Improved Dialog Strategies for non-Visual Web-Browsing
Research Proficiency Examination

Yevgen Borodin
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794, USA
borodin@cs.sunysb.edu

September 18, 2006

Abstract

Web sites are designed for graphical mode of interaction, making it easy for sighted users to visually segment and quickly process the information on the Web. Blind people, on the other hand, can browse the Web only with the help of screen readers. The latter process Web pages sequentially and read through everything on the page, making Web browsing time-consuming and strenuous. Although, the use of shortcut keys, navigation features, and searching offers some improvements, the problem still remains: the existing screen-readers have limited ways of presenting Web page content. This report describes preliminary work on improving non-visual navigation by developing better dialog strategies for browsing the Web. The research is done in the framework of the HearSay project for developing a non-visual Web browser. The goal of the project is to make the Web more accessible for visually disabled people. Since blind people use audio to interact with computers, non-visual Web browsing should take the form of a mixed-initiative dialog-based interaction. The proposed layers of interaction are basic screen-reading, DFS/BFS, and domain-specific. The dialogs are written in VoiceXML (VXML), a W3C's standard for specifying interactive dialogs. Because there is no suitable, complete, open-source, flexible VXML interpreter to process VXML dialogs, a VoiceXML interpreter, VXMLSurf, is being developed to serve as a dialog management system within the HearSay. VXMLSurf, will be fully compliant with VXML 2.0 specifications and geared toward accessing Web content. The interpreter implements a number of extended features that provide blind users with more control over interactive browsing dialogs. To help the blind users overcome information overload, summarization techniques have to be employed to condense the content of Web pages and provide better presentation of the information. Although other researches have made considerable efforts to improve summarization, no one single summarization technique can handle non-uniformity of Web page content. This report overviews common classes and a typical architecture of pipelined summarizers, and proposes to research partial Web-document summarization. To even further accelerate Web browsing, context-browsing is introduced to help visually disabled people quickly identify relevant information within Web pages. The essence of the technique is in capturing the context of the link and using it to identify relevant information on the following page. The content of that page is then rearranged, and a modified dialog is generated, so, that the relevant information is read out first. The preliminary work described in this report shows promise to save browsing time and substantially improve the browsing experience of visually disabled people.

Contents

1 Introduction

The Web has become an indispensable aspect of our lives. In our daily activities we turn more and more to it for information: from checking the weather to searching for air tickets or getting directions. We use it for reference, news, shopping, etc. The Web was designed for graphical interaction, making all these activities simple for sighted users. Users with visual disabilities, however, have to use screen readers, which process Web pages sequentially making the process of Web browsing complicated and time-consuming.

A typical screen reader simply speaks out the content of a page to the user. The only way to interact with a screen reader is by means of shortcut keys, which allow skipping paragraphs, searching, pausing, changing the rate of speech, etc. Some screen readers also provide their users with extended features, such as summarization [83], lists of current and visited links [30], etc. All these features greatly improve Web-browsing. However, they provide little control over the way the information is presented to the users. Also, the interaction between the user and the screen reader is one-sided and requires improvement.

This report reviews the existing research in the field of Web accessibility and presents the HearSay[63] project,[1] relating the work on dialog generation, VoiceXML interpretation, summarization, and context browsing, described in the subsequent sections. The HearSay project is developed in collaboration with Helen Keller School for the Blind (HKSB) at Hempstead, NY. The design of the system is guided by people with visual disabilities who are teachers at HKSB. The ideas have been obtained and clarified in meetings with instructors and students of the school.

The rest of the report is organized as follows. Section 2 briefly overviews existing screen readers and introduces the HearSay project. Section 3 presents preliminary work on dialog generation for the HearSay project, introducing the idea of multi-layered browsing. Section 4 describes the VoiceXML Interpreter, VXMLSurfer, an essential part of the HearSay dialog management system. Section 5 studies text summarization required for effective presentation of Web page content. The section reviews classes of summarizers, a typical architecture of a pipelined summarizer, and proposes to study partial Web-document summarization. Section 6 closes the presentation of preliminary work by introducing context-enabled browsing. Following the description of evaluation plans in Sections 7, the report is concluded with a brief summary of the report in Section 8.

[1]Supported by NSF grants CCR-0311512 and IIS-0534419.

2 Non-Visual Browsing

2.1 Review of Web Accessibility

Web accessibility has long been problematic for visually disabled people. With the progression of research in the field of accessibility, new guidelines were created to make the Web more user-friendly to visually disabled people [1]. Major software producers, such as Microsoft and Adobe, strive to make their products more accessible for blind users. Nevertheless, the problem still remains: as the Web developers continue to target sighted users, blind people are left at a serious disadvantage when it comes to browsing the Web.

Several research projects devoted to non-visual Web accessibility emerged [8, 17, 27, 31, 38, 78] in response to the need for adapting Web browsers to support accessibility [18, 23]. As a result, specialized audio browsers [40, 67] and screen readers have come out.

Screen-readers are specialized software products that are capable of reading the content of the screen, offering some Web browsing capabilities. The JAWS [30] screen reader and IBM's Home Page Reader [3, 11, 56, 73] allow moving from link to link when browsing a Web page. BrailleSurf [24] allows filtering of HTML into Braille. BrookesTalk [83] facilitates non-visual Web access by providing a summary of the Web page, by using NLP-based text abstracting and summarization techniques. Users get an overview of the content of the current Web page by listening to its summary. An example of a VoiceXML browsing system, which presents information sequentially, is described in [55].

The work described in this report strives to bring together the fields of natural language processing, information retrieval, and accessibility to produce a superior non-visual Web browser, HearSay [63]. The research is done under the leadership of Dr. I.V. Ramakrishnan, Dr. Amanda Stent, and Dr. Susan Brennan. The HearSay browser will include all of the basic functionalities of a regular screen reader. It will provide a flexible and intelligent dialog interface on top of an advanced Web-content analysis engine. The browser will also support multiple platforms and will be easily extensible.

The HearSay system already has basic non-visual browsing facilities and is capable of performing structural and semantic analysis of Web pages[49, 71]. The system provides an infrastructure for analyzing, grouping, and partitioning Web-page content. After processing a Web Page, HearSay uses interactive dialog strategies to present the content of the Web page to the users. The following subsection describes the architecture of the HearSay.

2.2 HearSay Audio Web Browser

The HearSay system is comprised of the following modules: Interface Manager, Context Analyzer, Browser Object, Frame Tree Processor, and Dialog Generator. The architecture of the system is shown in Figure 1.

Users communicate with HearSay through the **Interface Manager**. The module uses VoiceXML dialogs to interact with the users and present Web page content. In its current configuration, the system allows only keyboard input via a VoiceXML[2] interpreter, VXMLSurfer. The system will be soon extended with a speech recognition engine. The Interface Manager provides both basic and extended screen-reader navigation features, such as shortcuts.

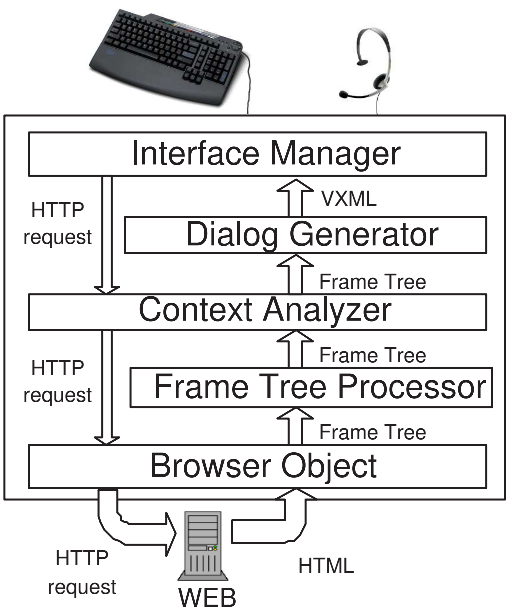


Figure 1: Architecture of HearSay

Context Analyzer module is called twice for each Web page access. When the user follows a link, the module collects the context from the current page. When a new Web page is retrieved, the module executes an algorithm to identify the relevant information within the page before it is presented to the user.

The **Browser Object** downloads Web content every time the user requests a new page to be retrieved. The module is built on top of the Mozilla Web Browser [48] coupled with JREX [32] Java API wrapper. JREX has been extended to extract Mozilla's internal representation of Web pages.

Frame Tree Processor uses JREX API to extract the *Frame Tree*[3] representation of the Web page from the Browser Object. The Frame Tree Processor uses a number of heuristic algorithms to clean, reorganize, and partition the frame tree. Subsequently, Context Analyzer reorders the frame tree before passing it to the Dialog Generator.

