Univ Access Inf Soc
DOI 10.1007/s10209-014-0399-3

LONG PAPER

An extensible environment for guideline-based accessibility evaluation of dynamic Web applications

Antonio Giovanni Schiavone · Fabio Paternò

© Springer-Verlag Berlin Heidelberg 2015

Abstract During the last decade, Web site accessibility and usability have become increasingly important. Consequently, many tools have been developed for automatic or semi-automatic evaluation of Web site accessibility. Unfortunately, most of them have not been updated over time to keep up with the evolution of accessibility standards and guidelines, thus soon becoming obsolete. Furthermore, the increasing importance of CSS in the definition of modern Web page layout, and the increasing use of scripting technologies in dynamic and interactive Web sites, has led to new challenges in automatic accessibility evaluation that few of the existing tools are able to face. This paper describes MAUVE, a software environment for Web site accessibility and usability evaluation. The tool is characterized by the possibility to specify and update the guidelines that should be validated without requiring changes in the tool implementation. It is based on an XML-based language for Web Guidelines Definition. It allows checking both HTML and CSS to detect accessibility issues and is able to validate dynamic sites as well, based on the use of a set of plugins for the most popular browsers.

Keywords Guidelines · Guideline specification language · Accessibility · Validator · Dynamic Web page

A. G. Schiavone (✉) · F. Paternò
HIIS Laboratory, CNR-ISTI, Via G. Moruzzi 1,
56124 Pisa, Italy
e-mail: antonio.giovanni.schiavone@isti.cnr.it

F. Paternò
e-mail: fabio.paterno@isti.cnr.it

1 Introduction

Although first introduced in the USA in the late 1990s, Web accessibility has become increasingly important over the last decade, both through efforts of many public institutions and through the work of several international organizations. For instance, the e-Europe Action Plan [1] can be mentioned, accepted by the member countries of the European Union in the early 2000s, which prescribed that any public Web site should be made accessible to people with disabilities. According to this resolution, companies in charge of developing Web sites for any European public administration should develop sites adhering to the latest standards in Web site accessibility.

Following this initiative, many European countries adopted laws about accessibility in public administration Web sites. For example, in Germany, in 2002, a law named "Barrierefreie-Informationstechnik-Verordnung" (BITV) [2] was approved, in 2004, in Italy, the "Legge Stanca" [3] was enacted, in 2007, in Spain, the law "Reglamento sobre las condiciones básicas para el acceso de las personas con discapacidad a las tecnologías" [4] was adopted, and in France, in 2009, the law named "Référentiel général d'accessibilité pour les administrations" (RGAA) [5] was approved. In the same years, many other non-EU countries adopted legislation to promote the accessibility of public administration Web sites (for example, South Korea in 2002, Brazil and Japan in 2004, Chile in 2006, Canada and Honduras in 2007). This growing interest in Web accessibility is explained by the facts that the Web is becoming one of the main means of communication between both individuals and institutions and citizens and that a significant share of the world's population is affected by disabilities of various types and severity. Indeed, numerous studies have shown that the proportion of people with a

Published online: 15 January 2015

disability varies between 10 and 20 % of the population of each country (for example, the United Nations estimated that 10 % of the world population suffers from a disability [6], the US Census Bureau believes that the 18.7 % of the US population is disabled [7], the European Commission estimates that "one in six people in the European Union – around 80 million – have a disability that ranges from mild to severe" [8]).

Other important actors in Web accessibility are international organizations, most notably the W3C, through its Web Accessibility Initiative (WAI), which has become the main reference in the Web community: For years WAI's Web Content Accessibility Guidelines (WCAG) 1.0 [9] and the subsequent WCAG 2.0 [10] played the role of "de facto" standards in Web accessibility, and since September 2012 WCAG 2.0 is also an ISO standard [11].

Following this growing interest in Web accessibility, in recent years, various groups have begun to develop software able to check, automatically or semi-automatically, the correspondence between the requirements of some accessibility guidelines and the characteristics of Web pages under consideration. These tools are useful for those involved in developing Web sites, as through them Web designers and developers can easily and quickly check whether their work meets (part of) the requirements of the considered guidelines, and, in case of failure, they are able to suggest the appropriate corrections.

Unfortunately, many of these tools, called validators, have not been updated over time, becoming in a few years inadequate with respect to the evolution of legislation and standards on accessibility and information technologies.

Examining the list of accessibility validators on the W3C's Web Accessibility Initiative homepage,[1] it can be seen that approximately 65 % of the validators listed are no longer available, and about 50 % of the available ones are not compliant with the latest WCAG 2.0.

The obsolescence of these tools could be due to several reasons: In some cases, it may be due to a decrease in interest by the validators' authors, though in other cases the authors might have considered the upgrading process, and the consequent extensive rewriting of the code, as too burdensome.

At the same time, Web-related technologies have evolved significantly, especially in the direction of a World Wide Web able to convey information more interactively. For this reason, over the years, the importance of the graphical layout in the development of Web sites (especially through the evolution of style formatting languages such as Cascading Style Sheets—CSS) has greatly increased.

[1] http://www.w3.org/WAI/ER/tools/Overview.html (accessed on 2 October 2014).

Furthermore, a growing trend in Web site development is the creation of dynamic Web pages through the use of various scripting languages such as JavaScript. With the use of such techniques, Web sites are able to convey content that can be updated dynamically even in part and through richer interactions with the user.

The use of such new Web-related technologies has determined new issues in automatic validation of Web site accessibility, requiring a deeper analysis of Web pages' characteristics.

This paper presents a solution for automatic accessibility evaluation of modern Web pages, based on a new approach that involves the specification of a high-level language for the definition of accessibility guidelines and the use of browser plugins to validate dynamic Web pages.

In particular, after discussing related work, the authors illustrate the architecture of the proposed environment and introduce the language to specify guidelines that have been developed based on previous related experiences. Then, the processing carried out in the validation is detailed, describing the dynamic Web page validation process and reporting on a comparative study with two other widely known validation tools. Lastly, the authors draw some conclusions and provide indications for future work.

2 Related work

As mentioned above, in recent years, various tools have been developed for the automatic evaluation of multiple guideline sets. Such environments can be grouped into two categories: Tools that have the guidelines' definition formalized through an external language and thus clearly separated from the evaluation logic, and tools that have the definition of guidelines internally defined or directly embedded in the evaluation logic implementation.

The second group includes the majority of the tools developed for the evaluation of guidelines, since their construction is certainly simpler and more immediate. However, updating this type of tool to new sets of guidelines is a process that requires an extensive rewriting of the validator code.

In many cases, the update process could be a commitment too burdensome for the tool's developers. There are few validators in this category still active and updated: One of the oldest existing validators is AChecker [12], an open source project started in 2004 and developed by the University of Toronto, which allows validation of HTML documents with respect to four different guidelines (BITV, the Stanca Act, WCAG 1.0 and WCAG 2.0).

Another validator belonging to the second group is Test Accessibilidad Web (TAW) [13], a project of the Spanish Fundación CTIC30, promoted in collaboration with the

Government of the Principality of Asturias. This tool provides validation for WCAG 1.0, WCAG 2.0 and mobileOK, a standard promoted by the W3C to ensure usability and interoperability of Web pages on mobile devices.

More recently, WaaT [14] proposed a different approach: This tool is based on an ontologies database. A Rules Inference Engine is able to extract guideline definitions from this database in order to perform the accessibility validation: Rules description and database queries are performed through two well-known languages (SWRL and SPARQL respectively). With this approach, the guidelines are expressed through a procedural language instead of a declarative one: For this reason, WaaT is considered as belonging to the second group.

The development of the first group of tools is based on some specific language for guideline definition, independent of the evaluation engine. Therefore, these tools are more flexible as it is possible to modify, update or aggregate new guideline sets without requiring changes in the implementation of the evaluation engine. Some examples of this kind of tools can be found in the literature, often with the support of XML-based languages for the guideline definitions.

The guideline definition language (GDL) [15] was one of the first languages for guidelines' definition supported by a related framework for static HTML page analysis for automatic usability evaluation.

The first accessibility validator to implement this type of approach was Kwaresmi (Knowledge-based Web Automated Evaluation with REconfigurable Guidelines Optimization) [16] that used an XML-based guideline representation called simple guideline specification language (SGSL). Takata et al. [17] proposed a pseudo-XQuery language for accessibility evaluation purposes and used XPath sentences for implementing WCAG guidelines: Implementation of the evaluation logic through this technology was less complex and, usually, more compact. However, these contributions only addressed simple structures for guidelines specifications.

Another XML-based language for guidelines definition was the Unified Guidelines Language (UGL), proposed by Vigo et al. [18]: This language adopted the same approach proposed by Takata, using XPath sentences to implement the guidelines.

UGL was based on the taxonomic analysis of some sets of guidelines and the classification of the possible relationships between the various components of a generic guideline. This approach can be cumbersome for expressing possible guidelines to check. The UGL language has been subsequently refined [19] to include the opportunity to express requirements related to the CSS code inside the guidelines.

Finally, Leporini et al. [20] proposed the GAL language: This solution although valid with regard the "state of the art" of the time, proved to be insufficiently expressive to formalize the new generation of more complex guidelines.

In recent years, one of the major innovations has been the rise of the technologies so-called "Web 2.0," which have radically changed the concept of the Web as a whole. From being a static repository of information (not very different from a book), Web sites have become complex objects able to provide the user with a compelling interactive user experience.

At the same time, the emergence of these technologies has created new issues on the accessibility of Web sites. An interesting example of the impact of "Web 2.0" technologies on Web sites accessibility and assistive technologies was presented by Brown et al. [21].

The literature contains some attempts to solve the issues related to the accessibility validation of dynamic Web pages: A first contribution was proposed by Fuertes et al. [22], who developed Hera-FFX, a Firefox add-on for client-side accessibility validation of dynamic Web pages. This add-on performs accessibility tests on the basis of guideline definitions contained inside a locally stored XML Configuration file: As a consequence, to update these guidelines the user must explicitly install an updated version of the whole add-on. Furthermore, accessibility validation was triggered after the browser had fully loaded the Web page and generated the Document Object Model (DOM) representation. As consequence, it is not possible to test the accessibility of Web pages if dynamically modified at a later time (for example in response to a user interaction with an element of the page or following the reception of some data via asynchronous communication).

