Universal Access in the Information Society
https://doi.org/10.1007/s10209-023-01043-5

LONG PAPER

Dynamic detection of accessibility smells

Fernando Durgam[1] · **Julián Grigera**[1,2,3] · **Alejandra Garrido**[1,2]

Accepted: 28 August 2023

Abstract
Automatic detection of accessibility problems is mainly performed by checking for compliance with guidelines on the HTML structure of web pages. While this method can find many problems, it has limitations in detecting difficulties that occur during user interaction. The purpose of this work is to find problematic sequences of interaction events, which we call Accessibility Events. These events occur dynamically as the user interacts with the page and can result in automatic detection of accessibility problems, called Accessibility Smells. We focus on visually impaired users interacting with the web through screen readers. Using previously and recently defined Accessibility Smells, we design Accessibility Events and heuristics to detect them. We describe an empirical study with visually impaired users accessing different pages with known Accessibility Smells. Using a logging tool, we capture Accessibility Events and report on their relationship (or lack thereof) with those smells. For the study, we recruited 8 volunteers, who performed user tests in different websites. During the study, we automatically captured the events on the interfaces and found that out of the 100 events detected during the sessions, 64 resulted in accessibility odors and 19 did not. The remaining 17 were inconclusive, but helped to reformulate the current odor heuristics to analyze potential new ones. The results indicate that it is possible to characterize special patterns of Accessibility Events that may be used to detect potential accessibility issues. While further studies are necessary, our findings provide a base ground for the dynamic detection of accessibility problems in web applications.

Keywords Web accessibility · Rich internet applications · Accessibility smells · User interaction events

1 Introduction

Web Accessibility continues to be a largely neglected practice. A recent study conducted by WebAIM for the top million websites showed over 96% of homepages with WCAG 2 [23] failures, with an average of more than 50 accessibility errors per page [28]. Moreover, the steep increase in web usage caused by the COVID-19 pandemic is causing deep inequalities for people with disabilities [21]. This makes an urgent call for action. Many governments and organizations have created policies and regulations that joined well-established accessibility guidelines. Ensuring conformance, however, can be a very demanding task, especially in already deployed software. To help in this task, many efforts have been made to automate the detection of accessibility issues.

Most works in the area of automatic accessibility evaluation focus on static analysis, i.e., parsing HTML code to find compliance with fixed sets of rules, such as the Web Content Accessibility Guidelines (WCAG) [23]. While this is a useful and effective approach, there are accessibility issues that can only be discovered by analyzing real interaction [14]. For example, consider a simple menu with different options at the same level, where the first three options are rarely used. In this case, a sighted user could ignore these and go directly to the relevant options, but the visually impaired relying on a screen reader must necessarily go through them each time. This could happen even in a perfectly WCAG-compliant site, but the issue could still be detected by observing that the visually impaired users generally skip the irrelevant options.

✉ Julián Grigera
julian.grigera@lifia.info.unlp.edu.ar

Fernando Durgam
fernando.durgam@lifia.info.unlp.edu.ar

Alejandra Garrido
alejandra.garrido@lifia.info.unlp.edu.ar

[1] LIFIA, Fac. de Informática, Universidad Nacional de La Plata, La Plata, Argentina

[2] CONICET, Buenos Aires, Argentina

[3] CICPBA, La Plata, Argentina

Published online: 22 September 2023

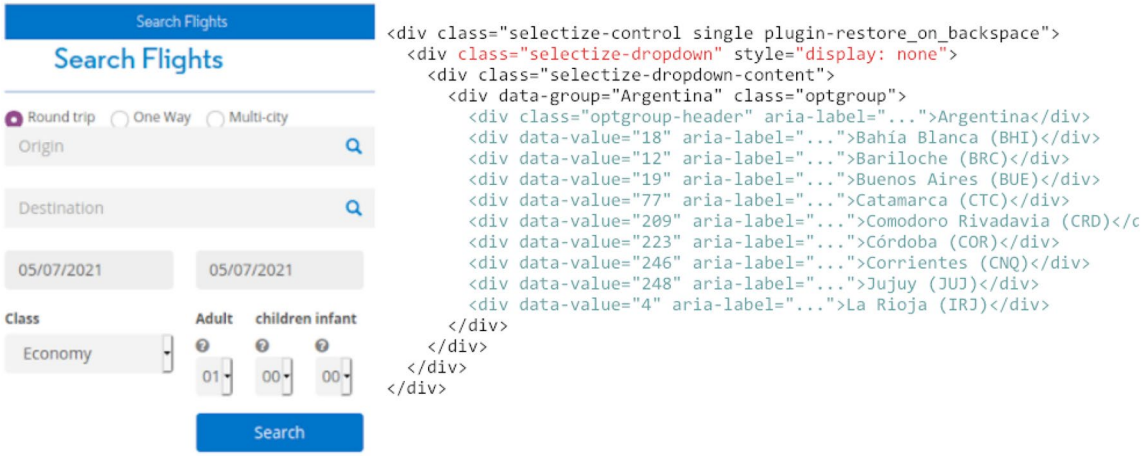

Fig. 1 Example of deleted input content. A search form shows text inputs for origin and destination for a flight, but the entered text can be deleted by a popup select box. The source code for this is also shown

Matters get worse when considering dynamic content and when this interaction involves some kind of assistive technology, like a screen reader [3, 4]. Single Page Applications (SPA), and more generally, Rich Internet Applications (RIAs) [8] usually rely on dynamic state changes of interface components to resemble desktop-like interactions. This enables complex interactive processes, but at the same time produces less accessible interfaces. To alleviate the lack of accessibility in RIAs, the Web Accessibility Initiative defined WAI-ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications), a specification that defines HTML attributes to better describe their semantics like roles or states, with special focus on screen readers. Currently, several frameworks provide ARIA-compatible components, although there are still several challenges. Ensuring compliance can be difficult, and automated approaches are usually limited to HTML structure, so several accessibility flaws may still be unidentified unless interaction-based analysis is performed [3].

As an example, Fig. 1 shows a case of an accessibility problem that occurs with dynamic content. The screenshot on the left shows the flight search form on the Aerolineas Argentinas website.[1] An origin location is required for the search. If the user enters a city name that is not available, the value gets deleted after exiting the field without a notification. As can be seen at the right side of Fig. 1, when loading the page, the HTML code includes a div container with the class “selectize-dropdown” in a “display: none" style that makes it invisible. This widget is designed so that when it receives the focus on an input field, it becomes visible. These mutations in the structure of the DOM of the page as a result of user interactions may include accessibility difficulties, and they cannot be detected by accessibility evaluation tools that analyze static code.

