Fona: quantitative metric to measure focus navigation on rich internet applications

http://www.producao.usp.br/handle/BDPI/50725

Downloaded from: Biblioteca Digital da Produção Intelectual - BDPI, Universidade de São Paulo
The Web 2.0 brought new requirements to the architecture of web systems. Web applications’ interfaces are becoming more and more interactive. However, these changes are severely impacting how disabled users interact through assistive technologies with the web. In order to deploy an accessible web application, developers can use WAI-ARIA to design an accessible web application, which manually implements focus and keyboard navigation mechanisms. This article presents a quantitative metric, named Fona, which measures how the Focus Navigation WAI-ARIA requirement has been implemented on the web. Fona counts JavaScript mouse event listeners, HTML elements with role attributes, and TabIndex attributes in the DOM structure of webpages. Fona’s evaluation approach provides a narrow analysis of one single accessibility requirement. But it enables monitoring this accessibility requirement in a large number of webpages. This monitoring activity might be used to give insights about how Focus Navigation and ARIA requirements have been considered by web development teams. Fona is validated comparing the results of a set of WAI-ARIA conformance implementations and a set of webpages formed by Alexa’s 349 top most popular websites. The analysis of Fona’s value for Alexa’s websites highlights that many websites still lack the implementation of Focus Navigation through their JavaScript interactive content.

Categories and Subject Descriptors: B. [Accessibility]: Accessibility Design and Evaluation Methods; Accessibility System and Tools

General Terms: Design, Human Factors

ACM Reference Format:
DOI: http://dx.doi.org/10.1145/2812812

1. INTRODUCTION

The web’s first goal was to provide documents that are accessible through multiple technological architectures and distributed systems. However, today, many users access the web not only for documents in the strict sense but also for applications and...
services [Munson and Pimentel 2008]. This shift in usage scenario characterizes the next generation of web technology, the “Web 2.0,” which emphasizes the user as part of the content authoring process, greater interactivity, and a desktop-like experience in websites [Cooper 2007].

The greater interactivity and desktop-like experience provided by Web 2.0 applications are empowered by a set of technologies (DHTML, CSS, XMLHTTPRequest, and DOM Events, among others) that enable the development of complex interactivity on the web in the so-called RIAs (Rich Internet Applications). RIAs rely on scripting languages and other advanced technology in order to build sophisticated and highly interactive interfaces. However, some of these characteristics demand visual perception and require mouse/touch interactions to be perceivable by users [Gibson 2007]. Moreover, ad hoc development of these interfaces poses severe obstacles to accessibility [Velasco et al. 2008].

In this context, the WAI (Web Accessibility Initiative) proposed the WAI-ARIA (Accessible Rich Internet Applications) specification [W3C 2014a], which presents an accessibility framework for RIAs. WAI-ARIA specifies semantic markup attributes that help assistive technologies, such as screen readers; identifies widgets’ predefined behavior to users; provides keyboard navigation strategies that enable assistive technology users to interact with and between widgets in a webpage; and searches for HTML attributes that describe relationships for webpage elements; among other resources [W3C 2013].

Even though many RIA implementation initiatives support the WAI-ARIA specification, such as jQueryUI,1 Dojo,2 and YUI,3 there is no guarantee that these accessible design solutions will be implemented by web developers [Watanabe et al. 2010]. Thus, this article presents the elaboration of a quantitative metric, named Fona, which measures how web applications support the Focus Navigation requirement of the WAI-ARIA specification. We report on the development of a tool to automatically calculate Fona’s value for any RIA available on the web. In order to calculate the Fona metric, the tool analyzes HTML elements’ TabIndex attribute to manage focus navigation between widgets. One requirement for a webpage to support keyboard interaction is to allow focus to be set to any element that reacts to mouse events. The TabIndex attribute can be used to include these elements in the tab order and to set programmatic focus to them [W3C 2013]. Fona considers the number of keyboard focusable elements (which present TabIndex attribute equal to or greater than zero) and the number of HTML elements that handle mouse events to determine how well the Focus Navigation functionality is supported by a web application. We validate Fona comparing its results between WAI-ARIA example implementations from iCITA,4 which implement the Focus Navigation requirement, and Alexa’s most popular websites.5

Fona’s evaluation approach enables monitoring how Focus Navigation and ARIA requirements have been considered by web development teams in a large number of websites. This information might be used to show how successful the ARIA specification has been in disseminating the technological solutions for Focus Navigation requirements.

The article is structured as follows: Section 2 presents the WAI-ARIA specification; Section 3 describes the state of the art on accessibility metrics, WAI-ARIA development, and evaluation techniques; Section 4 details the Fona metric; Section 5 presents the tool that automatically calculates Fona for websites; Section 6 describes the evaluation

2http://dojotoolkit.org/.
4http://test.cita.illinois.edu/aria/.
of the metric; Section 7 provides a discussion about the validation/investigation results; Section 8 presents the conclusions of the study; and Section 9 presents future works.

2. AJAX AND WAI-ARIA

The Web 2.0 enabled web applications to share and aggregate data collected directly from users [Cooper 2007]. Considering specifically the end-user’s perspective, Web 2.0 applications presented greater interactivity and allowed users to customize the way information was presented to them. The content is constantly changed and updated in different types of web applications, such as games, plane ticket trackers, and online stores, among others.

Web 2.0’s greater interactivity is empowered by the implementation of open architecture design patterns known as Ajax [Garrett 2005] and RIA [Velasco et al. 2008]. These design patterns highlight two main characteristics for web applications: using scripting languages at the client side of web applications to handle rich user interaction mechanisms [Fraternali et al. 2010] while using asynchronous calls to the server side to keep the user interface data up to date [Munson and Pimentel 2008].

However, Ajax and RIA applications present severe accessibility barriers to users who interact with the web using assistive technologies [Munson and Pimentel 2008; Thiessen and Hockema 2010]. Ajax and RIA enable dynamic changes and updates in the DOM structure of a webpage, as illustrated in Figure 1. Figure 1 shows the changes made to the DOM structure in order to implement a tooltip widget. As the user interacts with a specific HTML element in the web application, the DOM tree structure is changed and a new HTML element is added to it. This new HTML element represents the tooltip message box and is presented to the user to provide complementary information about the HTML element that was previously interacted with.

On the other hand, traditionally, assistive technologies work with the premise that a webpage’s content can be linearized. However, Ajax and RIA break this assumption by allowing the inclusion of new content in the DOM structure of a webpage. Ajax and RIA frequently rely on the users’ visual perception, so that DOM structure changes are perceivable by users [Gibson 2007]. In the tooltip example illustrated in Figure 1, users interacting with the widget rely on visual clues to identify the complementary information that is presented to them. However, users interacting with these web applications through assistive technologies might not be informed of these changes.
RIA and Ajax design patterns are implemented using HTML and client-side programming languages, such as JavaScript. Therefore, priorly identifying every widget’s behavior based on source code analysis would be impractical for assistive technologies. Moreover, sequentially reading every event that is dispatched and every change that occurs in a webpage’s structure would overload users with possibly irrelevant information [Munson and Pimentel 2008].

For instance, during an interaction with a tab widget (JavaScript interface component that implements the interaction design pattern named Tabs\(^6\)), if users activate a tab, multiple events would be dispatched in the DOM structure. Tab patterns are frequently used to structure information and forms in web applications, as illustrated in Figure 2. Tabs present a horizontal/vertical row of selectable elements, the tabs, and each of these elements represents a section label. Each tab has an associated panel. There is only one visible/active panel, which is visually connected to its associated tab. As each tab is activated, its corresponding panel is activated/presented and all other panels are hidden. Therefore, if assistive technologies notified users of every event that occurred in the browser, users interacting with a tab widget through a screen reader would be simultaneously notified of every visibility change made to tab panels (showing and hiding panels) and layout changes made to the tabs. Moreover, these users would not perceive the association between the hidden/shown panels and tabs, which consists of information that is only presented visually.