Another work on this topic by Chen et al. [23] focused on the identification and validation of widgets, i.e., small sections of Web pages frequently and dynamically updated. Although interesting, this approach focuses only on a subset of the issues related to the accessibility validation of dynamic pages; in fact, although widespread, widgets represent only a small part of the components of a Web page that can be dynamically modified.

One of the latest dynamic Web pages accessibility validator is QualWeb evaluator 3.0, a tool proposed by Fernandes et al. [24]: The 3.0 version includes a module that pre-processes the Web page, simulates the user interaction on all the clickable elements of the page and generates the resulting set of possible Web page states. All such Web page states are subsequently passed to the validator component in order to perform the accessibility validation.

As in the previous case, this approach ignores part of the possible changes that may occur within a dynamic page, such as the execution of animations that do not require user interaction or the automatic update of widgets' content.

3 Validators' categorization

In Sect. 2 a, categorization of tools for the automatic evaluation of multiple guideline sets has been presented, based on the fact that the description of the guidelines was expressed through an external language or internally defined. Another possible categorization of these types of tools can be based on the approach to the validation they perform: In fact, one can distinguish between a "deterministic" and a "probabilistic" approach to the validation.

The first approach is the most commonly used by guideline validators: According to it, since generally not all guidelines are easily translatable in controls executable by a computer, only a subset of the guidelines is considered in the validation performed by the tool. This implies that the control of the remaining guidelines should be performed by a human validator.

With the "probabilistic" approach, the validation tool tries to infer (nearly) all the validation issues, including those not easily calculable by a computer, indicating the most common errors.

One example of this approach is the aforementioned Achecker [12] tool, which classifies accessibility issues into three categories:

- Known Problems, i.e., problems that AChecker knows with certainty are accessibility barriers.
- Likely Problems, i.e., problems that AChecker identified as possible accessibility problems, for which a human decision is required.
- Potential Problems, i.e., aspects that AChecker cannot assess and may have some accessibility consequence, which also require a human check.

Due to the high number of false positives reported, this type of approach is not very useful and somewhat confusing for a user, especially if not expert. Referring to the example of Achecker, "Potential Problems" are often detected in large quantities and provide very general indications often poorly related with the real Web page accessibility.

Another possible categorization of guidelines validators is related to how they show the validation results: Most of them use a "code-oriented" approach, reporting (totally or partially) the HTML code of the validated page and highlighting the detected errors.

Some validators use a "graphical" approach to showing the validation results, visually rendering the Web page under analysis and locating the errors on the user interface page through some placeholders, labels or other graphical elements: One example of such graphical approach is the WAVE validator [25].

Although this approach allows evaluators to quickly locate page areas that are affected by errors, it often does not allow a clear identification of the type of error, nor an immediate suggestion on how to solve the issue in the implementation. Furthermore, in the case that more errors are found in the same area or in contiguous areas, graphics elements tend to overlap, making error reporting extremely confusing.

Table 1 summarizes the categorization of the accessibility validators cited in this paper, according to the criteria set out in paragraphs 2 and 3.

4 General architecture

This paper presents multi-guideline accessibility and usability validation environment (MAUVE), an environment for analysis and evaluation of Web sites based on their compliance to accessibility and usability guidelines.

In general, the tool has been developed with the intent of checking whether a Web site is accessible as well as usable according to a set of guidelines. For this reason, it is foreseen that the typical users of the tool will be Web designers, Web developers and Web programmers who want to test the quality of their work with respect to accessibility and usability.

Table 1 Validator categorization

Name	Guideline definition	Approach	Validation object	Report style
Achecker	Internally defined	Probabilistic	Static Web page	Code oriented
TAW	Internally defined	Deterministic	Static Web page	Graphical + statistical
WaaT	External semantic Web rule language	Deterministic	Static Web page	Code oriented
Kwaresmi	External XML language	Deterministic	Static Web page	Code oriented
Hera-FFX	External XML configuration file	Deterministic	Static and dynamic Web page	Graphical + code oriented
QualWeb	Internally defined	Deterministic	Static and preprocessed dynamic Web page	Statistical
WAVE	Internally defined	Deterministic	Static Web page	Graphical
TotalValidator	Internally defined	Deterministic	Static Web page	Code oriented
MAUVE	External XML language	Deterministic	Static and dynamic Web page	Code oriented

The tool checks how satisfactory is the application of the selected guidelines to the Web pages: This is obtained through automatic identification of the checkpoints associated with each guideline and analysis of the associated constructs and attributes to check whether they provide the necessary information. This new tool has been based on previous work [20] and aims to address various issues that such previous work was not able to consider: The proposed new solution is also able to analyze the CSS content associated with Web applications and check complex guidelines structures with more flexible relations among them and aspects that were not supported in various previous tools. Moreover, the environment presented also proposes a solution for the problem of dynamic Web page validation, through the use of a set of extensions for the most popular browsers.

MAUVE is composed of an abstract language for the definition of guidelines and its associated interpreter, which is able to validate Web pages according to the guidelines specified by the language. In particular, the abstract language used is named Language for Web Guideline Definition (LWGD) and enables a simple and flexible formalization of guidelines, usually defined using natural language. As it is an abstract language, it is not tied to a particular set of guidelines. Rather, it is proposed as a general tool with the ability to describe various types of guidelines.

The validation can be performed through a command line, a Web interface (useful in the case that the validation is performed by a human user) or a browser extension. Moreover, if the validation is performed via the Web interface (see for example Fig. 1), it is possible to validate Web pages by uploading them from the local computer, providing the remote URL or pasting the HTML code inside a dedicated box.

Furthermore, the tool offers the opportunity to perform the validation with respect to one of the pre-defined sets of accessibility guidelines, or to upload a custom set of guidelines (formalized through LGWD language) and to perform the validation with respect to this set.

The validation results are specified in XML files, useful in the case that the validator is used as a library by other software.

In recent years, Web developers have started to develop several versions of their Web applications. Thus, each one can be specific to a certain type of device (e.g., personal computers, smartphones, tablets, and also game consoles and SmartTVs in some cases). MAUVE has the ability to select and recover (and thus validate) the version of a Web site specific to a certain type of device: The evaluator can indicate the target platform and the tool accesses the Web site with the user agent corresponding to the selected platform.

The guidelines are stored in a repository external to the tool: At each page validation, a specific module retrieves the selected guidelines and checks guidelines consistency with respect to the LWGD language.

The structure of the MAUVE validator is made up of five main modules, as illustrated in Fig. 2:

- *MauveEngine*, main module and actual performer of Web page validation.
- *LWGDManager*, module for the management, access and check for the guidelines represented through the LWGD language. Guidelines are stored as XML files.
- *Commons*, module for retrieving Web pages and for the management of temporary data.
- *PreProcessor*, module for Web page clean up and DOM generation.
- *ReportManager*, module for validation report management and their possible export as an XML file.

The proposed environment has been designed to validate one Web page at a time. However, it is possible to extend the validation to entire sites by simply using the environment as a library and exploiting a crawler that, given the URL of a Web site, is able to generate the corresponding stream of pages URLs.

According to the validators categorization illustrated in the previous paragraphs, this environment has the guidelines' definitions formalized through an external language and clearly separated from the evaluation logic. It follows a "deterministic" approach to the validation and, if used through a Web interface or browser plugins, reports validations results with a "code-oriented" approach.

5 The guideline definition language

5.1 A preliminary analysis

In general, a guideline is defined as a rule or principle that provides indications for appropriate content and behavior. Generally, guidelines are formulated in terms of natural languages, which cannot be handled by automatic tools. Thus, a guideline should be specified in a more precise manner so that it can be automatically processed and managed.

A useful preliminary step to the development of a general language for guideline formalization is analyzing and defining the features and key elements characterizing a general guideline set.

These properties can be briefly summarized as follows:

- Features, such as name, id, description and importance;
- Objects involved in the verification process, e.g., which tags, attributes, properties must be considered in the check;
- Conditions relating to objects (e.g., what kind of verification must be performed on the selected object).

