Flexible Tool Support for Accessibility Evaluation

Barbara Leporini, Fabio Paternò, Antonio Scorcia
ISTI-CNR
56124 Pisa, Italy
{barbara.leporini, fabio.paterno, antonio.scorcia}@isti.cnr.it

ABSTRACT

The increasing need to check Web site accessibility has stimulated interest in tools to aid the various activities involved. While some tools for this purpose already exist, we believe that there is a demand for making their support more flexible. In particular, there is often a need for validation of multiple sets of guidelines, repairing Web pages and providing better reports for the evaluators. In this paper, we discuss such issues and how we have addressed them in the design of MAGENTA, our new tool for supporting inspection-based evaluation of accessibility and usability guidelines.

1. INTRODUCTION

The importance of tool support in accessibility evaluation of Web sites is gaining increasing acceptance because many countries have adopted legislation that imposes some level of compliance to accessibility guidelines. The goal of tool support is not to completely replace human evaluators and designers, but to help them to manage the complexity of many existing Web sites, apply evaluation criteria in a consistent manner, and thereby make their work more efficient. Some tools for this purpose already exist (such as Bobby [4], LIFT [22], A-Prompt [2]), but the increasing request for support poses new issues that deserve better answers. In particular, we have identified three important areas for accessibility tools: support for checking multiple sets of guidelines, code correction and reporting.

The issue of multiple guidelines involves many factors. One is that there are various organizations that issue standard guidelines, some are international (such as the W3C/WAI [25],[26] or the ISO IS 9241-171 [8] and TS 16071 [9]), others are national standards (such as the section 508 [19]). Often they share the same body of knowledge but also have slight differences, which designers have to consider in some contexts. Another reason is that, while accessibility guidelines usually provide an important step forward to make sites accessible to everyone, included the disabled, often their mere application does not provide usable results for some classes of users. Indeed, being able to access information is not enough. In fact, although a service may be accessible to certain types of users (such as the blind), it still may not be sufficiently usable to such users. While accessibility and usability are closely related to each other they address slightly different issues. Accessibility aims to increase the number of users who can access a system, this means to remove any potential technical barrier that does not allow the user to access the information. Usability aims to make the user interaction more effective, efficient and satisfactory. Thus, it is possible to have systems that are usable but not accessible, which means that some users cannot access the information, but those who can access it are supported in such a way as to find it easily. It is also possible to have systems accessible but not usable, which means that all users can access the desired information, but this can only occur after long and tedious interactions. Consequently, there is also a need for integrating these two aspects in order to obtain interactive services for a wide variety of users, including people who are disabled. To this end, for a specific class of users (vision-impaired users), we have identified a set of guidelines [12] aiming at supporting accessibility and usability when Web sites are accessed by users through screen readers.

Guidelines are intended for developers and designers: they are general principles that can be followed to improve the application accessibility and usability. We mainly focus on guidelines for Web applications because the Web is currently the most common user interface environment. Each guideline can include one or more checkpoints. Checkpoints are technical solutions that support the application/evaluation of the criteria and usually correspond to specific implementation constructs that guarantee the satisfaction of the associated guideline. For example, the guideline “Logical partition of interface elements” expresses the concept of well-structuring and organizing the page content. So, it provides a general principle that should be taken into account by developers during the Web site

design. Then, developers can decide how to apply this criterion. Usually, several solutions can be adopted. For example, the Web page content could be structured by using frames, or blocks customisable by CSS properties. Alternatively, the content within the page could be visualised by embedding it in the layout or data tables. Moreover, long page content could be partitioned through heading levels, paragraphs, or specific page parts could marked with “hidden labels”. So, all these cases apply the same general guideline (i.e. partitioning the content), but use different technical solutions.

In general, making a Web site more accessible and usable requires considerable effort by developers in handling Web page code and many specific design guidelines: they have to decide which principles to use for the specific case, how to apply them, and when. Evaluating Web pages requires a lot of effort as well. In this perspective, MAGENTA (Multi-Analysis of Guidelines by an ENhanced Tool for Accessibility) has been developed to assist developers in handling Web pages and guidelines. An early version of this tool (NAUTICUS) [5] supported our guidelines for visually impaired users alone. Next, we decided to extend its architecture in order to support several kinds of guidelines. In addition, other functionalities have been added to better address two important issues: repairing and reporting.

In the following sections, after discussing related work and introducing the design criteria for our tool, including a language for guidelines description, we discuss the tool developed to support designers and developers. Example tool applications are described together with a short report on its use. Lastly, some concluding remarks are provided along with indications for future work.

2. RELATED WORK

Various international projects have addressed accessibility and usability of user interfaces for people with special needs. Stephanidis' group has long been working on user interfaces for all, elaborating methods and tools allowing the development of unified user interfaces [21]. In the AVANTI project, a "Unified Web Browser" has been developed: it employs adaptability and adaptivity techniques, in order to provide accessibility and high-quality interaction to users with different abilities and needs (e.g., blind users or those with other disabilities). In particular, for vision-impaired people, it incorporates techniques for the generation of a list of large push buttons containing the links of a page. However, apart from this feature, the AVANTI browser focuses on accessibility issues, but does not specifically support Web site evaluation.

In order to assure a certain level of accessibility and/or usability of user interfaces (UIs), several design criteria and guidelines have been proposed in literature. Several detailed user interface guidelines were formulated for general user interfaces (for example, [15], [20] and [23]). In [13] a state of the art review of the major sources for usability guidelines for the Web is presented, along with an overview of ongoing efforts in Web usability guidelines, whereas [17] reports on a possible set of inclusive guidelines.

The analysis of Web site accessibility and usability by means of guidelines, similarly to other inspection methods used in usability/accessibility assessment, requires observing, analysing and interpreting the Web site characteristics. Since these activities require high costs in terms of time and effort, there is a great interest in developing tools that automate them in various phases. Ivory and Hearts [7] distinguish between automatic capture, analysis and critique tools. However, even the use of automatic tools has problems, such as the length and detailed nature of reports that make them difficult to interpret, accessibility guidelines require developers to fully understand their requirements, and often they still need some manual inspection [14]. Besides, another issue to be considered when an automatic support is used is the "repair process". In fact, even if accessibility and usability problems are detected automatically, repairing them can require a lot of effort. Although, a completely automatic repair process is not easy to implement, a semi-automatic support for this important phase is advisable. However, several common evaluation tools do not provide any support for repair functionality, and when some tools provide some support then they require working with the underlying HTML code, which can be tedious and difficult to do because of the may low-level details to handle.