It is worth noticing that RIAs implement desktop-like interactivity in web applications. Thus, design solutions that enable an accessible interaction in desktop applications might also be applied in RIAs [Thiessen and Hockema 2010]. Desktop applications rely on the use of widgets with a pre-established behavior, which is known to assistive technologies. Since these widgets present predictable interaction scenarios, assistive technologies are capable of informing users about relevant events and changes that are necessary to interact with the application.

However, desktop application design strategies cannot be directly applied in web applications. Web application widgets are implemented using generic markup elements; therefore, assistive technologies are not capable of identifying which widget behavior is implemented in the web application [Gibson 2007; Thiessen and Hockema 2010]. In this context, the WAI-ARIA specification presents an accessibility framework for

RIA [W3C 2014a]. This framework establishes a set of HTML attributes (roles, states, and properties) to be used by web developers. These attributes priorly identify widgets’ behavior in the markup elements so that assistive technology presents relevant information to users as changes are made to the webpage’s DOM structure. The WAI-ARIA specification is also part of the fifth major revision of the HTML specification (HTML5 specification [W3C 2014b]).

The WAI-ARIA HTML attributes that add semantic information for RIA widgets are described next [W3C 2014a]:

**Roles.** Roles identify the purpose of markup elements in the web application. Assistive technologies can use the role content to inform users about the behavior that is expected of the widget. Role attributes can be divided into four groups:

1. **Abstract Roles** compose a hierarchy structure of roles and can be extended by other types of roles.
2. **Widget Roles** represent interface components and should implement an interaction model that is recognized by users.
4. **Landmark Roles** consist of navigational landmarks and represent shortcuts for users searching for information on the webpage.

**States and properties.** Both attributes provide information about an object/widget. When combined with role attributes, the user agent monitors changes that might occur to these attributes. Thus, the information mapped in these attributes can be passed to assistive technologies, which might alert users about changes made to widgets.

The WAI-ARIA specification also addresses focus management design guidelines for web applications [Gibson and Schwerdtfeger 2005; W3C 2014a]. Some users rely solely on the keyboard as the navigation mechanism [Watanabe et al. 2012] and require that all functionality can be achieved using the keyboard [W3C 2008]. One of the most frequently used keyboard navigation strategies is the focus navigation using the TAB key in the web application. The focus navigation allows users to rapidly scan a webpage for interactive elements. When considering RIAs, which are characterized by complex interaction scenarios, focus navigation requirements become even more critical. Thus, the WAI-ARIA specification establishes that all parts of composite interactive controls need to be focusable or have a documented alternative method to achieve their function, such as a keyboard shortcut [W3C 2013].

As web developers map roles/states/properties descriptions in the structure of widgets and implement focus navigation in interactive elements that compose widgets, screen-reader users can benefit from RIA. For instance, a WAI-ARIA conformant tab widget must include elements with the following roles: tablist (elements that contain all tabs of the interface component), tab (elements that represent the interactive elements of the widget), andtabpanel (elements that are associated with each interactive tab element). The tab widget must also include the following states and properties markup: aria-labelledby for eachtabpanel element, containing the ID attribute of the associated tab element, and aria-hidden for eachtabpanel element, identifying if this panel is visible (activated) or not. Moreover, the active tab element must be made focusable (present the TabIndex attribute with value equal to or greater zero) in order to allow keyboard navigation scenarios in the widget. An example of a tab widget that implements all these specifications is illustrated in Figure 3.

A screen-reader user interacting with the previously described tab widget can interact with the widget as the active tab receives focus (since the active tab implements
Fig. 3. WAI-ARIA conformant tab widget HTML code.

**focus navigation requirements**). Once the active tab receives focus, the user would be notified that he or she is interacting with a tab (since the element has the `tab` role) and of the number of tabs that are available (since the tab element is inserted in an element with the `tablist` role). Then the user could navigate through the tabs using UP/DOWN/LEFT/RIGHT keys. As the user selects one tab to be active, he or she could move focus to the next interactive element of the web application and start interacting with the panel associated to the active tab. Then the screen reader would notify the user that he or she is interacting with a panel (since the panel has the `tabpanel` role), which is associated to the active tab (since the panel has the `aria-labelledby` attribute to identify the relationship between panel and tab).

In the tab widget example, if the widget does not implement the focus navigation requirements, screen-reader users would not be able to change the active tab. If the widget does not include the respective roles/states/properties attributes, users would not be notified of the type of widget they are interacting with. Thus, users might not recognize the interaction model that they need to use to interact with the widget. All these requirements must also be considered in the JavaScript code of the tab widget, since interactivity implemented via client-side scripts must also include mappings between the widget visual rendering state and WAI-ARIA states’ attributes.

In this context, in order to implement WAI-ARIA, a web developer must

—define which specific WAI-ARIA role map widgets are to be included in the web application;
—implement the WAI-ARIA role-associated attributes (states and properties) in each widget;
—implement the keyboard navigation strategies for each widget;
—include all the widgets in the web application;
—provide focus navigation between the widgets; and
—structure the widgets’ presentation inside the web application.

3. RELATED WORK
Web accessibility guidelines and specifications such as WCAG 2.0 [W3C 2008] and WAI-ARIA [W3C 2014a] are of the utmost importance in helping developers implement accessible web applications. However, they are not enough to fully support all steps conducted in web engineering practices [Freire et al. 2007].

Providing accessible web applications enables a more inclusive web and broadens access to everyone, regardless of disabilities [Freire et al. 2008b]. There are several reports regarding the elaboration of accessibility development approaches that guide the development process during the requirements elicitation, interface design, navigation design, and architecture design phases. Nevertheless, many studies have shown
that accessibility is still poor in many websites [Freire et al. 2008a; Goette et al. 2006; Hanson and Richards 2013].

Freire et al., for instance, presented a metrics-based approach to measure web accessibility in Brazilian municipalities websites [Freire et al. 2008a]. Monitoring the accessibility levels of websites is an important task to improve this attribute in web applications. Since web applications can be frequently updated, monitoring the evolution of accessibility requires accurate metrics in order to avoid having an inaccessible application deployed [Vigo et al. 2007; Freire et al. 2009].

Accessibility metrics frequently use guidelines, such as WCAG, to calculate a quantitative value that represents the accessibility level of websites [Vigo et al. 2007]. For instance, Sullivan and Matson [2000] proposed a proportion rate between the number of accessibility failures (considering WCAG 1.0 checkpoint verifications) and the number of potential accessibility barriers to calculate an accessibility metric value for a webpage. Other studies describe the inclusion of weight values for each accessibility failure and number of webpages that belong to the web application [Parmanto and Zeng 2005], define weight values based on experiments with users who present different disabilities [Cluster 2006], and consider the proportion rate between the number of similar failures and the total number of failures for each verification [Bühler et al. 2006], among others.

Manually monitoring accessibility levels of a variety of websites is especially costly in environments in which software is frequently updated and deployed [Watanabe et al. 2012]. Therefore, Freire et al. adapted an automatic evaluation tool (Hera) to identify accessibility barriers in websites and calculate their accessibility metrics [Freire et al. 2008a]. Even though automatic evaluations do not consider many aspects that should be manually evaluated, the results showed that, on average, webpages present a nondesirable medium to high level of accessibility barriers.

In another report, Freire et al. presented a survey on accessibility awareness of web developers in Brazil [Freire et al. 2008]. Identifying how developers perceive accessibility, how accessible their webpages are, and why they implement webpages that are accessible or not is critical to propose new approaches to boost web accessibility. The results showed that few people are aware of accessibility issues in web development.

It is worth noticing that both studies [Freire et al. 2008a, 2008] presented investigation results about how accessibility is considered in web development projects and focused on WCAG 1.0 [W3C 1999] recommendations. Thus, they do not cover WAI-ARIA-specific accessibility recommendations.

In regards to RIA accessibility, many studies propose automatic evaluation strategies for the dynamic changes in the DOM structure that are triggered via JavaScript, such as the following:

—Checking DHTML (Dynamic HTML) accessibility based on static JavaScript code analysis [Tateishi et al. 2007]. The Takaaki et al. approach uses a set of rules that identify a widget element’s attribute values (role/states/properties). The rules represent a state diagram, where the states represent a widget element’s valid state and the transitions represent an event that can be dispatched in that state. Tateishi et al. describe the use of these rules to automatically validate any widget’s elements by analyzing the JavaScript code that make changes to the attributes of this widget.

