Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 235 (2009) 107–124
www.elsevier.com/locate/entcs

Bridging the Web Accessibility Divide

I.V. Ramakrishnan, Jalal Mahmud, Yevgen Borodin, Muhammad Asiful Islam, Faisal Ahmed [1]

Department of Computer Science
Stony Brook University
Stony Brook, NY, USA

Abstract

The Web has become the primary medium for accessing information and for conducting many types of online transactions, including shopping, paying bills, making travel plans, etc. The primary mode of interaction over the Web is via graphical browsers designed for visual navigation. Sighted users can visually segment web pages and quickly identify relevant information. On the contrary, screen readers - the dominant assistive technology used by visually impaired individuals - function by speaking out the screen's content serially. Consequently, users with visual impairments are forced to listen to the information in web pages sequentially, thereby experiencing considerable information overload. This problem becomes even more prominent when conducting online transactions that often involve a number of steps spanning several pages. Thus, there is a large gap in Web accessibility between individuals with visual impairments and their sighted counterparts. This paper we describe our ongoing work on this problem. We have developed several techniques that synergistically couple web content analysis, user's browsing context, process modeling and machine learning to bridge this divide. These techniques include: 1) context-directed browsing that uses link context to find relevant information as users move from page to page; 2) change detection that separates the interface from the implementation of web pages and helps users find relevant information in changing web content; and 3) process modeling that helps users find concepts relevant in web transactions. We describe these three techniques within the context of our Hearsay non-visual web browser.

Keywords: Context, Web Accessibility, Screen Reader, Non-Visual, HearSay, Partitioning, Semantic Blocks, Web transaction, Content Adaption, Dynamic Content.

1 Introduction

The Web has become an indispensable source of information and we use it more and more in our daily activities. The primary mode of interaction with the Web is via graphical browsers, which are designed for visual interaction. However, most Web pages contain banners, ads, navigation bars, and other data distracting us from the information. As we browse the Web, we have to filter through a lot of irrelevant data. This is straightforward for sighted individuals, who can process visual data in no time at all and can quickly locate the information that is most relevant to them.

[1] Email: {ram,jmahmud,borodin,maislam,faiahmed}@cs.sunysb.edu

1571-0661/Published by Elsevier B.V.
doi:10.1016/j.entcs.2009.03.007

However, this task can be time-consuming and complicated for people with visual disabilities, who are typically forced to browse the Web with screen-readers [2,1]. Such screen-readers process Web pages sequentially, i.e. they first read through the menus, banners, commercials, or anything else that comes before the desired information. This makes browsing time consuming and strenuous.

In addition, Web applications facilitated by technologies such as JavaScript, DHTML, AJAX, and Flash use a considerable amount of dynamic web content that is either inaccessible or unusable by blind people. Server-side changes to web content cause whole page refreshes, but only small sections of the page update, causing blind web users to search linearly through the page to find new content.

Web accessibility problem is even more prominent in web transactions(e.g.: shopping, registrations, bill payments, etc.) that typically span multiple pages. Sighted users can quickly identify the relevant content and perform appropriate actions (e.g.: add to cart, continue shopping, checkout, etc.) to conduct a Web transaction. However, locating relevant content, buttons, and links, which are required to make progress in online transaction, is very difficult for users with vision impairments.

The problem of finding relevant information while navigating from page to page, browsing dynamic web sites, or performing Web transactions begs the question: Is it possible to devise techniques that will help blind people find relevant information faster? To address these three browsing scenarios, we propose context-directed browsing, Dynamo, and model-directed transactions approaches, which can potentially make web browsing more intuitive and user-friendly, in this way, bridging the web accessibility divide between visually impaired and sighted users.

Context-directed browsing. 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 and present it to the user first.

Dynamo. To unify different types of content changes and make dynamic content accessible to blind web users, we developed Dynamo, the system that treats web page updates uniformly. Its methods encompass both web updates enabled through dynamic content and scripting, and updates resulting from static page refreshes, form submissions, and template-based web sites.

Model-directed transactions. Non-visual web transactions can be facilitated by giving the users a way to quickly access the most relevant concepts at any step of the transaction. We used a process model to direct Web transactions. The model was represented by a finite state automaton where each state corresponded to a web page with concepts, and the edges were the actions leading to those states. When a transaction was in a particular state, the most relevant concepts would be identified, extracted, and presented to the user.

Context-directed browsing, Dynamo, and model-directed transactions are embodied in our Hearsay non-visual web browser. The HearSay browser brings together Content Analysis, Natural Language Processing (NLP), and Machine Learning algorithms to help blind users quickly identify relevant information while navigating between web pages, browsing dynamic web content, and doing web-

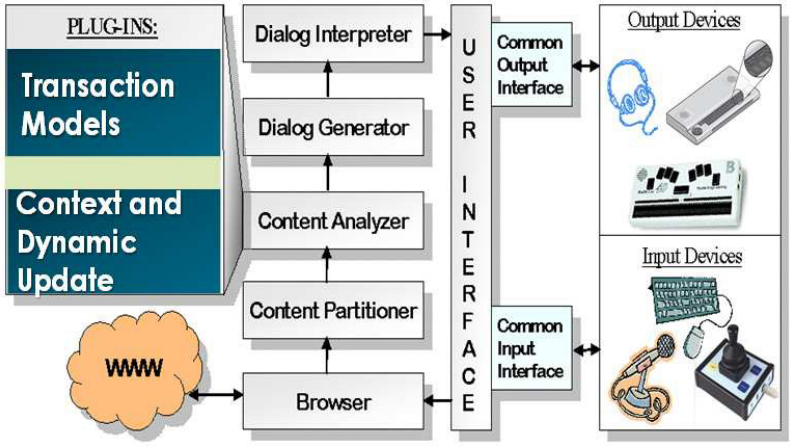

Fig. 1. System Architecture

transactions; thus, considerably reducing the browsing time and information overload for blind people.

