Automatic Accessibility Evaluation of Rich Internet Apps

Vivian Genaro Motti, Jean Vanderdonckt
Université catholique de Louvain, Belgium
{Vivian.GenaroMotti, Jean.Vanderdonckt}@uclouvain.be

ABSTRACT
The Web 2.0, with its Rich Internet Applications (RIAs), exploits dynamic interactivity to provide a rich interactive environment for web users. On important common denominator is the support to user collaboration via the highly dynamic "desktop like" generated content. Such a context makes it difficult for the designer to tackle a diversity of users, in particular in terms of accessibility issues. WAI-ARIA is a specification that focuses on accessibility issues when developing RIAs. This paper describes an algorithm to automatically evaluate *EcmaScript* code commonly deployed in RIAs, and presents a case study applied to pop-up windows.

Categories and Subject Descriptors
H.5 [**Information Interfaces and Presentation**]: Miscellaneous
H.5.4 [**Hypertext/Hypermedia**]: User Issues

General Terms
Design, Human Factors, Standardization, Languages, Legal Aspects.

Keywords
Web 2.0, accessibility, technologies, Rich Internet Applications, RIA, Automatic Evaluation.

1.INTRODUCTION
The dynamics in creating and retrieving large scale information has brought new requirements towards technologies available on the web. This capability leads users into contributing with content generation. Applications such as MySpace (www.myspace.com), Wikipedia (www.wikipedia.org), Amazon (www.amazon.com) and Flickr (www.flickr.com) are examples of tools that allow user remote asynchronous collaboration.

The term "Web 2.0" implies using evolved patterns into applications development [1]. Its characteristics include: users as part of the authoring process, greater interactivity than before and a desktop like experience [2]. It is important to notice that these conditions were made available by the technologies; however users experience and interactivity are its focus.

The development of Web 2.0 requires specific kinds of applications called RIAs (Rich Internet Applications). RIAs are built using technologies such as HTML, XHTML, CSS, Microformats, EcmaScript, AJAX, DHTML and FLASH. These technologies are used in order to provide greater interactivity and dynamics in displaying information – when compared with the earlier versions of Web applications.

Tim Berners Lee stated "The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect" [20]. Accessibility emphasizes that content must be developed considering different user profiles and technologies in order to present accessible information not only for specific groups of people.

In order to evaluate the accessibility level of Web Applications, there are different approaches which include: accessibility guidelines, users' evaluation and experts' evaluation. These approaches are based on formal defined specifications, such as the WCAG 1.0, a suite of guidelines to accessibility evaluation [8].

The Web 2.0, as an emergent phenomenon, does not have formal accessibility specification. Not only its nature of fast evolution, but also its characteristics of dynamic interaction, makes it difficult for the development of accessibility patterns. Difficulties raised by new technological improvements can restrict its use for the people who need it the most [2].

This paper introduces the technologies that are commonly used to build RIAs and its concerns with accessibility (Section 2). It also accessibility issues (Section 3). Section 4 describes limitations found when considering accessibility and RIAs. Section 5 presents WAI-ARIA, a specification for considering accessibility in Web 2.0 applications. Section 6 presents related works in literature. Section 7 describes an approach of automatic accessibility evaluation using a parser algorithm for EcmaScript source codes. Section 8 discuss and presents conclusions. And Section 9 presents possibilities of Future Works.

2.TECHNOLOGIES
Although HTML is the central document language of the web, given its static nature it is not sufficient to provide the interactivity demanded by RIA applications. Other technologies have been designed to accomplish these requirements [3].

RIA applications use high programmable script languages, advanced protocols and technologies to improve interaction of client browsers with database servers. They also need dynamic content manipulators, services to provide interfaces to data and rich interfaces (with interaction improvements).

As Web applications use these technologies, new kinds of them were created. An important example it the approach of MASHUP applications, which use two or more Web Services to provide a new functionality or service. The Gmail Webmail manager, for instance, has chat, search and tagging functionalities making use of services provided by other applications.

RSS (Really Simple Syndication) is an XML (Extensible Markup Language) format to indicate content provided by other applications [3]. RSS feeds are a simple way of websites noticing new contents, services or updates. Instead of periodically accessing pages, users receive from the site an update note on the RSS file.

The applications described above were developed in the Web 2.0 context, and make use of technologies such as CSS, EcmaScript, XML, DHTML, HTML-DOM and AJAX to provide interactivity and rich user experience.

Rich Internet Applications (RIAs) are the concept behind the de-

velopment of interactive Web applications. These applications implement local processing functionalities, like responsiveness towards events, control over the layout, changes in the DOM structure of the HTML, among others. Servers interact with the local application providing data in text format, whenever an application requests it.

