Building a WCAG2 Compliant Site with ARIA

Thomas M. Logan

Abstract – The Web Accessibility Initiative Accessible Rich Internet Applications (ARIA) specification is nearing completion [1]. Any nascent web standard will struggle with inconsistent implementations. Consistent and accurate documentation of best practices is critical in ensuring that a new format becomes adopted broadly and consistently. This paper explores the developer perspective of creating an accessible web application. Accessibility APIs have failed in the past because of the lack of documentation and clear guidelines. Real world examples have been requested from developers across all fields to understand how accessibility is supported.

In general, accessibility issues with rich internet applications can be characterized as:

- Providing the semantic structure of page areas and functionality (e.g., navigation, main content, search, etc.)
- Allowing certain non-focusable page elements to receive keyboard focus (e.g., setting focus to an error message within the page)
- Providing keyboard and screen reader accessibility with complex widgets and navigation elements (e.g., sliders, menu trees, etc.)
- Maintaining accessibility of content that is dynamic and may change within the page (e.g., AJAX content updates)

ARIA provides solutions to each of these problems. It allows developers to specify a role for page areas and elements, such as navigation, main content, and search. This allows quick access directly to these page areas. In web content, typically only links and form elements can receive keyboard focus or will allow keyboard interactivity. With ARIA, the author can allow any page element to become interactive with the keyboard. This allows greater levels of accessibility and interactivity for keyboard and screen reader users. The developer can specify if and how intrusive the notification for user will be when these areas are updated. ARIA provides accessibility support for dynamic, complex page widgets (interactive controls). Finally, ARIA also allows the creation of live regions, special areas of the page that can be updated dynamically.

I. Page structure

Proper structure of HTML content is essential in ensuring efficient navigation of large amounts of content. Section 508 contains a requirement for skip navigation links to allow a keyboard user to jump to the main content of the page [1]. The skip navigation requirement is an important accommodation for allowing a user to skip to the most important area of the page. There are many new web design patterns that benefit from providing quick access to additional common areas on the page. Developers of web content need to begin with thinking about what logical blocks of information exist on the page. Most commercial pages have a banner along the top that contains the logo. This can be identified by setting the ARIA role attribute to "banner". The main content should be contained within a div element that has the role attribute set to "main". This will enable an assistive technology to quickly jump to this location on the page. This will also allow the requirement of placing a skip navigation link on the page to be removed.

It is also important to note that more than one instance of a type of landmark role can be used on a page. The ability to tag multiple navigation bars on an application is a perfect example. One common design pattern is to have a top navigation bar and a left navigation bar. Marking this content up with ARIA will allow an assistive technology to efficiently jump between the two or more navigation regions on the page. Tagging the footer content of a page will enable it to be deconstructed from the presentation of the page. When using any of these roles, a strong visual border should be used to outline the content to assist in understanding the structure of the page through visual comprehension. Each region must have a proper heading that is identifiable for the content and unique.

II. Keyboard support

Keyboard support is of increased importance to the web as more complicated UI behaviors have become available. The original interfaces of the web are largely a point and click pattern. A user navigating a page with the TAB and ENTER key could effectively access everything. This is one reason that initial Section 508 requirements for the web did not include keyboard requirements [1]. Now that web components can be built from many different sub components it is more difficult to enforce a uniform keyboard model. It is best to follow a standard recommendation so that users will know how to interact with a particular piece of the user interface. A web application developer should work through their application using only the keyboard. This will illuminate the areas that need additional work from the keyboard. When deciding what keystroke should correspond with a particular action, the developer should follow the guidelines at the DHTML Style Guide to ensure a standard experience [3].

Web development projects now require the ability to differentiate between web content and web applications. A web application will require a comparable keyboard model to its desktop application counterpart. Assistive technologies are able to customize their commanding models

when finding the role attribute set to “application” on the body element of a web application.

Web developers can support accessibility best practices by choosing to use web controls from component developers that have implementations for ARIA and keyboard access. This saves developers a large amount of work and ensures that good accessibility support is rewarded.

III. Role and state information

Providing role and state information about a component is covered in the Web Content Accessibility Guidelines under Guideline 4.2, “For all user interface components (including but not limited to: form elements, links and components generated by scripts), the name and role can be programmatically determined; states, properties, and values that can be set by the user can be programmatically set; and notification of changes to these items is available to user agents, including assistive technologies[4].” These role and state requirements in WCAG2 can be met through supporting various ARIA attributes on elements. These requirements are difficult to comprehend due to every type of control requiring different specific pieces of information to be present. When the information is made available it is important to test with that a set of assistive technologies will be able to interpret it.

Automation plays an important role in ensuring compliance with a specification that contains as many requirements as ARIA does. By having automated test cases, a more complicated specification can have stronger adherence. Often authors strive to make their site as accessible as possible but find it difficult to interpret the vast requirements that exist in accessibility specifications. Without automated enforcement checks, bad coding practices become standard across applications. If assistive technologies need to enact specific workarounds to enable a particular application, other application authors will have to incorrectly implement a specification to function.

IV. Live regions

Live regions address the dynamic nature of modern web applications. A live region can be used to notify a screen reader that a specific update on the screen is important to be read. It can also be used to tell a reader not to automatically announce a particular update on the screen. It is fair to say that the majority of current web sites have areas where live regions can be made use of. Developers need the ability to provide a notification to a user that a particular action has successfully occurred. For example in the Queue Music[5] application, the author wishes to notify a user that a particular song is loading after the user presses enter to request an item to play. This can be achieved through using the aria-live attribute to mark the dynamic region that contains the status of the player.

In the Queue Music application, there is also a piece of content that should not be read automatically to the user. There is a timer value that indicates the current position in the song that the video player is currently playing. If this notification was announced to a reader every time it updated on the screen, there would be no opportunity to hear any other content. In these cases the web author should mark the region with the aria-live attribute set to”off”. Reader technologies may still access and present the value of the timer through navigation of the browser’s Document Object Model.

V. Conclusion

A common assumption of web developers in 2009 is that use of advanced JavaScript coding techniques is forbidden. With ARIA this assumption is fortunately no longer valid. ARIA techniques enable most web content to be made accessible. The challenge will be ensuring that best practices are enforced through documentation and automation. Good implementations of ARIA must be available through commonly used assistive technologies to ensure wide adoption of ARIA. For a company or institution to use ARIA they need to be able to see the solution working end-to-end with an assistive technology. Coordinated efforts are ongoing in 2009 between assistive technologies, corporations, and tool vendors to ensure that content best practices work reliably.

References

[1] Accessible Rich Internet Application (ARIA) 1.0. 18 May 2009. W3C. 18 June 2009 <http://www.w3.org/WAI/PF/aria/>.

[2] Section 508: Section 508 Standards. Access Board. 18 June 2009 <http://section508.gov/index.cfm?FuseAction=Content&ID=12#Web>.

[3] DHTML Style Guide. 13 May 2009. AOL LLC. 22 June 2009 <http://dev.aol.com/dhtml_style_guide>.

[4] Web Content Accessibility Guidelines (WCAG) 2.0. 11 Dec. 2008. W3C. 18 June 2009 <http://www.w3.org/TR/WCAG20/>.

[5] Queue Music. 1 May 2009. Thomas Logan. 22 June 2009 <http://www.queuemusic.org>.