Easing Web Guidelines Specification

Barbara Leporini, Fabio Paternò, Antonio Scorcia
ISTI - CNR
Via G. Moruzzi, 1 - 56124, Pisa
{barbara.leporini; fabio.paterno; antonio.scorcia}@isti.cnr.it

Abstract. More and more accessibility and usability guidelines are being proposed, especially for Web applications. Developers and designers are experiencing an increasing need for tools able to provide them with flexible support in selecting, editing, handling and checking guidelines. In this paper, we present an environment for addressing such issues. In particular, an interactive editor has been designed to assist designers and evaluators in abstracting and specifying new and existing guidelines in an XML-based Guideline Abstraction Language (GAL). Our tool has been designed in such a way to be able to check any guidelines specified in this language without requiring further changes in the tool implementation.

Keywords: Guidelines, editor, abstraction language, accessibility, usability**.**

1. INTRODUCTION

Design guidelines are increasingly applied in order to improve Web user interfaces and make them consistent, in particular to attain better accessibility and usability. Indeed, several guidelines have been proposed in the literature for general user interfaces (e.g., [10]) as well as for Web interfaces (e.g. [5, 9, 12]). Thus, in order to assure a certain level of accessibility and usability, developers have to orient themselves among several sets of guidelines.

In order to support designers in checking guidelines, a number of automatic tools have been made available on the market, such as Bobby, now called WebXACT [14], Web ATRC [13], Lift [8], and so on. However, such tools have several limitations. For example, they only support the assessment of a fixed set of guidelines. In order to overcome such limitations, we have investigated new solutions for more flexible support [7]. To this end, we have designed and implemented an environment, MAGENTA (Multi-Analysis of Guidelines by an ENhanced Tool for Accessibility) which is a tool that supports the inspection-based evaluation of accessibility and usability guidelines. One of its main features is that it is guideline-independent. In fact, our tool has been developed considering the limitations of most current tools, in which the guidelines supported are specified in the tool implementation, with a certain

number of consequent drawbacks, such as having to change the tool implementation each time a guideline is modified or added. Thus, the tool has been developed in such a way to be able to deal with any type of guidelines, which are defined externally from the tool. To this end, an abstract language for guidelines has been defined. If the guidelines are expressed through it and stored in external files then the tool can check any of them without modifications to its implementation.

However, we soon realised that many Web designers can have difficulties in handling and formulating guidelines, usually specified through XML-based languages. For these reasons, we have extended the environment with a new tool conceived to assist developers and evaluators in proposing, specifying and modifying guidelines.

In this paper we present our solution to this issue, which includes a Guideline Editor designed to assist developers in specifying new guidelines and handling already existing ones (such as those for accessibility and usability for the Web). We also report on the results gathered through a user test conducted in order to evaluate the editor in terms of user interface as well as functionalities. Based on the collected data during the evaluation, we decided to further improve the guideline editor tool in order to ease its use and the guideline specification process as well.

After a short overview of related work on the topic addressed, in this paper we firstly describe our Guideline Abstraction Language (GAL) used for specifying guidelines. Next, we introduce the main functionalities of the proposed editor for Web Guidelines. We also report on the data and opinions collected through a user test. Then, changes made on the editor to enhance its functionalities are described. Lastly, we draw some conclusions and provide indications for future work.

2. Related Work

National and international legislations more and more promote inclusive policies and practices. With regard to accessibility, the number of proposed guidelines is rapidly increasing, thus requiring an increasing effort by developers who have to handle them. Indeed, in addition to the W3C guidelines almost each Western country has national guidelines for accessibility, and new guidelines focusing on specific types of users have started to appear. To this end, several automatic or semi-automatic supports have been proposed for easily handling guidelines. In particular, several automatic tools have been 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 a tool that provides designers and evaluators with good support to easily incorporate new additional accessibility guidelines. Another contribution in this area is DESTINE [2], which supports W3C and 508 guidelines specified through another language. Also, in the BenToWeb project two tools are proposed to support test case management for accessibility test suites [4]: Parsifal - a desktop application which easily allows editing test description files – and Amfortas - a Web application that allows controlled evaluation of the test suites by users. A different approach is presented in [6], which supports the creation of an internal

design knowledge management tool for Web developers as a means to encourage user-centred development practices.

