Accessibility Object Model: a new API for improved web accessibility

- Alice Boxhall, Google, Inc.
- James Craig, Apple, Inc.
- Dominic Mazzoni, Google, Inc.
- Alexander Surkov, Mozilla Corporation.

The primary standard used to make web apps accessible is WAI-ARIA (Web Accessibility Initiative Accessible Rich Internet Applications) [1]. ARIA is used today across the web to make everything from simple informational pages to complex web apps accessible to users with disabilities. However, there are some gaps in what's possible with ARIA, especially compared to what's available to native applications. Accessibility Object Model (AOM) is an experimental new web standard from Apple, Google, and Mozilla that lets web apps go beyond ARIA to have more advanced accessibility, like native apps. In this paper we'll describe some gaps in ARIA, the solutions provided by AOM, and a roadmap for adoption and standardization.

Introduction

While there are a number of important aspects to web accessibility, in this paper we limit our scope to support for assistive technology. Assistive technology, in this context, refers to a third party application which augments or replaces the existing UI for an application. One well-known example is a screen reader, which typically replaces the visual UI and pointer-based UI with speech and/or braille output, and keyboard and/or gesture-based input.

Assistive technologies interact with applications via accessibility APIs, such as UI Automation on Windows, or UIAccessibility on iOS. These APIs allow an application to expose a tree of objects representing the application's interface, typically with the root node representing the application window, with various levels of grouping node descendants down to individual interactive elements. This is referred to as the accessibility tree.

An assistive technology user interacts with the application almost exclusively via this API, as the assistive technology uses it both to create the alternative interface, and to route user interaction events triggered by the user's commands to application.

When the user is browsing the web, assistive technology communicates with the web browser via platform-specific accessibility APIs, and it's the web browser's job to convert the web page into an accessibility tree that the assistive technology can access, and relay commands from assistive technology back to the web page.

Native HTML elements are mapped to accessibility APIs. For example, an

`<img>` element will automatically be mapped to an accessibility node with a role of "image" and a label based on the alt attribute (if present).

WAI-ARIA (Web Accessibility Initiative Accessible Rich Internet Applications, [1], and henceforth just *ARIA*) allows developers to annotate elements with attributes to override the default role and semantic properties of an element. For example, the following code changes the semantics of a generic `div` element and exposes it as a checked checkbox to assistive technology.

```
<div role="checkbox" aria-checked="true">...</div>
```

Web apps that push the boundaries of what's possible on the web struggle to make them accessible because the APIs aren't yet sufficient - in particular, they are much less expressive than the native APIs that the browser communicates with.

The Accessibility Object Model

The authors of this paper are proposing a new web standard called the Accessibility Object Model (AOM, [1]), to address gaps in ARIA and enable web applications to offer a more accessible experience.

This proposal is divided into four phases, which will respectively allow authors to:

1. Modify the semantic properties of the accessibility node associated with a particular DOM node,
2. Directly respond to events (or actions) from assistive technology,
3. Create virtual accessibility nodes which are not directly associated with a DOM node, and
4. Programmatically explore the accessibility tree and access the computed properties of accessibility nodes.

In the following sections we'll outline each of these phases.

Phase 1: Modifying Accessible Properties

This phase will make it possible to set accessibility properties of an HTML element using JavaScript instead of by setting an attribute. For example, here's the HTML for an ARIA checked checkbox.

```
<div role="checkbox" aria-checked="true">...</div>
```

Instead, it'd be possible to write the following code:

```
element.accessibleNode.role = "checkbox";
element.accessibleNode.checked = true;
```

One use case this solves is that Custom Elements [3] "sprout" attributes in order to express their own semantics. AOM allows custom elements to behave more like native HTML elements, where their accessibility behavior is encapsulated.

Furthermore, writing Accessible Properties would allow specifying accessible relationships without requiring IDREFs, as authors can now pass object references. Currently attributes like aria-describedby and aria-activedescendant take one or more IDREFs, like this:

```
<div role="listbox" aria-activedescendant="id3">
```

Accessible Properties would allow passing object references instead:

```
element.accessibleNode.activeDescendant = accessibleNode3;
```

Moreover, writing Accessible Properties would enable authors using Shadow DOM to specify relationships which cross over Shadow DOM boundaries.

While AOM and ARIA both affect the computed accessible properties of a node, and have the same vocabulary, they are separate interfaces. ARIA does not reflect to AOM. In the case where and AOM Accessible Property and the corresponding ARIA attribute have different values, we suggest the AOM takes precedence.

