What's New? – Making Web Page Updates Accessible

Yevgen Borodin* Jeffrey P. Bigham[#] Rohit Raman* I.V. Ramakrishnan*

*Stony Brook University, Comp. Sci. Department, Stony Brook, NY 11790, {borodin, rraman, ram}@cs.sunysb.edu
[#] University of Washington, Dpt. of Comp. Sci. & Eng., Seattle, WA 98195, jbigham@cs.washington.edu

ABSTRACT

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. The connecting theme is the need to quickly and unobtrusively identify the segments of a web page that have changed and notify the user of them. In this paper we propose Dynamo, a system designed to unify different types of dynamic content and make dynamic content accessible to blind web users. Dynamo treats web page updates uniformly and 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. From an algorithmic and interaction perspective Dynamo detects underlying changes and provides users with a single and intuitive interface for reviewing the changes that have occurred. We report on the quantitative and qualitative results of an evaluation conducted with blind users. These results suggest that Dynamo makes access to dynamic content faster, and that blind web users like it better than existing interfaces.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia – *architectures, navigation*

General Terms

Design, Experimentation, Human Factors, Standardization, Algorithms

Keywords

HearSay, Blind Users, Web Browser, Dynamic Content, Screen Reader, Non-Visual Aural Interface.

1. INTRODUCTION

The Web is evolving from a collection of static pages into a platform for interactive web applications. To facilitate this transformation, web developers are increasingly using technologies like DHTML, AJAX, and Flash to make their web sites more interactive. The popular screen readers, however, have lagged behind the new technologies, resulting in a widening of the accessibility gap between sighted and blind web users. As a result, accessing dynamic web pages remains difficult, frustrating, and sometimes impossible for more than 10 million[1] partially-sighted and blind people in the United States alone.

The W3C standard for Accessible Rich Internet Applications (ARIA) [10] addresses the problems of accessing dynamic content using content markup. ARIA provides a standard method for assigning roles and states to DHTML elements, and for describing how such elements dynamically update. While popular screen-readers and web browsers are already beginning to support ARIA markup, they are currently unable to automatically identify and present to users dynamic content changes that occur in web pages without the appropriate markup. Although ARIA may be the ultimate solution, like most annotation standards, the ARIA approach will only work if the developers choose to implement it; unfortunately, most dynamic content available today does not implement the ARIA standard. As a result, blind users may not be aware of important dynamic updates occurring in the web pages they view, and can in turn waste time and edge on the inability to perform online transactions, such as shopping or banking. Observations of browsing patterns have shown that blind people tend to avoid dynamic web pages altogether [2].

This paper introduces Dynamo – a unified approach to handling content changes in web pages that is agnostic of the technical methods used to update the pages. Dynamo enables users to interact with most types of dynamic content, even that which has not been annotated according to the ARIA specifications. In addition, Dynamo unifies the interface used to explore dynamic content with the one used for other types of content changes that occur while browsing the Web. As a result, users need not know how web content updated – whether it was from JavaScript, page refresh or navigation, whether it happened on the client or server side – all of these are functionally equivalent. From the user's perspective, JavaScript-powered web page updates are just the latest symptom for the underlying trend of inaccessible web content.

Although traditionally dynamic content is associated with technologies such as DHTML, AJAX, and Flash, we expand it to include: (i) page refreshes, both manual and automatic, which happen without warning, and interrupt and confuse users (Figure 2); (ii) template-based web sites that only change part of the page, as users browse from one page to the next (Figure 4); (iii) links that bring up content in new frames, and form submission pages

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ASSETS'08, October 13–15, 2008, Halifax, Nova Scotia, Canada.
Copyright 2008 ACM 978-1-59593-976-0/08/10...$5.00.

