Alfred State College of Technology (SUNY), Alfred, NY*

Bachelor of Technology in Web Development (GPA **4.0**) with **Highest Honors**
Most requirements for BTech in Software Applications and Network Administration

2001-2002 *Alfred State College of Technology (SUNY), Alfred, NY*

A.S. in Computer Science (GPA **4.0**) with **Highest Honors**

2000-2001 *Shorter College, Rome, GA*

Business and Economics curriculum (GPA **4.0**)

1998-2000 *State Institute of Foreign Languages, Gorlovka, Ukraine*

Toward MA in Education/Linguistics: English, German, and Literature (GPA 3.5+)

Selected Employment

2006-2009 Research Foundation of SUNY at Stony Brook – Grad RA/Project Leader

- ✓ Lead a team of 10-15 M.S. students to develop the HearSay Non-Visual Web Browser
- ✓ Set development goals, assign and verify tasks, design and develop critical modules
- ✓ Represent the HearSay research group at conferences, demos, and presentations
- ✓ Establish and maintain industry connections, explore commercialization venues
- ✓ Conduct user studies and evaluations (certified for human subjects testing)
- ✓ Assist in procurement and allocation of funding, equipment purchases, lab management
- ✓ Write and review publications and grant proposals: contributed technical sections to the NSF grants recommended for funding: $1.5 mil research and $120 thou equipment grants
- ✓ 1st author on a patent application – US patent pending (61/153,058)

Summer 2008 IBM T.J. Watson and Tokyo Research Labs – Intern in Research

- ✓ Study user interactions with dynamic web content (Social Accessibility Project)
  Mentor: Hironobu Takagi; Managers: Chieko Asakawa and Roger A. Pollak
- ✓ Contributed significant code to the CoScripter project at IBM Almaden Research Lab
  Collaborators: Jeffrey Nichols, Tessa Lau
- ✓ 1st author on a patent application – Japan patent pending (2009-91450)
- ✓ 1st author on a book chapter about Social Accessibility

Summer 2007 IBM T.J. Watson Research Labs – Intern in Research

- ✓ Studied query recommendation strategies for information seeking systems

Mentor: Shimei Pan; Manager: Michelle Zhou.

Summer 2002 Olean General Hospital (OGH), Olean NY – Intern/Project Manager

- ✓ Led a team representing several departments to develop the OGH Web site
- ✓ Provided training, user-support, and documentation for Web site maintenance
- ✓ Built an Intranet work-order notification system (ASP)
- ✓ Participated in an OGH Information System conversion project

Teaching Experience

2005-2009 Stony Brook University (SUNY), NY – Guest-lecturer, Teaching Assistant

- ✓ Guest-lectured in undergraduate and graduate theory courses
- ✓ TAed undergrad Advance System Programming in Unix/C, CSE376, Dr. Erez Zadok
- ✓ TAed graduate Operating Systems, CSE506, Dr. Erez Zadok

2001-2003 Alfred State, Alfred, NY – Certified Tutor in Computer Science and Math

- ✓ Served as a supplemental instructor for Java OOP course (*Spring 2003*)
- ✓ Initiated and led group help-sessions for Information Systems students

Languages

- ✓ Fluent: English, Russian, Ukrainian
- ✓ Intermediate: German
- ✓ Beginning: French, Spanish

Selected Awards

- ✓ Honorable Mention, Student Competition at Microsoft Imagine Cup Accessibility Award'09
- ✓ SigmaXi Excellence in Research Award 2009, SigmaXi Scientific Research Society
- ✓ Honorable Mention for Increasing Accessibility, Student Application Contest, AVIOS'09
- ✓ Catacosinos Fellowship'08, outstanding C.S. contribution, (stipend) SBU, Stony Brook, NY
- ✓ Grant-in-Aid of Research 2007-08 ($1000), SigmaXi Scientific Research Society
- ✓ Excellence in Graduate Research Award'07, Computer Science, SBU, Stony Brook, NY
- ✓ 1st Place at International ACM Student Research Competition Grand Finals'07
- ✓ Travel Award 2007, SigmaXi Scientific Research Society
- ✓ Outstanding Teaching Assistant'06, Computer Science, SBU, Stony Brook, NY
- ✓ SUNY Chancellor's Award for Student Excellence'03, Albany, NY
- ✓ Alstom Power Inc. Air Preheater Scholarship Award'03, Alfred State, NY
- ✓ Paul B. Orvis Award for contribution to student life'03 at Alfred State, NY
- ✓ Scholarship Award for Outstanding Student Leadership'02 from Alfred State, NY
- ✓ Outstanding Senior Awards in CS'02 and Web Development'03 from Alfred State, NY
- ✓ Excellence in Education Full-Coverage Scholarship 2001-2003 from Alfred State, NY
- ✓ Dean's List at Shorter College, GA and Alfred State, NY (every semester 2000-2003)
- ✓ Finalist of FSA Undergraduate Exchange Program ACCTR/ACCELS (2000-2001)

Selected Affiliations

2008-Present	Student Member, the Institute of Electrical and Electronic Engineers (IEEE)
2006-Present	Student Member, Association for Computing Machinery (ACM)
2007-Present	Distinguished Member, SigmaXi International Scientific Research Society
2002-2003	Co-founder and Vice President, ACM chapter, Alfred State College, Alfred, NY
2001-2003	Treasurer, International Club; Member, Honors Society Alfred State College

Invited Talks

"Bridging the Web Accessibility Divide: A Vision for a Universally Accessible Web," Stony Brook University Provost's Graduate Student Lecture Series, 2009

"HearSay Vision for the Universally Accessible Web," Center of Disabilities Conference, California State University Northridge, 2009

"Bridging Web Accessibility Divide with the HearSay Browser," Computer Science Department, Arizona State University, 2008

"User Context and Dynamic Content in Non-Visual Web-Browsing," Computer Science Department, University of Washington, 2008

Patents

Japan patent pending (2009-91450), IBM Corporation (2009), "A Method And Computer Program For Improving Accessibility of Rich Internet Applications Using Collaborative Crawling."

U.S. patent pending (61/153,058), Y. Borodin**,** I. V. Ramakrishnan, "Remote Access to Personal Computer via the Phone." U.S. Provisional Application filed 2/17/09 by Stony Brook University (2009)

Reviewing Activities

2010

Program Committee Member:
12th ACM SIGACCESS Conferences on Computers and Accessibility (ASSETS'10)
7th International Cross Disciplinary Conference on Web Accessibility (W4A'10)

2009

Coauthor and member of the editorial board for a book titled:
"*End User Programming for the Web*"

Reviewer for the Journal of Transactions on Accessible Computing (TACCESS)

Program Committee Member:
28th ACM CHI Conference on Human Factors in Computing Systems (CHI'10)
11th ACM SIGACCESS Conferences on Computers and Accessibility (ASSETS'10)

2008

Member of the editorial board for a book titled:
"*Multimodality in Mobile Computing and Mobile Devices: Methods for Adaptable Usability*."

Program Committee Member:
27th ACM Conference on Human Factors in Computing Systems (CHI'09)
UI and Mobility Track, 18th Intl. World Wide Web Conference (WWW'09)
International Cross Disciplinary Conference on Web Accessibility (W4A'09)
10th ACM SIGACCESS Conferences on Computers and Accessibility (ASSETS'09)

2007

Program Committee Member:
9th ACM SIGACCESS Conferences on Computers and Accessibility (ASSETS'08)

Publications

M. A. Islam, F. Ahmed, Y. Borodin, J. Mahmud, Y. Borodin, I.V. Ramakrishnan, "Improving Accessibility of Transaction-centric Web Objects," To appear in *SDM 2010.*

Y. Borodin, S. Kawanaka, H. Takagi, M. Kobayashi, D. Sato, C. Asakawa, "Chapter 18: Social Accessibility: A Collaborative Approach for Improving Web Accessibility," *To appear in book: End-user Programming for the Web, Morgan Kaufmann, 2010*

J. Mahmud, Y. Borodin, I.V. Ramakrishnan, C.R. Ramakrishnan, "Automated Construction of Web Accessibility Models from Transaction Click-streams," *Intl. World Wide Web Conference 2009*

Y. Borodin, Glenn Dausch, I.V. Ramakrishnan, et al. "TeleWeb: Accessible Service for Web Browsing via Phone," Intl. Cross-Disciplinary Conf. on Web Accessibility, *W4A 2009* (**Judges' Award Winner**).

Y. Borodin, H. Takagi, S. Kawanaka, C. Asakawa, "Infrastructure and Methods for Improving Accessibility of Rich Internet Applications," *End-User Programming Workshop, CHI 2009*

I.V. Ramakrishnan, J. Mahmud, Y. Borodin, A. Islam, and F. Ahmed, "Bridging the Web Accessibility Divide," In *Electronic Notes in Theoretical Computer Science (Invited), 2008*

Y. Borodin, "Automation of Repetitive Web Browsing Tasks with Voice-Enabled Macros," *10th Intl. Conf. on Computers and Accessibility – ACM SIGACCESS ASSETS 2008.*

Y. Borodin, J. Bigham, R. Raman, and I.V. Ramakrishnan, "What's New? – Making Web Page Updates Accessible," *10th Intl. Conf. – ACM SIGACCESS ASSETS 2008* (**Best Student Paper**).

S. Kawanaka, Y. Borodin, J. Bigham, D. Lun, H. Takagi, and C. Asawaka, "Accessibility Commons: A Metadata Infrastructure for Web Accessibility," *10th Intl. Conf. on Computers and Accessibility – ACM SIGACCESS ASSETS 2008*.

Y. Borodin, J. Bigham, I.V. Ramakrishnan, and A. Stent, "Towards One World Web with HearSay3," Web Accessibility Challenge, *Intl. Cross-Disciplinary Conf. on Web Accessibility – W4A 2008*.

J. Mahmud, Y. Borodin and I.V. Ramakrishnan, "An Assistive Browser for Conducting Web Transaction," *International Conference on Intelligent User Interfaces – IUI 2008.*

H. Guo, J. Mahmud, Y. Borodin, A. Stent, and I.V. Ramakrishnan, "A General Approach for Partitioning Web Page Content Based on Geometric and Style Information," *ICDAR 2007.*

Y. Borodin, J. Mahmud, and I.V. Ramakrishnan, "Context Browsing with Mobiles **-** When Less is More," *Intl. Conf. on Mobile Systems, Applications and Services, ACM/USENIX MobiSys 2007.*

Y. Borodin, J. Mahmud, A. Ahmed, and I.V. Ramakrishnan, "WEBVAT: Web Page Analysis and Visualization Tool," *International Conference on Web Engineering – ICWE 2007.*

Y. Borodin, J. Mahmud, I.V. Ramakrishnan, A. Stent, et al., "The HearSay Non-Visual Web Browser," *Intl. Cross-Disciplinary Conf. on Web Accessibility – W4A 2007* (**Delegates' Award Winner**).

J. Mahmud, Y. Borodin, and I.V. Ramakrishnan, "CSurf: A Context-Driven Non-Visual Web-Browser," *International World Wide Web Conference – WWW 2007* (**1 of 3 Best Paper Nominees**).

J. Mahmud, Y. Borodin, D. Das, and I.V. Ramakrishnan, "Content Analysis Techniques to Ease Browsing with Handhelds," *MobEA 2007.*

J. Mahmud, Y. Borodin, D. Das, and I.V. Ramakrishnan, "Combating Information Overload in Non-Visual Web Access Using Context," *Intl. Conference on Intelligent User Interfaces, IUI 2007.*

J. Mahmud, Y. Borodin, D. Das, and I.V. Ramakrishnan, "Improving Non-Visual Web Access Using Context," *Intl. Conf. on Computers and Accessibility – 8th ACM SIGACCESS ASSETS 2006*.

Y. Borodin, "A Flexible VXML Interpreter for Non-Visual Web Access", *Intl. Conf. on Computers and Accessibility – 8th ACM SIGACCESS ASSETS 2006* (**2nd place ACM Student Research Comp.**)

1. Introduction

In this chapter, I provide the motivation for this work, present statistics on blindness, and describe some common problems in non-visual web browsing in Section 1.1, followed by a summary of my research contributions in Section 1.2.

1.1 Challenges in Non-Visual Web Browsing[1]

The World Wide Web has come to take an important part in our lives: it has become an indispensable source of information and means of communication. It is also used for conducting many types of transactions, including shopping online, paying bills, making travel plans, applying for college, employment, or social services, participating in civic activities, and many others. Three out of five adults go online regularly, and the majority of U.S. residents now use the Web to shop. In the US alone, 114.1 million adults searched for product information on the Web last year, and 98.9 million of this group went on to make purchases either online or offline (Shop.org 2004). K-12 schools, colleges, and universities increasingly use the Web for posting announcements, managing courses, *e.g.,* the BlackBoard Course Management System (Enagandula 2005), and even offering courses online. For people from all walks of life, as well as businesses, the Web has become the primary medium for accessing all sorts of digitized information and conducting e-commerce.

However, in its evolution from single-author text-based web pages to interactive web applications with user-generated content, the Web has grown to be less accessible to people with vision impairments and blindness, a population that is large and growing ever larger. The Web was designed by sighted people and for sighted people, so the primary mode of interaction over the Web is via graphical browsers designed for visual navigation (*e.g.,* Internet Explorer, Firefox, etc.). This critically hinders access to the Web for people with impaired vision or blindness.

With 23.6 million diabetics in the US alone, diabetes is one of the leading causes of blindness among adults ages 20 to 74 years; each year diabetic retinopathy causes 12,000 to 24,000 new cases of blindness (NDIC 2007). Unfortunately, vision impairments are not limited to diabetics alone. According to the 2002 report by the World Health Organization (WHO 2002), the leading causes of blindness worldwide include glaucoma (12.3%), age-related macular degeneration (AMD) (8.7%), corneal opacities (5.1%), diabetic retinopathy (4.8%), childhood blindness (3.9%), trachoma (3.6%), and onchocerciasis (0.8%). Just to give a sense of the size of the target population, there are over 20 million people with low vision and approximately 1.3 million people are

[1] Materials for this section have been adapted from a funded NSF proposal that I co-authored.

considered legally blind in the U.S. alone, as reported by the American Foundation for the Blind (AFB 2009). Worldwide, according to the World Health Organization (WHO 2003), there are more than 175 million people with visual impairments (40-45 million blind, 135 million low vision).

In 2003, the U.S. Bureau of the Census (http://www.census.gov) reported that, among people who are blind or have severe vision impairment, 63% of the adults (ages 25-60) in the labor force, and 40% of adults not in the labor force were using the Internet. Furthermore, 23% of the people with vision impairments over age 60 were also using the Internet, a percentage that will surely increase as baby boomers, who already rely on the Internet, continue to develop vision impairments associated with diabetes and aging. Forrester Research projects 70 million users will depend upon various assistive technologies by 2010 (Forrester 2004).

A number of software companies have already realized the need for assistive technologies to aid low-vision and blind people with computer use. For example, major operating systems (*e.g.,* Microsoft Windows, OsX, and Linux) provide a choice of accessibility options such as magnifying glass, text narration, etc. There is also a variety of other software tools used for screen-reading and web browsing. Well-known screen-readers include Freedom Scientific's (JAWS 2009), G.W. Micro's (Window-Eyes 2009), Dolphin's (SuperNova 2009), Apple's (VoiceOver 2009), and Sun's (Orca 2009). Also there are special-purpose applications that specifically target web browsing such as IBM's Home Page Reader (Asakawa 1998), aiBrowser (Miyashita 2007), WebAnywhere (Bigham 2008), and FireVox (Thiessen 2007). Screen-readers and non-visual web browsers typically read out the content of the screen ignoring the graphics and layout of web pages, while giving other audio or Braille feedback to help navigate Web pages. These tools enable blind and low vision people to interact with Web browsers and perform basic online activities.

Assistive technology software often relies on Web Content Accessibility Guidelines (WCAG 2009) that have to be followed by web designers. Specifically, the designers have to incorporate accessibility metadata into their websites, *e.g.*, provide alternative text for images (Section 2.2). Many businesses have already realized the strong business rationale for making their sites accessible; and as for web sites run by the government, they are required to be so by law. Section 508 of the Rehabilitation Act of 1973 was amended in 1998 to address Web accessibility. The scope of such laws keeps expanding.

In reality, however, many web designers do not follow the guidelines, in part because the content of today's sites is often generated by end users, who are posting information on content-sharing services such as forums, blogs, etc. in a volume, which can hardly be controlled by the site owners. Consequently, the accessibility metadata incorporated into websites is often inadequate in both quality and quantity. Site owners are not able to give higher priority to their websites' accessibility compared to their business and technology needs. As a result, interactivity and richness of content and its visual appeal remain the primary focus of most web developers. In real life, however, screen-reader users face numerous accessibility problems from having to listen to irrelevant content to not being able to complete their tasks at all for the reason that they cannot access information, fill forms, follow links, etc. Having opened web pages with image-links without alternative text, Flash, Java Applets, and JavaScript widgets, screen-reader users frequently find themselves stuck. To date, a number of techniques have been developed to overcome these accessibility problems, *e.g.*, (Bigham 2006; Miyashita 2007; Chen 2008).

Nevertheless, a large gap remains between the ways sighted and blind users browse the Web due to the differences in their perception and the mode of interaction (Bigham 2007). Sighted users can promptly process Web content by visually segmenting any web page into sections, classifying the whole page and its parts, finding patterns, filtering out irrelevant information, and quickly identifying possible steps and actions they can take on any web page. On the other hand, users with visual impairments are forced to process information sequentially, as the assistive software reads content in the order it is presented in web pages. While screen readers provide shortcuts for skipping through text or reading it more rapidly, screen-reader users still have to listen to much irrelevant content before reaching the content of interest since screen-readers provide almost no content analysis to facilitate access to relevant information. When a screen-reader user visits a web page for the first time, s/he cannot easily tell, without listening to all of it, how much information it contains. Navigating back and forth among pages, screen-reader users often have to listen to redundant information, design strategies to find relevant content, or remember the page structure to make web browsing more efficient. All these problems make non-visual web browsing *time-consuming* and *difficult*, which may result in *information overload*. There exists, therefore, a clear need for a new assistive technology which can enable screen-reader users to navigate web content more efficiently.

1.2 Contribution of the Dissertation

Having established the need for better assistive technology, I present several techniques that employ user context in order to find relevant information in web pages and, in this way, help screen-reader users browse the web more efficiently. I overview relevant Web Accessibility research and the current state of assistive technology tools and then present the architecture, algorithms, and non-visual interfaces for bridging the Web Accessibility Divide. The main contributions of this dissertation include: 1) a usable prototype of the HearSay3 aural web browser; 2) an innovative non-visual interface for accessing various types of information; 3) a unifying approach to accessing changes in web pages with Dynamo; 4) experiments evaluating the effectiveness of the Dynamo approach; 5) a novel context-directed browsing approach to finding the beginning of main content with CSurf; and 6) experiments evaluating the effectiveness of context-directed browsing with CSurf.

To push forward the state of the art in non-visual web browsing, I led the development of a usable prototype of the HearSay3 browser. The HearSay aural web browser presented in Chapter 3 helped us better understand the difficulties of implementing assistive technologies and the accessibility problems commonly encountered by screen-reader users. The HearSay3 prototype, currently in alpha testing, is now being used to conduct user studies and verify various approaches to improving web accessibility, including those presented in this dissertation. In the course of its development under my supervision, HearSay has won a number of prestigious accessibility awards and was used to train a generation (~100) of young computer scientists. Over the past two years, various features of the HearSay browser have been demonstrated to over 100 blind users, 30 of whom also took part in the HearSay user studies.

In Chapter 4, I present various improvements for the non-visual web browsing interface including the TeleWeb, Macro, and Social Accessibility. I describe a different way of seeing web content as a combination of layers of certain element types. A layered approach provides a unifying interface for accessing various types of information and allows users to combine layers to create custom views on information. A layered interface complements and extends the common screen-reader list-view interface for accessing links, headings, etc. The layered interface allows users to focus on the content of their choosing and to browse their personalized views with the usual navigation keys. The interface also makes it easy to switch between custom views and a full page-view allowing users to explore the rest of the web page content. Finally, the concept of layering web content provides an easy and intuitive interface to the Dynamo approach.

I present a unified view of changing web content, as well as an interface agnostic of the type of technology used to update the content. The approach implemented in the Dynamo module covers both dynamic and static content changes, thus creating a layer of abstraction for users. I describe the Dynamo module and its custom DOM-Diff algorithm that enables the Dynamo interface. The DOM-Diff algorithm is used to analyze dynamic and static changes in web pages and filter out unchanged content. I hypothesize that, while browsing, users are often interested in the changes that occur in web pages. The

Dynamo module embodying the approach automatically detects changes in web pages and allows users to review the changes with the layered interface. Similarly, Dynamo assists users with staying focused on their task by repositioning the HearSay cursor, in case of page updates (Chapter 5).

To evaluate the Dynamo approach and the underlying algorithms, we conducted both automated and human-subjects experiments. The automated experiments confirmed high accuracy of the Dynamo-Diff algorithm demonstrating that it can correctly identify differences between web pages. Two sets of human-subjects experiments, with eight screen-reader users in each, confirmed the usability and effectiveness of the Dynamo approach. Both human-subjects experiments involved performing tasks with a baseline non-visual browser reproducing the behavior of a typical screen-reader and the advanced system implementing the Dynamo approach. The first experiment studied subjects' ability to handle a simple dynamic update, overcome form-filling errors, and recover from page refreshes. The second experiment studied how efficiently subjects could find information in template-based websites. Both human-subjects experiments demonstrated a significant improvement in the subjects' ability to find relevant information and cope with common accessibility problems (Chapter 5).

I present CSurf – a novel context-directed browsing approach to finding the beginning of main content in web pages as users are navigating from page to page. The CSurf-New algorithm leverages the context of a followed link from the source webpage to match and find the beginning of main content on the target page. The main contribution of the CSurf approach is that it helps find the starting position of main content from which the page should be read to the screen-reader users (Chapter 6).

Both automated and human-subjects experiments were conducted to evaluate context-directed browsing with CSurf. The automated experiments confirmed high accuracy of my CSurf-New algorithm and provided justification for its design. A user study with 16 human-subjects was conducted to evaluate the CSurf approach. The study included variations of two browsing tasks which were performed on a baseline non-visual web browser reproducing the behavior of a typical screen-reader and a system that implemented the CSurf approach. The first series of tasks evaluated users' ability to find the beginning of main content in web pages with varying degrees of accessibility. The second series of tasks measured subjects' ability to retain information when their reading was interrupted by irrelevant information. The experiment demonstrated a significant improvement in user browsing efficiency (Chapter 6).

Finally, throughout this dissertation I provide detailed overviews of existing web accessibility problems and describe coping strategies employed by screen-reader users to overcome these problems.

2. Related Work

In this section I give a broad overview of related work on Web Accessibility. Topic-specific literature reviews are provided in the relevant chapters of this dissertation.

2.1 Accessibility Technologies

Desktop screen readers. Although major operating systems (*e.g.*, Microsoft Windows, OsX, and Linux) provide a choice of accessibility options such as the magnifying glass, text narration, etc., the majority of screen-reader users do not use the built-in accessibility features and prefer to use specialized screen-readers[1] (JAWS 2009; NVDA 2009; SuperNova 2009; Window-Eyes 2009) and magnification software (MAGic 2009; ZoomText 2009), which have limited users mostly to the Windows platform. The recent improvement of Apple's built-in screen reader, however, (VoiceOver 2009) has boosted the number of Mac users (WebAIM 2009). The HearSay browser does not yet implement magnification, but it has been modeled on popular screen readers. Because HearSay does not aim to replace screen readers, but rather complement them in web browsing tasks, HearSay implements similar shortcuts and functionalities as those implemented by screen readers in order to reduce the learning curve for novice users. The purpose of this dissertation, however, is to explore a number of advanced functionalities that are not available in screen readers.

Remote and mobile access. Audio-based applications for remote access are also popular among screen-reader users (SaToGo 2009). Plain remote desktop tools (PCNow 2009) can also be used by blind users if the remote computer they connect to has a screen reader. WebAnywhere (Bigham 2008) makes any web browser accessible as soon as the WebAnywhere website is loaded in the browser. WebAnywhere uses a transcoding proxy to enable web accessibility from any computer terminal with restricted access, as long as it has a web browser. As we have shown in (Borodin 2007a), HearSay algorithms can also be used on a transcoding proxy, in this case, to improve visual browsing for mobiles. With the appearance of smart-phones, blind users are now beginning to use mobile screen readers (MobileSpeak 2009; VoiceOver 2009) that in some cases involve touch and gesture interfaces.

Telephony access. Voice driven phone-based services targeted at the general population are a new emerging technology that can also help blind people access information. The 1-800-2ChaCha (ChaCha 2009) service enables users to search the web over the phone. Google has recently released its Voice Search (GOOG-411 2009) service, which is essentially a voice interface to Google maps. IBM is developing the Telecom Web (Kumar 2007), which is envisioned as a network of interconnected Voice

[1] A comprehensive listing is available at http://en.wikipedia.org/wiki/Comparison_of_screen_readers

Sites hosting voice driven applications created by users themselves. Tellme Networks, Inc. (TellMe 2009) provides online voice services such as directory assistance and search via mobile phones. There are also telephony services for accessing email via phone (email2phone 2009; JConnect 2009), Wikipedia (Sherwani 2007), and even the general Web (NetEcho 2009). Since HearSay has not been optimized to run on mobile platforms, more development work is required to make it run on mobile devices. However, HearSay-TeleWeb (Borodin 2009) has enabled remote-mobile telephony access to the desktop HearSay browser via any telephone (Sections 3.7 and 4.5).

Non-visual web browsers. Voice-browsers' category is perhaps the best fit for the HearSay project. While enabling computer interfaces to the fullest is a challenging engineering effort, a number of projects have taken an easier approach and targeted only web browsing functionalities. Voice browsers are often implemented by augmenting existing web browsers (Hanson 2008). One of the oldest representatives of voice browsers is IBM's Home Page Reader (Asakawa 1998). The latest IBM's prototype aiBrowser (Miyashita 2007) implemented as a plug-in for the Microsoft Internet Explorer has mostly been targeting multimedia web content. The support of multimedia content has not yet been implemented in the HearSay browser, but it is an important future direction of work. FireVox (Thiessen 2007) supported by Google is a Firefox browser extension enabling the basic browsing functionalities, as well as the support of ARIA on Windows, OsX, and Linux platforms. HearSay uses Firefox and Internet Explorer extension to integrate with the browser, but the HearSay's analytics pipeline is outside of the browser extension (Section 3.3). This design allows HearSay to connect to multiple browsers and makes the HearSay code faster and easier to debug since it is written in Java, rather than JavaScript. Although HearSay does not yet provide support for ARIA specifications, it supports dynamic content in a more general way (Chapter 5). WebAnywhere (Bigham 2008) is a server-side solution for remote web browsing from any computer terminal. We are exploring the possibility of developing HearSay-Anywhere solution, which is akin to WebAnywhere, but which can provide remote access to the HearSay browser via a web terminal.

The state of assistive technologies. This section gives a representative overview of the modern-day assistive technologies that enables non-visual computer access and web access, in particular. Although these technologies empower blind users to browse the Web, most of the assistive tools available today perform minimal content analysis and limit their functionalities to navigation. Modern assistive technologies use at best simple heuristics such as attaching leftmost labels to form elements. At present, the latest version of Apple's VoiceOver does page-segmentation making it easier to navigate between segments. Overall, however, the majority of assistive technology developers have been trying to make the Web accessible", rather than trying to make non-visual web browsing *more* efficient. In contrast, the goal of this dissertation has been to narrow the gap between the ways in which blind and sighted users browse the Web by making browsing faster through automation (Section 4.4), finding updated content (Chapter 5), and finding the beginning of main content (Chapter 6).

2.2 Accessibility Guidelines and Compliance

The importance of promoting Web accessibility for individuals who are blind or visually impaired has prompted the World Wide Web Consortium (W3C 2009) to launch the Web Accessibility Initiative (WAI 1997) resulting in a number of web accessibility guidelines, *e.g.*, Web Content Accessibility Guidelines (WCAG 2009) addressing information in a web site, Accessible Rich Internet Applications (WAI-ARIA 2009) addressing dynamic web content and web applications, Authoring Tool Accessibility Guidelines (ATAG 2009) addressing web development tools, User Agent Accessibility Guidelines (UAAG 2009) addressing web browsers and media players, etc.

Although these are just guidelines, governments around the world are increasing their accessibility regulations. In the United States, the rights of people with disabilities are protected by Section 508 of the Rehabilitation Act of 1973, further amended in 1998 to address Web accessibility and mandate that "electronic and information technology developed, procured, used, or maintained by all agencies and departments of the Federal Government be accessible both to Federal employees with disabilities and to members of the public with disabilities, and that these two groups have equal use of such technologies as federal employees and members of the public that do not have disabilities." Similar laws now exist in a number of countries (Paciello 2000).

A 2006 case of the *National Federation of the Blind (NFB 2009) vs. Target Corporation* has shown that laws go beyond paper. Based on the NFB's allegations that the inaccessibility of the *target.com* website violated the Americans With Disabilities Act, the California Unruh Civil Rights Act, and the California Disabled Persons Act, the U.S. Federal District Court for the Northern District of California certified the case as a class action lawsuit, which was later settled by Target.

Major software companies, in an attempt to comply with these laws, have formed divisions that regulate internal compliance with industry standards and accessibility guidelines, *e.g.,* IBM's Human Ability and Accessibility Center (HAAC 2009), Sun Microsystems' (Sun 2009), Microsoft Corporation's (Microsoft 2009), etc.

Nevertheless, many web sites remain inaccessible because web developers focus on visual attractiveness of their web sites instead of making them accessible. As a result, many screen-reader users are left behind as they cannot or will not visit the web sites plagued by numerous accessibility problems. The goal of the HearSay project has been the improvement of the user experience in browsing any website. Like typical screen readers, the HearSay browser makes use of the accessibility metadata incorporated into web page, *e.g.*, reads alternative text for images. However, HearSay does more than screen readers, as it uses webpage analysis algorithms to create missing metadata.

2.3 Web Accessibility Research

Automated accessibility verification. Because understanding accessibility guidelines and verifying compliance of web pages are not always straightforward (Kelly 2005), a number of automated accessibility checkers were created to help web developers verify the accessibility of their websites, *e.g.*, (Bobby 1995; aDesigner 2004; ACChecker 2009; MVS 2009). While automated verification can check websites' adherence to simple rules, it cannot check the semantics of accessibility metadata, *e.g.*, it cannot check whether the alternative text for images is meaningful, or whether an image used for design purposes should or should not have alternative text. The goal of the HearSay browser is to make websites accessible automatically, whenever possible, regardless of whether they comply with web accessibility guidelines or not. However, not all problems such as those relating to semantics can be resolved automatically.

Manual accessibility verification. Facing multiple accessibility problems, screen-reader users often report to website administrators. Regrettably, it may take months before any changes are made, if at all. In an attempt to overcome this problem, the Social Accessibility Project (Takagi 2008) has created a social network that joins end-users and supporters, who collaboratively create external accessibility metadata and, in this way, improve the accessibility of web sites. Screen-reader users submit requests describing accessibility problems, and the supporters create accessibility metadata to resolve the problems. Unfortunately, this approach may be too time-consuming. HearSay has implemented some initial support of the Social Accessibility network and will use manually created metadata. Screen-reader users, however, are not always able to identify accessibility problems unless these problems prevent the users from completing their current task. A comparative study of various accessibility verification methods (Mankoff 2005) has shown that web developers can best perform validation when forced to do it with a screen reader. HearSay may be a cheaper solution for manual accessibility verification compared to expensive commercial screen readers.

Automatic transcoding approaches. The continuing problems with web accessibility have led to numerous attempts to automatically improve accessibility of web sites. One popular approach is transcoding – automatic modification of the original content before it reaches end-users. Transcoding of web sites was originally developed to adapt them for mobile devices (Bickmore 1997) and to personalize pages (Maglio 2000). By now, this technique has been extensively explored for web accessibility (Asakawa 2008). Many transcoding approaches are based on a proxy server performing the transcoding. Some representative examples include WebInSight (Bigham 2006) automatically adding alternative text to images, SADIe (Harper 2007) using an ontology and CSS to make pages more accessible, AxsJAX (Chen 2008) transcoding web pages by injecting ARIA metadata, etc. HearSay does not transcode web pages for consumption by other assistive technologies, but rather uses its content analysis algorithms to construct its own dialogs.

3. HearSay3 Aural Web Browser

In this Chapter, I introduce the HearSay project. I describe its origins in Section 3.1, present its achievements in Section 3.2, and overview my design of the HearSay3 architecture in Section 3.3. In the remainder of the chapter, I highlight some of the HearSay modules; specifically, I provide more details on my design of the vxmlSurfer VoiceXML interpreter in Section 3.4, support of Social Accessibility in Section 3.5, automation of repetitive web browsing tasks with macros in Section 3.6, and TeleWeb remote telephony browsing in Section 3.7. Finally, I conclude this chapter by describing some further directions of HearSay development in Section 3.8.

