Journal of Web Engineering, Vol. 4, No.2 (2005) 102-117

AUTOMATED WEB EVALUATION BY GUIDELINE REVIEW

JEAN VANDERDONCKT and ABDO BEIREKDAR
Université catholique de Louvain (UCL), School of Management (IAG)
Unit of Information Systems (ISYS), Belgian Lab. of Computer-Human Interaction
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
{vanderdonckt, beirekdar}@isys.ucl.ac.be
Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique
rue Grandgagnage, 5 – B-5000 Namur, Belgium
abe@info.fundp.ac.be

Received October 18, 2004
Revised March 16, 2005

A novel approach is presented for automatically evaluating of the usability and accessibility (U&A) of web sites by performing a static analysis of their HTML code against U&A guidelines. The approach relies on separating guidelines evaluation logic from the evaluation engine. Due to this separation, the whole evaluation process can be divided into two main phases: specifying formal guidelines and web page evaluation. In the first phase, the formal structure of a guideline is expressed in terms of Guideline Definition Language (GDL). In the second phase, the web page is parsed to identify its contents and structure and link them to relevant guidelines to be evaluated on the page parsed. This approach enables the simultaneous evaluation of multiple guidelines selected on demand from different sources. It also optimises evaluation by automatically identifying common sub-structures among structured guidelines. It also supports the expression, by evaluators with different usability practises, of alternative evaluation strategies.

Key words: usability, accessibility, automated evaluation of web sites, static analysis, guideline definition language, evaluation engine
Communicated by: N Koch & P Fraternali

1 Introduction

The World Wide Web has become a predominant mean for communicating and presenting information on a broad scale and to a wide audience. Unfortunately, web site usability and accessibility continue to be a pressing problem [2]. An estimated portion of 90% of existing sites does not provide adequate usability [3], and an estimated 66% of sites are inaccessible to users with disabilities [4]. Although numerous assistive devices, such as screen readers and special keyboards, facilitate use of Web sites, these devices may not improve a user's ability to find information, purchase products and complete other tasks on sites. For example, sites may not have links to help blind users skip over navigation bars, or sites may not enable users to increase the text font size, so that they can read it. A wide range of Usability and Accessibility (U&A) evaluation techniques have been proposed and a subset of these techniques is currently in common use. Automation of these techniques became much desired [5,6] because they required U&A specialists to conduct them or to analyse evaluation results, which is very resource consuming especially for very large, continuously growing web sites. In addition, there is a lack of experts due to an increased demand. A possible solution to that problem consists in relying on U&A guidelines and recommendations to be reviewed and applied by designers and developers. Some

studies show that applying guidelines by designers is subject to interpretation, basically because of the inappropriate structuring or formulation [7]. For this reason and others, automation has been predominately used to objectively check guideline conformance or review [6]. Several automatic evaluation tools were developed to assist evaluators with guidelines review by automatically detecting and reporting ergonomic deviations from these guidelines and making suggestions for repairing them. In this paper, a novel approach is presented that automate the evaluation of a web site against U&A guidelines by checking a formal representation of these guidelines on the web pages of interest. The aim of the approach is to overcome the major shortcomings of existing tools. The main characteristic of this approach is the separation between the evaluation logic and the evaluation engine. In this way, the U&A guidelines can be expressed in terms of conditions to be satisfied on HTML elements (i.e., tags, attributes). A formal specification language supporting this approach implements a framework [8] that enables the trans-formation of such U&A guidelines from their initial expression in natural language into testable conditions on the HTML code. Once expressed, the guidelines can be evaluated at evaluation-time by configuring their formal expression in an improved way depending on the guidelines to be evaluated and the HTML elements contained in the page. This process consequently considers guidelines relevant to the targeted evaluation context, and factors out sub-structures that are common across these guidelines, even if they come from different sets of guidelines.

This paper is structured as follows: Section 2 briefly describes some automatic U&A evaluation tools. Section 3 presents a global view of the evaluation process and the fundamental concepts of our evaluation approach. Section 4 exemplifies the approach on guidelines which is not found in existing tools. Section 5 underlines the possibilities of evaluation improvement, the most original part of our approach. Sec-tion 6 concludes the paper by stressing major advantages of the proposed approach.