[1] According to the American Foundation for the Blind.

that return the user back to the form with the errors in form fields highlighted (Figure 3). Common among all these content changes is that the user performs an action, and the relevant content appears or changes somewhere on the page. Furthermore, this content is often updated without notifying the user and often with no easy way to find it. The Dynamo approach unifies all of these types of web page updates by automatically detecting the underlying changes and providing users with a single and intuitive interface for reviewing the changes.

In this paper, we address the problems of accessing changing content and propose the Dynamo approach, as a solution. Below is a summary of our contributions:

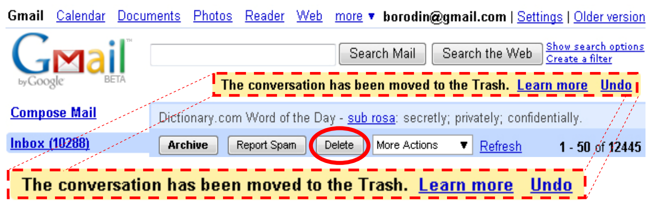
1. A unique and novel aspect of Dynamo is that it treats all types of page updates mentioned above as "dynamic" and handles them uniformly.
2. We implemented Hearsay-Dynamo (HD), a Dynamo prototype in the framework of the HearSay non-Visual Web Browser, specifically:
    a. We extend the buffer-based interface of the HD browser to partially update its VoiceXML dialogs as the changes occur on the web page.
    b. We propose a diff algorithm to filter out content that does not change as a result of dynamic updates, including refreshes and page-to-page navigation.
    c. We provide a usable interface in HD to give users the ability to review the changes and control the level of intrusion caused by dynamic updates.
    d. We maintain user context in HD, and then use it to reposition the users on the web page in case of dynamic updates, refreshes, and simple page-to-page navigation.
3. We report on a study with blind users, which confirms that our approach can help identify relevant information in web pages, allow users to stay focused on their tasks, and, as a result, save a significant amount of time in non-visual web browsing.

In the remainder of the paper we will give an overview of the scenarios that benefit from our approach, discuss the details of the architecture and algorithms, and present the results of the user study.

2. USE CASES

In this section we discuss two scenarios that motivate Dynamo. Figure 1 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.

A more serious problem is captured in Figure 3, where an incorrectly entered zip code results in error messages (dashed boxes) appearing on the page. But only the first message is actually new, the other one just changed the color. From the point of view of a novice user, the form did not submit, because the page almost did not change. A more experienced user would know about form validation and would tab through all fields to check for errors. However, the error may not be easy to find, especially for a novice, and tabbing through the form skips the error message describing the problem. If users had an easy way to review the changes, they would be able to read the error message and then jump to the highlighted field in no time at all. In fact, it should not matter to the users whether the changes resulted from a dynamic or server-side form validation. All that matters is finding the errors fast.



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

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

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

Figure 1 – Dynamically appearing "Undo" link at Gmail.com

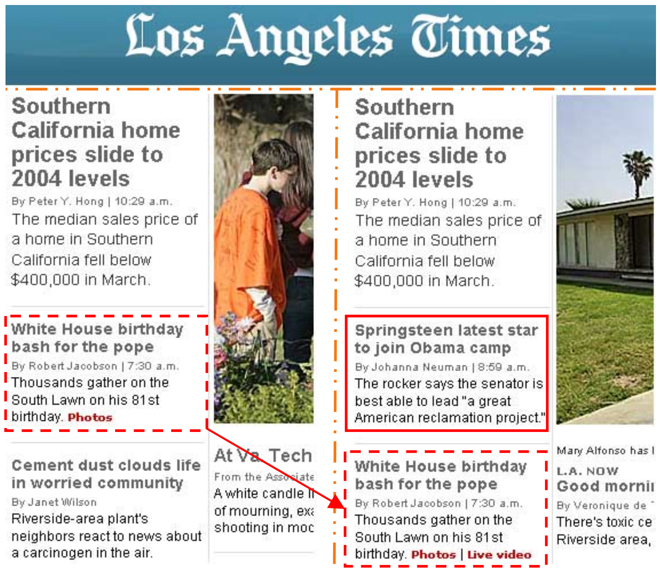
Figure 2 illustrates an automatic page refresh that causes changes to the content of the Los Angeles Times news web site. Some screen-readers, such as JAWS, have an option of suppressing automatic refreshes, but in some cases, users may choose to refresh the page manually. Not only should they then be able to review the newly added content (solid box Figure 2b), 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 2a), instead of starting from the beginning after the refresh, the cursor should be repositioned to the same article (dashed box Figure 2b). 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. HANDLING WEB CONTENT UPDATES

