A Crawljax Based Approach to Exploit Traditional Accessibility Evaluation Tools for AJAX Applications

F. Ferrucci[1], F. Sarro[1], D. Ronca[1], S. Abrahao[2]

Abstract In this paper, we present a Crawljax based approach to automatically evaluate the accessibility of AJAX applications. Crawljax is a tool able to crawl an AJAX application for inferring a corresponding state-flow graph. Thus, combining Crawljax with a traditional tool for accessibility testing we realized a plugin that provides an automatic generation of accessibility evaluation report for AJAX applications. The proposed approach has been experimented carrying out a case study that analyzed the accessibility of Google Search and AskAlexia, as representative of Web applications that use AJAX technology. The case study highlighted the effectiveness of the approach based on the use of a state-based representation to automate the accessibility evaluation of AJAX applications. Nevertheless, it also revealed some shortcomings of the current implementation of Crawljax that should be addressed to make its exploitation in this context more reliable.

1. Introduction

Recently the Web is changing significantly; we are assisting the transition from static pages to dynamic content and rich interaction. Many recent Web applications are based on AJAX technology. AJAX allows achieving a high level of user interactivity through a combination of different technologies, such as XHTML, CSS, JavaScript and XML, and asynchronous communication between client and server. Shifting from the asynchronous request–response protocol to one based on asynchronous communications, allow us to request and serve content without having to refresh the entire page, making user interface more responsive and reducing the delay in user experience. However, in spite of these benefits AJAX technology brings a set of new challenges as well [11]. Indeed, the replacement of the synchronous request-response protocol with asynchronous communications makes AJAX applications very different from traditional Web applications. These latter

[1] University of Salerno, Dipartimento di Matematica e Informatica, Via Ponte don Melillo, 84084 Fisciano (SA), Italy, {fferrucci, fsarro}@unisa.it

[2] Universidad Politécnica de Valencia Camino de Vera, s/n, 46022 Spain, sabrahao@dsic.upv.es

are based on the multi-page interface paradigm where each page has a unique URL, while AJAX applications can consist of a single-page with a single URL that dynamically changes state. This aspect makes the evaluation of AJAX applications accessibility not a trivial task since the intrinsic highly dynamic nature of these applications makes very difficult and time consuming to manually examine whether all the states of an AJAX application meet certain accessibility requirements. As a matter of fact, existing tools employed to evaluate accessibility of traditional Web applications are not appropriate for AJAX applications because they are able to evaluate only static HTML pages and ignore all the dynamic elements that are the main components of an AJAX application. A way to address this challenge is the use of a crawler able to explore all the dynamic states of an AJAX application and build a traditional web navigational model which can be used to test the generated static pages of the AJAX applications. Crawljax [10] is one of the most promising tools concerning automatic crawling of AJAX applications. Indeed it was successfully employed in previous study to explore automatic testing of AJAX applications [3][8][9][11][12].

To the best of our knowledge there are no works in the literature exploring automatic evaluation of accessibility in AJAX applications. In this paper, we present a Crawljax based approach to automatically evaluate the accessibility of AJAX applications. In particular, we realized a Crawljax plugin that provides an automatic generation of accessibility evaluation reports for AJAX applications exploiting the finite state machine inferred by Crawljax and a traditional accessibility evaluation tool. The proposed approach has been experimented carrying out a case study that evaluated the accessibility of Google Search [6] and AskAlexia [2], as representative of Web applications that use AJAX technology. The case study revealed the effectiveness of the approach based on the use of the Crawljax state-based representation to address the accessibility issue for AJAX applications. Nevertheless, it also reveals some shortcomings of the current implementation of Crawljax that should be addressed to make its exploitation in this context more reliable.

The paper is organized as follows. Section 2 discusses the challenges for accessibility evaluation of AJAX applications. Section 3 recalls the main features of Crawljax. Section 4 presents the Crawljax plugin we developed for evaluating the accessibility of AJAX applications. Section 5 reports on a case study where the plugin is validated in terms of its effectiveness and performance. Finally, section 6 presents conclusions and future work.