The **Dialog Generator** module uses a collection of Voice-XML dialog templates to convert the frame tree into Voice-XML dialogs. The latter is then delivered to the Interface Manager. Exhaustive description of the HearSay modules is not in scope of this report. Some more architectural details appear in [63].

[2] *VoiceXML* (VXML) [75] is the W3C's [76] standard XML format for specifying interactive voice dialogues between a human and a computer.

[3] *Frame Tree* is defined as a tree-like data structure that contains Web page content, along with its formatting information, specifying how the Web page has to be rendered on the screen.

3 Dialog Generation

3.1 Review of Dialog Generation

The weakest point of all major screen-readers and non-visual Web browsers is the simplicity of content presentation. An effective non-visual browser has to use a sophisticated dialog system to give users more control over the dialog flow and provide more ways to access the information. To provide blind users with multi-layer Web access, a non-visual browsing system has to first segment and analyze Web page content; it then has to implement effective dialog generation and processing techniques. This subsection briefly reviews the research related to these tasks.

Web content analysis uses segmentation of Web pages into blocks of data, which are then analyzed, grouped, labeled, etc. Most of the techniques used for segmentation are either domain specific [19, 73] or rely on sets of manually specified rules [13, 20, 80, 82]. Some of these approaches are not suitable for dynamically changing Web sites. This work builds on the previous research on structural and semantic analysis described in [50, 63]. The geometric partitioning method used in HearSay is fully automated and scalable over domains and does not depend on manually specified rules or domain knowledge. Semantic partitioning of Web pages has been described in [50, 51, 52]. The partitioning methods used in the HearSay system depart from these works, as they do not require any semantic information (in the form of ontologies) to partition a Web page. In contrast, the system uses geometric information available in the frame tree (Section 2.2).

Dialog generation is a broad area of research in Natural Language Processing (NLP) field. Substantial research has been done on various aspects of dialog generation, including VoiceXML dialogs. Mixed initiative dialogs and generation of cooperative responses using VoiceXML are described in [35]. An example of use of VoiceXML dialogs in Web browsing is given in [55]. However, dialog generation for non-visual Web Browsing is not a well-studied subject. This work is based on and extends the research on the use VoiceXML dialogs described in [63].

Further study of dialog generation for non-visual Web access is required to provide blind people with better ways of exploring Web content. This research will improve non-visual Web browsing by offering multi-layer dialog-based interaction with the HearSay Web browser. The approach will build on features of the best screen-readers, extending the levels of interaction and information presentation[4]. This work extends and builds on Chapter 5 of the Ph.D. thesis [70].

3.2 Preliminary Work on Dialog Generation

Non-visual Web navigation can be viewed as a mixed-initiative dialog interaction between a human and a Web browser, in which the person asks questions and the browser gives answers or asks for clarification. The browser can make suggestions and prompt the user for input, e.g. if a Web form has to be filled out. User input can be in text or speech. The new approach will allow to access Web pages at three different layers of dialogs. The system will generate all three layers and will allow the user to switch between these layers and browse the Web at the level, which is most convenient for a given task.

[4]An extended abstract was accepted to Doctoral Consortium at ACM SIGACCESS 2006.

Basic Screen-Reader functionality is the first layer of dialog generation. At this level the interaction between the user and the screen-reader is limited to an event-driven dialog, in which the page is simply read out to the user. The user can press standard screen-reader shortcut keys that control the flow of the dialog, e.g. skip, pause, repeat, change voice settings, etc. All these shortcuts are implemented within the dialog interpreter, while the dialog itself can be a simple narration, in which the screen-reader has the initiative and the user only selects the dialog direction.

DFS/BFS navigation is the second level of content presentation, requiring more sophistication in generating dialogs. The user should be able to freely navigate on the Web page using breadth-first, depth-first, and mixed approaches. The user will be able to choose which part of the page to listen to, at which level of detail, as well as get brief summaries of the page parts. The implementation of this level will require a mix of simple partial-document summarization techniques to be able to abstract, classify, and label different types of data.

So far, most of the research on Web page and document summarization concentrated on full- and multi-document summarization. Section 5 reviews the preliminary work on summarization that is aimed at provide intelligent labels, summaries, and abstracts of parts of Web pages. The choice of techniques will depend on the type of data in the selected section of a Web page.

Domain-Specific level of dialog generation will require specialized templates, e.g. news, shopping, etc. The dialog generator will analyze the Web page content and try to fill the corresponding template with information. The resulting dialog will give the users a more structured view of the page. The implementation of this layer may also require the use of classifiers to determine the domain a page belongs to.

Dialogs are enriched with *earcons*, special sounds that could help distinguish between plain text, links, visited links, taxonomies, etc. Dialogs will be made adaptable to the user's choice of vocabulary, verbosity, navigation style, etc. This will require the use of sophisticated shortcut controls and speech grammars to be able to support and interpret a variety of user commands.

3.3 Future Work

Dialog Generator. The HearSay system already implements the first layer of dialog interaction, supporting basic screen-reading and extended speech controls. The implementation of the second layer of dialog generation will start as soon partitioning module is added to the Web page processing engine of HearSay and as reasonable partial-document summarizer is available. Dialog templates have to be developed for the third layer of domain-specific dialogs. And, finally, the dialog generation system and the VoiceXML interpreter have to be extended to support adaptive dialogs. More research has to be done on machine-learned selection of mixed dialog strategies based on the type of data present in different parts of Web pages.

VoiceXML Interpreter. Since no suitable open-source VXML interpreter is available, a custom VXML interpreter has to be developed to process the VoiceXML dialogs. The interpreter will be designed to comply with the VoiceXML 2.0 specification and allow both voice and keyboard input. The interpreter will go beyond the specifications and provide more control over the dialogs, speech properties, and event-handling.

4 VoiceXML Interpreter

4.1 Review of VoiceXML

As the Web matures, more and more services are provided online. Web standards emerge to facilitate the development of new tools and services, VoiceXML 2.0 [75] being one of such standards. VoiceXML (VXML) is a W3C's [76] standard XML format for specifying interactive voice dialogues between a human and a computer. VXML employs speech recognition grammars, embedded JavaScript, event handlers, etc. It is a powerful language that can describe complex audio dialogs and have multiple applications.

VXML is widely used to describe menus in telephone systems. It can be used to disseminate information to public through phones or computer terminals, e.g. [35] describes interactive bus-information dialog system in Kyoto City, Japan. Another project, GEMINI, uses VoiceXML dialogs in a multilingual interactive natural language interface [25]. VXML can also be used in computer games [34]. A soon-to-become-important application of VXML is non-visual voice browsing, where Web page content is converted to VoiceXML dialogs to help blind people browse the Web. Examples of VoiceXML dialogs for Web browsing are given in [55, 63].

Popular screen-readers, such as JAWS [30] and IBM's Home Page Reader [73], 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 dialog management system has to be modified to accommodate any changes. On the other hand, systems using VoiceXML dialogs are more flexible, because every VXML dialog is a dialog manager in itself. HearSay [63] is one such system that employs the flexibility of VoiceXML dialogs. The system endeavors to dynamically create multiple layers of dialogs, which, in time, will allow mixed-initiative dialog-based interaction between a user and a Web browser. VoiceXML allows programming interactivity within dialogs.

A VXML interpreter is required to process VoiceXML dialogs. Unfortunately, there are no completed, open-source VXML interpreters. The existing open-source interpreters are supporting 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, like OptimTalk [57]. Besides, most VoiceXML interpreters are geared toward telephony applications [4].

The goal of this project is to develop a open-source, modular, multi-platform, extensible VXML interpreter that will fully implement and extend the VoiceXML 2.0 specifications. From the start, the interpreter is geared toward non-visual Web browsing to provide blind users with more control over dialog flow as they access the Web. The additional features will go beyond current VXML specifications. The results obtained in the course of the project will be submitted to W3C's for review to potentially contribute to future VoiceXML 3.0 recommendations[5].

[5] An extended abstract was accepted to ACM Student Competition at ACM SIGACCESS 2006.

4.2 VXMLSurfer

This section describes VXMLSurfer, an implementation of a VXML interpreter. VXMLSurfer is already used as a part of HearSay system for interpreting VoiceXML dialogs. It is supporting a large subset of VXML tags, including a full implementation of event and variable spaces. However, the interpreter is still in the development. For a complete listing of the features that have to be implemented please refer to VoiceXML 2.0 specifications [75]. This section concentrates on the extended capabilities and the application of the interpreter.