3.1 Overview of Traditional Dynamic Content

With the advent of Web 2.0, the use of dynamic content in web pages has become ubiquitous. Some of the most popular technologies are JavaScript, DHTML, AJAX, Flash, and Java Applets. The last two are not in scope of this paper. With many free tutorials and shared JavaScript repositories, it is now very easy even for beginners to insert some script to spice up web pages with pull-down menus, dynamic tabs, etc. Dynamic content is used in most flight reservation and car rentals web sites. It is often used in web forms to point out the errors in the fields. Many ads now update periodically.

But not all dynamic updates are created equal. Depending on the type of dynamic content it may or may not be accessible with regular screen-readers. Very often web developers use dynamic content to pack more information in less space. For example, a modest menu with just 5 menu elements can easily hide 50+submenu items, which will eagerly pop out as soon as you mouse-over. Since most screen-readers get page content from the HTML DOM trees and do not filter out hidden content, such dynamic content is technically "accessible". However, it may be very difficult to navigate if web developers did not follow the Accessible Rich Internet Applications (ARIA) [10] or even Web Content Accessibility Guidelines (WCAG) guidelines. In case of form-filling, all possible error messages are often hidden from sight, but not from the screen readers, which can read them all, thus, making it impossible for blind users to find the actual



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

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

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

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

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

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

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

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

a) Before b) After

Figure 2 – Automatically refreshing front page at L.A. Times

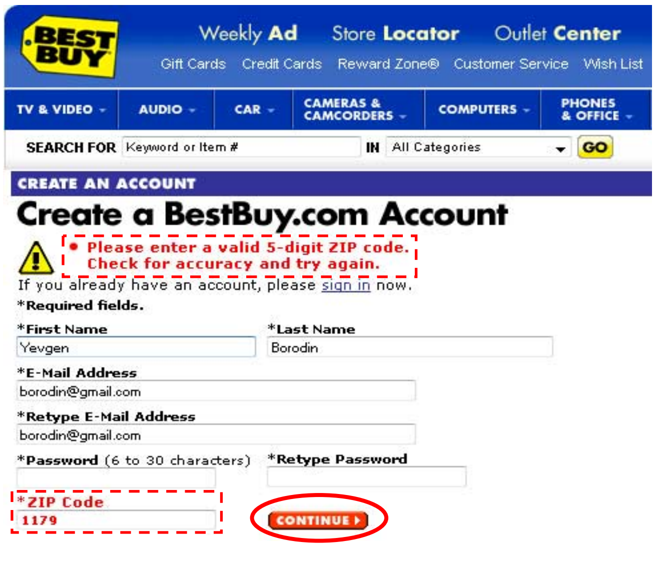
errors without carefully reviewing the content of each form element. In addition, our observations showed that, although hidden content visually appears in the correct position, it is often placed at the end of the DOM tree, often in no particular order. This means that most screen-readers will get to this content at the end of the page, and blind users are unlikely to find spatial or even sequential relationships that are obvious to sighted people.

In other situations, dynamic content maybe pulled from the web server by Asynchronous JavaScript and XML (AJAX), which updates portions of page content without refreshing the entire web page. Additionally, dynamic content may also be hard-coded in the JavaScript and, thus, made invisible to screen-readers. In both cases, to make the dynamic content "appear", a screen-reader has to refresh its buffer, hopefully, without losing the position of its cursor, which, if not successful, can result in the screen reader restarting from the beginning. Often users will end up browsing stale content or not be able to access it at all, often without knowledge that the page has changed. Both of these situations could be addressed by proper use of the ARIA guidelines, but it seems unlikely that web developers will do this consistently.