The rest of this paper is organized as follows: In Section 2, we describe the architecture of the system. In Section 3 we describe our approaches to bridging the web accessibility divide. Related work appears in Section 4, followed by concluding remarks in Section 5. The details of experimental setup and user evaluations can be found in our papers [3,9,6].

2 System Architecture

The approaches presented in this paper are embodied in the HearSay Web browser. The architecture of the browser is shown in Figure 1.

The **Browser** module downloads Web content every time the user requests a new page to be retrieved. The browser parses the content of the web page into a DOM tree representation, adding the page layout information produced by the renderer. In its current implementation HearSay uses Firefox web browser, but, in theory, it can support a variety of different web browsers.

Content Partitioner uses the layout and alignment of the web page to segment it into distinct sections [3]. The algorithm uses the observation that semantically related items share similar geometric alignment and exhibit spatial locality.

Content Analyzer interacts with various plug-ins to identify relevant content during browsing, dynamic changes and web-transactions. Details of these plug-ins are described in Section 3.

The **Dialog Generator** module generates VoiceXML dialogs for page navigation. A separate functionality is provided through **Dialog Interpreter** to quickly switch between the segments with concept instances. The dialogs are then delivered to the User Interface Manager, and, once the user chooses an action, the process is repeated.

Users communicate with the system through the **User Interface**, which uses VXMLSurfer [4] to interpret interactive VoiceXML dialogs. The Interface Manager

enables basic screen-reading navigation and provides additional controls to access the concepts identified by the system.

3 Approaches to Bridging the Divide

3.1 Context-driven Web Browsing

Here we present our algorithms for context-driven Web browsing. The two main algorithms are *Context Identification* and *Relevant Block Identification*. Both of these algorithms utilize a *Geometrical Clustering* algorithm to partition Web pages into segments, containing semantically related content.

To collect the context, a topic-detection algorithm is applied to the information surrounding the followed link. We gather the text that shares a common topic with the link, and use this context to identify the relevant information on the destination Web page. Then, a support vector machine is used to compute the relevance score of these sections with respect to the context. Subsequently, the Web page is presented to the user starting with the highest ranking section. If the relevant section was not identified correctly, the user can always skip to the beginning of the page. The following subsections discuss the algorithms in detail.

3.1.1 Geometric Clustering

Instead of implementing its own HTML parser, HearSay utilizes the DOM tree data structure created by the Browser module. While rendering a page on the screen, the Browser creates a tree-like structure of nested *nodes* that holds the content of the Web page: the leaf nodes of the tree contain the smallest individual elements, e.g. a link, an image, etc.; non-leaf nodes "enclose" one or more leaf nodes and/or other non-leaf nodes; and the root node "contains" the entire page. Figure 2(b) shows an example of graphical representation of the DOM tree.

We use an observation that semantically related information exhibits spatial locality and often shares the same alignment on a Web page. Since a DOM tree represents the layout of a Web page, we infer that geometrical alignment of nodes may imply semantic relationship between their respective content. If all descendants of a node are consistently aligned either along X or Y axes, we call such a node **consistent**.

A *Maximal Semantic* Block, or simply **block**, is the largest of the consistent nodes on the path from a leaf to the root of a DOM tree. Thus, it is likely to be the largest possible cluster containing semantically related items of information. For example, Figure 2(a) shows how the alignment information is used to cluster the New York Times Web page into maximal semantic blocks: banner labeled as 1, search - 2, taxonomy - 3, and news - 4.

The algorithm to find blocks is described in [3]. The maximal semantic blocks are further used by the Context Identification and Relevant Block Identification algorithms.

Fig. 2. Context Identification

3.1.2 Context Identification

Once the Geometric Clustering algorithm has segmented the Web page into maximal semantic blocks, and the user selected a link to be followed, the Context Identification algorithm collects the context of the link. We formally define the notion of context as:

Context of a link is the content around the link that maintain the same *topic* as the link.

Consider Figure 2, showing the front page of The New York Times Web site and the corresponding DOM tree. The context of the link, indicated by an arrow, is the text surrounded by the dotted line. Notice how the topic changes from one headline to another.

A block, produced by the Geometric Clustering algorithm, ideally represents a segment of text on the same subject, but may have several topics within it. Therefore, we limit topic boundary detection and context collection to the block containing the link. Context collection begins from the link and expands around the link until the topic of the text changes. A simple cosine similarity technique is used to detect the boundaries of the topic, see equation 1.

In the NYTimes example, Figure 2(a), the user follows the link "Top General Warns Against Iraq Timetable", indicated by the mouse pointer. We initialize the multiset with the text collected from the link node of the DOM tree, indicated by a mouse pointer in Figure 2(b), as well as from the non-leaf sibling which follows the link node. The multiset now contains single words (e.g. "general", "david", "stout", "gen" "john", etc.), their bigrams (e.g. "david stout", "gen john"), and trigrams (e.g. "gen john abizaid").

