WAI-ARIA Live Regions and HTML5

Peter Thiessen
eBuddy B.V.
Amsterdam, The Netherlands
thiessenp@acm.org

ABSTRACT

The W3C Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) and HTML5 are exciting and relatively new specifications with many new semantics that together help describe the complex desktop like behavior found in many Web applications. One aspect of ARIA, Live Regions, define markup that an Assistive Technology can use to understand how to treat a Document Object Model (DOM) update. Past work has been done showing live regions effectively expose DOM updates. However, little testing has been done on the combination of HTML5 elements with live region attributes. Test cases as well as the results of the test cases and vendor support are discussed in this paper.

Categories and Subject Descriptors

H.1.2 [Models and Principles]: User/ Machine Systems—*human factors, human information processing*; **K.4.2 [Computers and Society]**: Social Issues—*assistive technologies for persons with disabilities*

General Terms

Human Factors, Design, User Agents

Keywords

Accessibility, Usability, Web 2.0, AJAX, ARIA, User Agents, Blind Users, Dynamic Content, Screen Readers.

1. INTRODUCTION

The W3C Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) [6] is an exciting and relatively new specification that allows a developer to add semantics that help describe the complex desktop like behavior found in many Web applications. A section of ARIA, live regions, deals with describing DOM mutations. For example, adding an `aria-live=polite` attribute to a Web stock ticker widget would allow any stock updates to be exposed to an Assistive Technologies (AT) and be announced to the user. Throughout the development of ARIA, live regions have undergone several changes.

WAI-ARIA became a draft in September 2006 originally with an XML Name Space (NS) that allowed developers to extend and customize the pre-defined roles. For example, a live region attribute would be define as `aaa:live="polite"`. At this point, live regions had in increasing priority: off, polite, assertive, and rude properties. Additionally, a channel property allowed developers to map live region updates to different AT devices. In 2008, based on user feedback, it was decided that a majority of Web developers were not comfortable with XML and HTML attributes should be used instead. At this time, a live region attribute would be defined as `live="polite"`. In 2009, the ARIA specification added an aria prefix to many elements, live regions included. Fore example, a polite live region is currently defined as `aria-live="polite"`. Additionally the live region `rude` property was removed as well as the channel property in an effort to help simplify the number of live region settings. More recently WAI-ARIA advanced to become a Candidate Recommendation (CR). As a CR, developers are now encouraged to use ARIA in their Web applications and can expect few future changes to the specification.

During the development of WAI-ARIA, an up and coming specification was also being developed, HTML5. The beginnings of HTML5 began outside of the W3C but in 2007 the Web Hypertext Application Technology Working Group (WHATWG) began work on the W3C HTML5 specification. HTML5 adds many exciting new elements and attributes that help better describe a Web page or application. Additionally the specification includes instructions for how vendors should implement new technologies such as geolocation, web workers, local storage, and so on. A growing concern among the ARIA community has been the semantic conflicts [8] between HTML5 and WAI-ARIA. The current strategy to deal with these conflicts has been to identify the conflicts, and outline HTML5 to ARIA mappings. The mappings can be found on the W3C HTML5 API Map site [7].

The HTML5/ARIA conflicts are of little concern to live regions specifically, as no related conflicts exist. Additionally, despite the many new technologies in HTML5, DOM mutation events will continue to be exposed to browsers and consequently AT the same as in HTML4. The reason for this consistency is the W3C DOM3 specification that maps out how events, such as dynamic insertions, should be handled by a browser. This means that the same dynamic Web page techniques for adding, removing and modifying content used in HTML4 can be confidently used in HTML5 documents. It is possible that in the future DOM mutation events will be handled differently in HTML5 as discussed on the WHATWG list [5] but as the specification now stands; dynamic insertions are treated the same way in HTML4 and HTML5.

Neither of HTML5 nor ARIA are finalized specifications but some vendors have begun supporting both specifications. At the time of writing this paper no browser fully supports HTML5. To better understand the current state of vendor support a set of test cases have been created and is the major contribution of this paper.

2. ARIA and HTML5 Support

As mentioned, neither ARIA nor HTML5 are a finalized specification. Some vendors however have become early adopters and are currently adding support for these specifications to their applications.

2.1 Browser Support

Both HTML5 and ARIA are individually supported by most recent browsers, or at least partially. The combination of the two specifications, as mentioned, poses problems where semantics conflicts exist. Both the Paciel Group [3] and the W3C [7] have created convenient tables that help identify how the conflicting semantics in HTML5 and ARIA can be mapped. With the mapping of conflicts identified, browsers "simply" need to add the related logic to their application code to expose this information to AT. Mozilla for example, has already begun adding these maps by adding landmark role maps [10] between HTML5 and ARIA to Firefox 4 (FF4). Consequently, FF4 will be the focus of the Test Cases discussed later in section 3 and 4.

As a side note, HTML5 has new elements that aid in accessibility. The Paciel Group has created a convenient table that identifies the related accessibility elements and the current state of browser support for these elements. The table is available at: http://www.html5accessibility.com.