Figure 1 shows the three main parts that characterize RIAs. The characteristics of client-server processing of Web applications allied to responsive components of Desktop applications, and the transparency of asynchronous communication between the parts of the applications (Communications technologies), form the basis of RIA. The overall approach can be explained as the use of communication technologies such as Ajax or HTTPRequest, between client and server side, to provide users a web application with a interaction experience similar to the one provided by traditional Desktop systems.

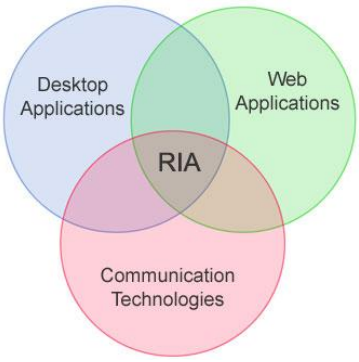

Figure 1. RIA diagram showing its three most important characteristics. Source: http://hendese.net/en/userfiles/images/ria-venn-diagram.jpg

The development of RIA applications relies on technologies such as HTML, for the displaying of the information; CSS, to describe the layout of the pages; EcmaScript and DHTML, to control the elements and events of the page; AJAX, that allows asynchronous communications of the client side application with the server side; among others. The remaining of this section presents the main technologies involved in the RIAs building.

2.1 HTML

HyperText Markup Language (HTML) was created to provide semantic context to text documents published on the Web, making the shared information more flexible. HTML files are text files containing the document elements. An element is an information unit of one type denoted by its tag. An element can be a single tag, or text and other elements delineated by tags [7].

HTML is unquestionably the main document language of the Web, but it is by no means the only language. The appearance of an HTML page can vary between browsers or be affected by the size of the window displaying the page. This behavior is not acceptable in commercial settings where maintaining an attractive presentation or adhering to corporate appearance standards has real importance.

The semantics of HTML's elements do not map well many common document types. As a result, HTML often comes to be used more as a high-level page description language than as the structured document language it was envisioned to be. HTML is designed for non-interactive documents whose content does not change as the user interacts with them [3].

2.2 XML

XML was developed by World Wide Web Consortium (W3C) in 1996. XML documents are made of entities, storage units with data. Parsed data is made of characters, which form character data, or markup. Markup encodes a description of the document's storage layout and logical structure.

XML provides a mechanism to impose constraints on the storage layout and logical structure. It is a general-purpose specification for creating custom markup languages. It is classified as an extensible language because it allows its users to define their own languages using XML as the specification syntax. The primary purpose of XML is to facilitate the sharing of structured data across different information systems, particularly via web and it is used both to encode documents and to serialize data [5].

2.3 HTML DOM

Document Object Model (DOM) is an interface to allow programs and scripts to dynamically access and update the content structure and style of documents. The document can be further processed and the results can be incorporated back into the presented page [14].

The HTML DOM (Document Object Model) defines a standard way for accessing and manipulating HTML documents. The DOM presents an HTML document as a tree structure, with elements, attributes, and text as nodes.

2.4 CSS

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts, colors, and spacing) to Web documents [4]. CSS is a declarative language that can be used to specify the appearance of HTML elements with considerable precision. CSS was designed to give authors better control over the appearance of their HTML documents and to reduce the inconsistencies in how browsers render HTML pages.

All modern graphical Web browsers support CSS, though the level of support is not yet consistent. Most modern Web authoring tools make extensive use of CSS.

A CSS style sheet is composed of a series of rules. Each rule has two parts: a selector and a list of declarations. The selector specifies which HTML elements the rule will apply to and the declarations specify how those elements should look [3].

2.5 EcmaScript

EcmaScript code allows defining functions that are activated by events such as key presses, mouse clicks, and timers (as well as other code). The functions initiated by these events can, for instance, modify or refresh page content (with or without user's awareness), provide auto-complete options, create new windows including alert, confirmation and prompt pop-up windows, and change the keyboard focus while a form is being completed. EcmaScript can also be used to define pull-down menus, selection boxes and to automate user navigation on Web pages [3].

2.6 DHTML

Dynamic HTML can be described as the combination of HTML,

style sheets and scripts that allow documents to be animated [13]. DHTML supports the building of dynamic and interactive web sites. Some typical DHTML technologies are HTML, for static and markup language, CSS, for presentation configuration, and EcmaScript (described above) [3].

Simple DHTML pages are still basic documents, and map easily to HTML semantics. They may contain client-side form interaction and validation using in-built user interaction mechanisms like alerts, pop-ups, and server-trip page navigation. The DOM is not typically modified on the client-side, and there is no asynchronous content updating (AJAX). These pages are well supported by current user agents and assistive technology [6].