Many automatic analysis tools were developed to assist evaluators by automatically detecting and reporting guideline violations and in some cases making suggestions for fixing them. EvalIris [1] is an example of tool that provides designers and evaluators with a good support to easily incorporate new additional accessibility guidelines. This is obtained through a language for guidelines represented by an XML schema, which has been used to specify WAI guidelines. The result report of the single page evaluation is provided through a representation defined by XML as well. The tool proposed herein aims at addressing also other ways to support designers, such as providing interactive support for correcting the Web pages violating the guidelines considered. Another contribution in this

area is DESTINE [3], which supports W3C and 508 guidelines specified through another language for guidelines and provides reports that include statistics at different levels (site, page, guideline). However, even this tool does not address the possibility of supporting repairing of Web pages. A different approach has been considered by Lee and Hanson [11]. They aim at creating a tool able to automatically change the structure of a Web page in order to make it more accessible. However, it is difficult to find rules that generally provide good results in this solution, which does not provide explanations to the Web designers when the modifications are made. A-Prompt [2] is another tool that identifies potential accessibility problems and provides support to correct them. The tool works with local Web pages. Although it allows developers to check several Web pages at a time, it handles pages sequentially and does not store evaluation reports. In this way, if developers want to examine again the result of a certain guideline application then the evaluation process must be carried out once more. Our tool addresses this issue allowing developers to visualize the reports without performing again the evaluation process because the reports are stored in external XML files. Imergo [16] is an another interesting contribution in the area of tools for accessibility, which aims to provide integrated support for Content Management Systems and engineered support for multiple guidelines. In our tool we provide a different solution to such issue based on the definition of an abstract language for guidelines and we also want to provide support for automatic repair.

In general, our review of the state of art highlights that there is a lack of tools able to provide integrated flexible support in different respects (multiple guidelines, efficient reports, repairing). When support for guidelines has been proposed then it has been applied only for the most known guidelines (W3C/WAI and 508), while our tool has already been applied to support three sets of different guidelines with different goals (W3C/WAI, guidelines for usability and accessibility for vision-impaired users [12], those requested by the Italian law for accessible Web sites in public services [10]).

3. GUIDELINE ABSTRACTION

In the early version of our tool, only one set of guidelines was supported for automatic inspection. The specification of the guidelines to check was implemented directly in the tool code. Such an approach has several drawbacks: firstly, the tool is designed to support just one type of guideline-based evaluation; secondly, if a guideline changes or if a new guideline has to be added or deleted, the tool must be modified at the implementation level. This makes the tool limited and difficult to maintain and update. Thus, in order to overcome such limitations we have designed a solution based on the definition of an abstract guideline language.

3.1 Overview

Our solution is based on the definition of a language for specifying guidelines that are stored externally to the tool, and then we have designed and implemented a tool that is able to interpret such externally specified guidelines. In general, a guideline, is defined as a rule or principle that provides guidance to appropriate behaviour. Generally speaking, a guideline is expressed in natural language, which automatic tools cannot handle. Hence, in order to support developers in using guidelines, our approach is aimed at expressing guidelines so that they can be dealt with automatically. Furthermore, our investigation has focused on developing a tool free from guideline definition constraints. In practice, the solution consists of abstracting guideline statements in a well-defined language. For example, a statement such as “all images presenting in a Web page must have an alternative description” or “Tables should have a short summary description” have to be formalised in some manner that can be handled by the automatic tool. Thus, we want the guideline statement to be abstracted and appropriately specified. This implies that the guideline is previously structured and then specified in our XML-based language. Briefly, such an approach allows designers to:

- Easily add new guidelines, modify or delete existing ones;
- Define and use different sets of guidelines;
- Separate implementation checks from the definition of guidelines;
- Avoid repetitive recoding of the tool by the implementers.

This makes the tool more adaptable to different kinds of automatic guideline-based inspections. In addition, guidelines can be better handled by persons different from those who have implemented the tool.

3.2 Abstraction

In defining a guideline abstraction, the main features and elements characterizing the guideline itself can be shortly summarised as:

- *General features*, such as name, identification, description, source, etc.;
- *Objects* involved in the checking process, i.e. which tags, attributes, properties have to be considered in the inspection;
- *Conditions* referred to objects (i.e. what type of check has to be performed).

All the above elements have to be considered for structuring a guideline. It is necessary to specify the general features, the tags or properties involved in the checking process and the conditions that have to be verified for detecting if the principle has been applied or not. So, in order to define a certain abstraction level we have defined guidelines in terms of:

- *Criteria*, which express the general design principles of the guidelines and their features, such as description, type, etc.;
- *Checkpoint/s*, which indicates what has to be technically verified in the static code (i.e. which tag or attributes must be checked);
- *Checkpoint relations*, which indicate the Boolean conditions to consider in order to satisfy the guidelines (i.e. and|or); a criterion to be satisfied could require one or more checkpoints, so relations among the checkpoints should be specified.

In brief, each guideline expresses a principle to be complied with by applying one or more conditions at checkpoints. All this information and these conditions have to be structured by a specific guideline abstraction.

3.3 Example guideline abstraction

In order to explain how a guideline can be structured and specified, we discuss some examples:

- Number of links, a guideline composed of one checkpoint;
- Frame definition, a guideline composed of one checkpoint with two conditions;
- Proper link content, a guideline with two checkpoints.
- Contrast colour verification, a guideline with an external tool execution
- Javascript and links, a guideline to avoid Javascript usage with links

In practice, we have to take the guideline statement, identify its main components, elements and conditions, so that objects, actions and conditions can then be checked.

Example 1: Number of links

Let us consider a guideline related to the number of links that a single Web page can contain. Suppose the guideline statement is “The number of links in the page should be less than 20”. This statement expressed in natural language has to be abstracted, structured and lastly specified in terms of criteria and checkpoints. In practise, the objects, actions and conditions have to be identified.

The guideline refers to the “page”; the tag involved is “<a>”; the operation to apply to the tag is “count” and the condition to verify is “<= 20”.

So, schematically we have the following data:

<a> with href, XHTML, (count<n)

In other words, the object having a specific feature (attribute), the code is XHTML, the operation “count” and lastly the condition “<20”. Note that we considered the elements <a> having the “href” attribute in order to not consider the local bookmarks within the page.

Example 2: Frame definition

The guideline statement regarding frames existing in a page could be something like ”All frames existing in the page should have both name and title attributes with the same value”.

Applying the same object and operation extraction used for the previous example, we obtain:

<frame>, XHTML, (Exist name with significant value & Exist title with significant value & name=title)

Thus, a single checkpoint is identified with three conditions.

The tag involved is <frame>, the code to consider is XHTML and the three conditions refer to the existence of the attributes and comparison of their values (which have to be equal).

Example 3: Proper link content

Another example is a guideline related to two types of checking: (1) textual and (2) graphical links. In this case its statement could be something like

”All links must have clear and significant content”.

In practise, checking involves both textual and graphical links, i.e. the criterion has two checkpoints. Thus, the extracted structure can be:

<a> with href, XHTML, (content not_belong not_proper_list_values)
and
<a> with href, XHTML, (for <img> content: Exist alt & (alt not_belong not_proper_list_values | alt+content not_belong not_proper_list_values))

As we can note, two checkpoints are necessary because the extracted objects and elements involved in the conditions to be verified are slightly different. The objects considered are tags <a> with “href” attributes (i.e. bookmarks are avoided); operations are verification of the existence of content or alt attribute for tag <img> used as link content; and lastly the conditions consist of checking that the values are different from a list of “non-appropriate” values.