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

3.2 Content Updates: A Unified View

Recall that we take an expansive view of dynamic content encompassing both the traditional ones reviewed above as well as updates caused by page refreshes, inter-page browsing, form submission, etc. We therefore propose a unified framework for handling all these 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.



Create a BestBuy.com Account

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

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

*Required fields.

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

Figure 3 – Server-side validation at BestBuy.com

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

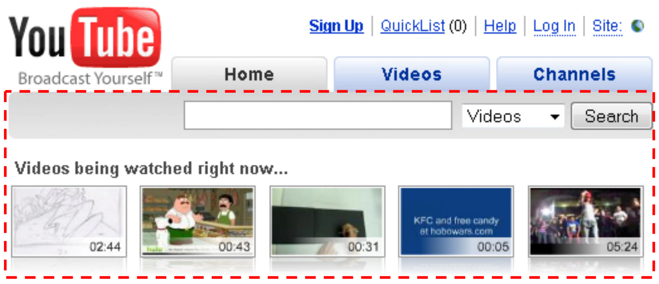


Figure 4 – Static tabs in the design of YouTube.com

Most static content can be replaced with AJAX, so that instead of requiring multiple pages to be loaded, the system can simply load new content and place it into the current page. GMail is one such example. Each bit of new information or every user action could require a full page refresh; however, because the site uses AJAX, the site appears to operate similar to equivalent desktop applications. 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". For example, following a link and loading a new page can be thought of as a dynamic change of web content within one generic template of your web browser.

Regardless of whether web sites are implemented using dynamic or static approaches, users browsing for information usually care only about the information that 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.

To achieve this transparency and make browsing more usable, we can abstract from the specific web technologies and underlying methods for detecting content changes, and treat both dynamic and static content universally. 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.

4. HEARSAY-DYNAMO

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

4.1 System Architecture

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[2] dialogs, and manages the user interface through the vxmlSurfer - a custom VoiceXML interpreter. The vxmlSurfer maintains a separate dialog thread for each tab opened in the web browser. Although HD users work directly with the regular browser (e.g.: while tabbing or filling web forms), the HD backend also sends messages to the HD extension, for example to simulate click-events on non-focusable content.

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. This also allows us to unify the treatment of both static and dynamic pages, handling all cases uniformly by injecting the new content into VoiceXML dialogs in much the same fashion as AJAX updates web page content.

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

4.2 Dynamo-Diff Algorithm

We designed the Dynamo-Diff algorithm to filter out content that does not change as a result of web page updates. The algorithm uses a top-down approach with max-flow bipartite matching of two HTML DOM Trees.

Algorithm DynamoDiff
Input: DOM1 & DOM2: the DOM Trees for two web pages
Output: S: list with tuples if nodes matching in the two DOMs
Output: D: list of nodes which are different in the two DOMs

```
1.  M1 ← BuildMap(DOM1)
2.  M2 ← BuildMap(DOM2)
3.  M ← (M1 U M2) // Combine Maps
4.  MS ← [] // Initialize Matching List
5.  For each (S:{L1, L2}) in M,
6.      Do For every node N1 in L1
7.          Do For every node N2 in L2
8.              Do If hasSameContent(N1,N2) = true
9.                  Then MS.add((N1,N2))
10.     For each (N1, N2) in MS
11.         Do If ( N1 ∈ L1 and N2 ∈ L2)
12.             Then S.add((N1, N2))
13.                 L1.remove(N1)
14.                 L2.remove(N2)
15.     For each node N in L1 or L2
16.         Do D.add(N)
17. Return (S, D)
```