User tests are an excellent source of empirical data that may provide behavioral indicators for accessibility problems that occur dynamically. Several of these problems have been identified in previous works and cataloged as Accessibility Smells [14]. Similarly to code smells, Accessibility Smells may indicate design problems, though in this case, the problems affect the accessibility of a web page as perceived by its users instead of internal code quality. Moreover, Accessibility Smells are also solved by applying transformations that do not affect the underlying functionality, called Accessibility Refactorings. Some Accessibility Smells may be found by inspecting code just like code smells, for example “Unpredictable size” for lists and tables [13], or the absence of some WAI-ARIA label. Nevertheless, in this work we are interested in Accessibility Smells of user interaction, that is, those that appear in poorly designed interaction paths for screen reader users. Examples are long navigation paths, useless link activations or defective/inaccessible validation messages. Note that these smells may only be found during user testing or by analyzing interaction logs. Meanwhile, the refactorings that solve these smells apply transformations that, while preserving functionality, are meant to improve the way disabled users access the application’s content or trigger its behavior. Examples of these refactorings are “Merge pages" (to solve long navigation paths) and “Add content summaries" (as a possible solution for useless link activations).

While user tests are good sources of real usage data, they require the time and budget to hire users and usability/accessibility experts that may detect problems by observing users’ interactions while performing the test. Motivated to provide

[1] https://www.aerolineas.com.ar.

an affordable solution for small- to medium-sized companies that allows them to evaluate the accessibility of their RIA applications without consuming so many resources, the goal of our work is to find an automatic detection mechanism for Accessibility Smells.

Earlier work has been proven effective to automatically detect Usability Smells by using a service with minimal setup that collects interaction events from real users [15, 16]. By capturing problematic sequences of interaction events, called "Usability Events", Usability Smells can be detected on the fly and suggest solutions in terms of "Usability Refactorings". In order to create an automated interaction-based accessibility analysis, we take advantage of the existing approaches for Usability Smell detection. However, there are fundamental differences between the cursor-based interaction that is typical of sighted users and interaction with screen readers, not only in the way elements are accessed, but also on the threshold values for pauses.

The work that we describe in this article is intended to characterize problematic patterns of interaction events, called Accessibility Events, and show through an empirical study how they may be used to automatically detect Accessibility Smells in dynamic web applications. Going back to the example shown in Fig. 1, we propose to detect that problem with the Accessibility Event "Deleted Input Content". It occurs on a text input field when the data entered by the user is automatically deleted without reporting it, possibly as a consequence of some data validation. Moreover, we have performed a preliminary study with visually impaired users in order to find suitable threshold that would allow us to automatically detect Accessibility Smells.

Summarizing, the contributions of this article are:

- A proposal of the concept of Accessibility Events as a way to detect Accessibility Smells automatically in applications with dynamic content;
- A catalog of Accessibility Events and the Accessibility Smells that these events allow identifying;
- The results of an empirical study with visually impaired participants in which we tested the automatic detection of Accessibility Events and study their potential to lead to Accessibility Smells.

2 Related work

There are a large number of studies related to the static evaluation of websites, either for a general or specific domain like education [5, 17, 26], government [20, 24] or healthcare [29, 30]. The evaluation method used is mainly based on automatic tools like AChecker [5], which check compliance with accessibility guidelines (WCAG), although some studies apply user testing [29] or a combination of automatic and manual inspection [30]. The use of automatic tools (like Hera, TAWS, AChecker) has the advantage of being much cheaper, but the disadvantage of not considering changes that occur dynamically.

Apart from static evaluation, there are several works on the evaluation of dynamic content, that is, the content found in Rich Internet Applications (RIAs) [3]. The enhanced interactivity and dynamic nature of RIAs make them very hard to evaluate and, at the same time, cause the most accessibility problems found on the web, since screen reader users are rarely able to perceive the dynamic updates that occur on the page [2, 7]. While the WAI-ARIA framework may help web developers to improve accessibility by defining special attributes for dynamic components, they may not use it properly, and automated methods for checking compliance face several challenges [3]. Surprisingly, a study over a million homepages detected that those with ARIA labels have an average of 70% more detected errors than those without ARIA labels [28]. Moreover, it was shown through an experiment that even when using WAI-ARIA compliant components, mobile interfaces still carry significant accessibility problems for screen reader users [7]. Thus, it becomes crucial to have the support of tools to assess their accessibility.

The work of Zhang et al. 2017 [31] shows that some screen reader users make use of the shortcuts to skim over portions of text. This kind of behavior can only be observed by analyzing interaction events, which is the core of our proposal. This was also observed by Antonelli et al. [3], who have recently surveyed current tools for accessibility evaluation of RIAs, discussing several limitations which show that this is still a challenging problem, and no tool has been entirely successful. Some of the limitations of RIA evaluation tools are related to the difficulty in identifying RIA components in source code [3]. In other cases, the evaluation of dynamic content is performed on the DOM generated after page load [9, 22]. While this is more accurate than static analyzers, it does not consider user interactions; hence, many accessibility issues can go undetected [10].

To overcome that problem, some works attempt to identify dynamic state changes by simulating user events [6, 10], or keyboard events [18, 27]. The advantage of this method is that it may find more accessibility issues than considering the source code of individual widgets in isolation, but as a disadvantage, simulating events may cause a downgrade on performance and may create noise on events that may never occur in real interactions. In the case of Watanabe et al. [27], their approach relies on acceptance tests over keyboard events. Our approach is similar in that we also consider a sequence of keyboard events, although these are not simulated but real, and our "acceptance criteria" is defined in a completely generic way, which does not depend on the domain, page structure or specific ARIA labels in widgets,

but on a symptomatic sequence of events. Bostic et al. recently developed the Demodocus framework for automated accessibility evaluation specialized in the dynamic web, finding more violations than human evaluation baseline [6]. Although they do tackle JavaScript-based applications, this tool is still based on guidelines.

The web refactoring approach [12] proposes the use of the refactoring technique, originally defined as changes that improve internal quality of software [11], to also identify changes that improve external quality factors like usability or accessibility [14]. Particularly, problematic patterns of user interaction have been catalogued as Usability Smells, which can be automatically detected while real users navigate a web application [15] or perform user tests [19]. Moreover, solutions to each Usability Smell have been proposed in the form of Client-Side Web Refactorings (CSWR), scripted changes applied on the DOM structure of web pages in web browsers [15]. The CSWR approach has been proposed to improve web accessibility and personalization [13], but there have not been previous attempts, to our knowledge, for the automated detection of their corresponding Accessibility Smells.