The primary contributions of this chapter include: 1) a flexible architecture for the HearSay aural web browser, 2) a vxmlSurfer VoiceXML interpreter, and 3) a TeleWeb plug-in for remote telephony access to the HearSay browser.

3.1 Origins of HearSay

The HearSay non-visual Web browser[1] is the outcome of a five-year long research and development. The HearSay project started in 2004 with the goal of developing a state-of-the-art aural web browser for users with vision impairments. In 2005, the project received 2-year funding (Award IIS-0534419, 2005-2007, extended to 2009) from the National Science Foundation (NSF). I joined the HearSay project in the fall of 2005 and took the lead role in the project in the spring of 2006, co-advised by Dr. I.V. Ramakrishnan and Dr. Amanda Stent. Since then, the project received additional NSF funding to create an Accessibility Laboratory (Award CNS-0751083, 2008-2011) and to study cognitive aspects of web browsing among blind people (Award IIS-0808678, 2008-2012).

Since its inception, the HearSay browser has had three reincarnations. My advisor, Dr. I.V. Ramakrishnan, led the research and development of HearSay with his Ph.D. students Saikat Mukherjee (HearSay1) and Zan Sun (HearSay2). Both HearSay1 and HearSay2 (Sun 2006a) remained research prototypes and have never been released to the public. To make the browser more scalable and meet new research needs, in the spring of 2006, I was entrusted with leading a complete rebuild of the HearSay3 browser, whose current architecture is presented in Section 3.3.

[1] HearSay videos can be viewed at: http://www.sbhearsay.net

3.2 Related Work on HearSay

3.2.1 HearSay Publications

To date, HearSay research has resulted in a number of publications in selective conferences and journals. The HearSay research team developed several algorithms for segmenting web content (Mukherjee 2003; Mukherjee 2004; Mukherjee 2005a; Mukherjee 2005b; Guo 2007; Mahmud 2007b), enabled access to Blackboard (Enagandula 2005), built a flexible VoiceXML interpreter (Borodin 2006), implemented approaches to automatic generation of VoiceXML dialogs from segmented web pages (Sun 2006c), invented context-directed browsing (Mahmud 2006a; Mahmud 2007a; Mahmud 2007b) and adapted it for handhelds (Borodin 2007a), implemented general methods for constructing (Mahmud 2006b; Sun 2007) and automatically learning process models for conducting online transactions (Mahmud 2009), developed an approach to automating repetitive tasks in non-visual web browsing (Borodin 2008a), and introduced a new paradigm for unified handling of both static and dynamic changes in web pages (Borodin 2008b).

3.2.2 HearSay Awards

HearSay-related research and prototypes have won numerous awards, including **Best Paper nominee awards** at the 15th (Sun 2006b) and 16th (Mahmud 2007b) International World Wide Web Conferences for construction of process models and the context-directed browsing approach, respectively; the **second place** in the graduate category of the Student Research Competition at ASSETS'06 for the vxmlSurfer VoiceXML interpreter (Borodin 2006) and the **first place** in the ACM's 2007 **Grand Final Student Research Competition** for the context-directed browsing approach; the **Delegates Award** for the HearSay prototype (Borodin 2007b) and **the Judges Award** for the TeleWeb prototype (Borodin 2009) at the Web Accessibility Challenge Competition; the **Best Paper Award** at the 10th ACM Conference on Computers and Accessibility for the Dynamo approach (Borodin 2008b). The HearSay prototype has also received **honorable mentions** at the Microsoft Imagine Cup Accessibility Award Competition (2009) and AVIOS Student Application Contest (2009).

3.3 HearSay Architecture

The HearSay Architecture consists of the HearSay Application and Browser Plug-ins (Figure 1). The Browser Plug-ins enhance regular web browsers and enable communication between the HearSay Application and the browsers. The HearSay Application acts as a server, allowing multiple Browser Plug-ins to connect to and exchange information with HearSay. The "client-server" architecture permits users to run the HearSay Application and the browsers on the same machine or across the network. In the former case, HearSay can be used just like a standalone screen reader; in the latter case, HearSay can be run as a service. The HearSay Application executes a pipeline of algorithms for analyzing web content and managing the dialog interface. In this section, I will give a brief overview of each of the HearSay modules.

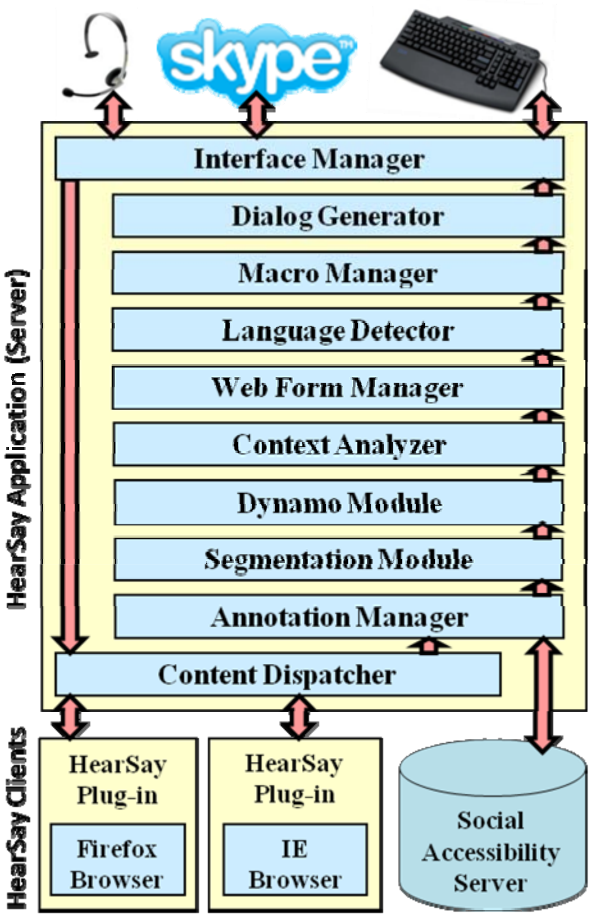


Figure 1 – Architecture of HearSay

3.3.1 Browser Plug-in

The Browser Plug-in is a web browser extension that provides seamless integration of HearSay features with the browser. The plug-in listens on browser and user events (*e.g.*, key presses, tab switches, focus changes, DOM mutation events, page loads, etc.) and sends them all to the HearSay Application. When the plug-in detects a page load event, it extracts the DOM tree corresponding to the newly loaded HTML page, and enriches the tree with geometric and style information inferred by the web browser's HTML rendering algorithm. The rest of the page processing logic is offloaded to the HearSay Application; this design minimizes the amount of code that has to be written for the browser, which is especially important because different plug-ins have to be written for different browsers.

HearSay currently supports the latest versions of Firefox and Internet Explorer (IE) browsers, with only some parts of the browser interface still remaining inaccessible to HearSay. These two browsers have been chosen for having the largest customer bases – 46.6% and 39.6% of the market, respectively[1]. Depending on the browser implementation, a separate instance of the plug-in is launched for every browser window or even tab. For example, Firefox launches a separate instance of the HearSay Plug-in for every window, while IE runs the HearSay Plug-in for every tab. Once launched, every

[1] September 2009 statistics: http://www.w3schools.com/browsers/browsers_stats.asp

plug-in instance seeks to establish a TCP[1] communication with the HearSay Application. Such a design, where the Browser Plug-in is a client and the HearSay Application is a server, opens possibilities for running HearSay and Browsers on different computers, which may be important in remote access applications and handhelds.

3.3.2 Content Dispatcher

As soon as the HearSay Application is started, the Content Dispatcher module opens a TCP socket where it listens on incoming connections. For every new connection, it spawns a new thread, which is maintained for the duration of the session. The session ID number is used as the identifier for the browser window or tab connected to HearSay. The Content Dispatcher module enables bidirectional communication between Browser Plug-ins and HearSay. The Browser Plug-ins send HearSay the events captured in the browser, while HearSay sends the plug-ins commands that control browser behavior.

Although a simple browsing functionality could also be implemented in the HearSay Application by fetching HTML pages, the separation of the main HearSay codebase from the browser implementation carries a number of benefits. As web browsers have grown to be complex applications (running JavaScript and Flash, handling cookies and secure connection, maintaining history and bookmarks, preventing phishing and blocking pop-ups), outsourcing browser functionality to established browsers helps the HearSay Application stay focused on making web content accessible. The separation also results in higher flexibility and independence of the modules, *e.g.*, the browsers and the HearSay Application can be running on different machines. Finally, web browsers provide a graphical interface, which can be useful for people with low vision. In addition, HearSay may eventually reach the stage when sighted people will be able to use HearSay's multimodal interface to browse the web both visually and via voice.

3.3.3 Annotation Manager

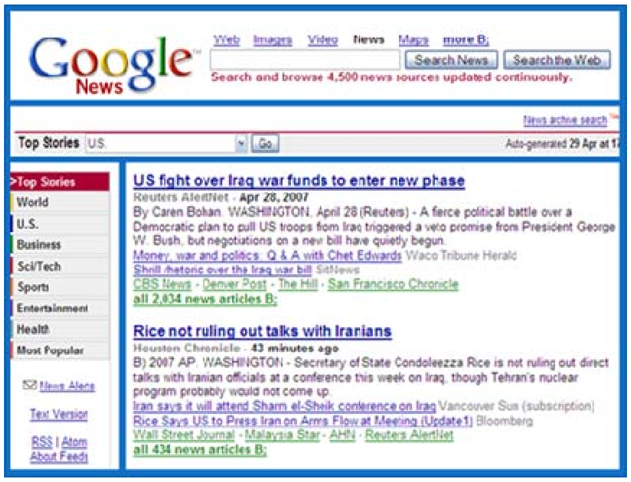
HearSay users can annotate web content, *e.g.*, mark up headings, provide alternative text for images, etc. The Annotation Manager keeps track of user annotations (accessibility metadata), stores them at the Social Accessibility server in the form of stand-off (external) annotations. The Annotation Manager establishes a connection with the Social Accessibility server, commits new annotations to the server, queries the server database to retrieve metadata for every loaded page, and applies the retrieved metadata to web pages so that HearSay users can enjoy previously created annotations and browse fully accessible web pages.

The HearSay project has initially maintained its own accessibility database; however, it has recently migrated to the IBM's Social Accessibility Server (Takagi 2008; Borodin 2010). Social Accessibility (SA) is a collaborative framework that uses the power of the crowd-sourcing to improve the accessibility of existing web content. SA brings together end-users and supporters who can collaboratively create accessibility metadata. By using the SA Network, HearSay can now benefit from accessibility metadata created by users of other assistive technology tools. Additionally, HearSay can contribute to the SA network the metadata produced by other modules described later in this section. More on the Social Accessibility project can be found in Section 3.5.

[1] Transmission Control Protocol (TCP) was used because it has reliable ordered information delivery

3.3.4 Segmentation Module

The Segmentation Module analyzes the structure and geometric layout of web pages and tries to segment them in a way that will enable screen-reader users to navigate between meaningful pieces of information (Figure 2). Work is underway to enhance the module so that it can find structural and visual patterns in web pages that will provide a higher granularity of navigation for HearSay users and more information about the page structure to other HearSay modules.



US fight over Iraq war funds to enter new phase
Reuters AlertNet - Apr 28, 2007
By Caren Bohan WASHINGTON, April 28 (Reuters) - A fierce political battle over a Democratic plan to pull US troops from Iraq triggered a veto promise from President George W. Bush, but negotiations on a new bill have quietly begun.
Money, war and politics: Q & A with Chet Edwards Waco Tribune Herald
Shrill rhetoric over the Iraq war bill SitNews
CBS News - Denver Post - The Hill - San Francisco Chronicle
all 2,034 news articles B;

Rice not ruling out talks with Iranians
Houston Chronicle - 43 minutes ago
B) 2007 AP. WASHINGTON - Secretary of State Condoleezza Rice is not ruling out direct talks with Iranian officials at a conference this week on Iraq, though Tehran's nuclear program probably would not come up.
Iran says it will attend Sharm el-Sheik conference on Iraq Vancouver Sun (subscription)
Rice Says US to Press Iran on Arms Flow at Meeting (Update1) Bloomberg
Wall Street Journal - Malaysia Star - AHN - Reuters AlertNet
all 434 news articles B;

Figure 2 – Web Page Segmentation of Google News

HearSay currently implements a variation of the Microsoft's Vision Based Page Segmentation Algorithm (VIPS) (Cai 2004). VIPS is a top-down algorithm that analyzes the page layout and identifies blocks of information. VIPS then tries to find horizontal and vertical separators that are used to reconstruct the semantic structure of the web page.

3.3.5 Dynamo Module

The main function of the Dynamo Module is to handle dynamic and static changes in web pages and provide an accessible user interface for reviewing the changes. The Dynamo module uses a custom Diff algorithm to compare the DOM trees of web pages visited by the user and mark up all information that changed between the web pages. This functionality allows users to focus on the changes and easily skip repetitive information, which often includes the links and menus that are present on every page of a web site. The Dynamo interface also helps review differences in the web pages that periodically refresh to update their content.

The Dynamo interface allows users to navigate between groups of changes in a web page or review a layer of changes as described in Section 4.3. In case of page refreshes, Dynamo tries to reposition the HearSay cursor. The module also notifies users of incoming dynamic updates by playing an earcon (Brewster 1993), a short audio clip associated with some event. In handling dynamic content, Dynamo will complement ARIA (WAI-ARIA 2009) mark-up that may or may not be used by web designers. More details on the Dynamo algorithms, interface, and user studies are presented in Chapter 5.

3.3.6 Context Analyzer

The Context Analyzer module plays the central role in identifying relevant information in web pages. The underlying CSurf-New algorithm analyzes the context of user actions to infer the beginning of main content in web pages.

The module collects the context of the link that was clicked by the user on the source web page, and then uses it to match the context against the target web page. The algorithm uses a combination of the cosine similarity metric and the font size to find the starting position for the main content of the page. In the future, the context analyzer will use more contextual clues to identify what is relevant for the user. More details on the CSurf algorithm, interface, and user studies are presented in Chapter 6.

3.3.7 Web Form Manager

The Web Form Manager helps associate labels with the corresponding form elements. Whenever available, the module uses the labels provided by web designers (*e.g.*, "*<label for='formId'>search</label>*"), or else it tries to make its best guess. The Web Form Manager currently uses simple heuristics, *e.g.*, picking the closest text to the left of the form element. Correct label association is very important for screen-reader users since they cannot fill out unlabeled web forms without the assistance of someone who can see the labels. Getting the labels right would mean a substantial increase in the productivity and independence of screen-reader users.

Machine learning techniques have been previously used to extract form element labels to facilitate information extraction (Nguyen 2008). Work is underway (the effort led by a Ph.D. student Asiful Islam) to design a new probabilistic method for making accurate associations between form elements and their labels with the help of Finite Mixture Models (McLachlan 2001). In the future, we envision that the Web Form Manager will provide assistance to users with filling out forms and performing other form-related actions.

3.3.8 Language Detector

The Language Detector module analyzes web pages to determine the language of the entire page, as well as that of page sections. In case the language is not specified with the *lang* attribute, the module uses statistical analysis to determine the most probable language for the passage. Subsequently, if the corresponding setting is turned on, HearSay dynamically switches between the available synthetic voices to read the content in the original language. In pages that mix multiple languages such as wikipedia.org, the system seamlessly switches between languages, as it reads the content.

The Language Detector module currently implements a popular *n-gram* based classification algorithm (Cavnar 1994) that recognizes languages by the frequency of use of sequences of *n* letters. The language classifier is first trained on foreign language corpora and is subsequently used to identify the most probable language. While the classifier works reliably for whole-page language detection, smaller text segments often do not provide sufficient n-gram samples for consistent classification. We are currently working on improving the algorithm to take into account page specific features and try to identify the language of smaller web page elements in the context of their neighbors.

3.3.9 Macro Manager

The Macro Manager allows screen-reader users to automate repetitive browsing tasks that are performed on a regular basis. For example, checking email, weather or news, paying bills, and shopping – all this could all be accomplished more efficiently and faster if users had a way to record and replay their transactions in a flexible way. To this end, in (Borodin 2008a), I introduced an approach for recording and replaying Macros, both visually and non-visually, by tracking keyboard shortcuts, mouse-clicks, and content changes in form elements and logging them in XML format. A macro recording is saved as a sequence of actions associated with the corresponding element addresses.

HearSay users can initiate a macro recording by pressing a shortcut or saying "Record", at which point, the browser starts tracking user actions. If the user starts by loading a new page, the macro recording will be page-specific. Conversely, starting from an already opened web page will make the recording page-independent. Once the recording is saved, it can be replayed with the macro player by either entering or speaking the name of the recording. The player then interprets the recorded log and simulates the corresponding actions using the standard HearSay interface. Although a lot of research has been done on automating web browsing tasks with macros, *e.g.,* (Anupam 2000; Bolin 2005; Leshed 2008; Bigham 2009b), HearSay is the first non-visual browser to use macro automation for Web Accessibility.

3.3.10 Dialog Generator

The Dialog Generator module uses the results of all content analysis algorithms described in this Dissertation to convert web page content into interactive dialogs that will be played to the user. The module linearizes the content of the web page into a sequence of "spoken elements." The separation from the underlying DOM tree data-structure creates a level of abstraction that allows dialog designers to implement simpler logic for content interpretation and delivery to the end-users.

The Dialog Generator module uses a collection of VXML[1] (VoiceXML 2009) dialog templates that are designed to provide standard screen-reading functionalities, as well as the advanced interface features described in this dissertation. The module converts any HTML DOM tree into an array of spoken elements that contains attributes for every element. This dynamic approach allows HearSay to unify the treatment of page-loads and dynamic updates by injecting the new content in the executing dialogs at run-time, similar to the way AJAX updates web pages.

3.3.11 Interface Manager

Users interact with HearSay through the Interface Manager, which is a flexible VoiceXML interpreter vxmlSurfer (Borodin 2006) that I have initially developed and subsequently led a team of M.S. students to improve it further. The module interprets the VoiceXML dialogs that implement the logic for voicing web content. The Interface Manager supports a variety of inputs, including keyboard shortcuts, speech commands, and DTMF[2] signals.

For every window or tab opened in the target web browser, HearSay instantiates a separate dialog thread in vxmlSurfer; switching between browser tabs dynamically switches the dialog threads. This approach helps preserve the dialog context in every tab so that, when the user switches back to an already visited tab, s/he can find it in the same state in which it was left. The module currently supports Microsoft Speech Recognizer (MicrosoftSpeech 2009) and the SAPI compliant text-to-speech engines, *e.g.,* (Cepstral 2009). Further work is needed to support more input/output devices such as Braille displays and note-takers. Section 3.4 provides more details on the vxmlSurfer architecture and implementation.

[1] W3C's XML standard for specifying interactive voice dialogues between a human and a computer.
[2] *DTMF* or Dual Tone Multi-Frequency is used to create the touch-tones corresponding to telephone keys.

3.4 vxmlSurfer[1]

VXML (VoiceXML 2009) is a standard XML format for specifying interactive voice dialogues between a human and a computer. VXML is a powerful language that can describe complex audio dialogs with multiple applications employing speech recognition grammars, embedded JavaScript, etc.

VXML is widely used to describe menus in telephone systems. It can be employed to disseminate information to the public through phones or computer terminals. For instance, (Kazunori 2003) describes an interactive bus-information dialog system in Kyoto City, Japan. The GEMINI project uses VXML dialogs in a multilingual interactive natural language interface (Hamerich 2004). VXML can also be used in computer games (JSmart 2009).

Another application of VXML is in non-visual voice browsing, where Web page content is converted to VXML dialogs to enable blind people to browse the Web, e.g., (Ramakrishnan 2004; NetEcho 2009). Popular screen-readers such as (JAWS 2009) and aiBrowser (Miyashita 2007) speak out the content of Web pages in a straightforward manner and do not require complicated dialog management systems. They provide very little interactivity or control, and their entire interface has to be modified to accommodate any changes. On the other hand, systems using VXML dialogs are more flexible because every VXML dialog is a dialog manager in itself. HearSay employs VXML dialogs to enable interactivity.

A VXML interpreter is required to process VXML dialogs. Regrettably, there are no complete, open-source VXML interpreters. The available open-source interpreters either support only a small subset of VXML or have other drawbacks. Commercial interpreters are not extensible, even though some of them allow free evaluations for research purposes, *e.g.,* OptimTalk (OptimTalk 2009). Besides, most VXML interpreters are geared toward telephony applications, *e.g.,* (NuanceCafe 2009).

To satisfy our need for a flexible VXML interpreter, we have developed an open-source, modular, multi-platform, extensible VXML interpreter, vxmlSurfer (Borodin 2006), which complies with the VoiceXML 2.0 specification. The interpreter is geared toward non-visual Web browsing to provide screen-reader users with more control over dialog flow, as they access the Web. vxmlSurfer is used as the Interface Manager of the HearSay system. vxmlSurfer supports most VXML tags and includes a full implementation of event- and variable-spaces.

To improve the user experience, the interpreter implements advanced controls that enable shortcuts for pausing, resuming, restarting the utterance, changing the pitch, voice, rate, and intensity of the speech, etc. Key strokes are treated as events that have predefined event handlers. Event handling is implemented in a way that allows vxmlSurfer clients to define new events and override default events, which makes the interpreter very flexible.

[1] Adapted from (Borodin 2006) awarded 2nd place in ASSETS'06 Graduate Student Research Competition.

To be platform independent, vxmlSurfer is written mostly in Java, with some JNI plug-ins to interact with external text-to-speech and speech recognition engines in Windows, Linux, and OsX. The interpreter is designed to be modular and to be able to use various speech-recognition and text-to-speech engines. The interpreter currently supports Cepstral (Cepstral 2009) and Microsoft text-to-speech engines and the Microsoft Speech Recognition engine (MicrosoftSpeech 2009). In its current configuration, the interpreter supports both keyboard and voice input.

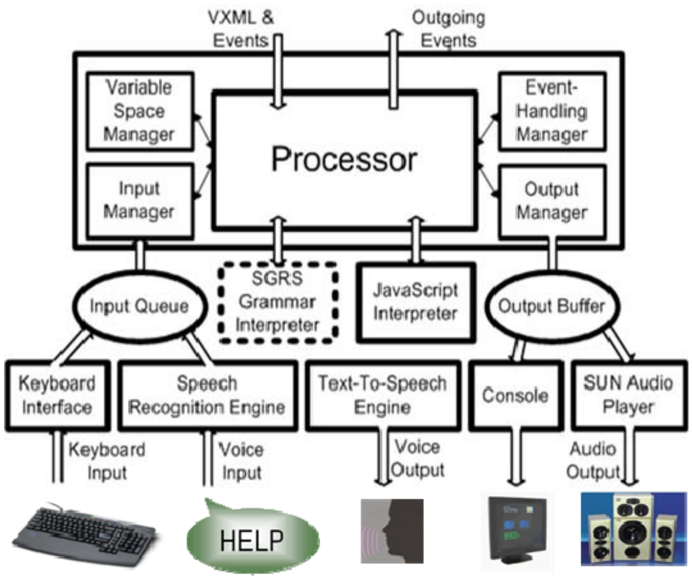


Figure 3 – Architecture of vxmlSurfer

The current architecture of vxmlSurfer is shown in Figure 3. The core of vxmlSurfer is the Processor, which receives and processes VoiceXML dialogs and events, and interacts with users. The Processor relies on the Event-Handling Manager to keep track of and fetch VoiceXML event handlers. It uses the Variable Space Manager to handle the scopes of variables defined in VXML dialogs. The variables are maintained in the Rhino JavaScript interpreter, which is also used for interpretation of JavaScript embedded in VXML. The Input Queue and Output Buffer handle multiple input and output modalities, including keyboard, speech input, speech output, visual output, and generic audio output. vxmlSurfer is still in development and is currently undergoing conversion to VoiceXML 2.1 standard and addition of SRGS[1] grammar support.

[1] Speech Recognition Grammar Specification (http://www.w3.org/TR/speech-grammar/)

3.5 Social Accessibility[1]

The Social Accessibility Network (SA) (Takagi 2008) is a collaborative framework that uses the power of crowdsourcing to improve the accessibility of existing web content. SA brings together end-users and supporters who can collaboratively create accessibility metadata. The HearSay's Annotation Manager module connects to the SA Service where it can obtain and post accessibility metadata.

When HearSay users encounter an accessibility problem in a web page, they can submit help requests. SA supporters are notified of incoming service requests through the Supporter Tool, which is a Firefox sidebar plug-in. The plug-in helps visualize user requests, as well as any other accessibility problems. By using the plug-in, the supporters can act on requests to fix accessibility problems. With every fix, the Supporter Plug-in submits the accessibility metadata to the SA Open Repository hosted on the Social Accessibility Server. Figure 4 shows the entire work-flow using an example: 1) a person with vision impairments reports an inaccessible image; 2) a supporter fixes the problem by adding alternative text; and 3) the end user listens to the alternative text of the image.

When HearSay loads a website, the Annotation Manager retrieves accessibility metadata associated with that website from the Accessibility Commons Repository. Next it transcodes the web page by applying the metadata to the browser's HTML DOM representation of the web page. The users can then browse the website and enjoy the improved accessibility of the content. In addition, users can create their own annotations, which are then submitted to the Accessibility Commons Repository. Work is underway to submit the results of the CSurf, Dynamo, and other algorithms to the SA service automatically, which will allow the users of other screen-readers to benefit from the metadata created by HearSay.

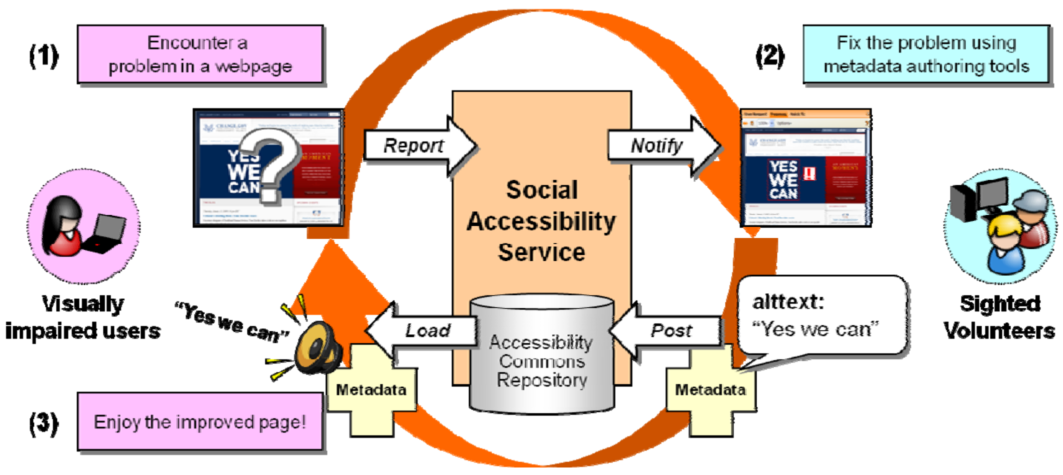


Figure 4 – Workflow of Social Accessibility

[1] Adapted from (Borodin, et al. 2010), a book chapter in *End-User Programming for the Web*.

3.6 Macro Manager[1]

Many productivity tools such as Microsoft Office already provide methods for recording and replaying macros to automate tedious tasks. Similarly, there are wrapper applications that use web crawlers to automatically extract the content of the "deep web" (Gandhre 2004). Although there exist macro-recorders for web browsers (Allen 2007; Leshed 2008; AutoMate 2009; iMacros 2009), they have complex visual interfaces, inaccessible or impractical to be used with screen-readers. Until recently (Borodin 2008a; Bigham 2009b), there have been no solutions that could help automate everyday web browsing tasks and provide a usable dialog interface to simplify non-visual web browsing.

The HearSay pipeline includes the Macro Manager module, which implements macro recording functionality. Macro recordings can be made visually and non-visually, as the recorder tracks keyboard shortcuts, mouse-clicks, and form filling. The macro-recorder tracks user actions and logs them in an XML format. Once the recording is saved, it can be replayed with the macro player by either entering or speaking the name of the recording. The player then interprets the recorded log and simulates the corresponding actions using the standard HearSay3 interface (see Section 4.4 for more interface details).

Every user action saved by the macro recorder is associated with some HTML DOM tree (HTML parse tree) element, on which the action was performed. This requires a method for DOM tree addressing that can reliably match the recorded action to the same DOM element every time the macro is replayed. Various DOM tree addressing techniques are explored in (Kawanaka 2008), but our current implementation uses only XPath, as the only method that can be used to uniquely address *any* DOM element.

XPath addressing is known to fail when the structure of the original page changes. Therefore, rather than capturing an absolute XPath, the system "learns" the shortest relative XPath which uniquely addresses a given DOM tree element (Gandhre 2004). This approach helps increase fault-tolerance to structural changes in web pages. Both automatic and assistive mechanisms for repairing broken macros may further increase fault-tolerance. Additionally, general purpose macros can also be shared among users contributing to the Social Accessibility project (Takagi 2008; Borodin 2010); and a shared repository of macros increases the chance that broken macros can be fixed in a timely fashion. However, sharing macros will require additional security mechanisms to protect users against possible scam and phishing attacks, *e.g.*, one could create a macro for a bank website that takes you to a replica website that phishes for user log-in information. More details on the Macro interface can be found in Section 4.4.

[1] The materials for this section were partially adapted from (Borodin 2008).

3.7 TeleWeb Architecture[1]

To meet the needs of mobile users, both with and without vision impairments, we developed TeleWeb (Borodin 2009) – an assistive voice-enabled application empowering users to access the Web remotely, through the most ubiquitous device – the phone. The uniqueness of the technology is that it enables users to access information from almost anywhere, using a plain old telephone, cell-phone, or smart-phone. TeleWeb users will be able to call their own personal phone numbers, authenticate themselves, and then search the Web remotely, check their email, and perform other web browsing tasks on their home computers by speaking commands and/or using the phone key-pad.

To operate TeleWeb users have to: 1) install the TeleWeb module for the HearSay browser; 2) install a free Skype client application; and 3) subscribe to a paid ($60/year) Skype-In service for a personal phone number (www.skype.com). TeleWeb module interacts with the Skype client application by sending text-to-speech output to Skype and forwarding the incoming audio stream to the DTMF and speech recognizers. TeleWeb will then provide a “bridge” to the HearSay browser, which can be used remotely via the phone. Figure 5 illustrates the TeleWeb concept.

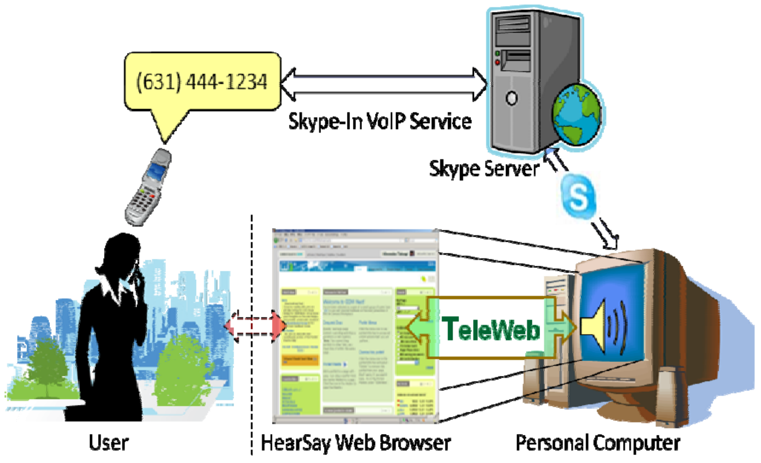


Figure 5 – Using TeleWeb

In contrast to many approaches that offer public telephony access to web-based information (Kumar 2007; GOOG-411 2009), TeleWeb does not require maintaining its own dedicated telephony service and can, instead, use third party telephony services such as Skype-In (www.skype.com) because information processing, text-to-speech generation, and speech recognition all happen on the end-user's computer. Compared to other remote access solutions for computer accessibility (Bigham 2008; SaToGo 2009), users do not require Internet connectivity to access the web browser at home. TeleWeb’s simple telephony interface, controlled by a combination of speech commands and telephone keypad shortcuts, can be used by people with a variety of needs, including older adults, individuals with cognitive and motor impairments, and anyone who wants to access the Web on the go. More details on the TeleWeb Interface can be found in Section 4.5.

[1] Adapted from (Borodin, Dausch, et al. 2009) – Judges Award at W4A 2009 Accessibility Challenge.

3.8 Future Work on the HearSay Browser

HearSay development. HearSay has come a long way from its inception to its current version. In the course of iterative improvements and redesign, HearSay has grown from a simple and unusable concept prototype into a reasonably stable and assistive technology that can already be used by blind people. Nevertheless, a lot of work remains to be done to improve the architecture of the system further.

