Dynamic injection of WAI-ARIA into Web content

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

ABSTRACT
WAI-ARIA enables Web developers to make dynamic content accessible to users of assistive technologies (ATs) but there remain many sites on the Web that do not use it. Unfortunately the default behaviour of ATs when handling such pages is often sub-optimal, leaving users struggling to use the content. We present ACup: a flexible approach that injects JavaScript into the page to detect and classify any changes to the Document Object Model (DOM). These changes are then presented to the user using a WAI-ARIA live region that was injected when the page was loaded. The style of presentation varies according to the characteristics of each update (using rules previously bound to be effective) and can simply be changed, for example to test novel presentation approaches, or to apply a more fine-grained classification. This may be used to enable AT users to benefit more rapidly from advances in user-interface design.

Categories and Subject Descriptors

General Terms
Human Factors, Experimentation

Keywords
Web 2.0, Accessibility, WAI-ARIA, User-Interfaces.

1. INTRODUCTION
Web content accessibility is optimum when an assistive technology (AT) with a well-designed user interface (UI) is used in combination with an up-to-date standards-compliant browser to view content that is generated using correct syntax and semantics. This combination is often not achieved however [3, 9, 10], leaving some users with significant accessibility problems, which are particularly acute when the page contains ‘widgets’ — sections of content that change within the lifespan of a page. WAI-ARIA markup allows developers to make the function and behaviour of dynamic content explicit so that widgets may also be operated by AT users, but many pages still do not include it, and in these cases the default behaviour of the AT is not as effective as it could be [5].

Previous research used eye-tracking to model how sighted users interacted with dynamic Web content and understand how the content benefited them [4]. This understanding was used to develop rules [5] for tailoring the presentation style of updates according to their characteristics; evaluation with visually impaired users demonstrated that this approach was effective. The evaluation was performed using an experimental prototype, which was suitable for testing the research hypotheses, but was not in a form that would directly benefit end-users. In response, we investigated how new developments in user-interface design for accessibility might be made more rapidly available (‘accessibility catch-up’: ACup). This requires information to be presented to the user, but without a direct mechanism to ‘push’ information from page/browser to AT and leads to the following set of research questions:

1. Can JavaScript be injected into a Web page that is capable of monitoring the page for updates, and classifying those changes?
2. Can WAI-ARIA markup injected by JavaScript following an update be used to modify screen reader behaviour?
3. Can update detection and behaviour modification be performed in a robust manner and with acceptable performance?
4. How generalisable is the process: for what other types of advance might this type of approach benefit users?

This paper attempts to answer these questions, and describes the design of scripts that fulfil the requirements. Section 3 describes an implementation that dynamically detects and classifies updates, section 4 introduces the rule-based system for presenting the updates, and section 5 discusses the how the research questions have been answered.

1See http://www.w3.org/TR/wai-aria-primer/.
2. BACKGROUND

The development of HTML markup to improve accessibility, such as WAI-ARIA, has provided authors of Web content the tools necessary to make their content accessible to a wide range of users. However, engineering an effective user interface is difficult, particularly for highly dynamic content in a non-visual environment [8, 11, 1], and many pages do not use these tools. For these cases, screen readers must decide how to present the content and any changes that occur. The way in which this is done varies depending on the combination of technologies (operating system, Web browser, screen reader, etc.) that are being used, and indeed upon user preferences, but presentation is rarely tailored to the way in which the content changed [3].

The SASWAT project showed that users are able to use dynamic Web pages more effectively and efficiently if the way in which updates are presented are tailored according to how the update was initiated and how it affected the page [4]. The project also demonstrated that it was possible to do this by monitoring users actions and analysing the DOM during page use — it can be seen as extending Hearsay-Dynamo [1], which monitored the page for changes, but did not tailor their presentation.

The AxsJAX framework [7] injected JavaScript and WAI-ARIA into Web a page to improve its accessibility, and included the facility for the user to specify rules that determine how updates to a page should be handled. Creating rules, however, required an understanding of coding and the ability to identify regions of a page using xpaths. ACup seeks to address those areas that AxsJAX does not, namely modify the presentation of updates where the position and type of change is not known in advance, and to do so in a way that could be used with a much lower level of technical knowledge and skills. This paper, therefore, presents a new technique for modifying how dynamic Web content is presented by screen readers, using injected JavaScript and WAI-ARIA markup. The styles of presentation used for these updates have been described previously [4], but this new technique can apply a range of styles according to easily set rules.

3. UPDATE CLASSIFICATION

The SASWAT system classified updates based on how the update affected the page (content inserted, removed, or replaced) and how it was initiated (automatically, or as a result of action by the user). While classification can be performed in fundamentally the same way in this approach as the SASWAT system, there are limitations imposed by performing it within the page rather than integrated within the screen reader. The first research question was to determine whether classification is possible in this more limiting environment.