—Identification of differences between HTML static code and rendered DOM structure accessibility evaluations [Fernandes et al. 2011]. In this work, Fernandes et al. present a comparison between command-line evaluation, based on the analysis of HTML code retrieved in a single HTTP request, and in-browser evaluation, based on
on the analysis of the DOM structure as it is rendered in a browser. Fernandes et al. argue that the DOM structure in a browser is affected by client-side scripts (like JavaScript) and thus might differ significantly from the original HTML, which is loaded in an HTML request. Moreover, the DOM structure that is rendered in a browser is the exact same environment that is used in user evaluation. Therefore, accessibility requirements should be evaluated in the DOM structure, not in the command-line environment.

—Separate analysis of changes made to the DOM structure of a document, as events are triggered in HTML elements [Fernandes et al. 2012]. In this work, Fernandes et al. use the in-browser environment described in Fernandes et al. [2011] to analyze WCAG 2.0 conformance in multiple states of the DOM structure of a web application. Their approach simulates clicks in the web application and analyzes the accessibility of the resultant DOM tree, considering the possibility that the DOM tree might have changed after the click event.

—Using acceptance test case scenarios for automatically testing RIA accessibility [Watanabe et al. 2012]. Watanabe et al. describe the elaboration of a tool that uses acceptance test cases to map screen-reader users’ usage scenarios. Then, these test cases are used to automatically test the DOM structure of a web application, identifying accessibility barriers that might be inserted in RIA.

—Design of RIA accessibility evaluation tool [Doush et al. 2013]. In this work, Doush et al. describe the elaboration of a conceptual framework to evaluate RIA accessibility. Their framework is composed of an RIA events controller that identifies HTML elements that might be interacted with, a web robot that dispatches real events in an application interface, the WAI-ARIA specification as a set of rules to be tested, an evaluator that will use the WAI-ARIA specification to test DOM-rendered elements for accessibility, and a results handler.

All these studies work toward enhancing the web development process through automatic strategies that assist developers while testing their code for accessibility in RIA. However, none of them has the goal of providing insights about how the WAI-ARIA specification has been implemented by web developers. Moreover, both works from Fernandes et al. [2011, 2012] focus on the evaluation of WCAG 2.0 requirements. And Tateishi et al. [2007] and Watanabe et al. [2012] require a model as input (a set of rules for each widget and acceptance test cases, respectively) and thus cannot be generalized to evaluate every interactive element that is inserted in the web applications. Our study also tests a single WAI-ARIA requirement, the Focus Navigation, and synthesizes multiple websites’ analysis in a quantitative value to provide insights about how this requirement has been implemented on the web, differently than the conceptual framework proposed in Doush et al. [2013], which aims at assisting the development of a single WAI-ARIA.

Other tools, such as iCita Firefox Accessibility Addon (in its beta version) and FireEye, also assist web developers through their coding activities. These tools determine if the WAI-ARIA properties and states are correctly set given a specific role, validating whether the DOM structure follows the WAI-ARIA roles categorization. However, they do not evaluate whether behavioral WAI-ARIA requirements (such as Focus Navigation) have been implemented in the DOM elements or not.

In this context, this study has the objective of providing quantitative data about how WAI-ARIA has been implemented in popular websites by means of a quantitative metric that can be automatically calculated. The conducted study analyzes TabIndex and Role HTML attribute implementation in the DOM structure of webpages to present...
quantitative data about how the Focus Navigation WAI-ARIA requirement has been implemented in popular websites.

4. FONA: MEASURING FOCUS NAVIGATION

Focus Navigation is a mandatory accessibility requirement for RIAs [W3C 2014a]. Many users are not capable of interacting with the web using pointing devices, such as the mouse. In order to make the web accessible for these users, websites must implement keyboard navigation mechanisms for every functionality that is available, according to WCAG 2.0 guideline 2.1 [W3C 2008].

Even though WCAG 2.0 specifies technology-neutral guidelines for developers [Reid and Snow-Weaver 2008], they do apply to RIAs as well. The technology necessary for implementing WCAG 2.0 guideline 2.1 is addressed in W3C [2013], which states that keyboard support is an essential feature to Web 2.0 widgets. Moreover, widgets must implement full operation and functionality through keyboard-only events. However, unlike links, anchors, and form controls, which present inherent keyboard support, Web 2.0 widgets are frequently implemented using generic markup. Thus, widgets’ keyboard support must be manually designed and implemented by web developers.

Additionally, providing an effective navigation experience is critical for usability [W3C 2013]. And navigating within a great amount of interactive elements that compose each widget can be very tedious and characterizes an inconsistent behavior when comparing this navigation scenario with desktop interface components. Therefore, the WAI-ARIA specification describes that developers must implement navigation between widgets using the TabIndex HTML attribute, controlling the order in which each widget receives focus (as the user presses the TAB or SHIFT+TAB keys), while using arrow keys for navigation within a widget.

In this context, we present the quantitative metric, named Fona (Focus Navigation Assessment). Fona determines a quantitative value that represents the amount of interactive elements in a web application that are keyboard navigable according to the WAI-ARIA specification. Fona analyzes the TabIndex and role attributes of elements that present an associated JavaScript mouse event handler attached to them to check if these elements can be programmatically accessed via keyboard-only interactions. This metric’s goal is to investigate how focus navigation has been implemented on the web, providing insights about how WAI-ARIA development should be applied in the web engineering process.

Fona’s evaluation approach starts by analyzing all HTML elements that have any type of JavaScript mouse event listeners attached to them. Then, it checks how many of these JavaScript interactive HTML elements are inserted in the tab order of a webpage, checking if the TabIndex HTML attribute is greater than or equal to zero. If the elements that have JavaScript mouse event listeners attached to them are inserted in the tab order of a webpage, it means they might implement keyboard navigation mechanisms in the JavaScript interactive HTML element. If they are not inserted in the tab order of a webpage, it means they do not implement keyboard navigation mechanisms, unless these elements can be later inserted (through user interaction with other HTML elements) in the tab order programmatically or directly focused (using the focus method of DOM elements, via JavaScript, implementing WAI-ARIA-specific behavior for widgets). This verification step matches the WAI-ARIA specification that states that all interactive objects should be focusable [W3C 2014a].

It is worth noticing that some WAI-ARIA widget implementations do not have all their JavaScript interactive elements inserted in tab order, as the webpage is loaded. For instance, in the tab panel WAI-ARIA implementation,10 even though all tab role

10http://test.cita.illinois.edu/aria/tabpanel/tabpanel1.php.
elements have an attached mouse event listener to activate each tab, only one tab role element is inserted in the tab order (has a TabIndex attribute equal to or greater than zero). However, as the widget receives focus and a user interacts with it via keyboard events, the TabIndex attribute changes for all tab role elements and other tab elements receive focus. These elements receive focus considering a predefined user interaction behavior; therefore, they should be marked with a specific role attribute that describes this behavior. In regards to that, Fona’s evaluation approach also counts the number of HTML elements with JavaScript mouse event listeners that present any nonnull role attribute. If the element contains a nonnull role attribute, it means it might implement the behavior associated to the widget it represents. If the element does not contain a role attribute, it means it does not correctly implement the WAI-ARIA role specification. In regards to this behavior of the metric, WAI-ARIA widgets and landmarks (correctly marked with the role attribute) would not impact negatively on the metric. Other WAI-ARIA property/state attributes that might impact keyboard accessibility in HTML elements, such as aria-activedescendant, must be inserted in DOM elements that present a role attribute. Hence, Fona’s evaluation approach indirectly considers these attributes in WAI-ARIA widgets.