After the initialization stage, we collect the context of the link, starting from the parent node of the link node, by expanding the context to include the node's siblings. Again, in our example, the parent node of the link in the DOM tree is the node labeled as "a" in Figure 2(b). We start with the sibling "b", construct multiset *SText* from the sibling's text, and compare its content to the content of the

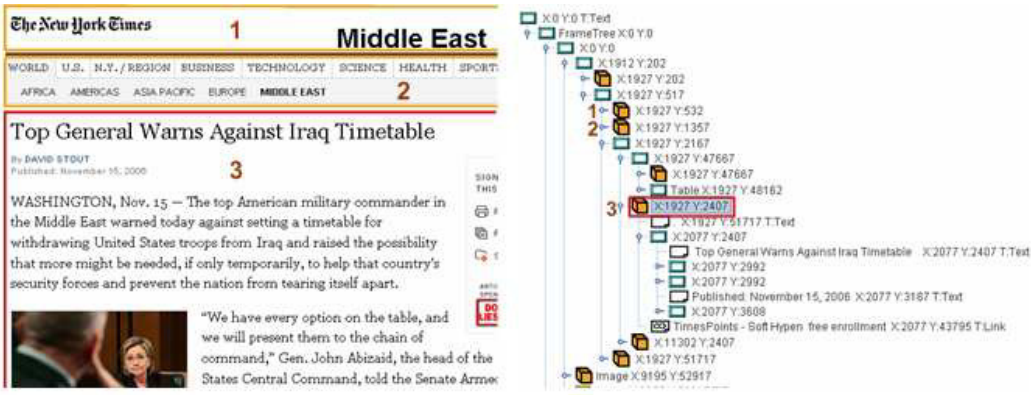

Fig. 3. Most Relevant Block Identification

Context multiset. The comparison is done using cosine similarity of the multisets. More formally, for any two multisets M1 and M2, their cosine similarity is defined as:

$$\cos(M_1, M_2) = \frac{M_1 \cap M_2}{\sqrt{M_1}\sqrt{M_2}} \quad (1)$$

We consider two multisets to be similar if their cosine similarity is above a threshold. We have statistically computed the threshold (See [3] for details) that best determines whether a topic changes between the *Context* and the *SText* multisets. If the cosine similarity between the multisets is above the threshold, i.e. topic boundary is not detected, the multi-sets are merged. Otherwise, we stop expanding the context window in that direction. The details appear in [3].

3.1.3 Relevant Block Identification

After the context of the link has been gathered on the source page, the Browser Object module downloads the destination Web page and generates a new DOM tree. Again, we use our Geometric Clustering algorithm to segment the page into maximal semantic blocks: 1, 2, 3 in Figure 3. Then, the Relevant Block Identification algorithm matches the context against every block in the DOM tree and computes the relevance of each block with respect to the collected context.

We define "block ranking" as a learning problem and use a support vector machine (SVM) [5,7] to learn a block relevance model. We define two classes for our block relevance model: relevant and not relevant, and describe each block of the destination page with a set of feature values, which we compute by trying to match single words, bigrams, trigrams, and their stemmed counterparts, contained in context, to the text in the blocks.

The machine-learned SVM model is now used to predict the relevance of a given block with respect to some context. Given a set of blocks $B_1, B_2, ..., B_m$ we compute the feature values for each block by matching the context against the text in the block. Next, we use the learned SVM model to label the blocks as either relevant or

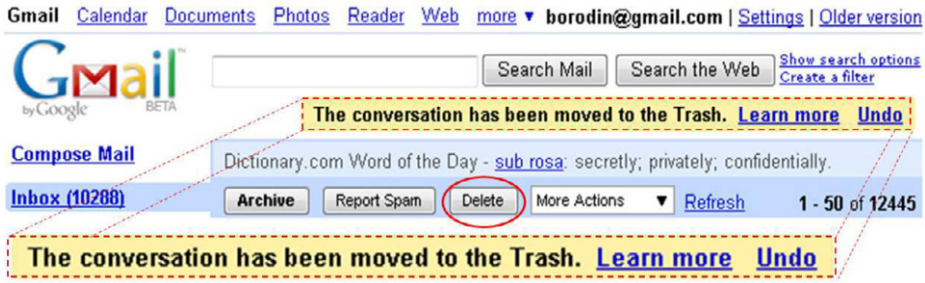

Fig. 4. Dynamically appearing "Undo" link at Gmail.com

not-relevant, and get the associated probability values. Then, we pick the highest ranking block, in terms of the probability values, as the most relevant one. Block 3, expanded in Figure 3(b), was chosen by the SVM as the most probable candidate for contextual relevancy. Subsequently, the HearSay will read the page starting from section 3 of the Web page in Figure 3(a).

The next subsection describes our solution for identifying relevant information by keeping track of the changes in web pages.

3.2 Handling Web Content Updates

3.2.1 Use Cases

In this section we discuss the scenarios that motivate Dynamo. Figure 4 shows a dynamic message (dashed box) appearing on the Gmail.com web page as soon as one deletes an email. While there are other ways to recover an email, it would be considerably easier for users to simply jump to the update message and click the "Undo" link. Using current screen readers, users are not notified of this change and do not have access to the updated content.