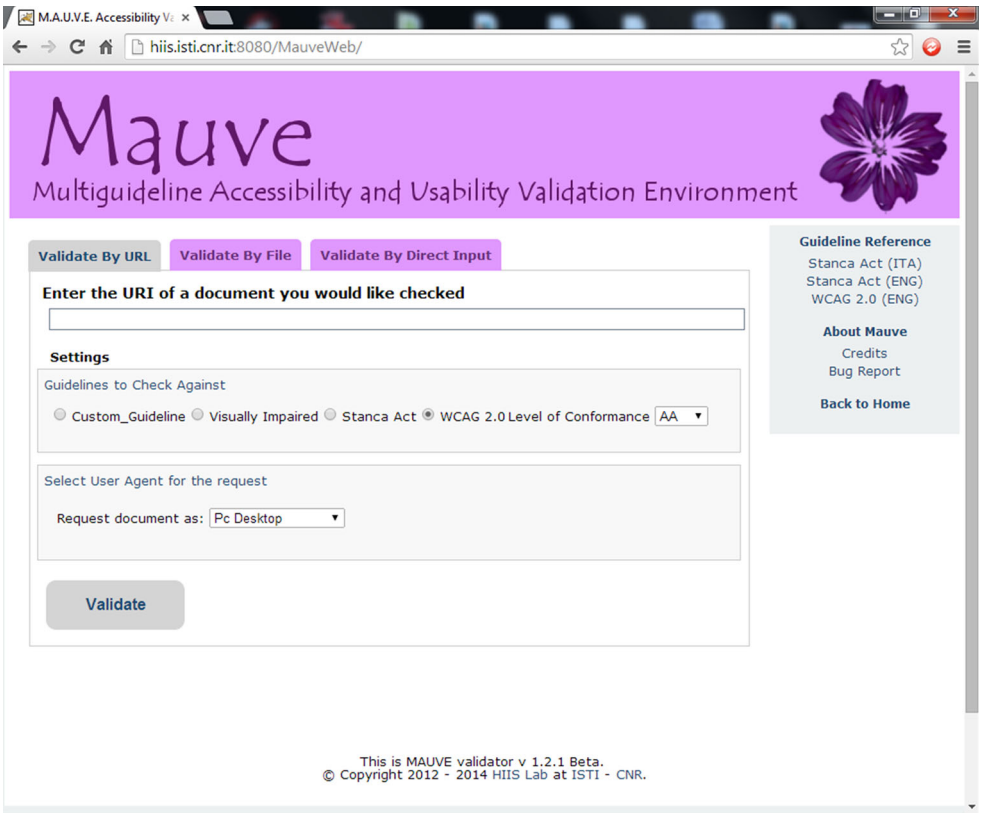

Fig. 1 MAUVE Web interface

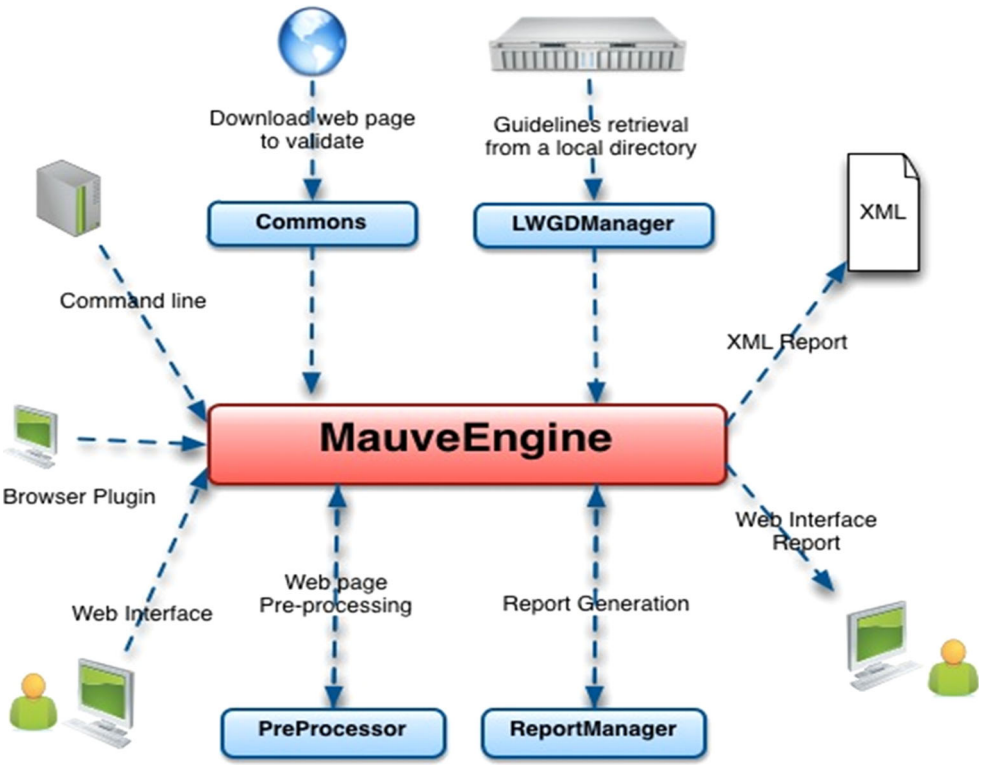

Fig. 2 MAUVE validator structure

All such items are basic and indispensable components for structuring a guideline set: It is necessary to specify the general characteristics of the tag or property involved in the validation process, and the conditions that must be checked to detect whether the guidelines have been satisfied or not.

By analyzing the most recent accessibility guidelines, it has been noticed that sometimes, in order to satisfy a guideline, it is necessary to be able to express relationships between requirements regarding different objects of analysis. It may also be necessary for more than one requirement to be satisfied simultaneously, or satisfied in a mutually exclusive manner, or related to each other in some other way.

For instance, let us consider the WCAG 2.0 Failure of Success F65: "This describes a failure condition for text alternatives on images: If there is no alt attribute, then assistive technologies are not able to identify the image or to convey its purpose to the user." This statement can be translated in this procedure: "Identify any img, area and input elements of type "image" and check that the alt attribute for these elements exists."

To meet this criterion, it is therefore necessary to ensure that all <img>, <area> and <input> tags of type image (if present on the page) simultaneously satisfy the requirement of having an alt attribute.

Further useful information for the development of a general language for guideline formalization comes from the analysis of the existing languages for the same purposes.

GDL was one of the first languages for the formalization of guidelines, which indicated some of the basic features that a language developed for this purpose should have. For instance, one characteristic is the "naturalness, i.e., the possibility offered by the language to straightforwardly map the informal statement of initial guidelines onto formal statement." Another characteristic was the possibility to define evaluation conditions of a certain complexity, formalized as expressions and able to describe some requirements regarding more than one object involved in the verification process.

Despite these positive aspects, GDL has several limitations: The requirements expressed are fairly simple (for example, it is not possible to specify complex relationships between two tags such as "a specific tag is child of another specific tag"), the analysis was limited to only HTML (no analysis of CSS), the "meta evaluation" (i.e., the formalization of conditions through the definition of macros) contrasts with the idea of naturalness that should be the basis of the definition of language.

UGL is a GDL derived from the analysis of a set of guidelines and the categorization of the typical requirements of such guidelines in "test cases." It has been revised and upgraded several times in order to be compatible with new Web technologies such as CSS or with new and more complex guidelines, such as the WCAG 2.0. In recent years, the language has thus demonstrated the need for constant maintenance, and the system of "test cases" showed poor naturalness, so that the authors of the language have repeatedly recommended the use of tools for computer-assisted guidelines formalization.

In addition, due to some problems in the language interpreter, UGL has shown some limitations related to the formalization of the style sheets-related requirements, in particular in relation to the application of the requirements in the case of inheritance and overriding of style sheets rules.

5.2 Defining a new guideline definition language

In previous work, GAL [20] was equipped with an ordered and well-organized structure, though it turned out to suffer from significant limitations.

First of all, its formalization of guidelines allowed expressing rather simple requirements, involving a single object of analysis. As discussed previously, this does not satisfy the requirements of the most recent sets of guidelines: For example, it was not possible to express a rule such as the aforementioned F65, as the analysis must be carried out simultaneously on three types of objects (<img>, <area> and <input>).

Secondly, the operators offered a limited set of validation functions on the elements of a Web page. For instance, it was not possible to require that an element in a Web page not have two specific children (as required by WCAG 2.0 Failure of Success F46) or verify that two elements were connected by the value of some of their attributes (as required by Techniques for WCAG 2.0 H44).

Finally, it was not possible to express requirements concerning the analysis of style sheets, increasingly important elements in Web page development and consequently more and more important in the evaluation of Web site accessibility.

From these observations, the authors were prompted to develop a new language for the definition of guidelines, which goes beyond such limitations and offers greater expressive power. In particular, the goal was to obtain a new language able to:

- Handle more fine-grained guidelines;
- Express complex conditions, regarding multiple objects
- Select the objects through more fine-grained filters
- Express complex requirements, based on a greater number of types of relations between objects
- Extend the analysis to the style sheets also considering style inheritance and overwriting.

In developing this abstraction language, the possibility of specifying the objects involved in the verification

process more precisely was introduced. For example, it is possible to select specific objects (tags) having given attributes with specified values (e.g., links with the class = "navbar") or to select objects (tags) that are children of other tags in the DOM structure (e.g., image links, coded in HTML as an <img> tag, which is a child of the tag <a>).

A hierarchical organization of the guidelines has also been provided, which starts from a more abstract level (expressing the general principles of design guidelines and their characteristics, such as the description and type), and then reaches a more detailed level indicating what must be technically checked in the code (which tag or attribute characteristics must be checked). The aspects considered are the following:

- Guideline set: set composed of one or more guidelines;
- Guideline: composed of one or more criteria, it expresses general concepts about the accessibility of Web pages (for example, "Provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language");
- Criterion: composed of one or more checkpoints, it specializes concepts from a guideline, focusing on a particular aspect of the Web pages (for example, "All non-text content that is presented to the user has a text alternative that serves the equivalent purpose");
- Checkpoint: consists of one or more checks and expresses concretely the requirements that must be met by one or more components of a Web page (tags, attributes, CSS properties etc.) (for example, "Accessibility issue, due to omitting the alt attribute on img elements, area elements, and input elements of type image");
- Check: focuses analysis on one type of Web page element (for example, on all <input> tags on the page or on all the "font-weight" properties defined in the CSS code), expressing the requirements that they must satisfy through one or more conditions;
- Condition: expresses a requirement that must be met by one type of Web page element (for example, express the requirement "must have an alt attribute and its value must be not empty"). It is the atomic element of the validation.

To be able to express relationships between requirements associated with different objects, a recursive approach is used to define relations between different checks according to the following grammar:

$$\begin{aligned} L &= L\ \mathrm{OpBin}\ L|\mathrm{OpUn}\ L|T; \\ T &= A|B|C|D\ldots; \\ \mathrm{Opbin} &= \mathrm{and}\,|\mathrm{or}|\ldots; \end{aligned}$$

where OpBin are binary Boolean functions, OpUn are unary Boolean functions, L is the non-terminal symbol from which the language may be derived and terminals A, B, C, D… are requirements regarding a single selected object. With this approach, it is possible to formalize requirements as Boolean functions of any complexity.

This recursive approach has also been applied to formalize relations between several "conditions": As such, they are also formalized as Boolean functions with arbitrary complexity.

To better understand the differences between GAL and LGWD, a fragment of the XSD Schema of the two languages is compared.

