The Fruit of the Accessibility Tree

Agenda

1. A little background
2. The browser path
3. The creation of the tree
4. Looking at the accessibility tree
5. Bring it on home

Why should we care about this junk

“A firm grasp of the technology is paramount to making informed decisions about accessible design.”

- Léonie Watson

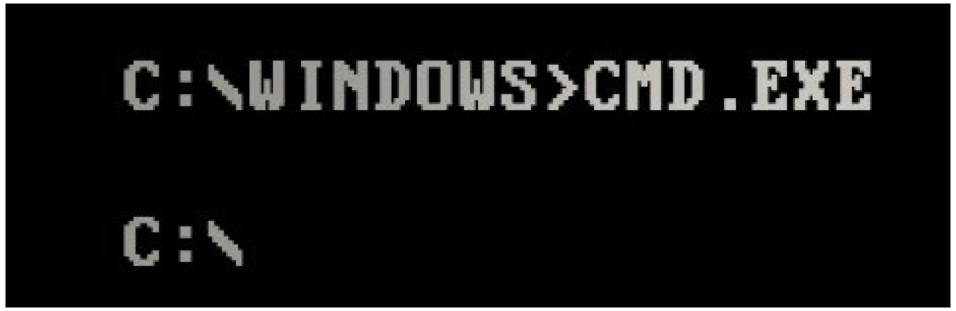
“As a web developer, learning the internals of browser operations helps you make better decisions and know the justifications behind development best practices.”

- *Paul Irish, Chrome Developer Relations*

A little background

When everything was text

- Personal computers originally relied on text based operating systems (e.g. DOS aka Disk Operating System).
- Assistive technology could access the text directly from the screen.



```
C:\WINDOWS>CMD.EXE

C:\
```

Image by Mega super editorman / CC BY (https://creativecommons.org/licenses/by/4.0)

Then came Graphic User Interfaces (GUI)

- Graphical Interfaces for operating systems become popular (e.g. Windows, Apple Macintosh) “draw” information on the screen.
- Assistive technology had to rely on heuristics to understand what was being presented to users.
- Accessibility **Application Programming Interfaces (APIs)** were introduced to operating systems to more reliably and accurately pass information to assistive technologies.
- As new browsers were introduced in the mid 2000’s they began integrating with the Accessibility APIs of different operating systems.

The Accessibility APIs

Windows:

- Microsoft Active Accessibility (MSAA)
- IAccessible2 (IA2)
- UI Automation (UIA)

Apple:

- NSAccessibility (AXAPI)

Linux:

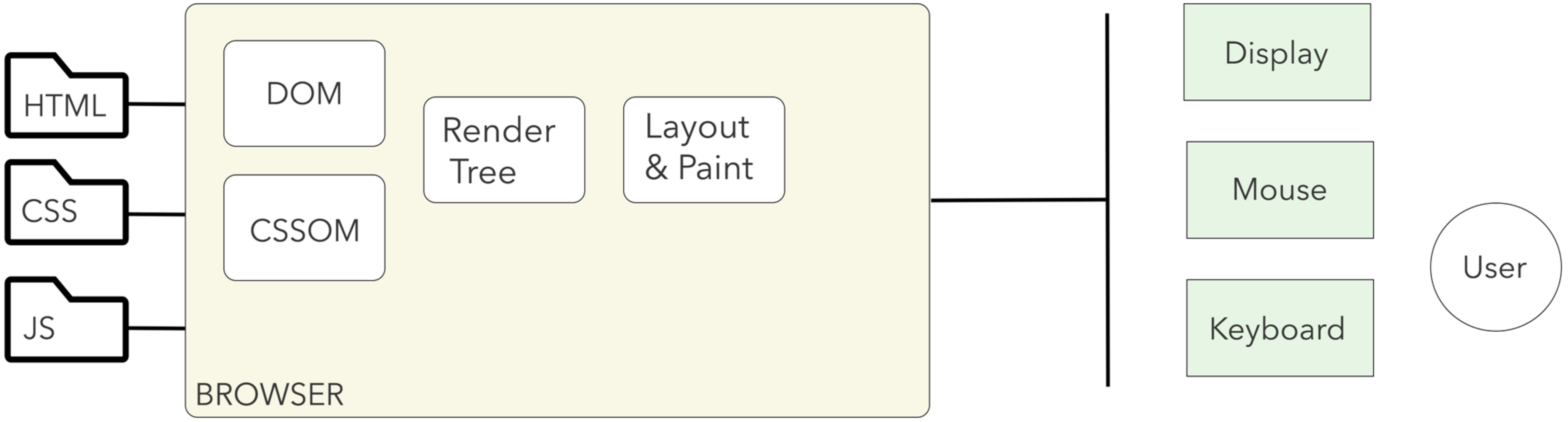
- Accessibility Toolkit (ATK)
- Assistive Technology Service Provider Interface (ATK-SPI)

