2015-03

Keyboard navigation mechanisms in widgets: an investigation on ARIA's implementations

Journal of Web Engineering, Paramus, v. 14, n. 1-2, p. 41-62, Mar. 2015
http://www.producao.usp.br/handle/BDPI/50991

Journal of Web Engineering, Vol. 14, No. 1&2 (2015) 041–062

KEYBOARD NAVIGATION MECHANISMS IN WIDGETS: AN INVESTIGATION ON ARIA'S IMPLEMENTATIONS

WILLIAN MASSAMI WATANABE
UTFPR - Universidade Tecnológica Federal do Paraná , Campus Cornélio Procópio
Cornélio Procópio, PR, 84016-210, Brazil
wwatanabe@utfpr.edu.br

RAFAEL JOSÉ GERALDO and RENATA PONTIN DE MATTOS FORTES
Intermídia, ICMC-USP - Institute of Mathematical and Computer Sciences of the University of São Paulo
São Carlos, SP P.O.Box 668, 13560-970, Brazil
{*rafaeljg,renata*}*@icmc.usp.br*

This study presents an investigation on how keyboard accessibility has been delivered in RIA - *Rich Internet Applications*. We conducted an evaluation on 32 websites which contained Tab Widgets and 74 websites which contained Menu Widgets, from the 150 websites of Alexa's top most accessed websites list. The evaluation process consisted of checking if the Widgets implemented ARIA - *Accessible Rich Internet Applications* requirements, like the use of role/state semantic attributes and presentation of keyboard interaction strategies. The results showed that, even though the ARIA specification achieved the status of W3C Candidate Recommendation in 2011 and W3C Recommendation in 2014, few websites implemented Tab and Menu Widgets according to ARIA in the Web. The study also identified alternative keyboard navigation mechanisms that are accessible to Assistive Technologies users, despite the disadvantages they might represent. Moreover, the study proposes a framework to classify these alternative keyboard navigation mechanisms and map the technological requirements which need to be addressed in order to make the Tab and Menu Widgets implement the ARIA specification.

Keywords: Web accessibility, ARIA, Widgets, Keyboard Navigation, Navigation Mechanisms

1 Introduction

Since 2005, with the "Ajax Movement" [1], web applications interaction experience has been gradually enhanced. Ajax is defined as an architectural style based on the use of open Internet standards [2]. This architectural style represents the technology foundation which enables desktop-like experience in the web platform and the web applications which implement these technologies are referred to as RIA - *Rich Internet Applications* [3].

RIAs frequently use scripting languages, like JavaScript, in order to implement logic constructs and adapt the document structure of the HTML, which is presented to users [4]. These constructs and changes in the document structure are dispatched while users are interacting with webpages, and compose sophisticated interaction mechanisms, Widgets, which impact the usability and personalization of web systems [5].

However, the increased complexity in interactivity is also pushing the limits of Assistive Technologies [6]. Assistive technologies need to be notified of changes and updates made to the document structure. Otherwise, users that interact with the webpage through screen readers,

for instance, will not perceive these changes and the interface component will be inaccessible to them. Traditional Assistive Technologies present web content following a linearized structure. RIAs break this assumption, since content can be dynamically included within any part of the document, as the user interact with the web application [7].

In order to standardize and assist web developers while implementing accessible RIA, the WAI - *Web Accessibility Initiative* elaborated the ARIA - *Accessible Rich Internet Application* specification [8]. The ARIA specification has the goal of establishing an accessibility framework, which includes semantic markup to HTML elements in order to inform Assistive Technologies about updates and changes made to the document structure [9]. However, developers are not always aware of the design solutions and recommendations provided by accessibility specifications [10, 6].

Freire et al. conducted a survey about accessibility awareness on web developers in Brazil [11]. The survey made use of a questionnaire to identify which accessibility concepts were known by web developers from the academy, industry and government. The results showed that only 19.9% of the participants stated that accessibility requirements were considered in their projects. In another study, Freire et al. presented a metric based approach to monitor the accessibility levels in Brazilian Municipalities websites [12]. The study proposed the adaptation of the Hera web accessibility evaluation tool[a] to calculate accessibility metrics. The approach was, then, run against a group of Brazilian Municipalities websites. The results showed, in average, webpages presented a non-desirable medium to high number of accessibility barriers. It is worth noticing that both studies presented findings based on WCAG - *Web Content Accessibility Guidelines*, thus do not provide insights about how the ARIA specification has been considered in web projects.

This article presents an investigation on keyboard navigation mechanisms that are implemented in Tab and Menu Widgets delivered in the most successful websites according to Alexa[b] (extending the findings reported in [13]). The investigation focuses on Tab and Menu Widgets implementations across multiple websites and has the objective of identifying whether the ARIA specification has been implemented in the Web or not. The ARIA specification describes a set of user scenarios that must be implemented for multiple Widgets [14], thus we compare the behavior delivered by Tab and Menu Widgets through different websites. The results show that the ARIA specification has been poorly implemented in the Web, according to the sample considered in the investigation, and identify distinct accessibility design solutions. This study also proposes a framework to group distinct accessibility solutions to Tab and Menu Widgets and to map the technological requirements which are required for these Widgets enhance their implementation of the ARIA specification.

The ARIA specification achieved the status of W3C Candidate Recommendation by 2011 and was promoted to W3C Recommendation in 2014. Thus, analysing how web developers have implemented ARIA technological requirements provides insights about how successful is the specification among web developers and might guide future efforts to enhance the engineering process of web applications. JavaScript has been long used in web development environments and this study has the goal of providing evidence which support that the specification is enough to guarantee accessibility in RIA or other approaches should be considered.

[a] http://www.sidar.org/hera/index.php.en
[b] http://www.alexa.com

This study is structured as follows: Section 2 describes the main concepts of the ARIA specification; Section 3 describes how to implement an accessible Tab and Menu Widget according to the ARIA specification; Section 4 presents the investigation, which consists of the methodology and gathered results; Section 5 addresses a discussion on the results of the investigation; and Section 6 presents final remarks and future works.

2 ARIA

The interactivity implemented in RIA can pose severe accessibility barriers to users interacting with webpages through Assistive Technologies [15, 16]. RIAs use scripting languages and other advanced technologies to deploy interface components with sophisticated interactivity. However, these interface components sometimes demand visual perception and mouse interactions to be operated by users [9]. Thus, ad-hoc development of these interface components (without considering how these technologies might impact disabled users interaction scenarios) pose severe obstacles to accessibility [3].

RIAs are developed with HTML and general purpose programming languages, like JavaScript. Thus, it is difficult for Assistive Technologies to predict the behavior of interactive components with basis in static source code analysis. There are several possibilities of Widgets implementations inside RIAs, like Drop-Down Menus, Tree Views and Tab Widgets; and each of these Widgets can dynamically change any part of the document structure. Also, a user agent or Assistive Technology that presented all the changes made to the document structure would overload the user with irrelevant information [15].

In order to make these Widgets accessible, the WAI elaborated the ARIA specification, which describes an accessibility framework for RIA [8]. Since RIA interface components implement user interaction scenarios that resemble desktop interface components, the same accessible behavior implemented in desktop applications can be applied in web applications [16].

Desktop application interface components present a pre-determined behavior which is known by Assistive Technologies. Therefore, screen readers are capable of informing the user about changes made to the interface, since the interface components are updated following a predictable behavior.