In order to verify various usability aspects related to accessibility issues, other tools have been proposed. The Functional Accessibility Evaluation (FAE) Tool [3] provides a means to estimate the functional accessibility of Web resources by analyzing Web pages and estimating their use. FAE uses rules for testing each of the functional accessibility features of navigation, text descriptions, styling, scripting and the use of standards. Such rules are defined on X/HTML tags. This means that they are directly implemented in the source tool code. Consequently, to add or modify a rule the source code must be modified.

Also the approach reported in [11] is based on the separation between the guideline specification and the evaluation engine. Guidelines are stored into a XML-based repository. A guideline can be expressed in terms of conditions to be satisfied on HTML elements (i.e., tags, attributes). The formal guideline is expressed according to Guideline Definition Language (GDL), a XML-compliant language. Although the GDL allows mapping the informal statements associated with initial guidelines onto formal statements, such a process has to be manually carried out by the developers who have to directly handle the XML constructs specifying the guideline structure. To this end, we propose an editor that supports designers in the guideline specification, even if they do not have any knowledge of XML and its constructs.

In summary, our contribution in this area is represented by an environment, which is able to work with different sets of guidelines, provided they are appropriately specified in external files. This means that developers or evaluators should formalize and specify the guidelines in a specific format based on XML. Since this may require particular abilities as well as a long time, we decided to develop a tool supporting guidelines editing and modifications, which is presented in this paper, since there is a lack of similar tools as indicated in the above discussion.

3. The Guideline Abstraction Language

The tool we propose aims to provide novel support for Web designers, such as providing interactive support for defining new guidelines as well as customizing new groups starting with other existing guideline sets. Since our intent is to provide designers with a tool independent of the guidelines to check, the proposed approach is based on a language able to abstract the usual types of conditions that designers aim to assess. The tool includes an engine able to interpret and check any guideline specified by GAL without requiring any further modifications in its implementation. Currently, in the MAGENTA tool three sets of guidelines can be specified and interpreted at run-time: (1) the Italian law for accessibility requirements; (2) W3C WCAG 1.0 guidelines; (3) accessibility and usability guidelines for the visual impaired.

3.1 Guideline abstraction

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. Indeed, a guideline should be specified in a rigorous manner so that it can be automatically processed. and managed. In practice, the solution consists of abstracting guideline statements into a well-defined XML-based language. To this end, an XML-Schema has been defined in order to express the general structure of our Guideline Abstraction Language (GAL).

In defining a guideline abstraction, the main features and elements characterizing the guideline itself must be defined. In brief, each guideline expresses a principle to be complied with by applying one or more conditions to checkpoints, which are constructs in the implementation language. Hence, general features, objects involved in the checking process, and conditions referred to such objects must be structured and specified through our GAL specification language. In particular, we define a guideline in terms of a number of elements:

- checkpoints expressed with objects,
- operations (Exist, Count, Check, Execute), which identify the general processing required to check a guideline,
- conditions to be verified.

Such Abstraction Language has been refined over time in order to better define more complex guidelines. We added new attributes in order to be able to specify more precise conditions. It is possible to select specific objects (tags) having given attributes with specified values (e.g. links with the class=”navbar”). Such a selection allows designers to include in the guideline specification also cases in which only objects (tags) with given features (attributes) should be considered. Moreover, the language supports the selection of those objects (tags) being children of other tags in the DOM (Document Object Model) structure. For example, in this context the tags <head> and <body> are children of <html> tag (root). This allows designers to specify specific restrictions associated with the tags considered. For instance, to check graphical links - <img src=””> - in the XML-based guideline we can specify the tag <img> as child of the tag <a>. In this way, the checkpoint only focuses on the tags <img> included between the tags <a> and . This feature has been added to be more precise and able to restrict the selection of objects to involve within the checkpoints.

3.2 Abstraction examples

In order to understand how the abstract description of a guideline can be obtained through GAL, in this paragraph we provide the reader with some examples.

Example 1: Language identification

When using different languages on a page, any language change should be clearly identified by using the "lang" attribute. In this context, let us consider the guideline "Use mark-up elements to facilitate document language identification. In particular, in multi-language pages designers should use the appropriate attributes to mark links

written in a different language.". This means we need to check the presence of the attribute "lang" in the tag <html> as well as the link tags <a> that do not refer to anchors (i.e. tags with the attribute href).