Different paths: the browser

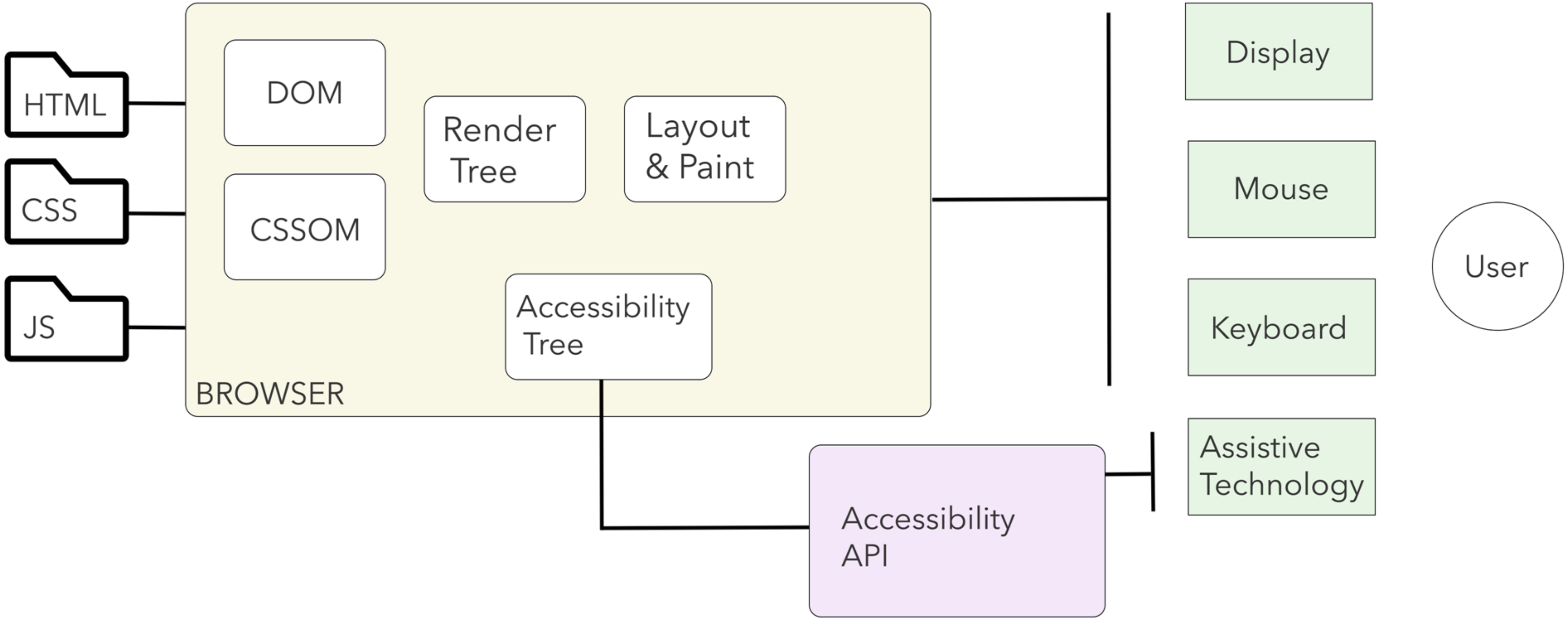
Browser breakdown

- Browser requests HTML/CSS/JavaScript resources
- Browser's render engine parses the HTML/CSS/JavaScript and creates the Document Object Model (DOM) from the HTML and CSS Object Model (CSSOM) from the CSS.
- Rendering Engine then creates a render tree based on the CSSOM and DOM which is the visual representation of the content.
- The page goes through a layout and painting process to place items on the screen.

User interaction with mouse/keyboard/display



User interaction with assistive technology



A different path

- The browser uses the “accessibility tree” to communicate with the operating systems Accessibility API.
- Assistive technology users interact with the browser through through the accessibility API.
- Two different and parallel “information architectures” to build and design for (although not separately).