Example 4: Contrast colour verification

In the case of contrast colour verification, an accessibility guideline suggests using a specific W3C algorithm, which combines background and foreground colour components. As major accessibility principles suggest separating rendering from content, we assume that another specific guideline checks rendering separation. In this case we check the information stored in the CSS style files, So, schematically we have the following guideline structure:

any with colour & background, CSS, (execute (ToolName(colour, background)))

In other words, the objects to check are the CSS properties “background:” and ”colour:”; the code is CSS; the action to perform in this case is an external algorithm which can be appropriately implemented. So, the operator is something like “Execute (ExternalToolName(parameters))”.

Example 5: Javascripts and links

Let us now consider the guideline “Do not use a javascript function as reference of a link”. If this guideline is not complied with, when javascripts support is not available in the browser, clicking on a link has no effect. So, it is important that a link has a URL address as destination rather than a javascript function. Such a guideline could be abstracted as follows:

<a> with href, HTML, (href!=empty & href not_contain “javascript:” & href not_contain “()”)

In practice, we check that all the links (i.e. tags <a>) have a non-empty href attribute, and that the attribute does not contain “javascript:” and/or “()” strings. The second construct indicates a function call (e.g. FunctName()).

3.4 Discussion

As described in the examples presented in the previous section, specifying a guideline in our language means identifying its main parts (objects, operations, and conditions). This type of work requires a good experience in handling page code and guidelines. In any case, it is important that it can be carried out even by someone other than the developers of the evaluation tool. As future work we plan to develop a graphical editor for assisting designers in abstracting and defining guidelines in our proposed XML-based language.

As shown in the examples discussed, in the guideline abstraction process some types of operations, values, and so on have to be checked. In short, the main elements are:

- *Description*, a natural language description of the guideline, with examples and pointers to further documentation;

- *Object types*, a set of possible types to associate with the objects (such as tag, attrib, page, etc.);
- *Properties*, to define characteristics, parameters, etc. (e.g. id, class, et.);
- *Operations*, a set of possible operations (such as Exist, Count, Check, Execute, etc.);
- *Conditions*, a set of conditions obtained by composing Boolean operators (such as ==, !=, <, etc.).

Table 1 shows how the guideline discussed in example 1 can be specified in an extended format. In addition to number of links, the number of frames is considered as well. So, the guideline has two checkpoints: one for checking the number of links, and the other for the frames. Errors and warnings are specified by the attribute “iderr” whose value is a number concatenated to the checkpoint id as well as the current condition. So, it is possible handling them appropriately by implementing externally the relative action to be performed (e.g. printing a message, pointing out an error or warning, providing a specific suggestion, and so on).

```
<gdl_set shortname="visually-impaired">
<name> Accessibility and usability criteria for visually-impaired users </name>
...
<guideline id="2" summary="Number of links and frames">
        <criterion type="usability" target="page">
        <description> The number of links being in a Web page should be not greater than 20. The number of
        frames should be between 2 and 5. </description>
                <checkpoints rel="and">
                <cp id="2.1" summary="Number of links" priority="2">
                <cp_descript> Checking of the number being within the page; only tag &lt;a&gt; with the href
                attribute are considered. </cp_descript>
                        <eval_object code="html" mandatory="yes">
                                <object type="tag"> a </object>
                                <select type="attrib"> href </select>
                        </eval_object>
                        <conditions rel="and">
                                <evaluate operator="count" cond="less" iderr="2.1.1">
                                        <el type="tag"> a </el>
                                        <el type="value"> 21 </el>
                                </evaluate>
                        </conditions>
                </cp>
                <cp id="2.2" summary="Number of frames" priority="1">
                <cp_descript> Checking that the number of the frames is in the range 2 - 5. </cp_descript>
                        <eval_object code="html" mandatory="no">
                                <object type="tag"> frame </object>
                                <select type="tag"> </select>
                        </eval_object>
                        <conditions rel="and">
                                <evaluate operator="count" cond="greater" iderr="2.2.1">
                                        <el type="tag"> frame </el>
```

```
                                        <el type="value"> 1 </el>
                                    </evaluate>
                                    <evaluate operator="count" cond="less" iderr="2.2.2">
                                        <el type="tag"> frame </el>
                                        <el type="value"> 6 </el>
                                    </evaluate>
                                </conditions>
                        </cp>
                        </checkpoints>
                </criterion>
</guideline>
<guideline id="3" summary="Specific sections">
                ...
```

Table 1 - Excerpt of Example Guideline Specification.

3.5 Guideline Abstraction Language

As described in previous sections, we define a guideline in terms of a number of elements: checkpoints expressed with objects, operations and conditions to be verified. In specific, an XML-Schema has been defined in order to define the general structure of our Guideline Abstraction Language (GAL). Through the XML-Schema [27] some restrictions on data types can be defined and controlled, such as URL source addresses, evaluation date, and especially possible values for <eval_object> and <el>. Below a fragment of the XML-Schema used for defining the guideline language is reported.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:html="http://www.w3.org/1999/xhtml" xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
        <xs:element name="gdl_set">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="name"/>
                                <xs:element ref="guideline" maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="shortname" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>
…
        <xs:element name="guideline">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="criterion"/>
                        </xs:sequence>
…
<xs:element name="eval_object">
```