Algorithm: BuildMap
Input: HTML Dom Tree T
Output: Map M

```
1.  M ← {}
2.  Do DFS on T
3.  For each node N in T
4.      Do If N.isImportant = True
5.          Then If N.isLeaf = True
6.              Do M.add(N.signature: N)
7.              Else
8.                  For each attribute A of N
9.                  Do M.add(A.signature:A)
10. Return H
```

Dynamo-Diff first does a DFS on each of the trees to construct two maps (*DynamoDiff:Lines 1-2*), which have two sets of vertices for the bipartite graph. The algorithm only considers the nodes that are "*Important*" or have "*Important*" child nodes,

[2] W3C Standard for creating interactive dialogs.

which may affect content or visual presentation of the content (*BuildMap:4*). 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 5, which illustrates the two HTML DOM Trees of the L.A. Times web page before and after refresh in Figure 2.

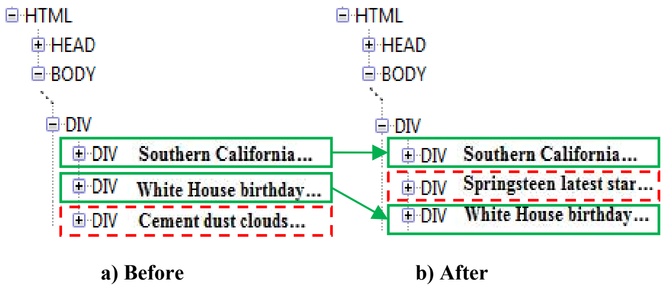

HTML
HEAD
BODY
DIV
DIV Southern California...
DIV White House birthday...
DIV Cement dust clouds...
HTML
HEAD
BODY
DIV
DIV Southern California...
DIV Springsteen latest star...
DIV White House birthday...
a) Before
b) After

Figure 5 – Dom snapshots of L.A. Times refreshing front page

DynamoDiff combines the maps (Line 3) and then performs bipartite matching of the nodes. Every pair of nodes, that have the same content and same presentation style, are added to the Matching List (*MS*) as a tuple (N1, N2) in Line 9. Thus, each tuple (N1, N2) corresponds to an edge in the bipartite graph. Continuing the example in Figure 5, the *MS* will contain the matching nodes marked with green solid boxes. Now, we apply the max flow routine to find the maximum number of matches by including the maximum number of edges (tuples) in the list S. Thus, the nodes that did not change are added to list S (line 12) – the same nodes in Figure 5; and the different nodes are added to list D (Line 16) – marked with red dashed boxes. It may happen that 1 node in DOM1 would be a candidate for matching with 2 or more nodes in DOM2, i.e. MS may contain the tuples { (Na, Nb), (Na, Nc), (Na, Nd) }, in that case, our max flow would take care of it by considering the 1st matching pair present in MS (Na, Nb) and adding the rest (Nc, Nd) to the list D.

HD then uses the content in D list to remove from or add to the VoiceXML dialogs and updated groups the content by proximity. Some more details about the Dynamo-Diff algorithm and the comparison with other diff algorithms are given in Section 7.2.

4.3 Interface for Accessing Content Updates

HD converts web page content to VoiceXML dialogs and then uses vxmlSurfer, a customized voiceXML interpreter, to manage and interpret the dialogs.

Hidden content is generally filtered out, however hidden links used for accessibility purposes should remain accessible. When a new page loads or a dynamic update arrives, HD compares the actual content and its attributes to the previous page state, groups the updates, and then dynamically updates the corresponding VoiceXML dialog. HD provides a menu to consistently access the changes and has several notification modes to control the level of update notification and, thus, distraction. Users have an option of either using keyboard shortcuts or using voice commands.