It is worth noticing that this approach could not be directly applied to web application Widgets. The Widgets are implemented using generic HTML elements, therefore the Widgets do not inform Assistive Technologies about their behavior [9, 16]. In this context, the ARIA specification describes the inclusion of semantic markup to Widgets' instances. These semantic markup identify which pre-defined usage scenarios are implemented by Widgets (using HTML attributes such as **state**, **properties** and **roles**), so that Assistive Technologies can inform users about possible changes made to the document structure.

The ARIA specification also defines a set of user interaction scenarios that should be implemented by Widgets, in order to enhance the navigation of users through multiple ARIA widgets in the webpage. These interaction scenarios present that focus navigation (using the **tabindex** HTML attribute and TAB key strokes [8]) should be provided for navigation between Widgets, while other keyboard shortcuts should be used to provide navigation within a Widget. Unlike traditional HTML form controls, RIA Widgets have no inherent keyboard support, thus web developers must manually implement keyboard navigation mechanisms

that match the specification requirements [14].

The following paragraph shows an HTML code fragment which presents a Tab Widget structure in a web application. Since this HTML code does not present ARIA roles / state / property attributes, screen reader users are not informed of the type of widget which is implemented in this markup structure. Even if this Tab Widget structure can receive focus and responds to keyboard navigation commands, as the tab elements receive focus, screen reader users will not the type of widget they are interacting with. Thus, screen reader users will not know which interaction mechanisms are implemented in this interface component, neither how other elements (for instance, how the tab panels visibility change the next tab element becomes active) will respond to keyboard events dispatched in the focused element.

```
<div>
    <div tabindex="0">Tab 1</div>
    <div>Tab 2</div>
</div>
<div>Tab panel 1</div>
<div>Tab Panel 2</div>
```

On the other hand, the following paragraph shows an HTML code fragment which present a Tab Widget structure in a web application with ARIA roles/state/property attributes correctly set. As screen reader users interact with the component elements of this Tab Widget, they will be informed of the type of Widget they are interacting with. For instance, as screen reader users set focus to the first tab element (identified by the id "tab1"), the users will be informed they are interacting with a tab element identified as "Tab 1" inside a Tab Widget with two tab elements. Moreover, as the users are informed they are interacting with a Tab Widget, the users will know the keyboard navigation mechanisms which can be used to search for specific information in the widget (such as using down/right arrow keys to activate the next tab).

```
<div role="tablist">
    <div role="tab" id="tab1" tabindex="0">Tab 1</div>
    <div role="tab" id="tab2">Tab 2</div>
</div>
<div role="tabpanel" aria-labelledby="tab1">Tab panel 1</div>
<div role="tabpanel" aria-labelledby="tab2">Tab Panel 2</div>
```

The ARIA specification has been implemented by some JavaScript toolkits and frameworks, like jQueryUI[c], Google Web Toolkit[d], YUI[e], DOJO Toolkit[f]and KISSY[g]. These toolkits implement the ARIA specification in a set of Widgets, and these accessible Widgets can be reused across any website that uses these JavaScript libraries.

[c]http://jqueryui.com/
[d]http://www.gwtproject.org/
[e]http://yuilibrary.com/
[f]http://dojotoolkit.org/
[g]http://docs.kissyui.com/

3 Accessible Widgets

ARIA provides a documentation guide to authors to help them understand and implement accessible Widgets. Even though this W3C documentation has the status of working draft (on progress specification), it represents the only association between the ARIA specification and accessible interaction models which need to be implemented by Widgets. The ARIA specification provides an accessible framework based on semantic markup for Widget's elements, however it does not provide information about the use cases that need to be implemented by Widgets. Thus, [14] is the only ARIA related resource available to developers who are concerned with the engineering of accessible Widgets. Next subsections present, respectively, the summarized concepts regarding the implementation of Tab and Menu Widgets.

3.1 Tab Widgets

Tab Widgets stand as JavaScript implementations of the interaction Design Pattern named Tabs[h]. The pattern suggests the use of Tabs structure as the main navigation structure of websites. Tabs present an horizontal/vertical row of clickable elements, the tabs, and each of these elements represents a section label. Each tab has an associated panel with content related to the tab label. There is only one visible/active panel which is visually connected to its associated tab. As each tab is activated, its correspondent panel is activated/presented and all other panels are hidden.

In the context of RIA, Tab Widgets are built using Web Standards such as JavaScript, HTML and CSS. These Widgets are presented inside the document structure, changing the visibility of HTML elements (the tabs and panels), as the user interacts with the tabs. This behavior is a result of dynamic changes and updates to the document structure, then if it is not correctly identified by Assistive Technologies it will not be perceived by users accessing the webpage through a screen reader, for instance.

ARIA defines the following set of use cases (user keyboard navigation scenarios) that need to be implemented by Tab Widgets [14]:

- The focus navigation is enabled only for the active tab of the Widget;
- The (UP/DOWN/LEFT/RIGHT) arrow keys change the selected tab;
- The CTRL+HOME keys change the focus of the active tab to the first tab of the Widget;
- The CTRL+END keys change the focus of the active tab to the last tab of the Widget);
- The CTRL+UP arrow keys set the focus back to the active tab as the focus is on the Widget's panel;
- The CTRL+PAGEUP (for the next tab) and CTRL+PAGEDOWN (for the previous tab) change the selected tab as the focus is on the Widget's panel.

The ARIA specification also specifies that the tab elements (the clickable elements) should be marked semantically with the role attribute set to `tab`. All tab elements must be contained by an HTML element which is marked with role attribute set to `tablist`. Finally, the panel

[h] http://www.welie.com/patterns/showPattern.php?patternID=tabbing

elements should be marked semantically with the role attribute set to `tabpanel` and present an ARIA attribute which identifies the panel with a textual description (`aria-labelledby` / `aria-describedby` / `aria-label`). Many other attributes can be used to further describe the Widget behavior to assistive technologies.

If the Widget's elements present the correct roles, states and properties attributes set, the user will be informed that they are interacting with a Tab Widget and the total number of tabs, as the active tab receives focus. Then, based on this context information, the user can predict changes realized in the document and identify the keyboard navigation strategies which can be applied. As the user is navigating inside a panel, he/she is also notified that the focus is set to a Tab Widget panel associated with an specific tab element.

3.2 Menu Widgets

The Menu Widget is a container of options that usually consists of a list of choices to the user, as for example, a list of links to important sections of the site. The Menu Widget's elements often present submenu items, and so, the Widget must provide keyboard support that allows to navigate all menu items.

The Menu Widget which was investigated in this study refers to JavaScript/CSS/HTML implementations of the Fly-out Menu interaction design pattern[i]. This pattern establishes a user interface component which enables navigating through multiple navigation links even when the amount of screen estate for navigation is limited. Initially, only section labels, which establish a hierarchy between menu links, are presented to users. As users move the mouse cursor over the section labels, the links which are contained in the section labels become visible to users and can be activated. This interaction pattern supports nested menus enabling expert users to quickly access menu items.

ARIA Authoring Practices define that the menu container may have a role of `menu` or `menubar`. The menu items may have one of the following roles: `menuitem`, `menuitemcheckbox` or `menuitemradio`, according to its implementation.