Multi-platform support. HearSay3 can run on Windows and, with some limitations, on Linux and OsX. However, HearSay has to undergo a number of improvements before it becomes truly multi-platform. Even though Java is considered to be a universally multi-platform programming language, none of the mobile platforms offer full support of Java. One of the possible directions for HearSay enhancement could be the development of a server-based architecture that will allow users to run HearSay components on different devices, *e.g.*, a mobile device could run the interface portion of HearSay, while other components could be running on a server. An alternative direction of improvement could be migration to C++, especially because HearSay has to interact with native libraries to access text-to-speech and speech recognition engines. This would also make it run faster and facilitate the migration of HearSay to mobile platforms, which is very important for assistive technology users who want their software to have minimal latency.

Contextualization. As it will be demonstrated in the subsequent chapters, the utilization of user context has huge potential for analysis of web content and user actions. Making use of more contextual clues can help make non-visual web browsing a more useful process. User context can help infer what information is relevant for the user, determine possible further steps in web transactions, label images and other web objects, etc.

Automation is another cornerstone of efficient web access. Automation can streamline repetitive processes and tasks performed by users. HearSay's support of macros is only a first step in that direction, and, more work remains to be done to make automation seamless and intuitive. Together with contextualization, further automation has the potential to make non-visual web browsing more efficient and usable.

Reusability of components. Although the components of HearSay were initially developed for non-visual web browsing, most of them could be used in any Web page processing application. For example, CSurf, Dynamo, and the Segmentation Manager can be used by automated crawlers to analyze web content more intelligently. Any speech-enabled application could use vxmlSurfer as a dialog manager. The Macro Manager can be used to record and replay macros outside of HearSay. In summary, the modular components of HearSay leave ample opportunities for reusability and utilization.

4. Non-Visual Web Browsing Interface

In this chapter, I discuss a typical screen-reader interface in Section 4.1 and strategies for web browsing with screen readers in Section 4.2. I describe my innovative layered interface in Section 4.3, together with some other interface improvements that are not available in typical screen-readers in Section 4.4. I describe the TeleWeb interface in Section 4.5 and summarize other HearSay interface improvement in Section 4.6. I conclude by presenting a qualitative end-user evaluation of HearSay interface in Section 4.7 and an outline of future directions for the development of the HearSay interface in Section 4.8. The primary contributions of this chapter include: 1) a review of the standard browsing behaviors of screen-reader users; 2) innovative interfaces for layer-based information access, macro recording/replaying, telephony browsing, and other interface improvements; and 3) an overall qualitative user evaluation of the HearSay interface.

4.1 Standard Screen-Reader Interface

Most state-of-the-art screen readers employ a shortcut-driven interface, which allows users to navigate between webpage elements in a serial manner. For instance, “Up” and “Down” arrow keys are often used to skip between adjacent elements in the order they appear in the HTML source page. To navigate between elements of a specific type, *e.g.*, links, buttons, edit fields, headings, etc., users can press *[Shift+] A* | *B* | *E* | *H* | etc., to go to the [previous]next item of the respective type.

Screen-readers typically read out text content, as well as the types and states of complex web page elements, *e.g.*, “textbox blank”, “list with 10 items”, etc. Optionally, many screen-readers can also narrate text formatting and colors. “Left” and “Right” arrow keys allow users to spell out letters and numbers, when necessary. This is especially useful for reading unusual combinations of characters and editing web forms.

Screen readers often have one or more edit modes that allow users to interact with different types of form elements. For example, in the navigational mode, pressing “E” on the textbox simply moves the screen-reader cursor to the next textbox, without typing “E”. However, after pressing “Enter” on the textbox, the user can type in the textbox. Everything the user types is echoed aloud (except passwords), and when the user changes the position of the edit cursor, screen-readers read the character that follows the cursor. Pressing “*[Shift+]Tab*” moves the screen reader cursor to the next editable or focusable element. Some screen readers switch into the edit mode when their cursor is on an editable element.

To access elements of certain types (*e.g.*, links), the user can open an auxiliary window containing a list of all elements of that type (Figure 10). The user can then navigate the list to get an overview of the page, jump to the element selected in the list, or return to the initial position on the page. Some screen-readers support landmarks, allowing users to set a landmark and then return to it from anywhere on the page.

4.2 General Web Browsing Strategies

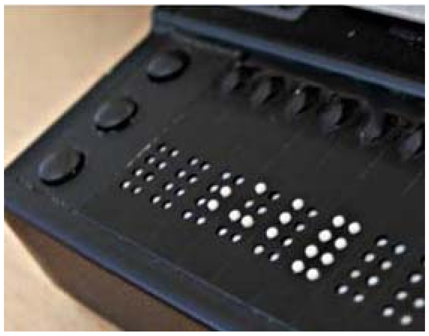
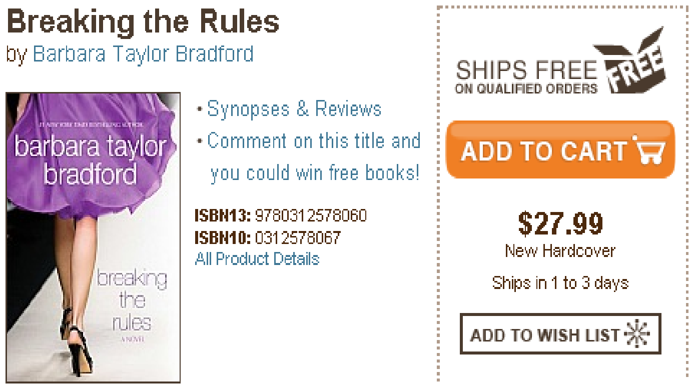


Figure 6 – Braille Display

Web access with screen readers has been, so far, inefficient due to the limitations of the serial audio interface. Refreshable Braille displays (Figure 6), which are supported by some screen-readers, allow users to read web content in Braille. The displays typically have a low resolution of only 40-80 cells. Such displays do not offer much improvement, other than providing an alternative modality that is especially important for deaf-blind people. Expert users also find Braille displays more convenient for text editing tasks. However, the biggest problem in non-visual browsing remains the speed of information processing. To overcome the limitations of screen-reader interfaces, blind users develop browsing strategies that allow them to browse more efficiently. In this section, I describe some general browsing strategies that were observed in the course of the user studies described later in this chapter.

To accelerate reading, many users speed up the speech rate with which the screen-reader narrates the content. Newly-blind individuals prefer naturally sounding voices with normal speech tempo, which are easier to understand. However, expert users speed up the speech rate and choose older format-based speech synthesizers whose quality does not degrade at a high speech rate compared to that of newer concatenative speech synthesizers. Congenitally blind users who grew up listening to synthesized speech are able to understand it at an astounding speed of up to 500 words per minute. An important aspect of speed is also the responsiveness of the screen-reader to key presses. Expert users expect the delay between a key press and the following speech output to be under 50-100ms.

Screen-reader users often develop favorite strategies for browsing web pages. For example, on unfamiliar web pages, many users first try to get a general overview of the page by navigating all headings with an "H" key. If they cannot locate the information of interest, they return to the beginning of the page to read through the entire page. Screen-reader users tend to remember the order of headings and other "landmarks" on frequently visited web pages. Then, by using the landmarks as points of reference, users can quickly navigate to the part of the page that they want to read. For example, knowing that the product price often precedes or follows the *"add-to-cart"* button, screen-reader users may press "B" to find the button and then easily find the price (Figure 7).



Breaking the Rules
by Barbara Taylor Bradford

• Synopses & Reviews
• Comment on this title and you could win free books!

ISBN13: 9780312578060
ISBN10: 0312578067
All Product Details

SHIPS FREE ON QUALIFIED ORDERS FREE

ADD TO CART

$27.99
New Hardcover
Ships in 1 to 3 days

ADD TO WISH LIST

Figure 7 – Shopping for Books

Browsing strategies also vary based on the task at hand. For example, in form filling tasks, screen-reader users press the "Tab" key to iterate over focusable elements, or press "E" to navigate to the next editable element. While reading headline news, the users may press "A" to iterate over the headlines links to get an overview of the latest news (headline news are now increasingly labeled as headings). Users can also choose to navigate over visited or unvisited links. If they know about some specific text occurring on the page, users can sometimes search directly for the text. If the screen-reader can store user-created landmarks on a web page, *e.g.*, beginning of main content, users can directly jump to the location indicated by the landmark.

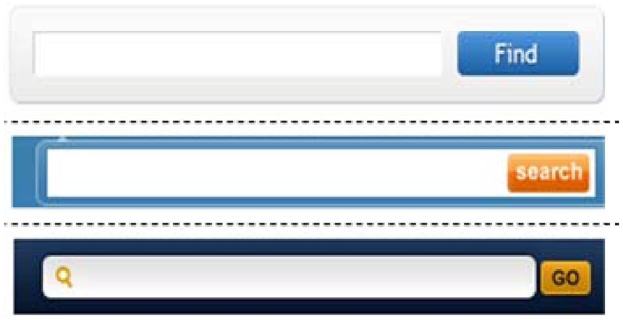


Figure 8 – Variations of Search Boxes

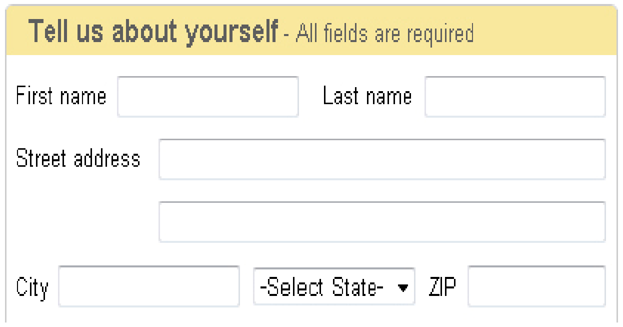


Figure 9 – Filling Out an Address Form

Typically, inaccessible content includes image-links and buttons without alternative text, form elements without labels, dynamic content and widgets, Flash, etc. Sometimes users manage to infer the roles of image buttons by exploring other web elements nearby, *e.g.*, search boxes are followed by search buttons in Figure 8. In other cases, users will follow an image-link to read the title of the target page. On subsequent visits, screen-reader users may remember the ordinal position of the link they need. While editing unlabeled form elements, users can often guess the label by using their prior experience with other forms. For instance, the typical order of address forms is First Name, Last Name, Street Address, City, State, and Zip (Figure 9); so, if there is an unlabeled textbox between the "Street Address" and "City" textboxes, then it is likely to be the second line for the "Street Address". By examining the content of the form element, it is sometimes possible to guess the purpose of the element, *e.g.*, the "State" form element is often represented by a list-box with the choices of the states. When all fails, screen-reader users have to ask for help from their sighted friends and family. The Social Accessibility (Takagi 2008) network was created to bring together end-users and volunteers who can collaboratively create and utilize accessibility metadata. The network allows end-users to submit requests for help with inaccessible content and allows volunteers to suggest strategies for overcoming problems, as well as label inaccessible content (Section 3.5).

Many users avoid visiting dynamically changing web pages because screen readers still do not adequately support dynamic content. In the cases where this may not be possible, *e.g.*, dynamic form-filling validation, users manually refresh the screen-reader's buffer and look for the changes. A general strategy for looking for statically changed information is to review the list of unvisited links (*e.g.*, look for new headlines on self-refreshing news sites – Figure 16) or read through the entire page (*e.g.*, looking for error messages in form filling – Figure 15). A more detailed overview of strategies employed in dealing with dynamic and static content changes is given in Section 5.2.

Users often try to avoid reading through menus and other irrelevant content that appears at the top of the page. One of the primary strategies for the beginning of main content is the use of the “H” key to navigate over headings, which, if used consistently, can substantially improve the accessibility of a web site. Another common strategy is searching for text that is expected to occur in the main section. A more detailed overview of strategies employed in finding main content is given in Section 6.2.

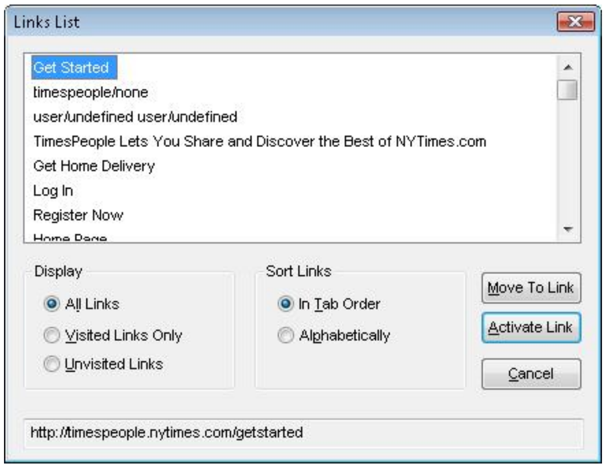
Although all users develop favorite web browsing strategies, the existence of sophisticated error-handling strategies among screen-reader users indicates that there are still a number of web accessibility problems. It is my hope that this overview of browsing strategies sheds some light on current Web Accessibility problems and will help researchers in the field focus on solving these problems. The results presented in this dissertation are a first step in this direction.

4.3 Layered Interface

In this section, I overview the limitations of the existing list-view interface used by screen-readers and describe an innovative layered interface for non-visual web browsing. The layered interface extends the functionality of the list-view interface implemented in many screen-readers, and lets users create custom views by combining layers of different types of web content. The proposed interface unifies navigation over all types of content by: 1) considering content of each type as a separate layer, and 2) providing options for filtering content by type. In contrast to list views, the layered interface allows users to browse personalized views with regular navigation keys while easily switching to the "normal" view to explore the rest of the content.

4.3.1 Limitations of the List Views

As described in Section 4.1, the list view interface in screen readers provides users with the ability to navigate elements of certain types as a list, *e.g.*, headings or links (Figure 10). A list-view can help users get an overview of a page without being distracted by other content. At the same time, list-views leave little room for customization; for instance, screen-readers do not offer an easy way for creating lists with combinations of elements, *e.g.*, a list showing all elements that are either headings or links. One of the reasons why such functionality has not been offered until now is the limited number of available keyboard shortcuts and a large number of possible element combinations. Providing too many features will unnecessarily complicate the interface and make it unusable for the novice to average screen-reader user, while most of the features will be rarely used. This makes screen-reader interface designers think twice before providing more features; consequently, they only implement the most frequently used features. This may explain why, for example, most screen readers provide a list of links, but not a list of buttons.


Links List

Get Started
timespeople/none
user/undefined user/undefined
TimesPeople Lets You Share and Discover the Best of NYTimes.com
Get Home Delivery
Log In
Register Now
Home Page

Display: [✓ All Links] Visited Links Only, Unvisited Links

Sort Links: [✓ In Tab Order] Alphabetically

http://timespeople.nytimes.com/getstarted

Figure 10 – List of Links in JAWS

Incidentally, not all elements can be put in a list; list-views only make sense for accessing web page elements that naturally form a list. For example, all text content, or all changed content, does not logically form a list. Further, once inside the list view, there is no easy way to explore the context of list items. For instance, having found one of many "*learn more>>*" links on the page, the user can follow the link, but s/he cannot easily explore the text that precedes the link. Screen-readers allow users to jump from the element in the list to the element on the page, but once out of the menu, there is no easy way to return to the menu and continue reading from the same menu element. With all of the above limitations, there is a clear need for a better interface that is more usable and flexible than the existing list-view interface.

4.3.2 The Concept of Layers

The content of any web page can be thought of as a combination of multiple layers, where each layer contains only elements of a given type, *e.g.*, links, headings, form elements, and so on. For example, Figure 11 shows how a Google News webpage (a) can be decomposed into 3 different layers: form elements (b), headings (c), and links (d). As can be seen from the illustration, some content may be present in several layers at once.

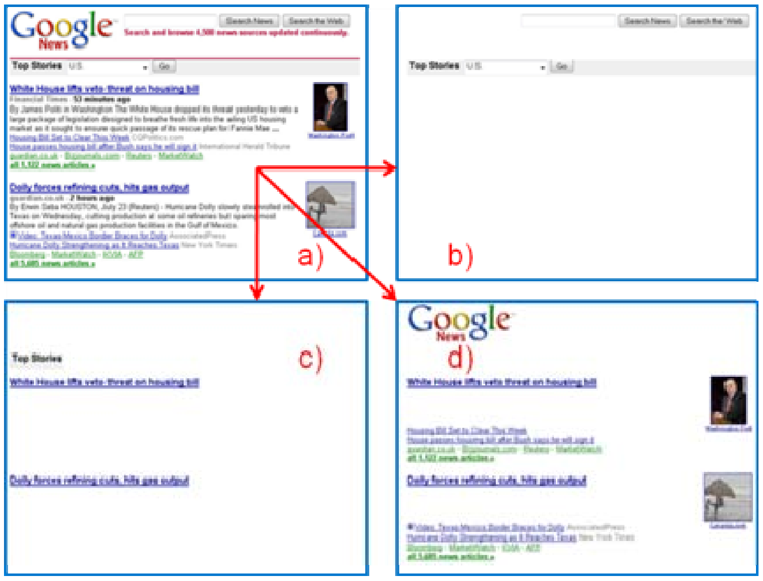

a)
b)
c)
d)
Top Stories
White House lifts veto threat on housing bill
Daily forces refining cuts, hits gas output

Figure 11 – Google News web page in Layers

By combining the layers, which is equivalent to using the “OR” operator, one can create custom views of content. For example, all elements that are either links or headings can be accessed by combining layers (c) and (d) in Figure 11.

HearSay by default supports layers of all elements that traditionally have element-specific navigation shortcuts (Section 4.1), *e.g.*, layers of headings, links, form elements, images, etc. HearSay additionally supports layers of complex types that cannot be put in a list view, *e.g.*, dynamically-updated or statically-changed content. In principle, any element attribute distinguished by HearSay can be used to create a separate layer, *e.g.*, hidden content. Layers can be turned on and off independently or combined to create complex views, *e.g.*, a layer with elements that are either updated or are headings.

HearSay allows users to navigate and search only within the currently active layers. For example, in the layer of links in Figure 11 (d), users can navigate link by link by either pressing “*A*” or “*Down*”; by pressing “*Ins+Down*”, a standard JAWS shortcut to start reading continuously, users can read the content of the entire link-layer, without having to press shortcut keys repeatedly. Users can quickly turn on all layers and explore the rest of the content, *e.g.*, having found an interesting article in the layer of headings in Figure 11 (c), users can turn on all layers to read the article summary in Figure 11 (a).

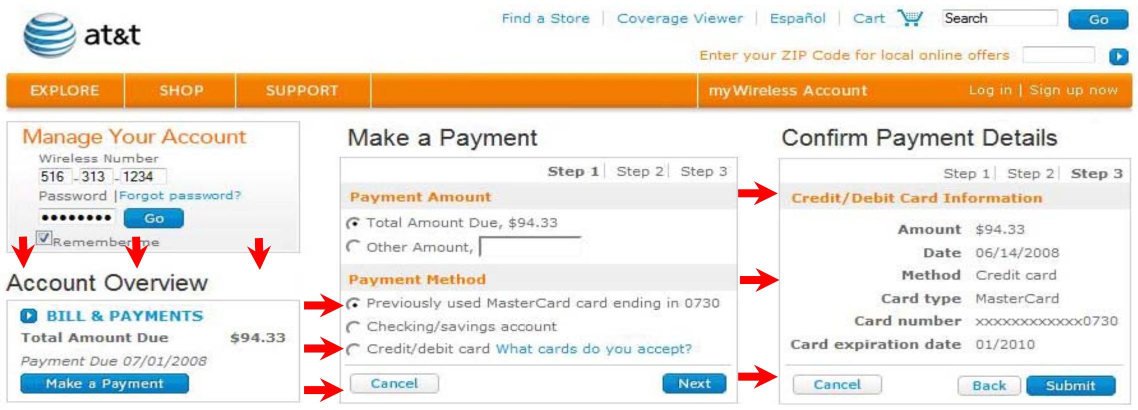
The layered approach provides a unified framework for tying together the results of all algorithms presented in this dissertation. In the future, identifying layers with changed content may prove to be useful for sighted users as well. A recent Microsoft Research publication suggested visually highlighting differences between web pages (Teevan 2009), and an Apple development team implemented basic layers in the (VoiceOver 2009) screen reader. Expert screen-reader users might enjoy the flexibility of creating custom views with logic operators, *e.g.*, “*Changed OR links*” or “*Heading AND NOT Links*”.

4.4 Macro Interface[1]

HearSay allows users to record macros to automate commonly performed tasks. A macro can record various user actions, *e.g.*, clicking links and buttons, filling out forms, etc. A macro can be recorded so that the exact actions can be replayed on the same web page, where the macro was recorded, *e.g.*, paying a cell phone bill at the AT&T webpage (Figure 12); or it can be recorded so that the same steps can be replayed on any web page that has the same structure, *e.g.*, having recorded purchasing of a product on Amazon, one could use the same macro to purchase any other product from the same web site.

While some transactions can be performed end-to-end without stops, confirmations, or progress feedback, others may require some customization. For example, users can make generic and more secure macros for logging in to secure websites by requiring that the user name and password be entered manually (Figure 12); then, the player will pause and wait for the required input to be provided. To make the player report the status and provide feedback, users can enter custom prompts or specify which part of the page should be read. The recording can be paused manually while it is playing, or pauses can be built in during the recording phase.

Paying a cell phone bill at the AT&T website can be confusing for a blind person, but, having done it once with macro recorder turned on, users can replay the same macro in subsequent uses. When the user instructs HearSay to play the AT&T bill payment macro, HearSay opens a new tab and loads the AT&T wireless website (Figure 12). The macro player then enters the phone number and password (Figure 12-a), and logs into the user's account. At the account overview page (Figure 12-b), the player clicks the "Make a Payment" button. On the following page (Figure 12-c), it makes and reads the selections: "Total amount due $94.33" followed by "Previously used MasterCard card ending in 0730", and then clicks the "Next" button. On the last page of the transaction (Figure 12-d), if suitably customized, the browser reads: "Confirm Payment Details," focuses on the "Submit" button, and says: "Press 'Enter' to submit."


at&t
Find a Store | Coverage Viewer | Español | Cart | Search | Go
Enter your ZIP Code for local online offers
EXPLORE | SHOP | SUPPORT | myWireless Account | Log in | Sign up now

Manage Your Account
Wireless Number 516 - 313 - 1234
Password | Forgot password?
•••••••• Go
☑ Remember me

Account Overview
BILL & PAYMENTS
Total Amount Due $94.33
Payment Due 07/01/2008
Make a Payment

Make a Payment
Step 1 | Step 2 | Step 3
Payment Amount
● Total Amount Due, $94.33
○ Other Amount,
Payment Method
● Previously used MasterCard card ending in 0730
○ Checking/savings account
○ Credit/debit card What cards do you accept?
Cancel Next

Confirm Payment Details
Step 1 | Step 2 | Step 3
Credit/Debit Card Information

Amount	$94.33
Date	06/14/2008
Method	Credit card
Card type	MasterCard
Card number	xxxxxxxxxxxx0730
Card expiration date	01/2010

Cancel Back Submit

Figure 12 – Steps of paying cell phone bill at AT&T Wireless

[1] The materials for this section were partially adapted from (Borodin 2008).

4.5 TeleWeb Interface

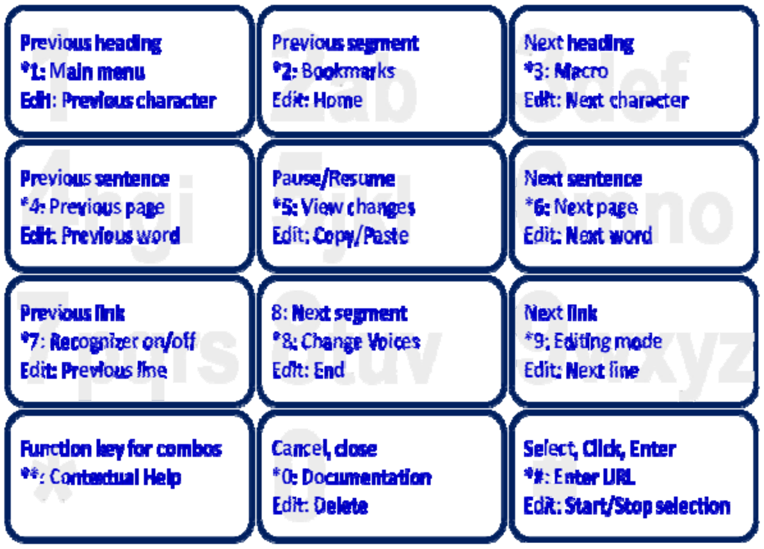


Figure 13 – TeleWeb Key Mappings

To make the TeleWeb interface more flexible and to enable efficient control over web browsers, it was designed to support phone keys, speech commands, and voice-enabled macros. The touch-tone interface has the lowest latency (50-100ms). However, since most of today's phones have only 12 keys that generate touch-tone signals, the available keys were mapped to the most frequently used actions (Figure 13). To minimize memorization, semantically opposite actions involving navigation to the next/previous item were assigned to keys located on the opposite sides of the phone keypad. Specifically, going to the next/previous heading was mapped to 3/1, the next/previous sentence to 6/4, the next/previous link to 9/7, and the next/previous segment to 8/2. To preserve the conventional semantics of the pound (#) key, we reserved it for selecting, clicking, or pressing "Enter." And from the remaining phone keys, we chose 5 to pause/resume reading and 0 to close/cancel/escape.

To increase the number of actions that can be performed with the keypad, we introduced a telephony menu where actions can be selected from a list and reserved Star (*) for key combinations. Pressing *1 opens the main menu, which allows users to listen to and quickly select available actions. Pressing *2 and *3 opens bookmarks and macro selection lists, respectively. Pressing *4 and *6 takes users to the previous and next web pages, respectively, while *5 allows users to review the differences between web pages. Pressing *7 turns on/off the speech-recognizer, which may not be desirable in noisy environments. Pressing *8 changes text-to-speech voices, as voices can be less or more intelligible for different environments and different users. Pressing ** provides contextual help for the current state of the TeleWeb interface. Pressing *0 opens TeleWeb documentation. And, finally, pressing *# allows users to edit the address bar.

TeleWeb also supports speech commands. A voice interface, although slower, is more efficient for complex tasks and those tasks that are performed rarely, such as opening bookmarks; however, every action has one or more synonymous speech commands associated with it. With improving speech recognition accuracy, it may soon become possible to use voice for text entry. Voice macros enable recording and then replaying complex browsing tasks, thus improving the efficiency of the interface.

4.6 Other Improvements in the HearSay User Interface

HearSay was designed to target specifically non-visual web browsing. It does not aim to replace general-purpose screen readers, but rather work in parallel with them, while providing a better interface to web browsing. The basic interface of the HearSay browser is similar to that of screen-readers. HearSay uses a standard set of navigation shortcuts (Section 4.1) to lower the learning curve for new users. However, HearSay builds on top of the basic interface to provide more features and make non-visual web browsing more efficient. In this section, I describe additional features of the HearSay interface.

Reviewing updated web content. The Dynamo interface adds shortcuts to review dynamic and static changes in web pages. The shortcuts allow users to navigate to the previous/next group of updated content. In addition, the changes can also be reviewed in the respective layers. Finally, the Dynamo interface tries to reposition the HearSay reading cursor after page refreshes to help users stay focused on what they were reading before the page refresh. For more details on the Dynamo interface, please refer to Section 5.5.2.

Accessing relevant content. When users navigate from page to page, context-directed browsing makes HearSay-CSurf start reading web pages from the main content, thus helping users avoid menus, ads, and other irrelevant content that happens to precede the main content. Section 6.7.1 presents more details on context-directed browsing with CSurf and the user interface for finding the beginning of main content.

Additional levels of navigation. HearSay uses a segmentation algorithm to separate web content into logically-related sections, providing users with an additional level of navigation. With the help of shortcuts, users can move back and forth among the segments, which can complement or even replace heading navigation, as headings are not used consistently in web pages. The segments also allow HearSay to group content when it is sent to the text-to-speech synthesizer. Synthesizing content in chunks, rather than element by element, helps generate speech with correct intonation and avoid pauses in the middle of a sentence, *e.g.*, due to links.

The use of earcons. Most screen readers use verbal prompts to indicate content elements, *e.g.*, “*link* Post Comment” or “*image* Sunset in Grand Canyon”. Many screen-reader users have complained about the distraction created by verbal prompts, especially while reading sequential text. For instance, Wikipedia articles can sometimes have every other word linked to some other article, making screen readers read:

> *link* 1852 – *link* Count Cavour became *link* prime minister of the *link* Kingdom of Piedmont-Sardinia, which *link* soon expanded to become the *link* Kingdom of Italy.

Therefore, HearSay implements *earcons* – short audio clips that are used to indicate certain types of content or events (Brewster 1993). By using both verbal prompts and earcons together, novice users can learn to associate certain types of content with corresponding earcons, *e.g.*, a typewriter sound is used for buttons, *ding* sound for links:

link 1852 – *link* Count Cavour became *link* prime minister of the *link* Kingdom of
ding *ding* *ding* *ding*

Piedmont-Sardinia, which *link* soon expanded to become the *link* Kingdom of Italy.
ding *ding*

By removing the verbal prompts, users can listen to text uninterrupted by the word "*link*" pronounced before every link:

1852 – Count Cavour became prime minister of the Kingdom of Piedmont-Sardinia,
ding *ding* *ding* *ding*

which soon expanded to become the Kingdom of Italy.
ding *ding*

Languages and voices. HearSay supports SAPI compliant voices, which include and are not limited to: Microsoft TTS, Dectalk, Cepstral, RealSpeak, etc. Although HearSay's interface has only English prompts (system messages), with these voices, HearSay can read web content in up to 20 different languages. If foreign voices are installed on the computer, HearSay uses the language tags in web pages and its own statistical language-detection algorithm to change voices and read content in the original language. In pages that mix languages, such as Wikipedia, HearSay seamlessly switches between voices as it reads the content.

Speech commands. In addition to keyboard shortcuts, HearSay also supports speech commands. Although using speech commands may be inefficient for element-level navigation, they can relieve users of the need to memorize a multitude of rarely-used shortcuts. Since the number of possible features can easily exceed the number of available shortcuts, a command-driven interface can give users quick access to features which, otherwise, can only be accessed through a structure of nested menus. In addition, a multiple speech commands can be assigned to the same feature, making the interface more intuitive as users can guess commands (Appendix A), instead of memorizing them.

Labeling and Social Accessibility. HearSay enables users to assign labels to any web page elements. In this way, for example, users can label search image buttons which may be completely inaccessible if they do not have alternative text (Figure 8). User annotations are then stored in the Social Accessibility (Kawanaka 2008) database so that, if any user visits the same web page, HearSay can retrieve the annotations and apply them on that page. HearSay's interface enables users to easily navigate between existing labels, rename them, delete them, and create new labels.

Co-existing with screen readers. Since HearSay does not enable general-purpose screen-reading, many users will continue using screen readers for tasks, other than web browsing. HearSay provides users with a setting that allows users to automatically turn off JAWS, when a web browser is in focus, and use the rest of computer applications with the JAWS screen reader.

4.7 Evaluation of the HearSay Interface

Over the past two years, various features of the HearSay browser have been demonstrated to over 100 blind users at various accessibility conferences, trade-shows, workshops, and presentations at disability support centers. Nearly 100 blind users have been recruited to participate in alpha testing of HearSay. Over 30 blind users have taken part in the HearSay user studies, conducted with the purpose of evaluating various features of the HearSay browser. This section overviews qualitative findings and user feedback about the HearSay user interface.

From the beginning of my work on the HearSay project, I have worked in close consultation with expert screen-reader users Jerry Randell (Helen Keller Services for the Blind) and Glenn Dausch (Stony Brook University Disability Support Services), who have provided me with their invaluable feedback. Their experience as both trainers and users of assistive technologies helped them acquire deep insights into the needs of screen-reader users. Regular consultations with Mr. Dausch and Mr. Randell have helped me clarify development priorities and stay on track in the development of HearSay.

Controlled user studies have been conducted to evaluate the Dynamo interface and its ability to help HearSay users cope with dynamic updates, refreshes, and changes in web pages. The evaluation has demonstrated that the Dynamo interface has significantly improved subjects' ability to locate dynamic changes in web pages, stay focused on content in case of page refreshes, and avoid reading repeated information while browsing template-based websites. Detailed results of the Dynamo evaluations appear in Chapter 5.

User studies have also been conducted to evaluate context-directed browsing with CSurf. These user studies have shown that HearSay significantly shortens the time users require to find main content in web pages. The study has also demonstrated that having to read irrelevant content somewhat impaired ability of blind users to retain information. More findings obtained during the evaluation of CSurf can be found in Chapter 6.

Most screen-reader companies have not been able to keep up with rapidly developing web technologies, and still have basic interfaces that simply read the screen, without trying to analyze web content. Therefore, most HearSay study participants welcomed any intelligent features that could save their time in web browsing, provide access to previously inaccessible content, and help them do more than they could with screen-readers. Apart from Dynamo and CSurf, users also praised HearSay's ability to create macros, recognize content language, label form elements, etc.