```
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="object"/>
                                <xs:element ref="select" minOccurs="0" maxOccurs="unbounded"/>
                        </xs:sequence>
…
```

Table 2 - Fragment of XML-Schema used to define the guidelines

The elements defined allow designers to handle X/HTML tags/attributes and CSS properties. The XML tags <eval_object> and <el> in our language can refer also to other XML-based language elements. Thus, with small refinements of the XML-schema we can specify guidelines that can be checked on other languages, such as SMIL or XForm. This is one of the next enhancements that we plan to do as future work.

4. THE TOOL

As mentioned in previous sections, several sets of specific usability and accessibility guidelines should be applied in order to improve Web interaction for different typologies of users. This requires a lot of work by developers who have to handle page code and many sets of guidelines. Moreover, Web sites often require an increasingly large number of pages to create and, especially, maintain. For all these reasons it can be useful to provide support for applying different sets of guidelines in order to simplify the developers' work.

4.1 The Tool Goals

The MAGENTA tool has been developed with the intent of checking whether a Web site is accessible as well as usable and MAGENTA provides support to improve it. To this end, the tool checks how satisfactorily the selected guidelines are applied 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. Firstly, the guideline set to be considered can be selected (see Figure 1). Then, through a list of checkboxes, the evaluator can decide which guidelines of the selected set(s) have to be checked.

The tool is not limited to checking whether the guidelines are supported but, in case of failure, it also provides support for modifying the code in order to make the resulting Web site more usable and accessible. Thus, when an error is detected, the tool points out what parts of the code are problematic and provides support for corrections indicating what elements have to be modified or added. The process is not completely automatic because in some cases the tool requires designers to provide some information that cannot be generated automatically. The tool knows which parts must be corrected (i.e. tags or attributes) and developers write those parts or values which can not be added automatically (see Section 6). Examples of criteria that require the designer's intervention are:

- Proper link content, in this case the tool asks the designer to provide meaningful text for the link;
- Proper style sheets, in this case the tool requires an indication of the file containing the external style sheets;
- Proper names for frames, tables and images; here the designer may have to provide the value for the summary attribute for tables or for the alternative text attribute associated with images. This can also happen for frame titles and names, if the two values are different, then the tool makes them consistent and provides the designer with the possibility of modifying the resulting value.

The tool supports access to external services such as those of W3C for checking whether the page has been written correctly according to the language syntaxes (the W3C validator and the W3C CSS validator). Another feature considered in designing the tool is the multi-language support. The user interface has been structured so that all interface labels for menus, items, buttons, and so on, are stored in external files, one for each language supported. When the user chooses to change the current language, the label content is loaded from the corresponding file. In order to adapt the tool to another new language, a simple translation of the labels is required in order to add a new corresponding language file.

4.2 The Tool User Interface

The main layout of the tool user interface is structured into three main areas (Figure 1):

(1) Guidelines supported, which list the currently supported sets of guidelines and allows designers to select any of them;

(2) Guideline list, which provides access to the list of guidelines associated with the selected set;

(3) Guideline descriptions, which shows a brief description and the list of checkpoints for each guideline listed.

The report of the application is shown in a separate application window when the evaluation process is finished (see Figure 2). In an early version of our tool, the report was within the main application window [5]. In MAGENTA we decided to show it into a separate window panel in order to make the application more accessible by assistive technologies. By this solution, evaluators can easily switch by keyboard between the main application window and the report window.

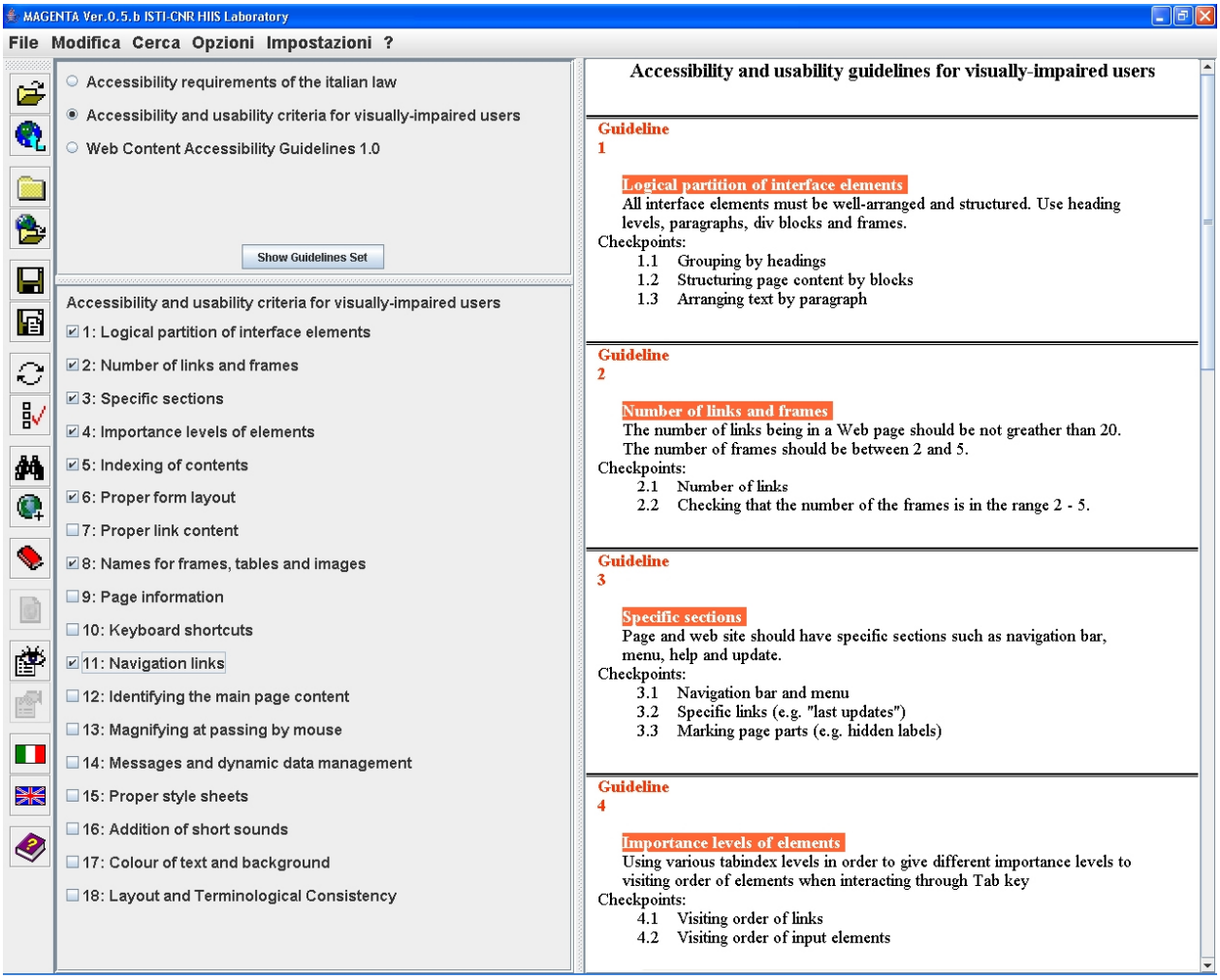

Figure 1 - Main user interface of MAGENTA

Currently, the supported guidelines that can be checked can be selected from three available groups (visually-impaired guidelines [12], W3C guidelines [25] and Italian Accessibility Law guidelines [10]). The designer can select the application of all or only part of them through check-boxes. This feature can be useful for better focusing on specific criteria and identifying application parts involved by the selected criteria.

At any time evaluators can get assistance on any guideline. Through a specific panel within the main application window (on the right), a brief description of the guidelines belonging to the selected set (e.g. visually-impaired) is shown. The online guide contains a short description and the relative checkpoints associated to each guideline. This could be useful for understanding the structure of the guideline considered. In addition, examples and relevant bibliography and Web sites are indicated as well.

Regarding the evaluation process, it is worth to note that in order to perform an evaluation of the frame names, appropriate links, adequate summaries for tables, and so on, a complete automatic objective evaluation can not be done. Thus, for this purpose we defined a set of dictionary files in which a list of potential wrong or appropriate terms are stored. For example, terms such as “click here”, “here”, “pdf”, “more information”, and so on, are stored as inappropriate text for links; or names such as “left”, “central”, “sx”, etc., are listed as frame names to be avoided. All these files can be updated and modified, so that evaluators can customize them. In addition, the use of such dictionaries is associated to the tool language. Thus, changing a language implies changing the dictionary used for evaluating / repairing.

4.3 The tool architecture

The tool has been implemented in Java. It first checks through the Tidy library whether the page is well-formed and then corrects any syntactical errors. Each evaluation criterion, defined in XML, is parsed by an interpreter class that executes the corresponding controls. Mainly, the analysis is performed through the DOM (Document Object Model) [24], a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of Web documents. Each set of guidelines is stored in a XML file, thus there are different XML files for different sets of accessibility and/or usability guidelines. The developer selects a guideline set before starting the evaluation process. The tool takes all the information on the guidelines from the XML file. Therefore, when a guideline is added to the file, or the referred file is changed, the tool can check the updated set of guidelines without having to modify the implementation. This implies that the tool has not to be recompiled when a set (file) of guidelines to be checked is modified.

5. EVALUATION REPORTS

When any evaluation or analysis is carried out, the resulting report plays an important role. In fact, the report should be clear and easy to understand. When many pages and several guidelines are simultaneously evaluated, the resulting report could be long and difficult to read. The main purpose of an automatic evaluation is to provide developers with some support in detecting and repairing potential problems with limited effort and in a short time. Furthermore, evaluators may have limited knowledge of how to handle the page code or the specific guidelines. Thus, the evaluation report should address all such aspects. In our new tool, MAGENTA, a report is provided taking into account the above mentioned possible drawbacks. A simple report showing just the main checking results on a single-page evaluation has been replaced with a more structured and flexible one. In this section we discuss its main features: we first introduce the XML format defined for structuring the evaluation results, and then the main properties considered are described.

5.1 Report Definition Language

As the evaluation and repair process is very important, the Evaluation and Repair Tools W3C working group (ERT WG) is developing an Evaluation and Report Language (EARL) [6] in order to support it. The EARL language has been conceived for documenting evaluation and repair information. Indeed, the Evaluation and Report Language (EARL) is a general-purpose language for expressing test results. Requirements for what needs to be tested are collected, then documented in a test specification. The tests are performed according to the test specification and the results may be stored in EARL. If several tests are performed, they are collected, analyzed, and presented in some sort of report. EARL is an RDF vocabulary used to make statements about how a resource performed in a test. Resource Description Framework (RDF) [18] is a general-purpose language for describing information. RDF uses "triples" to describe information (subject, predicate and object).

In our work we decided to use XML languages for defining both guidelines and evaluation results in order to generate files that can be imported and processed by different applications. In Section 3 we described a Guideline Abstraction Language (GAL); herein we propose a Report Definition Language (RDL) for structuring reports

generated by MAGENTA. The XML file structure that we have defined is also able to record all the information suggested by ERT to be reported in EARL: the context information, the test subject, the result and the test criteria.

All the evaluation results are stored in XML files, which are then used for generating a certain kind of report (textual or graphical). When the evaluation report is generated, all the necessary information is taken from the evaluation result XML file. Such a solution offers some useful advantages, which are worth to be considered:

- Building a report from an automatic or manual evaluation:
  using an external file for developing a textual or graphical test report can allow integration of different evaluations modalities (e.g. automatic, semi-automatic and manual inspection). When an evaluation report is generated it is not so important how the evaluation result file was written. This modality allows the evaluator to carry out an automatic check as much as possible, and then perform those inspections that can not be done automatically. All results can be stored in the same XML file;
- Creating quickly different types of reports:
  When all the evaluation results are coded in an external file, various kinds of reports can be created on fly depending on the stored data;
- Rebuilding reports without having to perform evaluation again:
  When an evaluation is performed and all its results are stored in the XML file, any report can be built at any time without having to carry out the evaluation again;
- Exchanging evaluation results with other tools:
  All data stored in an external file can be exchanged and used by different tools developed for various purposes (e.g. user interface generation, comparing of results provided by different tools, etc.);
- Comparing easily several evaluation results:
  using several report files allows comparing evaluation results. Different kinds of comparisons can be made, such as comparing pages belonging to the same or different Web sites, among guideline and checkpoint evaluations, tests performed on the same subjects (pages or Web sites) at different date and time, and so on. Statistics and assessment of data can be done as well.

Table 3 shows the Schema used for structuring the XML report file. Table 4 reports a portion of an example of XML file generated by the evaluation process.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
<xs:element name="eval_report">
        <xs:complexType> <xs:sequence>
        <xs:element ref="context_info"/>
        <xs:element ref="results"/>
        </xs:sequence> </xs:complexType> </xs:element>

<xs:element name="context_info">
        <xs:complexType><xs:sequence>
        <xs:element ref="evaluator"/>
        <xs:element ref="eval_info"/>
        </xs:sequence></xs:complexType></xs:element>
<xs:element name="evaluator">
        <xs:complexType><xs:sequence>
        <xs:element ref="name"/>
```

```
        <xs:element ref="disab_type"/>
        <xs:element ref="assistive_tech"/>
        <xs:element ref="browser"/>
        </xs:sequence>
        <xs:attribute name="type" use="required">
        <xs:simpleType><xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="automatic"/>
        <xs:enumeration value="semi-automatic"/>
        <xs:enumeration value="manual"/>
        </xs:restriction></xs:simpleType></xs:attribute></xs:complexType></xs:element>
<xs:element name="name">
        ...
        <xs:attribute name="skill" use="required">
        ...
        <xs:enumeration value="beginner"/>
        <xs:enumeration value="intermediate"/>
        <xs:enumeration value="advanced"/>
        ...
        </xs:element>
<xs:element name="disab_type">
        ...
        <xs:enumeration value="none"/>
        <xs:enumeration value="visual"/>
        <xs:enumeration value="auditory"/>
        ...
<xs:element name="eval_info">
        <xs:complexType>
        <xs:attribute name="date" type="xs:dateTime" use="required"/>
        ... </xs:complexType> </xs:element>
<xs:element name="results">
        <xs:complexType> <xs:sequence>
        <xs:element ref="page_report" maxOccurs="unbounded"/>
        </xs:sequence> </xs:complexType> </xs:element>
        ...
</xs:schema>
```