2 Related Work

Depending on the evaluation method, U&A may involve several activities such as:

1. **Capture**: it consists of collecting U&A data, such as task completion time, errors, guideline violations, and subjective ratings.
2. **Analysis**: it is the phase where U&A data are interpreted to identify U&A problems in the web site.
3. **Critique**: it consists of suggesting solutions or improvements to mitigate the previously identified problems.

Many evaluation tools were developed to provide automation of some of the above activities. In this section we would like to report on some tools used for Web evaluation by guideline review, a particular evaluation method that has been selected for its simplicity, its capability to be conducted with or without users, and its wide applicability. Some of the tools are dedicated only to usability, some others to accessibility, but none of them to both U&A:

- A-Prompt [10] is an off-line tool designed to improve the usability of HTML documents by evaluating web pages for accessibility barriers and then providing developers with a fast and easy way to make the necessary repairs. The tool's evaluation and repair checklist is based on accessibility guidelines created and maintained by the Web Accessibility Initiative (http://www.w3.org/WAI).
- Bobby [11] is a comprehensive web accessibility tool designed to help evaluate and repair accessibility errors. It tests for compliance with WCAG1.0 and Section 508 guidelines. It offers prioritised suggestions based on the WCAG1.0. It also allows developers to test web pages and generate summary reports highlighting critical accessibility issues sorted by rank on priority.

- LIFT OnLine (http://www.usablenet.com) tests web pages against a subset of all usability and accessibility guidelines (rules), and then sends an e-mail with the link to a usability report on line. As soon as the analysis is completed, LIFT Online shows a list of the pages in the web site containing potential usability problems. Each problem is ranked by severity and is described in details.
- WebSAT (http://zing.ncsl.nist.gov/WebTools/WebSAT/overview.html) inspects the HTML composition of web pages for potential usability problems. It could perform inspections using either its own set of usability rules or those of the IEEE Std 2001-1999.
- WebCriteria (http://www.webcriteria.com) is a tool for comparative evaluation of a web site with respect to a benchmark derived from similar well-established web sites that are considered as reference.

The common major shortcoming of the above existing U&A evaluation tools is that the evaluation logic is hard coded and hard wired in the evaluation engine, which makes them very inflexible for any modification of the evaluation logic. Introducing a new guideline, possibly a custom one, or modifying an existing guideline remains impossible. In addition, many of them do not offer many possibilities of controlling the evaluation process like choosing which guideline to evaluate, the level of evaluation at evaluation time, or the level of priority. For example, Bobby only provides the choice of the guidelines set to evaluate: W3C or Section508. Only very recent tools begin to investigate how to separate the evaluation logic (containing the evaluation rules) from the evaluation engine, thus making them independent from each other. An example of such a tool is EvalIris [1] which structures guidelines and related evaluation techniques and checkpoints as defined in the case of Web Accessibility Guidelines (WAI) WCAG1.0. However, EvalIris is limited only to those accessibility guidelines and cannot handle other types of guidelines.

3 The Evaluation Approach

To address the above shortcomings, the evaluation process is structured in our approach by decomposing the whole evaluation process into two distinct but related phases (Fig. 1): specifying formal guidelines (which is achieved only once before any evaluation) and evaluating a web site (which is conducted any time an evaluation is performed). The two main phases remain totally autonomous, thus giving many possibilities to improve each of them. The different steps of these phases are now further described.

Structuring. The first step in our approach consists of structuring U&A guidelines in terms of evaluation conditions so as to obtain a formal guideline, expressed in a format that can be processed by an automaton as opposed to the natural language format of the initial guideline found in the literature. Guidelines structuring requires a thorough understanding of both the U&A knowledge and the HTML language to bridge the gap between them. It is also influenced by the understanding of the original guideline semantics that may lead to several interpretations of the same guideline. The formal guideline is expressed according to Guideline Definition Language (GDL), a XML-compliant language supporting the approach by formalising the structuring of U&A guidelines toward automatic evaluation. The most original characteristic of the GDL is its naturalness, i.e. the possibility offered by the language to straightforwardly map the informal statement of initial guidelines onto formal statement expressed in the GDL language. GDL is aimed at modelling only those HTML aspects that could be evaluated such as, but not limited to: colour contrast, alternative text for visual content). Other frameworks have to be used to cope with other direct usability aspects such as user satisfaction,

consistency, and information organisation. Meta-information is then added to each GDL-compliant guideline according to taxonomy on indexes [11].