Extended Features. VXMLSurfer has been extended with additional features that control voice properties, flow of dialogs, and advanced event-handling. The ideas of key controls were borrowed from the state-of-the-art screen reader, JAWS [30]. Shortcuts will be continuously added per request of end-users. For the complete listing of current interpreter- and HearSay-specific shortcuts please refer to Appendix A, Table 1. Advanced features can be turned on/off for complete backward compatibility with VXML 2.0 specifications.

To improve user experience, the interpreter implements advanced voice controls that allow the use of shortcuts, such as pause, resume, restart the utterance, as well as change the voice, pitch, rate, and intensity of the speech. Additional shortcuts will allow skipping the content at various scopes, e.g. sentence or paragraph. Key strokes are treated as events, and can have predefined event handlers. Event handling is implemented in a way that will allow to define new events and event handlers at different scopes of the VXML dialog. Thus, any event handlers can be overridden, making the interpreter very flexible.

VoiceXML can be used to specify both the dialog and the dialog manager of the voice browser and is powerful enough to implement most of the navigational controls. However, while that works well for smaller dialogs, it was experimentally determined that it often leads to significant increase of the VoiceXML dialog size. Implementing some of the controls to navigate through a dialog within the interpreter will reduce the complexity of creating VoiceXML dialogs and programming the dialog manager. The navigation features that have yet to be implemented include searching, moving in any direction in the dialog, and switching between different views of the same dialog. This will require saving various states of the interpreter and will be similar to dialog debugging.

Application. VXMLSurfer is a part of the HearSay system. The Hear-Say infrastructure has superior browsing facilities and can perform simple structural and semantic analysis of Web page content. The system uses Mozilla Web browser [48] to obtain a frame tree representation of Web pages, which are then further analyzed, grouped, and partitioned (Section 2.2).

Subsequently, a dialog generator uses the information of the frame tree and a number of VoiceXML templates to generate multiple layers of dialogs, which are then processed by the interpreter. The three layers of Web browsing dialogs include basic screen-reading, BFS/DFS navigation, and domain-specific dialogs. The generator is logically separated from the interpreter, so that they are easier to maintain and, if necessary, can be replaced individually. The exhaustive description of the Dialog Generator and other HearSay modules is not in scope of this paper. For more information refer to [63] and Section 3.

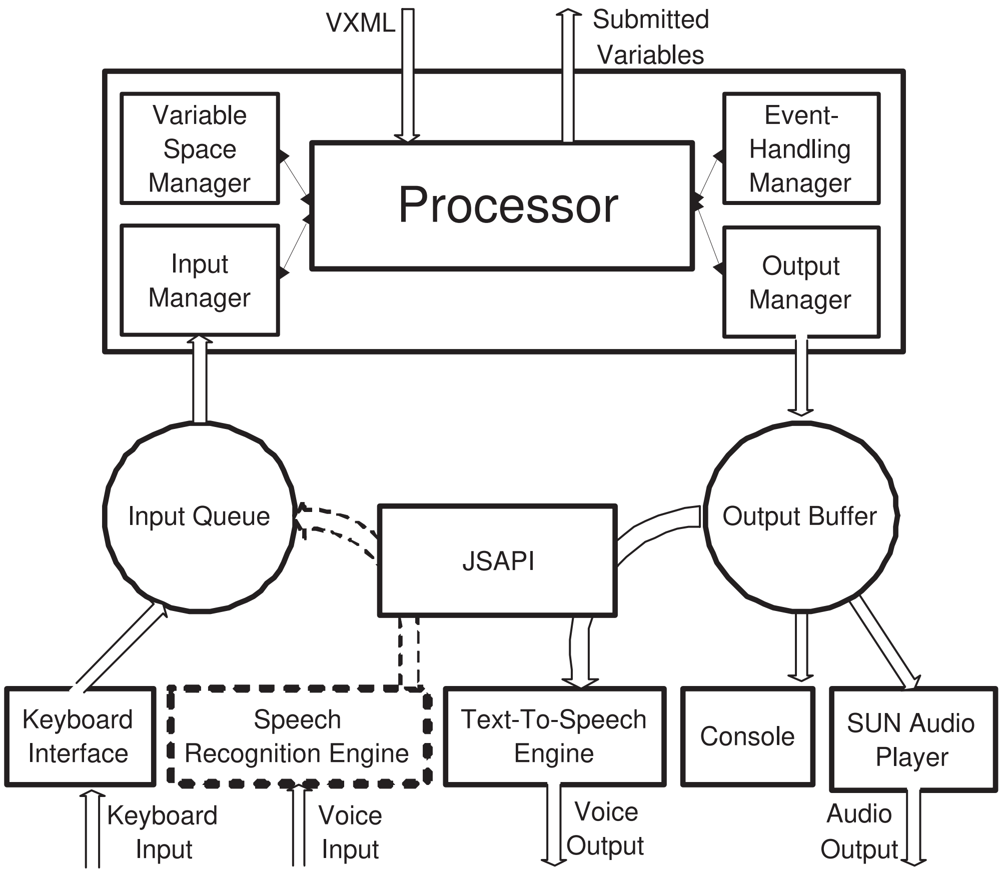


Figure 2: Architecture of VXMLSurfer

4.3 Architecture of VXMLSurfer

Architecture of VXMLSurfer. To be platform independent, VXMLSurfer is written entirely in Java. It can be viewed as a multi-threaded application with separate threads for input, output, and interpreting VoiceXML (Figure 2).

The core thread of the VXMLSurfer is the *Processor*, responsible for interpretation of the VXML dialogs. The Processor closely interacts with its sub-units: *Variable Space, Event Handling, Input*, and *Output Managers*. Due to the nature of VXML, where input and output take most of the time, the processor-run between input and output is considered atomic to enforce state consistency. The interpreter can be interrupted by events during the input and output operations.

Input and Output Managers communicate with the input and output threads through a synchronized *Input Queue* and an *output buffer*. Multiple input commands including shortcuts can occur concurrently filling the input queue. The Input Manager consumes the queue as it processes the input. Only one output process is allowed at any point of time. The interpreter supports output to text-to-speech engine, audio player, and console.

The Event-Handling Manager loads default session-level event handlers and maintains event scoping. Default VXML event handlers are supported as directed by the VXML 2.0 specifications. Event handling has been extended to support shortcut key-pressed events. The users of the interpreter are enabled to define any VXML event handlers to process the shortcuts, as well as reassign any key combinations to certain hard-coded event handlers. Variable Space Manager handles the implements variable look up tables and scoping.

The interpreter is designed to be modular and to be able to use various voice-recognition and text-to-speech engines through Java Speech API (JSAPI) [33]. Text-to-speech conversion is done via FreeTTS v1.2.1 [21], a freely available speech engine. The VXMLSurfer is currently supporting only three American-English voices: a female and two male voices. In its current configuration, the interpreter allows only keyboard input. However, CMU Sphinx [69] voice recognition engine will be integrated in the interpreter as soon as speech-grammar support is added. VXMLSurfer will provide its users with more control through both shortcut keys and voice commands.

4.4 Future Work

The VXMLSurfer is a work in progress. The implementation of the interpreter is interesting from an engineering perspective and is absolutely necessary for further research in dialog generation. VXMLSurfer already supports the first layer of HearSay dialog interaction, allowing basic screen-reading and extended speech controls. Processing of the second and the third dialog layers will require implementation of more advanced dialog navigation controls. And, finally, VXMLSurfer has to be extended to support adaptive dialogs. A series of progressive evaluations of the HearSay system will be performed by the students of the Helen Keller School for the Blind as the system takes shape. A number of features have yet to be implemented before VXMLSurfer is completed. Among the ongoing sub-projects there are:

- Adding a Java spell-checker module with accompanying VXML dialogs and event handlers to provide an extensive voice interface to spell-checking.
- Adding more voices for FreeTTS, including English and foreign voices.
- Adding other JSAPI-compliant text-to-speech processors.
- Adding JSAPI-compliant speech recognition engines.
- Adding a character converter with accompanying VXML dialogs to allow on-the-fly control over conversion maps of certain groups of characters, symbols, punctuation, etc. to be translated to the corresponding English words.
- Adding JavaScript/ECMAScript interpreter in Java to support scripting capabilities of the VXML.
- Adding ABNF speech-recognition grammar support to interpret the speech input.

5 Summarization

5.1 Review of Summarization

Recent years have shown a tremendous increase in the amounts of textual data available on the Internet. The number of pages indexed by Google is estimated to be near 10 billion and is still growing. Among the factors contributing to this growth are massive digitalization of printed texts (Google Books) and the increased reliance on electronic media for information distribution (news, blogs, personal Web sites)[2].