HearSay user studies have involved the use of the layered interface or simulations of the layers. Although some of the study participants found the concept of layers hard to grasp, most users enjoyed the easy access to updated content provided through the layered interface. The layered interface also allows participants to access lists of links and headings right on the page, instead of in a separate window. However, screen-reader users also wanted to have some options provided by the list view interface, *e.g.*, the ability to alphabetically order the list of items and press any key to access the list elements starting with the corresponding letter.

Although HearSay user studies conducted so far did not involve the use of other screen-readers, participants have voiced their concerns about using HearSay together with their screen-readers. Because HearSay only makes web-browser windows accessible, users who want to do more than browse the Web also have to use their standard screen-reader. Using two self-voicing systems simultaneously can be a serious usability problem, as users may not be able to understand either of them. In addition, screen-readers often prevent system key events from getting to the browser where they are interpreted by HearSay. Although any screen-reader company would prefer it if users would utilize only the products of that company, they do provide some facilities for pausing their screen-readers. For example, the JAWS (JAWS 2009) screen reader supports scripts that make it inactive for certain applications, the Window-Eyes (Window-Eyes 2009) screen reader provides a set of APIs to interact with it, and the VoiceOver (VoiceOver 2009) screen reader can be paused with a keyboard shortcut.

One of the common complaints expressed by the expert screen-reader users who tried HearSay was the responsiveness of HearSay to key presses and a lack of the formant-based Eloquence synthetic voices that are available with the JAWS screen-reader. Trying to perform web browsing tasks faster, power users often increase the speech rate up to 500 words per minute. The extreme speech rates often make concatenative speech synthesizer unintelligible; in addition, it takes time to get used to a new voice at high speeds. Eloquence, on the other hand, while sounding artificial, has lower latency and less degradation at high speech rates. Also, many power users find it annoying if a screen-reader does not start reading immediately after a key press. Although HearSay supports any SAPI5 compliant voice, only Microsoft voices provide low latencies of approximately 100ms, which is still more than that of Eloquence voices.

Overall, people who tried the HearSay browser left positive feedback about its features. End-users especially liked the intelligent features of Dynamo and CSurf. Many of the people who tried HearSay wanted to have a copy of our software that they could take home. However, before the HearSay beta version is made available for evaluation, more work is required to make HearSay more usable and responsive to key presses. Many users have also inquired whether there are any plans to support ARIA, HTML5, Flash, PDF documents, etc. Finally, evaluators have requested a number of features that would make HearSay more similar to their favorite screen readers. Although we have diligently implemented many standard screen-reader features in HearSay, a number of end-users wanted their favorite features that may not have been implemented, or implemented differently, in HearSay. Since many interface features also differ between popular screen-readers, it may be eventually possible to make interface profiles that match various screen-reader features and key combinations, thus reducing the learning curve and easing the adoption of the HearSay browser.

By the end of 2009, an alpha version of the HearSay browser will be released to a limited group of up to 100 alpha-testers who are expert users of screen-reading software. The group of alpha testers has been recruited in the course of user studies and demos at the CSUN International Technology and Persons with Disabilities Conference. The goal of the alpha testing phase is to collect anonymous feedback from a larger subset of the target population, determine further directions of research, and give end-users the opportunity to request new features.

4.8 Future of HearSay Interface

Universal Accessibility. My long-term vision for HearSay is universal access and multimodality. According to the principle of *Universal Accessibility*, the Web has to be accessible to anyone, anywhere, and in any use scenario. Therefore, to become universally accessible, HearSay will have to provide efficient web access to people with different abilities and needs. HearSay will have to efficiently support various modalities of input and output. It will have to transform from assistive software into an everyday tool, in which accessibility is a built-in feature that can be personalized for anyone's specific needs. In the meanwhile, smaller steps will have to be taken to get HearSay closer to this grand vision.

Dialog interface. HearSay has to evolve its interface from being command-driven to truly interactive. Therefore, HearSay has to learn how to maintain a conversation, summarize, and answer questions about the content presented in web pages. HearSay should also employ more types of audio feedback and sonification, besides earcons, to use more concurrent channels of information and, in this way, accelerate web browsing by delivering more information per unit of time. By combining speech and various sound effects, users will be able to use more than one stream of information.

Touch interface. With the growing popularity of touch interfaces, HearSay should also support gestures. Gestures may be more intuitive than keyboard shortcuts and, hence, be easier to remember. A touch-based interface can also give blind users a frame of reference and orientation in 2D web pages, as opposed to not knowing which part of the page exactly the screen reader is reading (Schmandt 1998). HearSay should also support a variety of haptic displays, starting with the simplest and already commonly used Braille displays.

Visual feedback. To achieve universal accessibility, HearSay should also provide magnification for partially-sighted users, as well as visual feedback for all. For example, HearSay's ability to identify updated content could be combined with usable highlighting; making these types of content easier to see may appeal to people with cognitive disabilities and anyone who wants to focus only on updated content.

Remote accessibility. HearSay has to become ubiquitously accessible on a variety of platforms, devices, and situations. It has to be accessible to people with situational disabilities (*e.g.,* drivers). In summary, it has to be accessible to anyone and anywhere.

User-centric goals. Finally, to make non-visual web browsing more efficient, HearSay should better maintain and use the context of user actions, minimize access-time to relevant information, keep users focused on tasks and information, facilitate multi-tasking and concentration, and enable automation of repetitive tasks.

5. Dynamo: Using Content Changes as Context[1]

I motivate the problem of the accessibility of dynamic web content in Section 5.1. I describe common strategies employed by screen-reader users to deal with dynamic content in Section 5.2 and overview related work on handling dynamic content in Section 5.35.2. In Section 5.4, I expand the definition of dynamic content and offer a unified view on dynamic content. In Section 5.5, I introduce the architecture of Dynamo and the underlying custom Diff algorithm that analyzes dynamic and static changes in web pages to help users find dynamically updated information, filter out unchanged content, and stay focused in case of page refreshes. I then describe automated (Section 5.6) and human subjects experiments (Sections 5.8-5.11) that confirm the validity and usability of the Dynamo approach. The chapter concludes with a discussion and future work on Dynamo in Section 5.12.

5.1 Introduction

The Web is evolving from a collection of static pages into a platform for interactive web applications. To facilitate this transformation, web developers are increasingly using such technologies as DHTML, AJAX, and Flash to make their web sites more interactive. Popular screen readers, however, have lagged behind the new technologies, resulting in a widening accessibility gap between the ways sighted and blind users browse the Web. As a result, accessing dynamic web pages remains difficult, frustrating, and sometimes impossible for people with vision impairments. And JavaScript-powered web page updates are just the latest symptom for the underlying trend of inaccessible web content.

With the arrival of Web 2.0 and the trend toward web pages that behave more like applications, the W3C WAI has developed a standard for Accessible Rich Internet Applications (WAI-ARIA), which enables screen readers to convey dynamic changes to users through its “live regions” specification. ARIA provides a standard method for assigning roles and states to DHTML elements, and for describing how such elements dynamically update. While popular screen-readers and web browsers are already beginning to support ARIA markup, they are currently unable to automatically identify and present to users the dynamic content changes that occur in web pages without the appropriate markup. Although this standard enables screen readers to convey changes to users, it requires that developers provide the annotation.

Although ARIA may be the ultimate solution, like most annotation standards, the ARIA approach will only work if the developers choose to implement it; unfortunately, most dynamic content available today does not implement the ARIA standard. Without the support of dynamic content, screen-reader users may not be aware of important

[1] Adapted from (Borodin, Bigham, et al 2008), awarded Best Student Paper Award at ASSETS'08

dynamic updates, occurring in the web pages they view, and, as a result, may not be able to perform online transactions such as shopping or banking. Observations of browsing patterns have shown that screen-reader users tend to avoid dynamic web pages altogether (Bigham 2007).

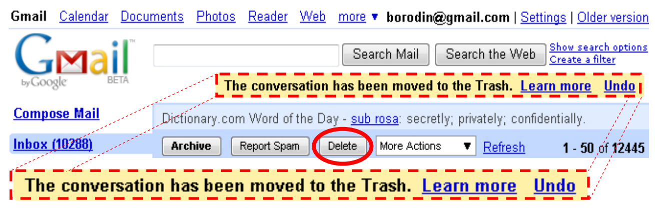
This Chapter presents Dynamo – a unified approach to handling content changes in web pages that is agnostic of the technical methods used to update the pages. Dynamo enables users to interact with most types of dynamic content, even the content which has not been annotated according to the ARIA specifications. In addition, Dynamo unifies the interface used to explore dynamic content with the interface used for other types of content changes that occur while browsing the Web. As a result, users need not know how web content is updated – whether it is via JavaScript, page refresh, or navigation; whether it happens on the client or server side – all of these are functionally equivalent.

1. The uniqueness and novelty of Dynamo is that it treats all types of page updates mentioned above as “dynamic” and handles them uniformly.
2. I describe HearSay-Dynamo (HD) – a prototype of the Dynamo approach in the framework of the HearSay non-visual web browser, specifically:
    a. I extend the buffer-based interface of the HD browser to partially update its VoiceXML dialogs as the changes occur in the web page.
    b. I propose the Dynamo-Diff algorithm to filter out content that does not change as a result of dynamic updates, including refreshes and page-to-page navigation.
    c. I provide a usable interface in HD to give users the ability to review page changes and control the level of intrusion caused by dynamic updates.
    d. I maintain user context in HD, and then use it to reposition users on the web page in case of dynamic updates, refreshes, and simple page-to-page navigation.
3. I confirm the effectiveness of the Dynamo approach and the Dynamo interface by:
    a. Reporting on the results of automated experiments with Dynamo-Diff, demonstrating high accuracy of identification of differences between web pages;
    b. Reporting on two separate studies, with 8 screen-reader users each, and demonstrating that my approach can help identify relevant information in web pages that are updated, allow users to stay focused on their tasks, and, consequently, save a significant amount of time in non-visual web browsing.

5.2 Common Strategies in Dealing with Content Changes

Although, according to (Bigham 2007), the most common strategy for dealing with dynamic content is avoiding it, sometimes dealing with dynamic changes is unavoidable. In this section, I discuss several different and yet similar scenarios that demonstrate the problems faced by screen-reader users and motivate the Dynamo approach.

Using current screen readers, users are not notified of dynamically appearing messages and do not have easy access to updated content unless it is marked up with ARIA. In some cases, users can simply ignore dynamic messages either because the messages are irrelevant or because they do not preclude users from accomplishing their tasks. For example, Figure 14 shows a dynamic message (dashed box) appearing on the Gmail.com web page as soon as one deletes an e-mail. In this instance, users can ignore this message because there are other ways of recovering a deleted e-mail, *e.g.*, going to the "Trash" folder and moving the deleted e-mail to the "Inbox." However, it would be considerably easier for users to simply jump to the update message and click the "Undo" link.



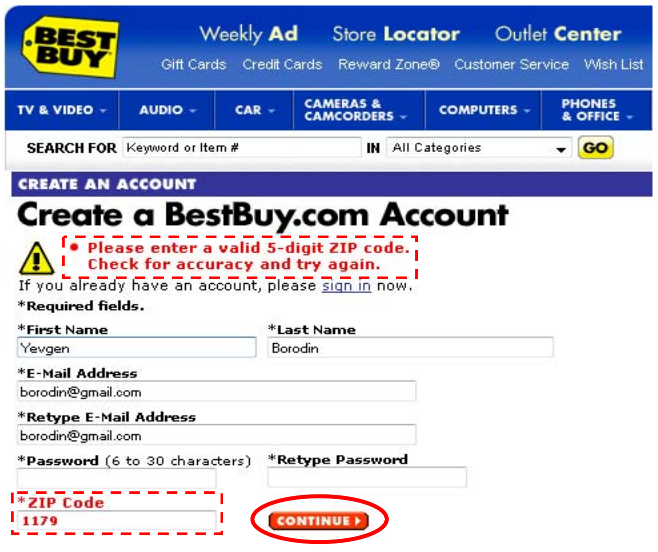
The conversation has been moved to the Trash. Learn more Undo

Dictionary.com Word of the Day - sub rosa: secretly; privately; confidentially.

The conversation has been moved to the Trash. Learn more Undo

Figure 14 – "Undo" link appearing at Gmail.com

A more serious problem is captured in Figure 15, where an incorrectly entered zip code results in error messages (dashed boxes) appearing on the page. Only the first message is actually new, the other one just changes the color to red to visually attract users' attention. Form validation can be done both dynamically (on the client-side) or statically (on the server-side). Either way, from the point of view of a novice user, the form did not submit because, at first glance, nothing has changed. A more experienced user would know about form validation and would tab through all fields to check for errors. However, the error may not be easily detected, especially by a novice, and tabbing through the form skips the error message describing the problem. If users had an easy way to review the changes, they would be able to read the error message and then jump to the highlighted field in no time at all. In fact, it should not matter to the users whether the changes resulted from a dynamic or server-side form validation. All that matters is finding the errors fast and moving on to the next task.



Create a BestBuy.com Account

• Please enter a valid 5-digit ZIP code. Check for accuracy and try again.

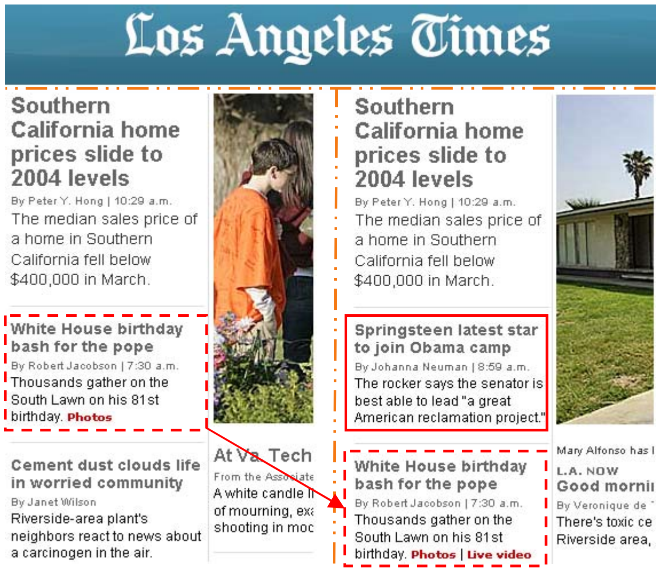
If you already have an account, please sign in now.

*Required fields.

*First Name: Yevgen	*Last Name: Borodin
*E-Mail Address: borodin@gmail.com	
*Retype E-Mail Address: borodin@gmail.com	
*Password (6 to 30 characters)	*Retype Password
*ZIP Code: 1179	

Figure 15 – Server-Side Validation at BestBuy

Figure 16 illustrates an automatic page refresh that causes changes in the content of the Los Angeles Times news web site. Some screen-readers such as JAWS have an option of suppressing automatic refreshes, but, in some cases, users may choose to refresh the page manually. Whatever the case may be, screen readers typically make users read the page from the beginning. Instead, not only should users be able to review the newly added content (solid box Figure 16b), they should also be able to continue reading what they were reading before the refresh. So, if the screen-reader's cursor was on the article about the Pope's birthday party before the refresh (dashed box Figure 16a), instead of starting from the beginning after the refresh, the cursor should be repositioned to the same article (dashed box Figure 16b). To users, it does not have to matter whether the content changed due to page reload or an AJAX update (the Gmail example above). In either case, users should be able to choose to continue reading from where they left off.


Los Angeles Times

Southern California home prices slide to 2004 levels
By Peter Y. Hong | 10:29 a.m.
The median sales price of a home in Southern California fell below $400,000 in March.

White House birthday bash for the pope
By Robert Jacobson | 7:30 a.m.
Thousands gather on the South Lawn on his 81st birthday. Photos

Cement dust clouds life in worried community
By Janet Wilson
Riverside-area plant's neighbors react to news about a carcinogen in the air.

At Va. Tech
From the Associate
A white candle li
of mourning, exa
shooting in moc

Southern California home prices slide to 2004 levels
By Peter Y. Hong | 10:29 a.m.
The median sales price of a home in Southern California fell below $400,000 in March.

Springsteen latest star to join Obama camp
By Johanna Neuman | 8:59 a.m.
The rocker says the senator is best able to lead "a great American reclamation project."

White House birthday bash for the pope
By Robert Jacobson | 7:30 a.m.
Thousands gather on the South Lawn on his 81st birthday. Photos | Live video

Mary Alfonso has l
L.A. NOW
Good mornii
By Veronique de
There's toxic ce
Riverside area,

a) Before b) After

Figure 16 – Page Refresh at L.A. Times New

Because the design of most web sites is based on templates, screen reader users have to learn how to use a number of strategies to skip repeated template content. Figure 17 shows an example of such a page, where, after following a link (indicated by the mouse cursor), only the central section (enclosed by the solid box) changes on the next page. Typical user strategies include: skipping to the nearest heading (JAWS shortcut "H"), which works only if the HTML source code implements heading tags; skipping to the next block of contiguous non-link content of a predefined number of words (JAWS shortcut "N"), which may or may not work, depending on the density of links in the template and main content; skipping to the next paragraph (JAWS shortcut "P"), which only works if the HTML source code implements paragraph tags, etc. Overall, the strategies used to avoid repeated template content are similar to those used to locate main page content (Section 6.2). An easy and reliable interface for reviewing changes in web pages could help users skip repetitive template content and proceed to browsing main page content. Finally, the interface has to be the same, regardless of whether the content is updated by statically loading every new page after following a link, or if the AJAX-like approach is used to change the main content dynamically, without actually reloading the template.

5.3 Related Work on Screen Reader Support of Dynamic Content

5.3.1 Manual and Automated Approaches Enabling Accessibility of RIAs

Most Rich Internet Applications (RIA) are currently accessible only to users who visually interact with the dynamic content. If web developers properly exposed states and transitions of their websites, screen-readers would be able to interact with rich content. Unfortunately, interactive web applications are built with a variety of technologies and toolkits, many of which make RIA web sites partially or completely inaccessible. Over the last few years, there have been several efforts to improve the accessibility of dynamic content by manual and automatic authoring of accessibility metadata. All of the approaches listed below can potentially enhance the accessibility of RIAs with HearSay-Dynamo; however, they cannot replace the Dynamo approach, which can enable interactive web applications even without the appropriate accessibility metadata.

ARIA markup. The use of the W3C standard for Accessible Rich Internet Applications (WAI-ARIA 2009) was one of the first attempts to make RIAs accessible. ARIA markup is intended to be used by screen-readers to improve the accessibility of web applications for screen-reader users. ARIA metadata can be embedded into web pages and can be used to describe live areas, roles, and states of dynamic content. For instance, "*<p id='tag' aaa:live='rude'>Interrupting Message</p>*" instructs screen readers to immediately read the content of the paragraph tag when it changes dynamically. Regrettably, most of the dynamic content available today does not implement the ARIA standard. Also, web developers are unlikely to follow ARIA consistently, for they have not followed other accessibility guidelines. ARIA can also be supplied as part of reusable components or widgets; for example, Dojo Dijit (Dijit 2009) provides ARIA-enabled widgets and a toolkit to build custom accessible widgets. However, Dijit is only one of many available toolkits for building RIAs, and web developers continue creating new inaccessible custom widgets of their own.

Transcoding approaches. ARIA can also be dynamically applied through transcoding. Originally targeting Google's own web pages, the Google AxsJAX project (Chen 2008) now enables any programmer to write scripts that will automatically inject ARIA markup into existing web applications, eliminating the requirement that the creator of the content provide the markup. The downside of this approach is that it still requires manual effort.

Working with HTML source code. The majority of search engines index RIAs by statically crawling web sites and extracting text from the HTML source code. Similarly, screen-readers can make dynamic content accessible in a static fashion by simply reading the HTML DOM tree. To illustrate, the elements of a dynamic pull-down menu may visually appear only on mouse-over, yet screen-readers often make the entire menu accessible at once. This approach, however, fully depends on the way the dynamic content is implemented. For example, AJAX delivers content on demand; hence, it may not be in the DOM tree unless some event triggering AJAX takes place and the screen reader detects the update.

AJAX crawling. An alternative approach that can help a screen reader collect information about a web page is to open RIAs in web browser and simulate various user events on all objects to expose the resulting system events and hidden content (Mesbah 2008; Duda 2009). Hypothetically, such AJAX crawling could automate ARIA metadata authoring. In reality, however, a crawler cannot often access all content and, in addition, consumes substantial machine-time while suffering from: state explosion, irreversibility of actions (which requires that transitions be retraced from the start state), latency between actions and reactions (especially, in AJAX applications), and inability to access password-protected web sites. For these reasons, HearSay-Dynamo is similar to the AJAX-crawling approaches only in the method used for detecting dynamic updates.

Collaborative crawling. Authoring of the accessibility metadata for RIAs could also be accomplished by semi-automated approaches such as collaborative crawling (Aggarwal 2002). The collaborative crawling technique enables the discovery of the underlying structure of the Web by mining the browsing traces of a multitude of users. While the original approach was employed for standard web crawling, it can also be applied to mining user browsing behavior for identifying dynamic content (Borodin 2010). The approach delegates the discovery of dynamic content to regular computer users, with the expectation that, eventually, users will try all allowable actions (transitions) and experience all possible system reactions, avoiding the problems of fully automated AJAX-crawling. As soon as this approach is implemented in the Social Accessibility (SA) framework (Takagi 2008), the SA participants, acting as “distributed spiders” crawling the Web and discovering dynamic content, could automatically generate ARIA-style metadata while casually browsing the Web. Although this is a promising approach, the protection of privacy and security of users has to be worked out.

5.3.2 Screen-Reader Support of Dynamic Content

Most voice browsers and screen-readers integrate themselves with regular web browsers such as Internet Explorer or Mozilla Firefox, to obtain web content in the form of HTML DOM trees. Web browsers often give access to extensive APIs through plug-ins, add-ons, or extensions, which provide an easy way to track changes in web pages by exposing DOM mutations (dynamic changes) and page-load events. For example, when a new web page finishes loading, a screen reader can use the page-load event as a clue to start reading the new page. Similarly, when a DOM mutation event occurs, a screen reader could identify and present changed content to the users. HearSay-Dynamo also receives such events from the HearSay extension installed in the Firefox and Internet Explorer web browsers.

Traditional screen-readers, such as JAWS, maintain a static representation of web content, or buffer, and allow users to browse within it. Until recently, screen readers did not update their view of a web page, except when a new page loaded. Although this buffer can be refreshed manually or automatically at certain time intervals, a survey described in (Bigham 2007) has shown that screen-reader users prefer to suppress updates and often end up browsing and acting upon stale content. On the other hand, the FireVox (Thiessen 2007) browser, reading directly from the HTML DOM, announces the content updates as they occur. However, without ARIA attributes, FireVox is unable to filter out irrelevant updates and often interrupts and distracts users from the tasks they are working on. HearSay-Dynamo dynamically updates its reading buffer and can optionally notify its users of dynamic events by playing earcons (Brewster 1993), which are less distracting than reading out changed content.

Part of the reason why dynamic content is still inaccessible to current assistive technologies is due to the difficulty of distinguishing between important content changes (*e.g.,* dynamic form validation) and irrelevant updates (*e.g.,* dynamic ads). While dynamic web pages can be confusing even to sighted users, screen-reader users have to interact with dynamic content through a serial audio interface; therefore, even important content changes can be distracting to users if the changes are presented as they arrive. To address this issue, the proposed Accessible Rich Internet Applications (WAI-ARIA 2009) standard has a notion of "live regions" where changes can occur. Web developers can then use the "*aria-live*" attribute with the possible values of "*off*, " "*polite*, " or "*assertive*" to specify the importance of updates, and whether they should be ignored, delivered at a convenient time, or announced to users. To the best of my knowledge, no usable interface currently exists for browsing dynamic content without ARIA markup.

Other types of inaccessible content include Flash and Java Applets. Screen-readers currently provide limited support for Java Applets and Flash objects. Flash often remains inaccessible to users because web developers forget to add alternative text to flash controls such as buttons. aiBrowser (Miyashita 2007) helps improve the usability of Flash content. Although both Flash and Java Applet accessibility is important for the future of the HearSay browser, only a small fraction of web sites implement these technologies; hence, supporting these technologies remains a secondary priority for the HearSay project. Nonetheless, with the appropriate APIs for Flash and Java objects, the Dynamo approach could be adapted to handle Flash as well as Java Applets.

The use of ARIA will provide the best user experience, but it will not be implemented consistently in every web application in the near future, as was previously shown on the example of earlier accessibility guidelines. Although HearSay-Dynamo does not currently support ARIA markup, it features a usable interface (Section 5.5.2) for reviewing both dynamic and static changes occurring in web pages which are not annotated with ARIA. Various types of web page updates (static page refreshes, navigation on template-based web sites, or dynamic JavaScript) have required that the user learn different techniques for coping with updates. Dynamo brings functionally equivalent web page updates under control of the same interface, providing a more consistent view to users.

5.4 Supporting Web Content Updates

Technologies addressing the problem of dynamic content can make more web sites accessible for people using screen readers. However, as the majority of the Web remains static, we should also continue improving the existing interfaces to make general web browsing more efficient and assist screen-reader users with finding information faster. In this section, I draw a parallel between static and dynamic web content and show that there is no need to consider them separately. Since the underlying problems are the same, any technical differences in their implementation need not rise to the user interface level – a usable interface for accessing dynamic content can, at the same time, be a usable interface for accessing static content. In this context, traditional browsing of static content is simply a special case of web page updates, in which an entire page may update at once.

5.4.1 Overview of Traditional Dynamic Content

With the advent of Web 2.0, the use of dynamic content in web pages has become ubiquitous. Some of the most popular technologies are JavaScript, DHTML, AJAX, Flash, and Java Applets, with the last two being beyond the scope of this work due to technical challenges involved in making them accessible. With many free tutorials and shared JavaScript repositories, it is now easy even for beginners to insert some script to spice up web pages with pull-down menus, dynamic tabs, etc. Dynamic content is used in most flight reservations and car rentals web sites; it is often used in web forms to point out errors in the fields; and it is also frequently used in ads.

It is worth mentioning, however, that not all dynamic updates are created equal. Depending on the type of dynamic content, changes may or may not be accessible with regular screen-readers. Very often web developers use dynamic content to pack more information in less space. For example, a modest menu with just 5 menu elements can easily hide 50+submenu items, which will pop out as soon as you mouse-over. Since most screen-readers get page content from the HTML DOM tree and do not filter out hidden content, such dynamic content is technically "accessible." Yet, it will be very difficult to navigate if web developers do not follow the ARIA or WCAG guidelines. In case of form-filling, all possible error messages are often hidden from sight, but not from the screen readers, which can read them all, thus making it impossible for screen-reader users to find actual errors without carefully reviewing the content of each form element. In addition, my observations show that, although hidden content visually appears in the correct position, it is often placed at the end of the DOM tree, often in no particular order. Since screen readers read content in the order it appears in the tree, screen-reader users are unlikely to understand the spatial or even sequential relationships that are visually obvious.

In other situations, dynamic content may be pulled from a web server by AJAX (Asynchronous JavaScript and XML), which updates portions of page content without refreshing the entire web page. Additionally, dynamic content may also be hard-coded in the JavaScript and thus made invisible to screen-readers. In both cases, to make the dynamic content "appear," a screen-reader has to refresh its buffer, hopefully without losing the position of its cursor, and having to restart from the beginning of the page.

Often users will end up browsing stale content or will not be able to access content at all, without knowing that the page has changed. Both of these situations could be addressed by the proper use of the ARIA guidelines; however, web developers are unlikely to do this consistently.

A review of the front pages of 33 major news websites (2008) revealed that nearly 50% automatically refreshed their front pages at certain time intervals, which could make screen readers lose focus; 30% contained an automatic slide-show with both images and text; and 18% had dynamic menus and tabs. Furthermore, none of the 33 news sites followed the ARIA standard for live regions, which would have enabled them to communicate the types of updates that should be expected to occur.

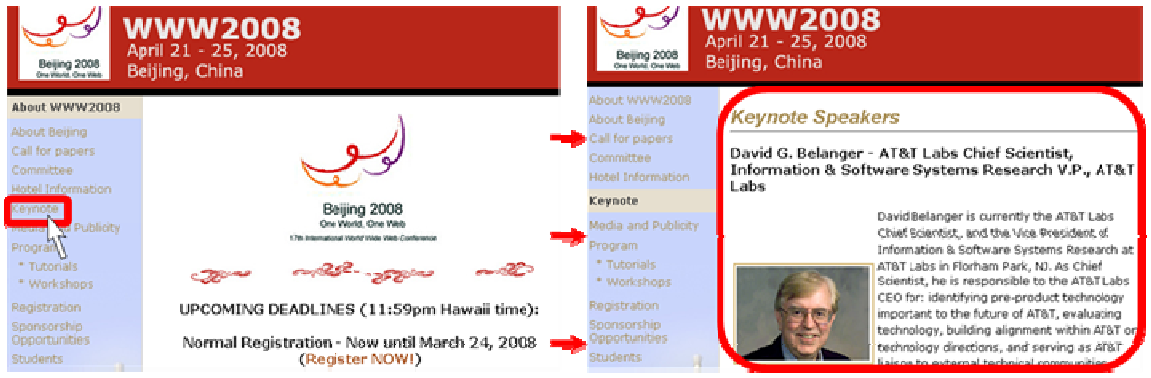
5.4.2 Content Updates: A Unified View

Although, traditionally, dynamic content is associated with technologies such as DHTML, AJAX, and Flash, I expand it to include: (i) page refreshes, both manual and automatic, which happen without warning and interrupt and confuse users (Figure 16); (ii) template-based web sites that only change part of the page as users browse from one page to the next (Figure 17); and (iii) links that bring up content in new frames and form submission pages that return the user back to the form, with the errors in form fields highlighted (Figure 15). Common among all these content changes is that the user performs an action, and the relevant content appears or changes somewhere on the page. Moreover, this content is often updated without notifying the user and often with no easy way to find the changes. The Dynamo approach unifies all of these types of web page updates by automatically detecting the underlying changes and providing users with a single intuitive interface for reviewing them.

Dynamic content is often used to pack more content in less space or attract attention to some information. For example, an entire website can be packed into one web page using dynamic tabs so that when a tab is clicked, the corresponding part of the content is made visible, and the rest is hidden. In the "wild," dynamic tabs can be often found in news websites. In most cases, dynamic content can be easily replaced with static solutions. For instance, simulated tabs are often used in the design of static websites such as YouTube, where clicking on the tabs causes the page to reload, but only the content inside the tabs changes. To give an example of an attention-grabber, many web forms use JavaScript validation to highlight fields or make error messages visible as soon as you tab out of a form field. On the other hand, other web forms perform server-side validation, reloading the same web page and highlighting the errors after you submit the form. In both of these cases, the same function can be performed by either dynamic JavaScript or static page load. Manifesting this technical difference to users is unnecessary and is likely to contribute to confusion.

Most static content can be replaced with AJAX so that, instead of requiring multiple pages to be loaded, the system can simply load new content and place it into the current page. GMail is one such example. Each bit of new information or every user action could require a full page refresh; however, because the site uses AJAX, the site appears to operate similarly to equivalent desktop applications. If we imagine that a web browser interface (with its menus, address bar, and buttons) is just a template that is common for all websites, we can then think of static web sites as web page updates appearing within this template, in which case any content change can be considered “dynamic.” In this context, traditional browsing of static content is simply a special case of web page updates, in which an entire page may update at once. For example, following a link and loading a new page can be thought of as a dynamic change of web content within one generic template of your web browser.

Regardless of whether web sites are implemented using dynamic or static approaches, users browsing for information usually care only about the information that has changed as a result of their actions. For example, we could use web browser APIs to detect an instance of a new dynamic tab having become visible, or an incorrectly filled form element having been highlighted red. By the same token, we could use some Diff algorithm to compare the previous and current pages, and identify static tabs as a part of the page design (Figure 17), or the error message (Figure 15) in the refreshed web form as the only difference. Again, people care about the information that has changed so that they can get their information, complete their task, and move on.



David G. Belanger - AT&T Labs Chief Scientist, Information & Software Systems Research V.P., AT&T Labs

David Belanger is currently the AT&T Labs Chief Scientist, and the Vice President of Information & Software Systems Research at AT&T Labs in Florham Park, NJ. As Chief Scientist, he is responsible to the AT&T Labs CEO for: identifying pre-product technology important to the future of AT&T, evaluating technology, building alignment within AT&T on technology directions, and serving as AT&T liaison to external technical communities

Figure 17 – Static Tabs in the Design of a Template-Based Web Site

To achieve this transparency and make browsing more usable, we can abstract from the specific web technologies and underlying methods for detecting content changes, and treat both dynamic and static content the same way. As a result, we can now design an integrated interface that will allow users to browse the web more efficiently and not have to worry whether they are browsing dynamic or static web pages. I next describe an implementation of this idea built into the HearSay-Dynamo web browser.