3 Accessibility smells and events

In this work, we will characterize accessibility problems involving user interaction as Accessibility Smells [14]. The concept of ”Smell” helps to indicate usual problems, and makes it easy for developers to determine when they need to apply a Refactoring. It was originally defined for “Code Smells”, but later extended to other areas. In this proposal, we aim at creating a catalog of smells and refactorings for web accessibility. Since we intend to detect Accessibility Smells automatically from interaction logs, we also defined Accessibility Events (AE), which have better detail than plain JavaScript events. The AEs may be considered a scaffold in the process of automatically detecting smells.

Our hypothesis is that by analyzing user interaction it is possible to recognize patterns of interaction (AE) that reveal accessibility difficulties in their behavior. In turn, processing the bulk of AEs generated by from several users will allow to detect Accessibility Smells in a web application. The AEs were defined after characterizing the Accessibility Smells, by studying the micro-behaviors that usually lead to the problems.

Some of the AEs involve measuring interaction features like repetitions or durations, so in these cases the thresholds were set according to a preliminary study. The study consisted of capturing AEs from a group of 5 users with visual difficulties, performing tasks on two open source Web applications (medical appointments and e-commerce). During these sessions, the AEs thresholds were relaxed to maximize sensitivity. From the complete set of AEs, we first removed outliers and then manually processed the remaining ones to determine which were considered problematic (i.e., helpful for capturing smells) and set the thresholds accordingly.

The rest of this section explains and exemplifies both accessibility events and smells, each with a catalogue.

3.1 Accessibility events

Accessibility Events are short interaction patterns that help reveal accessibility problems. They can be detected automatically from user input by analyzing low level events and composing them to generate more abstract events that describe the way visually impaired users browse the web.

We define AE as behavioral patterns while navigating or accessing web content, which may indicate the presence of some accessibility issue, i.e., an accessibility barrier that can be detected while a user interacts with the application. In particular, we concentrate on barriers for visually impaired users accessing web applications with screen readers (SR). Each AE is represented as an aggregation of interaction instances automatically detectable on an interface. An example of an accessibility event is Frequent Tab. This event, when detected, can be used to diagnose the Accessibility Smell called “Keyboard-Distant Content", which indicates that there is a part of the page that is far to reach using a SR, mostly prepared for linear navigation.

The following catalog details all the currently characterized AE. Their detection relies on heuristics based on existing literature, and user observation.

E01 - deleted input content: this occurs on a text input field when the data entered by the user is automatically deleted without a clear indication, possibly as a consequence of data validation. This AE is not frequent nowadays with large forms, but persists in several login forms.

E02 - unhelpful label: when a targetable input does not include a label, or the label is not correctly bound by the HTML attributes `for` and `id`, the screen reader will not use it as a description for the input. In these cases, the SR user will navigate the label as a separate element, or even skip it. This AE may be detected from observing both the interaction and the HTML code.

E03 - missing SR text: when a targetable input does not include a text, label or placeholder, the SR will speak a default message. This message, without a proper context, could be really unhelpful for the users. This AE is detected when users focus on an input field for a certain period of time, and this does not contain a linked label, “aria” role, or a placeholder.

E04 - frequent tab: this event characterizes bursts of consecutive `Tab` keypresses used to cycle through elements. It can be used to detect a large number of unnecessary or irrelevant “stops” in the path to reach the main

content or menu. During the preliminary studies, we found that in most cases, up to 3 tabs do not show a problematic interaction, so we used 4 consecutive tabs as a threshold to detect the event.

E05 - unfilled form: this event is raised when a form is partially completed but not submitted. To capture this event, any interaction with a form element triggers the start of the (potential) event, and then the `beforeunload` event is handled to determine how the user left the page. If a submit event is detected prior to `beforeunload`, then the event is discarded. Every Unfilled Form event saves the code of the form and the time that the user was completing the form before finally leaving it.

E06 - misleading speech synthesis: this event can be present in buttons and links, when the text that the SR synthesizes can be unintelligible to the user. Focusing excessively over an element without performing interaction may signal a confusing text synthesis, for example, terms in a different language from the user's context or the browser. To detect these particular situations, the known language of the page and its title are collected, the one of the Focus tag and predetermined of the browser.

E07 - winding tab sequence: The access sequence for focusable elements (like form links and controls) while using a SR may differ from the visual presentation on screen. Even alterations of the sequences defined in the "tabindex" attributes may not consider the impact on keyboard accessibility. This AE detects inconsistencies between the order of the widget in the visual presentation and the order of focused defined by "tabindex" (either explicit or implicit).

E08 - fast keyboard scrolling: The event occurs when the content on the web presentation is quickly moved by repeatedly pressing the space key. The recurrence of displacements exceeding portions of a page can be symptoms of accessibility difficulties present on a page.

3.2 Accessibility smells

We call Accessibility Smell of User Interaction to any accessibility difficulties that can be discovered from the analysis of user interaction logs. This concept was originally developed in a previous work [14] but we have refined it in our present research to approach the subset of smells that may be automatically discovered through algorithms that gather AEs, compose them and process them. We omitted smells that do not require interaction analysis to be discovered, hence out of the scope of this research.

For instance the smell Keyboard-Distant Content, indicates that, in order to reach a given element, the required path of `Tab` or `Shift` + `Tab` keypresses may be longer than necessary. Screen reader users generally rely on the `Tab` key to navigate, so the order in which elements are traversed is fundamental for them. Considering this, the Keyboard-Distant Content smell can be detected when users cycle through a large number of items to access the required element, indicating that this element should be closer in the navigation sequence, avoiding unnecessary effort.

Table 1 Accessibility Smells and WCAG 2.0 Techniques

Acc. smell	WCAG 2.0 guideline / Technique
D01	ARIA18, ARIA19, ARIA21, G83, G85, SCR19, SCR32, G13, G84
D02	G13, G83
D03	G83
D06	SCR18
D07	G80, G149, H32
D08	H84, SCR2
D09	G149, G199
D10	G98, H89
D11	G98, G149, H4, H89
D12	G162, H89, SCR18, G98

Our contribution focuses on accessibility in use by automatically detecting accessibility difficulties in and during interactions including metadata semantics. Note that these interactions cannot be evaluated with static guideline compliance analysis. While the AEs provide information about the actions in the interface, traceable Accessibility Smells, from these events, are related to the accessibility guidelines and their criteria defined by W3C. Some of these smells are related with known WCAG 2.0 techniques [1]. These relationships can be seen in Table 1.