Figure 5 illustrates an automatic page refresh that causes changes to 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. Not only should they then be able to review the newly added content (solid box Figure 5(b)), but also 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 5(a)), instead of starting from the beginning after the refresh, the cursor should be repositioned to the same article (dashed box Figure 5(b)). And it should not matter to the users whether the content changed from page reload or an AJAX update, such as those happening at Gmail.com when a new email arrives. In either case, users should be able to choose to continue reading from where they left off.

3.2.2 Content Updates: A Unified View

We propose a unified framework for handling all types of dynamic content and describe an integrated interface that can help users find changes in any web page, and, with them, find relevant information more efficiently.

Fig. 5. Automatically refreshing front page at L.A. Times

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.

Again 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. If we imagine that a web browser interface (with its menus, address bar, buttons) is just a template that is common for all websites, then we can think of static web sites as web page updates appearing within this template. Then, any content change can be considered "dynamic".

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

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. We next describe an implementation of this idea built in the HearSay web browser.

3.2.3 Hearsay Dynamo

We now describe Hearsay-Dynamo (HD) embodying Dynamo within the framework of the Hearsay non-Visual Web Browser.

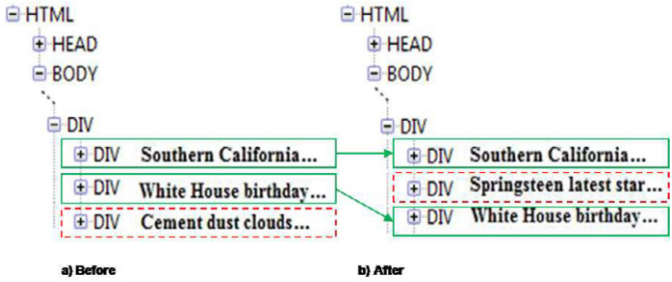

Fig. 6. Dom snapshots of L.A. Times refreshing front page

HD is built on the existing Hearsay non-visual web browser. HearSay uses a Firefox-extension to communicate with a Java backend. The HD backend processes all events sent to it by the Firefox extension, pushes new web content through the pipeline of various analytic algorithms, generates VoiceXML dialogs, and manages the user interface through the vxmlSurfer - a custom VoiceXML interpreter.

All dynamic updates in any given page can, in principle, be implemented using AJAX, as well as a whole web site, or even the entire Web! 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.

We implemented a custom HTML DOM Diff algorithm to verify if the content has changed, and then group the changes in packets used to update the dialogs. The algorithm starts by creating maps of the two DOMs under consideration. The maps have node signatures as keys and the corresponding nodes as values. A node signature is composed of the "/NodeName/NodeType" combinations of all nodes on the path from the root of the tree to the node under consideration, where NodeType is 1 for Element, 2 for Attribute, and 3 for #Text. For example, "/html/1/body/1/div/1/div/1/#text/3" is the signature of the #text node with the "Southern California..." under the DIV node in Figure 6, which illustrates the two HTML DOM Trees of the L.A. Times web page before and after refresh in Figure 5.

Our Diff Algorithm combines the maps and performs a bipartite matching to find out the matching nodes and makes a list of them. Later we use the max flow routine to find the maximum number of matches. And the rest of them are considered as different nodes.

HD then uses the content in different nodes list to remove from or add to the VoiceXML dialogs and updated groups the content by proximity. When a VoiceXML dialog is updated, the changed content is grouped and flagged as "new". Only the newly added or changed content is accessible to users; deleted content is removed from the dialogs transparently. At any time, users can access the updated content by pressing shortcuts that jump to the next or previous group of updates (see [9] for details).

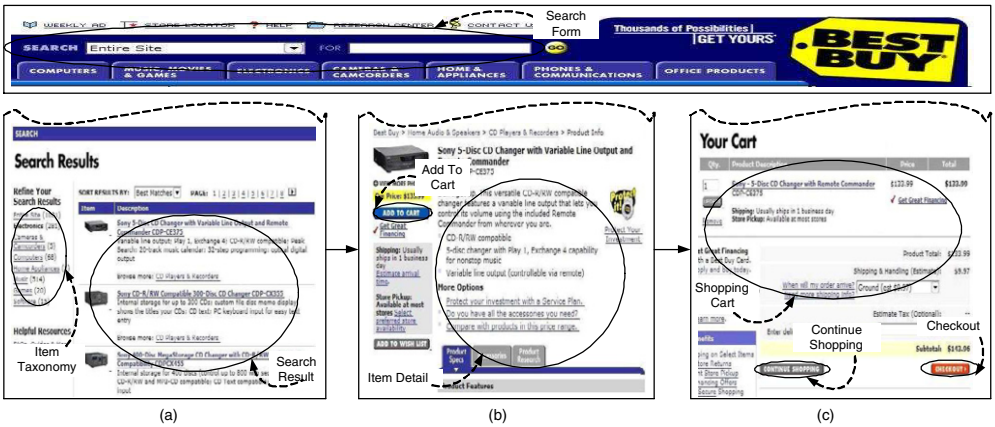

Fig. 7. A Web Transaction Example

3.3 Model-directed Web Transaction

Now that we have identified the relevant information in Ad-hoc web browsing, let us focus our attention to web transactions.