Considering the number of JavaScript interactive elements that both are inserted in the tab order and have a nonnull role attribute, Fona’s evaluation approach determines a quantitative value that represents the percentage of elements with mouse event listeners that are inserted in the tab order of a webpage or have a nonnull role attribute. This metric’s formula is as follows:

\[
Fona = \frac{\sum_{i=1}^{I} \min(T_i + R_i, 1)}{I}, \text{if } I > 0,
\]

where \( I \) is the number of mouse event listeners that are attached to an HTML element, \( T_i \) is a decimal value that indicates if the element with mouse event listener \( i \) is inserted in the tab order (1 if it is inserted in the tab order and 0 if it is not), and \( R_i \) is a decimal value that indicates if the element with mouse event listener \( i \) has a role attribute (1 if it has a role attribute and 0 if it does not). The metric’s values will range from 0 to 1 (100%). Fona’s values that are close to 100% report that the WAI-ARIA Focus Navigation requirement has been correctly implemented in almost every JavaScript interactive element of a webpage, while Fona’s values that are close to zero report that almost every JavaScript interactive element of a webpage is not keyboard accessible.

If \( I \) is equal to zero, the web application being evaluated does not present any JavaScript functionality that responds to mouse events. Thus, Fona is not applicable to this web application.

If Fona is used in a webpage with the following code fragment, which implements a set of tab elements inside a tab widget:

```html
<div onclick="change_tab(1)">Tab 1</div>
<div onclick="change_tab(2)">Tab 2</div>
<div onclick="change_tab(3)">Tab 3</div>
<div onclick="change_tab(4)">Tab 4</div>
```

then Fona’s calculation will result in zero, since all JavaScript interactive elements (all four DIV elements) of the code fragment cannot receive focus and do not present role attributes, indicating that the webpage does not implement a WAI-ARIA widget and cannot implement keyboard navigation scenarios.

On the other hand, if Fona is used in a webpage with the following code fragment, which also implements a set of tab elements inside a tab widget:
then Fona's calculation will result in 100%, since all JavaScript interactive elements of the code fragment can receive focus or present role attributes, indicating that the webpage implements a WAI-ARIA widget and might implement keyboard navigation scenarios. Moreover, widgets’ focus navigation can be implemented with no use of roles or explicit reference to the TabIndex attribute. For instance, a tab widget could be implemented using A (anchor) or BUTTON elements instead of DIV elements:

```html
<button onclick="change_tab(1)">Tab 1</button>
<button onclick="change_tab(2)">Tab 2</button>
<button onclick="change_tab(3)">Tab 3</button>
<button onclick="change_tab(4)">Tab 4</button>
```

Even though the previous code fragment does not implement WAI-ARIA role attributes, both these elements (A and BUTTON) present inherent TabIndex attributes set to zero, and thus this widget can receive focus and is keyboard navigable. The metric acknowledges this behavior and would also result in the value of 100%, which means that the previous code fragment correctly implements the WAI-ARIA Focus Navigation requirement (can be navigable via keyboard interactions), although it does not implement all other WAI-ARIA requirements.

It is worth noting that the metric does not guarantee that focus navigation is correctly implemented for an element with mouse event listeners. However, it does identify elements with mouse event listeners that are not inserted in the tab order and do not have a role attribute, and thus do not conform with the WAI-ARIA specification. Even though the metric does not contain many aspects that are necessary to guarantee that a web application correctly implements focus navigation for widgets, it does provide an upper-limit analysis of how this functionality is implemented in websites. And since the metric's variables (number of elements with JavaScript mouse event listeners, number of elements inserted in the tab order, and number of elements that have a nonnull role attribute) can be automatically calculated for any website with no domain restrictions (such as the study of Watanabe et al. [2012], which requires that developers implement acceptance test cases for every widget functionality), the metric can be used to collect quantitative data in a large number of websites.

The metric also does not provide an absolute value in regard to the number of elements with mouse event listeners that might implement an accessible keyboard-only interaction functionality. The metric consists of a percentage value that represents a fraction of the amount of JavaScript mouse event listeners that might implement an accessible interaction over the total number of JavaScript mouse event listeners.

5. AUTOMATICALLY CALCULATING FONA

We implemented a tool to automatically collect the metric from a group of websites.\textsuperscript{11} The tool uses CasperJS,\textsuperscript{12} a navigation scripting and testing utility based in

\textsuperscript{11}http://github.com/watinha/tabindex-counter.
\textsuperscript{12}http://casperjs.org/.

ACM Transactions on the Web, Vol. 9, No. 4, Article 20, Publication date: September 2015.
PhantomJS, a headless browser that renders the DOM structure using a Webkit-based rendering engine implementation and downloads any other external content (other JavaScript/CSS/Images resources, for instance) that compose the webpage (similarly to how Google Chrome and Safari load HTML content). This technology setup allows us to collect the metrics in a browser evaluation environment [Fernandes et al. 2011, 2012]. The tool waits for 10 seconds in order to allow the webpage to complete the execution of every JavaScript functionality it has implemented and to wait for possible DOM event handlers that could be set after the page is completely loaded. Then the script calculates the metric’s value.

The tool separately calculates the number of HTML elements with JavaScript mouse event listeners, the number of HTML elements inserted in the tab order, and the number of HTML elements that present a nonnull role attribute. After calculating each of these values, the tool combines them to provide the value of the metric described in the previous section.

In order to calculate the number of JavaScript mouse event listeners, the tool counted the number of calls made to the addEventListener method of HTMLElement objects, which include event listener callbacks to mouse events (click, mousedown, mousemove, mouseout, mouseover, and mouseup). The tool also counted the HTML elements that presented a callback function set in onclick, onmousedown, onmousemove, onmouseout, onmouseover, and onmouseup attributes of every HTMLElement object of a webpage. The sum of both values results in the number of JavaScript mouse event listeners that are attached to HTML elements in a webpage.

Figure 4 illustrates a schema that represents the tool’s main functionality. The tool receives the URL of a web application as input. This URL is accessed in a headless browser (inside the CasperJS environment). Then the metric’s value is calculated considering all mouse interactive elements in the input URL.

6. EVALUATION

In order to validate Fona, we conducted an investigation of how focus navigation has been implemented in two groups of web applications: a group of high-traffic RIAs, representing a sample of the web, and a group of WAI-ARIA implementations. The research questions that guided this investigation were the following:

---

13http://phantomjs.org/.
(1) Is the proposed metric enough to identify WAI-ARIA requirement implementations on the web?
(2) Is the proposed metric capable of identifying WAI-ARIA feature differences between websites that implement ARIA and websites that do not?
(3) Do web developers implement WAI-ARIA requirements associated to focus navigation?

In regards to Research Questions 1 and 2, we compared how the group of high-traffic RIAs and the group of WAI-ARIA implementations scores in Fona. Then we analyzed the data for the group of high-traffic RIAs to track how the Focus Navigation requirement is being implemented on the web (considering Research Question 3).

The software engineering research community has not yet reached a consensus on a standard set of rules to be validated for each metric elaborated. Fona’s evaluation used Meneely et al.’s criteria selection process to determine which criteria should be validated in our metric [Meneely et al. 2013].

Fona’s definition follows a goal-driven philosophy according to Meneely et al. [2013], in which Fona’s goal is to enhance the software engineering process by means of monitoring the accessibility of web applications according to one specific WAI-ARIA requirement (Focus Navigation). Fona stands as a quality-focused metric (Quality metric [IEEE 1990]) that provides a proportional rate of the degree to which a web application possesses Focus Navigation. Henceforth, the validation criterion conducted in this study consists mainly of Empirical Validity (which is a subcategory in the External Validity criterion).

The next section presents details about both groups of our study.

6.1. Sample Groups

This report made use of two groups of websites: the top 349 most popular websites according to Alexa (Subject group) and a group of 41 WAI-ARIA implementation examples from iCITA,14 the Illinois Center for Information Technology and Web Accessibility (Control group).

The Subject group consists of a group of websites extracted from Alexa’s list of most popular websites. Alexa is a web information company that provides free and global web metrics. These websites were collected on May 5, 2013, with a user agent (web browser) localized in Brazil. This group’s list contains the first 349 websites from Alexa, considering that many websites placed after the 349th ranking position were not available to Brazil (websites placed after 349th returned a 404 HTTP status message, meaning they were not found by the browser).