The menu focus is managed by using `tabindex` or `aria-activedescendant` attributes. When the activation of a `menuitem` produces a popup menu, this `menuitem` may have the attribute `aria-haspopup` set to `true` to improve the user navigation who uses an assistive technology. Furthermore, the first item in the `menu` or `menubar` should be in the tab order as `tabindex=0`.

The following set of user keyboard navigation scenarios are defined by ARIA and need to be implemented by Menu Widgets [14]:

- When a letter key is typed the focus moves to the next menu item whose title begins with that printable letter.
- With focus on a menu bar:
  - The RIGHT/LEFT arrow keys change focus to the adjacent menu bar item;
  - The DOWN arrow key expand the submenu and set the focus on the first submenu item if the menu item with focus has a submenu;

[i] http://www.welie.com/patterns/showPattern.php?patternID=fly-out-menu

  - The ENTER/SPACE keys invoke the menu item action. If the menu item with focus has a submenu, the menu action must expand the submenu and set the focus on the first submenu item.

- With focus on a menu item:
  - The UP/DOWN arrow keys cycle focus through the submenu items;
  - The ENTER/SPACE keys invoke the menu item action.
  - The LEFT arrow/ESCAPE keys close the submenu and return the focus to the parent menu item.

If the Menu Widget is appropriately implemented according the ARIA roles, states, and properties, screen reader users will be notified that they are interacting with a Menu Widget, thus they can assume the Widget will present a similar behavior to a Menu Interface Component of a desktop application and all keyboard navigation scenarios which will be implemented by the Widget.

4 Tab and Menu Widgets in Most Successful Websites

Many websites have included JavaScript and Ajax functionality to deliver highly interactive Widgets, like drop-down menus, tree views, tab panels, among others [17]. These Widgets need to implement keyboard navigation mechanisms, in order to be accessible to users interacting with the Web through Assistive Technologies, such as screen readers.

The ARIA specification acquired the status of W3C Recommendation in 2014 [8], even though Ajax and RIA have been popularized since 2005 with the Ajax Movement [1]. However there is no guarantee that keyboard navigation mechanisms will be included in web applications [6]. Web developers are not always aware of Web accessibility recommendations, such as WCAG - *Web Content Accessibility Guidelines* [18] and Section 508[j][11]. In this context, this article has the objective of investigating how the ARIA specification has been implemented in the Web.

We conducted an analysis of the interaction mechanisms delivered by RIA Widgets contained in the most successful websites according to Alexa[k]. It is worth noticing that the report represents preliminary results of our research, thus the analysis focused RIA Tab and Menu Widgets, implemented using JavaScript, CSS and HTML to deliver client-side interactivity. During our analysis, we identified distinct keyboard navigation mechanisms designed in the Widgets. This study reports the current status of ARIA specification conformity and implementation rate, considering a websites sample based on Alexa's popularity metric.

Next sections describe the sample, methodology and results of this study.

4.1 Sample description

The sample of this study was extracted from Alexa most successful websites list. Alexa is a company which provides free, global traffic metrics for websites. Alexa's popularity rank is defined by the traffic amount of all websites relative to each other. In regards to that, Alexa's rank list was used as a representative sample of websites in the Web.

[j] https://www.section508.gov/
[k] http://www.alexa.com

Alexa's 150 websites presented the following themes:

Search: presents a webpage search engine solution. These web applications contain a single search text input element which enables users to search for other websites.
Thirty-five webpages were classified in this theme (23.33%).

Content portals: presents constantly updated content, mainly related to: news, sports, weather, movies, among other information. These webpages' goal is to present the most significant information to users in a single page and enable the user to further read an specific topic of interest in other webpages by means of links.
Thirty-three webpages were classified in this theme (22.00%).

Company/Product portfolios: presents information associated to a company's portfolio or an specific product. These webpages present photos, videos, instructional paragraphs and contact information, with the goal of advertising the company/product.
Thirty-two webpages were classified in this theme (21.33%).

Video Streaming: presents multiple image links which represent videos screenshots. These image links invite users into watching videos in the website.
Fifteen webpages were classified in this theme (10.00%).

Login: presents a single login form and require user authentication to make use of the website's main functionality. These websites representatives are social platforms which allow authenticated users to publish and read content from other users.
Fifteen webpages were classified in this theme (10.00%).

ecommerce: presents an online store which presents multiple products photos and their respective price. These webpage's goal is to invite users into buying its products.
Thirteen webpages were classified in this theme (6.67%).

404/403: all websites analyses were conducted from Brazil, from August 2013 to March 2014, and some websites were not found (returned 404 HTTP status messages in the browser) or were forbidden (returning 403 HTTP status messages).
Four webpages were classified in this theme (2.67%).

forum: presents a list of question/answers and comments to users.
Three webpages were classified in this theme (2.00%).

The website's number according to each theme is illustrated in Figure 1.

All 403/404 websites (4 websites) were removed from our sample, since they represent static HTML webpages which are not expected to be presented to users.

Then, all remaining websites (146 websites) first rendered page (root or front page) were manually analyzed to identify Tab and Drop-down Menu Widgets. As a result of this analysis, two groups of websites were formed: a group of websites which present Tab Widget implementations (the **Tab Widgets Group**) and a group of websites which present Drop-down Menu Widgets (the **Menu Widgets Group**). It is worth noticing that some websites present both Widgets, meaning these websites are part of both groups; and some websites did not present any Widget, meaning these websites are not part of any group. Both analysis methodology and the number of websites classified in both groups are described, next:

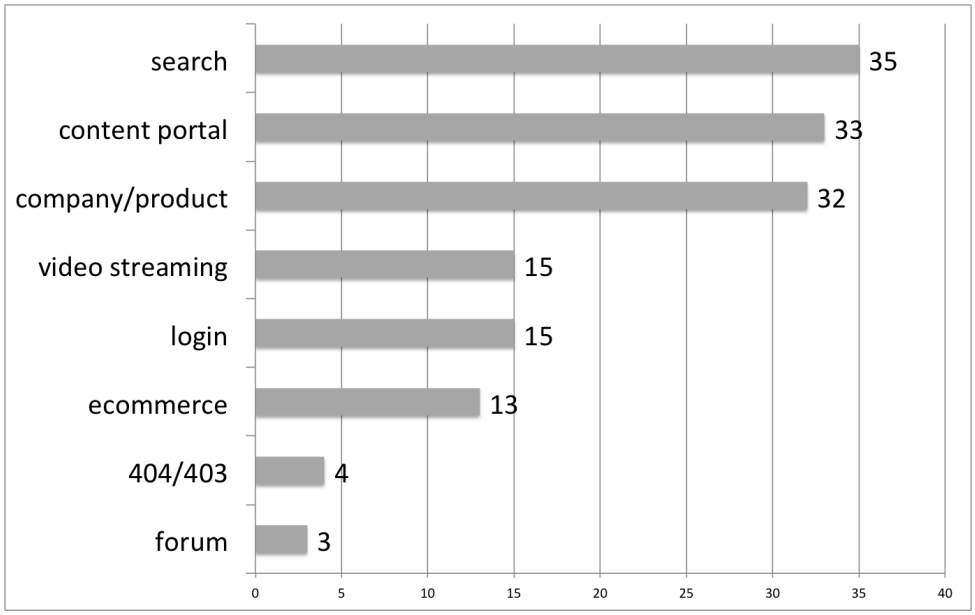

Fig. 1. Number of websites classified according to an specific theme.