To abstract this guideline, the objects involved in the guideline must be extracted. Firstly, there are two aspects to consider: (1) the object to examine is the tag <html>; (2) the elements to analyse are all links. These two controls indicate that two checkpoints should be specified. For each checkpoint, in addition to the main object involved - i.e. <html> tag in the first case, and <a> in the second one - the corresponding conditions must be defined. The main structure could be abstracted as indicated below:

GDL: $[HTML^{tag}\ (Exist\ LANG^{attrib})]^1$ and $[A^{tag} \mid HREF^{attrib}\ (Exist\ LANG^{attrib})]^2$

In order to satisfy the guidelines, the checkpoints must be applied. A checkpoint defines a series of conditions to verify or to apply in order to satisfy an aspect of a guideline. In our case, each checkpoint is included in the square brackets marked 1 and 2; round brackets are conditions to be verified.

Next figure shows how the guideline example can be specified by using the XML-based GAL language. This guideline example was also used for a task (T5) assigned in a user test conducted to evaluate the editor (see section "Editor evaluation").

```
<guideline id="22" summary="Language identification">
<checkpoints rel="and">
<cp id="22.1" summary="Main document language" priority="1">...
<object type="tag">html</object>...
<conditions rel="or">
<evaluate operator="exist" idErr="22.1.1">
<el type="attrib">lang</el></evaluate>
</conditions></cp>
<cp id="22.2" summary="Marking links with different language" priority="3">
<object type="tag">a</object>
<select type="attrib">href</select></eval_object>
<conditions rel="and">
<evaluate operator="exist" idErr="22.2.1">
<el type="attrib">lang</el></evaluate>
</conditions></cp></checkpoints></guideline>
```

Table 1 - Example of guideline abstraction with GAL.

Example 2: Equivalent alternatives to auditory and visual content

Let us consider the guideline n. 1 of WCAG 1.0, which is related to alternative contents. This guideline is composed of several checkpoints. For our example we consider the checkpoint 1.1: "Provide a text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content). This includes: images, graphical representations of text (including symbols), image map regions, animations, applets and programmatic objects, images used as list bullets, graphical buttons, sounds, and so forth.". To simplify our example, we only report the specification of the "image" case. The next figure shows the XML-based specification of the simplified checkpoint 1.1 according to the GAL schema.

```
<guideline id="1" summary="Alternatives to auditory and visual content">
<checkpoints rel="and">
<cp id="1.1" summary="Provide a text equivalent for images" priority="1">
                    <object type="tag">img</object>
                    <conditions rel="or">
                    <evaluate operator="exist" idErr="1.1.1">
                      <el type="attrib">alt</el></evaluate>
          <evaluate operator="check" cond="not_equal" idErr="1.1.2">
                                    <el type="attrib">alt</el>
                                    <el type="value">""</el></evaluate>
                    <evaluate operator="check" cond="not_belong" idErr="1.1.3">
                                    <el type="attrib">alt</el>
                                    <el type="file">"noimages.dic"</el></evaluate>
<!-- repeated code for longdesc→
                    </conditions></cp></checkpoints></guideline>
```

Table 1 - Example of abstraction according to GAL

Actually the checkpoint should be extended with other conditions (<conditions>) to include all the non-text elements, such as objects, applets, and so on.

The checkpoint involves the tag <img> to which five evaluations are applied: (1) the presence of attribute "alt"; (2) the "non-empty" verification for that attribute; (3) the "not belonging" of alt content to a black list of possible values which are commonly used for alternative descriptions (e.g., "logo" or "file-name.jpg"); (4) the control of the presence of "longdesc" attribute and (5) that its value is not empty. These evaluations are mainly specified with the operators "exist" as well as "check" with the conditions "not_equal" and "not_belong". The black list of values are stored in external files, such as "noImages.dic".

The kind of error is coded with the attribute idErr defined in GAL. A progressive number is associated to each evaluation statement (<evaluate>); it is obtained by adding a progressive number to the checkpoint id (e.g., 1.1.2 where 1.1 is the checkpoint id). The list of possible errors is externally coded by matching the type - (e) error, (w) warning and (a) alert - with corresponding message for each idErr.

Example 3: Logical partition of interface elements

As third example let us consider the guideline that requires well structured content in a page in order to simplify Web navigation through screen readers: " All interface elements should be well-arranged and structured. Use heading levels, div blocks and paragraphs". To specify this guideline three checkpoints must be defined: (1) for heading levels; (2) for div blocks; and (3) for paragraphs. The next figure reports fragments of the guideline specification based on GAL.