Alexa’s most-popular list contained a wide variety of websites. These websites were classified into the following groups:

Portal websites. Present constantly updated content and multiple services links. These websites’ goal is to provide a single webpage that rapidly presents the user with the most relevant information. Access to this information is publicly available, and no user credentials are required. These websites frequently presented news, sports, weather, stock plans, and vehicle feeds. Some of these websites presented login and search forms; however, these forms were hindered by the other information presented in them.

Ninety-seven websites were classified in this group (27.80%).

Company/Product websites. Present a company or product introduction webpage. These websites’ goal is to advertise companies’ or products’ portfolios. These

14http://cita.illinois.edu.
websites presented slideshow widgets, advertising videos, and contact information. This group's websites frequently advertised informatics companies, cloud infrastructure services, and security companies, among others.

**Eighty-six websites were classified in this group (24.64%).**

**Search websites.** Present a search text input element at the center of the webpage. These websites' goal is to provide a service for searching websites considering a specific parameter (set of keywords). Most websites of this group contained a single search text input, with a group of links to navigate to other services, like mail, messaging, online stores, and news portals, among others.

**Sixty-eight websites were classified in this group (19.48%).**

**E-commerce websites.** Present an online store that informs products' prices and advertises sales offers. These websites' goal is to invite users to buy products available in the store. The websites frequently presented fly-out menus\(^\text{15}\) to navigate through multiple product categories, product slideshows, and price information. Most e-commerce websites presented products related to informatics, electronics, books, and cars, among other topics.

**Thirty-two websites were classified in this group (9.16%).**

**Video streaming websites.** Present multiple videos to be watched online. The first representative website of this group, according to Alexa's ranking, is Youtube\(^\text{16}\) and most websites in this group present its same characteristics. This group's goal is to invite users to watch videos from the website. The websites from this group present lots of links with images, representing single frames and textual descriptions for videos. The websites' domain content ranges from soap operas to music video clips and entertainment in general.

**Twenty-eight websites were classified in this group (8.02%).**

**Login websites.** Present a single login form and the credentials of the user are required to access the main functionality of the website. These websites frequently implement social platforms, and thus users need to be registered to navigate through other users' profiles and published content. The login page presents a form and images/text that present the web application's goal to users.

**Twenty-five websites were classified in this group (7.16%).**

**403/404 websites.** Some web applications were forbidden or not found (returning 403/404 HTTP status messages to the browser). These websites presented textual information to explain why they were forbidden or unavailable (for the URLs that presented a customized 403/404 webpage) or a generic webpage for the forbidden or not-found resource.

**Nine websites were classified in this group (2.60%).**

**Forum.** Present a list of questions or notes to users. These websites' goal is to provide a service that allows users to comment on a topic to answer questions and allow any user (registered or not) to read the content.

**Four websites were classified in this group (1.14%).**

The number of websites for each group is illustrated in Figure 5.

The **Control group** consists of 41 websites extracted from the iCITA WAI-ARIA examples list. iCITA provides these examples in order to help web developers learn about the WAI-ARIA specification. These websites were collected on May 4, 2013, with a user agent localized in Brazil. iCITA is a study center in the University of Illinois and it promotes events, resources, best practices, and research in the area of web accessibility. In the iCITA website, there is a set of WAI-ARIA implementations


\(^{16}\) [http://www.youtube.com.]

ACM Transactions on the Web, Vol. 9, No. 4, Article 20, Publication date: September 2015.
to be used as reference implementations of other WAI-ARIA requirements. We used these WAI-ARIA implementations from the iCITA website as a sample of websites that correctly implement the Focus Navigation requirement.

From the 41 websites that formed the Control group, we removed four websites, because they implemented testing routines for focus navigation and presented widgets that do not match WAI-ARIA requirements (as bad examples of WAI-ARIA). The four webpages that were removed from the Control group were focus tests for button elements,\(^{17}\) input button elements,\(^{18}\) anchor elements,\(^{19}\) and div elements.\(^{20}\)

The next section describes the methodology of this investigation.

6.2. Methodology

In this investigation, we compare the top 349 most popular website scores (Subject group) with the group of WAI-ARIA implementation example scores (Control group) using Fona.

In regards to Research Questions 1 and 2, we established the first hypothesis to be tested in this investigation:

\(H1: Fona \text{ is capable of differentiating a group of websites that implement the WAI-ARIA Focus Navigation requirement from a group that does not.}\)

Comparing the two groups’ scores (the top most popular websites from Alexa and iCITA WAI-ARIA implementation examples) provides insights about Research Questions 1 and 2. If Fona’s value for the Subject group is greater than or equal to Fona’s value for the Control group, then our hypothesis \(H1\) is rejected by our investigation. Thus, the metric is not capable of differentiating websites that implement WAI-ARIA Focus Navigation and websites that do not. If, on the other hand, Fona’s value for the Subject group is lower than Fona’s value for the Control group, then our hypothesis \(H1\) is supported by our investigation.

\(^{17}\)http://test.cita.illinois.edu/aria/focus-tests/button.php.
\(^{18}\)http://test.cita.illinois.edu/aria/focus-tests/input-button.php.
\(^{19}\)http://test.cita.illinois.edu/aria/focus-tests/href.php.
\(^{20}\)http://test.cita.illinois.edu/aria/focus-tests/div.php.
Moreover, if the hypothesis $H_1$ is supported by the investigation, analyzing Fona's value for Alexa's separate groups of websites provides insights about how WAI-ARIA requirements have been considered in web application projects and how the WAI-ARIA implementation examples' keyboard accessibility strategies are being used on the web (Research Question 3).

In order to determine Fona's value for a website, we had to count the number of HTML elements that have a mouse event listener attached to them and the number of HTML elements with role attributes. In regards to that, we also collected these values with the goal of better characterizing both samples.

The next section presents the results of the comparison between both groups.

6.3. Results

The results are presented as a separate quantitative data report for each of these metrics: number of JavaScript mouse event listeners attached to HTML elements, number of HTML elements with role attributes, and focus navigation metric value.

6.3.1. Number of JavaScript Mouse Event Listeners. The Subject group, composed by 349 websites from Alexa, presents an average of 126.7 JavaScript mouse event listeners, and approximately 75% of this group present 14 or more JavaScript mouse event listeners attached in the DOM structure of the webpage. Figure 6 presents a histogram of the number of JavaScript mouse event listeners in the top 349 most popular websites from Alexa.

In the Subject group, the Portal and Video Streaming groups presented the highest mean values of JavaScript mouse event listeners registered in the webpages (254.2 for the Portal group and 245.2 for the Video Streaming group). The lower number of JavaScript mouse event listeners were identified in the groups of 403/404 (all webpages from this group did not present any listener) and Login (17.08 in average) webpages.

The Control group presented an average of 16.68 JavaScript mouse event listeners, and 75% of the websites presented six or more mouse event listeners attached to HTML elements.

All minimum, median, mean, maximum, and standard deviation (S.D.) values for each group are presented in Table I, and Figure 7 presents a boxplot representation for illustrating the distribution of the number of JavaScript event listeners for each group.