Moderate-complexity DHTML pages use common UI widgets like toolbars, menus, dialogs, and sliders. They may involve unusual variations on these controls. These applications frequently modify the Document Object Model (DOM) at runtime, to add content, change layout, display error messages, etc. Areas of the page may be updated asynchronously, and full-page navigation is kept to a minimum [6].

2.7 AJAX

Asynchronous JavaScript and XML (Ajax) is an approach that combines extensive use of typical DHTML technologies (HTML, CSS, EcmaScript) with XMLHttpRequest (XHR) to provide asynchronous client-server communication via text-based messages. As a result, Ajax allows the contents of a page to be automatically updated by the server without an explicit request from the user.

A typical use of Ajax is an email service that automatically suggests destination addresses as the user types some initial characters (auto-complete function). Although such automatic completion is more than common in traditional local GUI desktop applications, its use on the Web requires quick and transparent client-server communication so that the list of possible completions is dynamically updated as the user enters and deletes characters from the start of the address.

3.WEB ACCESSIBILITY

Accessibility intends to provide universal access regards of disabilities or old technologies. Accessibility is a very important issue in all types of computational systems. However, its importance becomes even more evident in the context of the Web. Making people involved in the development of Web applications aware of accessibility issues is very important to promoting an effective inclusive agenda [19]. The development of accessible web applications must consider standard guidelines, for instance WCAG 1.0 [8], WCAG 2.0 [9], MobileOk Basic Tests [10].

Although guidelines do not guarantee full accessibility, they are the main way to consider accessibility issues in the application development – along with the expertise of the developers.

These guidelines consider the use of correct elements and attributes markups (HTML). These markups, when well used, can be correctly interpreted by assistive technologies, providing content access to users with disabilities or different technologies. For example, when blind users access web content with a screen reader, unless images have an alternative text, they won't be able to access them. The alt attribute of *img* element provides an alternative text to the image displayed, as in:

```
<img src="bird.png" alt="bird flying">  </img>
```

Developers must consider accessibility issues from the beginning and during all the life cycle of the web application development. Moreover, in the context of the Web 2.0, users must be able to build accessible content once they are also content providers [11].

There are many ways of evaluating how accessible an application is. Guidelines are the base of the evaluation process once they guarantee that assistive technologies can read the source code of the application, even though semantic and coherence of content can't be assured.

The accessibility automatic evaluation tools parse the source code (in HTML) and compare it with standard guidelines providing a report with all results found (fails, correct characteristics and issues that need manual review).

4. LIMITATIONS

There are a set of limitations found in evaluation and repair accessibility tools. Four of them are more important in the context of this paper [11]:

1. Limited crawling since the evaluation cannot be limited to static HTML links, given that there is a wide variety of document formats and the intrinsic dynamics of RIAs which generating page content as response to user actions;
2. Lack of validation against grammars, like HTML, XML or CSS. It is critical for browsers running in mobile devices, which have lower processing capabilities, are less fault-tolerant and require valid documents to parse them;
3. Limited or non-existent localization support. It is an adaptation to different policy frameworks worldwide. This issue is particularly relevant within RIAs where users become content providers;
4. Inability to analyze or repair dynamically generated DOM (Document Object Model) elements. This issue is critical in RIAs as, for instance, JavaScript widgets modify the structure of the page in response to user actions.

These limitations provide a set of functional requirements for the development of the WAI-ARIA evaluator.

5.WAI-ARIA

Dynamic content in RIA applications cannot be evaluated by WCAG guidelines, since the specification of WCAG only comprehends static HTML code. RIA applications deal with lots of EcmaScript and Dynamic HTML to provide rich interfaces. Figure 02 illustrates that web applications composed mainly by HTML (XML, JavaScript and CSS can be included), with static content allow evaluation with WCAG.

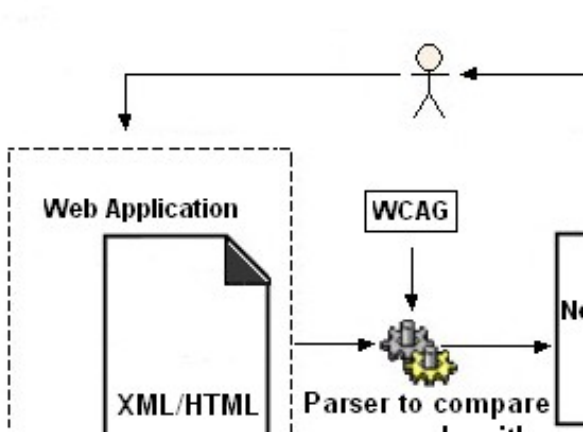

Figure 2. Static web applications use WCAG in their accessibility evaluation.