Websites with Tab Widgets: the Tab Widgets identification was performed by means of scanning the webpage, searching for mouse interactive elements with associated panels of information which resembled the Tabs interaction design pattern (with a single visible panel which is visually connected to its associated interactive element, the tab). It is worth noticing that some websites did present Tabs, however these Tabs interface component were not implemented using client-side interactivity, thus they cannot be classified as a RIA Widget and were not classified in this group. We also removed websites which automatically changed the active panel (given a specific amount of time), since this behavior characterizes Carousel and Slideshow Widgets.
Thirty-two websites from the sample presented a Tab Widget implementations according to this criteria (21.00%).

Websites with Drop-down Menu Widgets: the Drop-Down Menu Widgets identification was performed by scanning the webpage, searching for interactive elements which react to mouse events (click or hover), presenting additional links as menu options. These menu options should be presented using client-side interactivity (using JavaScript or CSS functionality), so that they represent RIA Widgets.
In the sample, many similar websites (in regards to the HTML, CSS and JavaScript resources) were served on distinct domain-names, like: www.google.com, www.google.ca, www.google.co.uk, among others. Even though these websites are served in distinct domain-names, they represent the same web application, presenting similar Widgets and interface structure. In order to prevent these RIA equivalent websites from significantly impacting the quantitative results of this investigation, all domain-name websites which represented the same web application, with similar Widgets and interface struc-

ture, were unified into a single representative website. Three web applications presented this behavior: Google (with 23 websites unified), Amazon (with 3 websites unified) and Ebay (with 2 websites unified).
Seventy-four websites from the sample presented Drop-Down Menu Widget implementations according to these criteria (49.33%).

After the filtering process and the identification of Tab and Drop-Down Menu Widgets, the sample of our study was composed of two groups: the **Tab Widget group** composed by 32 websites[l] and the **Menu Widget group** composed by 74 websites[m]. It is worth noticing that both groups contain multiple distinct Widget implementations.

4.2 Methodology

The methodology of our investigation consisted of extracting all HTML elements that were not associated to the Widget and conducting the following verifications in each website:

Widget Roles: check if the Widget elements present `tab`, `tablist` and `tabpanel` role attributes for Tab Widget instances and `menubar`, `menu` and `menuitem` role attributes for Drop-Down Menu Widget instances.

Tabindex: check if the interactive elements of each Widget (the Tabs for the Tab Widgets and the Menu Items for the Drop-Down Menu Widgets) can receive focus, checking if the interactive elements present the HTML atribute of `tabindex` set to an integer value equals or greater than zero.

Keyboard Navigation Mechanisms: compare the navigation model between the Widgets and validate all keyboard navigation scenarios that are presented in the ARIA specification.
As keyboard navigation scenarios for Tab Panel Widgets, the following 13 Use Case verifications were considered:

1. Checks if the focus is changed to the next tab element as the DOWN arrow key is pressed in all tab elements.
2. Checks if the focus is changed to the next tab element as the RIGHT arrow key is pressed in all tab elements.
3. Checks if the focus is changed to the previous tab element as the UP arrow key is pressed in all tab elements.
4. Checks if the focus is changed to the previous tab element as the LEFT arrow key is pressed in all tab elements.
5. Checks if the focus is changed to the first tab element as the CONTROL and HOME keys are pressed in all tab elements.
6. Checks if the focus is changed to the last tab element as the CONTROL and END keys are pressed in all tab elements.

[l] https://gist.github.com/watinha/7551097
[m] https://gist.github.com/watinha/11305993

7. Checks if the respective tab panel element is visible as the DOWN arrow key is pressed in all tab elements.
8. Checks if the respective tab panel element is visible as the RIGHT arrow key is pressed in all tab elements.
9. Checks if the respective tab panel element is visible as the UP arrow key is pressed in all tab elements.
10. Checks if the respective tab panel element is visible as the LEFT arrow key is pressed in all tab elements.
11. Checks if the focus is set back to the tab element when the user is interacting with the tab panel element and when the user presses the CONTROL and UP arrow keys.
12. Checks if the focus is set to the previous tab element when the user is interacting with the tab panel element and when the user presses the CONTROL and PAGEUP keys.
13. Checks if the focus is set to the next tab element when the user is interacting with the tab panel element and when the user presses the CONTROL and PAGEDOWN keys.

As keyboard navigation scenarios for Menu Widgets, the following 7 Use Case verifications were considered:

1. Checks if the focus is changed to the next adjacent menu element as the RIGHT arrow key is pressed in all menu elements.
2. Checks if the focus is changed to the previous adjacent menu element as the LEFT arrow key is pressed in all menu elements.
3. Checks if the focus is changed to the next sub-menu item as the DOWN arrow key is pressed in all sub-menu item elements in the menu.
4. Checks if the focus is changed to the previous sub-menu item as the UP arrow key is pressed in all sub-menu item elements in the menu.
5. Checks if the menu/sub-menu is closed and the focus is set to the parent menu item as the ESC key is pressed in all menu item elements.
6. Checks if the menu item elements are activated as ENTER or SPACE keys are pressed in all menu item elements.
7. Checks if the first menu item element is focused as the letters that compose its title are typed in the keyboard.

All Tab Widgets were analyzed from September 1st to 7th of 2013, while all Menu Widgets were analyzed from April 22nd to 25th of 2014, with two evaluators. The analysis were conducted using a Mozilla Firefox Browser 23.0[n] to render the Widgets and the Firebug extension[o] to check the document structure of the webpage.

[n]http://www.mozilla.org/en-US/firefox/fx/
[o]https://addons.mozilla.org/pt-br/firefox/addon/firebug/

Two researchers conducted all verifications steps in the Tab and Menu Widgets. Both researchers were graduated students at Universidade de São Paulo in the Ph.D. program of the ICMC - Instituto de Ciências Matemáticas e de Computação. Both Ph.D. researchers worked in the area of web accessibility and present more than 4 years of experience in the field. Moreover, both researchers also present more than 5 years of experience in web development applied fields.

The verifications were manually conducted since, by the time of the study, there was no available tool capable of automatically or semi-automatically identifying Tab/Menu Widgets and further analysing their dynamic DOM structure, through user keyboard interaction scenarios.

4.3 Results

The results of the Widgets analysis were separated according to the verification of **Widget Roles**, **Tabindex** and **Keyboard Navigation Mechanisms**. The results are presented in the next sections.

4.3.1 Widget Roles

In the **Tab Widget Group**, only 1 of the 32 websites which contained Tab Widget implementations presented the ARIA role attribute correctly set (Taobao[p]). This Tab Widget was correctly marked with the `tab`, `tablist` and `tabpanel` role attributes. The Widget's elements also presented the `aria-hidden` (which hide information from Assistive Technologies) and `aria-labelledby` (which presents text labels to Widgets components) properties. The website used the KISSY JavaScript library[q] to implement the Tab Widget.

In the **Menu Widget Group**, three of the 74 websites which contained Menu Widget implementations presented the ARIA role attribute `menu` and `menuitem` correctly set (Twitter[r] , Youtube[s] and Ebay[t]). These three websites also used the `aria-haspopup` attribute to identify the element which showed/enabled the menu, `aria-labelledby` / `aria-describedby` / `aria-label` attributes to improve the semantic description of the menu item elements and `aria-controls` attribute to explicitly set the association between elements which show/activate the menu and the menu elements themselves.