2. Evaluating Accessibility of Ajax Applications

Accessibility is a crucial aspect of Web applications. The World Wide Web Consortium (W3C) proposed standard guidelines [15][16] to support developer in making accessible Web sites and a working draft was proposed to suggest techni-

cal accessibility specification of Rich Internet Applications [1]. Also a conformance evaluation method of Web site accessibility is suggested by W3C to determine if a Web site meets accessibility standards, such as the Web Content Accessibility Guidelines (WCAG) [15][16]. Such conformance evaluation method combines some manual checking along with the use of several semi-automatic or automatic accessibility evaluation tools. Indeed, simple manual techniques such as changing settings in a browser can determine if a Web page meets some accessibility guidelines. A comprehensive evaluation to determine if a site meets all accessibility guidelines is much more complex and there are several evaluation tools [5] that help with this evaluation. However, there not exists a single tool which determines if a site meets all accessibility guidelines. Indeed, each tool is capable to identify specific accessibility issues depending on the guidelines taken into account, the types of automatic checking provided, and the web page formats supported. Thus, evaluating web sites for accessibility can be a non trivial task especially in case of large web site or sites that uses rich technology and contents. As a matter of fact, the highly dynamic nature of AJAX applications makes ineffective the use of traditional accessibility evaluation tools. Indeed, these tools are able to evaluate only static HTML pages and ignore all the dynamic elements that are the main components of an AJAX application. Thus, presently all the accessibility evaluation tasks for AJAX applications need to be carried out manually resulting very time consuming.

In this work we describe an approach to provide a support to a tester for carrying out the accessibility evaluation task. In particular, we exploited the Crawljax based approach to overcome the dynamic nature of AJAX applications in order to make more effective the use of traditional accessibility evaluation tools. In the next section, we will recall the main aspects of Crawljax and the state-based testing approach that this tool supports, since it will be exploited in our approach that is illustrated in section 4.

3. Using Model-Based Testing for AJAX Applications: the Crawljax Approach

There are many examples of important applications that use AJAX technology such as Google Suggest, Google Groups, GMail, Google Maps, and Amazon. These commercial web sites demonstrate that AJAX is practical for real-world applications and more and more complex and sophisticated applications make use of AJAX. For all these reasons it is very important to find an efficient technique for testing those applications. Existing Web testing techniques are not appropriate for AJAX applications because there are different features of AJAX that make the test extremely difficult to realize. One of these characteristics depends on the fact that AJAX makes an intensive use of client-side scripting code to realize the rich event-based GUI. Another aspect is related to the use of a single-page approach,

where the navigation among pages used in the traditional applications is replaced by dynamic changes of the page structure. AJAX approach changes also the navigation structure building, since every element of the page can contribute being clickable at runtime. Another aspect is related to the asynchronous communication between client and server components based on raw data such as string or text instead of whole HTML page. Therefore understanding the evolution of AJAX page is very difficult observing the communication between client and server.