D01 - unreadable validation message This smell is detected when failed data validation tests lack changes on the interface detectable by SR or whose electronic texts cannot be synthesized to notify the user. This is based on WCAG 2.0 techniques such as *ARIA18: Using aria-alert dialog to identify errors*, or *G83: Providing text descriptions to identify required fields that were not completed.* [1]. As an example of this smell we return to the case of Fig. 1, where the value is deleted after leaving the field without notification. This is because it does not provide client-side validation that adds error text via the DOM nor does it describe what will happen before a change is made to a form control that causes a context switch (as described in *G13: Describing what will happen before a change to a form control that causes a change of context to occur is made.*). In this way we can link a Deleted Input Content Event with this AS.

D02 - unlabeled input: A text input may lack a label properly linked by the HTML attribute. The text of the labels can still be synthesized but not necessarily informing the user of the expected input.

D03 - undescripted entry fields: This smell denotes the absence of associated electronic text through the label, position marker and ARIA label in an entry field. Under these conditions, SRs do not report the expected entry to the user.

D04 - keyboard-distant content: This smell refers to functionalities or contents which are placed far in the linear navigation path that SR users need to go through. Based on Distant Content Event [15].

D05 - distant accessible navigation path: Navigation routes between pages that must be crossed by remaining short intervals of time in the intermediate nodes. Based on Navigation Path Event [15].

D06 - inaccessible captcha: Inaccessible security codes included in the web forms that try to ensure the user is not a bot. These inaccessible captcha can prevent users with visual disabilities.

D07 - missing submit button: Detects web forms that do not have focusable submit buttons, which makes it impossible to send them by keyboard. Visually elaborate alternative structures are used on occasions that cannot be focused with a keyboard and only respond to mouse events.

D08 - mouse-dependent datepicker: Describes popups that offer a graphical interface to select dates from a calendar. Most of these widgets respond to events associated with the mouse, which is why users using the keyboard cannot interact with the component.

D09 - inaccessible search results message: It may occur on search forms do not provide electronic texts in the results list. When this happens, the SR cannot synthesize the description and lack of feedback may puzzle the impaired users.

D10 - unexpected language: They are literally synthesized texts in a language that is not expected by the user. This occurs when the languages of the elements on the website are not declared correctly. Difficulty may arise when text is literally synthesized in a language other than that expected by the user.

D11 - confusing layout: Web presentation that despite complying with the functional requirements have an intricate design for visually impaired users. Navigation with SRs can be difficult when the inner structure is not coherent with the visual layout.

D12 - visual-dependent context: These are navigation difficulties where the text synthesis performed by the SR is not sufficient to express the semantics of a component. This might be due to dependence on clues or arrangement that can only be perceived visually and are not described in the code or metadata.

3.3 Examples

This section shows some examples of AE, and how they can lead to the finding of specific Accessibility Smells.

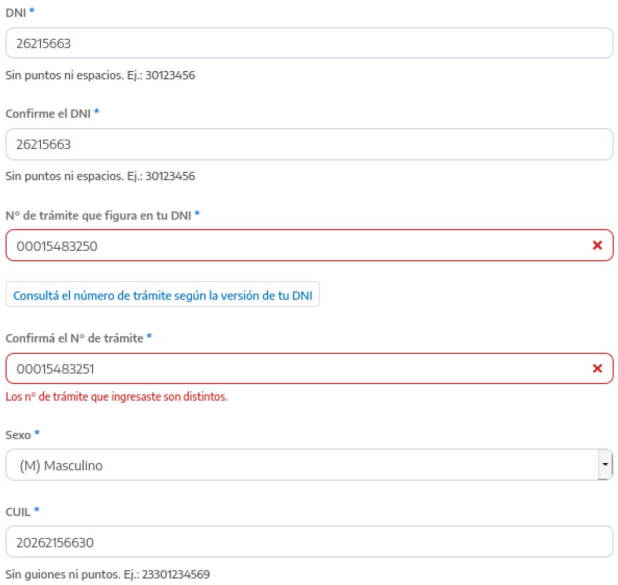

Fig. 2 Form showing an example of winding tab sequence for confusing layout smell

A case of *Winding Tab Sequence* occurs during the backing out of the keyboard focus with `Tab` and `Shift+Tab` with the intention of re-synthesizing previously entered text to verify the data entered. For instance, in the form shown in Fig. 2, the user is prompted to repeat a procedure number ("No. de trámite" in the screenshot), much like some forms ask to repeat a registration email. The procedure number is a code that is printed in the ID card in Argentina ("DNI"), which is very unlikely for people to memorize, and it is not available in braille system. In this example, *Winding Tab Sequence* reflected SR users going back to the first input, probably to revisit the first entered number. Notice also that the hint explaining where to find such number is in between the two inputs.

The recurrence of these return situations with focus can indicate the presence of some accessibility difficulty. This can happen since, unlike sighted users that can get a quick general look, SR dependent users must travel in linear way, synthesizing the electronic on each step - and also back step. In cases like this, where SR users are forced to revisit inputs, we consider the presence of a *Confusing Layout Smell*, because the representation from voice synthesis cannot guide the user well enough - even if sighted users do not have trouble with it.

The case of *Misleading Speech Synthesis* occurs when text synthesis in response to the focus of links and keypad is unintelligible to the user. In Fig. 3, if `Tab` is used after completing the input field, the SR will jump directly to the link captioned as "acá" ("here"), skipping the text label in gray ("Si no conoces el código postal podés consultarlo" / "If you don't know the postal code you may consult it"), so

Fig. 3 Example of misleading speech synthesis for visual-dependent context

only the electronic text “acá enlace" (“here link") is synthesized. The recurrence of these situations may indicate the smell *D12- Visual-Dependent Context*. In this case, there is information that is crucial for filling out the input, but SR users cannot easily access it, only the sighted users.

4 Smells detection tool

We have built a tool to automatically capture AEs, and to detect smells from them. This tool works as a service, featuring a client component that can be installed on any web application, and a server component that generates the accessibility reports in terms of smells. As the system is used, user interaction in the host application is scanned with the client component in order to detect AEs. Then, the server processes the AEs with heuristics based on reference values and tolerance thresholds in order to detect and report new smells.