3.1 Approach

Update detection occurs in two phases. First, DOMMutationEvents are passed through two filters: the first removes those that result in changes that are not visible to the user (e.g., the node has no content that can be spoken); the second tests whether the node is from a list of regions that the user has specified to be ignored). The second phase is to group together (or ‘chunk’) the DOMMutationEvents that are generated so that each group represents what would have been perceived by the user as a coherent, discrete change.

To ensure all DOMMutationEvents from an update are collected together, filtered events are buffered for 500ms before updates from elements that neighbour in the DOM are grouped. Finally, removals and insertions from the same region are paired as a replacement, before a final filtering process removes duplicates. This completes classification by update effect.

![Figure 1: An overview of the processes performed by the injected code.](http://www.alexa.com/topsites/countries/GB)

The second axis of classification depends on how the update was initiated. It is not possible for injected JavaScript to follow exactly where the user’s attention is focused, so it is necessary to monitor for mouse and keyboard events, and use these to estimate what the user is doing. ‘Click’ events allow us to monitor much of the important user input, and are analysed to determine the type of element the click occurred on: external link, internal link, or JavaScript link. Keystroke analysis is more complex, but by detecting which key was pressed, and where the user’s focus was when it was pressed and at the time of the update, simple heuristics allow the user’s activity to be classed one of the following: typing in an input field; navigation around the page; Escape pressed; interaction with a Widget; form submission; typing outside a form, and; idle. While these do not cover all types of activity, and are fairly loosely defined, it is anticipated that this information is sufficient for correct classification of the update.

The update detection and classification system was clearly limited in capability compared to the one that was integrated into the screen reader. It was therefore necessary to perform a technical evaluation to determine the accuracy of the detection, filtering, chunking, classification system when used on real Web pages.

3.2 Evaluation

Twenty sites were used for the evaluation, selected from the Alexa top 100 sites for the UK, as reported on 25 Nov 2011\(^2\). Five sites were not suitable for the evaluation (e.g.,
a login was required); those used are shown in Table 1. The homepage for each was used, but note that these are time and user-dependent, so this is not exactly repeatable. To perform the evaluation, the detection and classification scripts were bundled into an extension for the Google Chrome browser. Each site was opened and interacted with in the browser while recording JavaScript console output; this was examined during and after browsing to determine how ACup had chunked and classified the updates. Table 1 summarises the nature of the dynamic content on each page, and how each was classified; details are given in [2].

<table>
<thead>
<tr>
<th>Site</th>
<th>ASL</th>
<th>Tab</th>
<th>SS</th>
<th>Replace</th>
<th>Ins</th>
<th>Rem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Youtube</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>eBay</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IMDB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>n/a</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Yahoo!</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Amazon</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Twitter</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Blogspot</td>
<td>n/a</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Daily Mail</td>
<td>n/a</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guardian</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSN</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Wordpress</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Apple</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IMDB</td>
<td>✓</td>
<td>n/a</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Telegraph</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flickr</td>
<td>✓</td>
<td>n/a</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HSBC</td>
<td>n/a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft</td>
<td>✓</td>
<td>✓</td>
<td>n/a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Summary of the evaluation results by site and update type (auto-suggest list, tabbed region, slide-show or carousel, or generic content replacement, insertion or removal). A ‘✓’ indicates that ACup gave correct classification and chunking; an ‘x’ indicates that this was not the case; ‘n/a’ indicates that no DOM mutation events were detected, and; blank indicates that no updating content of that type was found on the page.

In summary, classification by initiation was always correct, but the chunking of changes was occasionally poor. This could lead to replacement updates being presented as removal then insertion rather than as replacement; classification by effect was otherwise correct. Problems were encountered with auto-suggest lists on two sites (Google and Wikipedia), where updates involved a complex sequence of content and style modifications that ACup did not collect as a coherent update. There were also two cases (Daily Mail and Flickr) where the DOMMutationEvents were handled in the expected way, but did not match the visual changes to the page. This is thought to be due to either inadequate handling of changes in content styling, or to dynamic content being ‘pre-loaded’. The IMDB content replacement was handled broadly correctly, but the changes were grouped in the order in which they changed, which did not match the visual layout (or indeed the DOM). In addition to these 5 problem cases, there were 12 content instances on 9 sites where changes to the content did not result in changes to the DOM, suggesting that the apparent changes were modifications to content style.

While a wider range of pages would need to be tested to prove that the ACup approach can handle a sufficiently broad range of frameworks and techniques for presenting dynamic content, this set of 20 pages should cover many of the popular implementations for these types of content, and the evaluation does at least indicate that the technique is viable.