5.5 HearSay-Dynamo

In this section I overview HearSay-Dynamo (HD), which embodies the Dynamo approach within the framework of the HearSay non-visual web browser. I describe the architecture and interface that help users find changes in web pages, and, with them, find relevant information more efficiently. Finally, I survey related work on web page comparison and describe the Diff algorithm used by Dynamo.

5.5.1 System Architecture

As described in Chapters 3 and 4, HearSay converts web page content to VoiceXML dialogs and then uses vxmlSurfer (Section 3.4), a customized VoiceXML interpreter, to manage and interpret the dialogs. Various content-processing algorithms employed by HearSay's pipeline add attributes to the web page elements. The Dynamo module has access to all events received from the HearSay browser extension. When a new page loads, or a dynamic update arrives, the Dynamo module compares the new content to the previous page, groups content updates, and then dynamically updates the corresponding VoiceXML dialog.

All dynamic updates in any given page can, in principle, be implemented using AJAX. In fact, a whole web site can be implemented as one AJAX-enabled web page. Therefore, we decided to handle all page changes within one browser tab uniformly and extended the AJAX model to VoiceXML dialogs. This approach allows us to keep VoiceXML dialogs running uninterrupted and only refresh the content of the dialogs with new web page content or dynamic updates. This also allows us to unify the treatment of both static and dynamic pages, handling all cases uniformly by injecting the new content into VoiceXML dialogs in much the same fashion as AJAX updates web page content.

Not all dynamic updates result in content changes; for example, GMail refreshes the entire inbox just to add one email, and the New York Times front page reloads every 5 minutes, often resulting in no content changes at all. To handle these situations and filter out unchanged content, we implemented a custom HTML DOM Diff algorithm to verify if the content has changed, and then group the changes. Only the changed content then makes it to the user interface. I next describe Dynamo-Diff algorithm in detail and present the HD user interface for accessing dynamic content.

5.5.2 Interface for Accessing Content Updates

When a VoiceXML dialog is updated, the changed content is grouped and flagged as "new." Only the newly added or changed content will be made accessible through the Dynamo interface; the deleted content is removed from the dialogs transparently. HearSay-Dynamo (HD) provides a menu to consistently access the changes, and has several notification modes to control the level of update notification and, thus, distraction. At any time, users can access the updated content by pressing a shortcut that jumps to the next or previous group of updates, which works similarly to jumping to the next link ("A") or next heading ("H"). There is also an option to return to the original location after reviewing the updates.

HearSay takes a "layered" view of content, allowing users to isolate and combine different types of content by turning on/off layers with the corresponding content types (Section 4.3). The user can examine the changes by switching to the updates layer, which makes all unchanged content disappear and allows the user to review the changes with the standard set of navigation shortcuts, including those for jumping between groups of updates. At any moment, users can switch back to the default view and continue browsing the page from the current location. HearSay also allows users to clear reviewed changes so that it may be easier for them to find new changes that may happen due to dynamic updates. In the future, it may prove useful to provide a way to order the dynamic updates by time of arrival, or some other important criteria, such as those based on whether the updates were caused by users or the system.

HD can also handle page refreshes so as to minimize the distraction caused by the refresh and help the users stay focused on their current actions. Using the results of content comparison algorithm, HD repositions its cursor to continue reading from the same spot. To illustrate, 16 out of 33 major news websites in my review automatically refreshed their front pages, making it easy for users to lose focus if a screen-reader was to start reading from the beginning of the page every time the page refreshed. In other scenarios, users may choose to refresh the page manually; for example, if they want to check for any new messages in some static web-mail site quickly.

HD has several modes of notification. By default, the system will play an earcons (Brewster 1993) – a short sound clip to notify users of content changes. The notification earcon can also be suppressed to remove any distraction. In some cases, where dynamic updates are minimal, the user may prefer to have the updated content spoken out as it arrives, for example, in form filling with JavaScript form validation. In the future, it may also prove useful to control the frequency of update notification so that users are notified as soon as the content changes, with the following notification being held back for a certain time interval.

5.5.3 Related Work on Diff Algorithms

LCS-based algorithms. The development of text editors and the need to compare text versions were the original motivation for diff utilities. Most diff algorithms are based on the Longest Common Subsequence (LCS) approach (Bergroth 2000). Among the first diff algorithms described in the literature was the O(ND) algorithm (Myers 1986), which builds an edit graph (grid) between two documents and then looks for an optimal path with the least number of differences. For any documents A and B containing m and n tokens, respectively, the algorithm constructs an (m + 1) by (n + 1) grid and then adds diagonal edges between elements (x - 1, y - 1) and (x, y), if x-th token in document A is the same as y-th token in B. The algorithm then looks for an optimal path from (0, 0) to (m, n), which contains the maximum number of diagonal edges. The complexity of the dynamic programming implementation is O(N*D), where D is the number of differences, and N=|A|+|B| is the size of the combined documents. The algorithm provides refinements for linear space and promises fast execution for small differences of D << |A|. The LCS algorithms, however, do not guarantee 100% accuracy. Since most web pages are not all plain text, HtmlDiff (Berk 1996) improves the accuracy of the O(ND) algorithm by treating two HTML pages as sequences of tokens (sentence-breaking markups or sentences). HtmlDiff then uses a weighted LCS algorithm to find the best match between the two sequences.

XML-based comparison. Both the accuracy and complexity of diff algorithms can be improved if the hierarchical structure of HTML documents is taken into account. After a web browser parses HTML into a DOM Tree, an XML diff algorithm can be used to compare web pages. For example, X-Diff generates a minimum cost edit script to convert one XML document into another (Wang 2003). X-Diff uses the notion of edit-distance, which is calculated using max-flow matching of nodes in the tree. The algorithm uses XHash, a special hash function, to compare the nodes – two nodes are considered to be similar if they have the same XHash value. X-Diff uses the location of a node as the node's signature (similar to XPath). X-Diff uses a dynamic programming bottom-up approach and is capable of matching the whole sub-trees at once. It runs in $O(n^2*d*\log(d))$, where d is the maximum degree of any node in the tree. While the X-Diff algorithm works well for XML documents, it does not consider HTML-specific features. In contrast to XML-tree comparison, the Dynamo-Diff algorithm only looks for visual differences; it, therefore, does not consider all nodes of the tree, but only the spoken elements – the content nodes will be read out by HearSay.

5.5.4 Dynamo-Diff Algorithm

I designed the Dynamo-Diff algorithm to filter out content that does not change as a result of web page updates. The algorithm combines the advantages of the algorithms described in the previous section and introduces several improvements that make the diff algorithm more suitable for non-visual web browsing. Specifically, the Dynamo-Diff algorithm uses the LCS algorithm over the "spoken" elements heuristically collected from the DOM tree and compares them by matching their text. It can also compare attributes and/or node signatures encoding the paths of the elements in the DOM tree.

The HearSay pipeline linearizes the HTML DOM trees received from the browser by doing a Depth-First Search (DFS) and extracting the spoken elements from the DOM tree into a list. The spoken elements list includes all webpage elements that need to be verbalized by HearSay, *i.e.* text nodes, form elements, and alternative text. Figure 18 shows two (simplified) sub-trees from the DOMs corresponding to the web pages shown in Figure 16. The DIVs from each of the DOMs in Figure 18 will form a spoken-elements list.

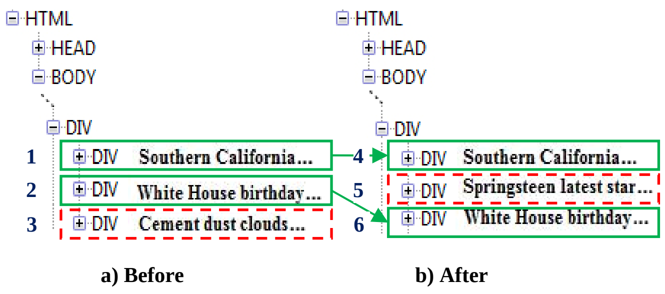

HTML
HEAD
BODY
DIV
1
DIV Southern California...
2
DIV White House birthday...
3
DIV Cement dust clouds...
a) Before
HTML
HEAD
BODY
DIV
4
DIV Southern California...
5
DIV Springsteen latest star...
6
DIV White House birthday...
b) After

Figure 18 – DOM Sub-Trees from L.A. Times

The two spoken-elements lists, corresponding to the two web pages, are then cross compared by creating the LCS matrix shown in Table 1. The matrix cells are filled with increasingly progressive numbers (weighted by the amount of text content) as more elements find their matching pairs. Then, a standard dynamic-programming trace-back approach is used to compute the optimal sequence of nodes; an optimal trace-back path is illustrated by red arrows in Table 1. By weighing the elements based on the length of the text contained in them, the algorithm avoids erroneous matching of unimportant nodes, *e.g.*, those containing separators and whitespace, which are abundant in web pages.

LCS	**1**	**2**	**3**
4	52	52	52
5	52	52	52
6	52	90	90

Table 1 – LCS Matrix

When matching the spoken nodes, the algorithm can optionally take into account text formatting, which may be important in cases such as form validation when errors are often identified by color highlighting (Figure 15). The HearSay plug-in extracts some of the formatting information such as font size and font color from the browser's internal data structures and adds it in the form of DOM node attributes. To make the comparison stricter, the Dynamo-Diff algorithm can also take into account node signatures (similar to XPath) composed of the combinations of all node names on the path from the root of the DOM tree to the node under consideration, *e.g.*, "/html/body/div/div/#text" is the signature of the #text node with the "Southern California…" under the DIV node in Figure 18. Signature-based comparison can help identify elements that are changing level in the DOM tree; to illustrate, sometimes the style of elements is changed because a formatting HTML tag is inserted as their ancestor, *e.g.*, "*<b>bolded text</b>*."

5.6 Automated Experiments with Dynamo

To verify the accuracy of the Dynamo-Diff algorithm, I designed a data collection tool that allowed users to highlight differences in web pages by hovering the mouse cursor over the web page elements that had to be highlighted. Using this tool, we collected over 100 pairs of webpage DOM trees serialized in the HearSay-compatible format – each pair from a different template-based web site. In each pair, one of the DOM trees had all of its differences from the other DOM manually highlighted.

We then constructed an experimental framework that allowed us to automatically push the DOM trees through the HearSay pipeline pair by pair, and automatically compute the accuracy of the Dynamo-Diff algorithm. We ran the same test with four different variations of the Dynamo-Diff algorithm: 1) comparing text content only; 2) comparing text and element attributes; 3) comparing text and element paths; and 4) comparing text, attributes, and element paths.

The summary of the results is presented in Table 2, which shows precision (P), recall (R), F-measure (F1) – all three based on the amount of text and percent of page text retrieved (%) for all four conditions that were tested with the Dynamo-Diff algorithm. Figure 19 visualizes the differences in the performance of the algorithms. As can be seen from the results, making the Dynamo-Diff algorithm stricter by adding more conditions reduced precision, as the algorithm found more differences than users were able to identify. Simultaneously, it increased recall since more text was retrieved. In all four conditions, the algorithm returned 77.4%-78% of the text from the test web pages as different.

With Text comparison in all cells	***w/o Attributes***				***with Attributes***			
	P	*R*	*F1*	*%*	*P*	*R*	*F1*	*%*
w/o Path	0.940	0.976	0.958	0.774	0.932	0.982	0.956	0.787
with Path	0.938	0.981	0.959	0.780	0.932	0.984	0.957	0.788

Table 2 – Accuracy of the Dynamo-Diff Algorithm

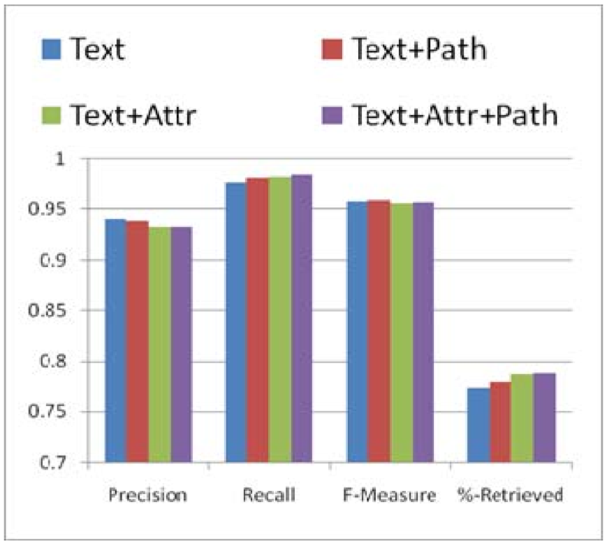


Figure 19 – Comparative Accuracy of Dynamo-Diff

A manually examined sample of web pages pairs that did not produce 100% accuracy showed that this less than perfect accuracy was mostly due to human error during data collection, or because LCS-based algorithm could not handle situations when two elements exchanged places. In cases like that, the algorithm picked the element that had more weight.

5.7 HearSay-Dynamo User Studies

I conducted two evaluations of HearSay-Dynamo. The 1st study explored the ability of HearSay-Dynamo to improve access to changes with 8 screen-reader users over the following 3 categories of content updates: dynamically appearing messages, form-filling with an introduced error, and page refresh (Section 5.8.1). In the 2nd study, 8 screen-reader users extensively evaluated HearSay's support for efficient navigation of web sites whose pages use a common template (Section 5.10).

Participants were recruited from email lists and local centers providing services for the blind. For the purposes of this study, "screen-reader users" will refer to participants who self-described as primarily using screen readers to browse the web. Participants varied in their screen reader proficiency from relatively inexperienced to expert. Eight participants (1 female) ranging in age from 18 to 56 participated in the 1st user study. Eight participants (1 female) ranging in age from 26 to 47 participated in the 2nd user study. Some, but not all, of the subjects in this study also participated in our 1st study described in Section 5.8.

The following two versions of HearSay were used in our evaluation: (i) HearSay-Basic (HB), providing basic navigation shortcuts and including the capability to transparently update its VoiceXML dialogs with new content, and (ii) HearSay-Dynamo (HD), having all functionalities of HB, but also notifying users of updates and providing navigation shortcuts for reviewing updated content.

I considered evaluating HD using a popular screen reader with which our participants would have already been familiar, such as the JAWS screen reader, but could not easily integrate the Dynamo approach into these proprietary software programs. I also considered evaluating HD against JAWS; however, some variations in shortcuts, text-to-speech engines, and interface familiarity could have introduced uncontrolled random variables and biased our evaluation. In addition, JAWS did not provide even basic access to all of the dynamic content used in the evaluation, thus making it impossible to use JAWS during the study. The subjects noted that both HD and HB were somewhat slower than JAWS, but, nevertheless, treated it as JAWS and expected it to have the same shortcuts as JAWS.

5.8 Experimental Setup for the 1st Study – Dynamic Content

5.8.1 Categories of Tasks

The following three categories of tasks were used during the 1st user study. The participants of the study performed the tasks in the following order.

Responding to a Dynamic Message. Tasks in this category required that subjects find and then click on a link that is dynamically introduced into a web page. Such dynamic messages are often used to convey updated information or to respond to a user-initiated action. For this task, participants first deleted a specified email in a GMail email inbox or a document in a Google Docs document selector view. In response to that action, both pages inserted a DIV element at the top of the page, confirming the delete action and providing a link labeled "Undo" for undoing the action. The task was to find and click the "Undo" link. The task time was measured from the time the participants hit the "Delete" button to the moment they clicked "Undo." HD alerted the participants with an earcon that the content on the page had changed and provided an interface for reviewing the changes. HB only alerted the participants of the updated content; however, they had to use standard shortcuts to find it. It is notable that JAWS does not detect the dynamically inserted content and does not update its buffer, so the "Undo" link would not be accessible through JAWS.

Form-filling with Error Recovery. For the next task, the evaluators completed a web form reproduced from either the BestBuy order form or Delta SkyMiles registration page. The form asked users for their name, address, phone number, and email address. Upon submission, the form programmatically introduced an error into either the phone number or zip code field (a digit was removed). The participant was then returned to the form with an error message inserted above the form and the field with the error highlighted in red. The task was to find the error and fix it. To ensure that users were able to complete the task in the allotted time, participants were informed that there was an error in the form they submitted. HD automatically detected that the page was updated with error messages, alerted the user of the changes with an earcon, and provided the interface for reviewing the errors. Emulating the behavior of a typical screen reader, HB provided no alerts; and the users had to find the errors using the standard navigation interface.

Maintaining Context on Page Refresh. The final task was to find the author of a specified article on the New York Times or Los Angeles Times home pages. These web pages periodically refresh their content as news stories are added. On page refresh, popular screen readers either begin reading at the top of the page or keep reading the old version of the page. For this task, a page refresh was triggered when the user read the title of the target article. HD detected the page refresh, automatically found the previous reading position, and restarted reading at the element where the participants had previously been reading. Emulating the behavior of a typical screen reader, HB restarted at the beginning of the page.

5.8.2 Procedures

Each participant completed 1 task from each of the 3 task categories (Section 5.8.1) with both HB and HD for a total of 3 x 2 = 6 trials. The 3 tasks performed with a particular version of HearSay (either HB or HD) were conducted sequentially. Both the system order (HB/HD) and ordering of the task groups were counter-balanced across participants using a 4-cell design.

The experiment used a 2 x 3 within-subjects factorial design with factors and levels as follows:

System	{HearSay-Basic, HearSay-Dynamo}
Task	{Dynamic Message, Form-Filling with Error Recovery, Page Refresh}

The dependent variable of interest was task completion time. An upper limit of 5 minutes was set per task, which applied to 2 out of the 48 total trials completed by all of the participants. The time was analyzed using repeated-measures ANOVA.

The following hypotheses were tested in this experiment: 1) Providing an interface to locate changes in web pages makes it easier to locate dynamic updates; 2) Navigation over content updates makes it easier to fill forms in case of errors; and 3) Repositioning the reading cursor in case of dynamic updates helps users maintain their focus.

Before beginning the study, participants practiced for 5-10 minutes to familiarize themselves with the shortcut keys used by HearSay. Participants practiced for an additional 5 minutes before the second session, during which they were introduced to the two keyboard shortcuts used to navigate between the update groups and the earcon, indicating a web page update.

To avoid learning effects (since participants completed each task twice), for each category, I chose two sites that I believed were similar in structure. Although the chosen sites were specific instances of each category, I believe the characteristics of each are representative of the types of updates in each category.

5.9 Results for the 1st Study – Handling Dynamic Content

5.9.1 Speed of Task Completion

Overall, HD decreased the time required for users to complete the tasks (Figure 20).

For the Dynamic Message task, the participants spent a mean of 94.5 seconds (std. dev.=100.8) using HB, and a mean of 36.5 seconds (St.dev.=27.4) using HD. This represented a significant effect of System on the task completion time ($F_{1,7}$=3.67, p<.01), indicating that HD improved participant performance. Although the participants varied greatly in time to complete this task, 7 out of 8 completed it more quickly using HD. P8 completed it more slowly because he forgot the shortcut key used to review web page updates and spent >30 seconds trying to remember it.

For the Form-Filling with Error Recovery task, the participants spent a mean of 192.5 seconds (St.dev.=94.9) using HB and a mean of 52.8 seconds (St.dev.=18.3) using HD. 7 out of 8 users completed this task more quickly with HD. P6 completed it more slowly because, although HD enabled him to reach the form label more quickly, he did not recognize the error as being an incorrect value. There was a significant effect of System on the task completion time ($F_{1,7}$=15.07, p < .01), indicating that HD improves this task.

For the page refresh task, participants required a mean of 34.9 seconds (St.dev.=16.9) with HB compared to a mean of 18.5 seconds (St.dev.=13.1) with HD. This represented a marginally significant effect of System on the task completion time ($F_{1,7}$=4.87, p=.06). Because the participants could use heading shortcut keys to skip between headline stories quickly, the task time was relatively short across both conditions.

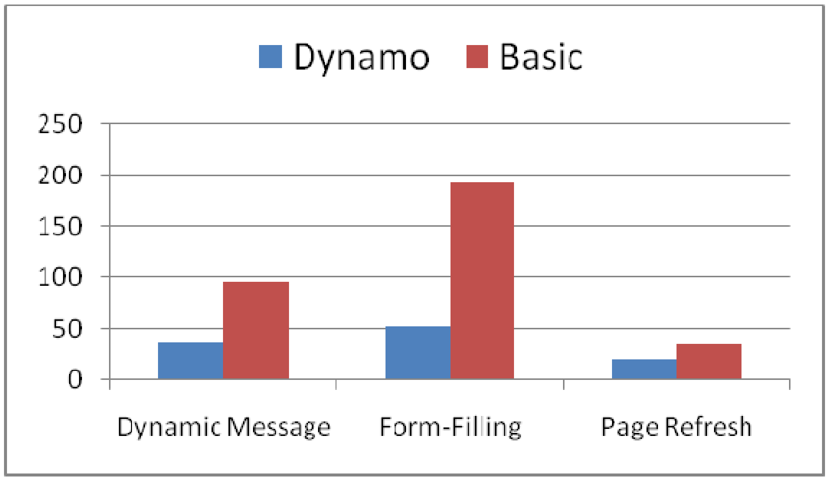


Figure 20 – Average time required to complete each task (seconds)

5.9.2 Subjective Feedback

Following each task, the participants rated the difficulty of completing each task on a 5-point Likert scale (1=Easy to 5=Hard – See Table 3). The participants consistently rated the dynamic message and form-filling tasks completed with HD to be easier than those completed with HB on average, but these differences were not significant, according to an analysis with the Wilcoxon Signed-Rank significance test (Wilcoxon 1945). The participants nearly uniformly rated the Page Refresh task to be Easy (1). The subject who did not become confused when the page began to refresh pressed shortcuts that skipped past the article before HD began reading again, and had difficulty returning to the prior location.

Task Category	HD	HB
Dynamic Message	2.00 (0.38)	3.14 (0.59)
Form Filling	2.14 (0.51)	3.57 (0.48)
Page Refresh	1.43 (0.43)	1.14 (0.14)
Average	1.86 (0.29)	2.62 (0.33)

Table 3 – Average 5-Point [1=Easy to 5 =Difficult] Likert scale values (St. Dev.)

At the end of the study, we asked participants to what extent they agreed with several statements on a 5-point Likert scale (from 1=Strongly Disagree to 5=Strongly Agree – See Table 4). Overall, the participants agreed that many different kinds of web page updates made web browsing more difficult using a screen reader; they thought that HD could improve access to web page updates and wanted to use these features of the system in the future.

General Statements	**Response**
Dynamic content makes web pages difficult to use.	4.14 (0.55)
Pages that refresh automatically are difficult to use.	3.86 (0.34)
It is often difficult to find errors when I make an error filling out a web form.	4.00 (0.22)
HearSay-Dynamo	**Response**
HearSay made interaction with dynamic content easier.	4.00 (0.38)
HearSay improved the tasks that I used it on.	4.29 (0.18)
I want to use the features of HearSay that handle dynamic content in the future.	4.43 (0.30)

Table 4 – Average 5-Point [1=Strong Disagree to 5=Strongly Agree] Likert scale values (St. Dev.)

5.9.3 Discussion

The participants generally found that dynamic content made browsing difficult for them. They found HB particularly frustrating because it required that they search for content using an inefficient linear search through the page. Many participants mentioned how useful it was to be able to skip to the parts of the page that had updated. Although other screen readers may provide more shortcuts than HB, none currently provides a shortcut to directly skip between pieces of updated content.

Several participants had suggestions on how the system could better convey the updated content to them. For instance, some wanted the system to automatically jump to the portions of the page that had updated. Although we developed the capability in HD to automatically jump to updated content, we did not evaluate it because we believed that overriding user intent would be too disorienting in general. In the tasks explored here, our handling of the updated content was consistently useful, which may have caused users to believe that the system was more knowledgeable than it was. One user mentioned that he would have preferred it if the system had announced the semantic purposes of the changes that had occurred. For instance, during form-filling, the system could have specifically announced "Errors were detected in your submission – please correct your telephone number". Such an announcement would have required semantic knowledge of the updates that had occurred, which is difficult to surmise automatically.

Because the headings were provided by the news web pages in the "Page refresh" task, the participants were able to quickly find the appropriate article and then its author. All but one participant was able to complete this task more quickly using HD. The aforementioned participant became disoriented when the page refreshed and quickly pressed the "next heading" shortcut twice in succession, which caused him to inadvertently skip past the correct article. After he had gone past the article, he became disoriented and finding the correct article required backtracking. Unpredictable system behavior can easily disorient and confuse users. However, had the participant been warned that the Dynamo interface repositioned the HearSay cursor after page refreshes, the participant would not have gotten disoriented. One avenue of improvement for the Dynamo interface is providing users with more feedback on what the interface is doing, at least in the verbose mode that is more appropriate for novice users.

The participants generally saw a need for better handling of dynamic content, and, as expected, most did not realize the technical differences between the tasks in our evaluation. My unified approach to web page updates does not require that users be aware of technical details of different kinds of web page updates.

5.10 Experimental Setup for the 2nd Study – Template Detection

To expand on our understanding of the benefits of HearSay-Dynamo, we conducted the 2nd, more in-depth study considering the ability of the system to help users navigate web sites that have a template design. Browsing template-based websites with normal screen readers can be frustrating because often the template content – navigation links, headers, advertisements, etc. – are repeated on each page. Without a quick way to skip them, screen-reader users can be stuck, repeatedly listening to the same content or figuring out a way to bypass it in each page they visit.

5.10.1 Categories of Tasks

Governmental web sites. The two tasks in this category required that the participants follow a link from one page to another on each of two template-based governmental websites: Medicare.gov and Irs.gov. After following the specified link, participants were instructed to find the answer to the following questions: "Who is eligible for prescription drug coverage" on the Medicare site and "What tax fraud is the IRS warning against?" on the IRS site. The questions were chosen so that their answers were approximately the same reading distance from the top of the page, and so that the answer to each of the questions was the second sentence (in the changed content), preceded only by a single heading. To control the experiment, the participants were not allowed to use the browser search feature or use the "skip-to-main-content links," if such were present in the web pages. Depending on the experiment, the subjects used HB or HD to complete the task; the HD interface allowed them to turn on the layer with the changed content and review the differences compared to the previous page. While measuring the task-completion time, we did not measure the time it took the participants to find the link on the first page.

Encyclopedia web sites. For the second task in this category, the participants were asked to perform searches on each of the two template-based encyclopedia websites: Wikipedia.org and Encyclopedia2.thefreedictionary.com. Again, the sites were chosen for their structural similarity. The subjects searched for "New York" in The Free Dictionary and had to answer the question, "Which states does the state of New York border on?" The participants searched for "Helen Keller" in Wikipedia and had to find the answer to the question, "When was Helen Keller born?" The questions were chosen so that answers to them would be found at approximately the same reading distance from the top of the page. In contrast to the tasks on governmental websites, the encyclopedia tasks required that the participants go through more content before they were able to find answers. Depending on the experiment, the participants used either HB or HD to complete the task; the HD interface allowed them to turn on the layer with the changed content and review the differences compared to the previous page. Task completion time was measured beginning from the page load.

5.10.2 Procedures

Each participant completed 1 task from each of the 2 task categories (Section 5.10.1), with both HB and HD for a total of 2 x 2 = 4 trials. The 2 tasks performed within a task category were performed sequentially. Both the system order and the ordering of the task categories were counter-balanced across the participants using a 4-cell design.

The experiment used a 2 x 2 within-subjects factorial design with factors and levels as follows:

System	{HearSay-Basic, HearSay-Dynamo}
Task Category	{Government Site, Encyclopedia Site}

The dependent variable of interest was task completion time. The time was analyzed using repeated-measures ANOVA. The participants were limited to 3 minutes in all tasks, affecting 3 of 32 trials, or 9% of the trials.

The following hypothesis was tested in this experiment: Providing a layer of changes between web pages makes browsing on template-based websites more efficient.

Before beginning the study, participants practiced for 30 minutes to familiarize themselves with the shortcut keys used by HearSay and were introduced to the template-detection functionality of HearSay. We gave all of the participants a long practice session because some of them found the concept of layers difficult to grasp.

To avoid learning effects (since participants completed each task twice), for each task, I chose two template-based web sites that I believed were approximately similar in structure for each category. Although the chosen sites were specific instances of each category, I believe the characteristics of each are representative of the types of updates in each category.

5.11 Results for the 2nd Study – Template Detection

5.11.1 Speed of Task Completion

Overall, HD decreased the time required for users to complete the tasks (Figure 21).

For the Government Site task category, the participants spent a mean of 69.2 seconds (std.dev.=62.77) using HB and a mean of 22.6 seconds (std.dev.=8.3) using HD. This represented a significant effect of System on task completion time ($F_{1,7}$=5.10, p=.05) indicating that HD improved this task.

For the Encyclopedia Sites, the participants spent a mean of 88.7 seconds (std.dev.=68.1) using HB and a mean of 38.9 seconds (std.dev.=44.1) using HD. There was a significant effect of System on task completion time ($F_{1,7}$=7.8, $p < .05$) indicating that HD improves this task. All but one participant completed this task more quickly with HD. Although HD enabled fast navigation of the page (16 seconds), P4 managed to complete the task with HB more quickly by skipping to the appropriate heading.

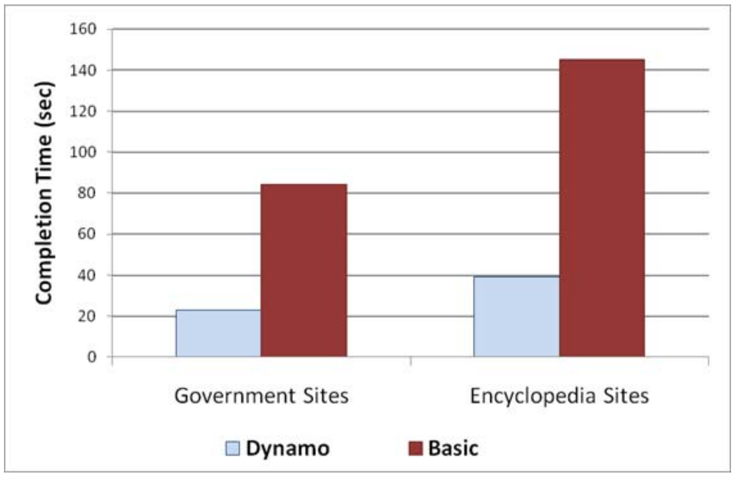


Figure 21 – Average time required to complete each task (seconds)

5.11.2 Subjective Feedback

Following each of the two tasks, the participants rated the difficulty of completing each task on a 5-point Likert scale (1=Easy to 5=Hard – See Table 5). Many of the sites used in the study could also be efficiently navigated using the existing shortcuts made available by the user's screen reader. For instance, users already familiar with Wikipedia were able to efficiently navigate to the main content using either the provided skip links or by selecting the heading of the appropriate level. Although HD did not provide an advantage in this case, it demonstrated that users of HD would likely experience a similar performance advantage of headings and skip links even on sites that did not use them. The participants unanimously rated the browsing task on the government websites as easy when they completed the task with HD.

Task Category	HD	HB
Government Site	1.00 (0.00)	1.75 (0.48)
Encyclopedia	2.00 (1.00)	1.00 (0.00)
Average	1.50 (0.50)	1.38 (0.24)

Table 5 – Average 5-Point [1=Easy to 5 =Difficult] Likert scale values (St. Dev.)

After completing the tasks, we asked participants to what extent they agreed with several statements on a 5-point Likert scale from (1=Strongly Disagree) to (5=Strongly Agree). Participants agreed that sites that use templates could be tedious to browse (Table 6), but also appreciated the familiarity that a consistent layout provided. The participants neither agreed nor disagreed that skip-links were an adequate solution. Some participants indicated that they often used skip-to-main-content links when such links were available. Depending on a particular site, other methods such as skipping by heading may be more appropriate. Or, skip links may often be broken, given that they are often not visible and, therefore, not checked as often (Bigham 2007). All participants agreed that HD made it easier for them to interact with template-based websites and improved the tasks on which HD was used. All participants wanted to use its template-detection feature in the future.

General Statements	Response
Sites that use templates are tedious to browse.	3.75 (0.95)
Skip-to-main-content links are an adequate solution to sites that use templates.	3.25 (0.25)
I usually use skip-to-main-content links.	2.25 (0.75)
HearSay-Dynamo	**Response**
The HearSay interface makes it easier to interact with sites that use templates.	4.75 (0.25)
HearSay improved the tasks that I used it on.	5.00 (0.00)
I want to use the features of HearSay that handle sites with templates in the future.	4.75 (0.25)

Table 6 – Average 5-Point [1= Strong Disagree to 5 = Strong Agree] Likert scale values (St. Dev.)

5.11.3 Discussion

All of the participants found the basic navigation shortcuts of HearSay both adequate in their functionality and comparable to those used by Jaws and Windows Eyes. They actively used the shortcuts made available by HearSay to skip between links, headings, and form elements.