Model-based testing is turned out to be quite useful for test Ajax Web applications. Indeed, it exploits reverse engineering and Web crawling techniques to build a model of the target application and then extract test cases by traversing the model. A Web crawler (or Web spider or robot) is a program that automatically traverses the Web's hyperlink structure and retrieves the content of the Web pages. It builds a graph (usually called navigation model) where each node represents a Web page and each edge represents a link. This approach is not completely applicable to test AJAX applications because resulting navigation model may be wrong with high probability, due to the single-page nature of AJAX applications. In order to apply this approach to AJAX, in [9] it is proposed a state-based testing approach based on traces of the application to construct a finite state machine. The constructed finite state machine differs from the navigation model since each node represents a different state of an AJAX page and each edge between vertices represents a clickable element that allows reaching the target vertex from the start vertex. Building finite state machines is not a simple task; there are different challenges. First of all, it is difficult to identify the elements that form the navigation structure because, as said before, each element of the page can became clickable at runtime. Another important challenge is related to understanding when a change of state in the page occurs, and when two states are identical. AJAX technology does not allow a correct use of the "back" and "forward" features of the browser due to the fact that the dynamic changes of DOM are not registered in the browser history, thus making more difficult the navigation within the state machine. In [10] a tool, named Crawljax, is proposed to navigate an AJAX application and incrementally infers a finite state machine. Initially, the state machine only contains the root state and new states are created and added as the application is crawled and state changes are analyzed. In order to obtain all the clickable elements in a page, Crawljax exploits an algorithm that uses a set of candidate elements which are all exposed to an event type. The creation of the states is doing when the comparison between the actual DOM and the DOM obtained after firing an event on a clickable candidate elements, returns a significant difference. When a new state is created, a new edge on the graph is also created between the state before the event and the current state. For each DOM state a hash code is also computed and used to compare every new state to the list of already visited states in order to recognize an already met state. Once a clickable element has been identified and its corresponding state was created, the crawl procedure is recursively called to find new possible states. Terminated the state machine generation, Crawljax also uses it to generate indexed pages that represent static instance of a dynamic page. To do

this, Crawljax follows the outgoing edges of each state in the state machine and transforms in hypertext link each clickable element, updating also the HREF attribute to link to the generated static page. After the linking process, each state of the state machine is transformed into the corresponding HTML string representation and saved on the file system. Each generated static file represents the content of the AJAX application as seen in the browser in a specific state at the time of crawling. Crawljax gives the tester the opportunity to specify the depth level of the state machine and the maximum number of states. Moreover it is possible to manually specify the elements that should be clicked and the input value for the form field.

Once the state machine and its static representation are available, it is possible to make several types of tests. In particular, in [12] it was proposed a way to do regression testing for AJAX applications, while in [11] it was proposed a method for automatic testing AJAX applications through invariants specifications and several kinds of invariants can be used for this scope. As an example, invariants can be defined to automatically detect HTTP error messages (e.g., “404 Not Found”, “400 Bad Request”). In [3] it was proposed an approach to automatically detect security problems in Web widgets interactions, such as malicious widget which changes the content other widgets.

4. A Crawljax Plugin for Generating Accessibility Evaluation Reports

As suggest by W3C guidelines a preliminary review of Web site accessibility combines some manual checking along with the use of several semi-automatic accessibility evaluation tools. Presently the use of these tools is not effective for AJAX applications or it requires a lot of manual work due to the high number of states that a single page can have. To address this problem, the plugin we realized exploits Crawljax to automatically infer a state-graph of AJAX applications, thus for each identified state it sends an HTTP request to a validation tool to evaluate the corresponding static page. The response (i.e., the accessibility report) is parsed and the information is recorded in HTML report file. The final report (see Fig. 1) resumes the number of errors and warnings (for each priority level) found in each state with reference to WCAG 1.0 [15]. In this way, all the steps required to evaluate a single Web page are automated and all the states of an AJAX application identified by Crawljax can be automatically evaluated.

index

Show details

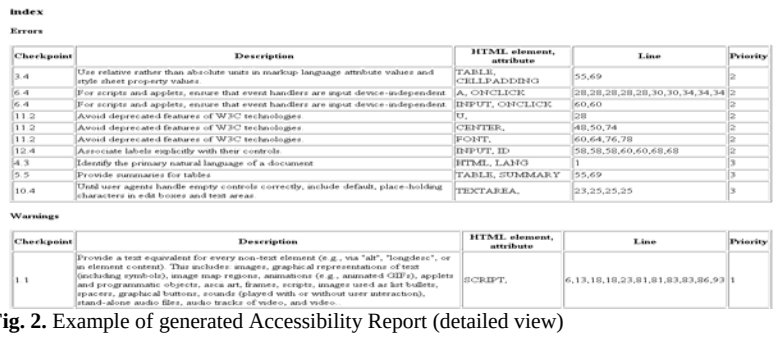
	Priority 1	Priority 2	Priority 3
Errors	0	30	7
Warnings	77	84	75