Table 3 - XML-Schema used for defining a report XML file

```
<eval_report>
        <context_info>
        <evaluator type="manual">
                <name skill="advanced">Barbara</name>
```

```
                <disab_type>visual</disab_type>
                <assistive_tech ver="7.0">jaws</assistive_tech>
                <browser ver="6">IE</browser>
                </evaluator>
        <eval_info date="2005-07-21T09:30:47-05:00" />
        </context_info>
        <results>
                <page_report src="http://www.mysite.it/index.html" result="not-passed">
                        <gdl idgdl="7" summary="Proper link content" gdlset="visually-impaired"
crtype="usability">
                                <cp id="7.1" summary="Textual links" priority="1" cpresult="N">
                                        <err type="e" iderr="7.1.1" occurrence="5">
                                                <msg>Found not clear links within the page</msg>
                                                <err_info>
                                                        <err_line>35</err_line>
…
                                                </err_info>
                                        </err>
                                </cp>
                        </gdl>
                </page_report>
        </results>
</eval_report>
```

Table 4 - Sample portion of an XML result file

The XML report file is structured for containing useful information, which can be used for various purposes. Indeed, the report contains general information about the evaluation carried out – i.e. evaluation date, tools used and evaluation type – and evaluation results for each checked page. In this regard, information such as page source, data on guidelines tested (gdlid=”7” and summary=”Proper link content”), and the evaluation result as well, is stored within the report. All these data can be used for generating various kinds of graphical, statistic reports, comparing evaluations carried at different time, and so on. In addition, specific data on types of errors are indicated as well by indicating their type and identification. The message stored within the <msg> tag is obtained by the iderr attribute. Lastly, the error lines are computed and stored in order to provide additional information to better detect the problems within the page code. Concluding, through all these information various kinds of reports can be generated. Next paragraphs describe some possible features and advantages that can be gained by using the XML report proposed. These XML-based reports, including the global evaluation report, can also exported in other common formats such as PDF, XHTML and so on. This functionality can be useful for reading the technical report outside the application and better printing it.