4. RE-ENGINEERING THE UI

When looking to modify how screen readers present dynamic content, update detection is only a background process: important, but only necessary to support new or improved user interfaces. The central aim is control over what information is presented to the user, and when.

4.1 Enabling a Modified User-Interface

We wish to modify the screen reader to present different information than it would by default. However, since there is no direct mechanism for a Web page, or associated scripts, to explicitly output information through the screen reader, modification of the screen reader’s default behaviour must be forced through an indirect route. The solution used in this case is a simple one. When the page is loaded, the script injects a snippet of HTML that contains an empty (and visually hidden) element that is marked as ARIA live. The content of this element can then be modified at any point, resulting in its content being spoken.

In addition to text output (as spoken by a speech synthesizer), the SASWAT system used non-speech sounds to notify the user about updates that it considered users were less likely to want to attend. In this implementation, HTML5 snippets are injected into the page that point to sound files. The script can then play the files on demand; the current implementation has two sounds available.

4.2 User Interface Control

Thus far, this paper has described a straight port of the SASWAT user interface from the prototypical screen-reader browser add-on to a script-based system that may be run in a variety of ways, including through a proxy, or as a browser extension. This has required overcoming some technical challenges and has resulted in a system that is novel in the way in which it injects WAI-ARIA code at browse-time to result in more accessible pages. This is, however, of limited use, particularly in the long-term, leading one to ask the final research question: how generalisable is this approach?

ACup currently allows the style of presentation to be controlled for each of six possible classes of update. As with the SASWAT prototype, updates are classified along two axes, first according to whether it was initiated by the user or occurred automatically, and second according to how the page changed (insert, remove or replace).

Three aspects of the presentation can be set: Notification — a non-speech sound to be played following an update (this may be null); Announcement — a string to be spoken for each update (may be empty), and; Content — how much of the new content to speak. This can be one of: None; no content is spoken; First sentence: the first sentence of the new content is spoken, or; All: all the new content is spoken.

For example, we may specify that all automatic updates
(with any effect) are to be presented only with a short non-speech sound, while user-initiated content insertion is spoken immediately following a ‘content inserted’ announcement. This approach not only allows control over how each update is presented, but can be extended for finer control, e.g., by applying a deeper classification system or refining by site or by page. Importantly, the rules can be updated in response to new findings in HCI, so that users can benefit from the latest research.

5. DISCUSSION

We have developed a novel approach to improve the way in which screen readers present dynamic content on pages that do not contain WAI-ARIA markup. The concepts presented here have been implemented as a set of JavaScript functions that can be injected into Web pages and run in the client’s browser, e.g., through a proxy service or using a browser plugin. In this instance the implementation takes the form of an extension for the Google Chrome browser.

ACup injects an ARIA ‘live region’ into the page and uses this to push information to the screen reader. Coupling this with techniques to detect and classify page updates, allows updates to be presented to the user in a manner which has been shown to be more effective than the ways in which screen readers currently present them. Furthermore, the user-interface is defined in a flexible way, that allows the presentation style to be modified, either to user preferences, or to reflect further research developments.

Four research questions were enumerated in the introduction; they can now be answered as follows:  

1. Injected JavaScript is capable of update detection and classification; the accuracy is lower than when built in to a screen reader but does appear to be acceptable, especially for the crucial distinction between user-initiated and automatic updates.
2. Injecting an ARIA live region allows announcements to be ‘pushed’ to the AT when required.
3. Update classification is less robust than if built-in to a screen reader, and the processing required impacts on the responsiveness of the page. The exact levels of performance have not been measured, however.
4. Flexibility has been built-in that will allow the presentation of each class of update to be modified independently, and for the classification.

While the results were generally positive, questions remain. First, the technical evaluation of the classification system revealed some weaknesses, and highlighted the need for the ability to handle widgets that only modify the style of existing content. Second, an evaluation is required to properly assess the performance impacts of running the monitoring and chunking processes within the browser JavaScript engine. Third, while the classification and associated presentation styles have been previously shown to be effective [5], another evaluation involving users would give the opportunity to determine if this approach still gives the same benefits, and would allow the presentation to be tested over a wider range of sites and content.

The implementation as described uses a simple taxonomy of updates, but future work could extend this to use a finer-grained classification. One way of doing this would be to use code analysis to identify widgets [6] in the page before any updates occur, e.g., detect one region as being a set of tabs and another as a slideshow; this could allow an even more tailored presentation to be used, improving the user’s understanding of how they are interacting with the page and reducing confusion and disorientation.

6. ACKNOWLEDGEMENTS

The ACup project was funded by a Research Award from Google; we would like to thank them for their support.

7. REFERENCES