CP	**Fragment**
1. Headings	... <conditions rel="or"> <evaluate operator="count" cond="greater" idErr="1.1.1"> <el type="tag"> h1 </el>

	<el type="value"> 0 </el></evaluate>...
2. div blocks	<conditions rel="or"> <evaluate operator="count" cond="greater" idErr="1.2.1"> <el type="tag"> div </el> <el type="value"> 1 </el></evaluate></conditions></cp>
3. Paragraphs	<conditions rel="and"> <evaluate operator="count" cond="greater" idErr="1.3.1"> <el type="tag"> p </el> <el type="value"> 0 </el></evaluate></conditions></cp>

Table 2 - Fragments of guideline specification according to GAL.

In the checkpoint 1, heading levels are counted. In practice, the control consists in verifying that the number of heading levels is at least 1. Thus, in this case the effective verification of an appropriate page structure is demanded to the evaluator; the checkpoint points out the usage or not of headings in the page. The evaluator will manually check if the structure has been appropriately applied. A similar checking is applied to the checkpoints 2 and 3. Checking is not easy to perform in completely automatic way. The main goal of this guideline and its evaluation is to stimulate attention on the page structure by the developers.

4. The Guideline Editor

When a guideline must be defined so that a tool can deal with it automatically, the problem is to transform its natural language description into technical terms. The abstraction process is not easy and requires some effort when implementation languages for the Web are considered. In order to facilitate this process, a graphical editor was designed and added to our environment.

4.1 Functionalities

The Guideline Editor (GE) has been designed to assist developers in handling single as well as groups of guidelines. The tool supports new guideline definition and various types of editing. In summary, the editor offers various useful features in order to facilitate multiple guideline sets management and general utilities. In particular, the guideline editor provides support for:

- Defining or modifying single guidelines;
- Importing guidelines from various sets in order to create customised groups;
- Organising, classifying and browsing guidelines into groups;
- Handling several sets of guidelines simultaneously.

Regarding manipulation of single guidelines, the tool allows developers to add new ones or modify/delete existing ones. In practice, the graphical editor guides the designer in specifying all the XML tags necessary to define a guideline. Figure 1

shows an example in which the designer specifies the guideline “The number of links in a navigational bar should be less than 7”. The left part of the graphical representation is devoted to defining the scope of the application, in this case the tag *a* with class attribute value equal to navbar in HTML documents. The right part indicates the condition to check: in this case that the indicated tags should be less than 7. Furthermore, it is also possible to add new guidelines by importing them from other guideline set(s). This allows designers to create customised set(s) of guidelines by reusing already specified ones. More precisely, regarding individual guidelines, the tool makes it possible to:

- Handle general information on the guideline (short name, description, etc.);
- Select the object to be considered in the guideline (i.e., language, tags and restriction on the object itself).
- Handle the checkpoints needed: general information for each checkpoint (e.g., short name, description) and checkpoint relations (and | or);
- Define the conditions to be verified, by selecting operators and conditions to be applied at a runtime (i.e. when the tool will carry out the evaluation).

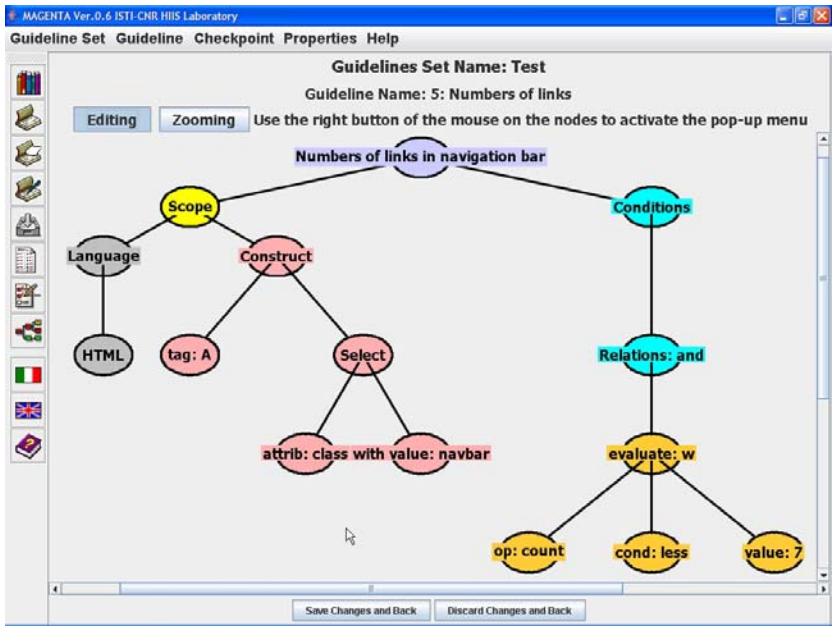

Figure 1. The Guideline Editor.