The majority of the participants thought the tasks were easy, but they generally found template-based web sites “a nuisance to browse”. P3 remarked that template content often helped him confirm that he was on the right web site, but the others noted that they often spent up to half of their browsing time “getting through the clutter” by skipping through repeated content while looking for the main page content. The participants were unanimous in agreeing that HearSay’s template detection feature, together with the layer of changed content, would be a major improvement to how they were browsing the Web.

The majority of the participants said they were not frequent users of Wikipedia, although P3, who classified himself as a “geek”, admitted to being familiar with both the Wikipedia website and the general layout of the governmental websites. All of the participants stated that they disliked the Wikipedia site for abundance of links, which made reading through it with a screen reader difficult.

Several participants noted that they would rather have HearSay automatically start reading from new content on page load, while P1 suggested that inserting the custom “HearSay skip to main content” link, which was already on our to-do list, might have given him more control. However, the participants did not agree on the usefulness of the “skip to main content” links. It is notable that the use of skip-links was not correlated with the screen-reader experience as two screen-reader experts had opposite opinions: one found skip-links “working 80% of the time,” and the other preferred to skip to main content using heading and paragraph navigation shortcuts. Of the two non-expert users, one always ignored the skip links, and the other always used them. Notwithstanding the small subject pool, these findings suggest that screen-reader users tend to develop and then use different browsing strategies, depending on their training and personal experience.

5.12 Conclusions and Future Work on Dynamo

We proposed, implemented, and evaluated the Dynamo approach to making dynamic web content accessible and web page updates usable, a problem, which is still growing, spurred by the advent of Web 2.0. Two separate user studies with the HearSay-Dynamo system indicated that using the system improved their access to dynamic messages, page refreshes, form filling, and navigation of template-based sites.

This work has opened up several avenues for improving HD implementation and conducting additional research. First, the participants suggested several improvements to HD's user interface that can be explored to optimize the user experience. Second, although HD can detect web page updates, many participants wanted to know the semantic roles of those updates. Third, a similar approach can be applied to the accessibility of Java Applets and Flash. And, fourth, the collaborative approach of Social Accessibility (Takagi 2008) could leverage a community of users to label content and share the labels among themselves. I believe these goals can build on the foundation outlined here and make dynamic content efficiently accessible to screen-reader users.

The proposed framework goes beyond Web accessibility for screen reader users. Dynamic content also limits Web accessibility for web spiders and any software tools that assist in aggregating and filtering information on the web, providing custom views of web pages, and automating repetitive browsing tasks. Finally, detecting and visualizing changes in web pages can even improve regular browsing interfaces.

6. CSurf: Context-Directed Browsing with Link-Context

In this chapter, I present a novel technique that leverages link-context to find the beginning of main content in web pages as users are navigating from page to page. In this chapter I provide a motivation for the problem of finding main content (Section 6.1), report on strategies employed by screen-reader users (Section 6.2), and describe existing approaches intended to help screen-reader users find main content in web pages (Section 6.3). Further, I overview related work (Section 6.4-6.6), detail my solution to the problem (Section 6.7), and confirm the effectiveness of the approach by reporting the results of the automated (Section 6.8) and human-subjects experiments (Section 6.9-6.13). I conclude by outlining future directions of research (Section 6.14).

6.1 Introduction[1]

6.1.1 Problem Statement

As we browse the Web, we have to filter through a lot of irrelevant data. For example, most Web pages contain banners, ads, navigation bars, and other data distracting us from information. Sighted individuals can process visual data in no time at all, quickly locating the information that is most relevant to them. Conversely, for people with visual disabilities, who browse the Web with screen-readers, this task can be time-consuming and difficult.

Most screen-readers process web pages serially, *i.e.* they start reading from the beginning of the page and read it sequentially – the way it is laid out in the HTML code. In most cases, this means that the user will first read through the menus, banners, and commercials before reaching the beginning of main page content. Furthermore, the main content may not be contiguous, split by patches of irrelevant information such as ads, buttons, etc. These simple yet fundamental problems make web browsing strenuous and time-consuming. Having to battle with irrelevant content reveals the differences between the ways sighted and blind people browse the web and access information.

There exists a variety of approaches for helping screen-reader users find relevant information faster. These approaches range from providing accessibility metadata in web pages (*e.g.*, HTML headings, skip-to-main-content links, and HTML5 tags) to changing the layout of web pages to provide a reading order for web pages. However, as I detail in Sections 6.2 and 6.3, many of these approaches fail due to the carelessness of web designers.

In the meanwhile, screen-reader users have developed a variety of browsing strategies (Section 6.2) and listen to or skip over substantial page content before they get to the beginning of main content. Currently, there is no interface that allows screen-

[1] Partially adapted from (Mahmud, Borodin et al. 2007), nominated for Best Paper Award at WWW'07.

reader users to filter out irrelevant information, leaving the accessibility of web pages in the hands of web designers.

In this chapter, I propose a novel technique for finding main content in web pages without relying on the structure or the accessibility metadata built into web pages. In general, identification of main content on any single web page is subjective because it depends on the context of user actions and the task at hand. However, by following a link, the user provides the "context" that can help detect relevant information on the target web page. More specifically, the text of the link, and sometimes the text around the link, on the source page can be used to match and find the beginning of main content on the target page.

In this chapter, I address the problems of locating the beginning of main content in web pages and propose the context-directed browsing approach as a solution. Below is a summary of my contributions:

1. I describe a variety of existing approaches to helping screen-reader users find the beginning on main content in web pages and report on the associated strategies employed by the users.
    a. I extend our prior work CSurf-Old and present a CSurf-New approach for finding the beginning of main content by using link-context from the source page and matching it to the content of the target page;
2. I present a usable non-visual interface that helps quickly locate main content.
3. I confirm the effectiveness of the CSurf approach and the CSurf interface by:
    a. Reporting on the results of automated experiments with CSurf-New, demonstrating high accuracy of finding the beginning of main content;
    b. Reporting on the results of a user study with 16 blind subjects, and demonstrating that my approach can help identify main content in tasks involving navigation between web pages, and consequently, save a significant amount of time in non-visual web browsing.

6.2 Strategies for Finding Main Content

Screen-reader users tend to develop their own strategies for getting past irrelevant web page content. Based on our observation of browsing behaviors of 16 screen-reader users, we have identified several user strategies for finding main content in web pages: using skip-to-main-content links; heading-level navigation; keyword searching; skipping by paragraph or non-linked text; and sequential browsing (continuously reading or going line by line through web pages). Advanced users have their favorite strategy and a sequence of strategies they fall back on. The strategies may also vary based on users' familiarity with a web page.

Skip-to-main-content links. In an attempt to follow web accessibility guidelines, web developers often place hidden links that allow screen-reader users to skip navigation menus and start reading from the main content of a page. However, since these links are hidden, and developers are often using visual web-development tools, these same developers, or the newly recruited developers, fail afterwards to maintain the skip links. A survey of browsing behaviors (Bigham 2007) has shown that screen-reader users often ignore skip-links because the links are often broken. In a WebAIM survey (WebAIM 2009), over 1,000 users responded that they use skip links: whenever they are available 22%, often 16%, sometimes 28%, seldom 19%, and never 10%. In our own survey of 16 screen-reader users, 5 users almost never followed skip-links at all, 2 used them occasionally, and 9 used skip-links when they were available (Table 9). The ones that did not use skip-links motivated their choice by a lack of control of where the skip-links take them, combined with prior experience of seeing broken skip-links, and also, for fear that they will skip something important. Our user study tasks did not include skip-links.

Heading navigation. Perhaps the most common technique for skipping to main content is the use of heading-navigation shortcuts that jump to the next or previous HTML heading tag (<H1>-<H6>). In the WebAIM survey, over 1,000 users responded that they used headings: whenever they were available 52%, often 24%, sometimes 14%, seldom 5%, and never 2%. In our own survey, 10 (62.5%) of 16 respondents said that most web pages had reasonable heading structure, while 6 (37.5%) of the more experienced screen-reader users remained neutral (Table 9). In our experiments, however, only 7 participants tried to use heading navigation when looking for the main page content. None of our 16 respondents have tried to use heading-level navigation keys 1-6, which indicates that they did not trust the developers to use the correct heading levels.

Keyword search is another technique employed by users to find main page content. Keyword search helps users when they know what they are looking for, *e.g.*, after following a link to an article, one can look for the words occurring in the title of the article to jump directly to the article. In our experiments, only 4 out of the 16 participants used this technique, and for all 4 of them, this was the first thing they tried. The participants searched for word combinations, single words, or even letter combinations to find main content. This strategy also validates the technique for finding the beginning of main content by the CSurf, as discussed further in this chapter (Section 6.7).

Skipping to non-linked content in paragraphs. Because the irrelevant content often contains many links, some users employ the "*P*" shortcut to skip to the nearest paragraph of text. This technique works only if <P> tags are used in the HTML source code of the webpage. A more efficient strategy is the use of the "*N*" shortcut that allows users to skip to the next section of contiguous non-linked content, which may not work in web pages abundant in linked content. In our study, 5 (31%) of 16 participants have used the "*N*" shortcut, and only 1 user has tried the "*P*" shortcut to find main content.

Sequential navigation. When all other strategies fail, or if users do not have an efficient strategy, they resort to sequential navigation, *i.e.* they either read continuously or arrow down through the entire page. Sequential browsing is the most inefficient way of navigation, which causes screen-reader users substantial information overload. In our study, 4 (25%) of 16 participants consistently chose to listen to sequential reading, and 4 (25%) more used line-by-line navigation to locate the beginning of main content. It is notable that 6 of these participants rated themselves as being very comfortable with computers.

Strategies for skipping irrelevant information. Very often the beginning of main page content coincides with the title of an article, product description, etc. However, having found the title with any of the described strategies, users still have to skip through irrelevant links, ads, and lists of related articles or products that web developers tend to place in between the title and the rest of the content. Even worse, irrelevant content is sometimes placed right in the middle of main page content, forcing users to guess if it represents the end of the content or just some annoying interruption. The majority of respondents in our 16-user survey confirmed that they periodically encountered irrelevant information in the middle of main page content (Table 9). Strategies for skipping irrelevant information usually include paragraph and non-linked-text navigation. Since ads often appear in inline frames (<iframes>), some users try to skip them using frame navigation or turn of iframes altogether, thus making frames invisible. In our experiments, 10 (62.5%) of 16 screen-reader users skipped through ads; the rest preferred to just read through the ads.

In this section, I surveyed a number of useful strategies employed by screen-reader users to locate main page content and skip irrelevant content on a web. Our observations of user behavior have shown that only power users employ advanced strategies such as keyword searching; the majority of users stick to heading navigation and sequential navigation. Although novice users should master all of these strategies, I believe it would have been better if the assistive technology provided an easy and transparent way of finding main content.

6.3 Making Web Pages More Accessible

The use of HTML headings. The simplest and most frequently used approach to improving web page accessibility is the use of HTML headings. Web accessibility guidelines (WCAG 2009) recommend that level-1 heading tags (*e.g.*, <*H1*>) be used to identify the main content. Screen-readers typically use key "*H*" to jump to the next heading and use keys "1-6" to jump to the next heading of the corresponding level. This allows screen-reader users to press a "1" key to jump to the next level-1 heading. Unfortunately, many web designers mix up heading levels or fail to use HTML headings altogether, thus making heading navigation unusable. In our survey of 57 major news web sites, we found that 35 (61.4%) of them used <H1> headings only for article titles, 4 (7%) used <H1> for article titles and other subtitles, 14 (24.5%) used headings of other levels instead of H1, and 4 (7%) of websites did not use headings at all.

Changing the layout of pages. A more exotic way of making a web page accessible for screen-reader users is the use of absolute positioning for web page content, *e.g.*, with Cascading Style Sheets (CSS 2009). This approach allows web designers to compose HTML source code in the order they want screen-reader users to hear it, *e.g.*, main content first. At the same time, this approach allows web designers to create any imaginable visual layouts. However, this approach adds extra work for the web designers, while not giving clear benefits. Finally, changing the layout of web pages has to be followed consistently, and, given the collaborative nature of web page authoring, very few web designers follow this approach. In our survey of 57 major news websites, we found only 2 (3.5%) web sites that followed this approach and had their main content read to users before the navigation links.

Skip-to-Main-Content Links. Another recommendation of accessibility guidelines is the use of hidden links (typically in the beginning of a page) allowing screen-reader users to jump to a specified location on the page. Skip-links do not change the layout of the page and are typically invisible to sighted people, but can be a great help to screen-reader users. Unfortunately, very few websites implement such links, and the ones that do often fail to maintain them consistently; as a result, screen-reader users often avoid such links (Bigham 2007), which was also confirmed by WebAIM and our surveys.

Accessibility mark-up. A more futuristic approach is the use of semantic tagging that would provide a standard way for marking up the beginning of main content in the HTML source code. Such tagging could be useful to a variety of web tools besides screen-readers, *e.g.*, search engines, mash-ups, wrappers, etc. Such mark-up will be available in HTML5[1] in the form of the <*article*> tag. But again, the effectiveness of mark up will depend on the level of conformance to it and the speed of adoption of the specifications.

[1] HTML5 Specifications: http://dev.w3.org/html5/spec/Overview.html#the-article-element.

6.4 Related Work

Intelligent Content Analysis Research. A number of projects have tried to improve the browsing experience by analyzing web pages and trying to extract semantic information. For example, (Harper 2005) used gist summaries for linked web pages to help users decide whether or not to follow links. The BrookesTalk (Zajicek 1999) system uses text abstracting techniques to facilitate "scanning" of web pages. Although summarization may be useful for users in some cases, presenting a summary of a page not only may not reduce, but even increase browsing time on that page.

Contextual analysis in summarization. It should be noted that the notion of context has been used in different areas of Computer Science research. For example, (Delort 2003) defines the context of a web page as a collection of words gathered around the links on other pages pointing to that web page. They used the context to obtain a summary of the page. Summarization using context is also explored by the InCommonSense system (Amitay 2000), where search engine results are summarized to generate text snippets. HearSay typically uses the words of the link as context to achieve higher accuracy in identifying the beginning of main content; however, the context can be extended to include the text around the link, for example for very short links.

Context in non-visual web browsing. The use of contextual information for non-visual Web access is not a well-studied problem. A technique resulting from early efforts at context analysis for non-visual Web access is described in (Harper 2004), where the context of a link was used to get a preview of the next Web page so that visually disabled individuals could choose whether or not they should follow the link. This idea is used in the AcceSS system (Bambang 2005), which employs content summarization of web pages and retains only the "important" sections. HearSay-CSurf does not yet find all relevant information, but only locates the beginning of main content. Although this may be an interesting direction to explore, suggesting to the user what is relevant in a webpage may substantially increase browsing time in case some important information is missing.

Ranking web page segments. Previous work aiming to rank web page segments includes (Song 2004), who uses the VIPS segmentation algorithm (Cai 2004) to partition a web page, extracts segment features (*e.g.*, spatial features, a number of images, sizes, links, etc.), and then uses a Support Vector Machine (SVM) to label segments, according to their importance. This approach, however, relies heavily on the segmentation algorithm, whereas the relevant information may take up only a part of a segment or may be split among multiple segments. At the same time, in the context of non-visual web browsing, users are more interested in the beginning of the main content, rather than the section that contains relevant information. HearSay-CSurf takes the former approach.

Relationship to the Dynamo approach. While template-based navigation with Dynamo (Chapter 5) helps find main content by examining the layer of differences, not all websites follow templates, and Dynamo does not work in inter-site navigation. The CSurf approach helps solve this problem.

6.5 Our Prior Work[1]

We first introduced the idea of using context for relevance detection in a poster (Mahmud 2006a), which we further expanded in a short paper (Mahmud 2007a), and finally, concluded with a full paper (Mahmud 2007b) providing details of our full sophisticated approach (CSurf-Old). In this section, I summarize our prior work and describe its deficiencies.

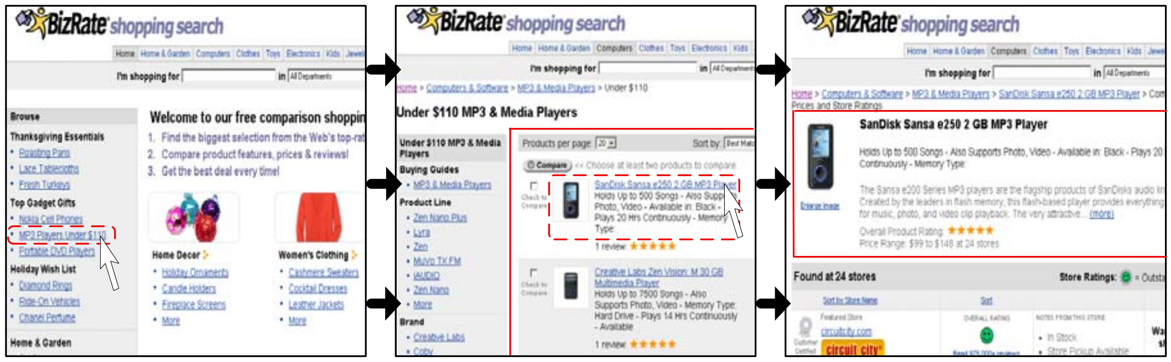

BizRate shopping search
Welcome to our free comparison shopping
MP3 Players Under $110
Under $110 MP3 & Media Players
SanDisk Sansa e250 2 GB MP3 Player
Creative Labs Zen Vision: M 30 GB Multimedia Player
Found at 24 stores

Figure 22 – Prior Work on Context-Directed Browsing

6.5.1 Context Identification

The first step of the context-directed browsing approach is to collect the context of the link followed by the user. We defined the *context* of a link as the content around the link that maintains the same *topic* as the link. To illustrate, in Figure 22(a-b) the arrows point to the followed links and the dashed red rectangles delineate the contexts that maintain the same topic as the links. In Figure 22(a), the context only includes the words of the link, while in Figure 22(a) the context also includes a brief product description.

To collect the context, we start with the words of the link and then expand the context on both sides of the link (assuming linear representation of the page content) for as long as the candidate text maintains the same topic as the already collected context. Context expansion is also limited by the boundaries of the webpage segment as identified by the HearSay segmentation algorithm. The topic boundary detection approach (Allan 2002) using the cosine similarity metric is employed to compare the topics of the already collected context and candidate context. If the candidate text cosine similarity score is above the empirically chosen threshold (0.5), then the candidate text is added to the collected context.

6.5.2 Relevant Segment Identification

The context-directed browsing approach first performs segmentation using a variant of the VIPS (Cai 2004) segmentation algorithm. The relevance of each segment is then computed and the highest ranking segment is selected. Subsequently, the HearSay browser starts reading the web page from this segment. To illustrate, the solid red

[1] Partially adapted from (Mahmud, Borodin et al. 2007) nominated for Best Paper Award at WWW'07.

rectangles in Figure 22(b-c) delineate the segments that are most relevant with respect to the context collected around the respective links.

To perform the ranking, we use a binary classifier that, given a feature vector representing a segment, returns a score between 0 to 1 representing the segment's relevance. We trained a linear Support Vector Machine (SVM) on a dataset of source-destination webpage pairs; each of the source pages had the followed link, and each of the destination pages had the most relevant segment manually labeled. As the feature vector $< f_1, f_2, \ldots, f_6 >$, we use the number of matches of words and word combinations in the context to the words and word combinations in the target segments. The different features include single words, bigrams and trigrams of whole words, and their stemmed (Porter 1997) counterparts. The SVM training phase simply computes the weights w_i in the segment relevance scoring function:

$$F: (\bar{f} \times \bar{w}) = w_1 \cdot f_1 + w_2 \cdot f_2 + \ldots + w_n \cdot f_6$$

6.5.3 Conclusion on Prior Work

Our original context-directed browsing approach with CSurf-Old (Mahmud 2007b) was nominated for the Best Paper Award in the World Wide Web conference for its impact on improving web accessibility. The approach has shown real promise and demonstrated fairly good accuracy in both context collection and relevancy scoring. However, the effectiveness of CSurf-Old was also affected by a number of limitations, which I describe next.

6.6 Limitations of the Prior Approach

Deficiency of segmentation. CSurf-Old (Mahmud 2007b) is over-reliant on the results of the segmentation algorithm, which is unpredictable. Segmentation often produces reasonable results, identifying sections the same way sighted people would do it. However, just as people may find more than one way to segment a web page, so does the segmentation algorithm. In particular, it is difficult to control the granularity of automatic segmentation. This means that, depending on the way the web page is designed, sometimes identified segments are too big and sometimes too small.

Effects of unpredictable segmentation. The unpredictable granularity of segmentation, coupled with the assumption that there is only one relevant segment, sometimes leads to unpredictable results. For example, it means that if the granularity is too low, the "most relevant" segment can also subsume irrelevant information at the segment boundaries, which means that irrelevant information preceding the beginning of main content could be read to the user first, and could make the user believe that the algorithm failed to identify the beginning of main content. If the granularity is high, relevant information can be split between multiple segments so that product descriptions and customer reviews often end up in two different segments, thus leaving out some relevant information.

Limitations of context collection and SVM. Due to unpredictable segmentation, context collection can fail because it is bound by the segment. The use of a threshold in context expansion is not a reliable indicator of whether candidate text should be included in the link text. At the same time, the irrelevant content collected as context can inadvertently affect the accuracy of the CSurf-Old. However, no experiments were conducted to compare the accuracy of relevant segment identification when using the context around the link, together with its caption, vs. only the caption of the link. One other limitation is caused by the use of a batch-mode machine learning approach, because of which the SVM would have to be periodically retrained to accommodate for new features. Finally, the use of stemming means that the approach is language specific.

Summary of limitations. Although the CSurf-Old approach identifies the most relevant segment in web pages, the algorithm cannot guarantee that the "relevant" information will be included in the segment in full. This means that, if the user tries to access the content of the segment through the HearSay layered interface (Section 4.3), the user may either miss important information and not be aware of the fact, or realize that the information is missing and reread the entire page all over again. Either way, the user may lose a substantial amount of time unless the accuracy of the algorithm is next to perfect. However, the problems described in this section suggest otherwise, essentially reducing the utility of the CSurf-Old approach to finding the beginning of main content; but, since the beginning of most relevant segment identified by CSurf-Old often does not coincide with the beginning of main content, a new approach is required to better assist users in finding relevant content.

6.7 New Approach to Context-Directed Browsing with CSurf

6.7.1 CSurf Architecture and Interface

The CSurf approach is implemented in the Context Analyzer Module of the HearSay Architecture (Section 3.3). The module keeps track of user navigation between web pages, and, as soon as it detects that a link was followed, it collects the context of the link from the origin webpage and matches the collected context to all spoken elements (the HTML elements that will be read out by HearSay) of the destination web page, after which it picks the best match as the beginning of main content. More details on the CSurf-New algorithm are presented in Section 6.7.

When the user follows a link and opens a new web page in the same browser tab, new tab, or a new window, HearSay-CSurf tries to find and start reading from the beginning of main content, which is typically a title of a newspaper article, product description, or some other content. In case HearSay-CSurf makes a mistake, the user is only one shortcut away from the top of the page ("*Ctrl+Home*"). In Section 6.9, I report on the user study that demonstrates the effectiveness of HearSay-CSurf and validates the effectiveness of the CSurf approach.

There are a number of cases where the CSurf feature may not work. In case of page refresh, or opening a web page from bookmarks or URL, the CSurf functionality does not activate because there is no user context to infer where the beginning of main content is. Since short link-labels do not provide sufficient context, the CSurf-New algorithm does not activate for links shorter than 2 words either. CSurf ignores same-page links as they already take the user to relevant sections of the page. Also, CSurf does not work when web pages redirect while loading as it cannot yet distinguish such redirects from page refreshes. Finally, users can turn off the CSurf feature in the HearSay settings.

HearSay-CSurf also inserts a hidden "HearSay skip to main content" link at the top of the page. The HearSay skip link works similarly to hidden links that web developers are supposed to insert in their web pages, according to the web accessibility guidelines (WCAG 2009). The element representing the beginning of main content is also marked as a level-1 heading (<*H1*>). In this way, even if the CSurf feature is turned off, or the user is browsing elsewhere on the web page, s/he can easily find the beginning of main content either by using heading navigation shortcuts or going to the top of the page and following the HearSay skip link. This versatility ensures that HearSay serves users, regardless of what their favorite browsing strategies may be.

6.7.2 CSurf-New

When the user follows a link, *e.g.*, "Lean manufacturing helps in recession" indicated with an arrow in Figure 23(a), the CSurf-New algorithm collects the words of the link and converts them into a context multiset and then into a context vector, *e.g.*, <lean: 1, manufacturing: 1, helps: 1, in: 1, recession: 1> – 1's indicating the number of occurrences of a given word in the multiset. The algorithm then waits for the target page to load and converts every spoken element of that page into a vector. CSurf-New then uses the cosine similarity metric to match the context vector to the vectors representing the spoken elements of the target page.

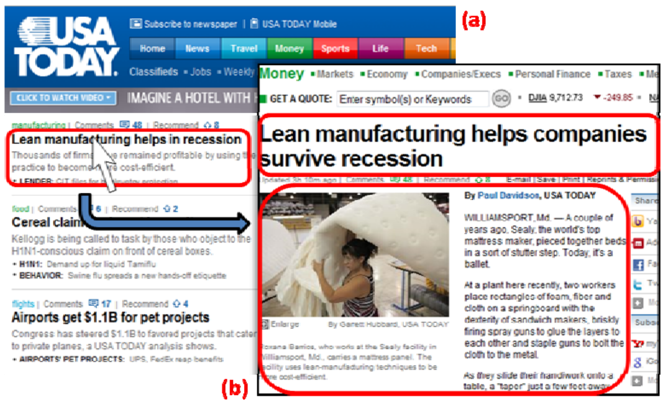

(a)
Lean manufacturing helps in recession
Lean manufacturing helps companies survive recession
(b)

Figure 23 – Finding the Article on USA Today News Site

The CSurf-New algorithm uses the cosine similarity metric to find the beginning of main content. Cosine similarity is frequently used in information retrieval methods dealing with vector comparisons of textual documents (Salton 1975). The intuition behind the cosine similarity metric is that documents are placed into a multidimensional space where every word/term represents a new dimension. If similar documents have more words in common, the corresponding vectors will have more dimensions in common. Then, the more similar two documents are, the smaller the angle between the corresponding vectors is. The Cosine similarity measure varies between 0 and 1, with 1 being a 100% match:

$$\cos(a, b) = \frac{a \cdot b}{\|a\|\|b\|}$$

By using an approximate matching approach, rather than an exact match, we are able to match the context to the title of the article despite some differences, *e.g.*, in Figure 23(a).

Context: a = <*lean*: 1, *manufacturing*: 1, *helps*: 1, in: 1, *recession*: 1>

Heading: b = <*lean*: 1, *manufacturing*: 1, *helps*: 1, companies: 1, survive: 1, *recession*: 1>

$$\cos(a, b) = \frac{1 + 1 + 1 + 1}{\sqrt{5} * \sqrt{6}} = 0.73$$

Thus, the comparison of the context vector and the vector of the article heading yields the similarity of 0.73. In Section 6.8 I compare the use of Cosine similarity with another popular similarity metric (Jaccard similarity) on the collected dataset.

Instead of keeping a collection of stop-words to reduce the effect of functional words such as prepositions and articles, as was done in CSurf-Old (Section 6.5), CSurf-New uses tf-idf to normalize the value of each vector component by the number of occurrences of such components in other vectors:

$$tf - idf = \frac{term\ frequency}{\log(document\ frequency)}$$

For example, if the article (determiner) “the” occurs 3 times in a given document and any number of times in 8 documents in a given collection of documents, then the corresponding vector component <…, the: 3, …> will be adjusted by log(8), resulting in <…, the: 1, …>. The tf-idf adjustment helps make the approach language-independent for most natural languages existing today. Continuing with the example, the spoken elements vectors from Figure 23(b) may be adjusted as:

Context: a = <*lean*: 0.5, *manufacturing*: 0.6, *helps*: 0.9, in: 0.2, *recession*: 0.7>
Heading: b = <*lean*: 0.5, *manufacturing*: 0.6, *helps*: 0.9, companies: 0.6, survive: 0.9, *recession*: 0.7>

$$\cos(a, b) = \frac{0.5 + 0.36 + 0.81 + 0.49}{\sqrt{1.95} * \sqrt{3.08}} = 0.58$$

Based on an empirical observation, context words may have high similarity to multiple spoken elements on the target web page. However, since the beginning of main content is typically marked with a larger font, the similarity between context and spoken-element vectors is adjusted (multiplied) by the font size of the spoken element.

Context: a = <*lean*: 0.5, *manufacturing*: 0.6, *helps*: 0.9, in: 0.2, *recession*: 0.7>
Heading: 20pt: b = <*lean*: 0.5, *manufacturing*: 0.6, *helps*: 0.9, companies: 0.6, survive: 0.9, *recession*: 0.7>
Text 10pt: c = <…, in: 0.2, …, facility: .9, uses: 0.4, *lean*: 0.5, *manufacturing*: 0.6, …>
Link 10pt: d = <*lean*: 0.5, *manufacturing*: 0.6, *helps*: 0.9, in: 0.2, *recession*: 0.7>

cos(a, b) = 0.58 * 20 = **10.8**; cos(a, c) = 0.36 * 10 = 3.6; cos(a, d) = 1 * 10 = 10

Thus, with the font-adjustment, the heading with a larger font gets a higher score than a smaller-font link, which is the exact match to the context vector. In Section 6.8, I will show on a dataset of web pages the improvement that the font-size adjustment offers.

6.8 Automated Experiments with Contextual Browsing

6.8.1 Experimental Setup

To verify the accuracy of the CSurf algorithm, we used the same data collection tool, which is described in Section 5.6, to collect over 300 pairs of webpage DOM trees serialized in the HearSay-compatible format. In the first page of every pair, a link leading to an article or a product description was highlighted. In addition to the link, we also highlighted the text (context) that was semantically related to the article. In the second page of each pair, the beginning of main content was highlighted.

We then constructed an experimental framework that allowed us to automatically push the DOM trees through the HearSay pipeline pair by pair and automatically compute the accuracy of the CSurf algorithm. We ran the test on two different categories of web pages: news and shopping, and with several conditions: 1) Jaccard similarity vs. Cosine similarity; 2) using only link text vs. using link context; and 3) *with* vs. *without* font size weighting.

Jaccard similarity is one of the metrics used in the experiments with the CSurf matching algorithm. Jaccard coefficient measures similarity between two multisets of words *A* and *B*, penalizing a small number of matching entries. The similarity measure varies from 0 to 1, with 1 being a 100% match:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Cosine similarity is another metric used in the CSurf experiments. This metric transforms a set of the sets of strings *A* and *B* into vectors and then computes a cosine of the angle between the two vectors. The more similar the two strings, the smaller the angle between the corresponding vectors. The Cosine similarity measure varies between 0 and 1, with 1 being a 100% match:

$$\cos(a,b) = \frac{a \cdot b}{\|a\|\|b\|}$$

Since verifying whether CSurf found the right starting position was a Boolean decision, the overall accuracy was computed as a percent of correctly identified starting positions:

$$Accuracy = \frac{\#\,Correct}{\#\,Trials}$$

6.8.2 Discussion of Results

The experiments showed that using Cosine similarity yielded the same or better results than using Jaccard similarity. Adding the text around the followed link to the context considerably reduced the accuracy of the CSurf-New algorithm. The use of font-size adjustment improved the accuracy of up to 10%. The summary of the results is presented in Table 7. Figure 24 and Figure 25 visualize the differences in the performance of the algorithm.

		Link Text Only		*Link and Context*	
		Jac	*Cos*	*Jac*	*Cos*
News	***w/o Font Size***	0.842	0.842	0.272	0.298
	With Font Size	**0.921**	**0.921**	0.395	**0.623**
Shop	***w/o Font Size***	0.830	0.830	0.642	0.698
	With Font Size	0.868	**0.906**	0.736	**0.792**

Table 7 – Accuracy of the CSurf Algorithm

Cosine similarity generally produced better results than Jaccard similarity. The only time when the accuracy was similar was on the news dataset in the experiments that used only link text with font-size adjustment. In part, the effect may be due to the tf-idf adjustment performed with cosine similarity.

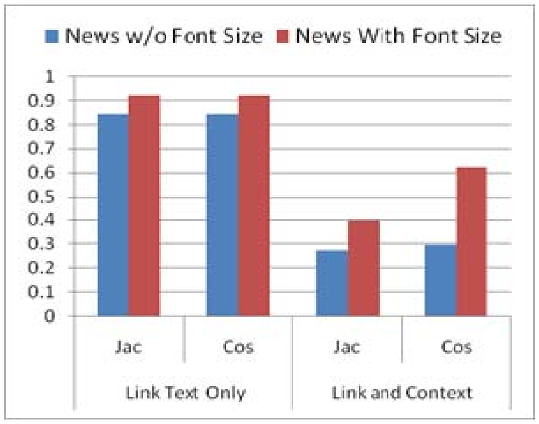


Figure 24 – Comparative Accuracy (News)

The use of the font-size adjustment resulted in a 5%-10% improvement of the accuracy when only link text was used, and 10%-40% improvement when the text around the followed link was used. We also conducted an experiment, in which we chose the largest font element as the beginning of main content. This experiment yielded the accuracy of 0.623 on the news dataset and 0.472 on the shopping dataset, which was 30%-40% lower than the accuracy obtained when using either similarity metric with font-size adjustment.