state2

Show details

	Priority 1	Priority 2	Priority 3
Errors	0	95	26
Warnings	91	149	381

Fig. 1. Example of Generated Accessibility Report Resume



index

Errors

Checkpoint	Description	HTML element, attribute	Line	Priority
3.4	Use relative rather than absolute units in markup language attribute values and style sheet property values	TABLE, CELLPADDING	55,69	2
6.4	For scripts and applets, ensure that event handlers are input device-independent	A, ONCLICK	28,28,28,28,28,30,30,34,34,34	2
6.4	For scripts and applets, ensure that event handlers are input device-independent	INPUT, ONCLICK	60,60	2
11.2	Avoid deprecated features of W3C technologies	U,	28	2
11.2	Avoid deprecated features of W3C technologies	CENTER,	48,50,74	2
11.2	Avoid deprecated features of W3C technologies	FONT,	60,64,76,78	2
12.4	Associate labels explicitly with their controls	INPUT, ID	58,58,58,60,60,68,68	2
4.3	Identify the primary natural language of a document	HTML, LANG	1	3
5.5	Provide summaries for tables	TABLE, SUMMARY	55,69	3
10.4	Until user agents handle empty controls correctly, include default, place-holding characters in edit boxes and text areas	TEXTAREA,	23,25,25,25	3

Warnings

Checkpoint	Description	HTML element, attribute	Line	Priority
1.1	Provide a text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content). This includes: images, graphical representations of text (including symbols), image map regions, animations (e.g., animated GIFs), applets and programmatic objects, ascii art, frames, scripts, images used as list bullets, spacers, graphical buttons, sounds (played with or without user interaction), stand-alone audio files, audio tracks of video, and video.	SCRIPT,	6,13,18,18,23,81,81,83,83,86,93	1

Fig. 2. Example of generated Accessibility Report (detailed view)

Several validation tools [5] can be employed to make a semi-automatic evaluation of Web sites accessibility. Generally, these tools follow different accessibility guidelines (such as WCAG 1.0 [15] and Section 508 [13]) and generate a report that highlights accessibility issues found in a Web page. In the current implementation of the plugin we exploited EvalAcces [4] as accessibility evaluation tool. This tool is based on WCAG 1.0 [15] guidelines and allows the tester to evaluate either single Web pages or an entire Web site. Exploiting this feature our plug-in is able to provide a detailed description (see Fig. 2) of the errors and warnings found in each state, including number of violated guideline and related checkpoint, short checkpoint description, names of the attributes that are missed or causes the error/warning, lines of code where the error/warning was detected, priority level of error/warning. We selected EvalAccess for the above mentioned features and for the fact that it turned out quite stable. It is obvious that the proposed approach can be easily extended by taking into account other accessibility evaluation tools, thus benefiting of the fact that different tools are able to capture different accessibility problems.

4. Case Study Planning

4.1 Planning

We experimented the proposed plugin on two AJAX-based Web applications, namely Google Search [6] and AskAlexia [2]. The former refers to the home page of the most popular Web search engine; through this page it is possible to search information in the Web and to reach several Google applications such as Gmail, Google Maps and Google Calendar. AskAlexia is a beta search engine entirely realized in AJAX that allows searching various contents including Web pages, images, videos, music, news, and blogs. We selected these applications because they can be considered a representative set of the real Web applications that use the AJAX technology. The goals of our case study were:

- **R1 (Effectiveness)**: to assess the effectiveness of the plugin in evaluating the accessibility issues of AJAX Web applications.
- **R2 (Performance)**: to analyse plugin performance in terms of input size versus time.