4.2 User Interface

The Guideline Editor UI is organized in various parts shown in cascade:

- *Main Guideline Management*, where all groups of guidelines are listed and can be handled. It is possible to create a new set, modify or delete an existing one.

- *Guideline set*, in this environment a new specific group of guidelines can be defined. General information on the set, as well as the list of guidelines existing in the group are presented.
- *Guideline specification*, where developers can structure and define a single guideline. Firstly, general data such as the short name and description of the current guideline as well as the main object – tag, attribute or property - involved in the guideline can be written and modified; next, all checkpoints and their conditions to be applied or evaluated can be edited through a direct manipulation graphical editor.

In guideline editing, the elements to be selected are listed in order to make them available for the evaluator and to avoid typing errors. In this way, the user is guided during the abstraction and definition process. For instance, when an object to be checked has been selected, the list-box for selecting the associated attributes displays only its attributes according to the implementation language considered. When the user has selected all the elements that express the objects to be checked, including their properties, checkpoints and conditions, the editor saves them in the XML-based file associated to the current guideline set.

While in the previous version many windows opened during the process of defining guidelines and checkpoints, an updated version utilised a single window to display such information, with the additional possibility to zoom in the parts of interest. In addition, a new interactive representation of the checkpoint structure was introduced in order to simplify the somewhat “complex” structure of a checkpoint via a hierarchical tree structure representation.

5. Editor Evaluation

5.1 Method

To get information about the actual use of the guideline editor prototype, we used two evaluation techniques: think aloud and questionnaires. We observed users interacting with the guideline editor. Users were asked to complete a set of predetermined tasks, while we watched and recorded the users' actions (using the paper-and-pencil analysis protocol). At the end of the test we asked users to fill in a questionnaire.

5.1 Participants

In order to evaluate the guideline editor prototype, ten designers were recruited, 7 men and 3 women. Since the guideline editor is conceived for an expert evaluator or developer, the choice in recruiting potential participants was limited. The user age varied from 22 to 38 years. All users were computer scientists: 40% were university graduates, whereas the other 60% were still students. On average, users had a poor

knowledge about guidelines (2.1/5), and a sufficient knowledge of the X/HTML language (3.1/5).

5.3 Tasks

The assigned tasks focused on the main functionalities offered by the editor, such as handling guideline groups or single guidelines. The five tasks assigned to the users were:

T1 Creating a new guideline group named "test".
The purpose was to create a new group associated to an XML file in which all guidelines elaborated by the user were stored. This file was considered the final result of the test carried out by each user.

T2 Importing various guidelines from existing groups.
The purpose of this task was working with several guidelines to populate the "test" group.

T3 Editing an existing guideline.
This task was mainly assigned to allow users to familiarize themselves with the guideline structure.

T4 Adding a new "guided" guideline.
This task was guided by providing the user with all the necessary information to describe the guideline formalization. The purpose was to observe users when they edited the elements without having to go through the guideline formalization process.

T5 Adding a new "unguided" guideline.
The purpose was to observe the user when formalizing a guideline expressed in a natural language.

5.4 Questionnaires

Since the purpose of our test was to collect information on the Guideline Editor (GE) interface and its functionalities, the questionnaire was basically composed of three kinds of questions: (1) general questions aimed at gathering information on users; (2) interface section, aimed at getting useful comments and impressions about the editor UIs; and (3) functionality section, to collect remarks on the main GE functionalities tested. To collect useful information and comments by the users, we included both open questions and "quantitative" questions (on a 1 to 5 scale).

5.5 Results

The observational data gathered through the think aloud test and subjective judgments collected by the questionnaires provided some useful empirical feedback. Additionally, all XML files created in T1 allowed us to analyse the task accomplishment for each user. Task T1 and T2 were carried out by all users, even if we observed that T2 presented some difficulties when extracting the desired guidelines from the groups. In fact, some users reported that the rendering structure used for grouping guidelines in the import procedure was not particularly clear.

Larger labels or a more guided procedure could be more useful for orienting users among many guidelines. Regarding the task T3 (guideline editing) – "verifying that the summary attribute of a table is not empty" - only 50% of users understood that the needed action was to add a new <evaluate> condition (i.e. a new branch in the checkpoint tree). Lastly, regarding the guideline creation requested in task T4 and T5, about half users encountered some problems in carrying out the tasks correctly. We noted some difficulties in formalizing a guideline due to some confusion encountered by the users in the logical flow of the needed actions. Concerning the last task (T5) – "abstracting and formalizing a guideline expressed in terms of natural language" - users worked better with the checkpoint structure than in the other tasks. This may be due to the greater familiarity with the tool acquired during the test (the sessions lasted on average 45 minutes per user).