We capture the two aspects of a transaction, namely its operation sequence and content identification by a process model and an ontology respectively. The ontology describes the set of semantic concepts occurring in Web pages, which are considered essential for conducting Web transactions in a particular domain. The circled elements in Figure 7 are examples of such concepts. The process model is a deterministic finite automata (DFA) that captures the set of transactional sequences. Each state, representing an atomic operation in a transaction, is associated with a set of semantic concepts drawn from the ontology. When the model makes a transition to a state during the course of a transaction, a Web page is provided to the state as an input. If the concepts associated with the state are present in the page, then they alone are identified and presented to the user. For instance, if the page shown in Figure 7(a) is given as the input to a state associated with the concepts "Item Taxonomy" and "Search Result" only the two circled items in the figure will be identified and presented to the user. Since transactions are essentially interactive, we associate each concept with a user operation, e.g., the submit searchform operation with the "Search Form" concept. Each such operation results in a state transition and a sequence of operations constitutes a Web transaction. Thus coupling content semantics with model-directed navigation can overcome the information overload problem by delivering relevant content at every step of the transaction.

3.3.1 Process Model

Formal definition of process model is given in [6]. Figure 8 illustrates a process model. The concepts associated with the starting state q_1 are "Item Taxonomy", "Item List", and "Search Form". This means that if these concept instances are present in the Web page given to q_1 as its input, they will be extracted and presented to the user. User can select any of these concepts. When the user selects the "Search

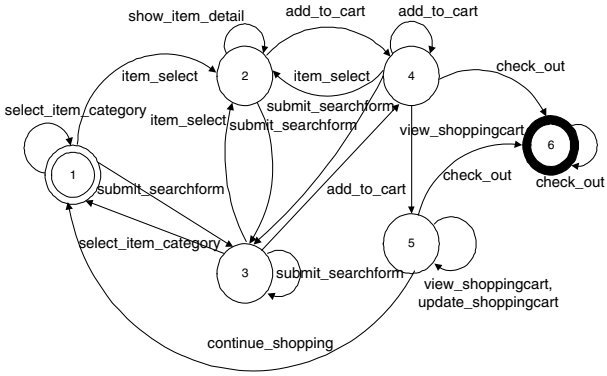

Fig. 8. Process Model Example

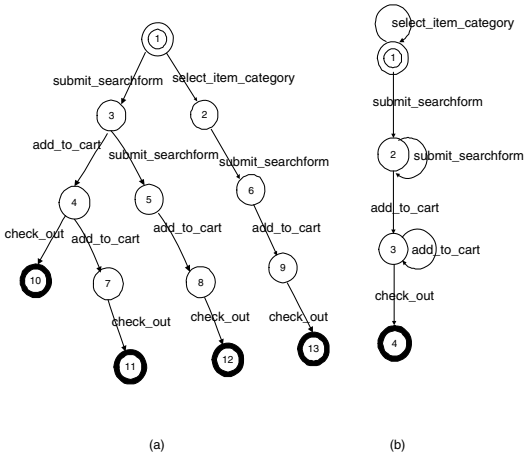

Fig. 9. (a) A Prefix Automata (b) Learned DFA

Form" concept he is required to supply the form input upon which the submit searchform operation is invoked. This amounts to submitting the form with the user-supplied form input. A Web page consisting of the search results is generated and a transition is made to q_3. q_6 represents the final state which is entered upon a check out operation.

3.3.2 Process Model Learning

We built the process model using DFA learning techniques, a thoroughly researched topic. In the DFA learning problem the training set consists of two sets of example strings, one labeled positive and the other negative. Only strings in the positive set are in the DFA's language. The objective is to construct a DFA that is *consistent* with respect to these two sets, i.e., it should accept strings in the positive set while rejecting those in the negative set. We adapted the heuristic in [12] for learning our process model, the choice being mainly dictated by its simplicity and low complexity.

The training sequences we used for learning the process model consist of strings whose elements are operations on concepts. The sequence $< submitsearchform, itemselect, addtocart, checkout >$ is one such example.

These training sequences are (manually) labeled "completed" and "not completed". The positive example set (S+) consists of sequences labeled "completed" while the negative example set (S-) consists of those labeled "not completed".

We first construct a prefix tree automata as shown in Figure 9(a) using only the examples in S+. In Figure 9(a), the sequence of operations along each root-to-leaf path constitutes a string in S+. For this example S_i consists of the strings: $\{< checkout >, < submitsearchform, addtocart >, < submitsearchform, checkout >, < selectitemcategory, addtocart, checkout >\}$. The prefix of every string in S+ is associated with a unique state in the prefix tree. The prefixes are ordered and each state in the prefix tree automata is numbered by the position of its corresponding prefix string in this lexicographic order. Next we generalize the prefix tree automata by state merging. We choose state pairs $(i, j), i < j$ as candidates for merging. The candidate pair (i, j) is merged if it results in a consistent automata. For example, merging the pair (1,2) is consistent whereas (3,4) is not merged as the resulting automata will accept the string $< submitsearchform, checkout >$ in S_i. The DFA that results upon termination of this merging process on the above example set is shown in Figure 9(b).

3.3.3 Content Analysis

Herein we describe the content analyzer module that extracts the relevant concepts. In a nutshell this is achieved by partitioning a Web page into segments of semantically related items and classifying them against concepts in the ontology. Below we provide an overview.

The Approach

It is based on our previous work on learning-based semantic analysis of Web content [8]. Briefly, the technique rests on three key steps: (i) inferring the logical structure of a Web page via structural analysis of its content, (ii) learning statistical models of semantic concepts using light-weight features extracted from both the content as well as its logical structure in a set of training Web pages, and (iii) applying these models on the logical structures of new Web pages to automatically identify concept instances.