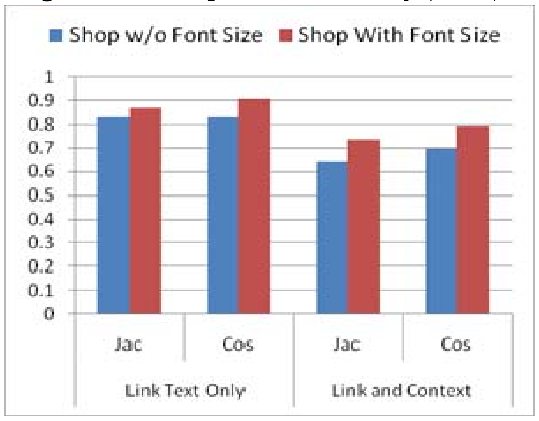


Figure 25 – Comparative Accuracy (Shopping)

Using the text around the link generally had a negative effect on the accuracy of the CSurf-New algorithm. In the best-case scenario, it caused 10%-20% degradation in accuracy. However, the use of wider context shows promise on short links, *e.g.*, "*more>>.*"

On close examination of cases where CSurf-New algorithm failed, we found that in news sites, the text of the link sometimes had very little similarity to the heading of the article, while shopping websites tended to overuse large font.

6.9 HearSay-CSurf User Study

6.9.1 General Information

We conducted a two-hour user study with 16 participants to evaluate my context-directed browsing approach. The study consisted of two experiments: one tested the HearSay-CSurf's ability to find the beginning of main content, and the other simulated the system's layered interface for viewing relevant content. The first part of the study investigated whether automatically starting to read from the main content improved the efficiency of browsing on three categories of websites: university, news, and shopping. The second part of the study tested subjects' ability to retain information while being distracted by irrelevant links in the middle of a news article.

The following two versions of HearSay were used in our evaluation: (i) HearSay-Basic (HB), providing the basic screen-reader navigation shortcuts, and (ii) HearSay-CSurf (HC), having all functionalities of HB, but automatically computing and reading from the beginning of main content after users followed a link. Participants were not told that they used the same system in all experiments. Both systems used Microsoft Anna's synthetic voice because it had the least latency compared to Cepstral and RealSpeak voices. To control the random variables, we fixed the rate of speech to a level comfortable to both expert and novice users. Most participants found Anna easy to understand, although several people complained that Anna's voice clipped words and mispronounced some content.

As in the Dynamo experiments (described in Section 5.7), I considered evaluating HC against JAWS, but again concluded that using the same browser in all tasks would have helped avoid biasing our evaluation with uncontrolled random variables. At the same time, the HearSay-CSurf functionality would have been difficult to integrate into the proprietary JAWS screen reader. Compared to the HearSay version used in the Dynamo user studies, HearSay-CSurf had faster response time (even though still slower than that of JAWS, possibly due to the low-latency Eloquence TTS used by JAWS). In HearSay we also carefully replicated JAWS shortcuts and functionalities. To further increase the similarity between JAWS and HearSay, we turned off the earcons and used exclusively verbal prompts.

Although users had two training sessions in the course of the user study, they were not told what technology they were using or trained in the use of shortcuts, to avoid biasing them toward a particular browser, or lead them to using any particular functionalities. The subjects were asked to commence browsing, as they normally would do with their regular screen-reader. All of the subjects reported using JAWS as their primary screen-reader. Despite the fact that HearSay is a little different from JAWS, during the debriefing phase, several subjects said they assumed they were working with JAWS and used the same functionalities as they would have used in JAWS.

6.9.2 Study Participants

We recruited 16 subjects to participate in the user study. The majority of the participants were recruited by the Disability Resource Center of Arizona State University. The variety of the programs and accessibility of the university attract students with vision impairments from all over the United States.

The participants varied in their age, gender, education, and occupation. The sixteen participants ranged in age from 18 to 50 with an average 32 years and standard deviation of 11 years. Approximately, half of the participants were male, and half were female. Among the 16 participants, 6 (37.5%) had high-school diplomas, 6 (37.5%) had Bachelor's degrees, 3 (18.8%) had Master's, and 1 (6.3%) had a Ph.D. (Figure 26). All, except two participants, reported English as their first language, while the remaining two participants were fluent in their use of English. Approximately half of the participants were working professionals, and half were students. Most of the participants had a liberal arts education in fields such as Journalism, Education, English, Psychology, and Sociology.

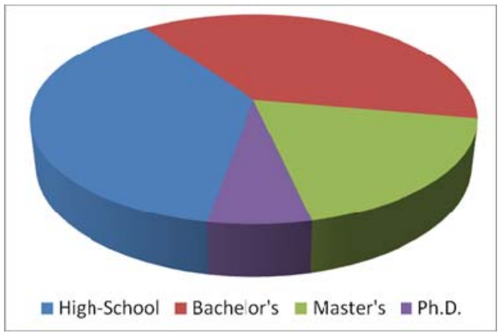


Figure 26 – Participants' Education

The participants had a wide range of causes of vision impairment, including 2 participants had ROP (Retinopathy of Prematurity), 2 participants lost sight in accidents, 2 participants who lost sight due to tumors, and 3 participants had glaucoma. The rest of the participants had more rare conditions for this age group, including diabetes, Retinitis Pigmentosa, scarred optic nerve, optical nerve atrophy, coloboma-bilateral, Peter's anomaly, and cardiac arrest. Among the subjects, 3 reported no light perception, 4 – faint light perception, 2 – strong light perception, and 5 had vision at 20x200, or worse. Among the participants, 6 were blind from birth, 5 lost vision when they were under 10 years old, 4 lost vision in their teens, and 2 – in their twenties.

The participants varied in their screen reader proficiency from relatively inexperienced to expert. Among the subjects, 2 (12.5%) were self-reported experts, 10 (62.5%) were very comfortable with computers, 2 (12.5%) were comfortable, and 2 (12.5%) were mildly comfortable (Figure 27). Also, 4 (25%) participants used computers before losing vision. Their general comments were that they had to relearn how to navigate with a mouse and be more creative and more patient. One of the participants was using computers for data entry and did not use the Internet before losing vision.

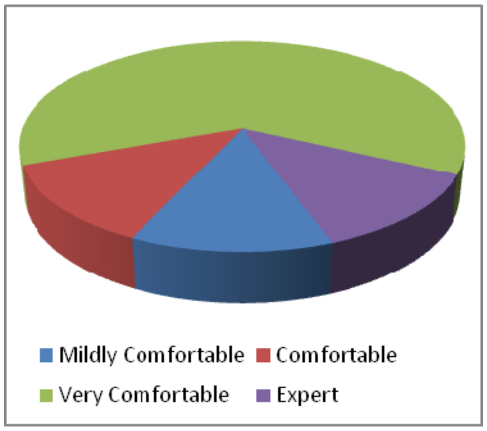


Figure 27 – Computer Expertise

The participants also reported their computer use: 8 (50%) participants more than 20 hours per week, 4 (25%) participants 10-20 hours per week, and 4 (25%) participants 1-5 hours per week (Figure 28). Among the participants, 6 (37.5%) reported using the Internet for work, 12 (75%) for school, 9 (56%) for socialization, 10 (62.5%) for shopping, 12 (75%) for acquiring information, and 13 for entertainment. The participants reported the use of a number of applications and websites, among which the prevailing were Microsoft Office, Acrobat Reader, Media Players, Skype, Linked-in, Twitter, and Facebook.

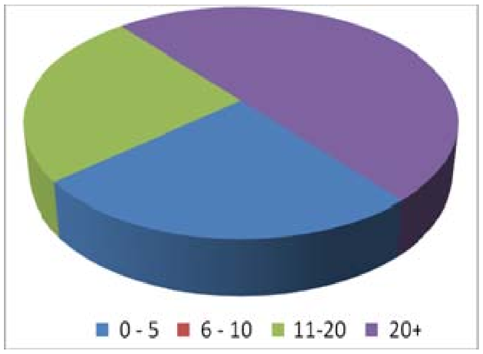


Figure 28 – Computer Use Hour per Week

All of the participants were predominantly JAWS users, using JAWS as their primary assistive technology. The JAWS versions used by the participants varied from 7 to 11 (the current version); some of the participants used different versions at home and at work. A number of participants noted that web browsing with JAWS had plenty of room for improvement, but they were using it because it was the best available screen-reader. Among the participants, 3 were also using ZoomText for casual browsing, but used JAWS to relax their eyes, and 3 participants reported using one or several secondary screen-readers: VoiceOver, System Access, and WindowEyes. All of the participants reported using Internet Explorer as their primary web browser. FireFox was used as a secondary browser by 6 participants, and Safari – by 2. During the debriefing phase, many users were surprised to find out that they were using the Firefox browser during the experiment. The participants noted that JAWS “works better with Internet Explorer,” and that Internet Explorer comes pre-installed on the Windows operating system, which predefined their choice of the primary web browser.

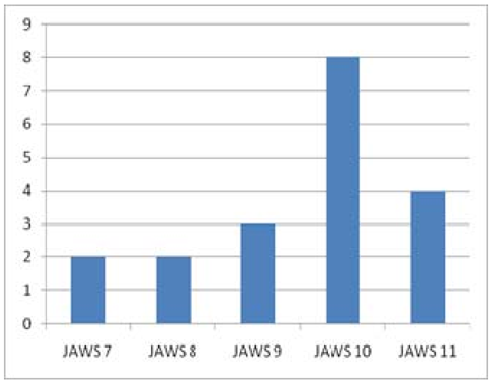


Figure 29 – Primary Screen Reader

6.10 Experimental Setup for Finding Beginning of Main Content

6.10.1 Categories of Tasks

The experiment for finding main content in web pages involved 6 web sites in 3 different categories: news, university, and shopping. The two websites in each category were carefully selected to have matching accessibility and similar layout. For each website, a pair of web pages was cached to avoid unexpected page updates. The first page was the website's front page, from which a link led to a second page, which included either an article or a product description. The three categories represented three accessibility conditions for heading navigation: fully accessible (university), partially accessible (shopping), and not accessible (news). The six websites did not have other accessibility problems, other than the abundance of image links without alternative text.

The task in the university category represented browsing a fully accessible website. The participants had to locate a link leading to a story about the university and find an answer to a specified question, with the answer being located in the first paragraph of the story. The two websites chosen for the task were the Stony Brook University and Alfred State College websites. Both of the websites were fully accessible for heading navigation and had a single heading level 1 (<H1>) marking up the story title. Both websites had in the range of 20-30 items preceding the article.

This task involved navigating shopping websites. Participants had to select a specified product link leading to a product description. The two websites selected for the task were FYE (shopping for music) and Powell's Books (shopping for audio books). On the product description page, participants had to find answers to the questions: “What band plays this song?” and “Who wrote this book?” respectively. Both websites used a number of headings of different levels, other than level 1. Both sites had 5 headings with over 100 sub-items preceding the heading of a product description. This task represented browsing a partially accessible website with confusing heading levels that made it is easy to navigate sections of the page, but made it harder to find the main section.

The task in this category involved navigating the headline news, finding a predefined story, following the link to a full new article, and answering a question, the answer to which was in the first paragraph of the article. The two web sites chosen for this task were USAToday and ESPN. Neither website used headings on the article web page, making their web pages inaccessible for heading navigation. Both websites had in the range of 30-40 items preceding the target article.

6.10.2 Procedures

Each participant completed 1 task from each of the 3 task categories (Section 6.10.1), with both HB and HC for a total of 3 x 2 = 6 trials. The 3 tasks performed with a particular version of HearSay (either HB or HC) were conducted sequentially. Both the system order and ordering of the task groups were counter-balanced across participants using a 4-cell design.

The experiment used a 2 x 3 within-subjects factorial design with factors and levels as follows:

System	{HearSay-Basic, HearSay-CSurf}
Task	{University, Shopping, News}

The dependent variable of interest was task completion time, measured from the time the target page loaded and became available for browsing. An upper limit of 5 minutes was set for locating a link to follow and 5 minutes for finding an answer to the question. The limit was reached only while looking for the link in 6 out of the 96 total trials. The time was analyzed using repeated-measures ANOVA.

The following hypothesis was tested in this experiment: Automatically starting from main content reduces the browsing time in looking for information.

6.10.2.1 Practice Time

Before beginning the first session of 3 tasks, participants practiced for up to 10 minutes to familiarize themselves with the interface and to complete a sample task. The participants practiced for an additional 5 minutes before the second session of 3 tasks. The practice time was longer in the beginning of the experiment to give users more time to familiarize themselves with the HearSay interface and get used to the synthetic voice, During the practice time, the participants tried their favorite navigation shortcuts and were allowed to ask about the browser functionalities that may have been different from what they were used to. All of the participants assumed that the functionalities and shortcuts are similar to those of JAWS.

6.11 Results for Finding Beginning of Main Content

6.11.1 Speed of Task Completion

Overall, HearSay-CSurf decreased the time required for participants to complete the two tasks, as summarized in Figure 30. The completion time with HB varied widely, depending on the screen-reader proficiency of participants. The completion time with HC varied somewhat due to the strategy of users upon arrival to the target page. For example, a number of participants started browsing every webpage by pausing the screen reader first.

On the University sites, participants spent a mean of 40.06 seconds (std. dev.=44.51), using HB, and a mean of 7.56 seconds (std.dev.=3.72), using HC. This represented a significant effect of System on task completion time ($F_{1,15}$=5.10, p<.005), indicating that HC improved this task. Using the HB, most participants were able to find the answer to a question within one minute, and only P3 took over 3 minutes.

On the Shopping sites, participants spent a mean of 82.94 seconds (std. dev.=78.95), using HB, and a mean of 4.94 seconds (std.dev.=2.91), using HC. This represented a significant effect of System on task completion time ($F_{1,15}$=16.28, p<.005), indicating that HC improved this task. Due to the abundance of links on shopping web sites, P2, P3, and P5 took over 3 minutes to complete the task with HearSay-Basic.

On the News sites, participants spent a mean of 50.06 seconds (std. dev.=23.50), using HB, and a mean of 16.63 seconds (std.dev.=7.61), using HC. This represented a significant effect of System on task completion time ($F_{1,15}$=37.33, p<.005), indicating that HC improved this task. Only P2, P4, P7, and P11 took over a minute to complete this task with HB. It took longer to complete this task with HC compared to the other two tasks because the answer to the question was located deeper in the news article.

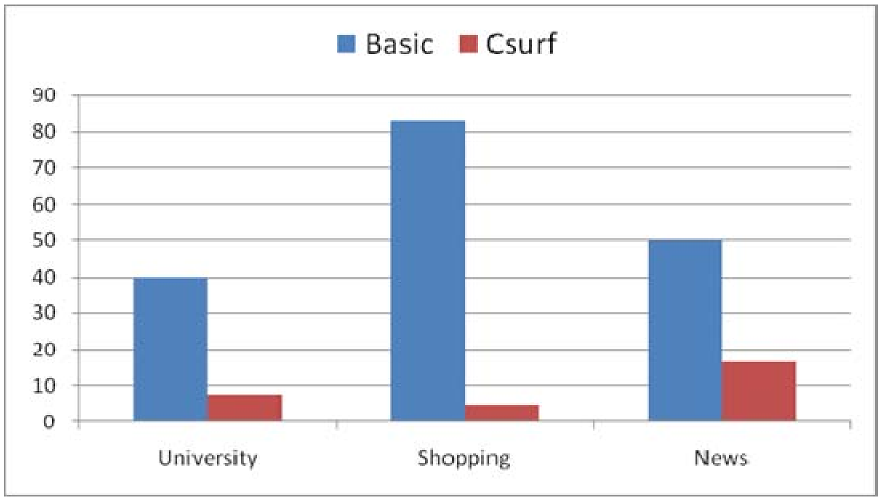


Figure 30 – Average time required to complete each task (seconds)

6.11.2 Subjective Feedback

Following each task, participants rated the difficulty of completing the task on a 5-point Likert scale (1=Easy to 5=Hard – See Table 8). Participants consistently rated the tasks completed with HC to be easier than those completed with HB on average. The participants considered most of the tasks rather easy. According to the analysis with the Wilcoxon Signed-Rank significance test (Wilcoxon 1945), the differences in the difficulty of the University tasks were not significant, and the differences on the Shopping and News sites were significant ($p<0.01$). This leads me to believe that the subjects did not consider longer tasks to be difficult, but rather rated the difficulty based on the presence of accessibility problems.

Task Category	HC	HB
University	2.00 (1.33)	2.21 (1.03)
Shopping	1.58 (0.90)	2.26 (1.24)
News	1.66 (1.16)	2.16 (0.96)
Average	1.74 (1.13)	2.21 (1.06)

Table 8 – Average 5-Point [1 = Easy to 5 = Difficult] Likert scale values (St. Dev.)

At the end of the study, we asked participants to what extent they agreed with several statements on a 5-point Likert scale (from 1=Strongly Disagree to 5=Strongly Agree – See Table 9), based on the prior browsing experience. Overall, participants somewhat agreed with the statement that they “often experience difficulty when trying to find main content in a web page.” However, almost all of the participants agreed that “automatically starting from main content makes browsing more efficient.” One of the participants felt disoriented by being taken to the main content. The participants generally agreed with the statement that “web pages often have heading levels that make it easy to navigate page sections.” The participants somewhat agreed that they “use skip-to-main-content links when they are available.” However, with a high variance of answers, there was little agreement between participants on this topic.

General Statements	Response
I often experience difficulty when trying to find main content in a web page.	3.44 (0.89)
Automatically starting from main content makes browsing more efficient.	4.38 (0.96)
Web pages often have heading levels that make it easy to navigate page sections.	4.00 (0.89)
I use skip-to-main-content links when they are available.	3.44 (1.59)

Table 9 – Average 5-Point [1 = Strong Disagree to 5 = Strong Agree] Likert scale values (St. Dev.)

6.11.3 Discussion

When asked about the assistive technologies used during the study, several participants said they automatically assumed that they were using the JAWS screen reader due to the similarity of the shortcuts and features. Because the participants did not know what assistive technology they were actually using, several of them got the impression that HC was faster and more responsive than HB, or that it hid some irrelevant content from them. Several participants expressed their frustration with HB and said it was more difficult to navigate web pages with HB, even though it was more familiar to them. Almost all of the participants liked and preferred HC for the tasks they used it on.

Several participants wished they could turn the HC feature on or off because they felt that the utility of starting from main content sometimes depended on the task at hand. One participant noted that he liked the fact that the system started reading from the main content after following a link, but would not want the same functionality if the page was loaded from a URL or bookmarks. HearSay provides the facility to turn on/off all of its content analysis features and starts reading from the top of the page if the page was not loaded after following a link from another page.

One of the participants in the study wanted to have a page structural summary, containing the number of headings and links on the page, read out before going to main content, to have a better orientation on the page. HearSay currently only provides a shortcut that reads a page summary. My current view on the page summary is that it creates information overload in cases when the users come to a familiar site, or know exactly what they need on the page. One of the participants noted that a page summary usually helps her confirm that she is on the right web site.

6.12 Experimental Setup for Information Retention Study

To expand on our understanding of the benefit of using layers, specifically the layer of relevant content, we conducted an experiment that simulated the system's ability to single out relevant information from irrelevant content that includes ads, links, etc. In the experiment, we measured the ability of participants to retain information if their reading was unexpectedly interrupted by ads.

6.12.1 Categories of Tasks

For the purpose of this experiment, I selected four articles from the FreePress local news website of Detroit, Michigan. A local news site ("far" from the experiment site) was chosen to minimize the likelihood of participant familiarity with the news items, and also because FreePress placed ads after the first paragraph of its news articles. I selected a total of 4 articles in two categories: two financial articles about budgetary problems of local counties (with more numbers) and two articles about events such as a burnt-down pub or bridge safety inspection (with more facts). The news items were chosen based on the abundance of possible questions one could ask about the articles, including dates, numbers, and other facts.

For the purpose of this experiment, I removed all content from the article web pages, except the body of the article. The articles were trimmed to 4 paragraphs of approximately the same length. I then chose two representative ads with links and text content of about the same length and created 3 versions of each article, with and without ads: ad0 (no ad), ad1, and ad2.

A set of 7 questions was assembled for each of the four news articles. The sets of questions across each of the two categories of news were similarly structured. The first question was a very general question about the news article, *e.g.*, "What happened to the pub?" The following two questions were about the 1st paragraph of the article and included questions either about some numerical value or a fact, *e.g.*, "On which occasions was the pub popular?" or "How many inmates may be released early?" The next two similar questions were about the 2nd paragraph, and the last two questions were about the 3rd and 4th paragraphs, respectively.

6.12.2 Procedures

Each of the 16 participants completed 1 task from each of the 2 task categories (Section 6.12.1), with ads and without ads, for a total of 2 x 2 = 4 trials. The order of tasks, categories, and the articles used with and w/o ads were permuted using a 4-cell design to counterbalance the differences among the articles and their content.

The experiment used a 2 x 2 within-subjects factorial design with factors and levels as follows:

System	{With Ad, Without Ad}
Task Category	{Financial Problems, Local Events}

The dependent variable of interest was the task completion time. The time was analyzed using repeated-measures ANOVA.

Participants were asked to read through the article *only* once and use navigation keys, if they needed to. Participants were asked to be ready to answer questions about the article.

The following hypothesis was tested in this experiment: Irrelevant content in the middle of the article inhibits the retention of information that precedes or follows the irrelevant content.

The participants were asked to read through the first two articles after the first set of three tasks described in 6.10.1, and the other two articles – at the end of the study. The order of the system and task category conditions were counterbalanced. Participants were not given a practice task for this experiment. The ads used in two out of four news articles were approximately the same size and structure, but had different content.

6.13 Results for Information Retention Study

The majority of the participants did not use shortcuts while reading the articles without ads; however, several participants preferred to read through the articles by arrowing down rather than listening sequentially. Only 6 (37.5%) out of the 16 participants skipped through the ads; the remaining 10 (62.5%) chose to listen through ads continuously. Several of them reported thinking of skipping, but the ads were over before they made up their minds.

After listening to every article, the participants were asked 7 listening-and-comprehension questions. In financial news articles, the participants answered correctly on average to 4 (St.Dev. = 0.87) questions in articles without ads, and 3.88 (St.Dev. = 1.45) questions in articles with ads. In news about local events, the participants answered correctly on average to 4.11 (St.Dev. = 1.65) questions in articles without ads, and 3.82 (St.Dev. = 1.51) questions in articles with ads. The difference in the participants' question-answering performance on the articles with and without news was not found to be statistically significant for either article category.

In Figure 31, I summarize the average question-answering performance on the 4 articles used in the user study. The participants answered correctly on average half a question more on Article 1 without ads than on Article 1 with ads. The presence of ads in Article 2 even caused the participants to answer on average 0.25 more questions correctly than on the article without ads. For Article 3, the participants gave approximately the same number of correct answers. And finally, for Article 4, the participants answered on average 1 question more correctly in the article without ads than in the one with ads.

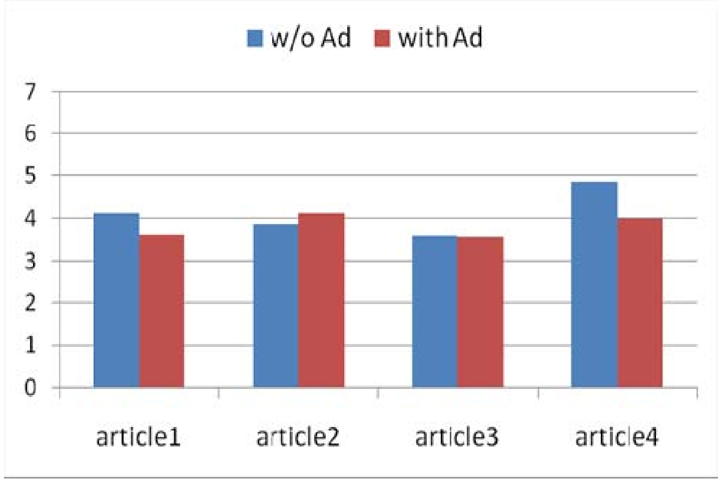


Figure 31 – Average number of correct answers per article

P 16 answered all 7 questions correctly on Article 2 with ads, and P10 answered 7 questions correctly on article 4 without ads. Only in 6 out of 64 trials, the participants answered 6 questions correctly. In 2 trials, the participants were able to answer only 1 question correctly. The experiment showed that the ability to retain information varied widely between the participants, but the participants on average demonstrated a well-developed ability to ignore irrelevant information and not let it distract them.

6.13.1 Subjective Feedback

Following each of the two tasks, the participants rated the difficulty of completing each task on a 5-point Likert scale (1=Easy to 5=Hard – See Table 10). Although some participants found all 4 tasks to be relatively easy, the participants rated, on average, the tasks with ads to be somewhat harder than those without ads. Using the Wilcoxon Signed-Rank significance test (Wilcoxon 1945), I found that differences in the difficulty of the tasks across all tasks to be significant ($p<0.005$). It is notable how the perception of the difficulty varied from the actual question-answering performance of the participants.

Task Category	With Ad	W/O Ad
Article 1	3.12 (1.46)	2.56 (0.53)
Article 2	2.78 (0.97)	1.88 (0.64)
Article 3	3.00 (0.82)	2.10 (0.88)
Article 4	2.20 (1.23)	1.57 (0.53)
Average	2.75 (1.56)	2.03 (0.76)

Table 10 – Average 5-Point [1 = Easy to 5 = Difficult] Likert scale values (St. Dev.)

After completing the tasks, we asked participants to what extent they agreed with the statement in Table 11 on a 5-point Likert scale from (1=Strongly Disagree) to (5=Strongly Agree). Users generally agreed that in their everyday life they "often find irrelevant information in the middle of an article." During and after the experiment, a number of participants expressed annoyance with the ads and complained that the ads were distracting, wiping the content preceding the ads out of their memory.

General Statements	Response
I often find irrelevant information in the middle of an article.	3.75 (1.00)

Table 11 – Average 5-Point [1 = Strong Disagree to 5 = Strong Agree] Likert scale values (St. Dev.)

6.14 Conclusions and Future of the CSurf Approach

I proposed, implemented, and evaluated the CSurf approach for helping screen-reader users locate the beginning of main content in web pages. Automated experiments were conducted to evaluate the accuracy of the CSurf-New algorithm. A two-part user study was conducted with users from the target population to gauge their reaction to and evaluate the efficiency of browsing with CSurf. The first part of the study demonstrated the effectiveness of starting to read from the beginning of main content. The second part of the study demonstrated the importance of filtering out irrelevant content such as ads in reading and comprehension tasks.

The work presented in this chapter demonstrates the benefits of using the context of a web link of the source page to find the beginning of main content on the target page. The approach may possibly be extended to identifying all relevant information in web pages. The feedback from the user study warns, however, that missing even a part of relevant content can increase browsing time for screen-reader users as they may have to start reading from the top of the page. A combination of methods and heuristics may have to be used to make the approach apply to more situations because no one single method will be able to handle the variety of web content found online. For example, by combining the results of the Dynamo and CSurf approaches, some repeated content may be flagged as irrelevant. By learning to interpret and use the context of user actions, the voice browser of the future could better assist users in a variety of browsing tasks. The feedback collected in the course of user studies has shown that users want to have more control of where the browser is reading. Also, the Social Accessibility project (Takagi 2008) could use the results of the CSurf algorithm and give users the ability to specify from what place they want the browser to start reading.

The CSurf approach can help partially-sighted users find the beginning of main content because using current magnification tools, users have to manually scroll down and look for the content. The proposed method can also be applied to browsing with mobiles; for example, mobile browsers could automatically scroll down (Borodin 2007a) or zoom in on relevant information. Detecting and visualizing relevant information can even improve regular browsing interfaces. Finally, web spiders, crawlers, and other content extraction and personalization software could use the CSurf approach to extract main content from web pages.

7. Conclusions and Future Work

In this dissertation I have explored algorithms and non-visual interfaces for bridging the Web Accessibility Divide – the divide that still exists between the ways blind and sighted users browse the Web. In this final chapter, I summarize the contributions of this dissertation in Section 7.1, describe broader impact of my research in Section 7.2, discuss possible future directions of my research in Section 7.3, and finally, I conclude with a broader message of this work in Section 7.4.

7.1 Contributions

The contributions of this dissertation include a usable prototype of the HearSay browser, a layered interface for non-visual web browsing, a unifying interface for accessing web page updates, and a context-directed browsing interface for finding main content in non-visual web browsing scenarios.

Architecture of HearSay. The result of this dissertation research is a scalable architecture for the HearSay voice browser that can potentially work with a number of web browsers and input/output devices. HearSay was developed both as an innovative assistive technology and as a platform for human-subjects and automated experiments testing novel algorithms and approaches to web browsing. The pipelined architecture of HearSay starts with an HTML DOM received from the browser, performs content analysis of web content, and finishes with the generation of interactive VoiceXML dialogs, which are then interpreted by a custom VoiceXML interpreter – vxmlSurfer. The HearSay architecture (Chapter 3) enables user annotations, webpage segmentation, tracking of content changes, context analysis, form labeling, language detection, and automation. To make use of these functionalities, HearSay implements a number of novel non-visual interfaces described in Chapters 4-6.

Basic HearSay interface. In close collaboration with users of assistive technology, I have designed a usable non-visual interface that replicates the basic screen-reading functionalities expected of any assistive web browser. Through studying the browsing behaviors and strategies employed by screen-reader users, I have introduced a number of interface improvements such as speech commands, language switching, and earcons, taking the HearSay browser to new heights of accessibility in non-visual web browsing. The TeleWeb interface enables remote telephony access to the HearSay voice browser. The Macro interface of HearSay enables easy automation of repetitive browsing tasks. The innovative layered interface allows users to filter different types of content. The layered interface extends the list-views of screen readers and enables easy access to changed information.

Unification of content updates with Dynamo. In Chapter 5, I overviewed the accessibility problems caused by dynamic content and the strategies employed by screen-reader users to cope with such content. To address these accessibility problems, I presented Dynamo – a novel approach to making dynamic and static content updates accessible. The Dynamo approach is agnostic of the technical methods used for implementing content updates in web pages. It uniformly considers any change of web content as “dynamic” and dynamically updates HearSay's dialogs with the updated content. To compare web content, I designed the LCS-based diff algorithm driving the Dynamo approach. Automated experiments were conducted for the purpose of confirming the accuracy of the Dynamo-Diff algorithm. Finally, two separate user studies with screen-reader users were conducted to evaluate the effectiveness of the Dynamo interface and get end-user feedback. The user studies have demonstrated that the Dynamo interface improves browsing time efficiency in tasks involving dynamic messages, automatic page refreshes, and template-based websites.

Context-directed browsing with CSurf. In Chapter 6, I overviewed typical strategies employed by screen-reader users for finding main page content. I presented CSurf – an innovative approach to utilizing the context of user actions to identify the beginning of main content in web pages. CSurf uses the context of the link followed by the user to match and locate the beginning of main content on the target web page. By identifying the beginning of main content in transitions from one web page to another, this approach can substantially improve the efficiency of browsing with non-visual interfaces. Automated experiments were conducted to confirm the accuracy of the CSurf algorithms. Finally, a user study was conducted to measure the gains in browsing efficiency, as well as get the feedback from the end-users. As a part of the study, I also found that irrelevant content, such as ads and other links introduced by web developers into web pages (*e.g.,* in the middle of news articles), does not significantly impair the comprehension and retention of information by screen-reader users. However, the difference in the perception of the difficultly of tasks by the study participants was found to be significant.

Other contributions. In this dissertation I reviewed some representative related work in the field of Web Accessibility. I examined a number of web accessibility problems and coping strategies. The feedback provided by the screen-reader users in the course of user studies has provided deep insight into the needs of the target population. Finally, my work opened up new directions of research in areas of ubiquitous access, contextualization, personalization, automation, collaboration, and natural language interfaces, among others.

7.2 Broader Impact

Anticipated societal benefits. The results presented in this dissertation can be immediately implemented in the assistive technologies for the large and growing population of visually impaired users. Due to their disability, visually impaired users face significant hurdles while trying to utilize the Web. Currently available screen-reading technologies are cumbersome to use and are ineffective when it comes to accessing information or performing any web-based transactions. Indeed, equity and ease of access for people with vision impairments have huge potential for improvement.

Wider applicability of research. The techniques and interfaces discussed in this dissertation can potentially be used across different categories of disabilities, including cognitive and mobility impairments. For instance, information updates in web pages can be highlighted, or, instead, the unchanged information can be dimmed, which will make it easier for the user to focus on what is relevant to the task at hand. Also, visual interfaces can be augmented with audio input and output, increasing the multimodality of human-computer interaction. With the continued trend for miniaturization and portability of computers, small-screen devices will especially benefit from audio interfaces. In addition, web content analysis techniques produced by this work can find use in many projects that need to handle dynamic content, segment web pages, find main content, etc.