Table 2 shows one limitation of the GAL language: According to the definition indicated, and naming A, B, C, D some atomic evaluation (i.e., the "evaluate" element), it is clear that with using language it is possible to formalize some complex conditions (for instance, "A and (B or (C and D))"), though not all possible conditions (for instance, "(A and B) or (C and B)" cannot be formalized).

Table 3 shows the fragment extracted from the LGDW XSD Schema corresponding to that shown in Table 2. It is easy to see that with this schema it is possible to formalize any complex condition, including "(A and B) or (C and B)."

With regard to the formalization of individual conditions, each of them can be expressed by four basic operators (Check, Count, Execute, Exist).

Each of the four basic operators is further specified by secondary operators: For instance, the semantic of the basic operator Check could be refined by the secondary operator "followedby," meaning that the page element under consideration, in the HTML code, must be followed by a particular tag (i.e., in the page DOM, the following element must be a particular tag).

More than thirty secondary operators (and their opposites) have been defined, which allow for the tool to check the main relationships between Web page elements (HTML tags and CSS properties). For instance, one can check whether an element in the HTML is preceded by a particular tag, whether it is visible or not, or is associated with a particular property, and whether the value of this property is expressed by a particular unit of measure.

In a preliminary version of the grammar previously shown, the definition of a unary Boolean function was also included for the formalization of negation inside the expressions. In a subsequent revision of this grammar, it was chosen to remove the negation from the definition of expressions and to include, for each secondary operator, its opposite function.

This choice is the result of an implementation need: The validator based on such language provides not only a

Table 2 Fragment of GAL's XSD Schema

```
...
<xs:element name="conditions">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="evaluate" maxOccurs="unbounded"/>
            <xs:element ref="conditions" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="rel" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="and"/>
                    <xs:enumeration value="or"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
</xs:element>
....
```

Boolean value, but also a list of items that violate the expressed requirement.

Including the negation would mean, for each test on atomic requirements (i.e., for each terminal), having to include both the list of items that meet the requirement and the list of those which do not (to be exchanged in the presence of negation): This, together with the recursion, would make report errors overly complicated and computationally heavy.

As mentioned above, the choice was therefore to "push" the negation to the terminal elements of the grammar, to create the negative for each secondary operator.

This solution, although detouring slightly away from natural language, does not limit the expressive power of the language developed: Rules to transform Boolean expressions to other, semantically equivalent ones that have negation pushed to the terminals are known in Boolean algebra, for instance, the Boolean expression:

Not ((A or B) and (Not (C and D)))
is semantically equivalent to:
((Not A and Not B) or (C and D))

5.3 A formalization example

In order to understand how the abstract description of a guideline can be obtained through LWGD, one example can be used, selected from among the more than 40 distinct WCAG 2.0 Techniques that, at the time of writing, had been formalized through LWGD. The WCAG 2.0—Technique H46 is considered, which is defined as "a technique to provide alternative content for the embed element in a noembed element. The noembed is rendered only if the embed is not supported. While it can be positioned anywhere on the page, it is a good idea to include it as a child element of embed so that it is clear to assistive technologies that a text alternative is associated with the embed element it describes."

This natural language description can be translated into a more structured sentence: "For each <embed> element inside the Web page, check whether it has a child <noembed> element or whether it has a <noembed> element that immediately follows it."

The formalization through LWGD of Technique H46 is shown in Table 4: The technique is formalized through a single check, which selects, as objects of evaluation, all of the <embed> tags on the page. The requirements are expressed through a condition expression: The first branch of the expression states that the objects of evaluation must be followed by the tag <noembed> and the second branch of the expression states that the objects of evaluation must have the tag <noembed> as child. The two branches are connected by an "or" operator, so the requirement is satisfied if at least one of the branches is satisfied.

6 The checking process

This section describes the evaluation logic of the MAUVE validator (illustrated in Fig. 4). The validation starts by inputting the needed data to the validator (via command line or via a Web interface), i.e.:

- URL of the Web page to validate

Table 3 Fragment of LGWD's XSD Schema

```
....
<xs:element name="conditions">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="condition"/>
      <xs:element ref="condition_expression"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
<xs:element name="condition_expression">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="conditions" minOccurs="2" maxOccurs="2"/>
    </xs:sequence>
    <xs:attribute name="rel" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="and"/>
          <xs:enumeration value="or"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="condition">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="evaluate"/>
    </xs:sequence>
    <xs:attribute name="cond">
      .....
      </xs:attribute>
    </xs:complexType>
  </xs:element>

.....
```

- The device type for which the tool should retrieve the specific version (if any)
- Name of the guideline set against which to perform validation or local path to retrieve the uploaded custom guidelines
- Level of compliance to the guideline set.

After receiving such information, the validator loads the version of the Web page for the selected device type and its associated style sheets and then retrieves the selected guideline set from a repository or, if the user uploaded a custom guideline set, from a temporary directory. In both cases, guidelines are stored as XML file.

With such data, it starts the validation phase: Each guideline in the considered set is hierarchically unfolded through its constituent levels (criteria, checkpoints, checks, conditions, as defined in the previous section), and then the validator starts to verify that the rules expressed by the guidelines' different levels are fulfilled by the Web page.

As explained in the previous section, relations between different "checks" and "conditions" can be specified using Boolean expressions of any complexity in a recursive approach according to the grammar described above.

If a guideline has been formalized through one of these Boolean expressions, the MAUVE validator carries out

Table 4 Example of guideline abstractions with LWGD

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<gdl_set xmlns="http://giove.isti.cnr.it"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://giove.isti.cnr.it lwgd.xsd"
 shortname="wcag20">