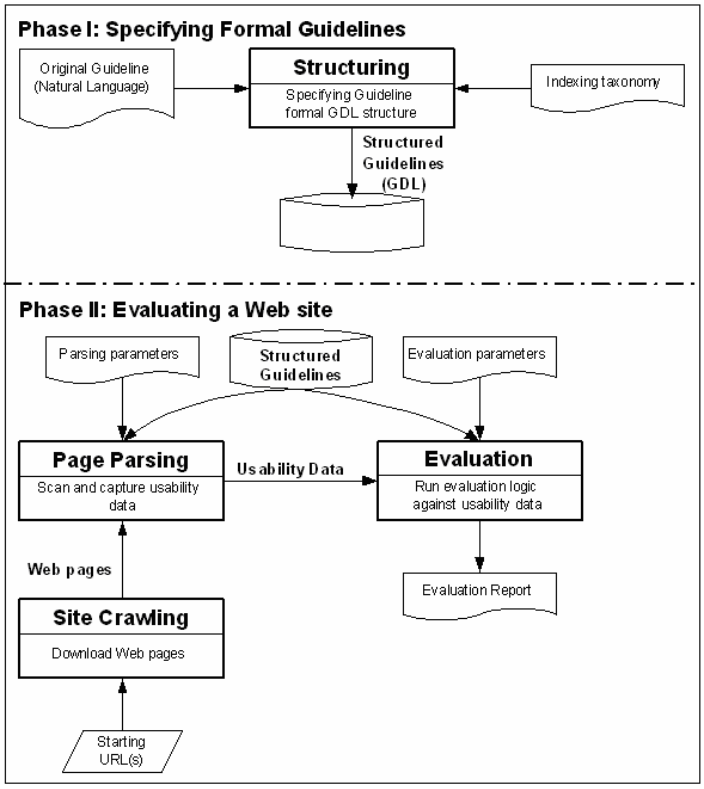

Figure 1 The two main phases of evaluation process based on the proposed approach.

Site crawling. The web pages of interest, whether they are static or dynamically generated, are simply specified by their URL, automatically downloaded from the web and stored in a dedicated directory where the rest of the process will take place.

Page parsing. This step is done by a single scan for each web page to be evaluated. During this scan, the parser captures the instances of the evaluation sets specified in the formal guidelines to be checked. Parsing parameters can be specified to control this process: which guidelines to evaluate, number of desired instances (e.g., a given maximum, all instances), whether incomplete set instances could be kept, etc.

Evaluation. After parsing the web page, the evaluation conditions that have been defined during guidelines structuring in GDL are processed to check their satisfaction by performing a property checking. Every condition is applied on the captured instances of its corresponding set to determine its respect or violation. In this way, a detailed evaluation report can be generated on respected/violated sets, number of detected instances, and percentage of respect/violation.

4 Fundamental Concepts

To facilitate the different phases of the evaluation process, several new concepts had to be identified and defined. Fig. 2 depicts a global view of the fundamental concepts of our approach and their interactions. These concepts form the building blocks of Guideline Definition Language (GDL). The language is based on the ontology of concepts defined in a framework that we proposed to structure, in a systematic and consistent manner, Web guidelines towards automatic evaluation. A specification is composed of the following sections (Fig. 2): the definition of the original guideline, the definition of guideline interpretations if needed, and the definition of the formal guideline (the structure) of the guideline or the formal guidelines of its interpretations. Fig. 3 depicts the XML schema of the GDL. This schema does not show the attributes of the schema elements due to space limitation. According to this schema, if the original guideline could be evaluated as it is, we define its formal form without the need for interpretation; else, we re-express the guideline differently to bring out concrete indicators about how to evaluate it. The formal guideline contains two parts: the structuring of HTML elements judged interesting for the evaluation of the original guideline (*Structure*), and the specification of how to use these elements to conduct the evaluation (*Logic*).