The process depicted in Fig. 4 consists of two steps: Accessibility Events Capture and Accessibility Smells Detection. As we mentioned earlier, capture happens on the client, and smells analysis happens on the server. The client side component evaluates user interactions by picking up JavaScript events, filtering and grouping them into the more abstract AEs. The server component then classifies and analyzes those events to discover Accessibility Smells.

In the example presented in the Introduction, the tool detects a “Deleted Input Content” event, which could in turn lead to reporting the smell “Unreadable Validation Message”. To do that, the client-side component observes the value entered in the “flight origin” input and compares it to the value after the blur event. The automatic deletion of the entered value is captured as a Deleted Input Content event. If many of these events happen on the same input field, the server component can determine that the validation message cannot be picked up by the Screen Reader.

Internally, the server side has a set of components called finders, each one dedicated to find a specific smell, which consumes and analyzes one or more types of AEs. They are configured with certain parameters found through experimentation, which define the number, proportion, or combination of AEs that trigger the presence of a specific smell in each finder.

In this work, we mostly analyzed the report of AEs generated by the tool, to measure the effectiveness of the approach and threshold values.

5 Evaluation

In order to evaluate the AE detection system, we used our tool to analyze interactions generated by a group of volunteers while completing typical tasks in replicas of three web applications with known accessibility smells. The objective of the study was to contrast the detections of accessibility events related to the smells (true positives) with those irrelevant or not leading to any cataloged smell. In this early stage we focus on the detection of AEs for two reasons (1) the events are the means to detect smells and we must establish their correct detection and (2) the detection of smells requires quantities of event substantial and will require a different evaluation.

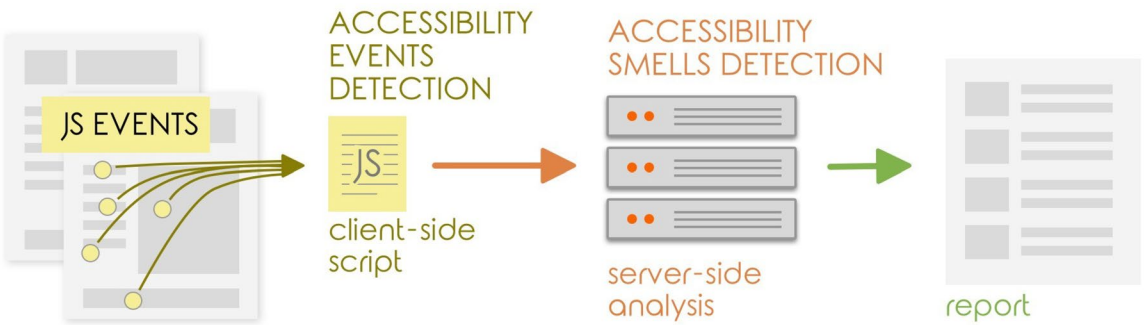

Fig. 4 Schematics of the detection process for accessibility events detection and accessibility smells reporting

5.1 Participants

We recruited a total 8 volunteers, 3 female and 5 male with mean age 31.5 (s2 5.375), 5 from Argentina and 3 from Central America, frequent users of NVDA and social networks. One of them declared 90% reduced vision, 5 indicated to be totally blind, 1 having a visual reduction of less than 30% and 1 only declared to have difficulty in their vision. One performed the test at a local work station, while the remaining interacted through a remote accessories to the SR without having physical access to the screen.

5.2 Preparation

The web applications used for the evaluation were 2 teaching platforms, Siu Guarani (SIUG) [2] and Siu Tehuelche (SIUT),[3] and a COVID-19[4] circulation permit form (COVID-19). To preserve the privacy of the volunteers, we set up replicas of all 3 applications. The tasks consisted in creating an account in each of the teaching platforms and finally requesting a certificate from the COVID-19 circulation permit website. We gave each participant instructions on the test software, a description of the required tasks, their estimated duration and some sample data to fill out (except for personal data).

5.3 Results

The tool detected a total 100 Accessibility Events of 6 different types. We evaluated each one of these events to determine if they were either “linked" with any of the catalogued Accessibility Smells, “unlinked" in the cases where the events occurred for unrelated reasons, or “undetermined", in the cases where the events did not provide sufficient evidence to determine or rule out the incidence on a concrete smell. Table 2 summarizes the detected Accessibility Events. Out of the total 100, we marked 64% as linked, 19% as unlinked, and 17% as undetermined. We next describe the details of each identified event.

Out of the 8 *Frequent Tab* events, 3 were reported on the *COVID-19* application and 5 on *SIUG*. In the SIUG events, we observed input fields accesses with bursts of between 5 and 6 consecutive tabs. Since the fields are not distant on the form, we can say that the users performed these actions in an attempt to check the content, structure and/or other data offered by the SR of the Web component. In the case of COVID-19, at the top of the page there is an invisible but focusable link with the legend “Go to main content". Analyzing the data, we observed two events in which more than 12 consecutive tabs are pressed to change the focus from this link to other fields of the page. This suggests that users quickly went through the form looking for something about the presentation without paying attention to the content in between. In the remaining event in COVID-19, the user went from the name of the doctor’s field to a check to indicate lack of symptoms. This can be interpreted as a path between fields for which the user may have not had information at hand, such as the zip code and address of the doctor, and decided to continue browsing with `Tab`.

Even if the amounts of consecutive tabulations in SIUG could have pointed at *Keyboard-Distant Content*, in this case a *Confusing Layout* is more likely to be the cause, since there was not a great distance between the elements. Similarly, for COVID-19 and considering the dimensions of the form, a *Visual-Dependent Context* could be more accurate, indicating that users wandered around for clues.

The 2 *Missing SR text* detected were found in the “Country" entry of residence of the *COVID-19* site. One for the entry of the residence locality of the applicant and another for the town the applicant had to attend. We considered these linked to the smell *Undescripted Entry Fields* because in the absence of a text hint, the SR synthesized the message *“this field is required, edit required invalid blank entry"* which was not informative for the field.

Of the 20 *Unfilled form* events, 8 were detected in SIUT, 8 in COVID-19 and the remaining 4 in SIUG. The latter were classified as “undetermined" since we could not determine the reason why users abandoned the form. In SIUT, in Fig. 5, there is a compound input, with 2 fields (type of DNI, number of DNI), and the focus is automatically set to the second one. In this context, SR users have no obvious way of telling the first part was skipped (or that there is a first part at all), which lead to a validation error that was too difficult to figure out. To make things worse, the error text did not clearly indicate the missing input, leading to the form abandonment.