Reusing the accessibility metadata. From a broader perspective, the accessibility metadata generated by the proposed approaches can be persisted in an external metadata repository such as Social Accessibility (Takagi 2008). The resulting metadata can then be used by spiders crawling and indexing the Web, software tools that help users aggregate and filter information, personalize web pages (Maglio 2000), tailor pages for mobile devices (Bickmore 1997; Nichols 2008), automate repetitive tasks (Borodin 2008a; Leshed 2008), etc.

Collaboration with other accessibility labs. The research described in this proposal has been conducted in collaboration with other accessibility laboratories around the U.S. I have worked with Richard Ladner’s group at the University of Washington, Jeffrey Bigham at the University of Rochester, Disability Resource Center of Arizona State University, Helen Keller Services for the Blind, and others.

Educational impact. Finally, this project has provided software development training for over 100 Master's students whose master's projects I have had a pleasure of supervising over the last four years. The HearSay browser prototype has also been used by 5 Ph.D. students who were able to test their ideas and conduct research using the existing HearSay framework.

7.3 Future Work

Contextualized browsing. My initial work on context-directed browsing is only a first step toward the contextualized browsing paradigm, in which the computer will maintain the context of user actions to help drive web transactions, information seeking, or any other task undertaken by the users. The context of user actions is what can help identify the information or actions relevant at any given point. In addition, the context can be enriched with information such as the session history, current location on the page, state of a web transaction, user environment, etc.

Ubiquitous access. Mobile and remote access solutions for web browsing (Bigham 2008; Borodin 2009; SaToGo 2009) have opened new directions for research in ubiquitous computing. A separate research effort has to be devoted to the effective use of various input (*e.g.,* touch-screen, speech, etc.) and output modalities (*e.g.,* small-size screen, voice, etc.) and their combinations tailored for various browsing situations.

Natural language interfaces. The improvement of automated speech recognition and text-to-speech engines has opened up new opportunities for the use of natural language interfaces with computer software. A separate direction of research is the use of such interfaces in web browsing. Deeper understanding of the needs of the assistive technology users with a variety of disabilities (including various degrees of vision loss, mobility impairment, situational disabilities, etc.) can help formulate the requirements for natural language interfaces that will work best for people with these disabilities.

Interface personalization. The concept of universal access calls for turning assistive technology into software personalization options that will best suit users with their needs. Personalization of the interface for the unique abilities and needs of individual users can make web browsing interfaces more usable. The steps toward better personalization include the support of multiple languages and vocabularies, the use of shortcuts profiles to match the user's preferred screen reader, customization of text and speech commands, and so on.

Collaboration. The success of collaborative approaches such as the Social Accessibility Initiative (Takagi 2008) has opened up opportunities for using the power of crowd sourcing to improve web accessibility. While collaborative approaches have targeted so far simple accessibility problems such as missing headings or alternative text in images, the collaboration can be taken further to combat the inaccessibility of dynamic content (Borodin 2010), improve the accessibility of user interfaces (Bigham 2009a), automate web transactions, etc.

7.4 Conclusion

Over the last decade, the Web has become one of the primary mediums of information access, communication, employment, education, entertainment, etc. Web Accessibility is an important problem that has not been receiving enough attention due to a relatively small size of the target population and, therefore, the problem has given web content providers and assistive technology producers little commercial incentive. The content providers tend to comply with web accessibility guidelines mostly upon coercion, while the technology produces seem to provide only some basic functionalities in their software which is lagging behind the rapid development of web technologies.

Nonetheless, Web accessibility problems, as any other accessibility problems, should not be ignored, as many of us sooner or later may get afflicted with various types of disabilities, ranging from mild effects of aging to severe impairments, which make it difficult for people to function in a society without assistive technologies. By making the Web more accessible, we are rehabilitating millions of people and enabling them to function normally and efficiently in our fast-paced information-driven society, which is participating in more and more activities on the Web.

Inspired by the ideas of Universal Accessibility, this dissertation has explored several new approaches to helping screen-reader users with finding information and performing other browsing tasks on the Web faster and more efficiently. This research is one of many steps necessary to bridge the Web Accessibility Divide between the ways sighted and blind people interact with the Web. Having shown a number of possible directions of research, I hope that this dissertation will be useful to researchers and practitioners who keep building the "bridge" to a more accessible Web.

I also hope that, just as speech synthesis was initially developed for people with speech impairments, and automated speech recognition – for people with hearing impairments, the Accessible Web developed for people with vision impairments will also hit the mainstream and will benefit everyone. As we continue to explore more ways of staying connected to information (*e.g.*, while driving or walking), we will experience the accessibility problems first-hand, and that will finally give us the motivation necessary to make the Web accessible for everyone once and for all.

8. References

ACChecker. (2009). "Web Accessibility Checker." 2009, from http://achecker.ca/checker/index.php.

aDesigner. (2004). "IBM Alphaworks project donated to Accessibility Tools Framework (ACTF)." 2009, from http://www.alphaworks.ibm.com/tech/adesigner.

AFB. (2009). "American Foundation for the Blind." 2009, from http://www.afb.org.

Aggarwal, C. C. (2002). Collaborative crawling: mining user experiences for topical resource discovery. Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Edmonton, Alberta, Canada, ACM.

Allan, J. (2002). Topic detection and tracking: event-based information organization, Kluwer Academic Publishers.

Allen, J., N. Chambers, G. Ferguson, L. Galescu, H. Jung, M. Swift and W. Taysom (2007). PLOW: a collaborative task learning agent. Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2. Vancouver, British Columbia, Canada, AAAI Press.

Amitay, E. and C. Paris (2000). Automatically summarising Web sites: is there a way around it? Proceedings of the 9th International Conference on Information and Knowledge Management. McLean, Virginia, United States, ACM.

Anupam, V., J. Freire, B. Kumar and D. Lieuwen (2000). Automating Web navigation with the WebVCR. Proceedings of the 9th International World Wide Web Conference on Computer Networks : the International Journal of Computer and Telecommunications Networking. Amsterdam, The Netherlands, North-Holland Publishing Co.

Asakawa, C. and T. Itoh (1998). User interface of a Home Page Reader. Proceedings of the 3rd International ACM Conference on Assistive Technologies. Marina del Rey, California, United States, ACM.

Asakawa, C. and H. Takagi (2008). Transcoding. Web accessibility: a foundation for research. S. Harper and Y. Yesilada, Springer Publishing Company, Incorporated**:** 388.

ATAG. (2009). "W3C Authoring Tool Accessibility Guidelines." 2009.

AutoMate. (2009). "Network Automation." 2009, from www.NetworkAutomation.com/AutoMate.

Bambang, P., F. Reza, S. Andi, S. Lijing, I. W. Sugiantara and H. Stephanie (2005). AcceSS: accessibility through simplification & summarization. Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A). Chiba, Japan, ACM.

Bergroth, L., H. Hakonen and T. Raita (2000). A survey of longest common subsequence algorithms. International Symposium on String Processing and Information Retrieval.

Berk, E. (1996). HtmlDiff: a differencing tool for HTML documents, Student Project at Princeton University.

Bickmore, T. W. and B. N. Schilit (1997). Digestor: device-independent access to the World Wide Web. Selected Papers from the 6th International Conference on World Wide Web. Santa Clara, California, United States, Elsevier Science Publishers Ltd.

Bigham, J. (2009a). Intelligent interfaces enabling blind web users to build accessibility into the web. Computer Science. Seattle, University of Washington. **Ph.D.**

Bigham, J. P., R. S. Kaminsky, R. E. Ladner, O. M. Danielsson and G. L. Hempton (2006). WebInSight: making web images accessible. Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility. Portland, Oregon, USA, ACM.

Bigham, J. P., A. C. Cavender, J. T. Brudvik, J. O. Wobbrock and R. E. Lander (2007). WebinSitu: a comparative analysis of blind and sighted browsing behavior. Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility. Tempe, Arizona, USA, ACM.

Bigham, J. P., C. M. Prince and R. E. Ladner (2008). WebAnywhere: a screen reader on-the-go. Proceedings of the 2008 International Cross-Disciplinary Conference on Web Accessibility (W4A). Beijing, China, ACM.

Bigham, J. P., T. Lau and J. Nichols (2009b). Trailblazer: enabling blind users to blaze trails through the web. Proceedings of the 13th International Conference on Intelligent User Interfaces. Sanibel Island, Florida, USA, ACM.

Bobby. (1995). "Accessibility Checker." 2009, from http://www.cast.org/products/Bobby.

Bolin, M., M. Webber, P. Rha, T. Wilson and R. C. Miller (2005). Automation and customization of rendered web pages. Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology. Seattle, WA, USA, ACM.

Borodin, Y. (2006). A flexible VXML interpreter for non-visual web access. Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility. Portland, Oregon, USA, ACM.

Borodin, Y., J. Mahmud and I. V. Ramakrishnan (2007a). Context browsing with mobiles - when less is more. Proceedings of the 5th International Conference on Mobile Systems, Applications and Services. San Juan, Puerto Rico, ACM.

Borodin, Y., J. Mahmud, I. V. Ramakrishnan and A. Stent (2007b). The HearSay non-visual web browser. Proceedings of the 2007 International Cross-Disciplinary Conference on Web Accessibility (W4A). Banff, Canada, ACM.

Borodin, Y. (2008a). Automation of repetitive web browsing tasks with voice-enabled macros. Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility. Halifax, Nova Scotia, Canada, ACM.

Borodin, Y., J. P. Bigham, R. Raman and I. V. Ramakrishnan (2008b). What's new?: making web page updates accessible. Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility. Halifax, Nova Scotia, Canada, ACM.

Borodin, Y., G. Dausch and I. V. Ramakrishnan (2009). TeleWeb: accessible service for web browsing via phone. Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibility (W4A). Madrid, Spain, ACM.

Borodin, Y., S. Kawanaka, T. Hironobu, M. Kobayashi, D. Sato and C. Asakawa (2010). Chapter 18: social accessibility: a collaborative approach to improving web accessibility. End-User Programming for the Web, to be published by Morgan Kaufmann.

Brewster, S. A., P. C. Wright and A. D. N. Edwards (1993). An evaluation of earcons for use in auditory human-computer interfaces. Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems. Amsterdam, The Netherlands, ACM.

Cai, D., S. Yu, J.-R. Wen and W.-Y. Ma (2004). VIPS: A vision based page segmentation algorithm, Microsoft technical report.

Cavnar, W. B. and J. M. Trenkle (1994). N-gram-based text categorization. Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval**:** 161-175.

Cepstral. (2009). "Cepstral." 2009, from http://www.cepstral.com.

ChaCha. (2009). "ChaCha mobile search." 2009, from http://www.chacha.com/.

Chen, C. L. and T. V. Raman (2008). AxsJAX: a talking translation bot using Google IM: bringing Web-2.0 applications to life. Proceedings of the 2008 International Cross-Disciplinary Conference on Web Accessibility (W4A). Beijing, China, ACM.

CSS. (2009). "W3C Cascading Style Sheets." 2009, from http://www.w3.org/TR/REC-CSS2/.

Delort, J. Y., B. Bouchon-Meunier and M. Rifqi (2003). Enhanced web document summarization using hyperlinks. Proceedings of the 14th ACM Conference on Hypertext and Hypermedia. Nottingham, UK, ACM.

Dijit. (2009). "Dojo Dijit JavaScript toolkit." from http://dojotoolkit.org/projects/dijit.

Duda, C., G. Frey, D. Kossmann, R. Matter and C. Zhou (2009). AJAX crawl: making AJAX applications searchable. Proceedings of the 2009 IEEE International Conference on Data Engineering, IEEE Computer Society.

email2phone. (2009). "Email phone service from Across Communications." 2009, from http://email2phone.net/.

Enagandula, V., N. Juthani, I. V. Ramakrishnan, D. Rawal and R. Vidyasagar (2005). BlackBoardNV: a system for enabling non-visual access to the blackboard course management system. Proceedings of the 7th International ACM SIGACCESS Conference on Computers and Accessibility. Baltimore, MD, USA, ACM.

Forrester (2004). Accessible technology in computing – examining awareness, use, and future potential, Forrester Research commissioned by Microsoft Corporation.

Gandhre, A., P. Santhanagopalan, P. Singh, D. Ramavat, I. Ramakrishnan and H. Davulcu (2004). Creating and managing personal information assistants via a web browser: The WinAgent experience. Workshop on Information Integration on the Web. Toronto, Canada.

GOOG-411. (2009). "Google." 2009, from http://www.google.com/goog411/.

Guo, H., J. Mahmud, Y. Borodin, A. Stent and I. V. Ramakrishnan (2007). A general approach for partitioning web page content based on geometric and style information. *ICDAR*

HAAC. (2009). "IBM Human Ability and Accessibility Center." 2009, from http://www-03.ibm.com/able/.

Hamerich, S., V. Schubert, V. Schless, R. Crdoba, J. Pardo, L. d'Haro, B. Kladis, O. Kocsis and S. Igel (2004). Semi-automatic generation of dialogue applications in the GEMINI project. The 5th SIGdial Workshop on Discourse and Dialogue.

Hanson, V. L., J. T. Richards and C. Swart (2008). Browser augmentation. Web accessibility: a foundation for research. S. Harper and Y. Yesilada, Springer Publishing Company, Incorporated**:** 215-230.

Harper, S., C. Goble, R. Stevens and Y. Yesilada (2004). Middleware to expand context and preview in hypertext. Proceedings of the 6th International ACM SIGACCESS Conference on Computers and Accessibility. Atlanta, GA, USA, ACM.

Harper, S. and N. Patel (2005). Gist summaries for visually impaired surfers. Proceedings of the 7th International ACM SIGACCESS Conference on Computers and Accessibility. Baltimore, MD, USA, ACM.

Harper, S. and S. Bechhofer (2007). "SADIe: structural semantics for accessibility and device independence." ACM Transactions on Computer-Human Interaction **14**(2): 10.

iMacros. (2009). "iOpus." 2009, from http://www.iopus.com/imacros/firefox/.

JAWS. (2009). "Screen reader from Freedom Scientific." 2009, from http://www.freedomscientific.com/products/fs/jaws-product-page.asp.

JConnect. (2009). "Email by phone service." 2009, from https://www.j2.com/jconnect/twa/page/emailByPhone?rqcp=1.

JSmart. (2009). "VXML in games." 2009, from http://www.jsmart.com.

Kawanaka, S., Y. Borodin, J. P. Bigham, D. Lunn, H. Takagi and C. Asakawa (2008). Accessibility commons: a metadata infrastructure for web accessibility. ASSETS. Halifax, Canada.

Kazunori, K., A. Fumihiro, U. Shinichi, K. Tatsuya and G. O. L. k.-f. Hiroshi (2003). Flexible spoken dialogue system based on user models and dynamic generation of VoiceXML scripts. 4th SIGdial Workshop on Discourse and Dialogue.

Kelly, B., D. Sloan, L. Phipps, H. Petrie and F. Hamilton (2005). Forcing standardization or accommodating diversity?: a framework for applying the WCAG in the real world. Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A). Chiba, Japan, ACM.

Kumar, A., N. Rajput, D. Chakraborty, S. K. Agarwal and A. A. Nanavati (2007). WWTW: the World Wide Telecom Web. Proceedings of the Workshop on Networked Systems for Developing Regions. Kyoto, Japan, ACM.

Leshed, G., E. M. Haber, T. Matthews and T. Lau (2008). CoScripter: automating & sharing how-to knowledge in the enterprise. Proceeding of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems. Florence, Italy, ACM.

MAGic. (2009). "Magnification software from Freedom Scientific." 2009, from http://www.freedomscientific.com/products/lv/magic-bl-product-page.asp.

Maglio, P. and R. Barrett (2000). "Intermediaries personalize information streams." Communications of the ACM **43**(8): 96-101.

Mahmud, J., Y. Borodin, D. Das and I. V. Ramakrishnan (2006a). Improving non-visual web access using context. Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility. Portland, Oregon, USA, ACM.

Mahmud, J., Z. Sun, S. Mukherjee and I. V. Ramakrishnan (2006b). Web transactions on handhelds with less tears. Proceedings of the Workshop MobEA IV - Empowering the Mobile Web.

Mahmud, J., Y. Borodin, I. V. Ramakrishnan and C. R. Ramakrishnan (2009). Automated construction of web accessibility models from transaction click-streams. Proceedings of the 18th International Conference on World Wide Web. Madrid, Spain, ACM.

Mahmud, J. U., Y. Borodin, D. Das and I. V. Ramakrishnan (2007a). Combating information overload in non-visual web access using context. Proceedings of the 12th International Conference on Intelligent User Interfaces. Honolulu, Hawaii, USA, ACM.

Mahmud, J. U., Y. Borodin and I. V. Ramakrishnan (2007b). CSurf: a context-driven non-visual web-browser. Proceedings of the 16th International Conference on World Wide Web. Banff, Alberta, Canada, ACM.

Mankoff, J., H. Fait and T. Tran (2005). Is your web page accessible?: a comparative study of methods for assessing web page accessibility for the blind. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Portland, Oregon, USA, ACM.

McLachlan, G. and D. Peel (2001). Finite mixture models.

Mesbah, A., E. Bozdag and A. v. Deursen (2008). Crawling AJAX by inferring user interface state changes. Proceedings of the 2008 8th International Conference on Web Engineering - Volume 00, IEEE Computer Society.

Microsoft. (2009). "Microsoft Accessibility." 2009, from http://www.microsoft.com/enable/.

MicrosoftSpeech. (2009). "Microsoft Speech Technologies." 2009, from http://www.microsoft.com/speech.

Miyashita, H., D. Sato, H. Takagi and C. Asakawa (2007). Aibrowser for multimedia: introducing multimedia content accessibility for visually impaired users. Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility. Tempe, Arizona, USA, ACM.

MobileSpeak. (2009). "Mobile screen reader from Codefactory." 2009, from http://www.codefactory.es/en/.

Mukherjee, S., G. Yang and I. V. Ramakrishnan (2003). Automatic annotation of content-rich HTML documents: structural and semantic analysis. Proceedings of the International Semantic Web Conference ISWC**:** 533--549.

Mukherjee, S., I. V. Ramakrishnan and M. Kifer (2004). Semantic bookmarking for non-visual web access. Proceedings of the 6th International ACM SIGACCESS Conference on Computers and Accessibility. Atlanta, GA, USA, ACM.

Mukherjee, S. and I. V. Ramakrishnan (2005a). Browsing fatigue in handhelds: semantic bookmarking spells relief. Proceedings of the 14th International Conference on World Wide Web. Chiba, Japan, ACM.

Mukherjee, S., I. V. Ramakrishnan and A. Singh (2005b). Bootstrapping semantic annotation for content-rich HTML documents. Proceedings of the 21st International Conference on Data Engineering, IEEE Computer Society.

MVS. (2009). "W3C Markup Validation Service." 2009, from http://validator.w3.org/.

Myers, E. W. (1986). "An O(ND) difference algorithm and its variations." Algorithmica **Volume 1**: 251-266.

NDIC. (2007). "National Diabetes Statistics." 2009, from http://diabetes.niddk.nih.gov/DM/PUBS/statistics.

NetEcho. (2009). "Internet Speech." 2009, from http://www.internetspeech.com.

NFB. (2009). "National Federation of the Blind." 2009, from http://www.nfb.org/nfb/Default.asp.

Nguyen, H., T. Nguyen and J. Freire (2008). "Learning to extract form labels." Proceedings of the VLDB Endowment **1**(1): 684-694.

Nichols, J. and T. Lau (2008). Mobilization by demonstration: using traces to re-author existing web sites. Proceedings of the 13th International Conference on Intelligent User Interfaces. Gran Canaria, Spain, ACM.

NuanceCafe. (2009). "Nuance." 2009, from http://cafe.bevocal.com.

NVDA. (2009). "NonVisual Desktop Access." 2009, from http://www.nvda-project.org/.

OptimTalk. (2009). "Optim Talk." 2009, from http://www.optimtalk.cz.

Orca. (2009). "Screen reader for Linux." 2009, from http://live.gnome.org/Orca.

Paciello, M. G. (2000). Web accessibility for people with disabilities, C M P Books.

PCNow. (2009). "WebEx." 2009, from http://pcnow.webex.com.

Porter, M. F. (1997). An algorithm for suffix stripping. Readings in Information Retrieval, Morgan Kaufmann Publishers Inc.**:** 313-316.

Ramakrishnan, I. V., A. Stent and G. L. Yang (2004). HearSay: enabling audio browsing on hypertext content. International World Wide Web Conference (WWW).

Salton, G., A. Wong and C. S. Yang (1975). "A vector space model for automatic indexing." Communications of the ACM **18**(11): 613-620.

SaToGo. (2009). "Screen reader from Serotek." 2009, from www.satogo.com.

Schmandt, C. (1998). Audio hallway: a virtual acoustic environment for browsing. Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology. San Francisco, California, United States, ACM.

Sherwani, J., D. Yu, T. Paek, M. Czerwinski, Y. C. Ju and A. Acero (2007). VoicePedia: towards speech-based access to unstructured information. INTERSPEECH-2007**:** 146-149.

Shop.org. (2004). "Statistics: US Online Shoppers." 2008, from http://www.shop.org/learn/stats_usshop_general.asp.

Song, R., H. Liu, J.-R. Wen and W.-Y. Ma (2004). Learning block importance models for web pages. Proceedings of the 13th International Conference on World Wide Web. New York, NY, USA, ACM.

Sun. (2009). "Sun Microsystems Accessibility." 2009, from http://www.sun.com/accessibility/index.jsp.

Sun, Z. (2006a). Machine learning in interface design for an audio browser. Computer Science, Stony Brook University. **Ph.D.**

Sun, Z., J. Mahmud, S. Mukherjee and I. V. Ramakrishnan (2006b). Model-directed Web transactions under constrained modalities. WWW. Edinburg, Scotland.

Sun, Z., A. Stent and I. V. Ramakrishnan (2006c). Dialog generation for voice browsing. Proceedings of the 2006 International Cross-Disciplinary Workshop on Web Accessibility (W4A): Building the Mobile Web: Rediscovering Accessibility? Edinburgh, U.K., ACM.

Sun, Z., J. Mahmud, I. V. Ramakrishnan and S. Mukherjee (2007). "Model-directed Web transactions under constrained modalities." ACM Transactions on the Web (TWEB) **1**(3): 12.

SuperNova. (2009). "Screen reader from Dolphin." 2009, from http://www.dolphincomputeraccess.com.

Takagi, H., S. Kawanaka and M. Kobayashi (2008). Social Accessibility: achieving accessibility through collaborative metadata authoring. ASSETS. Halifax, Canada.

Teevan, J., S. T. Dumais, D. J. Liebling and R. L. Hughes (2009). Changing How People View Changes on the Web. UIST. Victoria, BC, Canada.

TellMe. (2009). "Voice services." 2009, from http://www.tellme.com.

Thiessen, P. and C. Chen (2007). Ajax live regions: ReefChat using the Fire Vox screen reader as a case example. Proceedings of the 2007 International Cross-Disciplinary Conference on Web Accessibility (W4A). Banff, Canada, ACM.

UAAG. (2009). "W3C User Agent Accessibility Guidelines." 2009.

VoiceOver. (2009). "Screen reader from Apple." 2009, from http://www.apple.com/accessibility/voiceover/.

VoiceXML. (2009). "W3C - Voice Extensible Markup Language." 2009, from http://www.w3.org/TR/voicexml20.

W3C. (2009). "World Wide Web Consortium." 2009, from http://www.w3.org.

WAI-ARIA. (2009). "W3C Accessible Rich Internet Applications." 2009, from http://www.w3.org/TR/wai-aria.

WAI. (1997). "W3C Web Accessibility Initiative." from http://www.w3.org/WAI/.

Wang, Y., D. J. DeWitt and J.-Y. Cai (2003). X-Diff: an effective change detection algorithm for XML documents. Proceedings of the Data Engineering**:** 519- 530.

WCAG. (2009). "W3C Web Content Accessibility Guidelines." 2009, from http://www.w3.org/TR/WCAG10/.

WebAIM. (2009). "Web accessibility in mind survey." 2009, from http://www.webaim.org/projects/screenreadersurvey/.

WHO (2002). World Health Organization: causes of blindness and visual impairment.

WHO (2003). World Health Organization.

Wilcoxon, F. (1945). "Individual comparisons by ranking methods." Biometrics **1**: 80-83.

Window-Eyes. (2009). "Screen Rrader GW Micro." 2009, from http://www.gwmicro.com/Window-Eyes.

Zajicek, M., C. Powell and C. Reeves (1999). Web search and orientation with BrookesTalk. Technology and Persons with Disabilities Conference (CSUN).

ZoomText. (2009). "Magnification tool from aisquared." 2009.

Appendix A – HearSay Shortcuts and Speech Commands

Actions	Shortcuts	Speech/Text Command
Navigation		
Help		
Earcon Help		earcon help
Help for current element	Ctrl+F1	help
HearSay documentation	Ctrl+Shift+F1	documentation/docs
Element level		
To move to next element	Down	next
To move to previous element	Up	back/previous
Block level		
To move to next block	Ctrl+Down	next block
To move to previous block	Ctrl+Up	back/previous block
Word level		
To move to next word	Ctrl+Right, Ins+Right	next word
To move to previous word	Ctrl+Left, Ins+Left	back/previous word
Spelling level		
To move to next letter	Right	next spelling
To move to previous letter	Left	back/previous spelling
Special Navigation		
Going to the next / Previous Link	A / Shift+A	next link - back/previous link
Going to the next / Previous Non-Link	N / Shift+N	
Going to the next / Previous Button	B / Shift+B	next button - back/previous button
Going to the next / Previous concept	C / Shift+C	next concept - back/previous concept
Going to the next / Previous Heading	H / Shift+H	next header - back/previous header
Going to the next / Previous Heading of level...	(1-6) / Shift + (1-6)	
Going to the next / previous Image	G / Shift+G	next image - back/previous image
Going to the next / previous Editable element	E / Shift+E	next editable - back/previous editable
Going to the next / previous Radiobutton group	R / Shift+R	next radiobutton- back/previous radiobutton
Going to the next / previous checkbox	X / Shift+X	next checkbox - back/previous checkbox
Going to the next /previous Paragraph	P / Shift+P	next para - back/previous para
Going to the next /previous List	L / Shift+L	next List - back/previous List
Going to the next /previous Table	T / Shift+T	next Table - back/previous Table
Going to the next /previous Frame	I / Shift+I	next Frame - back/previous Frame
Navigation in the page		
Go to the First element on the page:	Ctrl+Home	home
Go to the Last element on the page:	Ctrl+End	end
Repeat the current element:	Ctrl+Alt+R	repeat
Follow a link or click on a button, start editing	Enter	follow/click/select/choose/enter
Inter-page navigation		
To go to the next page	Alt+Right	next page
To go to the previous page	Alt+Left	previous page

Actions	Shortcuts	Speech/Text Command
	Multi-tab navigation	
To open a new tab	Ctrl+T	new tab
To close a tab	Ctrl+W, Ctrl+F4	close tab
To go to the next tab	Ctrl+tab	next tab
To go to the previous tab	Ctrl+Shift+Tab	back tab, previous tab
	Speech-engine-related commands	
increase rate	Ctrl+Shift+PageUp	rate up, faster
decrease rate	Ctrl+Shift+PageDown	rate down, slower
increase volume	Ctrl+Ins+F8	volume up, louder
decrease volume	Ctrl+Ins+F7	volume down, quieter
increase volume of earcons	Ins+U	volume up, louder
decrease volume of earcons	Ins+D	volume up, louder
Set Content Voice (to set voice for speaking the webpage contents)	Available through GUI settings	content voice $VOICENAME
Set Content Language (to set language for speaking the webpage contents)	Available through GUI settings	content language $LANGUAGENAME
Set Interface Voice (to set voice for speaking hearsay interface messages)	Available through GUI settings	interface voice $VOICENAME
Set Interface Language(to set language for speaking hearsay interface messages)	Available through GUI settings	interface language $LANGUAGENAME
	Multi-lingual support	
Language detection ON / OFF	Available through GUI settings	lang detect on/off
Language switching ON	Available through GUI settings	langswitch on/off
	General commands	
Goto URL (location) bar	Ctrl+L, Alt+D	address bar
Close window	Alt+F4 (Default for windows applications)	exit
Enter the edit mode	Enter (Tab goes to next focusable element)	edit
Open Input/Command Window	Ins+C	command
Version	Ctrl+Ins+V	version
Title of the page	Insert+T	title
Summary of the page	Insert+S	summary
Goto to the google search toolbar	Ctrl+E, Ctrl+K	
Open Firefox Bookmarks	Ctrl+B	
Shutdown Hearsay	Ins+X	shutdown
HearSay Find	Ins+F, temp: Ctrl+F, continue: [Shift+] F3	search_page
	Static and dynamic updates	
Next/Previous dynamic Update group	N / Shift+N	next group/previous group
Next/Previous static Update group	M / Shift+M	next update/previous update
	HearSay bookmarks	
open HearSay Bookmark window	Ins+B	favorites/bookmarks/open
Bookmark current page.	Ctrl+Alt+B	bookmark this
	TeleWeb	
Open TeleWeb menu		menu
navigate up the list of items		up/previous
navigate down the list of items		down/next
select from a list of items		select/enter/choose
close the list of items		close/cancel

Actions	Shortcuts	Speech/Text Command
Toggling layers		
Toggle layer with statically changed content	Ctrl+Alt+M	show/hide differences
Toggle layer with dynamically changed content	Ctrl+Alt+N	show/hide updates
Toggling Link layer	Ctrl+Alt+A	show/hide links
Toggling Text layer	Ctrl+Alt+T	show/hide text
Toggling Images layer	Ctrl+Alt+G	show/hide images
Toggling Headings layer	Ctrl+Alt+H	show/hide headings
Toggling Relevent elements	Alt+5	show/hide relevant
Toggling concepts/labels layer	Alt+6	show/hide labels
Add all layers	Ctrl+Alt+0	show/hide all
Open list		
Open list of links	Ins+L, also JAWS Ins+F7	list of links
Open list of headings	Ins+H, also JAWS Ins+F6	list of headings
Macro		
To start/stop recording a macro	Ins+R	record
To start/stop playing a macro	Ins+P	play [filename]
To start/stop selection	Ins+M	select
To mark a form field as variable	Ins+V	variable
To enter a text while recording which will be spoken out while playing	Ctrl+Shift+8	speak_text
To copy single element or a selection	Ctrl+C	copy
To paste what has been previously selected	Ctrl+V	paste
TeleWeb		
Open TeleWeb menu		menu
navigate up the list of items		up/previous
navigate down the list of items		down/next
select from a list of items		select/enter/choose
close the list of items		close/cancel
Hearsay modes		
To open the settings window	Ctrl+Alt+S	settings
Pause HearSay	Ctrl, Pause (toggle)	pause, stop, wait
resume reading	Ins+Down, Pause (toggle)	continue, start
Verbosity ON / OFF	Available through GUI settings	verbosity on/verbose/wordy verbosity off/terse/brief/concise
Echo mode ON / OFF	Available through GUI settings	echo on/off
Earcon mode ON / OFF	Available through GUI settings	earcons on/off
Context ON / OFF	Available through GUI settings	context on/off
Recognizer mode ON / OFF	Ins+Z (Toggle)	--- / stoplistening
Concept detection ON/OFF	Available through GUI settings	concept on/off
Segmentation ON / OFF	Available through GUI settings	segmentation on/off
Template Detect ON / OFF	Available through GUI settings	templatedetect on/off
Update notification ON / OFF	Available through GUI settings	updatenotify on/off
Label Detection ON / OFF	Available through GUI settings	autolabel on/off
Concept Labeling and Navigation		
Going to the next / Previous concept	C / Shift+C	next concept - back/previous concept
Concept detection ON/OFF	Available through GUI settings	concept on/off
Create Label	Ctrl+Alt+L	label
Delete label	Delete	delete, remove

Appendix B – Surveyed News Web Sites

ABC News	ESPN	Philadelphia Inquirer
AFP	Fox News	Prison Planet
Aljazeera	Forbes	Register
Arstechnica	Guardian	Reuters
Associated Press	Information week	Reuters India
Atlantic Online	Inquirer	Scientific American
Baltimore Sun	ITProPortal	Skynet
BBC	KCchiefs	The street
Bizjournals	LA Times	Time
Bloomberg	Legit Reviews	Times of India
Boston Globe	Life Site News	Times Record News
Business week	Market Watch	Times Online
Channel Web	MSNBC	Trusted reviews
Christian Science Monitor	MTV	Usa Today
Chicago Sun Times	Network World	Voa News
Chicago Tribune	New York Times	Xinhua
CTV	PC Magazine	ZDnet
Detroit Free Press	PC World	Wall Street Journal
Entertainment Weekly	Peoples Magazine	Washington Post