While computers are keeping up with this growth, humans are easily overwhelmed with such amount of textual data. We are restricted by time, space, and/or physical abilities. Some of these constraints can be alleviated by text summarization - automatic restating of main points or facts presented in a text. Summarization finds its applications in search engines, e-news, handheld devices, and software for visually impaired people - to mention just a few[5, 9, 60].

Summarization is hindered by non-determinism of natural languages, imperfect or irregular input, word sense disambiguation, syntactic ambiguity, and other linguistic problems. Many of these difficulties can be eased by means of stochastic, probabilistic, and other statistical methods. The existing summarization techniques are deeply related to information extraction and machine learning[12]. Thus, they often involve multiple disciplines: Computer Science, Linguistics, and Statistics.

A number of methods have been developed for automatic summarization. These methods vary in their approach (e.g. extraction/abstraction/query), domain (e.g. independent/news/e-mail), dimension (e.g. multi-document/-lingual), genre (e.g. minutes/headlines/outlines), etc.[61, 60, 79] The latest research efforts were directed to simulations of language models, social networks, page rank models, centrality approaches, etc.[37, 61, 77].

Although considerable efforts have been made to improve summarization techniques, general purpose summarization still remains unsatisfactory due to complexities of the task. Besides that, Web page content is rarely homogeneous or uniform and could contain links, taxonomies, images, etc., often containing semantically unrelated passages that are not always punctuated. The following subsections review the common architecture, and attempt to classify methods commonly used by the summarization engines.

5.2 Architecture of a Typical Summarizer

Most summarizers have a pipelined architecture, in which the input data goes through several stages of preprocessing before the final summary is obtained. Preprocessing employs numerous natural language processing (NLP) and statistical techniques and often makes use of ontologies such as WordNet[47]. The pipeline usually consists of the following stages: classification, extraction, segmentation, tokenizing, tagging, parsing, analysis, and rewriting stages.

Classification. Document classification is a preprocessing step often employed by domain-specific summarizers [44, 2, 53]. Another type of document classification is employed by the Columbia's summarizer that routes the document based on whether it describes a single-event, multi-event, or person-centered event [45]. For partial Web-document summarization it is important to classify parts of the Web page as taxonomies, headlines, articles, etc., and use the appropriate summarization techniques on those parts.

Collection. Depending on the type of data, the input of a text summarizer can be a single document or multiple documents. It could be plain text, HTML, or any other document format. The text has to be extracted from the documents, discarding or storing the formatting information. Depending on the domain and dimensions of the summarization task, sophisticated information retrieval techniques can be used. For example, Columbia's NewsBlaster[44] crawls the Web, collecting and segmenting news Web pages into the text, accompanying pictures, etc.

Segmentation. Subsequently, the extracted information can be segmented into semantically related items [68, 6, 65]. HearSay uses Mozilla[48] to obtain a frame tree of the Web page, which is then clustered and partitioned. More on Web Page segmentation can be found in Section 3.1.

Tokenizing. The text then has to be tokenized into syntactic units, be it words for noun-phrase extraction [53], sentences for sentence-level extraction [66, 45], or even paragraphs [68]. Since the partitions obtained after Web page segmentation contain text of various length, different tokenizing approaches will have to be used. For example, sentence extraction can be used for news articles, paragraph extraction will be effective only for very long documents, while noun-phrase extraction will work best for Web-page segments that contain no well-formed sentences.

Tagging. Once the text is tokenized, it often has to be tagged. Different tagging schemes can be employed. Part-of-speech (POS) tagging is often used to identify the sentential roles of words. For example, in Columbia's multi-document news summarization, POS tagging has to be performed in the creation of template-based summaries [62]. Assigning weights to the tokens can also be done during tagging. For example, in noun phrase extraction, higher weights can be assigned to named entities and lower weights can be given to verbs.

Parsing. Parsing is an optional step that could be useful if it is necessary to understand the sentence structure for summarization. This is especially helpful when sentences have to be rephrased or even semantically labeled. One example of using parsing is in question-answering [10].

Analysis. The stage of analysis broadly encompasses any techniques used to determine the relevant information worthy to be included in the summary. These are often linguistic, statistical, machine-learned or heuristic methods.

Rewriting. Sentence rephrasing, or rewriting, is an advanced step often used in abstractive template-based summarization[62]. Other variations of rewriting include alias expansion, e.g. Bush, he, and president could be identified as aliases for the President of the United States; geography normalization, e.g. Baghdad is identified as the capital of Iraq; duplicate elimination, e.g. repeating phrases are removed or replaced by their aliases.

5.3 Classification of summaries

The field of automatic machine summarization is almost 50 years old [42]. Over the years numerous types of summarization approaches and techniques have been discovered. Generally, most summarizers can be categorized by thew following features.

Type.) The majority of summarizers produce *generic* summaries, i.e. summaries that contain most relevant information from their own perspective. Some other summarizers attempt to give *query-based* summaries, where the user determines the subject of interest. e.g. question-answering summarizers [10] or search engines fall into this category.

Domain.) Summaries can also be classified by the domain of knowledge they are covering. *Domain-independent* summaries [68] tend to be more difficult to produce and are less effective, because depending on the subject of the original document, different aspects can be deemed important. *Domain-specific* summarizer, on the other hand, are easier to fine-tune to produce summaries that are specific to news [44, 2], emails [53], journals, medical literature, etc.

Dimension. The majority of first summarizers were concerned with producing *single-document* summaries. The more recent summarization engines aimed at producing *multi-document* summaries. The major difficulty in dealing with many documents is in identification of related documents, removing overlapping information, and identifying inconsistencies. This is especially true for new summarizers [45, 62]. More recent attempts have been made to target *multi-lingual* documents [74].

Genre. Summaries can also be categorized by their genre. Producing summaries in a certain form often involves template filling or some kind of rewriting. Examples of possible genres of summaries include *biographies* [66], *chronologies* [62], *minutes, headlines* [15, 36], *outlines, personalized summaries* [46], etc.

Approach. Most existing summarizers are based on some type of *extraction*, i.e. they either extract words [53], sentences [66, 45], or even paragraphs [68]. The text (often unmodified) is extracted based on some type of weighting scheme, based on frequency, clue words, position, etc. Only the text units with highest weight are extracted into the summary. *Abstraction*, on the other hand, deals with rewriting the text. It is a more difficult approach, that often starts with extraction, and then proceeds to paraphrasing sections of the source document [62].

Automatization. The last feature that classifies summarization engines is automatization. There are *fully automated summarization (FAS), machine-aided human summarization (MAHS)*, and *human-aided machine summarization (HAMS)*. The majority of summarizers are fully-automatic. The use of the last two types of summarizers usually involves annotating texts before summarizing or using special software that helps identify and keep track of the relevant pieces of information in the text. Such summarizers are not in scope of this report.

5.4 Preliminary and Future Work on Summarization

Several open-source summarization systems have been selected for evaluation: Mead [74], OTS [58], and Classifier4J [14]. The experiments will consist of running the summarizers on different types of Web content, whole pages, parts of pages. The output of the summarizers will be evaluated by humans to determine the best summarization approaches applicable to certain types of data in Web pages.

A naive keyword-extractive summarizer has been written in Java. It extracts nouns based on their frequency in the text. The summarizer will be run on the data blocks of frame tree in the context of the HearSay system [63]. The obtained summaries will be used as labels in different types of dialogs generated by the dialog system (Section 3.2). For more information on the architecture of the HearSay system please refer to Section 2.2.

The summarizer is intended to produce generic, domain-independent summaries. Since Web page collection and segmentation are already done by the HearSay system, the pipeline of the summarizer currently includes tokenizing, part-of-speech tagging, and the analysis steps. The analytical engine of the summarizer collects the nouns, performs a naive frequency count, and select the most-often-occuring nouns.

The next step is to consider extracting noun phrases, rather than picking single nouns. The same procedure for frequency counting can be used, but the adjoining nouns should be clustered into groups. This technique will prevent splitting noun phrases containing named entities. Subsequently, frequency counting will be compared to *Tf*idf* weighting method [7].

It will be the responsibility of the HearSay modules to use the summarizer on the blocks of the frame-tree that contain homogeneous semantically-related data (see the definition of Marker Nodes in Section 6.3). It is planned to extend the summarizer to handle different types of content, e.g. the noun-phrase extraction will work best on short spans of text, while sentence extraction will be used on articles; paragraph extraction will be effective only when applied to really long texts.