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 a shortcut that jump to the next or previous group of updates. There is also an option to return to the original location after reviewing the updates. Another way to examine the changes is to turn on the filter for updates, which makes all unchanged content disappear and allows using regular navigation shortcuts to browse only the changes. At any moment, the filter can be turned off, and the users can continue browsing the page from their current location. They can also clear the reviewed changes, so that it is easier to find new changes that may happen due to dynamic updates. This interface helps review differences between pages and the changes that occur as a result of dynamic updates. In the future, it may prove useful to provide a way to order the dynamic updates by time of arrival or some other importance criteria.

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

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

5. HEARSAY-DYNAMO USER STUDY

In this section we present an evaluation of Hearsay-Dynamo with 8 blind web users over 3 categories of content updates.

5.1 Participants

Eight blind web users participated in our study (1 female) ranging in age from 18 to 56. Participants were recruited from email lists and local centers for the blind. For purposes of this study, *blind users* will refer to participants who were self-described as primarily using a screen reader to browse the web. Participants varied in their screen reader proficiency from relatively inexperienced to expert.

5.2 Experimental Setup

The following two versions of HearSay were used in our evaluation: (i) HearSay-Basic (HB) providing only the basic JAWS navigation shortcuts but including the capability to update its VoiceXML dialogs with new content; and (ii) HearSay-Dynamo (HD), notifying users of updates and providing update-navigation shortcuts. We considered evaluating HD using a popular screen reader to which our participants would have already been familiar, such as the JAWS screen reader, but could not easily integrate the Dynamo approach into these proprietary software programs. We also considered evaluating HD against JAWS, however, some variations in shortcuts, text-to-speech engines, and interface familiarity could have introduced uncontrolled random variables and biased our evaluation. In addition, JAWS does not provide even basic access to all of the dynamic content used in the evaluation, making access impossible using it.

Participants completed 3 tasks using both HB and HD. To avoid learning effects as participants complete each task twice, we chose two sites that we believed were approximately similar in structure for each category. Although the chosen sites were specific instances of each category, we believe the characteristics of each are representative of the types of updates in each category. Tasks were drawn from the three categories explained in detail next.

5.2.1 Responding to a Dynamic Message

Tasks in this category required that subjects find and then click on a link that is dynamically introduced into a web page. Such dynamic messages are often used to convey updated information or to respond to a user-initiated action. For this task, users first deleted a specified email in a GMail email inbox or document in a Google Docs document selector view. In response to that action, both pages inserted a DIV element at the top of the page, confirming the delete action and providing a link labeled "Undo" for undoing the action. The task was to find and click the "Undo" link. Task time was measured from the time the participant hit the delete button to when they clicked "Undo".

HD alerted the participants that content had changed on the page with an earcon, and provided an interface for reviewing the changes. HB only alerted participants of updated content; however, they had to use standard shortcuts to find it. In contrast, JAWS did not detect the inserted content and did not update its buffer, the "Undo" link was not accessible.

5.2.2 Form-filling with Error Recovery

For the next task, the evaluators completed a web form reproduced from either the BestBuy order form or Delta SkyMiles registration page. The form asked users for their name, address, phone number and email address. Upon submission, the form programmatically introduced an error into either the phone number or zip code field (a digit was removed). The user was then returned to the form with an error message inserted above the form and the field with the error highlighted in red. The task was to find the error and fix it. To ensure that users were able to complete the task in the allotted time, the participants were informed that there was an error once they had submitted the form.

HD automatically detected that the page was updated with error messages, alerted the user of the changes with an earcon, and provided the interface for reviewing the errors. HB provided no alerts; and the users had to find the errors using the standard interface.

5.2.3 Maintaining Context on Page Refresh

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

5.3 Procedure

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

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

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

System	*{HearSay-Basic, HearSay-Dynamo}*
Task Category	*{Dynamic Message, Form-Filling w/ Error Recovery, Page Refresh}*

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

5.4 Results

5.4.1 Speed of Task Completion

Overall, HearSay decreased the time required for users to complete the 3 tasks (Figure 6). For the Dynamic Message task, participants spent a mean of 94.5 seconds (std. dev.=100.8) using HB and a mean of 36.5 seconds (27.4) using HD. This represented a significant effect of *System* on task completion time ($F_{1,7}$=3.67, p<.01), indicating that HD improved this task. Although participants varied greatly in time to complete this task, 7 out of 8 completed it more quickly using HD. P8 completed it more slowly because he forgot the shortcut key used to review web page updates and spent >30 seconds trying to remember it.