- Structural Analysis: Structural analysis (see [8] for details) is based upon the observation that semantically related items in content-rich Web pages exhibit consistency in presentation style and spatial locality. Exploiting this observation, a pattern mining algorithm working bottom-up on the DOM tree of a Web page aggregates related content in sub-trees. Briefly, the algorithm initially assigns types, reflecting similarities in structural presentation, to leaf nodes in the DOM tree and subsequently restructures the tree bottom-up using pattern mining on type sequences. In the restructured tree, known also as the partition tree, there are three classes of internal nodes: (i) group - which encapsulates repeating patterns in its immediate children type sequence, (ii) pattern - which captures each individual occurrence of the repeat, or (iii) block - when it is neither group nor pattern. Intuitively the subtree of a group node denotes homogenous content consisting of semantically related items. The partition tree represents the logical organization of

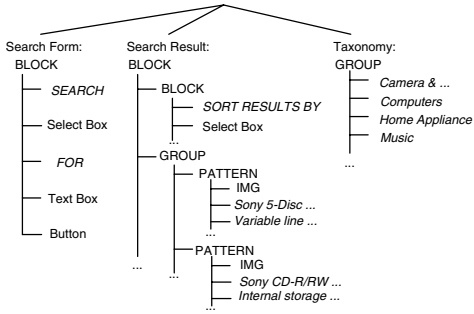

Fig. 10. Extracted Concept Tree

the page's content.

- Feature Extraction: The statistical concept models are based upon features extracted from the content of the partition trees. Given a partition tree node $p, n_{fi,p}$ denotes the frequency of occurrence of feature f_i in p. We use three different types of features in the analysis: Word features are features drawn from the text encapsulated within a partition tree node, p-gram features are features representing the visual presentation of content, and t-gram features represent the structure of the partition tree.

- Concept Identification: A concept model consists of two components: (i) a probability distribution on the frequency of occurrence of the word, p-gram, and t-gram features, and (ii) a probability distribution on the number of nodes present in the entire subtree of a partition tree node. A collection of partition trees whose nodes are (manually) labeled as concept instances serve as the training set for learning the parameters of these distributions.

A maximum likelihood approach is used to model the distribution of a feature in a concept. Given a training set of L partition tree nodes identified as instances of concept c_j , the probability of occurrence of a feature f_i in c_j is defined using Laplace smoothing as:

$$P(f_i|c_j) = \frac{\sum_{p \in L} n_{f_i,p} + 1}{\sum_{i=1}^{i=|F|} \sum_{p \in L} n_{f_i,p} + |F|}$$

where $n_{f_i,p}$ denotes the number of occurrences of f_i in partition node p and $|F|$ is the total number of unique feature including word, p-grams, and t-grams. Details of this maximum likelihood model is described in [6].

The partition tree node with the maximum likelihood value is identified as the concept instance. The concept tree created with the identified concept instances is shown in Figure 10.

4 Related Work

The work described in this paper has broad connections to research in non-visual Web access, Web content analysis, and contextual analysis.

4.1 Non-visual Web Access

Several research projects aiming to facilitate non-visual Web access include work on browser-level support [2,1,11], content adaptation and summarization [40,13,14], organization and annotation of Web pages for effective audio rendition [15,16,17], etc.

Some of the most popular screen-readers are JAWS [2] and IBM's Home Page Reader [1,11]. An example of a VoiceXML browsing system (which presents information sequentially) is described in [18]. All of these applications do not perform content analysis of Web pages. BrookesTalk [13] facilitates non-visual Web access by providing summaries of Web pages to give its users an audio overview of Web page content. The work described in [14] generates a "gist" summary of a Web page to alleviate information overload for blind users. However, summarization of the entire page does not help find the relevant information within it. HearSay goes beyond these systems in scope and approach: it analyzes the content of Web pages and helps find relevant information in them while navigating from one page to another. HearSay dynamically captures the contextual information and uses it to facilitate non-visual Web access.

Traditional screen readers, such as JAWS [37] and Window-Eyes [42] have a mixed and evolving history of approaches to handling dynamic content. Until recently, screen readers did not update their view of a web page except when a new page loaded. aiBrowser transcodes content but only helps improve the usability of Flash movies [38]. With the arrival of Web 2.0, and the trend toward web pages that behave more like applications, the W3C WAI has developed ARIA [41], which enables screen readers to convey dynamic changes to users through its live regions specification. Although this standard enables screen readers to convey changes to users in a usable way, it requires developers to provide this annotation. The Google AxsJAX [36] project enables any programmer to add this ARIA markup to any page, eliminating the requirement that the creator of the content provide the markup, but still requiring manual effort. Dynamo can enable access to web page updates that are not annotated according to ARIA. Web page updates occurring as a result of different methods (static page refreshes, navigation to another page with a similar template, or dynamic Javascript) have each required the user to know different skills to make proper use of them. Dynamo brings functionally equivalent web page updates under control of the same interface features, providing a more consistent view to users.

Web services expose very basic functionalities which by themselves are not sufficient to conduct complex transaction. For instance Amazon's Web service exposes basic tasks such as searching a product in to the shopping cart, etc. One has to compose these primitive complex transactions.This problem has been receiving attention lately [49,50,51,52,53]. All these works typically use process definitions and an ontology to create the composite service with varying degrees of automation. Note that our technique is based on composing operations over Web pages instead of services. A vast majority of transactions on the Web are still conducted over HTML pages. This focus on Web pages is what sets our work apart from those in

Web services.