2.2 Screen Reader Support

Screen reader support is even more limited than browser support for HTML5 and ARIA. Screen reader vendors however have a good excuse, screen readers depends on browsers to expose accessibility semantics. As mentioned, browser support for HTML5 and ARIA is only available in FF4. Information on what screen readers are early adopters of HTML5 and ARIA is limited and as a result will be left to manual testing and summarized in section 4.

2.3 Library Support

Some JavaScript (JS) libraries and more specifically JS User Interface (UI) libraries, add a layer of abstraction above manually adding ARIA syntax. This abstraction allows a developer to add a new widget to a Web application and have the related ARIA attributes automatically added to that widget. Additionally, the abstraction provides more convenient methods for setting and getting ARIA attributes. Currently JQuery UI [2] and Dojo Dijit [1] have ARIA support for many UI widgets. Unfortunately neither UI library currently has support for ARIA and HTML5 elements.

The JQuery UI progress bar widget for example, adds all the related ARIA attributes to an element but for that element to receive the default styling information it must be a `div` element. Replacing the `div` element with an HTML5 `progress` element and manually adding styling information can easily solve this problem. However, with the combination of semantics, ARIA and HTML5 attributes could potentially conflict. The HTML5 attribute `value` could conflict with the ARIA `aria-valuenow`, and the HTML5 attribute `max` could conflict with the ARIA attribute `aria-valuemax`. These conflicts could cause unexpected behavior in a browser.

Consequently, since no JS UI library has support for HTML5 and ARIA at the time of writing this paper, no library will be used or discussed in section 3 of the test cases.

3. TEST CASES

The test cases can be found at http://overscore.com/testcases and include the core basic live region features including nesting, prioritization, update relevance and atomic updates.

Past work on live region test cases has been by Charles Chen [4] and is available at http://accessibleajax.clcworld.net. For a more complete set of ARIA test cases beyond live regions, visit the iCITA ARIA examples site at http://test.cita.uiuc.edu/aria. Additionally, Jason Kiss has created a set of test cases specific to HTML5 and ARIA landmarks and is available at http://www.accessibleculture.org/research/html5-aria.

The scope of the test cases involve simple dynamic updates with live region properties that are designed to test whether the update is announced as expected. Each live region is contained within an HTML5 element that has a mapping to an ARIA landmark role as discussed in section 2.1. By combining live regions and HTML5 elements it will be possible to determine whether dynamic content is currently accessible in the latest available technology.

Nesting or inheritance is an important live region feature that allows developers to have child live region properties that either inherit or overwrite parent live region properties where the closest property to the update is given precedence. The nesting example takes a parent section element with a polite property, along with child a `nav` element `aria-live=off` attribute, and a second child `article` element with an `aria-live=assertive` property. Dynamic updates counting from 1 to 10 continuously occur in the two child elements. Only the child element with an assertive element should be announced. This example tests whether either child will incorrectly inherit the polite property from the parent.

The ability to prioritize live region updates is an important feature allowing a developer to customize how important an update is. The prioritization example has three article elements each with a different live region property of off, polite, and assertive. Three counters counting from 1 to 10 dynamically update the three live regions. This test is somewhat subjective but there should be enough updates to determine whether the assertive updates are announced before the polite updates and that the off updates are not announced.

Not all live region updates are desirable to be announced such as the removal of content. The `aria-relevant` attribute allows a developer to set the type of updates to be announced including text, removals, and additions. The relevant example has an assertive live region contained in a `section` element. A form with checkboxes is made available to allow a user to customize the relevant properties on the live region. Once the user clicks the Run Test button, the live region is populated with a roster of users, then a few seconds later the status text of each user in the roster is updated, and the test ends with the each user being removed from the roster. Only the related type of live region updates should be announced.

The surrounding text of a live region update is sometimes important and `aria-atomic` allows a developer to specify this. An `aria-atomic=true` property will treat updates to the live region as a whole and not just the updated area. The atomic example has two polite live regions contained in an `article` element, each with a different team and related score. A form with a true and false radio box allows a user to set whether the updates should be treated as atomic or not. If the user set atomic to true, then the team name and the new score for each update should be announced. If the user set atomic to false, then only the score should be announced.

For a complete set of ARIA specific test cases visit the CodeTalks set of ARIA test cases available at http://wiki.codetalks.org.

4. TEST CASE RESULTS

The test cases were against FF4 beta using a combination of Jaws 12, Windows Eyes7, and NVDA 2010. FF4 was chosen because as of its partial support for HTML5 and ARIA mappings. The mentioned screen readers were chosen based on their popularity. For each test case I followed the instructions on each test cases listening for the order of elements and updates announced or depending, not announced. The table 1 below gives a summary of the test results.