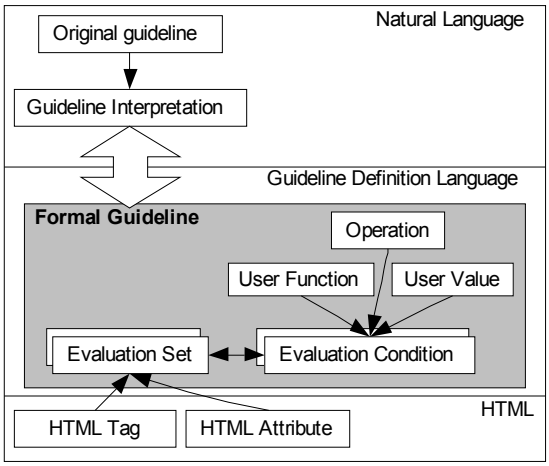

Figure 2 Fundamental Concepts.

The structure starts by listing HTML elements that will be used later in what we call evaluation sets [3]. Every element has an identifier and can be a tag only (e.g., `H1`) or a tag and attribute (e.g., `Table`, `bgcolor`). As mentioned above, our evaluation sets are very similar in their semantics to WAI WCAG1.0 HTML techniques. The original (and most complex) part in a GDL specification is the evaluation logic. Our objective is to provide a set on constructs rich enough to enable the specification of a wide range of evaluation conditions. At the current state of our GDL we can specify evaluation logic using basic data types: Integer, Float, Colour, and String. These types cover the majority of content that can be found in a HTML Web page. For example, String data type allows us to specify conditions like:

```
IMG.alt<>""
```

And String and Integer data types allow us specifying conditions like: Length(IMG.alt)<255. In addition, we defined some constructed data types: Sequence, Table, and Cartesian Product. For example, the Sequence data type allows us to specify conditions like:

Body.bgcolor IN [Red, Black, White]