In order to collect subjective opinions regarding the user interface and editor functionalities, users were asked to express some judgments on several aspects. The scale varied from 1 (lowest – negative value) to 5 (highest – positive value). Regarding the general user interface, the average rating was 3.2. Users expressed intermediate values - on average - also for the more specific UI aspects, such as location (3.6) and clearness (3.2) of elements, icon clearness (3.5) and graphical structures (3.4). The lowest average value reported (2.7) regards the logical flow in structuring a guideline. In the questionnaire, users expressed also some comments about difficulties encountered during the test.

Concerning the editor functionalities, users were asked to express opinions of the main functions tested. The averages are higher (around 4) than those expressed for the interface (around 3). The function considered most difficult by users is related to checkpoint handling. Indeed, handling a checkpoint was the most frequent error in performing the tasks T3 and T4. Task T1 and T2 were those considered the easiest. By analysing open questions, we observed that in general users appreciated the guideline editor. Most difficulties were related to the guideline formalization. This is probably due to the required advanced knowledge of X/HTML in terms of tags and attributes. To address this issue, we could improve the interaction by inserting a more descriptive and clearer list of elements. For example, instead of listing all tags <a>, <frame> and so on, we could use a list of more explicative items, such as link, frame, etc.. The same can be applied for the attributes. All in all, the editor had a positive impact on the users. However, some improvements were still needed in order to make it more usable and more intuitive.

6. The New Version of the Tool

Based on the gathered opinions and the observed user actions, various difficulties related to the user interface emerged. For this reason we decided to make some changes on the GE tool. Changes basically affected two kinds of aspects: (1) interface functionalities and (2) specification and abstraction process.

With regard to interface functionalities, the major revisions introduced are related to interface components (such as labels, buttons, graphical views, etc.). Concerning the labels, extended names rather than technical tags have been introduced to identify

the elements involved in the abstraction process. Indeed, users encountered some difficulties in orientating among the X/HTML tags. For instance, clearer labels (such as "Link", "Image", "Table", and so on) have been used to replace tags such as <a>, <img>, <table>, etc.. Although commands and functions are available on the tool bar as well as on the context menu (i.e. associated to the mouse right click), users showed some confusion in locating the necessary command. For this reason, various command buttons have been repeated in the current work area. In addition, in order to provide a visual structure of a guideline, a graphical tree was used. However, such a tree had various accessibility problems (for example, blind users interacting through screen readers are not able to use the mouse as required by editing such graphical representation), thus an alternative representation has been developed. The basis idea was to provide a more accessible technique consisting in reproducing the visual tree in a way similar to the typical “resource explorer”. This type of technique is probably less effective from a graphical viewpoint, but it is accessible even by people with disabilities using assistive technology, such as screen readers.

The second kind of introduced changes was more substantial and was related to the specification and abstraction process. The basic idea was to guide the user through the guideline abstraction process. For this purpose, the structure of the GAL guideline specification was analysed. As Figure 2 shows, there are four main components: the object of the evaluation, the operator indicating the type of processing, the specific aspect to evaluate, and the condition to check.

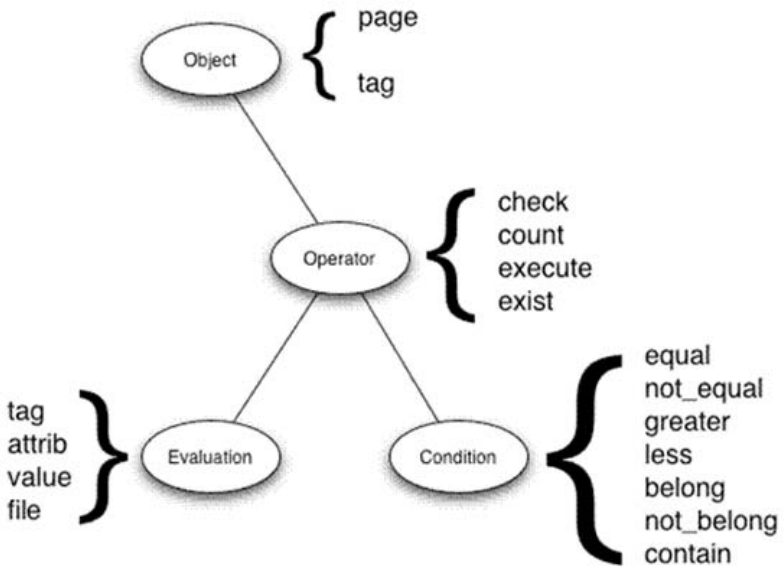

Figure 2. The Structure of a Guideline in GAL.