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

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

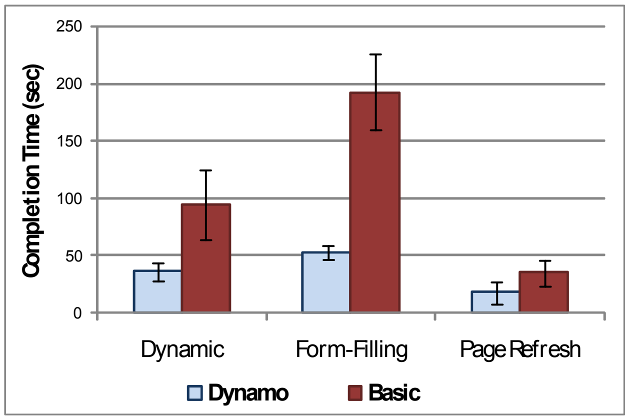


Figure 6 – Average time required to complete each task (seconds). Error bars +-1 Standard Error.

5.4.2 Subjective Feedback

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

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

Table 1: Average 5-Point Likert scale values by participants (scale from 1=Easy to 5 =Difficult) with SE.

Following the study we asked participants to what extent they agreed with several statements on a 5-point Likert scale from (1=Strongly Disagree) to (5=Strongly Agree). (See Table 2) Overall, participants agreed that many different kinds of web page updates made it web browsing more difficult using a screen reader, they thought that HD could improve access to web page updates, and wanted to use these features of the system in the future.

6. Discussion

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

General Statements	**Response**
Dynamic content makes web pages difficult to use.	4.1 (0.6)
Pages that refresh automatically are difficult to use.	3.9 (0.3)
It is often difficult to find errors when I make an error filling out a web form.	4.0 (0.2)
HearSay-Dynamo	**Response**
HearSay made interaction with dynamic content easier.	4.0 (0.4)
HearSay improved the tasks that I used it on.	4.3 (0.5)
I want to use the features of HearSay that handle dynamic content in the future.	4.4 (0.8)

Table 2: Average 5-Point Likert scale values by participants (scale from 1=Strong Disagree, to 5=Strongly Agree) with SE.

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

Because headings were provided by the news web pages in the "Page refresh" task, participants were quickly able to find the appropriate article and then its author. All but one participant was able to complete this task more quickly using HD; the participant became disoriented when the page refreshed and pressed the "next heading" shortcut twice quickly in succession, causing him to inadvertently skip past the correct article. After he had gone past the article, he became disoriented and finding the correct article required backtracking.

Participants generally saw a need for better handling of dynamic content and, when asked, most did not realize the technical differences between the tasks in our evaluation. Our unified approach to web page updates does not require users to be aware of technical details of different kinds of web page updates, and seemed to be preferred by participants in our evaluation.

7. RELATED WORK

7.1 Screen Reading Technology

A number of projects for making the web accessible have been reported (e.g. [7, 8, 9, 13, 14]). But the focus of these efforts is not on dynamic content in its full generality as addressed in this paper. aiBrowser transcodes content but only helps improve the usability of Flash movies [7]. Traditional screen readers, such as

JAWS [4] and Window-Eyes [12] 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. With the advent and increased prevalence of JavaScript and DHTML, users have been given the option to explicitly instruct screen readers to refresh their view of the DOM. With the arrival of Web 2.0, and the trend toward web pages that behave more like applications, the W3C WAI has developed ARIA [10], 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 [3] 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. Both of these techniques will provide the best experience for the content to which they apply, but neither will be implemented in every web application in the near future. 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. The technical means of facilitating a web page update, and not the function of the update, determined the features of the relevant interfaces. Dynamo brings functionally equivalent web page updates under control of the same interface features, providing a more consistent view to users.