  <name> Web Content Accessibility Guidelines (WCAG) 2.0</name>
  <guideline id="1.1" summary="guideline Summary" src="req_it.htm">
    <description>Guideline description</description>
    <criterion id="1.1.1" summary="Criterion summary" level="A"
   type="accessibility" target="page">
      <description>Criterion Description</description>
      <checkpoint id="H46" summary="Using noembed with embed">
        <description>The objective of this technique is to provide alternative content for the embed element in a
noembed element. </description>
        <checks>
          <cp>
            <cp_descript>Validation for Technique H46</cp_descript>
            <eval_object mandatory="no" code="html">
              <object type="tag">embed</object>
            </eval_object>
            <conditions>
              <condition_expression rel="or">
                <conditions>
                  <condition>
                    <evaluate iderr="H46-1" operator="check" cond="followedby">
                      <el type="tag">noembed</el>
                    </evaluate>
                  </condition>
                </conditions>
                <conditions>
                  <condition>
                    <evaluate iderr="H46-2" operator="check" cond="hasaschild">
                      <el type="tag">noembed</el>
                    </evaluate>
                  </condition>
                </conditions>
              </condition_expression>
            </conditions>
          </cp>
        </checks>
      </checkpoint>
    </criterion>
  </guideline>
</gdl_set>
```

guideline verification by recursively checking the expression's sub-branches, until it reaches the verification of its atomic components and then it continues to the next constitutive level.

Once the lowest level of the hierarchy of a guideline's definition is reached (the operators used to define the condition), the existing instances of the four basic operators (Check, Count, Exist, Execute) are checked. The validation results are stored in a special data structure, and, in the event that one or more elements of the HTML code do not meet the operators, the corresponding line numbers in the source code are stored.

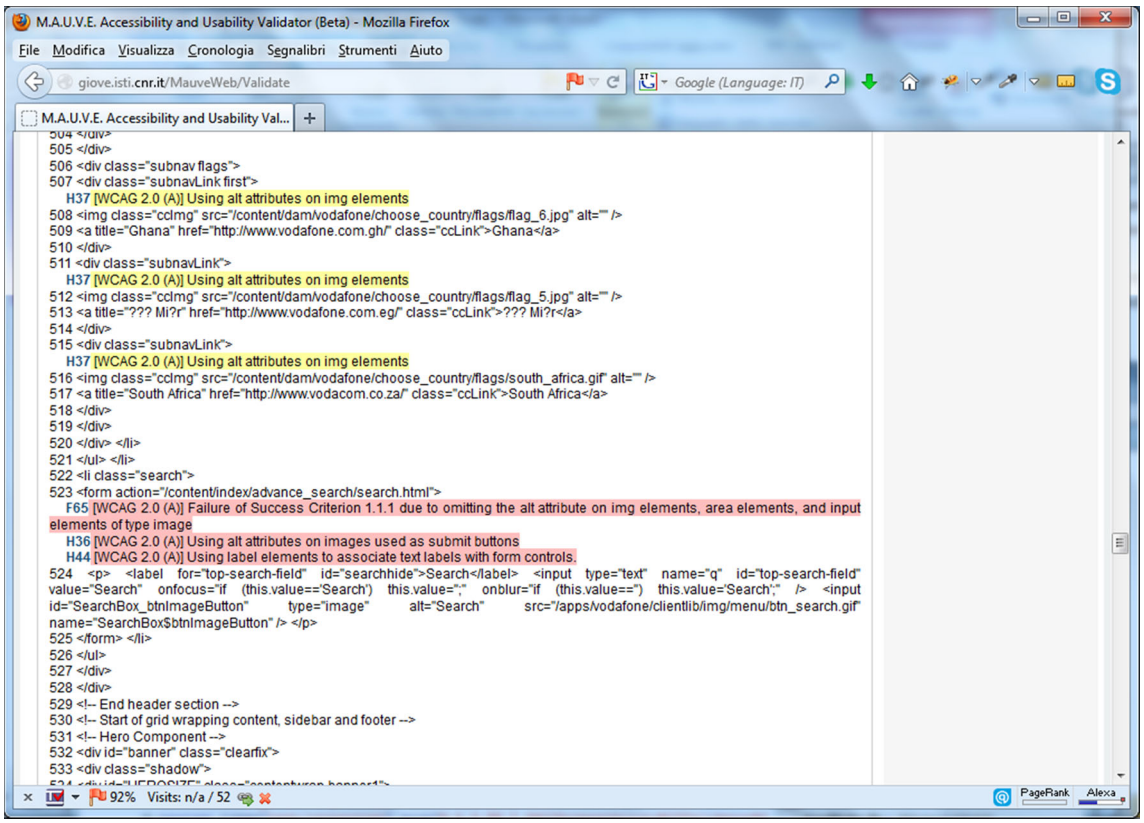

Fig. 3 Excerpt of a validation report

The results are collected "bottom-up" through the various validation levels: If the validator had previously made recursive calls for the "checks" or "conditions" levels, the results would have recursively merged in these levels in accordance with the Boolean operators.

After all the guidelines have been evaluated, the results of the validation are exported via an XML report or via the Web interface; in this second case, the validator generates a Web report showing the detected errors directly in the Web page source code (an excerpt in Fig. 3).

As Fig. 3 shows, after the validation, the interface of the validator displays the HTML code of the Web page, enriched, if necessary, with error or warning messages (respectively, in red or yellow), indicating the name of the rule violated and its short description, as well as a links to its reference page.

The choice of this "source-code-oriented" report model is consistent with the initial indication of Web designers and Web developers as typical users of the tool. Furthermore, other features, such as the possibility to use custom guidelines or to analyze HTML code inserted via direct input, have also been designed in order to make more versatile use of the instrument by this type of users. This approach is not very different from the one used by W3C for the development of its validators, such as Unicorn [26].

6.1 The validation algorithm

The evaluation logic is illustrated in Fig. 4, which shows the main components of the validation tool and how the processing is performed across them. The description of the algorithm is described in the following sub-sections.

6.1.1 Web page retrieval and DOM generation

As already described, the HTML file is downloaded from the network (through the Commons library) and passed to the parser that generates the associated DOM (through the PreProcessor library).

If the HTML file contains one or more links to style sheet files, they are downloaded and parsed through an external library: In accordance with the inheritance and overwriting rules of style sheets, this library also calculates, for each of the DOM's nodes, its CSS properties and binds them to the node.

6.1.2 Guideline unfolding and validation

The rest of the validation process essentially consists of unfolding the guidelines according to their hierarchical levels (as outlined in paragraph 5.2) up to their atomic elements.

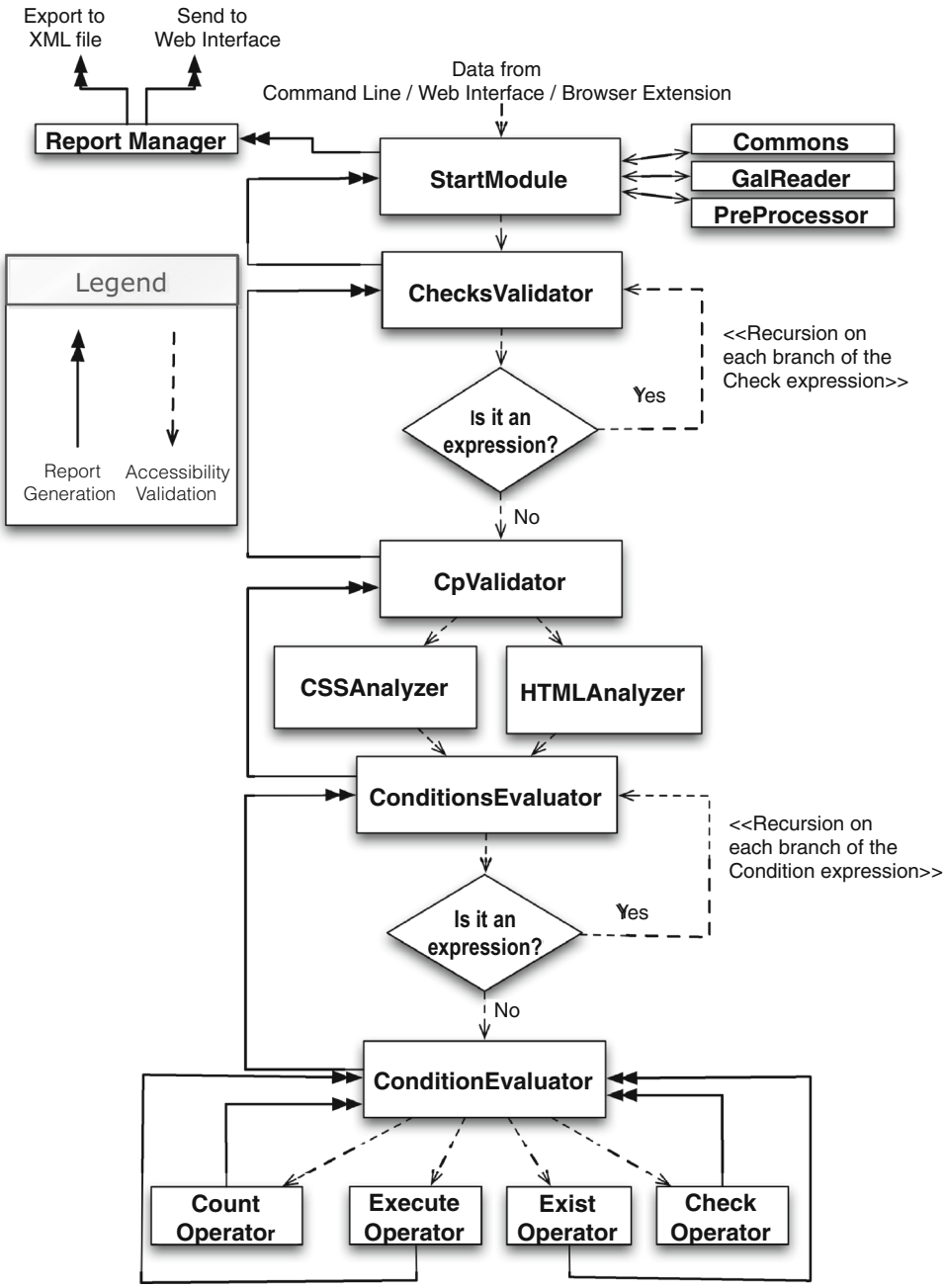

Fig. 4 MAUVE validator's evaluation logic

StartModule unfolds the first three hierarchical levels of the guidelines set: the guidelines that make up the set are selected iteratively, for each guideline the constituting criteria are selected iteratively, and for each criterion the checkpoints that compose it are selected iteratively.

The module ChecksValidator instantiates the first part of the recursive approach of the LGWD language as explained in Sect. 5.2: It checks whether the checkpoint should verify the requirements of more elements (i.e., in the checkpoint formalization, there is a check_expression) or a single type of element of the page (i.e., in the checkpoint formalization there is a single check).

In the first case, the ChecksValidator makes a recursive call on the two branches of the expression, and in the second case the validation continues toward CpValidator.

CpValidator verifies whether the check is related to an HTML object (i.e., a tag or the whole HTML page) or to a CSS type (i.e., a CSS property or a selector) and, depending on the result, redirects the computation to CSSAnalyzer or HTMLAnalyzer.

In both cases, the execution passes to ConditionsEvaluator, which instantiates the first part of the recursive approach of the LGWD language.

In fact, it checks whether the validator should verify different requirements, organized into expressions (i.e., in the checkpoint's formalization there is a condition_expression), or just one requirement (i.e., in the checkpoint's formalization, there is a single condition).

Again, in the first case, the ConditionsEvaluator makes a recursive call on the two branches of the expression, whereas in the second case the validation continues through ConditionEvaluator.

ConditionEvaluator verifies the condition type that should be evaluated and according to it redirects the validation to one of the four operators (CountOperator, ExecuteOperator, ExistOperator, CheckOperator).

The tool finally unfolds the guideline to the lowest level in its hierarchical organization: The selected operator, possibly specialized by secondary operators, scans the DOM and calculates the list of nodes that do not satisfy the condition.

With this information, the execution follows a backward path, in order to generate the report with a bottom-up approach that goes from the lowest level up to the highest in hierarchical organization of the guidelines.

The calculated list is returned to ConditionEvaluator and then to ConditionsEvaluator: If its execution is the result of a recursive call, the two lists resulting from the two recursive calls are merged in agreement with the Boolean operator that characterizes the expression.

After obtaining the list of nodes that do not satisfy the entire expression (i.e., calculating recursively the results of various sub-expressions), this information is passed to CpValidator and then to ChecksValidator.

Similar to what was seen for the ConditionsEvaluator, ChecksValidator recursively reconstructs the result of the Checks expression.

The result of the entire Check expression is the result of the validation with respect to the checkpoint: This information is passed to StartModule that stores it.

6.1.3 Report generation

The process described in the previous sub-section is repeated iteratively, until the page is validated against the whole guideline set. Following the validation of the page with respect to the set of guidelines, a validation report is generated through the ReportManager module.

To export the report to an XML file, a small report language is used, developed specifically for the purpose. In the future, the authors plan to provide the ability to use standard reporting languages as the W3C EARL language.

6.1.4 Further information

The MAUVE validator has been entirely developed in Java, while its Web interface is realized through JSP Servlets. For the tool implementation, some external open source libraries have been used, mainly Jsoup [27] for checking whether the page is well-formed (and eventually to correct of any syntax error), ValidatorNu [28] for the analysis of the HTML code and the generation of its Document Object Model, and jStyleParser [29] for parsing CSS code and binding CSS properties with the DOM's elements.

The LGWD XSD Schema is mapped into a set of nested Java classes through Java Architecture for XML Binding (JAXB), part of the JAVA SE platform. Through this library, the problem of managing, accessing and checking the guidelines is reduced to type checking and object data loading.

Currently, the WCAG 2.0, Stanca Act and Visually Impaired [30] guidelines have been formalized through LWGD and are available in MAUVE for Web page evaluation; more guidelines will be made available in the future. The tool can be accessed at http://hiis.isti.cnr.it:8080/MauveWeb/.

6.2 Validation example

To better understand the validation process, suppose one wishes to validate a Web page with the following HTML code (line numbers have been added for better readability), compared to the Technique H46 whose formalization has already been illustrated in Table 4.

As described in Sect. 6.1.1, the HTML code is parsed, the corresponding DOM is generated, the style sheet file is retrieved and parsed, and finally the CSS properties are bound with DOM's nodes (Table 5).

The StartModule starts to unfold the guidelines set (in this example, the set is composed by one guideline, the guideline is composed by one criterion, and the criterion is composed by only one checkpoint) and calls CheckValidator: The checkpoint corresponding to the formalization of the Technique H46 expresses the requirements inherent only to <embed> tag (single check), and then the execution is forwarded directly to CpValidator, with no need to launch recursion.

CpValidator verifies that Check is related at an object of HTML type (i.e., the <embed> tag), so redirects the computation to the HTMLAnalyzer.

Then, the execution passes to ConditionsEvaluator: It detects that there is a condition expression (i.e., more than one requirement is expressed), so it makes a recursive call to the two branches of the expression. The first branch will

Table 5 HTML code for validation's example