In the **Menu Widget Group**, eight websites presented alternative role attributes usage to semantically describe Menu Widget. These websites used the following alternative role attributes strategies:

- using the `navigator` role attribute to describe the menu container element. This strategy is also described in the ARIA specification, however it should be used complementary to the use of `menu`/`menubar`/`menuitem` roles Widgets. In this context, all websites that used this strategy were also missing the appropriate implementation of Menu Widgets. *Five websites presented this behavior.*

- using the `button` role attribute to describe the element which activated/showed the

[p] http://www.taobao.com
[q] http://docs.kissyui.com/
[r] http://www.twitter.com
[s] http://www.youtube.com
[t] http://www.ebay.com

menu and the menu item elements. This strategy assists users by identifying the elements which, when activated, dispatch changes in the DOM structure of the webpage. However, the use of this strategy does not identify which type of change is dispatched and neither the interaction model the user should expect of the Menu Widget.
Two websites presented this behavior.

- using a single `menu` role attribute with no use of `menubar` or `menuitem` role attributes. This strategy correctly identifies the Menu Widget for users, however users will not be notified of the number of elements which are contained in the menu.
One website presented this behavior.

Next section presents the `tabindex` attribute analysis of the widgets.

4.3.2 Tabindex

Focus Navigation is a mandatory accessibility requirement for RIAs [8]. In order to implement Focus Navigation in RIA, web developers need to manage the `tabindex` attribute of JavaScript interactive interface elements that are part of a Widget. HTML elements which present the `tabindex` attribute set with a value equals or greater than zero can receive focus events, thus these elements can implement keyboard navigation mechanisms.

Regarding to the Focus Navigation accessibility requirement in the **Tab Widget Group**, the sample presented 23 websites (71.9%) with Tab Widgets which could receive focus and 9 websites (28.1%) with Tab Widget which could not. The percentage of Tab Widgets with Focus Navigation is illustrated in Figure 2.

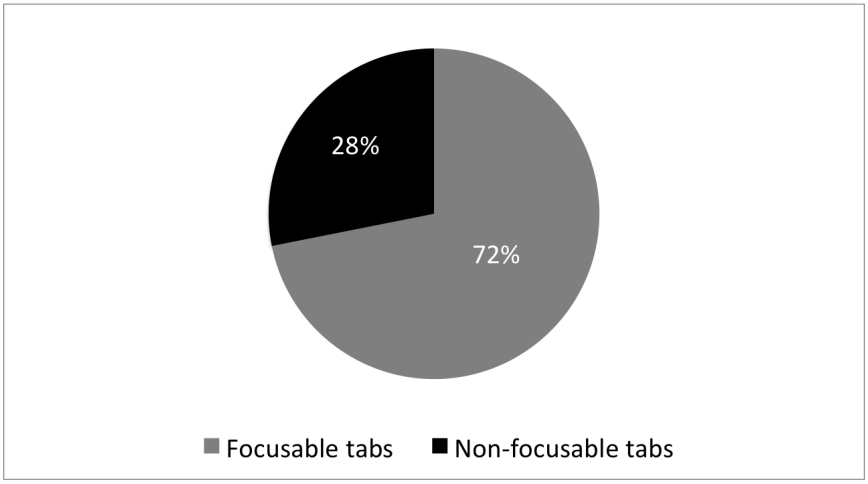

Fig. 2. Percentage of Tab Widgets which can obtain Focus, in the Alexa's ranking websites.

On the other hand, regarding the Focus Navigation accessibility requirement in the **Menu Widget Group**, the sample presented 25 websites (34%) with Menu Widgets which could receive focus and 49 websites (66%) with Menu Widgets which could not. The percentage of websites of the **Menu Widget Group** with Focus Navigation is illustrated in Figure 3.

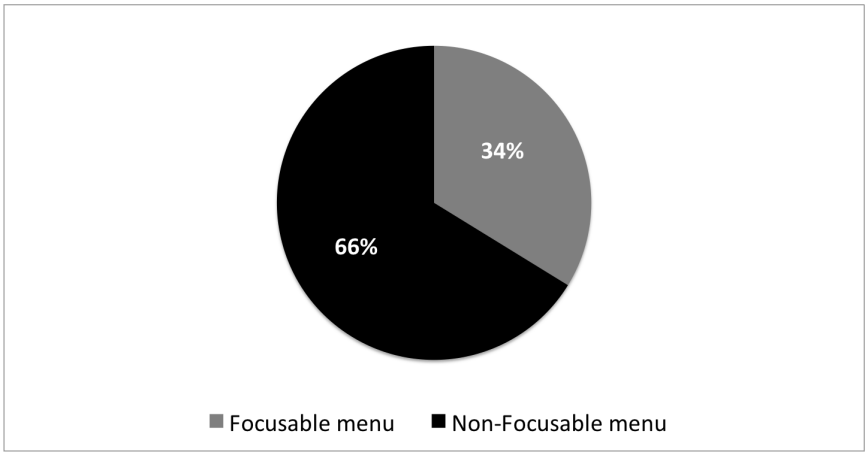

Fig. 3. Percentage of Menu Widgets which can obtain Focus, in the Alexa's ranking websites.

4.3.3 Keyboard Navigation Mechanisms

In order to evaluate the keyboard navigation mechanisms that were built in the Tab and Menu Widgets from Alexa's most successful websites, each Widget was manually tested to identify keyboard strategies that could be applied in it. The test consisted of checking each keyboard interaction scenario which should be supported by Tab Widgets (totaling 13 use cases) and Menu Widgets (totaling 7 use cases) [14].

In the **Tab Widgets Group**, only 1 Tab Widget from the Sample presented keyboard navigation mechanisms which implemented the accessible use cases described in the ARIA specification. However, even this Widget instance did not implement all the keyboard use cases described in the ARIA specification, delivering 9 of the 13 use cases expected in Tab Widgets. This Widget instance also implemented the correct use of ARIA roles attributes (using the ARIA role attributes: `tablist`, `tab` and `tabpanel`).

In the **Menu Widgets Group**, 5 Menu Widgets from the Sample presented keyboard navigation mechanisms which implement similar strategies to the ones defined in the ARIA specification (using the arrow keys to navigate through the menu item elements). However, only 1 of the five Widget instances delivered all 7 use cases expected in Tab Widgets and only 1 distinct Widget also implemented the correct use of ARIA roles attributes (using the ARIA role attributes: `menu` and `menuitem`).

Even though most Widgets did not strictly implement the same ARIA specification keyboard interaction model, many of them did implement a keyboard alternative interaction model. With regards to that, the Widgets keyboard navigation mechanisms from the Sample presented the following keyboard interaction models:

No keyboard navigation: characterized by Widgets which were not keyboard operable, regardless of the fact that their content could be reached by screen reader users by linear navigating through all elements of the webpage.