To address R1, we verified whether the plugin checked the accessibility for all the states of the application which satisfy the characteristics specified during the set-up (e.g., tag name, depth, …). Moreover, we verified whether or not the approach makes more effective the use of traditional accessibility validation tools. In particular, we assessed if the proposed combination of Crawljax and EvalAccess was able to provide more accurate evaluations with respect to the traditional use of EvalAccess. Thus, we compared the reports with the ones obtained by employing the “evaluate Web site” functionality of EvalAccess.

As for R2, we analyzed the performance of our approach in terms of time versus input size for each crawled state, where the input size is represented by the HTML code size. We also measured the time required by a tester to manually accomplish the tasks that the plugin automates. In particular, we estimated the time needed to manually use EvalAccess. This required to manually search each clickable page element, click on it and save the relative Web page, then copy the html code and paste it in the EvalAccess Web page.

The experiments were carried out using a laptop with Intel Pentium M processor 1.73GHz, with 2 GB RAM and Windows XP with Service Pack 3. Concerning the plugin configuration we employed (i) the default input specification for the Crawljax configuration process, (ii) the “clickDefaultElements” method provided by Crawljax for specifying the clickable elements, (iii) a depth level equals to 1 in order to avoid the access to other Google applications or external Web sites starting from Google and AskAlexia home pages.

4.2 Results

4.1.1 R1 – Effectiveness

The plugin execution generated a graph with 11 states on Google application and a graph with 6 states on AskAlexia. In both cases Crawljax did not produce two different crawled states for the same page state. However, the manual search for clickable page elements revealed that on the experimental object there were 21 clickable elements (i.e., states). Thus in this case Crawljax missed 10 states and the plugin could not produce the relative accessibility reports. Table 1 reports the overall number of accessibility errors and warning found for each guideline of WCAG [15] for Google and AskAlexia.

Table 1. Numbers of Error (E) and Warnings (W) reported by the plugin for Google and AskAlexia Accessibility Evaluation using WCAG 1.0 guidelines (GL).

		GL 1		**GL 2**		**GL 3**		**GL 4**		**GL 5**		**GL 6**		**GL 7**		**GL 8**		**GL 9**		**GL 10**		**GL 11**		**GL 12**		**GL 13**		**GL 14**	
	State	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**	**E**	**W**
Google	Index	-	12	-	23	2	11	1	2	2	8	13	13	-	24	-	11	-	-	4	64	8	3	7	1	-	61	-	3
	State2	-	22	-	79	26	50	1	2	13	52	58	9	-	30	-	7	-	-	-	295	20	3	50	1	-	61	-	3
	State3	-	5	-	21	34	57	1	2	25	100		5	-	9	--	3	-	-	-	393	48	3	13	1	-	19	-	3
	State4	179	185	-	251	18	402	1	2	12	48	120	7	-	191	-	5	-	-	1	739	207	3	61	1	-	777	-	3
	State5	2	7	-	8	23	15	1	2	13	52		6	-	12	-	4	-	-	-	27		3		1	-	27	-	3
	State6	14	58	-	59	34	59	1	2	18	72	27	6	-	63	-	4	-	-	-	54		3		1	-	71	-	3
	State7	6	6	-	12		22	1	2	-	-	19	6	-	11	-	4	-	-	-	36		3	5	1	-	51	-	3
	State8	-	12	-	22	2	11	1	2	2	8	16	13	-	24	-	11	-	-	4	66	8	3	9	1	-	69	-	3
	State9	-	15	-	15		26	1	2	-	-	-	12	-	25	-	10	-	1	-	72		3		1	-	133	-	3
	State10	-	9	-	22	1	25	1	2	5	20	37	9	-	17	-	7	-	-	3	128	1	3	9	1	-	187	-	3
	State11	1	26	-	31		96	1	2	7	24	-	15	-	40	-	13	-	-	-	433		3	1	1	-	819	-	3
AskAlexia	Index	-	20	-	26	2	16	1	2	1	4	1	9	-	28	-	7	-	-	24	63	1	3	1	1	-	85	-	3
	State2	-	20	-	26	2	16	1	2	1	4	1	9	-	28	-	7	-	-	24	113	1	3	1	1	-	89	-	3
	State3	-	8	-	14	2	4	1	2	1	4	1	9	-	16	-	7	-	-	-	41	1	3	1	1	-	41	-	3
	State4	-	8	-	14	2	4	1	2	1	4	1	9	-	16	-	7	-	-	-	113	1	3	1	1	-	45	-	3
	State5	1	9	-	15	2	5	1	2	1	4	1	9	-	17	-	7	-	-	-	60	1	3	1	1	-	49	-	3
	State6	1	9	-	15	2	5	1	2	1	4	1	9	-	17	-	7	-	-	-	60	1	3	1	1	-	49	-	3