7.2 Diff Algorithms

The development of text editors and the need to compare text versions were the original motivation for diff utilities for finding differences between text files. Most diff algorithms are based on the Longest Common Subsequence (LCS) approach. Among the first diff algorithms described in the literature was the *O(ND)* algorithm [6], which builds an edit graph (grid) between two documents and, then, looks for an optimal path with the least number of differences. The complexity of the dynamic programming implementation is O(N*D), where D is the number of differences, and N is size of combined trees. The LCS algorithms, however, do not guarantee 100% accuracy. Since most web pages are not all plain text, *HtmlDiff* [1] improves the accuracy of the *O(ND)* algorithm by treating two HTML pages as sequences of tokens (sentence-breaking markups or sentences). *HtmlDiff* then uses a weighted LCS algorithm to find the best match between the two sequences.

Both the accuracy and complexity of diff algorithms can be improved if the hierarchical structure of HTML documents was taken into account. After a web browser parses HTML into a DOM Tree, an XML diff algorithm can be used to compare web pages. For example, X-Diff [11] generates a minimum cost edit script to convert one XML document into another. X-Diff uses the notion of edit-distance which is calculated using max-flow matching of nodes in the tree. The algorithm uses XHash, a special hash function, to compare the nodes – two nodes are considered to be similar if they have the same XHash value. X-Diff uses the location of a node as the node's signature (similar to XPath). X-Diff uses a dynamic programming bottom-up approach and is capable of matching whole sub-trees at once. It runs in $O(n^2*d*\log(d))$, where d is the maximum degree of any node in the tree. While the X-Diff algorithm works well for XML documents, it does not consider HTML-specific features. Dynamo-Diff algorithm also uses max-flow approach to matching the trees, but adapts to the needs of Hearsay-Dynamo (HD). The algorithm takes top-down DFS approach ($O(n^2)$ worst case) instead of dynamic programming bottom-up matching used by X-Diff. In contrast, our algorithm only looks for visual differences; it therefore does not consider all nodes of the tree, but only the nodes that affect the content and visual appearance of the page.

8. CONCLUSION AND FUTURE WORK

We proposed and implemented 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, is growing. User studies with the HearSay-Dynamo system indicated that using the system improved their access to web page updates. This initial work has opened up several avenues for improving HD implementation and conducting additional research. First, the participants suggested several improvements to HD's user interface that can be explored to optimize the user experience. Second, although HD can detect web page updates, many participants wanted to know the semantic roles of those updates. 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.

9. ACKNOWLEDGEMENTS

We would like to thank NSF (Award IIS-0534419) and Sigma Xi (Award G20071013028487872) for supporting this research, and Cepstral.com for donating their realistic synthetic voices.

10. REFERENCES

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

[2] Bigham, J. P., et. al., WebinSitu: A Comparative Analysis of Blind and Sighted Browsing Behavior. *ACM SIGACCESS'07* Tempe, Arizona, USA, 2007

[3] Google-AXSJAX'08 http://code.google.com/p/google-axsjax

[4] JAWS 9.0 http://www.freedomscientific.com May'08

[5] Mahmud *et al.* CSurf: A Context-Driven Non-Visual Web-Browser. *WWW 2007.*

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

[7] Miyashita, H., *et al.* Sato, D., Takagi, H. and Asakawa, C. Aibrowser for multimedia: introducing multimedia content accessibility for visually impaired users. *SIGACCESS*'07.

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

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

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

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

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

[13] Yeliz Yesilada, Robert Stevens, Simon Harper, Carole A. Goble: Evaluating DANTE:Semantic transcoding for visually disabled users. *ACM Trans. Comp.-Hum. Inter*'07

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