The accessibility tree

The definitions

Browsers […] create an accessibility tree based on the DOM tree, which is used by platform-specific Accessibility APIs to provide a representation that can be understood by assistive technologies, such as screen readers.

-MDN Web Docs

The accessibility tree and the DOM tree are *parallel structures* […] Accessible objects are created in the accessibility tree for every DOM element that should be exposed to an assistive technology, either because it may fire an accessibility event or because it has a property, relationship or feature which needs to be exposed. Generally if something can be trimmed out it will be, for reasons of performance and simplicity.

– World Wide Web Consortium

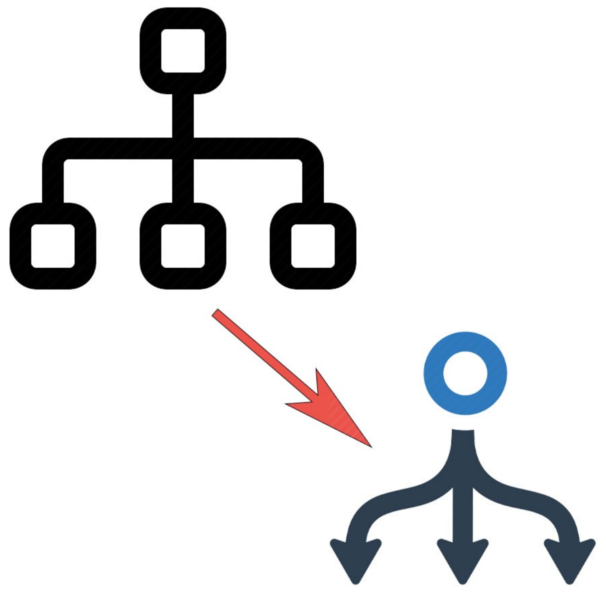
The DOM tree is translated, in parallel, into the primary visual representation of the page and the accessibility tree, which is in turn accessed via one or more *platform-specific* accessibility APIs.

- Accessibility Object Model Explainer

The browser takes the DOM tree and modifies it into a form that is useful to assistive technology. We refer to this modified tree as the *Accessibility Tree*.