The comparison between the report obtained with our plugin with those obtained employing only EvalAcces with the “evaluate Web site” functionality revealed

that the number of errors and warnings highlighted by the plugin is higher than those revealed by EvalAccess alone (see Table 2). This is due to the fact that in this case only a single state is found and then analyzed by EvalAccess for both the Web applications, against the 12 and 6 states analyzed by our plugin for Google and AskAlexia, respectively. Thus, we can argue that the combination of a traditional evaluation tool and Crawljax let to discover more problems than those found by applying only the traditional validation tool.

Table 2. Accessibility report resume obtained employing EvalAcces and the proposed plugin for Google and AskAlexia Web Applications

		Priority 1		**Priority 2**		**Priority 3**	
		Eval Access	**Plugin**	**Eval Access**	**Plugin**	**Eval Access**	**Plugin**
Google	**Errors**	0	0	23	30	4	7
	Warnings	50	77	64	84	60	75
AskAlexia	**Errors**	0	0	5	29	2	2
	Warnings	35	86	38	83	41	98

4.1.2 R2 – Performance

In Table 3 we reported the performance of the plugin in terms of HTML code size versus time to process it. We can note that the processing time required by the plugin to generate accessibility reports for each state of both Google and AskAlexia Web sites is proportional to the size of the HTML code of the state. However, the average time[3] required by the plugin for processing 1 kb is 0.1 seconds in case of Google, while is slightly higher for AskAlexia (i.e., 0.2 seconds for 1 kb). Since this time is the approximate time required to send the http request to EvalAccess server, to analyze the input code and to receive the http response, this result may depend on the fact that the HTML code size of both Web sites is very similar. We also measured the time spent to manually evaluate the crawled states for both Web sites. This task requires that a tester has to manually discover states and for each of them save the relative Web page, copy the html code, and paste it in the EvalAccess Webpage. This operation requires about a minute for each state. Thus, for the experimented objects the overall time spent by a tester is about 11 and 6 minutes for Google and AskAlexia application, respectively. It is worth to note that these values are higher than the ones needed to accomplish the task with the plugin (i.e., 37.6 seconds for Google and 10.5 for AskAlexia). As we can image the more the states the more burdensome is to manually do the task.

[3] This value is obtained dividing the sum of all processing time with the sum of all HTML code size.

Table 3. Performance time of the plugin for Google and AskAlexia Web Applications

	State Name	Input Size (in kb)	Processing Time (in seconds)	Processing Time for 1 kb (in seconds)
Google	Index	13	1.8	0.1
	State2	27	2.5	0.1
	State3	40	2.6	0.1
	State4	90	9.3	0.1
	State5	17	1.7	0.1
	State6	8	2.3	0.3
	State7	9	1.4	0.2
	State8	12	1.8	0.2
	State9	8	1.7	0.2
	State10	31	2.9	0.1
	State11	104	9.6	0.1
AskAlexia	Index	10	1.8	0.2
	State2	9.8	1.8	0.2
	State3	8.4	1.6	0.2
	State4	12.4	1.7	0.1
	State5	12.4	1.8	0.1
	State6	12.4	1.8	0.1