WAI-ARIA is an Accessible Rich Internet Applications Suite to make Web content and Web applications more accessible. Its application should help people with disabilities accessing dynamic content and interacting with advanced user interface controls developed with AJAX, HTML, EcmaScript and related technologies.

Assistive technologies allow users to access content in a more suitable way – according to their needs. However, in order to accomplish it, semantic information about presentation and layout must be added to the web site structure. If we consider that most dynamic contents are inserted into HTML *divs* and *span* elements and there is no semantics information about their displayed information.

Semantics are knowledge of roles, states and properties applied to elements within the content [12]. Roles are a way for an author to make custom widgets (new interactive elements) accessible, usable and interoperable [15]. The role taxonomy of the widgets is presented in the document in Web Ontology Language.

To implement WAI-ARIA specification in applications it is necessary to add attributes such as roles on the elements that make use of dynamic content. For example:

```
<div role="wairole:dialog" tabindex="-1" wairole="dialog">
        Information that are displayed on the dialog.
</div>
```

Note that the *div* element has a *role* and a *wairole* attribute that specifies semantics to the widget. The Dojo AJAX framework is one of the foundations that implement the WAI-ARIA specifications in its widgets.

The specification also denotes attributes that inform assistive technologies about regions in which updates may be received (live regions for dynamic updates) via AJAX requests.

WAI-ARIA specification is currently a W3C working draft.

6.RELATED WORK

Velasco et al developed a Web Compliance Engineering framework to support the development of accessible Rich Internet Applications. They defined Web Compliance Engineering as dealing with increasing complexity of web applications. And noticing the accessibility implementation difficulties found when users become content providers, they presented a Web compliance framework developed to support both users and developers to create accessible content for RIAs [11].

Gibson et al describe the utilization of semantic added information on HTML tags to denote graphical user interfaces that are mapped into accessibility platforms that will handle the presentation of the elements in an accessible way. It also shows the utilization of methods that can be used to give focus to any elements; these issues are particularly important considering the implementation of a fully keyboard accessible applications [14]. Gibson et al also describe some features of the WAI-ARIA specification such as the utilization of region attributes to denote the dynamic changes of the elements. Dojo is referenced as an example of implementation of WAI-ARIA specification in an AJAX framework [13].

Dojo is an Open Source DHTML toolkit written in JavaScript that allows adding dynamics into web pages and environments that support EcmaScript. The components that Dojo provides make web sites more usable, responsive, and functional. Dojo allows prototyping interactive widgets quickly, and animate transitions. The lower-level APIs and compatibility layers from Dojo can be used to write portable JavaScript and simplify complex scripts. Dojo's event system, I/O APIs, and generic language enhancement form the basis of a programming environment. Dojo can be used to build tools to write command-line unit-tests for JavaScript codes. The Dojo build process helps optimize JavaScript code for deployment by grouping sets of files together and reuse those groups through "profiles" [14].

Cooper says that accessibility represents a challenge to Web 2.0 and semantic web technologies can address some of its requirements. He acknowledges that Web 2.0 brings interactivity and information sharing as Semantic Web brings context information, which is beneficial to accessibility and to the robustness of the combined platform [2].

Thiesen implemented an accessible chat application using live regions to notify the browser of updates on the interface. The browser maps the live regions of the page and read then accordingly to the events that update the page (Firefox with the Fire Vox extension enabled). The navigation of the page was also considered during the implementation. It was provided for users scroll buttons handler events to filter the messages displayed and to navigate through them [18].

7.AN APPROACH FOR AUTOMATIC EVALUATION

The methodology of this study consists in presenting an algorithm for an evaluation tool that provides RIAs developers information about the accessibility state of their applications (considering the WAI-ARIA patterns). The algorithm itself is used as a proof of concept to the evaluator and is used in a case test described in this session.

As an entry the algorithm would receive a RIA code to be evaluated. The evaluation tool would look into every DOM structure element of the application HTML looking for elements that implement dynamic behavior (use of DHTML). These elements -that express dynamics- would be then classified as nodes that need WAI-ARIA attributes definitions. As a result the algorithm would warn the application developer about the elements that need WAI-ARIA attribute definitions so that he considers implementing the

specification. After discussing the usual processing, the algorithm and a case test for the evaluation tool.