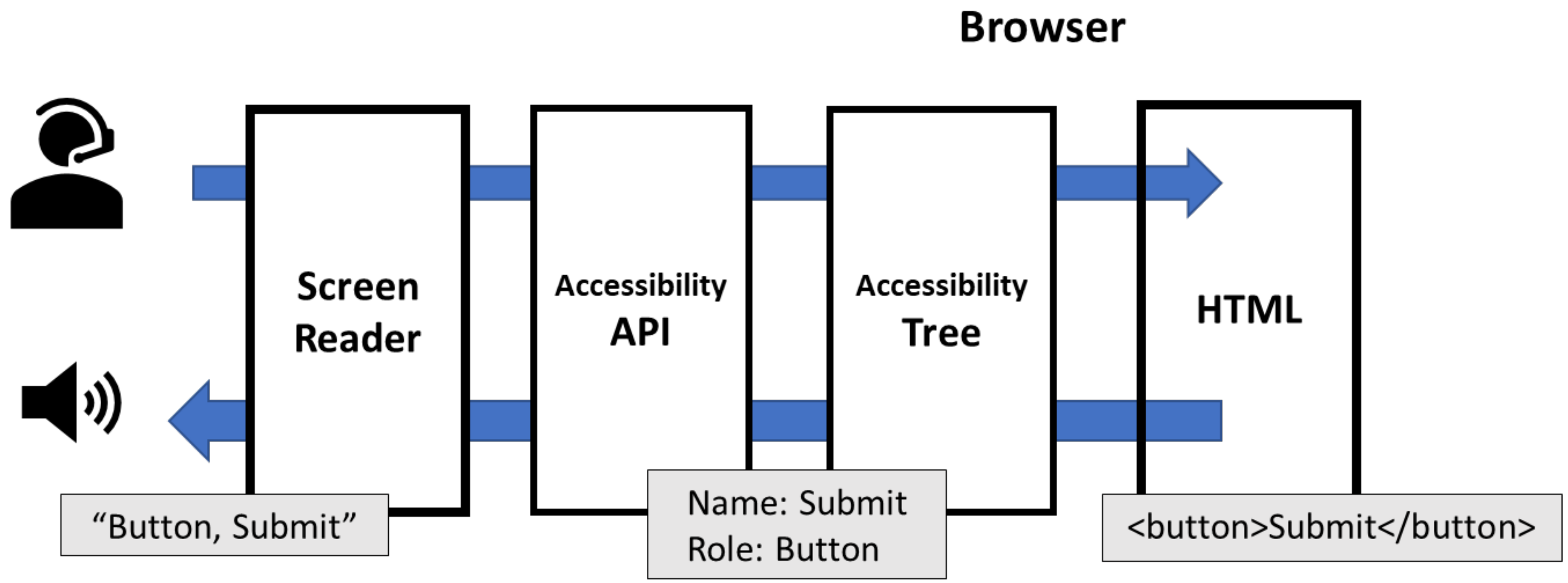
-Google Web Fundamentals

Parallel structures



- An accessibility tree represents the nodes in the DOM important to assistive technology.
- It is not a 1:1 copy.
- The accessibility tree provides a mechanism to communicate with a platform’s Accessibility API and augments the browsers default User Interface with assistive technology (e.g. screen reader)

Assistive Technology and the API



- Assistive technology sends request to Accessibility API.
- Accessibility API communicates to browser and accessibility tree.
- Browser updates web content and accessibility tree.
- Accessibility API communicates to assistive technology.

A closer look

[Simple Button demo on CodePen]

[HTML Accessibility API Mapping (W3C)]

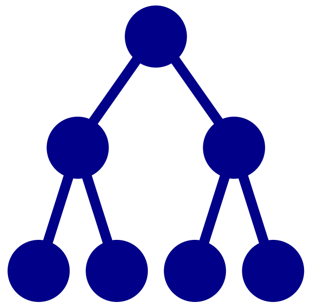
```html
<h1>The Button and the Accessibility API</h1>

<div>

  <p>
  Push the button to open the
    <a href="https://developer.mozilla.org/en-
US/docs/Web/API/Window/alert">alert dialog</a>.
  </p>
</div>

<button onclick="alert('Alerted’)” aria-label="“Alert
Announcement”>
         Alert Announcement
</button>
```

DOM and the accessibility Tree



```
heading name='The Button and the Accessibility API' role='heading' nameFrom=contents
  staticText name='The Button and the Accessibility API'
genericContainer ignored
  paragraph
    staticText name='Push the button to open the ' nameFrom=contents
    link focusable linked visited name='alert dialog' nameFrom=contents
      staticText linked visited name='alert dialog' nameFrom=contents
    staticText name='.' nameFrom=contents
button name='Alert Announcement' role='button' nameFrom=attribute hasAriaAttribute=true
  staticText name='Alert Announcement' nameFrom=contents
```

Accessibility tree and the API: `chrome://accessibility`

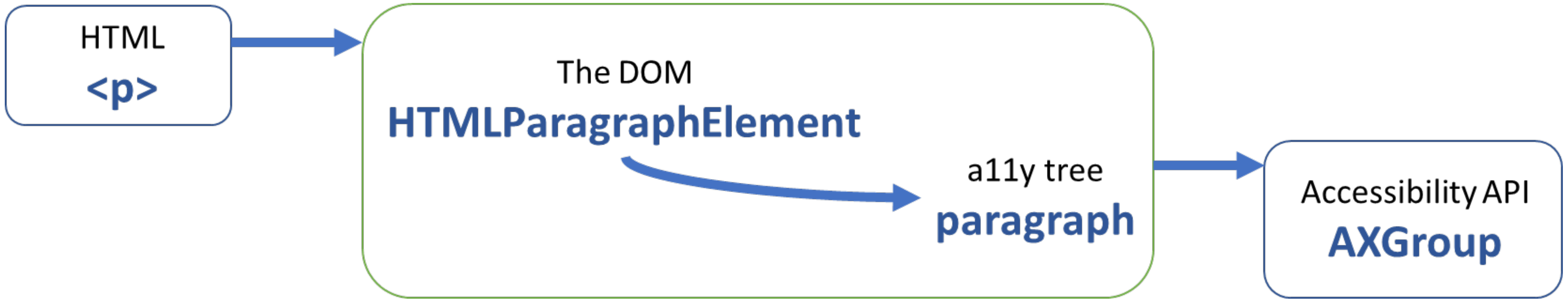
```
heading name='The Button and the Accessibility API' role='heading' nameFrom=contents
  staticText name='The Button and the Accessibility API'
genericContainer ignored
  paragraph
    staticText name='Push the button to open the ' nameFrom=contents
    link focusable linked visited name='alert dialog' nameFrom=contents
      staticText linked visited name='alert dialog' nameFrom=contents
    staticText name='.' nameFrom=contents
button focusable name='Alert Announcement' role='button' nameFrom=attribute hasAriaAttribute=true
  staticText name='Alert Announcement' nameFrom=contents
```

