WAI-ARIA

Qian Wan (qw13) COMP 531

1. INTRODUCTION

WAI-ARIA, the Accessible Rich Internet Applications Suite, defines a way to make Web content and Web applications more accessible to people with disabilities. It especially helps with dynamic content and advanced user interface controls developed with Ajax, HTML, JavaScript, and related technologies.

Web sites are increasingly using more advanced and complex user interface controls, such as tree controls for Web site navigation. To provide an accessible user experience to people with disabilities, assistive technologies need to be able to interact with these controls. However, the information that the assistive technologies need is not available with most current Web technologies.

Another example of an accessibility barrier is drag-and-drop functionality that is not available to users who use a keyboard only and cannot use a mouse. Even relatively simple Web sites can be difficult if they require an extensive amount of keystrokes to navigate with only a keyboard.

Many Web applications developed with Ajax (also known as AJAX), DHTML, and other technologies pose additional accessibility challenges. For example, if the content of a Web page changes in response to user actions or time- or event-based updates, that new content may not be available to some people, such as people who are blind or people with cognitive disabilities who use a screen reader.

So how does Assistive technologies work?

Assistive technologies use an API built into each operating system specifically designed to describe the roles, states, and structure of an application's user interface. For example, a screen reader uses this API to read the user interface with a text-to-speech engine, a magnifier uses it to highlight important or active areas of the screen, and an onscreen keyboard might use it to provide the most efficient keyboard layout for a given context or UI control. Assistive technologies often access a page's DOM as well through this API in order to understand the semantics and attributes of the page [5].

ARIA provides a bridge between the world of the DOM and the desktop. Browsers expose ARIA-enabled elements to the assistive technology API as if they were native widgets. As a result, the user potentially gets a more consistent user experience, where dynamic JavaScript-driven widgets on the web are comparable to their equivalents on the desktop.

2. USAGE

2.1. ROLES

ARIA roles work in many browsers and screen readers. When they don't, they are harmless. ARIA specifies that roles to describe the structure of the Web page, such as headings, regions, and tables (grids). Structural roles are added to markup as attributes of elements. In HTML5, for example:

```
<header role="banner">
<nav role="navigation">
```

Roles used in ways like the above example are navigation aids for machine readers and for assistive devices such as screen readers.
Roles that describe widgets are added to markup with additional information. The role in the example below identifies the element as a slider, with additional values using the aria prefix prepended to the property name. For example, in a slider widget that allows the user to select the day of the week, the values 1 – 7 indicate the days of the week. The first day of the week, number 1, is Sunday. The **aria-valuemin** and **aria-valuemax** restrict the options in the slider to 1 – 7.

```
<div role="slider"
aria-valuenow="1"
aria-valuemin="1" aria-valuemax="7"
aria-valuetext="Sunday"></div>
```

More roles usage can be found in [1].

2.2. KEYBOARD NAVIGATION

A primary keyboard navigation convention common across all platforms is that the tab and shift+tab keys move focus from one UI component to another while other keys, primarily the arrow keys, move focus inside of components that include multiple focusable elements. The path that the focus follows when pressing the tab key is known as the tab sequence or tab ring.

A typical example for this is the usage of tabindex. When using tabindex to manage focus in a composite UI component, the element that is to be included in the tab sequence has tabindex of "0" and all other focusable elements contained in the composite have tabindex of "-1". The usage demo is shown below [2]:

```
<div role="radiogroup"
     aria-labelledby="group_label_1"
     id="rg1">
    <h3 id="group_label_1">
      Pizza Crust
    </h3>
    <div role="radio"
       aria-checked="false"
       tabindex="0">
       Regular crust
```

```
    </div>
    <div role="radio"
       aria-checked="false"
       tabindex="-1">
       Deep dish
    </div>
    <div role="radio"
       aria-checked="false"
       tabindex="-1">
       Thin crust
    </div>
  </div>
```

2.3. LIVE REGION

In the past, a web page change could only be spoken in entirety which often annoyed a user, or by speaking very little to nothing, making some or all information inaccessible. Until recently, screen readers have not been able to improve this because no standardized markup existed to alert the screen reader to a change. ARIA live regions fill this gap and provide suggestions to screen readers regarding whether and how to interrupt users with a change.