5.2 Compactness and message differentiation

When a single page is evaluated, “compact” messages are shown to indicate success or failure results for each criterion checked. Message differentiations are also used for describing various types of errors. Errors, warnings and alerting or success results are clearly differentiated through specific words: “Errors”, “Warnings” and “Alerting” and/or specific messages associated to guidelines/checkpoints (e.g. Please, be careful that heading levels are used appropriately). The same results are also provided by graphical rendering (i.e. graph bar). Such a solution provides the same information (i.e. checking result and error type) both through visual modality (graphical) and accessible

indication (textual). In addition, the appropriate messages and occurrence of the detected errors are conveyed too (e.g. “Errors: found 3 images without alternative descriptions”). Similar messages are given when several pages are simultaneously checked, that is the number of detected errors and warnings. The evaluation report is shown in a specific window panel of the application, which contains both a textual and graphical formats for summarising all the detected results (see Figure 2).

5.3 Multi-pages evaluation report

When multiple pages are evaluated, a tabbed report with two panels is used. A first panel (global report) is used for summarizing the general evaluation results: the number of errors found is reported for each page. In a second panel (page report) the errors, warnings and alerts are detailed for a single page at a time. When the evaluator selects one page among those evaluated, the corresponding checking results are detailed as explained before. This report structure allows evaluators to receive an overview of the global evaluation and provides them with the details on a single page on demand as well.

Besides a concise statistical survey of the evaluation, results can be viewed using graphical representations, with the possibility to show the distribution of detected errors for each guideline considered (see Figure 2). By selecting the corresponding bar it is then possible to receive the detailed list of associated errors along with more information regarding where they occurred.

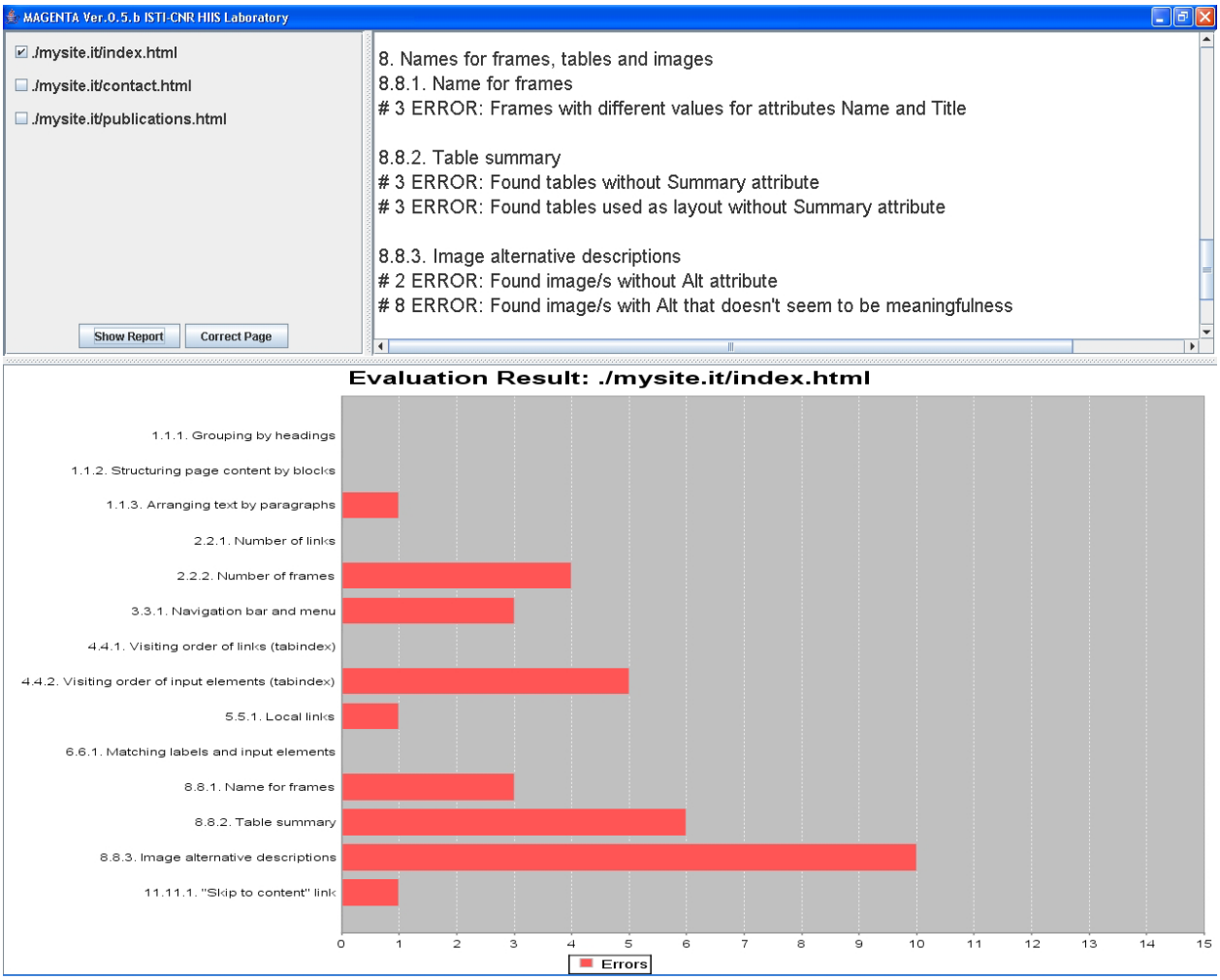

Figure 2 - Graphical report summarising errors detected

6. SUPPORT FOR REPAIR

Each detected error can be corrected through a semi-automatic support provided by the tool. Repair assistance is based on the Document Object Model (DOM). Thanks to the DOM platform interface [24], page elements and attributes can easily be navigated and handled dynamically. When developers perform evaluation of one or several pages, a multi-page report is generated. The report is split into three main panels: (1) list of evaluated pages, (2) list of detected errors/alerts, and (3) graphical report (See Figure 2). The list of checked pages is indicated in the first panel: by selecting any of them, the evaluation report associated with that Web page is shown into others two panels. At this point, the evaluator can decide to correct errors detected for a certain page. Repairing support can be activated through two modalities. In one case, the evaluator can select a Web page from the checked page list and then click on the "Correct page" button available in the panel (see Figure 2). The correction part first shows the DOM page structure. The focus moves to the element associated with the first error detected within the page. Then, the evaluator can decide to skip directly to a specific error detected. In fact, each detected error/alert listed within the second panel (Figure 2) is a link. It is also possible to select a certain error/alert message, then the correction window is activated and the focus moves to the element associated to the first error occurrence (see Figure 3). When the correction window is open, the evaluator can repair the error by writing/changing values and attributes. The tool allows designers to insert new attributes by using the "Create new attribute" button.