To make the extraction even more effective, the summarization engine should be extended to make use of anaphora resolution, alias expansion, and geography normalization. One system that is capable of accomplishing this task is Lydia [41], developed by Dr. Skiena's research group at Stony Brook University. The possibility of using Lydia as a server, that would perform on-demand processing, is currently being explored.

6 Context Browsing

6.1 Review of Context Browsing

As we browse the Web, we have to filter through a lot of irrelevant data. For example, most Web pages contain banners, commercials, navigation bars, and other data distracting us from the information. Sighted individuals can process visual data in no time at all. They can quickly segment any Web page and identify the information that is most relevant to them. Blind users, on the other hand, have to use screen-readers. The latter process Web page content sequentially, i.e. they first read through the menu bars, banners, ads or anything else that comes before the desired information. To alleviate this problem, screen readers often permit to skip chunks of data in the order they appear on the page. Unfortunately, in many cases, users still have to listen or skip through a substantial part of page content before they get to the information.

To help users locate the information quicker, a number of screen readers [30, 3] allow keyword searching. This assumes that the users know what they are looking for. In some cases searching may help skip directly to the information. However, simple searching has two problems: it works only for exact string matching and it disorients users in case of a wrong match. In both cases users have to start from the beginning of the page. One way to help the blind users locate the information quickly is to use the context of the followed link to locate the relevant information on the next page and re-arrange it before presenting to the users.

Information rearrangement in Web pages typically relies on rules [64] or logical structures [59, 72, 28, 54]. The ASTER system [64] of Raman, permits visually challenged individuals to manually define their own document reading rules. The work in [59] describes a manual annotation framework, based on contextual graphs, that allows for flexible navigation between the segments in a page. Other researchers proposed the idea of extracting content using semantics some se[29]. They describe a framework for manual annotation of the content w.r.t a schema representing the task a user wishes to accomplish. These annotation rules are also site specific, and, hence, not scalable over content domains. Other research work describing information rearrangement for better audio rendition include [28, 81]. In [28], syntactic changes, like shrinking or removing images, have been discussed. Noise elimination from Web pages is described in [81]. Removing images or eliminating noise (e.g advertisement) can improve browsing experience for visually challenged people. These ideas can also be incorporated in the HearSay to further improve user experience.

The use of context in non-visual Web navigation is not a well-studied problem. A technique resulting from early efforts on context retrieval for non-visual Web access is mentioned in [26]. The middle-ware described in that paper captures the context from the text around the link. Context is collected in both directions from a link until the first tag or punctuation mark is encountered. The system may not always perform well, e.g. it stops collecting the context and returns an incomplete set of context words when it encounters a bolded word in the middle of a sentence.

The following sections describes collaborative work of Ph.D. students Yevgen Borodin, Dipanjan Das, and Jalal Mahmud under the direction of Dr. I.V. Ramakrishnan[6]. This work can potentially help individuals with visual disabilities to quickly identify relevant information on following a link, thus, considerably reducing their browsing time.

[6] An extended abstract was accepted to ACM SIGACCESS 2006.

6.2 Implementation and Results of Context-Directed Browsing

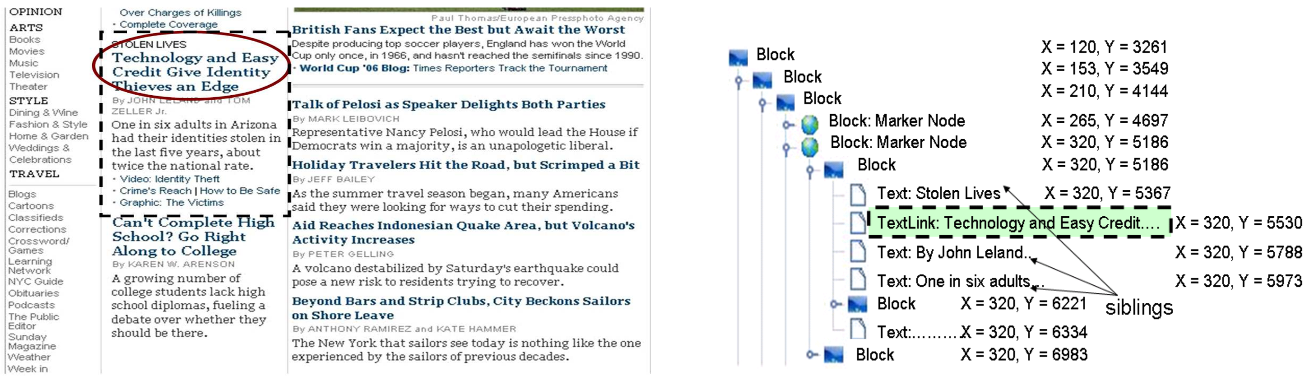
Context browsing is the technique for context identification and information rearrangement, based on the structural and visual organization of Web pages. We have come up with a robust context identification scheme, using the information from logical organization of a Web page. Observing that semantically related items exhibit spatial locality [52, 51] in Web pages, we capture a semantically meaningful set of context words by using frame tree logical structure. We collect context words from frame tree nodes (sibling nodes) that are geometrically adjacent to the link, as well as from the link node itself.

The identification of relevant information on any distinct Web page is subjective. However, as soon as the user follows a link, it is often possible to use the context of the link to determine the relevant information on the next page. For example, when the user follows the headline news link, circled in Figure 3 (a), the news article in Figure 4 (a) must be read out first. To identify that article as the "relevant information", all the words contained in the dotted box in Figure 3 (a) are identified as *context* and collected. Then, the content of the following page can be rearranged to present the article first. The rest of the Web page is read sequentially as before.

In contrast, we aim to help visually disabled users to quickly identify relevant information on following a link, thus, considerably reducing their browsing time. We have presented quantitative evaluation of our system, which shows the gains in browsing efficiency over a state-of-the-art screen reader, such as JAWS.

Evaluation. A series experiments have been conducted to evaluate the accuracy of the context browsing method, as well as its quantitative performance against JAWS, the state-of-the-art screen reader. In our evaluation we used twenty four Web sites spanning four content domains: *news*, *books*, *consumer electronics*, and *office supplies*. We focused on the time taken to access the information with and without context-directed browsing. We allowed the use of JAWS shortcut keys to skip blocks of text, thus, accelerating browsing. Still, compared with JAWS, our system achieved browsing time reduction of 63.7% in news domain, 57.1% in books domain, 45.1% in electronics domain, and 54.3% in office supplies domain.

We calculated how well our system identified relevant information in each Web site. Our algorithm showed a reasonable accuracy of over 80% in all four content domains. It is notable that it performed the best in the news domain (88%). The structural and geometric organization is often much better in News Web sites than in other domains. Since our context analysis algorithm depends on the geometric organization of Web pages, it explains why our system performed better in the news domain. The results showed that our technique works best on well-structured, information-driven, dynamic Web sites such as news engines, encyclopedias, online stores, etc. The evaluation demonstrated that the use of context can save browsing time and substantially improve browsing experience of visually disabled individuals.



(a) Source Page (b) Source Frame Tree

Figure 3: Context Identification and Ranking

6.3 Context Analysis Algorithm

This subsection assumes knowledge about the architecture of HearSay [63], presented in details in Section 2.2. Context analysis is performed every time the user follows a link using a HearSay system. The Context Analyzer module uses the frame tree of the source Web page to identify context around the link. It then utilizes this contextual information to identify relevant portions of text in the destination Web page. The module rearranges the frame tree of the Web page in such a way that the relevant information will be read out first. Before we proceed to describe the algorithm in greater detail, we formally define the notion of context.

Context: Given a frame tree of a Web page and a link with its corresponding tree node n_i, *context* is defined as a multiset that includes the text of the link node, along with the text contained in the m sibling nodes $\{n_1, ..., n_{i-1}, n_{i+1}, ..., n_m\}$ of n_i.

Consider an example when the source is the front page of *The New York Times* news Web site (see Figure 3 (a)). The context of the encircled link is the text surrounded by the dotted line. Figure 3 (b) shows the corresponding frame tree with the link and its siblings.

The context processing algorithm has five distinct phases: context identification, context ranking, context matching, block ranking, and block rearrangement. The first 2 phases work with the source Web page, while the remaining 3 operate on the destination page.