Thus, we decided to provide support able to guide the designers through dialogue windows. The dialogue window shows the corresponding four main questions to drive the developer in choosing the involved elements, and the possible answers. The main purpose is to lead the developer through the abstraction process by a logic flow. In

this way abstracting should be easier, especially when selecting objects in a correct order.

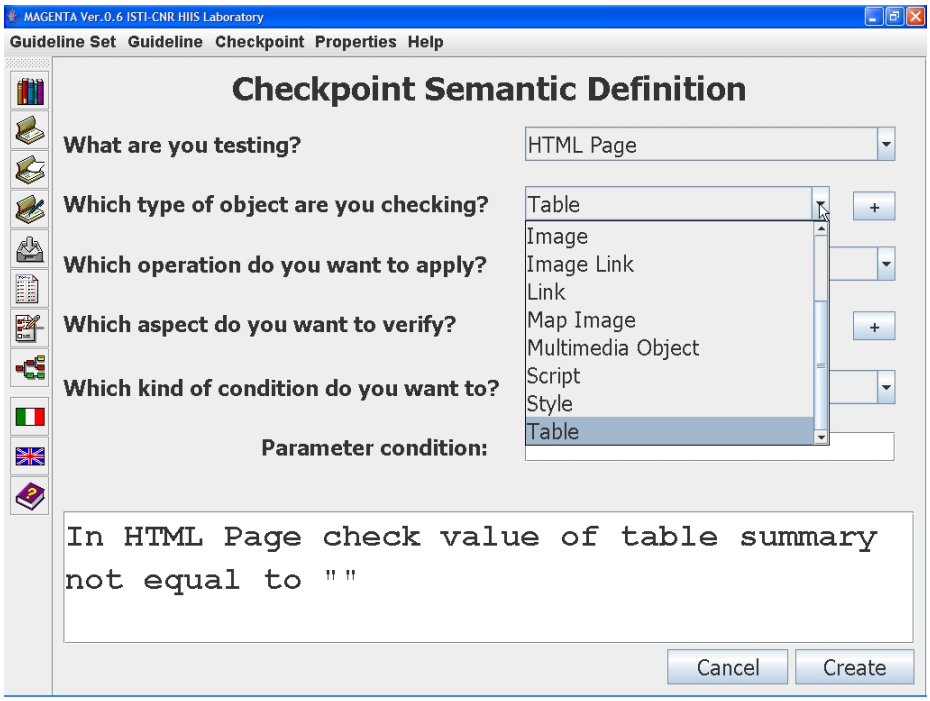

Figure 3. The Editor of the Guideline Textual Description.

As shown in Figure 3 the main questions are:

- What are you testing?
- Which type of object are you checking?
- Which operation do you want to apply?
- Which aspect to do you want to verify?

The possible answers for each question are predefined as well depending on the type of aspect considered. The resulting guideline specified through the answers is also shown in the text area at the bottom.

Consequently, the resulting environment provides two ways to specify the guidelines: the graphical editor and the natural language description obtained through predefined questions. The two possible representations are not independent, thus it is possible to show them at the same time (see an example in Figure 4) and if one representation is modified then the other one is automatically updated to reflect the changes.