4.2 Web Content Analysis

A critical piece of our context analysis algorithm is in partitioning Web pages into geometric segments (blocks). Substantial research has been done on segmenting Web documents [11,20,21]. These techniques are either domain-specific [20], site-specific [11], or depend on fixed sets of HTML markups [21]. Semantic partitioning of Web pages has been described in [8,22]. These systems require semantic information (e.g. ontologies). In contrast to all of these works, our geometric clustering method does not depend on rules, domain knowledge or ontologies.

Web page partitioning techniques have been used for content adaptation [23,24] and content caching [25]. VIPS [26] algorithm uses visual cues to partition a Web page into geometric segments. This algorithm is used in [27], where the segments are described by a set of features (e.g. spatial features, number of images, sizes, links, etc.). The feature values are then fed into an SVM, which labels the segments according to their importance.

HearSay also uses an SVM to rank the blocks on the destination Web page w.r.t. the context of the link in the source page. In contrast to [27], where the SVM model was learned using features only from the content of Web page segment, our SVM model uses a feature set, computed from both the context of the link and the content of the block.

The main difference between our research and the above-mentioned techniques is that we exploit geometrical and logical structure of Web pages both to collect context and to identify relevant information on the next Web page.

4.3 Contextual Analysis

The notion of context has been used in different areas of Computer Science research. For example, [28] defines context of a Web page as a collection of text, gathered around the links in other pages pointing to that Web page. The context is then used to obtain a summary of the page. Summarization using context is also explored by the InCommonSense system [29], where search engine results are summarized to generate text snippets.

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 [30], where context of a link is used to get the 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 AcceSS system [31], to get the preview of the entire page. However, presenting a preview does not guarantee the reduction of browsing time.

All of these works define the context of the link as an ad-hoc collection of words around it. In contrast, our notion of context is based on topic similarity of text around the link. We use a principled approach for context analysis with a simple topic boundary detection method [32], confined to geometric clusters that have

semantically related content.

5 Conclusion

In this paper, we described three techniques for bridging the web accessibility divide between sighted and visually impaired users. First, we described the design of context-directed browsing approach, which uses Web page partitioning and techniques from NLP and Machine Learning. Using our system, visually impaired individuals can potentially imitate the browsing behavior of sighted users, saving their time on not listening to irrelevant information. Contextual browsing also has implications for handheld devices with small screens. We implemented context-direct browsing algorithm to identify and display the most relevant sections of Web pages on small-screen handhelds [48].

We also described the Dynamo approach to making dynamic web content accessible and web page updates usable, a problem whose importance, spurred by the advent of Web 2.0, has been continually growing. User studies with the HearSay-Dynamo system indicated that using the system improved disability to web page updates. This initial work has opened up several avenues for improving HearSay implementation and conducting additional research. First, the participants suggested several improvements to the user interface that can be explored to optimize the user experience. Second, although HearSay can detect web page updates, many participants wanted to know the semantic roles of those updates. A collaborative approach could leverage a community of users to label content and share the labels among themselves. We believe these goals can build on the foundation outlined here and make dynamic content efficiently accessible to blind web users.

Finally, we talked about our process modeling for directing Web transactions. We showed that, by delivering relevant page fragments at each transactional step, a model-directed web transaction (i.e. process model) can improve Web accessibility and substantially reduce the digital divide between sighted and blind users.

Acknowledgement

We would like to thank NSF (Award IIS-0534419, CNS 0751083 and IIS 0808678), Sigma Xi (Award G20071013028487872), and CEWIT (http://www.cewit.org/) for supporting this research, and Cepstral.com for donating their realistic synthetic voices. We are grateful to the Helen Keller Services for the Blind for providing their invaluable feedback. And, finally, we would like to thank our developers for their dedication in implementing the HearSay browser.

References

[1] C. Asakawa and T. Itoh. User interface of a home page reader. In ASSETS, 1998.

[2] http://www.freedomscientific.com.

[3] J. Mahmud, Y. Borodin and I.V. Ramakrishnan, CSurf: A Context-Driven Non-Visual Web-Browser. WWW2007,2007.

[4] Y. Borodin, A flexible vxml interpreter for non-visual web access.ASSETS' 06, 2006.

[5] C.-C. Chang and C.-J. Lin, LIBSVM: a library for support vector machines, 2001.

[6] Z. Sun, J. Mahmud, S. Mukherjee, and I. V. Ramakrishnan. Model-directed web transactions under constrained modalities. sWWW 2006.

[7] V. Vapnik, Principles of risk minimization for learning theory. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, Advances in Neural Information Processing Systems 3, pages 831-838. Morgan Kaufmann, 1992.

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

[9] Y. Borodin, J. P. Bigham, R. Raman and I.V. Ramakrishnan, Whats New? Making Web Page Updates Accessible. ASSETS' 06, 2006.

[10] J. Oncina and P. Garc, Inferring regular languages in polynomial update time. World Scientific Publishing, 1991. Pattern Recognition and Image Analysis.

[11] H. Takagi, C. Asakawa, K. Fukuda and J. Maeda, Site-wide annotation: Reconstructing existing pages to be accessible. In ASSETS, 2002.

[12] J. Oncina and P. Garc. Inferring regular languages in polynomial update time. World Scientific Publishing, 1991. Pattern Recognition and Image Analysis.