5. Conclusions and Future Work

In this paper, we described a Crawljax based approach to automatically evaluate accessibility of AJAX applications. The proposed approach has been evaluated through a case study that, though preliminary, it is enough to suggest us the viability of the approach to automate the accessibility evaluation of AJAX applications. Indeed, the approach is able to increase the effectiveness of the use of traditional accessibility evaluation tools. However, there are several points to be improved. Indeed, the plugin inherits not only the advantages of Crawljax but also its shortcomings. The main problem is concerned with the fact that Crawljax did not detect all the reachable states, thus affecting the completeness of the accessibility evaluation plugin. Reinforcing these aspects of Crawljax will determine a great improvement for the obtained results making the plugin more accurate and useful. Moreover, the use of different evaluation tools can be a possible future work. Indeed, different accessibility evaluation tools could identify different accessibility issues, such as the ones developed for validating the WCAG 2.0 [16]. Furthermore, several future researches can be prospected in the line suggested by the paper. In particular, using the Crawljax approach tools can be realized to automate other accessibility evaluations tasks. As an example, an important accessibility issue that is still manually checked is the keyboard accessibility. This holds if every action triggered with a mouse is also available through the keyboard. Keyboard accessibility is fundamental for people who do not use pointing devices and for the use of many assistive technologies including screen readers and on-screen

keyboards. The proposed Crawljax state-based approach could be used to automate the verification that all the page elements of an AJAX application can be accessed by keyboard. The proposed approach could be also useful to compare the content between a standard version of the AJAX Web application and its accessible version. Indeed, an accessible version of a Web application is generally realized to improve the accessibility level of the application when the technologies used in the standard version does not allow reaching a high accessibility level. So, a current problem is that the two versions require a maintenance work to align the content when one version changes. Manually validating content consistence of both versions may be a very tedious work. A possible solution could consist in the development of a plugin that automatically compare the content exploiting the graphs generated by crawling both versions.

Commento [FS1]: Probabilmente possiamo toglierlo x rientrare nelle 8 pagine….

References

[1] Accessible Rich Internet Applications 1.0, http://www.w3.org/TR/wai-aria/
[2] AskAlexia Web application at http://www.askalexia.com.
[3] Bezemer, C.P., Mesbah, Van Deursen: A. (2009). Automated security testing of web widget interactions. ESEC/SIGSOFT FSE, 81-90.
[4] EvalAccess, http://sipt07.si.ehu.es/evalaccess2/
[5] Evaluation tools,http://www.w3.org/WAI/RC/tools/complete
[6] Google Search Web application at http://www.google.com.
[7] Kluge, J., Kargl, F., Weber, M., (2007). The effects of the AJAX Technology on Web application usability. Int. Conf. on Web Information Systems and Technologies.
[8] Marchetto, A., Ricca, F, Tonella, P., (2008). A case study-based comparison of Web testing techniques applied to AJAX Web applications. International Journal on Software Tools for Technology Transfer, (10)(6), 477-492.
[9] Marchetto, A., Tonella, P., and Ricca, F., (2008). State-based testing of Ajax web applications. In Proc. of 1st Int. Conf. on Software Testing Verification and Validation, 121–130.
[10] Mesbah A., Bozdag E., and van Deursen A., (2008). Crawling Ajax by inferring user interface state changes. In Proc. of the 8th Int. Conf. on Web Engineering, 122–134..
[11] Mesbah A. , and van Deursen A., (2009). Invariant-based automatic testing of Ajax user interfaces. In Proc. of the 31st Int. Conf. on Software Engineering, 210–220.
[12] Roest D., Mesbah A., van Deursen A., (2010). Regression Testing Ajax Applications: Coping with Dynamism. In Proc. of the 3rd Int.Conf. on Software Testing, Verification and Validation.
[13] Section 508, http://www.section508.gov/
[14] WebDriver library by Selenium, http://code.google.com/p/Web-accessibility-testing/
[15] Web Content Accessibility Guidelines 1.0, http://www.w3.org/TR/WCAG10/
[16] Web Content Accessibility Guidelines 2.0, http://www.w3.org/TR/WCAG20/