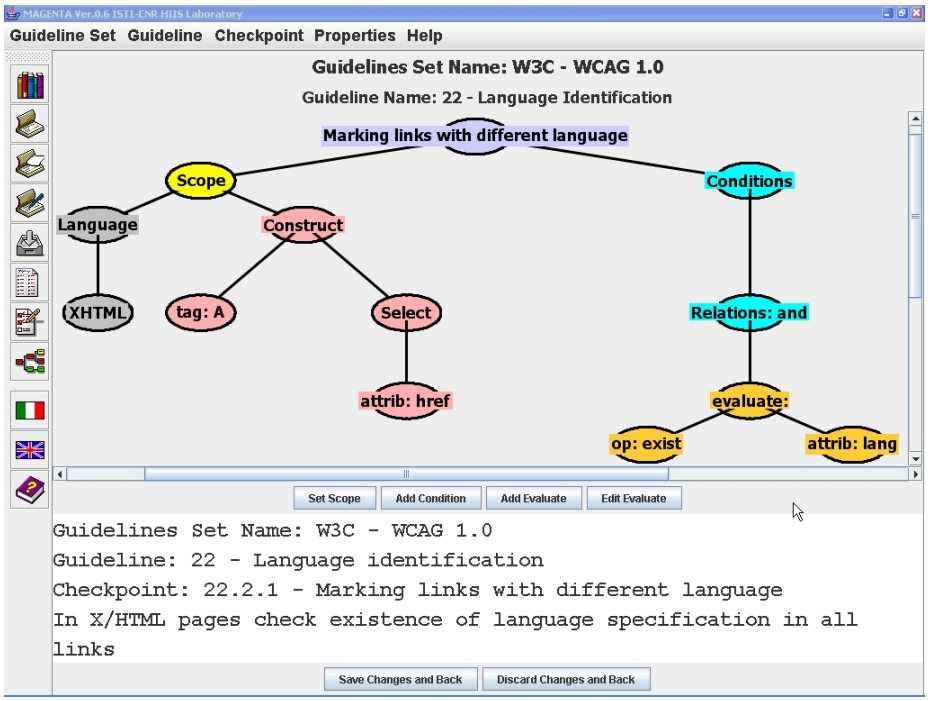

Figure 4. An Example of Two Representations of the Same Guideline.

7. 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 more and more complex. In order to develop a tool independent of guideline definition, guidelines should be specified separately and interpreted at runtime. To this end, we have introduced a Guideline Abstraction Language to define guidelines using an XML structure. These XML-based files can be managed by our tool, which loads and checks any guideline indicated by the developer through this language without requiring modifications to its implementation. Thus, designers can dynamically select and edit the guidelines to check in their Web site. Since abstracting and defining guidelines requires considerable effort, a Guideline Editor has been developed and integrated in our environment to assist evaluators in these activities. The user interface has been designed to guide developers as much as possible in order to facilitate their tasks. Currently, the editor supports guidelines for documents implemented in the X/HTML and CSS languages.

Future work is planned to support other Web languages such as SMIL and XForms. As we have shown in the paper, the last version of our guideline editor supports two possible representations (graphical and textual), with the associated editing techniques. Future work is also planned to better investigate what types of users prefer each guideline representation.

REFERENCES

1. Abascal J., Arrue M., Fajardo I., Garay N., Tomás J., Use of Guidelines to automatically verify Web accessibility. UAIS, Springer Verlag, Vol.3, N.1, 2004, pp. 71-79.
2. Beirekdar, A., Keita, M., Noirhomme-Fraiture, M., Randolet, F., Vanderdonckt, J., Mariage, C., 2005. Flexible Reporting for Automated Usability and Accessibility Evaluation of Web Sites. INTERACT 2005, pp. 281-294
3. Gunderson, J., Rangin, H. B., Hoyt, N. (2006). Functional Web accessibility techniques and tools from the university of Illinois. In Proc. of the 8th international ACM SIGACCESS conference on Computers and accessibility (Assets 2006), Portland, Oregon, USA, October 2006. pp. 269-270
4. Herramhof, S., Petrie, H., Strobbe, C., Vlachogiannis, E., Weimann, K., Weber, G., Velasco, C.A., 2006. Test Case Management Tools for Accessibility Testing. In Proc. of ICCHP'06, pp. 215-222
5. Italian Accessibility Law, 2004: Provisions to support the access to information technologies for the disabled.
6. Kuniavsky, M., Raghavan S., 2005. Common sense and reason: Guidelines are a tool: building a design knowledge management system for programmers. In Proc. of DUX '05
7. Leporini, B., Paternò, F., Scorcia, A. (2006). Flexible tool support for accessibility evaluation. Interacting with Computers, Vol. 18, No. 5, September 2006, pp. 869-890.
8. Lift. http://www.usablenet.com/products_services/lift_online/lift_online.html
9. 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.
10. Mayhew D. J.. Principles and guidelines in software and user interface design. Prentice Hall, Englewood Cliffs, 1992.
11. Noirhomme-Fraiture, M., Beirekdart, A., Vanderdonckt J., Automated Evaluation of Web Usability and Accessibility by Guidelines Review, ICWE 2004 (International Conference on Web Engineering), Lecture Notes in Computer Science 3140, Munich, July 2004, pp. 17-30.
12. Web Content Accessibility Guidelines 2.0, W3C World Wide Web Consortium Recommendation August-September 2006.
13. Web ATRC, http://checker.atrc.utoronto.ca/index.html
14. WebXACT, http://Webxact.watchfire.com/