1. **Context Identification:**

   The following steps describe the context identification phase of the algorithm:

   a. Given the frame tree of a Web page, identify all the nodes containing the URL selected by the user.

   b. Extract the text of the link from each of these nodes, tokenize it, and store it in a multiset, after removing all function words[7] from the multiset.

   c. Identify the siblings $\{n_1, ..., n_{i-1}, n_{i+1}, ..., n_n\}$ of the link node n_i in the frame tree.

   d. For each sibling n_j, check if it contains any text.

   e. If yes, extract, tokenize, and store the text in a multiset, after removing the function words.

In our example, Figure 3 (a), when the user follows the link, the area enclosed by the dotted line is identified as the context of the link. Step (a) of the algorithm identifies that there is only one instance of the URL present in the Web page. Step (b) creates a multiset containing the text of the link: {*Technology, Easy, Credit, ...*}. And, finally, steps *(c), (d), and (e)* identify the siblings of the link node, Figure 3 (b), and collect the remaining contextual information. We proceed to describe context ranking.

2. **Context Ranking:**

   After the context identification phase, the extracted words are ranked according to the following scheme. We choose the weight W_{link} of the text, contained inside the selected link, to be greater than the weight W_{other} of the text near the link, i.e. text contained in the siblings nodes. Through a set of experiments, we have chosen W_{link} to be twice as heavy as W_{other}, and this works well in practice. The following are the steps executed in this phase:

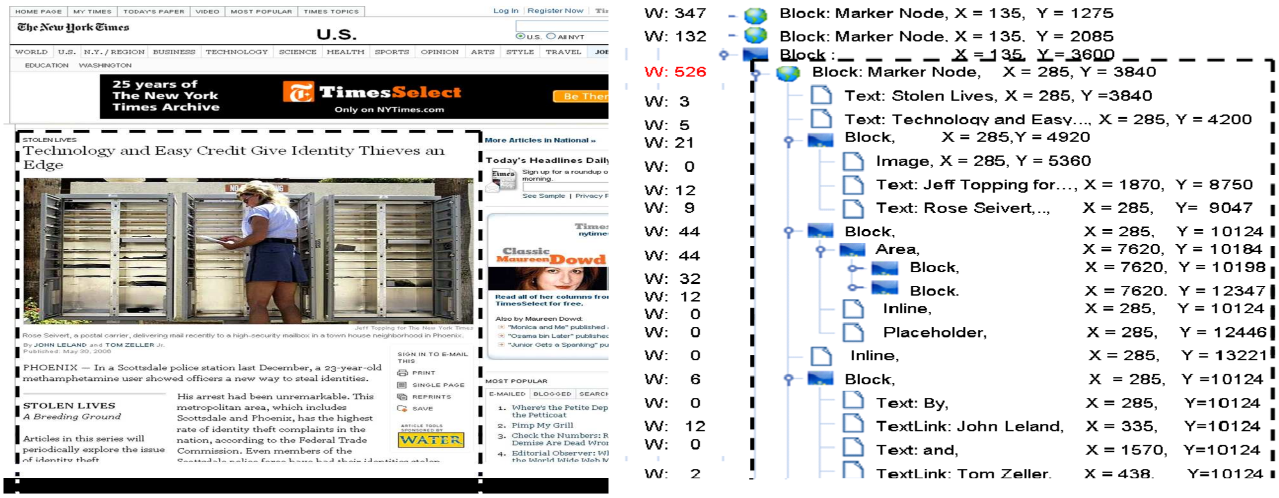
   a. Assign weight W_{link} to each of the words w_1, ..., w_n extracted from the text inside the link.

   b. Assign weight W_{other} to each of the words $w_{1'}$, ..., $w_{n'}$ other than the ones extracted from link.

3. **Context Matching:**

   After the context of the source page has been gathered and ranked, the Browser Object module downloads the destination Web page and generates a new frame tree, which is then searched for the most relevant block of information with respect to the context. We use the term *block* to refer to any non-leaf node in a frame tree, as well as to the corresponding section of the Web page. Context matching is performed using the steps enumerated below:

[7] *Function Words* or grammatical words are words that have little lexical meaning or have ambiguous meaning, but instead serve to express grammatical relationships with other words within a sentence, or specify the attitude or mood of the speaker. (Wikipedia.org)



(a) Destination Page (b) Destination Frame Tree

Figure 4: Most Relevant Block Identification

a. Run a depth first search to identify all leaves $\{Leaf_1, ..., Leaf_n\}$ in the frame tree of the destination page.

b. For each $Leaf_i$:

   i. Set its weight W_i to 0.

   ii. If $Leaf_i$ contains text, tokenize and store the text in a corresponding $List_i$, after removing the function words.

   iii. Perform a search of the context keywords from the multiset in $List_i$.

   iv. For each successful keyword match increment W_i by the weight of the keyword.

Continuing our example in Figure 4, the frame tree of the destination page will be searched for all words occurring in the context of of the link from the source page. By the end of this step, the leaf nodes, marked with clear-box icons in Figure 4 (b), will have accumulated their weights, e.g.: the leaf node containing text "Stolen Lives" has a weight of 3. The weights $W : n$ express the relevance of each leaf-node with respect to the context gathered from the source Web page. Higher weight implies greater relevancy

4. Block Ranking:

To reorganize the contents of the page in a meaningful way, we will use the concept of a *Marker Node* to identify the level of the frame tree, at which the tree has to be rearranged. For example, in Figure 4 (b), the dotted line encloses the subtree, rooted at the marker node, that will be moved within the tree.

Marker Node: We define a *Marker Node* as a node of a block tree, whose children are organized contiguously on the Web page and have the same geometric alignment, i.e. have matching X or Y coordinates. On the path from a leaf to the root there is one and only one marker node.

Web pages are usually organized in such a way that semantically related information is grouped together. Thus, the geometric alignment can help identify blocks containing semantically related information. For example, in Figure 3 (a), all Web page objects in the area surrounded by the dotted line have the same geometrical alignment and are otherwise related. The corresponding node will be identified as a marker node in frame tree. In Figure 3 (b) marker nodes are denoted by circular icons. We omit the details of the algorithm for marker node identification. The following steps describe the Block Ranking procedure:

a. Starting with one level above the leaf nodes, propagate the weights of the leaves up in the frame tree.

b. For a block node m, having n children, calculate its weight:

$$W_m = \sum_{i=1}^{n}(n - i + 1) * W_{child_i}$$

where $i = 1, ..., n$, and W_{child_i} denotes the weight of the ith child of the block node m.

c. Do not propagate the weights beyond the marker nodes.

d. Store all marker nodes in a list for further processing.

We observed that the first node in any block is usually more important than the subsequent nodes. We use this observation in multiplying the weight W_{child_i} of each child node in a block m by $(n - i + 1)$. Thus, with each new child node, we reduce its contribution to the total weight of the block m.

Figure 4 (b) shows a frame tree of the destination page with the weights that have been propagated up to the marker nodes.

5. Block Rearrangement:

At the final step of the context processing algorithm, the frame tree is reorganized based on the weights of the marker nodes:

a. Identify the maximum weighted node m in the list of the marker nodes.

b. Move the node m to the top of the frame tree, making it the first child of the root node.

The marker node, expanded in Figure 4 (b), happens to have the most weight, which makes it the most relevant block. And, as such, it will be moved to the top of the frame tree. The section of the Web page, corresponding to the selected marker node, is shown in Figure 4 (a). The frame tree is rearranged in such a way, that the Interface Manager will first read the most relevant block, which, in our case, is the article about the identity theft. Having finished with the article, the Interface Manager will continue reading the rest of the Web page content.

6.4 Future of Context Browsing

While the techniques for context-enabled browsing described in this section already show the promise of our approach, more work has to be done to improve context identification. We have identified several potentially useful areas for further research.

Incorporating NLP techniques. Natural language processing (NLP) techniques can be employed to enhance context processing and searching. For example, context ranking can give more weight to nouns than adjectives; and context matching and searching can benefit from approximate string matching (e.g. edit distance). More advanced NLP algorithms, such as summarization, semantic role labeling (e.g. [22]) etc., can further improve Web browsing experience. We are currently investigating summarization techniques to further save the browsing time.

Building a Statistical Model. We are also investigating the feasibility of applying machine learning algorithms to context identification and ranking. It may be possible to formulate statistical models (e.g Maximum Entropy Markov Model [43], Conditional Random Field [39], etc.) to describe context information. Then, a fixed point algorithm (e.g Expectation Maximization as outlined in [16]) may be used to identify and rank the context.

Smart Searching. Context processing algorithm can be easily extended to allow smart searching within a Web page. Search keywords represent the context, thus, eliminating the context identification step of the algorithm. Context ranking can assign different weights to the search keywords based on their position in the search string. Context matching will identify the most relevant sections of the page with respect to the search keywords. Subsequently, the remaining steps of the context processing algorithm will yield search results that can be immediately read to the user.