- *9 Widgets of the **Tab Widgets Group** implemented this interaction model.*

- *26 Widgets of the **Menu Widgets Group** implemented this interaction model.*

Link fallback: characterized by Widgets with interactive elements (representing the tabs in Tab Widgets and menu items in the Menu Widgets) that were HTML links. These links targeted webpages with the similar content offered in the displayed panels or menus. It is worth noticing that the use of links to provide keyboard operability does not imply the same user experience specified in ARIA. The links are set as fallback usage scenarios, so that content is still perceivable by users interacting through Assistive Technologies. As a user activates one tab or menu element implementing this approach, the user will have to wait a complete page reload and completely re-scan the webpage to find the information he/she was looking for. The approach can also be identified as a hybrid of the *graceful degradation* [19] and *progressive enhancement* [20] JavaScript accessibility strategies.

- *7 Widgets of the **Tab Widgets Group** implemented this interaction model.*
- *23 Widgets of the **Menu Widgets Group** implemented this interaction model.*

Focus Navigation: characterized by Widgets with interactive elements which represented buttons in the interface. These interactive elements were inserted in the tab order of the webpage (Focus navigation) and could be activated with an ENTER key stroke. When the user activated these elements in the Widgets, the document structure would be updated to display the newly selected tab or menu, providing the RIA experience (updates made to the interface with no page refreshes). Some of the Menu Widgets, presented the menu items just by setting focus to the parent menu item. This approach defined a keyboard operable navigation strategy, however it does not provide contextual information about the Widget behavior to screen readers. As the user interacts with the Widget, he/she might not be aware of the changes that will be triggered after activating each interactive element. As the user navigates through the Widget, he/she will not be informed of the association of the Widget components and the elements which showed/activated the components. Moreover, these Widgets lack the implementation of arrow keys interaction scenarios. The arrow keys should be used to provide navigation scenarios within a Widget, while Focus Navigation should be used to provide navigation scenarios through different Widgets, in order to enhance the usability for screen reader users [8].

- *15 Widgets of the **Tab Widgets Group** implemented this interaction model.*
- *20 Widgets of the **Menu Widgets Group** implemented this interaction model.*

Partial ARIA implementation: characterized by Widgets which partially implement the ARIA keyboard navigation scenarios (using the arrow keys to enhance the interaction between Widget's components).

- *1 Widget of the **Tab Widgets Group** implemented this interaction model.*
- *5 Widgets of the **Menu Widgets Group** implemented this interaction model.*

ARIA Full: characterized by Widgets which implement the complete set of ARIA keyboard navigation scenarios as well as all role attributes necessary for the Widget.

- *No Widget of the* ***Tab Widgets Group*** *implemented this interaction model.*
- *No Widget of the* ***Menu Widgets Group*** *implemented this interaction model.*

The groups quantitative data are illustrated in Figure 4 for the **Tab Widgets Group** and Figure 5.

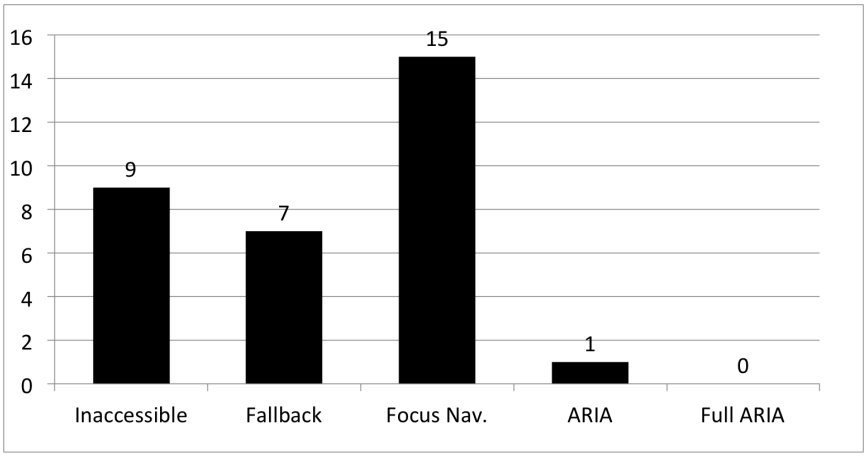

Fig. 4. Number of websites of the **Tab Widgets Group** which implemented each keyboard navigation strategy.

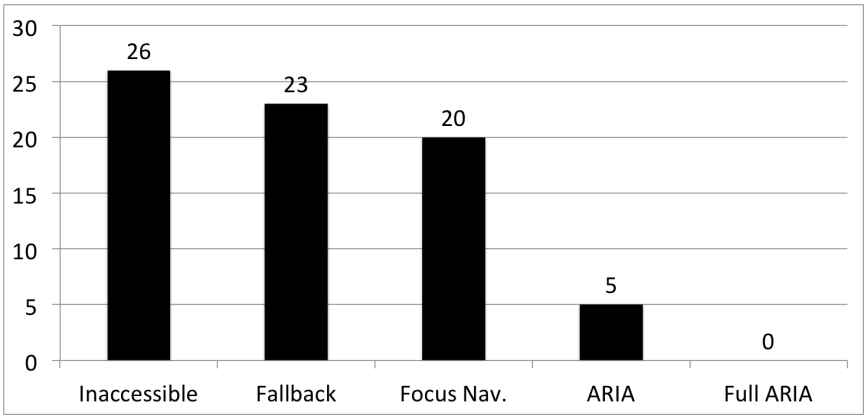

Fig. 5. Number of websites of the **Menu Widgets Group** which implemented each keyboard navigation strategy.

The keyboard interaction models observed in both groups present specific interface design characteristics which differently impact the way disabled users will interact with the Widget. The inclusion of these interface design characteristics one by one in a Widget, gradually

enhance the accessibility of the Widget for a screen reader user, until the Widget delivers the full set of navigation mechanisms described in the ARIA specification. Thus, if Widgets, previously classified in the **Inaccessible** group are improved to implement the set of interface design characteristics which characterize the **Fallback** model (using links to represent each interactive element of the Tab or Menu Widget), these Widgets would evolve to the **Fallback** model group of Widgets. This same behavior can be used to transition a Widget from the **Fallback** model group to the **Focus navigable** model group (adding JavaScript mechanisms to handle keyboard events dispatched to the previously focus navigable **Fallback** element) and so on, until the Widget implements the same interaction model presented in the ARIA specification.

Figure 6 illustrates this progressive enhancement accessibility framework, in which: **Inaccessible** Widgets can transition to **Fallback** model Widgets by using links to represent the interactive elements of the Widgets; **Fallback** model Widgets can transition to **Focus navigable** model Widgets by implementing keyboard interactivity through focus navigation requirements (accessing and activating tabs or menu items via ENTER keypresses, for instance); including ARIA roles and implementing arrow keys to improve the navigation mechanisms, to transition a **Focus navigable** model Widget to the **Partial ARIA** model Widget; and using all ARIA attributes and keyboard interaction scenarios to transition a **Partial ARIA** model Widget to a **Full ARIA** model Widget.