7.1 Evaluation process

For the evaluation tool implementation, it is necessary to understand some characteristics of DHTML technologies that represent the most important part of the implementation of dynamic and interactive applications, the focus of WAI-ARIA specification.

As previously described DHTML makes use of CSS, EcmaScript and HTML. WAI-ARIA specification makes use of dynamic elements attributes that map its functional behavior on the application. Given the technologies that are part of Dynamic HTML the technologies that directly handle the dynamic manipulation of elements are EcmaScript, DOM HTML and HTML events.

Events are activated in each user interaction. These events execute EcmaScript functions that change the content of the application in order to provide responsiveness to the interfaces. The changes of content are made by EcmaScript controlling the elements of HTML through direct use of the DOM structure of documents. On the elements that are manipulated over DOM HTML there are no indications about if they will be modified by EcmaScripts or about the exactly widget being implemented by that element (no semantics specified on *DIV* and *SPAN* elements that are usually used as classification elements for CSS or the widgets implementation[13]). Considering that, every implementation of automatic accessibility evaluation of RIAs using the WAI-ARIA must analyze the DOM Structure manipulation over EcmaScript in order to find the nodes that present dynamics in their behavior.

Therefore it was developed a model to represent the necessary architecture to implement the evaluation tool. Figure 03 shows that RIA Web applications combine functionalities of different languages in order to provide dynamic and richer interactivity. The WCAG model of HTML evaluation of the applications does not suffice RIA accessibility needs, since HTML elements may be created or updated during the execution of the page, therefore it is necessary an EcmaScript code evaluation, more specifically it is necessary a DOM HTML manipulation analyses to validate the WAI-ARIA conformance of the application.

Notice that DOM HTML manipulations have an extensive list of possibilities of implementations (find by ID, find by Tag name, navigate through the DOM structure, nodes creation, nodes removal) and these possibilities make it difficult to find exactly the elements that present the dynamic behavior in the HTML structure and that element might even not be in the structure yet (created and not inserted yet), however the line of the JavaScript code that makes use of the DOM structure is relatively easy to find and is used as the output of the algorithm.

HTML events does not need to be considered in all automatic evaluation algorithms since the evaluation of every EcmaScript use in the application guarantees that the functions that are executed by the events would have already been covered. However the evaluation of HTML events that execute EcmaScript functions determines the efficiency of the algorithm once not every function implemented in EcmaScript is actually executed or specifies dynamics to the application.

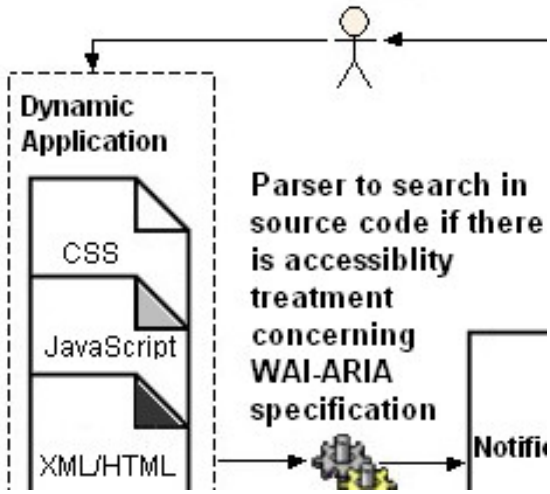

Figure 3. Diagram modeling WAI-ARIA evaluation method.

It is also important to understand that there is no current specification which describes how to implement a specific widget and the nature of widgets itself is extensible. This fact makes it difficult to map the DOM manipulations with its correct attributes of role and state. This justifies the fact that the algorithm described here does not infers which role or state attribute should be used in each element, it only notifies the developer about where ARIA attributes should be placed.

7.2 Algorithm

The algorithm described here is a way of evaluating ARIA in applications. Basically the evaluator would scan the HTML code looking for every EcmaScript declaration of code or external libraries call. These declarations would be used to find the lines of JavaScript code that makes use of DOM manipulation of HTML. For each one of the DOM uses found the user would be alerted that there is a dynamic element that needs WAI-ARIA attributes definition.

Given the characteristics of the evaluation tool, the algorithm does not evaluate whether the application dynamic elements already have WAI-ARIA attributes. The implementation of that functionality would require a semantic analyses of the variables that contains DOM elements values, once JavaScript library allows many ways of accessing the same elements (by Id, by tag name, navigating through the DOM structure) and with that possibilities there could be more than one variable keeping reference of the same element.