7 Planned Evaluation

All research work discussed in this report is a part of a HearSay project. The evaluation of sub-projects will be performed on the entire HearSay system. The research goals of the HearSay project have been formulated through collaboration with Helen Keller School for the Blind (HKSB) at Hempstead, NY. The design of the project is guided by individuals with visual disabilities and instructors at HKSB. The ideas have been obtained and clarified in meetings with instructors and students of the school. As the system takes shape, a series of progressive evaluations will be performed first by the sighted users and, then, by students of the Helen Keller School.

As soon as the HearSay infrastructure is ready to support summarization, several summarizers will be attached and a comparative evaluation will be performed to select the best summarizers. As the content of any given Web page is not homogeneous, and no one summarizer will deliver the optimal performance on all types of Web page content, it is important to determine, which method of summarization will suit best for a certain type of Web-page content.

After the dialog generation system is extended to support new layers of dialogs with integrated summarization and labeling facilities, the HearSay system will be ready for evaluation by the end-users. The system will be assessed for its usability and convenience. Following the evaluation, corrective steps will be taken to improve the system per request of the users.

8 Conclusion

This report reviews related research and presents preliminary work on improving dialog strategies for audio browsing. The research is done in the framework of the HearSay [63] project for developing a state-of-the-art non-visual Web browser. Several related projects on dialog generation, dialog processing, content summarization, and context-browsing were presented.

The goal of the dialog generation project is to provide better ways of presenting Web page content to blind people by means of audio and by bringing NLP techniques into the field of Web accessibility. A multi-layer dialog-based interaction has the potential to make the Web more friendly and accessible for people with visual disabilities. The system can be also adapted to provide Web access by phone.

To process the interactive dialogs, a flexible multi-platform VoiceXML interpreter, VXMLSurfer, is being developed. The project will provide the community of researchers with an extensible open-source tool for VoiceXML dialog exploration geared toward non-visual Web browsing. This work has the potential to have a positive effect on the W3C's [76] VoiceXML 3.0 recommendations.

Text summarization is identified to be important for effective dialog generation to facilitate voice browsing. This report reviews the current state of the text summarization field and proposes to research partial Web-document summarization to help condense non-uniform Web contents and provide effective labeling.

To further accelerate Web access for blind people, a context-directed non-visual browsing technique is introduced to help the blind users quickly identify relevant information on a Web page. The following are only a few potentially useful areas for further context-directed research. If search keywords are treated as context, the algorithm can be easily extended to allow smart searching within a Web page. NLP and statistical techniques can be improved to further enhance context processing and searching.

The progress of the Hearsay project is guided by the students and instructors of the Helen Keller School for the Blind at Hempstead, NY. A series of progressive demo sessions and evaluations are planned to ensure that the system meets the needs of the community of visually disabled people.

Acknowledgments

- I want to thank Dr. I.V. Ramakrishnan and Dr. Amanda Stent for their support and encouragement in the course of my work.
- I want to express my appreciation to my fellow colleagues Jalal Mahmud, Dipanjan Das, and Shrinand Javadekar for their input to this report.
- I want to express gratitude to my parents for their patience, and for having to deal with me during my examinational ordeals.
- And, finally, I want to extend my appreciation to anyone who reads this paper to the end.

A Shortcuts

VoiceXML Interpreter Shortcuts		
Combination	*Commands*	*Action*
Alt+Enter	-	Give a verbal command
Ctrl+Pause	exit, quit	Exits the interpreter
Insert+F1	help	Runs help handler
Ctrl+Alt+V	version	Tells the version number
Insert+F12	time	Tell the current time
Ctrl+Insert+NumPad/	append	Appends to clipboard
Ctrl+Insert+NumPad*	copy	Copies to clipboard
Ctrl+NumPad/	clip	Reads the contents of the clipboard
Alt+NumPad-5	repeat	Restart the audio or current utterance
Pause	pause, resume	Pauses and resumes a dialog
Ctrl+P	skip	Skips an utterance
Ctrl+BackQuote	-	Changes voice
Ctrl+1	-	Decrease rate of speech
Ctrl+2	-	Increase rate of speech
Ctrl+3	-	Decrease pitch of speech
Ctrl+4	-	Increase pitch of speech
Ctrl+5	-	Decrease volume of speech
Ctrl+6	-	Increase volume of speech
HearSay-Specific Shortcuts		
Combination	*Commands*	*Action*
Ctrl+Enter	follow	Follow link
Ctrl+N	new, start	Enter new address
Insert+A	address, link	Read address of current Web page
Insert+T	title	Read the title of the current page
Insert+Escape	refresh, reload	Refresh page

Table 1: Shortcut Keys

References

[1] http://www.w3.org/WAI.

[2] A. V. Aho, S.-F. Chang, K. McKeown, D. R. Radev, J. R. Smith, and K. A. Zaman. Columbia digital news project: An environment for briefing and search over multimedia information. *Int. J. on Digital Libraries*, 1(4):377–385, 1997.

[3] C. Asakawa and T. Itoh. User interface of a home page reader. In *ACM Intl. Conf. on Assistive Technologies (ASSETS)*, 1998.

[4] http://cafe.bevocal.com.

[5] B. Boguraev, R. Bellamy, and C. Swart. Summarization Miniaturization: Delivery of News to Hand-Helds. pages 99–110, 2001.

[6] B. Boguraev and M. S. Neff. Discourse segmentation in aid of document summarization. In *HICSS*, 2000.

[7] R. Brandow, K. Mitze, and L. F. Rau. Automatic Condensation of Electronic Publications by Sentence Selection. *Information Processing and Management*, 31(5):675–685, 1995.

[8] J. Brewer, D. Dardailler, and G. Vanderheiden. Toolkit for promoting web accessibility. In *Proceedings of Tech. and Persons with Disabilities Conf.*, 1998.

[9] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Seeing the whole in parts: text summarization for web browsing on handheld devices. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 652–662, New York, NY, USA, 2001. ACM Press.

[10] C. Cardie, V. Ng, D. Pierce, and C. Buckley. Examining the role of statistical and linguistic knowledge sources in a general-knowledge question-answering. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000)*, pages 180–187, 2000.

[11] A. Chieko and C. Lewis. Home page reader: Ibms talking web browser.

[12] W. T. Chuang and J. Yang. Extracting sentence segments for text summarization: a machine learning approach. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 152–159, New York, NY, USA, 2000. ACM Press.

[13] C. Y. Chung, M. Gertz, and N. Sundaresan. Reverse engineering for web data: From visual to semantic structures. In *Intl. Conf. on Data Engineering (ICDE)*, 2002.

[14] http://classifier4j.sourceforge.net.

[15] S. Debnath and C. L. Giles. A learning based model for headline extraction of news articles to find explanatory sentences for events. In *K-CAP '05: Proceedings of the 3rd international conference on Knowledge capture*, pages 189–190, New York, NY, USA, 2005. ACM Press.

[16] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(B):1–38, 1977.

[17] D. Dillin. Concept of a structural html page browser to support access to www for people with disabilities. *Technical Report, Yuri Rubinsky Insight Foundation*, 1997.

[18] C. Earl and J. Leventhal. A survey of windows screen reader users: Recent improvements in accessibility. In *Journal of Visual Impairment and Blindness*, 1999.

[19] D. Embley and L. Xu. Record location and reconfiguration in unstructured multiple-record web documents.

[20] D. W. Embley, Y. Jiang, and Y.-K. Ng. Record-boundary discovery in web documents. In *ACM Conf. on Management of Data (SIGMOD)*, 1999.

[21] `http://freetts.sourceforge.net`.

[22] D. Gildea and D. Jurafsky. Automatic labeling of semantic roles. volume 28, 2002.

[23] J. Gunderson and R. Mendelson. Usability of world wide web browsers by persons with visual impairments. In *Proceedings of the RESNA Annual Conf.*, 1997.

[24] D. Hadjadj and D. Burger. Braillesurf: An html browser for visually handicapped people. In *Proceedings of Tech. and Persons with Disabilities Conf.*, 1999.

[25] S. Hamerich, V. Schubert, V. Schless, R. Crdoba, J. Pardo, L. d'Haro, B. Kladis, O. Kocsis, and S. Igel. Semi-automatic generation of dialogue applications in the gemini project, 2004.

[26] S. Harper, C. Goble, R. Stevens, and Y. Yesilada. Middleware to expand context and preview in hypertext. In *Assets '04: Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility*, 2004.