Table 2 Captured accessibility events, along with an indication of relationship to a catalogued smell

Type	Count	Linked	Not linked	Undetermined
Frequent tab	8	8	–	–
Missing SR text	2	2	–	–
Unfiled form	20	16	–	4
Unhelpful SR text	5	5	–	–
Winding tab sequence	57	28	17	12
Misleading speech synthesis	8	5	2	1
Total	100	64	19	17

[2] Siu Guarani web site https://autogestiong3.unsa.edu.ar/, last accessed May 2022.

[3] Siu Tehuelch site http://cdcsiu.unsa.edu.ar/siu/tehuelche/, last accessed May 2022.

[4] Circulation permit form https://www.argentina.gob.ar/circular, last accessed May 2022 and unavailable after pandemic.

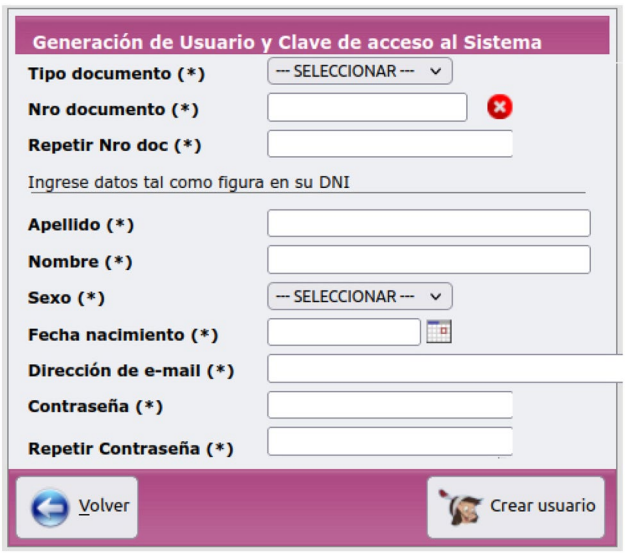

Fig. 5 Form affected by unfilled form in SIUT

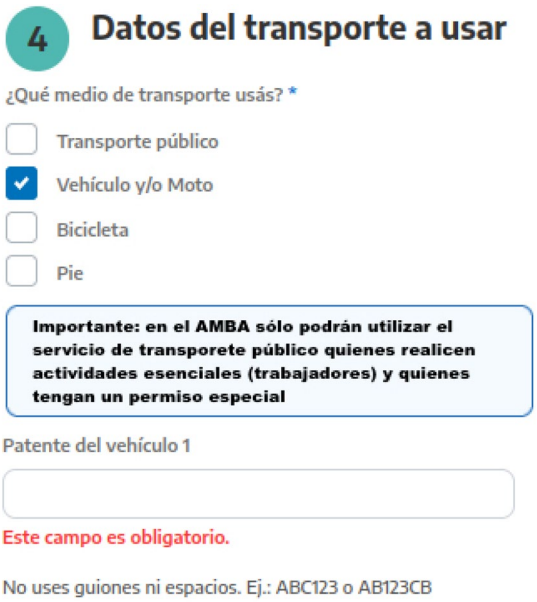

Fig. 6 An input field for a mandatory vehicle license place input in COVID-19

In this condition of disordered focus that confuses the user and prevents the submission of the form, it is more accurate to consider this as a potential *Confusing Layout* smell.

In the case of COVID-19, the extension of the form and its numerous mandatory input fields hinder completeness due to data required that is unlikely to be remembered by the users. We can also find an instance of inaccessible RIA application behavior, as seen in Fig. 6: when users indicate a particular vehicle and/or motorcycle, a license plate input will be dynamically required. These interactions are difficult to detect with current automatic tools and can significantly affect accessibility during the use of the page. Although we do not typify a particular Smell for this situation, these AEs denote accessibility difficulties and were marked as “linked" in consequence.

Table 3 Winding tab sequence event

	Smell Relationship			
Site	YES	NO	N/S	Total
SIUG	13	1	2	16
COVID-19	11	15	9	35
SIUT	4	1	1	6
Total	28	17	12	57

Out of the 5 *Unhelpful SR Text* detected in COVID-19, in the apartment data entry for voice synthesis is ineffective for users, indicating a potential Unlabeled Input smell. In these cases SR cannot retrieve a label for the input field. This happens in the field for the address to which the applicant must go: when focused, users are prompted with a validation error (“mandatory field"), which doesn’t provide sufficient information to amend the missing data. These AE occurrences are linked to the *Unlabeled Input* smell.

The distribution of the 28 events *Winding Tab Sequence* detected at the sites **(SIUG, SIUT and COVID-19)** is described in Table 3.

Regarding this particular AE, the focus interactions detected on *SIUG* and *SIUT* reflect erratic sequences. Analyzing the data more closely, we found a behavior that is quite common among SR users. They rapidly skip between tags, which allows them to orient themselves within the structure of the page. This would allow us a priori to infer that the semantics returned by the SR is unclear or that the electronic texts could be insufficient and that such a condition could be associated with difficulties of *Confusing Layout* or *Visual-Dependent Context*. However, in *SIUG* we found a particular case, in which a link outside of the form structure could lead to navigation sequences far different from the one designed in the visual presentation. For *SIUT* we found one of the winding paths that focused on the input of the ID type, which we analyzed in Fig. 5 for the event Unfilled Form. This shows that this situation requires to overcome the difficulty an additional navigation or outside the one planned in the design. We can add that *SIUG* and *SIUT* contain forms that distribute the inputs in a vertical presentation order in the first case and combined, between horizontal and vertical in the second, and since they belong to the same organization and try to appear uniform in their designs they should preserve the styles in their forms for the sake of accessibility.