HTML events are also not considered in the algorithm, since their implementation is not required for the evaluators. To implement the filter over HTML events it would only be necessary the analyses of the events that have handler attached on them and evaluate only the JavaScript functions that are used by these handlers. The handlers, just as the functions that are used by them, may be defined in any part of the HTML code or even in the JavaScript code, and can have its values attributed to JavaScript variables, given the characteristic of weak typing of the language. Therefore the events evaluation in the evaluation tool is not a simple task to be implemented. Even then it is possible to extract necessary

standards steps to evaluate application codes in this context. A high level description of the algorithm is presented in Figure 4:

1. Look into the HTML code of the application for every JavaScript library definition.
2. For each one of these libraries look for DOM HTML manipulations on the code.
3. For each one of these DOM manipulations (creation of elements, nodes removes, navigation between the nodes, seek by elements ID or TAGNAME) alert the developer of the line of the JavaScript code that presents the element been manipulated dynamically and alert the needs of a WAI-ARIA attribute definition for the element been manipulated.

Figure 4. Algorithm for automatic evaluation.

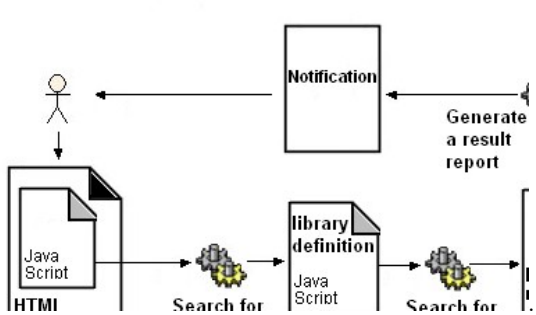

Figure 5. Algorithm for automatic evaluation.

Figure 5 illustrates that using the approach suggested the developer sends his HTML source code from RIA and an engine search first for library definitions in JavaScript. When it is found then the engine search for DOM manipulation, if there is a role defined the developer does not need to worry, but if there is no WAI-role definition this accessibility treatment must be inserted.

7.3 Case Test

A case test is presented using a pop-up window as a widget example. The idea is to apply the algorithm on the application described next and to point the functionalities of the algorithm. The application in Figure 5 implements a simple pop-up window that appears when the mouse button is clicked, and disappears if the button is clicked again.

As stated before the evaluation tool would focus on the application JavaScript code. Therefore the JavaScript code of the example of the case test is presented in Figure 6.

Considering that there is only one JavaScript library contained in the application, the algorithm would look for the commands that execute DOM HTML manipulation. The lines that contain *document.getElementById*, *document.createElement*, *document.body.appendChild* and *document.body.removeChild* would be the ones pointed by the evaluation tool as parts of the JavaScript code that determines dynamics to the page. The result can be interpreted as: these lines make use of HTML elements that need WAI-ARIA attributes.

```
window.onload = function(){
document.getElementById("pushButton").onclick = function(ev){
  dialog = document.getElementById("dialog");
  if(dialog == null){
    dialog = document.createElement("div");
    dialog.id = "dialog";
    dialog.style.display = "block;";
    dialog.className = "dialog";
    dialog.style.position = "absolute";
    dialog.style.left =  ev["pageX"] + "px";
    dialog.style.top = ev["pageY"] + "px";
    dialog.innerHTML = "Popup displayed after button pushed...";
    document.body.appendChild(dialog);
  }else{
    document.body.removeChild(dialog);
  }
  ev.stopPropagation();
};}
```

Figure 6. JavaScript code for pop-up window (example 01).

It is important to notice that there were multiple warnings towards the same dynamic element (dialog) when the actual modification on the DOM structure of the document only happened on the line where the node dialog is appended and removed. These multiple warnings are yet unavoidable considering the possibilities of implementing the same functionality in JavaScript. The other places indicated by the evaluator could be the places of actual DOM modifications in different JavaScript implementation of the same pop-up functionality.

In the case described, the dialog is created each time the button is clicked and removed when the dialog is appearing and the button is clicked. However it could be implemented as a *DIV* element with display equals to none and present in the HTML *BODY*, and the function that handles the event click would simply change the value of the attribute of display of the *DIV* into *block* and set the coordinates of the position to the mouse current position and to when the button was again clicked (clicked when the dialog was appearing) the function would just set the display of the dialog back to *none* as described in Figure 07.

In this other case the dynamic element dialog would only receive warnings in the evaluator on the line where the node is got by its ID even though it is the same functionality of the previous case.

There are no indications of which implementation (the first described or the second) is being implemented in the JavaScript code. And more possibilities for the same functionality could be available for implementation. Therefore all possibilities must be considered by the evaluation tool.

```
window.onload = function(){
document.getElementById("pushButton").onclick = function(ev){
  dialog = document.getElementById("dialog");
  if(dialog.style.display == “none”){
    dialog.style.display = "block";
    dialog.style.position = "absolute";
    dialog.style.left =  ev["pageX"] + "px";
```

```
    dialog.style.top = ev["pageY"] + "px";
  }else{
    dialog.style.display = “none”;
  }
  ev.stopPropagation();
};}
```

Figure 7. JavaScript code for pop-up window (example 02).