Furthermore, it is possible to navigate among errors and alerts. Through the button "Next error" (or "previous error") next (or previous) detected error can be visited. It is also possible, through the button "next similar error" to move the focus to the next error of the same type of the current one. When evaluators want to correct another page, "Correction window" must be closed and another Web page can be selected from the evaluated page list.

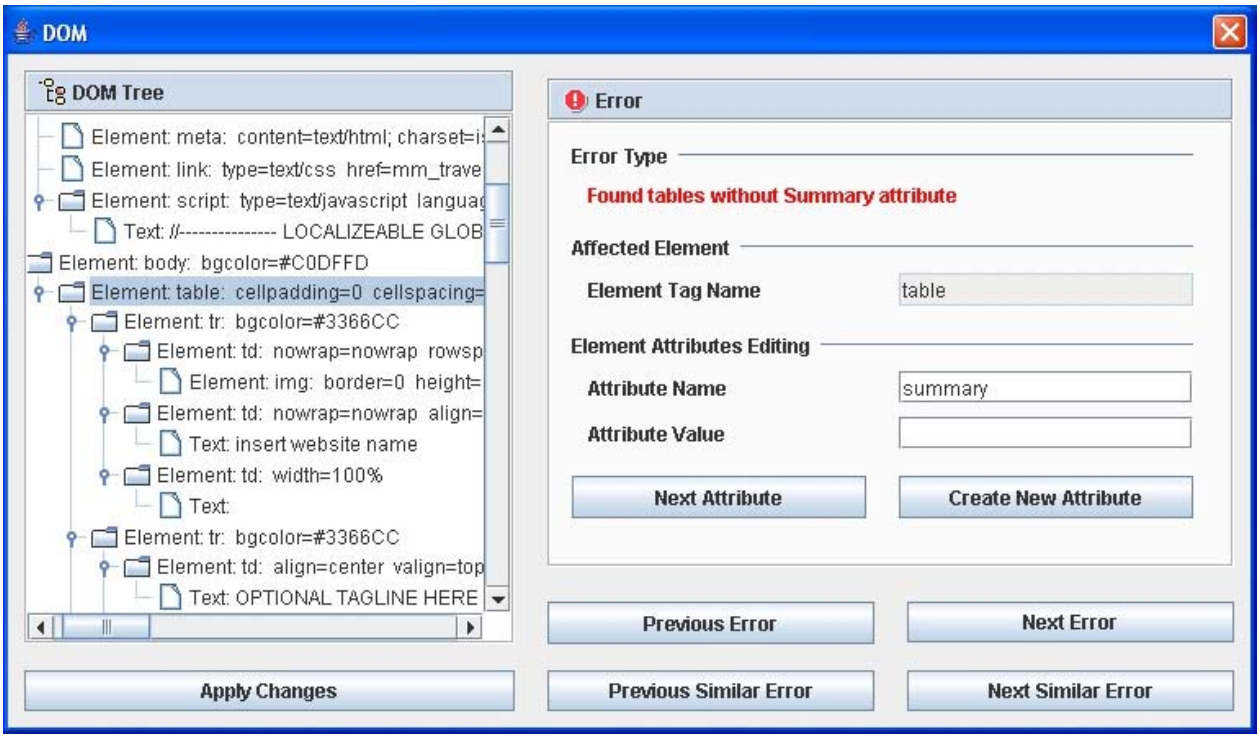

Figure 3 - Example of Page Correction

7. TOOL USAGE

In order to collect some feedback on tool usage, our tool was provided to a company that has a group dedicated to design and evaluation of Web sites. The goal was to receive some commends, remarks as well as doubts and questions. In general, the comments were positive. However, some remarks and additional specific requirements were provided. A first issue was related to the possibility of using the tool through a proxy server. Initially the tool did not support this possibility. Thus, a specific customisable function has been added for this purpose.

The UI of the tool has been considered simple and intuitive. About loading pages to check, evaluators suggested to insert four specific menu items for downloading Web pages ("Open file…", "Open URL…", "Open directory…" and "Open Web site"). This makes it clearer for less skilled evaluators how local or on line Web pages (or whole directory) can be downloaded and then checked. Some comments were related to the time spent for searching Web pages on the Internet and in downloading the pages selected for evaluation. Such remarks pointed out that the crawler used for this purpose is not particular efficient for quite large Web sites. So, we started to study how that crawler can be improved in order to make it quicker and more efficient.

Other issues were related to reporting. Early reports were generated and saved only in XML format. Evaluators requested to add the functionality to convert/save reports also in others formats, especially in X/HTML. In fact, if a compact report can be saved in X/HTML format, it can be easily accessed, read, and printed. Thus, such a file can easily handled and used as a final document, which can be signed and referred as a evaluation result to be used also for formal purposes. This is particularly useful for auto-certificating accessibility of Web sites by public administrations as requested by some national laws. So, we decided to implement the functionality for converting reports in various formats: X/HTML and PDF.

The tool was given also to a group of students who reported positive comments. After a twenty minutes introduction to the tool, students used it in evaluating some Web sites. They were able to carry out the evaluation and to identify accessibility errors without particular difficulties. The students expressed some comments on the alternative texts for images. They received alerts generated by MAGENTA for "probably" non-appropriate descriptions (e.g. click here, etc.). This functionality is performed by the tool thanks to the external dictionary we added for possible non-appropriate often used link contents. When a link has a text belonging to such list, the tool generates an alert message for the evaluator/developer. Similar support is provided for non-appropriate image descriptions, which are stored in an external dictionary as well. The students appreciated this feature. They suggested improving it by adding the possibility of displaying the pictures referred by the alternative description. This is particular useful when evaluators have to insert or correct an appropriate description for each figure.

In general, the various comments received have provided useful suggestions to improve the UI of the tool and its functionalities as well. Further tests with external groups will soon be performed in order to gather further feedback and suggestions for new additional features.