In the Subject group, 29 websites did not present any JavaScript mouse events (approximately 8.5%). It is worth noting that the fact that the 29 websites did not present any mouse event listener does not imply that the 29 websites did not have any JavaScript code in them. The tool did not analyze the content of each webpage; it
Table I. Distribution of JavaScript Mouse Event Listeners for Each Group of Websites

<table>
<thead>
<tr>
<th>Group</th>
<th>Min</th>
<th>Median</th>
<th>Mean</th>
<th>Max</th>
<th>S.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portal</td>
<td>0</td>
<td>118.0</td>
<td>254.20</td>
<td>2169</td>
<td>377.10</td>
</tr>
<tr>
<td>Company/Product</td>
<td>0</td>
<td>19.5</td>
<td>71.33</td>
<td>678</td>
<td>132.68</td>
</tr>
<tr>
<td>Search</td>
<td>0</td>
<td>28.5</td>
<td>34.26</td>
<td>232</td>
<td>39.89</td>
</tr>
<tr>
<td>E-commerce</td>
<td>0</td>
<td>64.5</td>
<td>104.47</td>
<td>643</td>
<td>119.46</td>
</tr>
<tr>
<td>Video</td>
<td>0</td>
<td>132.5</td>
<td>245.20</td>
<td>1627</td>
<td>321.48</td>
</tr>
<tr>
<td>Login</td>
<td>0</td>
<td>11.0</td>
<td>17.08</td>
<td>56</td>
<td>16.79</td>
</tr>
<tr>
<td>403/404</td>
<td>0</td>
<td>0.0</td>
<td>0.00</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Forum</td>
<td>21</td>
<td>71.5</td>
<td>115.80</td>
<td>299</td>
<td>127.98</td>
</tr>
<tr>
<td>Alexa's</td>
<td>0</td>
<td>35.0</td>
<td>126.70</td>
<td>2169</td>
<td>250.40</td>
</tr>
<tr>
<td>iCITA</td>
<td>0</td>
<td>13.0</td>
<td>16.68</td>
<td>64</td>
<td>16.80</td>
</tr>
</tbody>
</table>

Fig. 7. Boxplot illustrating the number of JavaScript mouse event listeners of all groups of the investigation.

only verified if there were JavaScript mouse event listeners being attached to HTML elements.

The 29 websites that did not present any JavaScript mouse events consisted of:

— **Unavailable/forbidden websites**: Seven websites (all websites from the 403/404 group) were forbidden/not available by the time our investigation was conducted. Some domain names are not available depending on the source of the HTTP - HyperText Transport Protocol request origin. This policy is applied to websites that intend to block access to users depending on the region in which their request is made. We believe that the seven websites that were forbidden/unavailable during the investigation were blocked due to this policy.

— **Simple login and search webpages**: Three websites presented a simple login or search HTML code that did not implement any JavaScript mouse event functionality.

— **News, advertising, and media content frontpage**: Nineteen websites presented a single HTML code with Flash content or a list of links that represented a portal initial frontpage.

The 29 websites with no JavaScript mouse event listeners were removed from this data analysis phase in the results, since they did not implement any kind of JavaScript mouse event listener functionality that characterizes RIA requirements. As a result, the Subject group and its subgroups were composed of 320 websites for the next result analysis phases.

One website from the Control group also did not present any JavaScript mouse event attached in its DOM structure. This webpage presented a WAI-ARIA landmarks example. Landmarks are used to improve the navigation of a webpage for screen-reader
Fig. 8. Number of HTML elements that are marked with roles on the top 349 most popular websites, according to Alexa.

<table>
<thead>
<tr>
<th>Group</th>
<th>Min</th>
<th>Median</th>
<th>Mean</th>
<th>Max</th>
<th>S.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portal</td>
<td>0</td>
<td>0.0</td>
<td>0.97870</td>
<td>24</td>
<td>3.44</td>
</tr>
<tr>
<td>Company/Product</td>
<td>0</td>
<td>0.0</td>
<td>5.23100</td>
<td>161</td>
<td>22.17</td>
</tr>
<tr>
<td>Search</td>
<td>0</td>
<td>0.0</td>
<td>0.07692</td>
<td>4</td>
<td>0.50</td>
</tr>
<tr>
<td>E-commerce</td>
<td>0</td>
<td>3.0</td>
<td>10.68000</td>
<td>83</td>
<td>20.92</td>
</tr>
<tr>
<td>Video</td>
<td>0</td>
<td>0.0</td>
<td>1.96200</td>
<td>37</td>
<td>7.20</td>
</tr>
<tr>
<td>Login</td>
<td>0</td>
<td>0.0</td>
<td>0.86360</td>
<td>56</td>
<td>1.45</td>
</tr>
<tr>
<td>Forum</td>
<td>0</td>
<td>1.0</td>
<td>24.25000</td>
<td>95</td>
<td>47.17</td>
</tr>
<tr>
<td>Alexa's</td>
<td>0</td>
<td>0.0</td>
<td>3.13400</td>
<td>161</td>
<td>14.29</td>
</tr>
<tr>
<td>iCITA</td>
<td>0</td>
<td>12.5</td>
<td>23.00000</td>
<td>93</td>
<td>23.67</td>
</tr>
</tbody>
</table>

and other assistive technology users. Landmarks do not require JavaScript code to be used, and thus are out of the scope of this investigation. This one website example was removed from the Control group.

6.3.2. Number of HTML Elements with Role Attributes. In the top 320 most popular websites from Alexa, the group presented an average of 3.134 role attributes per page, even though it presented an average of 126.7 JavaScript mouse event listeners per page. And approximately 80% of the websites did not present any role attribute. Figure 8 illustrates a histogram of the number of role attributes per page.

Among the separate groups of the Subject group, the Forum and E-commerce groups presented the highest number of role attributes per page: 24.25 average number of roles for the Forum group and 10.68 average for the E-commerce.

The Control group, which consists of WAI-ARIA example implementations, presented an average of 23 role attributes per page, with 75% of websites with more than seven role attributes. The number of roles in the Subject and Control groups are significantly different (Welch Two Sample t-test, with \( df = 37.926 \) and \( p-value = 8.165 \times 10^{-6} \) in a 0.95 confidence interval).

All minimum, median, mean, maximum, and standard deviation (S.D.) values for each group are presented in Table II, and Figure 9 presents a boxplot representation for illustrating the distribution of the number of roles for each group.

In the Subject group, 13 websites presented more than 20 HTML elements with role attributes. The role attributes ranged from landmark, button, and menu to tab and dialogue interface components. Four of these websites used the jQuery toolkit, but only one of the 13 websites used a component user interface library (YUI - Yahoo! User Interface) to build these widgets. The other nine websites implemented their own JavaScript solution to build WAI-ARIA widgets.
6.3.3. Focus Navigation Metric Results. The Subject group presented an average score of 61.15% for the focus navigation metric, and 75% of the websites of this group scored higher than 33.33%.

In the Subject group, the Search group presented the highest score on average: 89.51%. The other groups’ average score ranged from 37.85% (Forum group) to 56.82% (Company/Product group).

The Control group, on the other hand, presented an average score of 96.07%, and only one website presented a score lower than 75%.

Table III presents the minimum, median, mean, maximum, and standard deviation (S.D.) values of the Fona metric for all groups of the investigation. Figure 10 illustrates a comparison of the score’s distribution between all groups. In Figure 10, it can
be observed that the iCITA group (the Control group) presents the highest distribution of values for the metric (the first quartile, median, third quartile, and maximum representations are placed over the 100% value on the metric’s value axis).

The Sample group metric scores were significantly different from the Control group (in a Welch Two Sample t-test, with df = 80.313 and p-value = $2.2 \times 10^{-16}$). These results support the hypothesis H1 of the investigation.

A difference analysis between all subgroups that compose the Sample group and the Control group was conducted to further investigate differences between the subgroups’ metric scores. The one-way Analysis of Variance (ANOVA) revealed significant differences between the subgroups (using the seven subgroups and the Sample group classification as factors, with df = 7, F = 20.1, and p-value = $2 \times 10^{-16}$, in a 0.95 family-wise confidence level). Post hoc comparisons using the Tukey range test revealed significant differences between the Portal (p-value = $7.52 \times 10^{-10}$), Company/Product (p-value = $7.14 \times 10^{-10}$), E-commerce (p-value < $1 \times 10^{-20}$), Video (p-value = $5.23 \times 10^{-7}$), Login (p-value = $5.84 \times 10^{-6}$), and Forum (p-value = $2.78 \times 10^{-3}$) subgroups and the Control group. The only Sample subgroup that did not present a significant difference when compared to the Control group was the Search subgroup. Moreover, the Search subgroup also presented significant differences from all other Sample subgroups: Portal (p-value = $3.38 \times 10^{-11}$), Company/Product (p-value = $7.34 \times 10^{-10}$), E-commerce (p-value < $1 \times 10^{-20}$), Video (p-value = $3.68 \times 10^{-6}$), Login (p-value = $4.65 \times 10^{-5}$), and Forum (p-value = $1.03 \times 10^{-2}$). Figure 11 illustrates the confidence intervals for each comparison between the Sample subgroups and Control group according to Fona’s scores.