8.RESULTS AND DISCUSSION

There are few works in literature exploring automatic accessibility evaluation of Web 2.0 application. WAI-ARIA is the most important initiative concerning accessibility specification in this context is still a working draft, which probably means that lots of work still has to be done concerning accessibility issues in RIAs.

We propose a parser algorithm elaborated to evaluate codes and notify developers about the need of WAI-ARIA attributes in the nodes that implement widgets functionalities. In other words, the evaluation tool can be used to show the potential accessibility problems of RIAs.

Even though WAI-ARIA specification focuses on the HTML elements semantics, the evaluation tool, differently, focuses on the EcmaScript implementation of dynamic behavior. The characteristics of the language and its extent possibilities generate complex issues to any proposal of evaluation tools.

Many limitations still remain, considering the sort of possible combinations to be used in dynamic and Rich Internet Applications (like lack of patterns in the widgets implementation in JavaScript). This mean that new studies must be conducted to allow universal access for Internet users regardless of disabilities and technologies.

9.FUTURE WORK

More effort should be considered in the study of ways to implement widgets. If patterns could be observed in the implementation of each widget it would be possible to infer the WAI-ARIA attributes that should be given to each dynamic element in the EcmaScript.

Better analyses of the DOM manipulation functions in EcmaScript could allow more information to the developer about the attributes that should be used in the elements that were manipulated.

The evaluation tool limitations described in the algorithm session could be considered in a next version implementation. It also can be considered the semantic compiler described to implement the verification of the WAI-ARIA attributes presence in the elements that express dynamics.

The WAI-ARIA specifies ways of implementing accessibility in a DOM structured HTML that can be modified during the execution of the application. Therefore an element analysis during the application execution can be considered to evaluate the WAI-ARIA attributes presence in nodes that are modified. Considering that studies of evaluation tools implementation in the JavaScript interpreter of the browsers can be considered as an alternative to the development done so far.

Further work need to be done in order to implement a metric of evaluation for the WAI-ARIA specification. There are in the literature metrics to evaluate and compare the accessibility level in different web applications, these metrics can be extended so that they could be used in the ARIA context.

10.ACKNOWLEDGEMENTS

Our thanks to CNPq, FAPESP, CAPES and FINEP for financial support.

11.REFERENCES

1. Beirekdar, A., Keita, M., Noirhomme, M., Randolet, F., Vanderdonckt, J., Mariage, C., Flexible Reporting for Automated Usability and Accessibility Evaluation of Web Sites, Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT’2005 (Rome, 12-16 September 2005). Lecture Notes in Computer Science, Vol. 3585, Springer-Verlag, Berlin, 2005, pp. 281-294.
2. Beirekdar, A., Vanderdonckt, J., Noirhomme-Fraiture, M., KWARESMI - Knowledge-based Web Automated Evaluation Tool with Reconfigurable Guidelines Optimization, Proc. of 2nd Int. Conf. on Universal Access in Human-Computer Interaction UAHCI'2003 (Creete, 22-27 June 2003), Vol. 4, C. Stephanidis (ed.), Lawrence Erlbaum Associates, Mahwah, 2003, pp. 1504-1508.
3. Bouvier, D. J. 1995. The state of HTML. *SIGICE Bull.* 21, 2 (Oct. 1995), 8-13. DOI= http://doi.acm.org/10.1145/220230.220236
4. Bray, Tim; Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau (September 2006). Extensible Markup Language (XML) 1.0 (Fourth Edition) - Origin and Goals. World Wide Web Consortium. Retrieved on October 29, 2006.
5. Carlos A. Velasco, Dimitar Denev, Dirk Stegemann, Yehya Mohamad: A web compliance engineering framework to support the development of accessible rich internet applications. W4A 2008: 45-49
6. Cooper, M. 2007. Accessibility of emerging rich web technologies: web 2.0 and the semantic web. In Proceedings of the 2007 international Cross-Disciplinary Conference on Web Accessibility (W4a) (Banff, Canada, May 07 - 08, 2007). W4A '07, vol. 225. ACM, New York, NY, 93-98. DOI= http://doi.acm.org/10.1145/1243441.1243463.
7. Dojo Foundation. Dojo, the JavaScript toolkit. http://dojotoolkit.org, 2007.
8. FREIRE, André Pimenta ; RUSSO, C. M. ; FORTES, R. P. M.. A survey on the accessibility awareness of people involved in web development projects in Brazil. In: International cross-disciplinary conference on Web accessibility (W4A), 2008, Beijing, China. Proceedings of the 2008 international cross-disciplinary conference on Web accessibility (W4A). New York, NY, USA : ACM Press, 2008. v. 1. p. 87-96.
9. Gehtland, Justin; Galbraith, Ben; Almaer, Dion. Pragmatic Ajax: A Web 2.0 Primer. The Pragmatic Bookshelf. October, 2005.
10. Gibson, B. 2007. Enabling an accessible web 2.0. In Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A).(Banff, Canada, May 7-8, 2007). W4A2007, vol. 225. ACM, New York, NY. DOI= http://doi.acm.org/10.1145/1243441.1243442.
11. Gibson, B. 2007.DHTML accessibility: solving the JavaScript accessibility problem. In Proceedings of the7th international ACM SIGACCESS conference on Computers and Accessibility.(Baltimore, USA, October 9-12, 2005). ASSETS’05. ACM, Baltimore,MD. DOI=