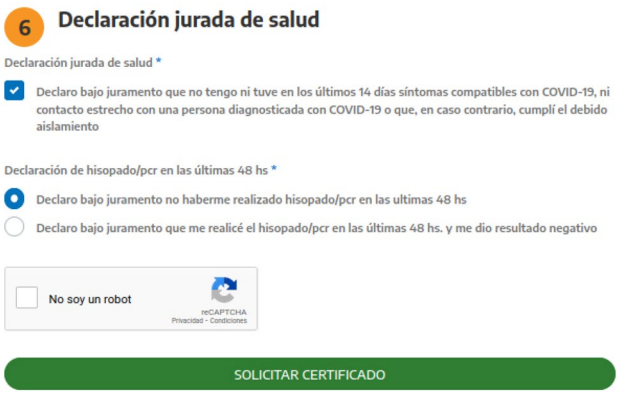

Fig. 7 Instance of a winding tab sequence for confusing layout smell

There is another example of *Winding Tab Sequence* in COVID-19, in the form in Fig. 7. The options presented as *Check Box* and *Radio Button* contain sworn statements with legal implications, which require an additional cognitive effort to be interpreted together. Notice that in this condition, a sighted user alternates the focus between the texts, analyzing to decide quickly without being conditioned by the events on the application.

In the analyzed case, when the *Check Box* was focused, the linked text was synthesized, but after focusing and synthesizing the *Radio Button* text, which required a new sworn statement, the user returned the focus from the SR to the previous *Check Box*, probably because now he has more information and can reconsider the previously selected action. If these actions were to be required recurrently, it would be necessary to adopt measures in view of the accessibility barrier. In short, in all these cases the positions of the elements in the screen differ from the typically traversed SR paths. Visually impaired users can be affected with the *Confusing Layout* smell.

In the case of *Misleading Speech Synthesis*, 2 events were detected on a button with the “Back" caption, which coincides with Fig. 5 on SIUT and 3 events in links whit captioned as “here” (“acá” in original Spanish) in COVID-19, which coincides with Fig. 3. These text literals are not enough to describe the functionality to which it accesses through the widget. A significant increase in reports of these conditions could indicate the presence of the *Visual-Dependent Context* smell.

5.4 Discussion

The results and interpretation described in the previous section show that there is a generation of accessibility problems for visually impaired users that cannot be detected with traditional techniques. Even if there is still work ahead to actually detect Accessibility Smells from the AEs in the study, results are promising. In our own work in accessibility refactoring [13], we created a catalog of Accessibility Smells, but these were in some cases too general and difficult to detect from JS events. In this work, we specialized that catalog to feature only those smells that can be detected from behavior logs.

These results could be contrasted with other tools that consider the dynamic behavior of RIAs, such as Watanabe et al. [27]. That work is based on detecting DOM changes, so there are AEs that could also be detected with their technique (E01, for instance). However, even if there is some overlapping, the kind of problems they will find are different, since their work is mainly focused on drop-down menus and they do not analyze user behavior directly. Another previous work of ours uses a similar technique [25] and could be combined with the present proposal nonetheless.

5.5 Threats to validity

The relatively low amount of AEs captured can be a source of bias. This was mainly due to the difficulty of recruiting volunteers with visual disabilities, even more difficult in the context of the COVID-19 Pandemic (which is why almost all recorded sessions were remote). Therefore, we describe associations between Events and Smells that could provide indications on causal relationships linked to accessibility difficulties. Additional experimentation is required to gather sufficient evidence for a more decisive formulation with respect to the detection of Accessibility Smells.

The two different access configurations for the SR, remote and local, could lead to an internal validity problem. We favored the remote configurations because they gave us access to volunteers we could not meet personally anyway (foreigners).

In the particular case of the volunteers residing in Central America, it may have been confusing for them to interpret terms in the contents due to regionalisms or expressions used in the Argentinian pages. For example, the Labor Identification Code corresponds to a number assigned by the National Social Security Authority to Argentine residents. The websites replicated to collect empirical data emulate real web pages that were downloaded and housed on dedicated servers, so this experimental context could differ from the real one. We tried to mitigate this by imitating the real actions of the web site, but at the same time preventing volunteers from being worried about entering personal data, even though we incurred a slight bias of external validity.

6 Conclusions

In this paper we presented a way of detecting accessibility problems from interactions that is oriented to visually impaired users of Screen Readers. It is based on the concepts

of Accessibility Event and Smell to facilitate the detection and characterization of the accessibility problem on the front end of a web site. We also presented two catalogs, one for Accessibility Events and another one for Accessibility Smells that show the scope and relationship of these concepts. We validated this technique with an evaluation where we show that automatic detection is possible and could lead to the eventual detection of concrete problems in terms of Accessibility Smells.

During the evaluation we found that most of the events are related to Accessibility Smells (64%), while the rest are either not clearly linked (19%) or not feasible to determine (17%), for which further analysis is required.

We plan to expand our evaluation to a larger scale, so that we can test the feasibility of automatic detection of accessibility problems of user interaction. We are also expanding the Events and Smells catalogs from new observations.

Acknowledgements This research was partially funded by the Argentinian National Agency for Scientific and Technical Promotion (ANPCyT), grant number PICT-2019-02485.

Author Contributions All authors contributed to the creation of the approach, design of the evaluation, as well as the writing and revision of the manuscript. Fernando Durgam executed the evaluation.