Table 1. Screen Reader Support with FF4 beta

	NVDA 2010	Jaws 12	Window-Eyes 7
Inheritance	partial	partial	fail
Prioritization	fail	fail	fail
Relevant	partial	partial	fail
Atomic	partial	pass	fail

The left most column identifies the related test case and the top row identifies the related screen reader for a run test case. No screen reader combination with FF4 fully supports the new HTML5 to ARIA mappings. These tests cases did not test whether the semantic mappings are accurate but whether an AT could simply understand a live region update within the new HTML5 elements. Also note that the table appears to be pointing solely to an AT for a pass/fail but rather; the table should be read as the combination of FF4 with that screen reader has a pass/partial/fail result.

Both NVDA and Jaws announced a majority of the dynamic updates regardless of the live region settings. Both screen readers had issues with nesting, prioritization, and the relevant property. In the nesting test cases, neither AT took the closest defined aria live region property but instead inaccurately kept the parent `aria-live=polite` property. The prioritization test case caused both NVDA and Jaws to fail. The updates would only be announced if the `aria-live` attribute was moved from the child element to the parent element. The aria-relevant test case caused both screen readers to fail when announcing removals of elements. Other combinations with text and additions were successfully announced. A note to developers: be careful to trim any trailing white space within the aria-relevant value as this caused AT to ignore the related properties. For the `aria-atomic` test case, only Jaws successfully announced the entire region where an `aria-atomic=true` update occurred.

Windows Eyes 7 did not pass any of the test cases.

The take away from the test results should be that support for HTML5 and ARIA live regions is currently at an early stage. However, all the needed technology in place and the specifications are ready for early adopters to begin developing accessible HTML5 applications.

5. CONCLUSION

The test cases cover live region settings and are intentionally simple to help isolate whether basic updates are announced by vendors. Additionally the HTML5 elements used in the test cases are restricted by the currently available HTML5 to ARIA mapped elements.

More advanced test cases are in the works that will focus on heuristics. One example is a chat test case that intentionally overloads an AT with different priority updates. This test will help determine how well the ARIA specification and AT handle floods of information updates. Another test case will be an automated form input with new HTML5 form elements/attributes that simulates a user entering data with error message updates. This test case will help test the accessibility of the new form elements.

As mentioned in the results of section 4, the test cases show the combination of HTML5, ARIA, browsers, and screen readers are at an early stage of development. The vendor combinations resulted in most test cases failing. Though, this failing of some properties should not deter adoption of ARIA live regions as basic DOM updates were shown to continue to be announced accurately by AT. Hopefully developers will begin implementing accessible HTML5 applications with live regions and help reveal edge cases in the WAI-ARIA 1.0 CR specification.

Internet Explorer 9 was ignored in the test cases but should not be ignored in the future when running test cases. IE9 RC was recently released and is a more standards compliant browser than previous versions of IE. For example IE9 currently supports the W3C DOM event model that will simplify cross-browser development. It is hard to say whether or not IE9 will follow a similar HTML5 to ARIA semantics mapping strategy to FF4. Regardless, the future of accessibility support in IE9 looks very promising.

6. ACKNOWLEDGMENTS

My thanks to eBuddy for supporting my work and having an open mind about accessibility in a highly competitive market. I also thank David Sloan for helping me come up with the idea for this paper.

7. REFERENCES

[1] Dojo Toolkit. “Rich UI Widgets”. 18 February 2011. <http://dojotoolkit.org/widgets>

[2] JQuery User Interface. 18 February 2011. <http://jqueryui.com>

[3] Paciel Group. "HTML5 Accessibility Support Work Arounds". 18 February 2011. <http://www.html5accessibility.com/index-aria.html>

[4] Thiessen, Peter, Chen, Charles. “Ajax Live Regions: Chat as a Case Example”. Proceedings of the 2007 international cross-disciplinary workshop on Web accessibility (W4A), 2007.

[5] WHATWG Public Mailing List. “public-webapps@w3.org from April to June 2009 by date”. 4 January 2011 <http://lists.w3.org/Archives/Public/public-webapps/2009AprJun>

[6] World Wide Web Consortium (W3C). “Accessible Rich Internet Applications (WAI-ARIA) 1.0: W3C Candidate Recommendation 18 January 2011”. 4 January 2011 <http://www.w3.org/TR/wai-aria/>

[7] World Wide Web Consortium (W3C). "HTML to Platform Accessibility APIs Implementation Guide". W3C Editor's Draft, 17 February 2011. 18 February 2011. <http://dev.w3.org/html5/html-api-map/overview.html>

[8] World Wide Web Consortium (W3C). “HTML5: Editor's Draft 18 January 2011: 3.2.7 Annotations for assistive technology products (ARIA)”. 4 January 2011 <http://dev.w3.org/html5/spec/Overview.html#annotations-for-assistive-technology-products-aria>

[9] World Wide Web Consortium (W3C). “HTML5: Editor's Draft 18 January 2011”. 4 January 2011 <http://dev.w3.org/html5/spec>

[10] Zeh, Marco. "New accessibility support for HTML5 elements and attributes". 18 February 2011. <http://www.marcozehe.de/2010/11/09/new-accessibility-support-for-html5-elements-and-attributes>