Tailored presentation of dynamic content

Andy Brown, Caroline Jay and Simon Harper
School of Computer Science
University of Manchester
Kilburn Building, Oxford Road
Manchester, UK
[andrew.brown-3 | caroline.jay | simon.harper]@manchester.ac.uk

ABSTRACT
The advent of ‘Web 2.0’ and Web applications means that documents often are not static, but update, either automatically or as a result of user interaction. This development poses a difficult question for screen reader designers: how should users be notified of page changes? The SASWAT Web browser is a self-voicing screen reading extension to the Firefox Web browser, based upon Fire Vox. It uses a set of rules, derived from studies of how sighted users interact with such content, that allow the presentation of updates to be tailored according to the way in which they were initiated, the effect they have on the page, and their content. These should allow users to deal with page changes in a more natural and more efficient manner.

Categories and Subject Descriptors

General Terms
Human Factors, Experimentation

Keywords
Web 2.0, AJAX, Visual Disability, Eye-Tracking.

1. INTRODUCTION
The increasing use of interactive Web pages, sometimes resembling applications more than documents (one aspect of ‘Web 2.0’), is proving problematic for screen reader users, as their technology lags behind Web development. It also poses difficult questions for screen reader developers, who need to change their interaction model to cope with detecting and presenting updates. The current situation is that screen readers generally do not present updates to the user, and when they do it is done in a simple way, neither accounting for the content of the update nor the user’s activity. The problem is such that many of these users have not knowingly encountered many common types of updating content.

Currently, most attempts to tackle the accessibility of dynamic updates focus on standards and content creation, the most significant of which is Accessible Rich Internet Applications, or WAI-ARIA [3] from the World Wide Web Consortium (W3C) Web Accessibility Initiative (WAI). ARIA markup1 can be broadly split into two areas: that which makes the controls keyboard accessible and their roles clear; and that which deals with the areas that update (‘live regions’). The tags associated with live regions allow assistive technologies to be given information about an update, including how important the update is, how much of the page needs to be re-presented to the user after the update, and how the DOM is affected by the change. While support for ARIA is increasing — it is now included in some browsers and assistive technologies (at least partial support in recent versions of Orca, Window Eyes and Jaws), and some JavaScript ‘widget’ libraries implement ARIA markup — it is not implemented by the vast majority of sites.

The system described here attempts to make updates on non-ARIA enabled sites accessible. While the SASWAT browser is primarily a mechanism for testing different methods of presenting updates (and hence informing the development and deployment of ARIA, and its interpretation by assistive technologies), it also has the potential to directly benefit users.

2. THE SASWAT BROWSER
Eye-tracking studies on sighted users revealed patterns in the ways in which they handled dynamic content [4]. Broadly speaking, those which were either requested by the user (e.g., the user followed a link to get information) or initiated by them (e.g., an update resulting from other user activity, such as typing in a form) were nearly always looked at. Those updates that occurred automatically, e.g., news tickers, were largely ignored. The SASWAT browser uses these findings to tailor the presentation of updates. It is a self-voicing extension to the Firefox Web browser based on the Fire Vox extension2.

The user interface for the experimental prototype was changed from that used in the standard Fire Vox, on the ba-

---

1Introduced at http://www.w3.org/WAI/intro/aria.php
2http://www.firevox.clcworld.net/
sis that this was not particularly intuitive, and might take more learning time than would be appropriate in a short (less than two hours) evaluation. The replacement interface [2] was loosely based on that used in the Orca screen reader for the Gnome desktop on Linux. Navigating around a page can be done in two ways: sequential navigation, or element-based navigation.

Sequential navigation allows users to move from element to element as they appear in the Document Object Model (DOM) — a depth-first traversal of the tree. This can be done at three levels: element, sentence or word. Moving at these three levels of granularity is achieved using the number key pad found on a typical desktop computer keyboard. Each row relates to a different level of granularity: the top row moves by element, the middle by sentence, and the bottom by word. The columns enable the user to move backwards (left column) or forwards (right column), or to query their current location (middle column).

Element-based (or structural) navigation is a method for jumping between elements of a certain type. For example, moving between, and reading, the headings on a page can be used to give an overview of its content. This type of navigation is achieved by using character keys (e.g., 'H' moves between heading elements) when in structural navigation mode. This mode may be toggled between on, off, and automatic. In automatic mode, structural navigation is enabled unless the focus is in a form element requiring typing, such as an input box.

The contribution of this system, however, lies not in the user interface, but in the handling, and particularly the presentation, of dynamic content. Updates are handled in a three-stage process:

1. Detecting page changes then clustering them into meaningful updates.
2. Classification, according to the attributes of the update and the user's activity.
3. Presentation, applying the rules appropriate for the class.

Updates are detected by listening for DOMMutationEvents, which are fired by the browser when the DOM changes. These are not immediately useful, however, since an event is triggered at a low-level. For example, the replacement of a DIV containing two paragraphs with one containing a different two paragraphs will result in multiple events, including ones for the addition or removal of white space. This is much below the level of change that is perceived by users (visually such an update would appear as a seamless replacement), so a 'chunking' algorithm must be applied to events. This groups DOMMutationEvents into user-level events, as one of: removal, addition, replacement, or rearrangement. This classification is accompanied by another according to how the update was initiated. This process is performed using heuristics that classify the initiation according to the recent activity of the user.

Once updates have been identified, grouped, and classified they need to be presented to the user. In many current screen readers this is done without regard for the type of update, indeed some simply ignore all updates leaving the user to discover page changes serendipitously. In the SASWAT browser the form of the presentation is based on the class of update. Automatic updates are not announced, but the user is notified of them by a brief non-speech sound. It is possible to listen to a list of recent updates. User-requested or initiated updates are presented with a brief non-verbal notification (different to that used for automatic updates) and announced. The announcement differentiates between types of update (whether content was added, removed, replaced or rearranged). In all cases apart from removal, the focus is moved to the first element in the new (or rearranged) content, with an announcement if this is distant from the original focus. The system then starts to speak this element. An update region may also be ignored, so no announcements or notifications are given for subsequent updates.

Further refinements were made on the basis of observations and eye-tracks from the sighted user studies. These involved a special mode to replicate the benefits of pop-up calendars for date entry, and an audio implementation of auto-suggest lists [1]. Finally, the tendency for sighted users to view the first chunk of information in an update then return to their original focus was replicated by automatically bookmarking controls as they activated. Thus, when viewing a slide show it becomes simple to browse the start of each slide, returning to the control quickly, even if this is not adjacent to the slide content.

3. CONCLUSIONS AND FUTURE WORK

The SASWAT browser is a research prototype, designed to test interaction metaphors. Those that prove successful can be implemented, either by screen reader developers, who will be able to interpret ARIA tags in a more informed way, or by site developers, who will be able to employ ARIA in a way that will suit users' needs.

4. ACKNOWLEDGEMENTS

This work is part of the Single Structured Accessibility Stream for Web 2.0 Access Technologies (SASWAT) project and is funded by the UK EPSRC (EP/E062954/1). As such the authors would like to thank them for their continued support.

5. REFERENCES