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

[14] S. Harper and N. Patel, Gist summaries for visually impaired surfers. In Assets '05, Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility, pages 90-97, 2005.

[15] A. Huang and N. Sundaresan, A semantic transcoding system to adapt web services for users with disabilities. In ASSETS, 2000.

[16] M. Hori, G. Kondoh, K. Ono, S. ichi Hirose, and S. Singhal, Annotation-based web content transcoding. In WWW, 2000.

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

[18] http://www.internetspeech.com.

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

[20] D. Embley and L. Xu, Record location and reconfiguration in unstructured multiple-record web documents. In WebDB, 2000.

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

[22] 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.

[23] O. Buyukkoten, H. Garcia-Molina, and A. Paepcke, Seeing the whole in parts: Text summarization for web browsing on handheld devices. In WWW, 2001

[24] S. Baluja, Browsing on small screens: recasting web-page segmentation into an ecient machine learning framework. In WWW, pages 33-42, 2006.

[25] L. Ramaswamy, A. Iyengar, L. Liu, and F. Douglis, Automatic detection of fragments in dynamically generated web pages. In WWW, 2004.

[26] S. Yu, D. Cai, J.-R. Wen, and W.-Y. Ma, Improving pseudo-relevance feedback in web information retrieval using web page segnmentation. In WWW, 2003.

[27] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma, Learning block importance models for web pages. In WWW, pages 203-211, 2004.

[28] B. B.-M. J.-Y. Delort and M. R. Enhanced, Enhanced web document summarization using hyperlinks. In HYPERTEXT'03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia, pages 208-215, 2003.

[29] E. Amitay and C. Paris, Automatically summarising web sites - is there a way around it? In Proceedings of ACM 9th CIKM, pages 173-179, 2000.

[30] 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.

[31] B. Parmanto, R. Ferrydiansyah, A. Saptono, L. Song, I. W. Sugiantara, and S. Hackett, Access, accessibility through simplification & summarization. In proceedings of the International Cross-Disciplinary Workshop on Web Accessibility W4A'05, pages 18-25, 2005.

[32] J. Allen, Topic detection and tracking: Event-based information organization. Kluwer Academic Publishers, 2002.

[33] W. van der Aalst and A. Weijters, Process mining: A research agenda. Computers and Industry, 53:231-244, 2004.

[34] K. Murphy, Passively Learning Finite Automata, 1995.

[35] D. Angluin, On the complexity of minimum inference of regular sets. Information and Control, 39(3):337-350, 1978.

[36] Google-AXSJAX08 http://code.google.com/p/google-axsjax.

[37] JAWS 9.0 http://www.freedomscientific.com May08.

[38] Miyashita, H., et al. Sato, D., Takagi, H. and Asakawa, C, Aibrowser for multimedia: introducing multimedia content accessibility for visually impaired users. SIGACCESS07.

[39] E. Pontelli, et al, Navigation of html tables, frames, and xml fragments. In ACM ASSETS 2002.

[40] J. T. Richards and V. L. Hanson, Web accessibility: a broader view. In WWW '04, NY, USA, 2004.

[41] WAI ARIA http://www.w3.org/WAI/intro/aria 2008

[42] WindowEyes http://www.gwmicro.com/Window-Eyes/ 2008

[43] Yeliz Yesilada, Robert Stevens, Simon Harper, Carole A, Goble: Evaluating DANTE:Semantic transcoding for visually disabled users. ACM Trans. Comp.-Hum. Inter07

[44] M. Zajicek, et al, Web search and orientation with brookestalk. Tech. and Persons with Disabilities Conf., 1999.

[45] Myers, E. W, "An O(ND) difference algorithm and its variations, " Algorithmica 1, 251-266. 1986

[46] E. Berk, "HtmlDiff: A Differencing Tool for HTML Documents", Student Project, Princeton University

[47] Yuan Wang, David J. DeWitt and Jin-Yi Cai, X-Diff: An Effective Change Detection Algorithm for XML Documents

[48] Yevgen Borodin, Jalal Mahmud and I.V. Ramakrishnan, " Context Browsing with Mobiles - When Less is More" , 5th International Conference on Mobile Systems, Applications and Services, ACM/USENIX MobiSys 2007, (http://www.sigmobile.org/mobisys/2007/index.html) 11-14 June 2007, San Juan, Puerto Rico.

[49] S. Agarwal, S. Handschuh and S. Staab, Surfing the service web. In Proc. of Intl. Semantic Web Conf. (ISWC), pages 211-216, 2003.

[50] L. Chen, N. Shadbolt, C.A. goble, F. Tao, S.J. Cox, C. Puleston and P.R. Smart, Towards a knowledge-based approach to sematctic service composition. In Proc. of Intl Semantic Web conf. (ISWC), pages 319-334, 2003.

[51] J. Rao, P Kiingas and M. Matskin, Logic-based web services composition: From service description to process model. In Proc of Intl. Conf. on Web services (ICWS), 2004.

[52] P. Traverso and M. Pistore, Automated composition of semantic web services into executable processes. In Proc of Intl. Semantic Web Conf. (ISWC), 2004.

[53] A. Wombacher, P. Fankhauser and E. J. Newhold. Transforming capitalization into annotated deterministic finite state automata for service discovery. In Proc of Intl. Conference on Web Services(ICWS), 2004.