Phase 2: Accessible Actions

Accessible Actions will allow authors to react to user input events coming from assistive technology.

For example, a mobile screen reader user may navigate to a slider, then perform a gesture to increment a range-based control. The screen reader sends the browser an increment *action* on the slider element in the accessibility tree.

Currently, browsers support by accessible actions for native HTML elements. For example, a native HTML `<input type="range">` already supports increment and decrement actions, and a native HTML `input type="textbox"` supports actions to set the value or insert text.

However, there is no way for web authors to listen to accessible actions on custom elements. A custom slider won't react to increment and decrement gestures.

Accessible Actions gives web developers a mechanism to listen for accessible actions directly, by adding event listeners on an `AccessibleNode`. This is analogous to listening for user interaction events on a DOM node, except that the interaction event arrives via an assistive technology API, so it is directed to the accessible node first.

To implement a custom slider, the author could simply listen for `increment` and `decrement` events.

```
customSlider.accessibleNode.addEventListener('increment', function() {
  customSlider.value += 1;
});
customSlider.accessibleNode.addEventListener('decrement', function() {
  customSlider.value -= 1;
});
```

Here are some of the proposed actions that would be supported:

- `click`
- `contextmenu`
- `focus`
- `setvalue`
- `increment`
- `decrement`
- `select`
- `scroll`
- `dismiss`

Phase 3: Virtual Accessibility Nodes

This phase will allow authors to expose "virtual" accessibility nodes, which are not associated directly with any particular DOM element, to assistive technology.

This mechanism is often present in native accessibility APIs, in order to allow authors more granular control over the accessibility of custom-drawn APIs.

On the web, this would allow creating an accessible solution to canvas-based UI which does not rely on fallback or visually-hidden DOM content.

Constructing a node can be achieved simply by creating an `AccessibleNode` object directly rather than accessing one from a DOM element:

```
var virtualNode = new AccessibleNode();
virtualNode.role = "button";
virtualNode.label = "Play Game";
```

To place the new virtual node in the accessibility tree, you can append a child to another `AccessibleNode`.

```
document.body.accessibleNode.appendChild(virtualNode);
```

A few additional properties are needed in order for virtual accessible nodes to be complete: the ability to set its bounding box, and control whether it's focused.

Phase 4: Full Introspection of an Accessibility Tree

The **Computed Accessibility Tree** API will allow authors to access the full computed accessibility tree - all computed properties for the accessibility node

associated with each DOM element, plus the ability to walk the computed tree structure including virtual nodes.

This will make it possible to write programmatic tests that make assertions about the semantic properties of an element or a page. It would also make it possible to build browser-based assistive technology.

One of the biggest challenges for this phase is that the accessibility tree is not standardized between browsers: Each implements accessibility tree computation slightly differently. In order for this API to be useful, it needs to work consistently across browsers, so that developers don't need to write special case code for each.

We want to take the appropriate time to ensure we can agree on the details for how the tree should be computed and represented.

Next Steps

We are developing this spec as part of the Web Platform Incubator Community Group (WICG). Portions of the AOM have been implemented in Google Chrome and there are plans to implement it in Mozilla Firefox and in WebKit. These experimental implementations will allow web developers to test this API and build working demos and proofs of concept.

We welcome feedback from anyone in the web developer or accessibility communities.

For more details, see the current draft spec [2] and follow links there to file bug reports.

Additional thanks

Many thanks for valuable feedback, advice, and tools from:

- Alex Russell
- Bogdan Brinza
- Chris Fleizach
- Cynthia Shelley
- David Bolter
- Domenic Denicola
- Ian Hickson
- Joanmarie Diggs
- Marcos Caceres

- Nan Wang
- Robin Berjon
- Tess O'Connor

- [1] Diggs, J., Schwerdtfeger, R., Craig, J., McCarron, S., Cooper, M. (2016). Accessible Rich Internet Applications (WAI-ARIA) 1.1. W3C Candidate Recommendation, available from: https://www.w3.org/TR/wai-aria-1.1/
- [2] Boxhall, A., Craig, J., Mazzoni, D., Surkov, A. (2017). Accessibility Object Model (AOM). WICG Unofficial Draft, available from: https://wicg.github.io/aom/spec/
- [3] Denicola, D. (2016). Custom Elements. W3C Working Draft, available from: https://www.w3.org/TR/custom-elements/