http://doi.acm.org/10.1145/1090785.1090830.
12. Munson, Ethan Vincent; Pimentel, Maria da Graca Campos . Specialised Documents. In: Simon Harper; Yeliz Yesilada. (Org.). Web Accessibility: A Foundation for Research (Human-computer Interaction Series). 1 ed. : Springer-Verlag London Ltd (31 Mar 2008), 2008, v. 1, p. 1-14.
13. Owen, S. and Rabin, J. (Eds.) (2007, November 30). W3C mobileOK Basic Tests 1.0. (W3C Candidate Recommendation). Available at http://www.w3.org/TR/mobileOK-basic10-tests/
14. Seeman, L. Roles for accessible rich internet applications (WAI-ARIA roles). W3C Working Draft 20 December 2006, http://www.w3.org/TR/aria-role/.
15. Shelly, C. C. and Young, G. 2007. Accessibility for simple to moderate-complexity DHTML web sites. In *Proceedings of the 2007 international Cross-Disciplinary Conference on Web Accessibility (W4a)* (Banff, Canada, May 07 - 08, 2007). W4A '07, vol. 225. ACM, New York, NY, 65-73. DOI= http://doi.acm.org/10.1145/1243441.1243460
16. Thiessen, P. and Chen, C. 2007. Ajax live Regions: ReefChat Using the Fire Vox Screen Reader as Case Example. In Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A).(Banff, Canada, May 07-08, 2007). W4A2007, vol. 225. ACM, New York, NY. DOI=http://doi.acm.org/10.1145/1243441.1243448.
17. Treviranus, T., et al. eds. "Authoring Tool Accessibility Guidelines 1.0." W3C Recommendation (2000) http://www.w3.org/TR/2000/REC-ATAG10-20000203
18. Treviranus, T., et al. eds. "Authoring Tool Accessibility Guidelines 2.0." W3C Working Draft (2004) http://www.w3.org/TR/2004/WD-ATAG20-20041122/
19. Vanderdonckt, J., Beirekdar, A., Noirhomme-Fraiture, M., Automated Evaluation of Web Usability and Accessibility by Guideline Review, Proc. of 4th Int. Conf. on Web Engineering ICWE'04 (Munich, 28-30 July 2004), N. Koch, P. Fraternali, M. Wirsing (eds.), Lecture Notes in Computer Science, Vol. 3140, Springer-Verlag, Berlin, 2004, pp. 17-30
20. Vanderdonckt, J., Beirekdar, A., Automated Web Evaluation by Guideline Review, Journal of Web Engineering, Vol. 4, No. 2, 2005, pp. 102-117.
21. Vanderheiden G, Chisholm W, Jacobs I (1998) WAI accessibility guidelines: page authoring. Technical report, WD-WAIPAGEAUTH-19980918. W3
22. W3C Accessible Rich Internet Applications (WAI-ARIA) Version 1.0, Cooper, M; Schwerdtfeger, R; Pappas, L. Editors, W3C Working Draft (working in progress). 4 February 2008. Avaible at http://www.w3.org/TR/wai-aria/
23. W3C, Cascading Style Sheets, Available at: http://www.w3.org/Style/CSS/
24. Wood, L., Hors, A. L., Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Nicol, G., Robie, J., Sutor, R., and Wilson, C. (Eds). 1998. Document Object Model (DOM) Level 1 Specification Version 1.0. W3C Recommendation. Available at http://www.w3.org/TR/REC-DOM-Level-1/.DOM