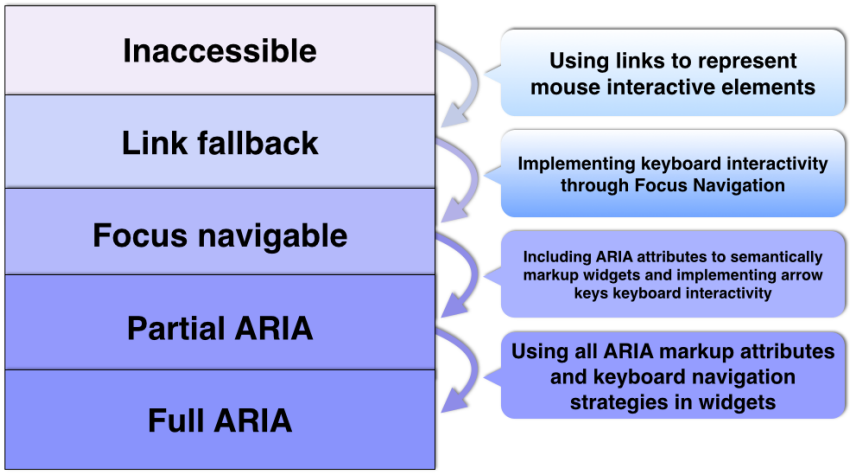

Fig. 6. Keyboard Navigation models observed in the **Tab Widgets Group** and **Menu Widgets Group**.

In regards to this progressive enhancement framework and the percentage of websites in each group (Inaccessible, Fallback, Focus Navigation, Partial ARIA and Full ARIA), the following results can be obtained from this experiment. In the **Tab Widgets Group**:

- 9 websites (28.1% of the group) need to use link elements to represent the interactive elements of Widgets.
- 16 websites (50% of the group) need to implement keyboard interactivity through Focus

Navigation (using the `tabindex` attribute) in all interactive elements of the Widget.

- 31 websites (96.9% of the group) need to implement ARIA attributes for tabs (`tablist`, `tab`, `tabpanel` and `aria-labelledby`/`aria-label`/`aria-describedby`) and arrow keys navigation mechanisms.
- 32 websites (100% of the group) need to implement the complete set of interaction mechanisms and ARIA attributes and structure described in the ARIA specification for Tab Widgets.

And in the **Menu Widgets Group**, the following results can be observed:

- 26 websites (35.1% of the group) need to use link elements to represent the interactive elements of Widgets.
- 49 websites (66.2% of the group) need to implement keyboard interactivity through Focus Navigation (using the `tabindex` attribute) in all interactive elements of the Widget.
- 69 websites (93.2% of the group) need to implement ARIA attributes for tabs (`tablist`, `tab`, `tabpanel` and `aria-labelledby`/`aria-label`/`aria-describedby`) and arrow keys navigation mechanisms.
- 74 websites (100% of the group) need to implement the complete set of interaction mechanisms and ARIA attributes and structure described in the ARIA specification for Tab Widgets.

5 Discussion

The increase of JavaScript and Ajax technologies adoption rates in web applications [17] was not followed by an increased number of websites implementing ARIA. Even though the ARIA specification achieved the status of W3C Candidate Recommendation by 2013 and was promoted to a W3C Recommendation in 2014, only one Tab Widget instance and four Menu Widgets used the semantic markup described in the specification and presented a similar keyboard navigation interaction model. The ARIA Tab Widget was built based on KISSY JavaScript library and only one of the ARIA Menu Widgets was built using the jQueryUI JavaScript framework.

The low number of ARIA Widgets identified in this study suggests that further efforts should be put into researches that work towards identifying the reasons of the low number of websites adequacy to the ARIA specification.

The investigation which is reported in this paper significantly differs from other reports made in the literature. Freire et al. conducted surveys on Web Accessibility knowledge and implementation, however both reports focus WCAG recommendations [11, 12], thus ARIA specific design solutions are not assessed in these studies. Other studies also present the same strategy of assessing how WCAG design solutions have evolved in websites through time[21, 22].

Our report presents accessibility assessment reports on individual interface components (Tab and Menu Widgets), while the other approaches [11, 12, 21, 22] present general accessibility analysis on the complete structure of websites. Fernandes et al. present an accessibility

evaluation strategy in Rich Internet Applications, however the ARIA specific recommendations are not evaluated [23, 24]

In order to systematically evaluate ARIA implementations in websites, our study focused the evaluation of technological characteristics (assessing ARIA roles and `tabindex` attributes implementation) and the interaction model of Widgets. Including the interaction model analysis as an evaluation criteria was necessary in this study, since many accessibility features presented in the ARIA specification represent interaction scenarios which need to be manually implemented by web developers. This additional step of evaluating the keyboard interaction scenarios built into web applications also significantly differ from other approaches.

JavaScript libraries and toolkits work towards the reuse of design solutions that implement Widgets in the Web platform. The reuse of the design solutions reduce the effort required to implement ARIA in Tab and Menu Widgets. However, in the sample, only two websites used JavaScript libraries to reduce the effort in building ARIA Widgets solutions. This low number of JavaScript libraries usage in the sample can be a result that most websites from Alexa's list consist of front page websites which have the goal of reducing the number of JavaScript files to download, in order to decrease the loading time of the web application. JavaScript libraries demand that the web application downloads dependency files to built Widgets. Thus, the authors believe that the strategy of reducing the number of external dependencies to improve the loading time of web applications represents the main reason for the low number of JavaScript libraries usage in the sample.

The Sample group in this study is composed of the most successful websites according to Alexa, therefore web developers responsible for these websites present great influence in the Web community. This fact also highlights the urgency in identifying the causes of low number of ARIA Widgets reported in this study.

The study identified alternative keyboard navigation strategies for Tab and Menu Widgets in the Sample group. Regardless of the fact that only one Tab Widget presented partial implementation of the ARIA specification, 71.9% of the websites from the **Tab Widgets Group** were operable by means of an alternative keyboard interaction model (**fallback** and **focus navigation**). In regards to the **Menu Widgets Group**, 58.1% of the websites were operable by means of the alternative keyboard interaction models of **fallback** and **focus navigation**. Both alternative navigation strategies present disadvantages when compared to the ARIA proposed interaction model. However, further studies are required to verify how severely these disadvantages would impact the user interaction with Tab and Menu Widgets.

The investigation also presented a technological framework to describe the keyboard navigation models and how to progressively improve ARIA navigation mechanisms in Tab and Menu Widgets, accordingly to the Widgets' instances available on the Web. The framework and the investigation highlights the technological requirements that need to be implemented by each group of Widgets.

In the **Menu Widgets Group**, five Widgets used distinct ARIA attributes to semantically describe Widgets' interactive components. Three of them used the `navigation` role to identify the menu, instead of using the `menu`, `menubar` and `menuitem` role attributes. The `navigation` role is defined in the ARIA specification and should be used to group multiple navigation mechanisms of a website. In regards to client-side interactive Drop-Down menus, the `navigation` role should be used as a container to multiple menus. The other two Widgets

used the `button` role to describe the menus. However, a button widget does not comprise all interaction scenarios nor semantic information that needs to be informed to screen reader users interacting with a Menu Widget. All these five distinct ARIA implementations, lack strict implementation of the ARIA specification, however they represent efforts that are being put into building accessible design solutions into RIA.

The results described in this article identified a low adoption rate of ARIA in the Web, having found no website which implements a Tab or Menu Widget which implement all interaction mechanisms presented in the ARIA specification. Both groups of the investigation (**Tab Widget Group** and **Menu Widget Group**) presented similar alternative keyboard interaction models. However, further research efforts need to be dedicated on other types of Widgets and identifying how the alternative navigation mechanisms impact in the interaction of Assistive Technologies users.

6 Final Remarks

This article presented an investigation report on how accessibility has been delivered in RIAs. The investigation conducted a manual evaluation on Tab Widgets instances of 32 websites and on Menu Widgets instances of 74 websites from the most successful websites according to Alexa.