A simple live region usually uses **aria-live** to make an impact on your web application. Normally, only aria-live="polite" is used. Any region which receives updates that are important for the user to receive, but not so rapid as to be annoying, should receive this attribute. The screen reader will speak changes whenever the user is idle. For regions which are not important, or would be annoying because of rapid updates or other reasons, silence them with aria-live="off".

For instance, imagine a website specializing in providing information about birds provides a drop down box. When a bird is selected from the drop down, a region on the page is updated with details about the type of bird selected. Then we can write our website code like [3]:

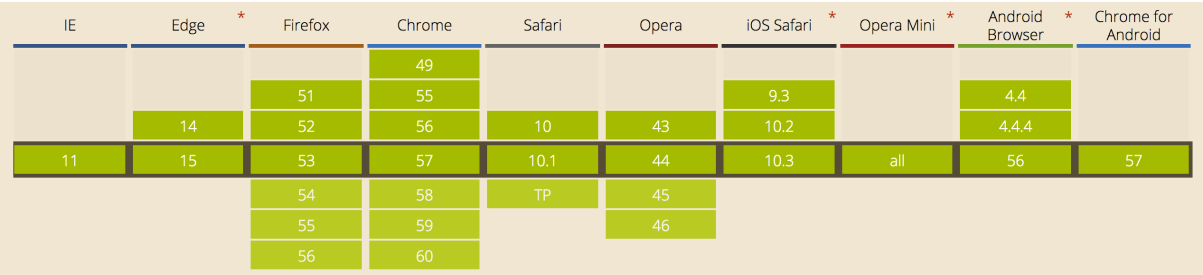
```
<select size="1" id="bird-selector" aria-controls="bird-
info"><option> .... </select>
<div role="region" id="bird-info" aria-live="polite">
```

As the user selects a new bird, the info is spoken. Because "polite" is chosen, the screen reader will wait until the user pauses. Thus, moving down the list will not speak every bird the user visits, only the one finally chosen.

3. CURRENT TECHNOLOGIES SUPPORT

3.1. BROWSER

The following figure shows all the supported browsers for ARIA [4]:



3.2. HTML 5 AND JAVASCRIPT

One common way we can use ARIA is by adding it to our HTML. We are familiar with semantic elements in HTML such as nav, button or header. It's quite easy to see what these elements would be used for. These elements give more meaning to the content of a page, and we can use a combination of these elements and ARIA in our markup. It's like to add those ARIA elements in your webpage. Remember, it's at least harmless and may benefit people who will view your web through assistive technologies like screen reader.

```
<div id="ads">
...
</div>
<div id="nav">
    <form id="searchform" ...>
    ...
    </form>
...
</div>
<div id="content">
...
</div>
```

The above is what your original HTML source code look like. After adding ARIA, it should look like below:

```
<div id="ads" role="banner">
...
</div>
<div id="nav" role="navigation">
    <form id="searchform" role="search" ...>
    ...
    </form>
...
</div>
<div id="content" role="main">
...
</div>
```

You can also separate the concerns by adding ARIA using JavaScript shown below:

```
function setupARIA()
{
    // Add ARIA roles to the document
    addARIARole('content', 'main');
    addARIARole('nav', 'navigation');
    addARIARole('searchform', 'search');
    addARIARole('ads', 'banner');
}

window.onload = setupARIA;
```

4. SUMMARY

WAI-ARIA is a technical specification that specifies how to increase the accessibility of web pages, in particular, dynamic content, and user interface components developed with Ajax, HTML, JavaScript, and related technologies. It is easily supported and has detailed documentation for developers [6]. You can adapt this whenever you are developing your web application. So don't hesitate to add ARIA to your web application now!

5. REFERENCES

[1] https://www.w3.org/TR/wai-aria/roles

[2] https://www.w3.org/TR/wai-aria-practices-1.1/examples/radio/radio-1/radio-1.html

[3] https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Live_Regions

[4] http://caniuse.com/#feat=wai-aria

[5] https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Web_applications_and_ARIA_FAQ

[6] https://www.w3.org/TR/wai-aria-1.1/