Data availability The datasets generated and analyzed during the current study are not publicly available to preserve volunteers' privacy but are available from the corresponding author upon reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Techniques and failures for web content accessibility guidelines 2.0. URL https://www.w3.org/TR/WCAG20-TECHS/Overview.html
2. Almeida, Leonelo Dell Anhol., Baranauskas, Maria Cecília Calani. Accessibility in rich internet applications: People and research. In Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems, IHC '12, page 3-12, Porto Alegre, BRA, 2012. Brazilian Computer Society. ISBN 9788576692621. https://doi.org/10.5555/2393536.2393538
3. Antonelli, Humberto Lidio, Sensiate, Leonardo, Watanabe, Willian Massami, de Mattos Fortes, Renata Pontin. Challenges of automatically evaluating rich internet applications accessibility. In Proceedings of the 37th ACM International Conference on the Design of Communication. ACM, October 2019. https://doi.org/10.1145/3328020.3353950
4. Baazeem, Ibtehal S., Al-Khalifa, Hend S.: Advancements in web accessibility evaluation methods: How far are we? In Proceedings of the 17th International Conference on Information Integration and Web-Based Applications & Services, iiWAS '15, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334914. https://doi.org/10.1145/2837185.2843850
5. Barricelli, Barbara Rita, Casiraghi, Elena, Dattolo, Antonina, Rizzi, Alessandro: 15 years of stanca act: are italian public universities websites accessible? Univ Access Inf Soc **20**(1), 185–200 (2020)
6. Bostic, Trevor, Stanley, Jeff, Chudnov, Daniel, Higgins, John, Tracy, Brittany, Brunelle, Justin F.: Demodocus: Automated web accessibility evaluations. In ICT Accessibility Testing Symposium: Time for Testing in Testing Times (Remote Work, Commerce, Education, Support...), page 81, 2020
7. Carvalho, Lucas Pedroso, Ferreira, Lucas Pereira, Freire, André Pimenta.: Accessibility evaluation of rich internet applications interface components for mobile screen readers. In Proceedings of the 31st Annual ACM Symposium on Applied Computing, SAC '16, page 181-186, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450337397. https://doi.org/10.1145/2851613.2851680
8. de Mattos Fortes, R.P., Antonelli, H.L., Watanabe, W.M.: Dynamic web content. In: Human-Computer Interaction Series, pp. 373–395. Springer, London (2019)
9. Duarte, Carlos, Salvado, Ana, Akpinar, M. Elgin, Yeşilada, Yeliz, Carriço, Luís.: Automatic role detection of visual elements of web pages for automatic accessibility evaluation. In Proceedings of the 15th International Web for All Conference, W4A '18, New York, NY, USA, (2018). Association for Computing Machinery. ISBN 9781450356510. https://doi.org/10.1145/3192714.3196827
10. Fernandes, Nádia, Batista, Ana Sofia, Costa, Daniel, Duarte, Carlos, Carriço, Luís.: Three web accessibility evaluation perspectives for RIA. In Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility - W4A '13. ACM Press, (2013). https://doi.org/10.1145/2461121.2461122
11. Fowler, Martin: Refactoring: improving the design of existing code. Addison-Wesley Professional, Addison-Wesley, USA (2018)
12. Garrido, Alejandra, Rossi, Gustavo, Distante, Damiano: Refactoring for usability in web applications. IEEE Softw **28**(3), 60–67 (2011). https://doi.org/10.1109/ms.2010.114
13. Garrido, Alejandra, Firmenich, Sergio, Rossi, Gustavo, Grigera, Julian, Medina-Medina, Nuria, Harari, Ivana: Personalized web accessibility using client-side refactoring. IEEE Internet Comput **17**(4), 58–66 (2013). https://doi.org/10.1109/mic.2012.143
14. Garrido, Alejandra, Rossi, Gustavo, Medina, Nuria Medina, Grigera, Julián, Firmenich, Sergio: Improving accessibility of web interfaces: refactoring to the rescue. Univ Access Inf Soc **13**(4), 387–399 (2013). https://doi.org/10.1007/s10209-013-0323-2
15. Grigera, Julián, Garrido, Alejandra, Rivero, José Matías., Rossi, Gustavo: Automatic detection of usability smells in web applications. Int J Human Comput Stud **97**, 129–148 (2017). https://doi.org/10.1016/j.ijhcs.2016.09.009
16. Grigera, Julian, Garrido, Alejandra, Rossi, Gustavo. Kobold: Web usability as a service. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, October 2017b. https://doi.org/10.1109/ase.2017.8115717
17. Máñez-Carvajal, Carlos, Cervera-Mérida, Jose Francisco, Fernández-Piqueras, Rocío: Web accessibility evaluation of top-ranking university web sites in spain, chile and mexico. Univ Access Inf Soc **20**(1), 179–184 (2019). https://doi.org/10.1007/s10209-019-00702-w
18. Mesbah, A., van Deursen, A., Roest, D.: Invariant-based automatic testing of modern web applications. IEEE Trans Softw Eng **38**(1), 35–53 (2012). https://doi.org/10.1109/tse.2011.28
19. Paternò, Fabio, Schiavone, Antonio Giovanni, Conti, Antonio. Customizable automatic detection of bad usability smells in mobile accessed web applications. In Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services. ACM, September 2017. https://doi.org/10.1145/3098279.3098558

20. Paul, Surjit, Das, Saini: Accessibility and usability analysis of indian e-government websites. Univ Access Inf Soc **19**(4), 949–957 (2019). https://doi.org/10.1007/s10209-019-00704-8
21. Scanlan, Mark: Reassessing the disability divide: unequal access as the world is pushed online. Univ Access Inf Soc **21**(3), 725–735 (2021). https://doi.org/10.1007/s10209-021-00803-5
22. Schiavone, Antonio Giovanni, Paternò, Fabio: An extensible environment for guideline-based accessibility evaluation of dynamic web applications. Univ Access Inf Soc **14**(1), 111–132 (2015). https://doi.org/10.1007/s10209-014-0399-3
23. Henry, Shawn Lawton. WCAG 2 Overview, 2022. URL https://www.w3.org/WAI/standards-guidelines/wcag/
24. Siqueira, M.S.S., Nascimento, P.O., Freire, A.P.: Reporting behaviour of people with disabilities in relation to the lack of accessibility on government websites: analysis in the light of the theory of planned behaviour. Disabil CBR Incl Dev **33**(1), 52 (2022)
25. , Toledo Maximiliano, Grigera, Julián, Garrido, Alejandra. Detección automática de problemas de accesibilidad a partir de eventos de interacción de usuario. In Anais do XXV Congresso Ibero-Americano em Engenharia de Software, pages 128–142, Porto Alegre, RS, Brasil, (2022). SBC. https://doi.org/10.5753/cibse.2022.20968
26. Silas Formunyuy Verkijika and Lizette De Wet: Accessibility of south african university websites. Univ Access Inf Soc **19**(1), 201–210 (2018). https://doi.org/10.1007/s10209-018-0632-6
27. Watanabe, W.M., Fortes, R.P.M., Dias, A.L.: Acceptance tests for validating ARIA requirements in widgets. Univ Access Inf Soc **16**(1), 3–27 (2015)
28. WebAIM Institute. The WebAIM Million, 2022. URL https://webaim.org/projects/million/
29. Yong Jeong Yi: Web accessibility of healthcare web sites of korean government and public agencies: a user test for persons with visual impairment. Univ Access Inf Soc **19**(1), 41–56 (2018)
30. Youngblood, Norman E., Capanoglu, Muhammet Fehmi, Sesek, Richard: The accessibility of state occupational safety and health consultation websites. Univ Access Inf Soc **20**(1), 85–92 (2020)
31. Zhang, Mengni, Wang, Can, Yu, Zhi, Shen, Chao, Bu, Jiajun. Active learning for web accessibility evaluation. In Proceedings of the 14th International Web for All Conference. ACM, (2017). https://doi.org/10.1145/3058555.3058559