The findings show that ARIA requirements are still not considered in many of Tab and Menu Widgets instances that represented the Web. Even though only one Tab Widget and four Menu Widgets implemented the keyboard navigation scenarios specified in ARIA, many websites provided alternative keyboard navigation strategies to users (71.9% of the Tab Widgets and 58.1% of the Menu Widgets). The results identified two alternative keyboard navigation strategies in Tab and Menu Widgets: fallback and focus navigation. Both approaches present disadvantages when compared to the interaction strategy described in the ARIA specification, however further studies are required to verify how usability is impacted when interacting with all these approaches.

This investigation proposed a progressive enhancement framework to map the keyboard interaction models observed in Tab and Menu Widgets with the technological characteristics which implement them. The framework also highlights the technological requirements which need to be implemented accordingly to each alternative keyboard navigation strategy implemented in the Tab and Menu Widgets.

In regards to the progressive enhancement framework, future works include using the interaction models identified in this paper to increase the awareness of developers about the ARIA specification. The framework could be used as an objective methodology of gradually improving the accessibility of Tab and Menu Widgets through the development process.

The progressive enhancement framework could also be used as reference to automatic ARIA evaluation tools. The framework classifies distinct Widgets instances considering to the technological requirements the Widgets implement and the ones they do not. The identification of these low-level details could help the elaboration of automatic strategies which can assess ARIA requirements in web applications.

Other future works include: extending this investigation to other types of Widgets, comparing ARIA requirements implementations with the JavaScript toolkits/solutions which are used and tracking the reasons for the low adoption rate of ARIA experience implementation

on the Web.

Acknowledgments

Our thanks to UTFPR, ICMC-USP, CNPq and CAPES for supporting this work.

References

1. J.J. Garrett (2005), *Ajax: A New Approach to Web Applications.* URL http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/
2. P. Fraternali, S. Comai, A. Bozzon, G.T. Carughi (2010), *Engineering rich internet applications with a model-driven approach*, ACM Trans. Web **4**(2), 1
3. C.A. Velasco, D. Denev, D. Stegemann, Y. Mohamad (2008), *A web compliance engineering framework to support the development of accessible rich internet applications*, in *W4A '08: Proceedings of the 2008 international cross-disciplinary conference on Web accessibility (W4A)* (ACM, New York, NY, USA, 2008), pp. 45–49
4. K.U. Schmidt, J. Dörflinger, T. Rahmani, M. Sahbi, L. Stojanovic, S.M. Thomas (2008), *An user interface adaptation architecture for rich internet applications*, in *ESWC'08: Proceedings of the 5th European semantic web conference on The semantic web* (Springer-Verlag, Berlin, Heidelberg, 2008), pp. 736–750
5. J. Kluge, F. Kargl, M. Weber (2007), *The Effects of the AJAX Technology on Web Application Usability*, in *3rd International Conference on Web Information Systems and Technologies (WebIST 2007)* (Barcelona, Spain, 2007)
6. W.M. Watanabe, R.P.M. Fortes, A.L. Dias (2012), *Using acceptance tests to validate accessibility requirements in RIA*, in *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility* (ACM, New York, NY, USA, 2012), W4A '12, pp. 15:1–15:10
7. P. Thiessen, C. Chen (2007), *Ajax live regions: ReefChat using the fire vox screen reader as a case example*, in *W4A '07: Proceedings of the 2007 international cross-disciplinary conference on Web accessibility* (ACM, New York, NY, USA, 2007), pp. 136–137
8. W3C (2011), *Accessible Rich Internet Applications - Version 1.0.* W3C Candidate Recommendation. URL http://www.w3.org/TR/wai-aria/
9. B. Gibson (2007), *Enabling an accessible web 2.0*, in *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A'07)* (ACM, New York, NY, USA, 2007), pp. 1–6
10. W.M. Watanabe, A.C. Jr., V.R. de Uzeda, R.P.M. Fortes, T.A.S. Pardo, S.M. Alusio (2009), *Facilita: Reading Assistance for Low-literacy Readers*, in *ACM SIGDOC 2009* (ACM, Bloomington, IN, USA, 2009), pp. 29–36
11. A.P. Freire, C.M. Russo, R.P. de Mattos Fortes (2008), *The perception of accessibility in Web development by academy, industry and government: a survey of the Brazilian scenario*, New Review of Hypermedia and Multimedia **14**(2), 149
12. A.P. Freire, T.J. Bittar, R.P.M. Fortes (2008), *An approach based on metrics for monitoring web accessibility in Brazilian municipalities web sites*, in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing* (ACM, New York, NY, USA, 2008), pp. 2421–2425
13. W.M. Watanabe, R.J. Geraldo, R.P. de Mattos Fortes (2014), *Keyboard Navigation Mechanisms in Tab Widgets: An Investigation on ARIA's Conformance*, in *Proceedings of the 29th Annual ACM Symposium on Applied Computing* (ACM, New York, NY, USA, 2014), SAC '14, pp. 721–726
14. W3C (2013), *WAI-ARIA 1.0 Authoring Practices - An author's guide to understanding and implementing Accessible Rich Internet Applications.* W3C Working Draft. URL http://www.w3.org/TR/wai-aria-practices/
15. E.V. Munson, M.d.G. Pimentel (2008), *Specialized Documents*, in *Web Accessibility*, *Human-Computer Interaction Series*, vol. 4 (Springer London, 2008), *Human-Computer Interaction Series*, vol. 4, pp. 274–285
16. P. Thiessen, S. Hockema (2010), *WAI-ARIA live regions: eBuddy IM as a case example*, in *Pro-*

ceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A) (ACM, New York, NY, USA, 2010), W4A '10, pp. 33:1–33:9

17. B. Gibson, R. Schwerdtfeger (2005), *DHTML accessibility: solving the JavaScript accessibility problem*, in *Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility* (ACM, New York, NY, USA, 2005), Assets '05, pp. 202–203
18. W3C (2008), *Web Content Accessibility Guidelines (WCAG) 2.0*. W3C Recommendation. URL http://www.w3.org/TR/WCAG20/
19. T.W.S. Project, *The JavaScript Manifesto*. URL http://www.webstandards.org/action/dstf/manifesto/
20. C.A. Hall (2009), *Web presentation layer bootstrapping for accessibility and performance*, in *Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibililty (W4A)* (ACM, New York, NY, USA, 2009), W4A '09, pp. 67–74
21. V.L. Hanson, J.T. Richards (2013), *Progress on Website Accessibility?*, ACM Trans. Web **7**(1), 2:1
22. V.F. de Santana, R.A. de Paula (2013), *Web Accessibility Snapshot: An Effort to Reveal Coding Guidelines Conformance*, in *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility* (ACM, New York, NY, USA, 2013), W4A '13, pp. 2:1–2:4
23. N. Fernandes, D. Costa, S. Neves, C. Duarte, L. Carriço (2012), *Evaluating the accessibility of rich internet applications*, in *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility* (ACM, New York, NY, USA, 2012), W4A '12, pp. 13:1–13:4
24. N. Fernandes, A.S. Batista, D. Costa, C. Duarte, L. Carriço (2013), *Three web accessibility evaluation perspectives for RIA*, in *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility* (ACM, New York, NY, USA, 2013), W4A '13, pp. 12:1–12:9