Twenty-one websites from the Subject group scored 100% in Fona’s metric. These websites’ categories were Portal (five websites), Company/product (eight websites), Search (five websites), E-commerce (one website), Login (one website), and Video Streaming (one website). Eleven websites from the Subject group scored zero in Fona’s metric. These 11 websites were classified as Portal (one website), Company/product (seven websites), Login (two websites), and Video Streaming (one website).

7. DISCUSSION

Ajax and RIA have been around since the Ajax movement in 2005 [Garrett 2005]. The Ajax movement led developers to use open technologies that compose the web, like JavaScript, to implement rich interaction functionalities for web applications. These investigation results show that top most popular websites according to Alexa present an average number of 126.7 JavaScript mouse event listeners attached to HTML elements in a webpage. This result highlights the increased popularity JavaScript has gained over the years and the importance of making this technology accessible to disabled users on the web. Only 29 websites did not present any JavaScript mouse event listeners; however, this does not imply that they did not implement any JavaScript functionality at all.

The high number of JavaScript mouse event functionalities observed in the Subject group was not followed by a high number of role attributes’ usage in websites. Even though the WAI-ARIA specification reached the status of W3C Recommendation in March 2014, a great proportion of websites from the Subject group in our investigation did not present any role attribute set to HTML elements (278 of the 349 websites did not present any role attribute). From this report, it can be observed that WAI-ARIA is still not considered in many web projects, despite the high number of JavaScript functionalities that are deployed every day on the web.

It is worth noting that, even though the Control group is composed of websites with a lower number of JavaScript functionalities (the Control group presents an average of 16.68 JavaScript mouse event listeners, while the Subject group presents
Fig. 11. Confidence intervals of the ANOVA post hoc analysis using Tukey range test to compare all Sample subgroups’ and Control groups’ Fona scores.

an average of 126.7), the Control group still presents a significantly greater amount of HTML elements with role attributes (according to a t-test analysis of differences). If we consider the WAI-ARIA specification, as the number of JavaScript functionalities is increased, the number of HTML elements with role attributes is also expected to be increased. These findings present the differences between both groups, in which the Control group presents a higher number of role attributes, indicating it implements
the WAI-ARIA specification, while the Subject group still presents a low number of WAI-ARIA role attributes’ usage in websites.

Thirteen websites from the Subject group presented more than 20 HTML elements with role attributes. However, in this group of 13, only one website used a WAI-ARIA conformant user interface library. This fact can result from the Subject group mainly being represented by frontpage websites, and many of them work toward minimizing the number of JavaScript source code files that need to be downloaded in order to make the webpage loading time faster. The authors believe this design strategy was the cause for the low number of user interface component libraries, because they increase the number of JavaScript files that need to be loaded by the webpage, even though they significantly reduce the effort of building WAI-ARIA conformant widgets.

All websites that compose the Subject group consist of frontpage web applications from Alexa’s list. Frontpages target a wide variety of users. Moreover, frontpages present simple interaction scenarios, since they generally orient users to other content and more complex webpages (meaning frontpages require less effort in the development of design solutions for widgets). Thus, by focusing our study on frontpage analysis, we believe that the websites in the Subject group performed better in Fona then they would have performed if an in-depth analysis of the webpages was made.

Separately evaluating the number of JavaScript functionalities and HTML elements with role attributes provides absolute values of how WAI-ARIA has been considered in web projects. Absolute values cannot be used in a direct comparison between a group of websites, since the size and complexity between different websites might negatively impact the comparison outcomes. For instance, analyzing solely the number of role attributes in a website with 300 role attributes and a website with 30 role attributes might lead to the interpretation that the first website presents improved WAI-ARIA conformance. However, the 300 role attributes in the first website might be inserted in an HTML code that presents 3,000 HTML elements that should present role attributes (leading to a 10% conformance rate), while the second website presents 30 role attributes in 30 HTML elements that should present the attribute (leading to a 100% conformance rate).

The focus navigation metric aims at calculating a proportional value that indicates the number of mouse event listeners that might also implement an accessible keyboard behavior. Since the metric value indicates a proportional value, it can be used to estimate the amount of JavaScript interaction scenarios that miss a WAI-ARIA keyboard navigation functionality. The use of a proportional value also enables the comparison between different websites, which is used in the comparison made between the Sample subgroups and the Control groups, even though these websites vary in size (number of HTML elements with JavaScript functionalities).

The metric makes use of the TabIndex attribute of HTML elements to verify if the HTML element that has a mouse event listener can be the target of keyboard events and then implement an accessible keyboard usage scenario. Evaluating the number of HTML elements with a TabIndex attribute equal to or greater than zero does not provide accessibility indicators, since links and form elements that do not have JavaScript listeners attached to them also present a TabIndex attribute equal to zero. Thus, we used the TabIndex attribute checking combined with the number of JavaScript mouse event listeners and the number of HTML elements with role attributes.

In regards to the research questions, the metric proved to be capable of differentiating a group of popular websites that did not necessarily implement any WAI-ARIA widgets (the metric showed significant differences in value between the Subject group and the Control group, supporting the hypothesis \( H_1 \)). Even though the metric provides an upper limit for an accessibility evaluation report, it was enough to identify
Fona: Quantitative Metric to Measure Focus Navigation on Rich Internet Applications

Fona's goal is to monitor how focus navigation functionalities have been implemented on the web, and thus the metric stands as a quality metric (Quality-focused advantage). Fona's validation approach consisted of an experiment conducted to analyze whether the metric's calculation could be used to discriminate WAI-ARIA and non-WAI-ARIA web applications. Running the metric on a group of WAI-ARIA web applications highlights the importance of focus navigation for WAI-ARIA (100% mean value). On the other hand, comparing WAI-ARIA and non-WAI-ARIA web applications scores shows the differences between each subgroup and how far they are in terms of implementing Focus Navigation requirements. This validation criterion is characterized as empirical validity, a validation criterion subgrouped in the external validity criterion group, according to Meneely et al. [2013]. The use of the external validity criterion was determined using the Criteria Application Process defined by Meneely et al. [2013].

The metric identifies JavaScript mouse event listener functions that do not implement a keyboard WAI-ARIA conformant alternative behavior. The results obtained from this investigation show that many web projects do not implement focus navigation for their widgets and that many widgets are not even capable of being the target of focus events; thus, these widgets cannot deliver the keyboard navigation scenarios that represent mandatory WAI-ARIA requirements.

The metric also highlights differences in focus navigation mechanism implementations between the Subject group and the Control group. The Control group scored an average value of 100%, while the Subject group scored an average value of 61.23%, both significantly different. This score difference shows that the web (represented with the sample of the most popular websites according to Alexa) does not implement the same navigation strategies that WAI-ARIA web applications implement. Thus, the web still presents severe accessibility barriers in its functionality delivered through widgets and other interactive components that characterize the Web 2.0. Nevertheless, the mean value of 61.23% in the Subject group leads to the following conclusion: at least 48.77% of every JavaScript mouse event functionality in the Subject group, which represents the web, do not implement keyboard interaction scenarios. The metric analyzes two accessibility features in each HTML element with an attached JavaScript mouse event functionality: focus navigation (by means of considering the TabIndex attribute) and WAI-ARIA conformance (by means of considering the role attribute). The metric's results for the Subject group indicate that 48.77% of the JavaScript mouse event functionalities of websites do not implement any of them. It is also worth noting that the metric is not capable of verifying if a keyboard interaction is actually implemented in each interactive element; however, it presents an upper limit of how accessible these elements might be on websites. Nevertheless, from the WAI-ARIA specification perspective, all JavaScript mouse event listeners are expected to present a keyboard interaction mechanism alternative, and thus web applications should strive for a 100% metric score in Fona.