[27] P. Hendrix and M. Birkmire. Adapting web browsers for accessibility. In *Proceedings of Tech. and Persons with Disabilities Conf.*, 1998.

[28] M. Hori, G. Kondoh, K. Ono, S. ichi Hirose, and S. Singhal. Annotation-based web content transcoding. In *Intl. World Wide Web Conf. (WWW)*, 2000.

[29] A. Huang and N. Sundaresan. A semantic transcoding system to adapt web services for users with disabilities. In *ACM Intl. Conf. on Assistive Technologies (ASSETS)*, 2000.

[30] `http://www.freedomscientific.com`.

[31] P. Jones. Speech-enabled web access: an instructional case study. In *Proceedings of Tech. and Persons with Disabilities Conf.*, 1998.

[32] `http://jrex.mozdev.org`.

[33] `http://java.sun.com/products/java-media/speech`.

[34] `http://www.jsmart.com`.

[35] K. Komatani, F. Adachi, S. Ueno, T. Kawahara, and H. G. Okuno. Flexible spoken dialogue system based on user models and dynamic generation of voicexml scripts.

[36] W. Kraaij, M. Spitters, and A. Hulth. Headline extraction based on a combination of uni- and multi-document summarization techniques, 2002.

[37] W. Kraaij, M. Spitters, and M. van der Heijden. Combining a mixture language model and naive bayes for multi-document summarisation. In *Proceedings of the DUC2001 workshop (SIGIR2001), New Orleans*, 2001.

[38] M. Krell and D. Cubranic. V-lynx: Bringing the world wide web to sight impaired users. In *ACM Intl. Conf. on Assistive Technologies (ASSETS)*, 1996.

[39] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[40] H. Lieberman. Letizia: An agent that assists Web browsing. In *Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, 1995.

[41] L. Lloyd, D. Kechagias, and S. Skiena. Lydia: A system for large-scale news analysis. In *SPIRE*, pages 161–166, 2005.

[42] H. P. Luhn. The Automatic Creation of Literature Abstracts. *IBM Journal of Research Development*, 2(2):159–165, 1958.

[43] A. McCallum, D. Freitag, and F. C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *ICML*, pages 591–598, 2000.

[44] K. R. McKeown, R. Barzilay, D. Evans, V. Hatzivassiloglou, J. L. Klavans, A. Nenkova, C. Sable, B. Schiffman, and S. Sigelman. Tracking and summarizing news on a daily basis with Columbia's Newsblaster. In *Proceedings of the Human Language Technology Conference*, 2002.

[45] K. R. McKeown, R. Barzilay, D. Evans, V. Hatzivassiloglou, B. Schiffman, and S. Teufel. Columbia multi-document summarization: Approach and evaluation. In *Proceedings of the Workshop on Text Summarization, ACM SIGIR Conference 2001*. DARPA/NIST, Document Understanding Conference.

[46] K. R. McKeown, S.-F. Chang, J. Cimino, S. Feiner, C. Friedman, L. Gravano, and V. Hatzivassiloglou. PERSIVAL: A System for Personalized Search and Summarization Over Multimedia Healthcare Information. In *Proceedings of the 1st ACM IEEE-CS Joint Conference on Digital Libraries*, pages 331–340, Roanoke, VA, January 2001.

[47] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to wordnet: An on-line lexical database*. *Int J Lexicography*, 3(4):235–244, January 1990.

[48] `http://www.mozilla.com/firefox`.

[49] S. Mukherjee, I. Ramakrishnan, and M. Kifer. Semantic bookmarking for non-visual web access. In *ACM Conf. on Assistive Technologies (ASSETS)*, 2004.

[50] S. Mukherjee, I. Ramakrishnan, and A. Singh. Bootstrapping semantic annotation for content-rich html documents. In *Intl. Conf. on Data Engineering (ICDE)*, 2005.

[51] S. Mukherjee, G. Yang, and I. Ramakrishnan. Automatic annotation of content-rich html documents: Structural and semantic analysis. In *Intl. Semantic Web Conf. (ISWC)*, 2003.

[52] S. Mukherjee, G. Yang, W. Tan, and I. Ramakrishnan. Automatic discovery of semantic structures in html documents. In *Intl. Conf. on Document Analysis and Recognition*, 2003.

[53] S. Muresan, E. Tzoukermann, and J. L. Klavans. Combining linguistic and machine learning techniques for email summarization. In *ConLL '01: Proceedings of the 2001 workshop on Computational Natural Language Learning*, pages 1–8, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[54] K. Nagao, Y. Shirai, and K. Squire. Semantic annotation and transcoding: Making web content more accessible. *IEEE Multimedia*, 8(2), 2001.

[55] `http://www.internetspeech.com`.

[56] T. Oogane and C. Asakawa. An interactive method for accessing tables in html. In *ACM Intl. Conf. on Assistive Technologies (ASSETS)*, 1998.

[57] `http://www.optimtalk.cz`.

[58] `http://libots.sourceforge.net.nyud.net:8090`.

[59] E. Pontelli, W. Xiong, D. Gillian, E. Saad, G. Gupta, and A. Karshmer. Navigation of html tables, frames, and xml fragments. In *ACM Intl. Conf. on Assistive Technologies (ASSETS)*, 2002.

[60] D. Radev and W. Fan. Automatic summarization of search engine hit lists, 2000.

[61] D. Radev, H. Jing, and M. Budzikowska. Centroid-based summarization of multiple documents: sentence extraction, 2000.

[62] D. R. Radev and K. R. McKeown. Generating natural language summaries from multiple on-line sources. *Comput. Linguist.*, 24(3):470–500, 1998.

[63] I. Ramakrishnan, A. Stent, and G. Yang. Hearsay: Enabling audio browsing on hypertext content. 2004.

[64] T. Raman. Audio system for technical readings. *PhD Thesis, Cornell University*, 1994.

[65] G. Salton, A. Singhal, C. Buckley, and M. Mitra. Automatic text decomposition using text segments and text themes. In *UK Conference on Hypertext*, pages 53–65, 1996.

[66] B. Schiffman, I. Mani, and K. J. Concepcion. Producing biographical summaries: Combining linguistic knowledge with corpus statistics. In *Meeting of the Association for Computational Linguistics*, pages 450–457, 2001.

[67] C. Schmandt. Audio hallway: A virtual accoustic environment for browsing. In *Proceedings of UIST*, 1998.

[68] M. M. Singhalt. Automatic text summarization by paragraph extraction.

[69] `http://cmusphinx.sourceforge.net`.

[70] Z. Sun. *Machine Learning in Interface Design for an Audio Browser*. PhD thesis, Stony Brook University, 2006.

[71] Z. Sun, J. Mahmud, S. Mukherjee, and I. V. Ramakrishnan. Model-directed web transactions under constrained modalities. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 447–456, New York, NY, USA, 2006. ACM Press.

[72] H. Takagi and C. Asakawa. Transcoding proxy for nonvisual web access. In *ACM Intl. Conf. on Assistive Technologies (ASSETS)*, 2000.

[73] H. Takagi, C. Asakawa, K. Fukuda, and J. Maeda. Site-wide annotation: Reconstructing existing pages to be accessible. In *ACM Intl. Conf. on Assistive Technologies (ASSETS)*, 2002.

[74] D. R. Timothy. Mead - a platform for multidocument multilingual text summarization.

[75] `http://www.voicexml.org`.

[76] `http://www.w3.org`.

[77] W. Wang and M. P. Harper. The superarv language model: investigating the effectiveness of tightly integrating multiple knowledge sources. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 238–247, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[78] G. Weber. Programming for usability in non-visual user interfaces. In *ACM Intl. Conf. on Assistive Technologies (ASSETS)*, 1998.

[79] H. Wu, D. R. Radev, and W. Fan. Toward answer-focused summarization using search engines. In *New Directions in Question Answering*, pages 227–236, 2004.

[80] Y. Yang and H. Zhang. HTML page analysis based on visual cues. In *Intl. Conf. on Document Analysis and Recognition (ICDAR)*, 2001.

[81] L. Yi and B. Liu. Eliminating noisy information in web pages for data mining. In *ACM Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.

[82] S. Yu, D. Cai, J.-R. Wen, and W.-Y. Ma. Improving pseudo-relevance feedback in web information retrieval using web page segnmentation. In *Intl. World Wide Web Conf. (WWW)*, 2003.

[83] M. Zajicek, C. Powell, and C. Reeves. Web search and orientation with brookestalk. In *Proceedings of Tech. and Persons with Disabilities Conf.*, 1999.