```
1 <!DOCTYPE html >
2  <html lang="en" >
3   <head>
4    <title>My favourite videos</title>
5    <meta charset="UTF-8" />
6    <link type="text/css" rel="stylesheet" href="./css/style.css">
7   </head>
8   <body >
9    <section>
10    <article>
11     <h1> My favourite videos </h1>
12     <div>
13      <h2>The History of Rome </h2>
14      <embed src="../movies/History_of_Rome.mp4" type="video/mp4" width="640" height="360" autostart="false"
/>
15      <noembed>
16       <a href="../transcripts/transcript_history_rome.htm">Transcript of "The History of Rome"</a>
17      </noembed>
18     </div>
19     <div>
20      <h2>The Battle of Thermopylae</h2>
21       <embed src="../movies/Battle_of_Thermopylae.mp4" type="video/mp4" width="640" height="360" auto-
start="false">
22       <noembed>
23        <a href="../transcripts/transcript_battle_thermopylae.htm">Transcript of "The Battle of Thermopylae"</a>
24       </noembed>
25      </embed>
26     </div>
27     <div>
28      <h2>Life of Genghis Khan</h2>
29      <embed src="../movies/Life_of_Genghis_Khan.mp4" type="video/mp4" width="640" height="360" auto-
start="false">
30      </embed>
31     </div>
32    </article>
33   </section>
34  </body>
34 </html>
```

be named "H46-1" and the second "H46-2" (as indicated by the values of the "iderr" parameter in Table 4).

6.2.1 Validation of H46-1

The ConditionsEvaluator verifies that the H46-1 branch expresses just one requirement (i.e., a single condition), so the execution passes to ConditionValidator: It verifies that the type of the condition is "Check," so redirects the execution to CheckOperator.

CheckOperator inspects the nodes of the DOM relative to the tag <embed>, checking whether any of them does not fulfill the requirement expressed by the secondary condition "followed by a <noembed> tag" since HTML code line 21 and line 29 do not fulfill the requirement (for line 21 the <noembed> tag is a child of <embed>, though

does not follow it, for line 29 there is not any <noembed> tag following an <embed> tag).

The list of information about these two errors is returned to ConditionValidator and then to ConditionsValidator.

6.2.2 Validation of H46-2

The ConditionsEvaluator verifies that the H46-2 branch expresses just one requirement (i.e., a single condition), so the execution passes to ConditionValidator: It verifies that the type of the condition is "Check," redirecting the execution to CheckOperator.

The CheckOperator inspects the DOM nodes relative to the tag <embed>, checking whether any of them does not fulfill the requirement expressed by the secondary condition "it has as child a <noembed> tag": In the HTML code, line 14 and line 29 do not fulfill the requirement (for line 14 the <noembed> tag follow it, but it is not child of the <embed> tag, for line 29 there is not any <noembed> tag).

The list of information about these two errors is returned to ConditionValidator and then to ConditionsValidator.

ConditionsValidator receives from the two branches of the recursion the two lists of errors found:

First branch: lines 21 and 29
Second branch: lines 14 and 29

In this expression, the two branches are connected by the Boolean binary operator "OR": This implies that an element of the page in order to not satisfy the entire expression must not satisfy both branches. Dually, this means that an error, to be an error for the entire expression, must be an error for both its branches: In this example, the line 29 is an error for both the branches, so it is an error for the whole condition expression.

The error list, consisting of a single element, is sent back to CpValidator and then to ChecksValidator: Because there is only one Check, there is no need to reassemble any recursive processing, and the error list can be sent back to the StartModule where the result is stored.

6.3 Validation of dynamic Web pages

During the last years, one of the most important trends in Web site development has been the emergence of increasingly interactive dynamic Web pages. The reasons for this trend are varied: One is the desire to create more graphically attractive Web sites (especially for commercial ones), another is the desire to create Web sites customizable to the level of the individual user, to provide information and services tailored to user's preferences (feature used especially by online newspapers, communities and social networks), and finally, the possibility of transferring part of the calculations, necessary for the provision of Web applications, from server to client, taking advantage of the increasing power of personal computers and mobile devices, and reducing costs for companies that provide online services.

From the technological point of view, the main technique to provide dynamic functionality within Web pages are scripting languages, along with the use of asynchronous communication mechanisms obtained through Ajax. After an initial phase, in which the most widely used language was Action Script (the scripting language of Adobe Flash), today the language used most often is JavaScript [31]. The success of such language is due to the possibility of being directly executed by all major browsers without the need for additional plugins, its ability to interact with the Web page DOM and the availability of numerous frameworks that greatly simplify its use. Although some of the new features of HTML5 and CSS3 will replace the use of scripts to performing certain operations (e.g., animations and checks of form's values), it seems no other technologies will replace JavaScript in the short term.

Regarding Web page accessibility, the creation of dynamic pages is problematic on the effectiveness of the validation performed by automated tools: In fact, usually such a validation is performed by inspecting the static source code of the page (HTML and CSS) and performing the required analysis on it.

In the case of dynamic pages, the source code of the page (and thus its static description) can significantly diverge with respect to the characteristics of the page actually displayed by the browser (its dynamic description) as a result of the changes arising from the execution of scripts. It is possible to create a perfectly accessible static page and modify it via some script until it becomes totally inaccessible.

To address this problem, there are three possible solutions:

- Inclusion in the validator of a JavaScript engine to run scripts
- Static analysis to determine the semantics of the script's code
- Validation based on the analysis of the page actually displayed (i.e., on the current representation of the DOM within the browser).

The first solution has been used by some of the tools cited in Sect. 2 (for example [21]), though it has some shortcomings: Even though including the JavaScript engine, changes to the DOM are restricted to those carried out by script automatically executed after page loading (onload event). The changes to the DOM, resulting from interactions with the user or resulting from communications with some servers, would not be detected.

The second solution can present problems: JavaScript is a weakly typed language with no type declarations only supporting run-time checking of calls and field access: It has higher-order functions and closures, exceptions, extensive type coercion rules, and a flexible object model where methods and fields can be added or change types and inheritance relations can be modified during execution. In recent years, many researchers have conducted studies about the static analysis of JavaScript code, mainly based on assumptions that, as shown by Richards et al. [32], are actually often being violated by the code written by programmers: JavaScript is described by Richards as "a harsh terrain for static analysis."

To solve the problem of validation of dynamic pages, it was decided to follow the third solution, i.e., to delegate the browser to load the page (scripts included), generate its DOM and execute the scripts (eventually) modifying the DOM.

In Sect. 2, some tools that used a similar approach to performing validation of dynamic pages are cited; however, each of them has shortcomings: For example, Chen's tool [23] focuses just on widgets, a small subset of all possible components of a Web page, Fuertes' tool [22] performs the validation just after the DOM generation ignoring the Web page changes due to user interaction or update via asynchronous communication, and Fernandes' tool [24] simulates interactions on clickable elements to check issues related to this type of element, ignoring part of the possible changes that may occur within a dynamic page.

In the tool presented in this paper, the dynamic page validation is performed extracting the modified DOM from the browser and sending it to the same servlet used to implement the Web interface as described in Sect. 4.

The DOM extraction is performed through a set of extensions for the most popular browsers (at the time of writing, the extension for the Mozilla Firefox browser was already available, whereas the extension for Google Chrome was under development). After receiving the DOM, the servlet forwards the data to the validator. Instead of retrieving the Web page, the validator uses directly the received DOM as page representation in the validation process and returns the result to the extension, which will then show it in a new browser window.

As shown in Fig. 5, the validation of the Web page performed using the Mauve Extension is triggered through a context menu that can be displayed by right-clicking on any point of the Firefox browser window.

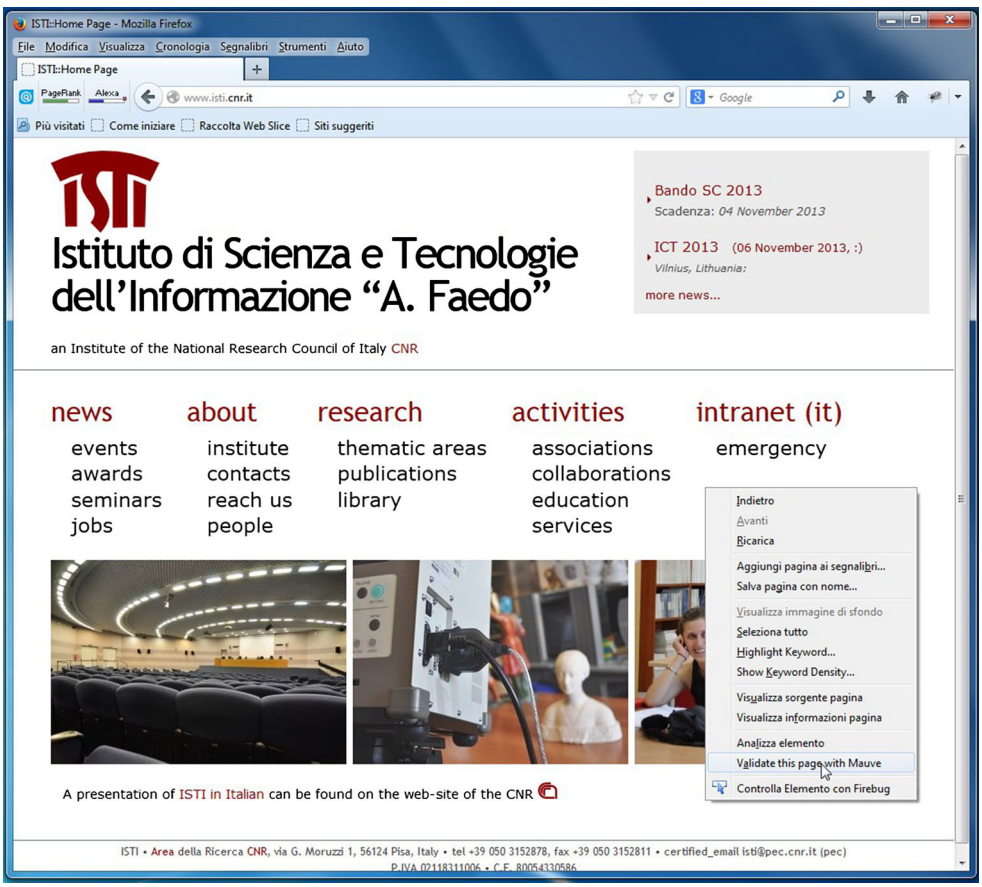

Fig. 5 Contextual menu activating MAUVE extension for Firefox

The use of a browser extension has many advantages and almost no shortcomings: This solution clearly applies validation to a "snapshot" of the state of the DOM of the page and also allows an accessibility expert to reproduce the interaction of a generic user with the site, checking all the various steps necessary for the completion of such interaction, while avoiding all the shortcomings of similar tools cited above.