(NSAccessibility – AXAPI - MacOS)

```
AXHeading AXRoleDescription='heading' AXTitle='The Button and the Accessibility API' AXRole:
'AXStaticText', AXRoleDescription: 'text' AXValue: 'The Button and the Accessibility API' id='29'
++AXStaticText AXRoleDescription='text' AXValue='The Button and the Accessibility API' id='36'
++AXGroup AXRoleDescription='group', AXRole: 'AXStaticText', AXRoleDescription: 'text', id='38'
++++AXStaticText AXRoleDescription='text' AXValue='Push the button to open the ', id='40'
++++AXLink AXDescription='alert dialog' AXFocused=0 AXRoleDescription='link' AXHelp: '', id='41'
++++++AXStaticText AXRoleDescription='text' AXValue='alert dialog' id='43'
++++AXStaticText AXRoleDescription='text' id='42'
++AXButton AXDescription='Alert Announcement' AXRoleDescription='button' id='33'
```

The journey of a semantic element

1. The HTML paragraph element enters the browser from the network as **<p>**
2. The Browser parses the <p> to the DOM node as **HTMLParagraphElement**
3. The **HTMLParagraphElement** becomes **paragraph** in the accessibility tree
4. The Accessibility API (AXAPI – NSAccessibility) in translates this to **AXGroup.**
5. Voice Over reads the AXGroup as text.



[Mapping HTML to Accessibility APIs]()

Useful tools

Tools in the browser to expose the accessibility tree:

- Use the **inspector tool** (available in Edge, Firefox, Safari, or Chrome)
- Chrome also has: chrome://accessibility

Tools to expose the operating systems accessibility API:

Mac OS: Accessibility Inspector of Xcode

Windows: Accessibility Insights

Demos & Resources

Simple Button demo on CodePen

Button Patterns demo on CodePen

Core Accessibility API Mappings (W3C)

HTML Accessibility API Mapping (W3C)

What does it all mean?
(Bringing it home)

The gist

- The accessibility tree is a parallel structure to the Document Object Model (DOM).
- The accessibility tree is also an augmented User Interface to the browsers default interface via assistive technology (e.g. screen reader, Braille display.))
- Everything in the DOM and/or presented visually is not necessarily in the accessibility tree.

What can we do with this?

Design

- Plan ahead. Design for different “information architectures” - provide the blueprint for all user Interfaces (UI).

Development

- Understand how to communicate UI to Assistive Technology.
- Prepare for the future and the Accessibility Object Model (AOM)

What *else* can we do with this?

Debugging

- Removing the mystery of the black box that often sits between Assistive technology and the browser.
- Tools to investigate and understand possible breakdown between HTML (i.e. the DOM) and Assistive technology.

A few resources

- Making the GUI Talk, (1991) by Rich Schwerdtfeger
- A Brief History of Accessibility Support (2011) by Steve Faulkner
- Accessibility APIs: A Key to Web Accessibility, (2015) by Léonie Watson
- Semantics to Screen Readers, (2019) by Melanie Richards
- Accessibility Object Model Explainer – Appendices: Background (original commit 2017, last update 2020) Alice Boxhall (Google), James Craig (Apple), Dominic Mazzoni (Google), and Alexander Surkov (Firefox)
- How Browsers Work: Behind the scenes of modern web browsers (2011) Tali Garsiel and Paul Irish
- Gecko Reflow Visualization – Mozilla.org (2009) (YouTube video)
- Accessibility Fundamentals with Rob Dodson (2018) (YouTube Video) by Rob Dodson
- Ryan Seddon: So how does the browser actually render a website | JSConf EU 2015 (2015) (YouTube Video) Ryan Seddon

Thank YOU!

Everyone who “attended” for listening, or even just logging on.

Accessing Higher Ground and AHEAD for letting me speak.

Jamie Spear, Digital Accessibility Consultant @Harvard University for help putting the slides together.

Harvard’s Digital Accessibility Services Team for feedback and support.

Me, me, me!

Derek Jackson

Digital Accessibility Developer at Harvard University

derek_jackson@harvard.edu

A11y Boston Meetup