The separate analysis between the different subgroups that compose the Subject group shows that the Search group performed significantly better than all other groups (considering its distribution of values). However, it is worth noting that search webpages present a significantly lower number of JavaScript interface components. Additionally, a great proportion of the Search websites consisted of the same search application, but rendered with a different language. For instance, there were multiple instances of the Google search web application: http://www.google.com, http://www.google.co.id, http://www.google.com.tw, among many others. These samples were not unified in the Subject group, because some websites scored differently when rendered in different languages. For instance, Amazon's e-commerce website presented
the score of 32.8358% in the domain amazon.com, while it presented the score of 100% in the domain amazon.com.jp. This implies that these websites rendered in different languages might present different implementation details. Thus, they were separately analyzed.

The next section presents the limitations of the approach.

7.1. Limitations of the Approach

It is important to acknowledge that the metric is not capable of evaluating CSS, only widgets, since the metric requires that JavaScript mouse event listeners are attached to HTML elements to calculate Fona’s value. Moreover, the metric is not capable of identifying keyboard navigation scenarios of spinner widgets. In the Control group, one website generated Fona’s value below 20%. This WAI-ARIA example implemented a spinner widget. This spinner widget had two buttons to increase/decrease the value of the spinner, and both buttons could not receive focus (had the TabIndex attribute set to -1) and did not present any role attribute set to them. However, these buttons had an immediate child element (a DOM element contained in them), which was semantically marked with a presentation role attribute, indicating that this element should not be part of a screen-reader user interaction scenario. The use of the presentation role in widgets is expected and correctly considered in the metric, but the inclusion of the presentation role in a child element is not considered by the metric. This behavior denotes a navigation strategy that is not mapped in the metric. Thus, investigating an element’s child nodes for additional role attributes should be regarded in future works, even though the Subject group did not present a high number of role attributes’ implementation during the investigation.

The metric also counts mouse event listeners, which are responsible only for decoration effects in the elements. For instance, a web application can have an attached hover event listener, which only changes the background color in an element, which is not focusable. This behavior might not impact negatively the focus navigation of a webpage if it regards a purely decorative effect in an element that does not provide any functionality. This implementation impacts negatively in Fona. However, the authors argue that this kind of behavior is frequently implemented in interactive elements to focus the attention of users in a specific functionality that is available to them. Therefore, since there is functionality attached to those kinds of decorative event listeners, they should be accessible via focus navigation. Nevertheless, most websites use CSS pseudo-classes to implement this kind of behavior and thus are not evaluated in Fona.

In the following code fragment, considering that all three DIV elements present mouse event listeners attached to them, Fona’s calculation will result in 1, since all three elements present a nonnull role attribute. If these three elements did not present a role attribute, Fona’s calculation would result in zero.

```html
<div role="listbox" tabindex="0" aria-activedescendant="f1">
  <div role="option" id="f1" tomato></div>
  <div role="option" id="f2" carrot></div>
</div>
```

This usage scenario highlights that Fona cannot completely assess whether focus navigation has been completely implemented in the web application. This relaxation in the evaluation approach was used considering that ARIA states that attributes such as aria-activedescendant should be set to DOM elements that present a role attribute.

In order to fully validate the use of the aria-activedescendant attribute, it would be required that the tool analyze the ID attribute set to aria-activedescendant and check if it corresponds to an existing DOM element. Then, user interaction events should be dispatched in order to analyze if the focus is changed to all role option DIV elements.
Other usage scenarios such as the verification of programmatic inclusion of DOM elements in the tab order or direct focus in DOM elements (calling the focus method of DOM elements) are other examples that would rely on this relaxation. Thus, further efforts are required to completely validate these usage scenarios.

Our approach only implements a subset of the components of the RIA automatic evaluation framework proposed in Doush et al. [2013]. In order to validate the use of the `aria-activedescendant` attribute, the web robot component (a native system input event generator) is required. This extension in the evaluation approach should be implemented in future versions of the tool.

8. CONCLUSIONS

This article reported the development of Fona (Focus Navigation Assessment), a metric that identifies how focus navigation has been implemented on the web. Focus navigation is a mandatory WAI-ARIA requirement that work toward enhancing keyboard navigation scenarios in RIA.

We make use of JavaScript mouse event listeners, HTML elements with role attributes, and TabIndex attribute observations in order to automatically calculate the metric. The metric enables estimating the amount of JavaScript scenarios that do not present a WAI-ARIA keyboard navigation alternative functionality.

The metric is validated comparing its execution on a group of websites that implement WAI-ARIA requirements and a group that does not. The results show that many websites on the web still do not implement WAI-ARIA requirements, despite the high number of JavaScript-based features that are deployed every day, even after the WAI-ARIA specification has reached the status of W3C recommendation.

Fona analyzes every HTML element that has an attached mouse event JavaScript listener in the DOM structure of webpages. The metric checks if these elements present one of the following: a TabIndex attribute equal to 0 (meaning it can receive focus and might implement an accessible keyboard interaction scenario) or a role attribute (meaning it implements a WAI-ARIA widget and might implement an accessible keyboard interaction scenario). Running the metric in a corpus of 320 high-traffic webpages according to Alexa revealed that at least 48.77% of every HTML element that has an attached mouse event JavaScript listener cannot be interacted with using keyboard-only interactions.

Fona's evaluation approach provides a narrow analysis of one single accessibility requirement. But it enables monitoring this accessibility requirement in a large number of webpages. This type of analysis might give insights about how focus navigation and ARIA have been considered by web development teams and if the standards set by WAI (Web Accessibility Initiative) have been applied by web developers.

Moreover, the analysis of the separate subgroups of websites highlights that many websites still lack the implementation of focus navigation through their JavaScript interactive content, even though Fona’s evaluation approach is not capable of completely validating every design characteristic that leads to an accessible keyboard navigation scenario. The experiment, fostered by Fona, identifies the groups of websites that require more development effort and those that are closer to the ideal accessibility implementation. This monitoring activity shows one main concern that needs to be addressed in the software engineering process: web developers still lack awareness of ARIA technological solutions such as focus navigation.

9. FUTURE WORKS

Throughout this article, Fona was used to investigate how focus navigation has been implemented through the web. The investigation revealed several groups of websites that do not implement focus navigation for all their JavaScript functionalities. In
order to allow this comparison between the websites’ groups, Fona was elaborated to determine a proportional value, which represents the number of JavaScript mouse interactive elements that do not implement focus navigation in relation to the total number of JavaScript mouse interactive elements in a webpage (composed of the number of elements that are focus navigable and the ones that are not).

As a future work, Fona could be adapted to assist web developers through the development process by identifying JavaScript functionality that does not implement focus navigation. Instead of providing a proportional metric value, the tool could be adapted to provide absolute values that represent the actual number of HTML elements with JavaScript functionality that are not keyboard navigable. The evolution of an absolute value through the lifetime of a single product could be more easily interpreted by stakeholders. Henceforth, more experiments should be conducted in order to verify the feasibility of applying Fona in a development process.

The development of Fona’s tool also works toward the elaboration of automatic RIA accessibility evaluation tools. Fona’s tool implements a subset of the components proposed in Doush et al. [2013], and future works might enhance the tool to completely validate ARIA requirements in web applications.

Moreover, considering the developer’s perspective, adapting the metric’s functionality to a browser add-on platform could improve the visualization of the HTML elements that are not accessible through focus navigation. Even though the tool reported in this article (Section 5) uses a headless browser engine (an in-browser evaluation environment [Fernandes et al. 2011, 2012]), reimplementing the same functionality as a browser add-on enables showing which HTML elements cannot receive focus in the window (highlights these elements, for instance). This functionality might assist the developers through their coding activities by providing instant feedback on how their implementations are conformant with the Focus Navigation requirement.

Future works also include investigating the evolution of the focus navigation metric for specific websites (using configuration management systems of open-source projects), including the metric’s approach for identifying keyboard-inaccessible elements in other accessibility evaluation tools and extending the metric concept to verify other WAI-ARIA requirements.

REFERENCES


ACM Transactions on the Web, Vol. 9, No. 4, Article 20, Publication date: September 2015.


ACM Transactions on the Web, Vol. 9, No. 4, Article 20, Publication date: September 2015.