Furthermore, regarding sites with continuous animations, it is easy to assume that an expert may be able to identify the most critical "frames" from the standpoint of accessibility and verify them in a simple and immediate way.

The use of browser extensions has benefits also with regards to the user experience and development points of view. In fact, browser extensions are well-known software components, especially by the Web designer and Web developer community, who are the target users of the software.

Furthermore, almost all browsers developers provide detailed documentation about the development of extensions, ensuring, for each new browser release, backward compatibility for extensions previously developed.

7 A first test of the validation environment

7.1 Validation test design

In order to carry out a first, non-exhaustive, validation test of the proposed environment, it was decided to select a set of Web sites of twelve world-renowned commercial brands, each related to different business categories, including:

- E-commerce sites
- Transportation companies
- Travel portals
- Utility companies (oil and gas, telecommunications, delivery services)
- Fashion
- Information Technology

For each brand, the international Web site was selected: If this was not present or was not relevant, the corresponding US site was selected. For each of these Web sites, four pages were selected, each belonging to one different page category in terms of structure:

- Homepage
- Page with forms
- Page with general contents
- Page with tables

These Web pages were validated against WCAG 2.0, supported by both validators: The validation was also performed with respect to compliance level AA, this being the level recommended by current legislation in many countries.

In order to have a comparison element for the tool's results, it was also decided to compare them with the results of another accessibility validator. For this purpose, the validators presenting the same deterministic approach used by MAUVE were considered. Furthermore, to make the comparison quicker and easier, it was decided to further restrict the choice to the validators that, as MAUVE, presented the results of the validation with a "code-oriented" approach.

With these restrictions, among all validators currently available, TotalValidator [33] was selected, a commercial software developed by the homonymous British firm, which has available for free a working version with some limitations (validation limited to only one page at a time, reports customization disabled, validation of password protected areas disabled). This tool is available as desktop software, it performs accessibility validation against WCAG 1.0, WCAG 2.0 and Section 508, and claims to be compatible with HTML5 and CSS3. This tool has been considered in numerous scientific publications, such as [34] and [35]. To carry out such a comparison, a methodology similar to that proposed by Brajnik [36] was followed, in particular referring to the issue classification and correctness test.

Each selected page was validated through the two tools and manually inspected by a human inspector for both tools to determine which issue can be categorized as a false positive or false negative. The following criteria were observed:

- *False positive* (FP): An issue generated by a tool was classified as a FP for either tool if, upon investigation of the HTML page the issue refers to, a human inspector would consider the issue irrelevant or wrong. For example, an issue such as "Nest headings properly" was classified as a FP on a page where no heading tags (i.e., H1, H2,…,H6) are used or where headings were used properly. A FP would also include those cases where the issue is plainly wrong: "Avoid using structural markup for visual formatting" on a page that correctly uses TH for marking up data table headers. If both tools generate issues that refer to the same accessibility problem (i.e., the issues refer to the same features in the Web page), then, if they are found to be false positives, each of the issues is labeled as FP twice, i.e., once for each tool.
- *False negative* (FN): An issue X generated by tool A is used as a reference for tool B; if B does not generate an issue that is equivalent to X, then X is classified as a FN for tool B (B missed it). For example, an issue such as

"Examine all CSS properties that define font size for each rule set and check that the value is a percentage" correctly detected from tool A (the style sheet linked by the page uses px, em or measurement units to define font-style property) was classified as a FN for tool B if tool B does not apply this test or if it does not detect the issue on the page under validation. Obviously, the converse is also valid (an issue properly detected by the tool B and not reported by the tool A is classified as FN for tool A).

- *OK*: otherwise. An issue generated by tool A was marked as OK for A if it is not an FP; it was marked OK for tool B if it was neither an FN nor an FP. If tool A generates a very specific issue X (such as "Spacer image has no valid ALT") and tool B generates a more general issue that addresses the same problem (such as "Provide ALT to images"), then X is not classified as a FN, but as an OK issue for tool B (since B catches it, even though at a more general level). As a result of the above, each issue is classified twice: Once (as OK or FP) with respect to the tool that generated it, and once (as OK, FN or FP) with respect to the other tool.

Notice that the generation issue has been considered regardless of which normative references are indicated by the two tools: For example, in the case of an issue such as "Nest headings properly," TotalValidator indicates as reference the violation of the WCAG 2.0 Failure of Success Criterion F43, while MAUVE refers to the violation of the WCAG 2.0 Technique H42. In this case, considerate is assumed that both tools indicate the same issue, so, according to the HTML code investigation, it is marked OK or FP for both tools.

Furthermore, in some cases, it has been noticed that an issue was reported as a warning by one tool and as an error by the other and vice versa: Also in this case, the severity level indicated by the tools has been ignored, considering the same issue for both tools.

All testing was performed during May 2013 using the latest available version of the tools (TotalValidator 8.0.0 and MAUVE v. 1.0.3 Beta). Table 6 shows the resulting numbers.

An initial analysis of the data collected seems to indicate a good capacity of analysis and precision by MAUVE regarding the accessibility of style sheets: This is manifested both directly, through the formalization of a greater number of checkpoint regarding requirements for CSS properties with respect to those analyzed by TotalValidator (for instance the Technique C12, regarding the measurement unit used defining the font-style values), and indirectly, through the application of some checkpoint also to elements of the page made invisible through CSS properties and that, as such, should not be subject to such rules.

Table 6 TotalValidator–MAUVE comparison

	TV FP	TV FN	MA FP	MA FN
Homepage	37	429	21	11
Page with forms	71	129	17	5
General content	23	186	41	7
Page with tables	21	821	32	9

For TotalValidator, this generates both false negatives (CSS-related issues are not detected) and false positives (incorrect application of guidelines).

Furthermore, TotalValidator seems to be less precise on the analysis of the accessibility of the tables, ignoring or not properly verifying certain requirements of the guideline used, such as "provide an summary attribute for the table element to give an overview of data" (Technique H73), or "use the scope attribute to associate header cells and data cells in data tables" (Technique H63).

On the other hand, MAUVE still presents some small imperfections in the validation: For example, in case of image links, the alternative text of the image is not recognized as a sufficient element to describe the purpose of the link (Technique H30), thus generating false positives.

7.2 Comparison between static and dynamic validation

Following the first draft of this paper, the authors decided to make a comparison of the effectiveness of validation conducted by static analysis of source code (i.e., via the Web interface) and validation conducted using the browser extension.

Since some months had passed from the test described in the previous subsection, and many of the selected Web sites had undergone changes of varying degrees and types, the data previously collected were not reusable.

To this end, from the previous set of Web pages, a subset of 15 Web pages was randomly extracted, which was then validated again using the Web interface and the browser extension.

In about half of the analyzed pages, the scripts did not change the HTML code of the page, or changed the code but did not generate any new accessibility issues: In this case, the same number of the accessibility issues was detected by both the Web interface and the browser extension.

In the remaining part of the subset, the scripts dynamically modified the HTML code of the page, generating new accessibility issues: The browser extension detected up to about 30 % more accessibility errors compared to the validation performed by the Web interface. On average, the browser extension detected about 12.6 % more accessibility errors than the validation performed using the Web interface.

8 Conclusions

The increasing number of design guidelines proposed for the Web, in particular for accessibility evaluation, makes the implementation of automatic tools for managing guidelines increasingly complex.

In order to develop a validation tool independent of guidelines definition, the guidelines should be specified separately and interpreted at run-time.