Some accessibility features have also been considered in the tool in order to support its use by assistive technology. First of all, some short sounds are generated in order to get attention on some specific events. For example, a short sound is used when the evaluation process is completed. This is aimed at improving the accessibility features of the tool as well as to inform on the end of the process allowing developers to do other activities by keeping in background the evaluation process. This added feature is useful especially when many pages are evaluated simultaneously. Besides, keyboard shortcuts were added in order to simplify the interaction when the mouse can not be used. Furthermore, in order to improve the navigation by a screen reader such as Jaws, the application focus is controlled by "driving" it towards specific interaction areas (e.g. guideline checkboxes to select when the evaluation has to be defined, or the evaluation report when the evaluation process is finished).

8. CONCLUSIONS and FUTURE WORK

In this paper, we have presented a tool aiming to providing flexible support to obtain accessible and usable Web sites through support of multiple sets of guidelines, repair of Web pages, and effective graphical reports. To this end, a Guideline Abstraction Language (GAL) has been defined through an XML-schema and applied in order to obtain a tool independent of the guidelines to evaluate. A Report Definition Language (RDL) based on XML-Schema has been proposed as well, in order to improve reporting.

The tool provides interactive support for checking the application of the selected guidelines and improving Web site implementations when it detects potential problems. This is an important contribution in reducing the cost of evaluation according to the considered guidelines, thus making such evaluation affordable for a larger number of Web designers and developers. It is also useful to obtain a more consistent application of the guidelines with respect to what can be obtained through manual approaches.

Future work will be dedicated to further extending the evaluation environment in several directions: integration of the current method based on automatic code inspection with assessments performed through other methods (such as

automatic log analysis) and integration of such support even in the development phase in such a way that designers are guided to implement their Web sites in accordance to the usability and accessibility design criteria. We plan to provide more interactive support for defining guidelines in terms of our abstract language for guidelines and to refine the tool in order to better support the check of guidelines not only for X/HTML, but also for other languages such as X+V, SMIL, XForm, etc.

Acknowledgments

We thank Domenico Natale (SOGEI) and Ivan Norcia for useful discussions on the topics of this paper.

References

1. Abascal J., Arrue M., Fajardo I., Garay N., Tomás J., Use of Guidelines to automatically verify Web accessibility. Universal Access in the Information Society, special Issue on "Guidelines, standards, methods and processes for software accessibility", Springer Verlag, Vol.3, N.1, 2004, pp. 71-79.
2. ATRC, A-Prompt: Web Accessibility Verifier, Adaptive Technology Resource Center (University of Toronto) and Trace Center (University of Wisconsin), Canada & USA, accessible at http://www.snow.utoronto.ca
3. Beirekdar A., Keita M., Noirhomme M., Randolet F., Vanderdonckt J., Improved Reporting for Automated Accessibility and Usability Evaluation of Web sites, proceedings INTERACT 2005, LNCS Springer Verlag.
4. Clarck D., Dardailler D. (1999) Accessibility on the Web: Evaluation and repair tools to make it possible. In proceedings of the CSUN Technology and Persons with disabilities Conferences, Los Angeles, CA. Available at http://www.cast.org/bobby.
5. Correani F., Leporini B., Paternò F., 2004. Supporting usability for vision impaired users in web navigation. In Lecture Notes in Computer Science (Vol. 3196/2004), Springer-Verlag Heidelberg, "User-Centered Interaction Paradigms for Universal Access in the Information Society: 8th ERCIM Workshop on User Interfaces for All , Vienna, Austria, June 28-29, 2004, pp. 242-253.
6. Evaluation and Report Language (EARL) 1.0 available at http://www.w3.org/TR/EARL10/
7. Ivory, M. and Hearts, M. (2001) The State of the Art in Automating Usability Evaluation of User Interfaces. ACM Computing Surveys, Vol, 33, No 4.pp. 470-516.
8. ISO IS 9241-171 Ergonomics of human-system interaction - Guidance on software accessibility. (a restructured version of ISO TS 16071).
9. ISO TS 16071 Guidance on accessibility.
10. Law Stanca, Law n. 4, January 9, 2004: Provisions to support the access to information technologies for the disabled http://www.pubbliaccesso.it/normative/law_20040109_n4.htm _20040109_n4.htm
11. Lee A., Hanson V., Enhancing Web Accessibility, Proceedings ACM Multimedia 2003, pp. 456-457, ACM Press.
12. Leporini, B., Paternò, F. (2004). Increasing Usability when Interacting through Screen Readers, International Journal Universal Access in the Information Society (UAIS), special Issue on "Guidelines, standards, methods and processes for software accessibility", Springer Verlag, Vol.3, N.1, pp. 57-70.
13. Mariage, C., Vanderdonckt, J., Pribeanu, C. State of the Art of Web Usability Guidelines, Chapter 41, in R.W. Proctor, K.-Ph.L. Vu (Eds), "The Handbook of Human Factors in Web Design", Lawrence Erlbaum Associates, Mahwah, 2005.
14. Mankoff J., Fait H., Tran T., Is Your Web Page Accessible? A Comparative Study of Methods for Assessing Web Page Accessibility for the Blind, Proceedings ACM CHI'05, pp.41-50, ACM Press.
15. Mayhew D. J.. Principles and guidelines in software and user interface design. Prentice Hall, Englewood Cliffs, New Jersey, 1992.
16. Mohamad Y, Stegemann D, Koch J, Velasco C A (2004). Imergo: Supporting accessibility and Web standards to meet the needs of the industry via process-oriented software tools. In: Miesenberger K, Klaus J, Zagler W, Burger D (eds). Proceedings of the 9th International Conference ICCHP 2004, Paris, LNCS 3118, pp. 310-316. Berlin-Heidelberg: Springer-Verlag.

17. Nicolle, C., Abascal J. Inclusive design guidelines for HCI, p. 285, Taylor & Francis, 2001.
18. Resource Description Framework (RDF) Model and Syntax Specification at http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
19. Section 508 standards. http://www.section508.gov
20. Smith S. L. and Mosier J. N.. Guidelines for designing user interface software. Mitre Corporation Report MTR-9420, Mitre Corporation, 1986.
21. Stephanidis, C., Paramythis, A., Karagiannidis, C., & Savidis, A. (1997). Supporting Interface Adaptation: The AVANTI Web-Browser. In C. Stephanidis & N. Carbonell (Eds.), Proceedings of the 3rd ERCIM Workshop "User Interfaces for All", Obernai, France, 3-4 November.
22. USABLE NET (2000) LIFT ON LINE. Available at http://www.usablenet.com/
23. Vanderdonckt, J., Guide ergonomique des interfaces homme-machine, Presses Universitaires de Namur, Namur, 1994.
24. W3C Document Object Model (DOM) http://www.w3.org/DOM/
25. Web Accessibility Guidelines 1.0. Web Accessibility Initiative, W3C Recommendation 5-May-1999. Accessible at http://www.w3.org/WAI/GL/WCAG10/
26. Web Content Accessibility Guidelines 2.0, W3C Working Draft 11 February 2005, available at http://www.w3.org/WAI/GL/WCAG20/
27. XML Schema Part 0: Primer Second Edition. W3C Recommendation 28 October 2004 available at http://www.w3.org/TR/xmlschema-0/