To this end, this paper reports on the LWGD, an XML-based abstract language for defining guidelines. A validator also had been developed, based on this formalism and able to manage, load and check any guideline defined through this language without requiring modifications to its implementation.

The tool is also able to perform the guidelines validation with respect to recent Web standards (such as HTML5 and CSS 3) and supports the selection of the version of a Web site specific for certain types of devices (such as tablets, smartphones, consoles and Smart TVs).

The analysis environment has been enriched by providing a solution to the problem of accessibility evaluation of dynamic Web pages, whose structure is made different from its static description by the execution of one or more JavaScript scripts. The tool was tested through the validation of several Web sites, and its results were compared with respect to another accessibility validator. At present, MAUVE is the only validator publicly available based on an abstract language for the definition of guidelines and able to validate compliance with WCAG 2.0. The presented approach can also be applied for checking other types of Web guidelines (e.g., usability guidelines).

Among the possible future developments, there is the possibility of extending the validation to other technologies considered in the most recent guidelines (e.g., PDF, Flash and Silverlight) and improving the effectiveness of the validation result presentation. Further possible developments include the use of MAUVE, for example, supporting its use on mobile devices by creating specific mobile applications.

Appendix

See Table 7.

Table 7 Set of Web sites used for validation

Airfrance	
Homepage	http://www.airfrance.us/cgi-bin/AF/US/en/common/home/flights/ticket-plane.do
Form	http://www.airfrance.us/cgi-bin/AF/US/en/local/process/standardbooking/BookNewTripAction.do
Contents	http://www.airfrance.us/US/en/common/guidevoyageur/reseau/reseau_af.htm
Table	http://www.airfrance.us/cgi-bin/AF/US/en/local/resainfovol/horaires/horaires.do
RyanAir	
Homepage	http://www.ryanair.com
Form	https://www.bookryanair.com/SkySales/Booking.aspx?culture=it-it&lc=it-it#Security
Contents	http://www.ryanair.com/it/about
Table	http://www.ryanair.com/en/questions/contacting-customer-service
Ebay	
Homepage	http://www.ebay.it/
Form	http://www.ebay.it/sch/ebayadvsearch/
Contents	http://pages.ebay.it/areaprofessionale/index.html
Table	http://pages.ebay.it/charity/
Kelkoo	
Homepage	http://www.kelkoo.co.uk/
Form	https://secure.kelkoo.co.uk/createAccount.html
Content	http://www.kelkoo.co.uk/co_4292-online-merchants-and-stores-partner-with-kelkoo.html
Table	http://www.kelkoo.co.uk/co_15138-idis-accreditation.html
Intel	
Homepage	http://www.intel.com
Form	http://downloadcenter.intel.com/?lang=ita&changeLang=true
Contents	http://newsroom.intel.com/community/it_it
Table	http://ark.intel.com/products/family/59136/2nd-Generation-Intel-Core-i7-Processors/desktop
Gazprom	
Homepage	http://www.gazprom.com/

Table 7 continued

Form	http://www.gazprom.com/investors/calc/
Contents	http://www.gazprom.com/press/gallery/extraction/
Table	www.gazprom.com/nature/environmental-protection
Shell	
Homepage	http://www.shell.com/
Form	http://www.shell.com/global/aboutshell/careers/professionals/app-xp-find-a-job.html
Contents	http://www.shell.com/global/future-energy.html
Table	http://reports.shell.com/annual-report/2011/servicepages/filelibrary/files/collection.php?cat=b
Fedex	
Homepage	http://www.fedex.com/us/
Form	https://www.fedex.com/ratefinder/home?cc=US&language=en&locId=express
Contents	http://www.fedex.com/us/customersupport/?cc=us
Table	https://www.fedex.com/myprofile/loginandcontact/?locale=en_US&cntry_code=us
Vodafone	
Homepage	http://www.vodafone.com/
Form	http://www.vodafone.com/content/index/about/about_us/privacy.html
Contents	http://www.vodafone.com/content/index/investors/shareholders/ordinary_shareholders.html
Table	http://www.vodafone.com/content/index/about/about_us/where.html
H&M	
Homepage	https://www.hm.com/us/
Form	https://www.hm.com/us/newsletter
Contents	https://www.hm.com/us/customer-service/gift-cards
Table	http://www.hm.com/us/sizeguide?show=sizeguide_ladies
MSC Cruises	
Homepage	http://www.msccruisesusa.com
Form	http://www.msccruisesusa.com/us_en/MyMSC/Registration-Form.aspx
Contents	http://www.msccruisesusa.com/us_en/About-MSC-Cruises/Overview.aspx
Table	http://www.msccruisesusa.com/us_en/an-msc-cruise/Faq.aspx
TripAdvisor	
Homepage	http://www.tripadvisor.com/
Form	http://www.tripadvisor.com/Flights
Contents	http://www.tripadvisor.com/TravelersChoice
Table	http://www.tripadvisor.com/pages/by_city.html

References

1. European Union: eEurope 2002 Action Plan. http://europa.eu/legislation_summaries/information_society/strategies/l24226a_en.htm
2. Bundesministerium der Justiz: Barrierefreie-Informationstechnik-Verordnung (DE). http://www.gesetze-im-internet.de/bitv_2_0/BJNR184300011.html
3. Centro Nazionale per l'Informatica nella Pubblica Amministrazione: Stanca Act. http://www.pubbliaccesso.gov.it/normative/law_20040109_n4.htm
4. Boletín Oficial del Estado: Reglamento sobre las condiciones básicas para el acceso de las personas con discapacidad a las tecnologías, productos y servicios relacionados con la sociedad de la información y medios de comunicación social (ES). http://www.boe.es/buscar/doc.php?id=BOE-A-2007-19968
5. Ministère du Budget, des comptes publics et de la fonction publique: Référentiel Général d'Accessibilité pour les Administrations (FR). http://references.modernisation.gouv.fr/rgaa-accessibilite
6. United Nations: Factsheet on Persons with Disabilities. http://www.un.org/disabilities/default.asp?id=18
7. U.S. Census Bureau: Americans with Disabilities: 2010. http://www.census.gov/prod/2012pubs/p70-131.pdf
8. Communication department of the European Commission: European Day for People with Disabilities. http://europa.eu/rapid/press-release_IP-12-1296_en.htm
9. W3C: Web Content Accessibility Guidelines 1.0. http://www.w3.org/TR/WCAG10/
10. W3C: Web Content Accessibility Guidelines 2.0. http://www.w3.org/TR/WCAG/
11. International Organization for Standardization: ISO/IEC 40500:2012. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=58625
12. Gay, G., Li, CQ.: AChecker: Open, Interactive, Customizable, Web Accessibility Checking. In: International Cross-Disciplinary Conference on Web Accessibility—W4A 2010, pp. 1–2 (2010)
13. Test Accesibilidad Web. http://www.tawdis.net/

14. Fernandes, N., Kaklanis, N., Votis, K., Tzovaras, D., Carriço1, L.: An Analysis of Personalized Web Accessibility. In: Proceedings of the 11th Web for All Conference, Article No. 19, ACM, New York, NY, USA (2014)
15. Beirekdar, A., Vanderdonckt, J., Noirhomme-Fraiture: A Framework and a Language for Usability Automatic Evaluation of Web Sites by Static Analysis of HTML Source Code. In: Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, pp. 337–348 (2002)
16. Beirekdar, A., Vanderdonckt, J., Noirhomme-Fraiture, M.: Kwaresmi—knowledge-based Web automated evaluation with REconfigurable guidelineS optimization. In: Forbrig, P., Limbourg, Q., Urban, B., Vanderdonckt, J. (eds.) DSV-IS 2002. LNCS, vol. 2545, pp. 362–376. Springer, Heidelberg (2002)
17. Takata, Y., Nakamura, T., Seki, H.: Accessibility Verification of WWW Documents by an Automatic Guideline Verification Tool. In: Proceedings of the 37th Hawaii International Conference on System Sciences (2004)
18. Arrue, M., Vigo, M., Abascal, J.: Including Heterogeneous Web Accessibility Guidelines in the Development Process, Engineering Interactive Systems 2008, LNCS, vol. 4940, pp. 620–637. Springer, Heidelberg (2008)
19. Aizpurua, A., Arrue, M., Vigo, M., Abascal, J.: Exploring Automatic CSS Accessibility Evaluation. In: Proceedings of 9th International Conference, ICWE 2009, pp. 16–29 (2009)
20. Leporini, B., Paternò, F., Scorcia, A.: Flexible tool support for accessibility evaluation. Interact. Comput. **18**(5), 869–890 (2006)
21. Brown, A., Jay, C., Chen, A.Q., Harper, S.: The uptake of Web 2.0 technologies, and its impact on visually disabled users. J. Univers. Access Inf. Soc. **11**, 185–199 (2012)
22. Fuertes, J.L., González, R., Gutiérrez, E., Martínez, L.: Hera-FFX: a Firefox add-on for semi-automatic web accessibility evaluation. In: W4A '09 Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibility (W4A) pp. 26–35 (2009)
23. Chen, A.Q., Harper, S., Lunn, D., Brown, A.: Widget identification: a high-level approach to accessibility. World Wide Web **16**, 73–89 (2013)
24. Fernandes, N., Costa, D., Neves, S., Duarte, C., Carriço, L.: Evaluating the accessibility of rich internet applications, W4A '12 Proceedings of the International Cross-Disciplinary Conference on Web Accessibility (2012)
25. WAVE. http://wave.webaim.org/
26. Unicorn-W3C's Unified Validator. http://validator.w3.org/unicorn/
27. Jsoup: Java HTML Parser. http://jsoup.org/
28. The Validator.nu HTML Parser. http://about.validator.nu/htmlparser/
29. jStyleParser Css Parser. http://cssbox.sourceforge.net/jstyleparser/index.php
30. Leporini, B., Paternò, F.: Applying web usability criteria for vision-impaired users: does it really improve task performance? Int. J. Hum. Comput. Interact. **24**(1), 17–47 (2008)
31. The Transparent Language Popularity Index. http://lang-index.sourceforge.net/
32. Richards G., Lebresne S., Burg B., Vitek J.: An Analysis of the Dynamic Behavior of JavaScript Programs. In: PLDI '10 Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM, pp. 1–12 (2010)
33. TotalValidator. http://www.totalvalidator.com/
34. Leitner, M., Strauss, C.: Organizational Motivations for Web Accessibility Implementation—A Case Study. Computers Helping People with Special Needs, Lecture Notes in Computer Science, vol. 6179, pp. 392–399 (2010)
35. Reis A., Barroso J., Gonçalves R.: Supporting Accessibility in Higher Education Information Systems. Universal Access in Human-Computer Interaction. Applications and Services for Quality of Life, Lecture Notes in Computer Science, vol. 8011, pp. 250–255 (2013)
36. Brajnik, G.: Comparing accessibility evaluation tools: a method for tool effectiveness. Univers. Access Inf. Soc. **3**(3–4), 252–263 (2004)