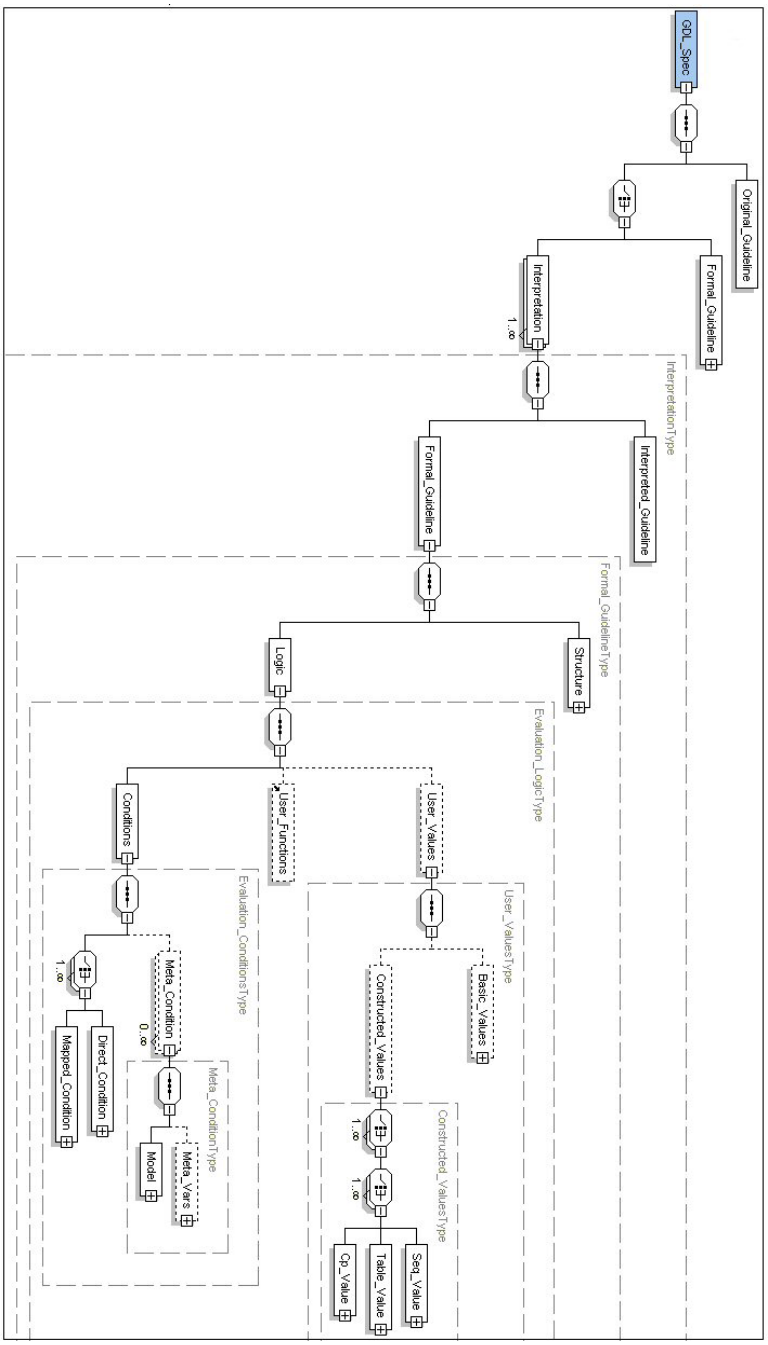

Figure 3 XML Schema for a GDL specification.

Evaluation conditions can be specified directly using elements of evaluation sets (*Direct condition*), or indirectly via a Meta evaluation condition that specify a Meta logic that can be applied on more that one evaluation set. This Meta condition is them mapped to every set as a Mapped condition.

To exemplify how GDL can be exploited to transform a natural language guideline into a formal counterpart, let us select a first guideline: "Use a limited number of fonts in a single web page".

As guidelines are generally expressed at a high level of abstraction, the original guideline expressed in natural language should be transformed into a concrete form that can be understood by designers and manipulated by the system. This re-expression is called an interpretation of the guideline. In general, interpretation is used to limit the focus of the original guideline to some extent that can be considered satisfactory in the targeted evaluation context.

Of course, even with interpretation, evaluation of some guidelines cannot be totally automated. For this reason, every interpretation is assigned to a factor indicating the level of abstraction reflected. In our example, the guideline needs to be interpreted because "limited number" is very abstract. If we have an evaluation logic that can be applied on many evaluation sets with some difference, we can define a meta-evaluation condition for all these sets, then we defined a mapped evaluation condition for every evaluation set. Otherwise, we specify a direct evaluation condition.

In a meta-evaluation condition, we use meta-variables to specify the evaluation logic. The instantiation of the condition for a given evaluation set is done by mapping between the meta-variables and the corresponding concrete set elements. Global conditions (meta or direct) are formed of operations. These operations provide a mechanism for identifying potential common parts among evaluation conditions. They are assigned a priority indicator to avoid ambiguity and to facilitate the execution of global conditions. A direct condition that can be associated to the evaluation set of our example could be NBR_INSTANCES(S1)<4, where NBR_INSTANCES is a predefined GDL function. In the GDL expression below corresponding to the condition, we start by finding the number of set instances captured from the parsed web page, then we test if it is smaller than 4.

```
<Direct_Condition set="S1">
    <Operation id="O1"
        Symbol="NBR_INSTANCES"
        return="number" Order="1">
    <Arg type="Set" value="S1"/>
    </Operation>
    <Operation id="O2"
        Symbol="<" return="Bool" Order="2">
        <Arg type="Operation" value="O1"/>
        <Arg type="number" value="4"/>
    </Operation>
</Direct_Condition>
```

5 Examples with Evaluation Improvement

5.1 Basic examples with meta-condition

The evaluation approach is now applied on two reasonably complex guidelines. The first one is: GDL1="Select colours that will make your page easy to read by people with colour blindness" [11]. As such, GDL1 cannot be automated in a straightforward manner as there is no calculable way to assess to what extent a page is easy to read or not, depending on the users. However, if we refer to the research conducted by Murch [12], an interpretation Inter_GDL1 of this guideline can be produced: "The

combination between background colour and foreground colour should belong to the best colour combinations or should not belong to the worst colour combinations proposed by Murch". This interpretation will not cover all colours because Murch dealt with basic colour only, but we will use it for simplification. Fig. 4 graphically depicts the best colour combinations for thin lines and text, depending on the colour of the background, whereas Fig. 5 depicts the worst colour combination in the same conditions. The first line of Fig. 4 can be read as follows: on a white background, 94% of people do not experience any trouble in reading blue on while. 63 of people do not have any problem with black on white and 25% do not have any problem with red on white. Other colour combinations could be considered, but here only the three first combinations that reached a consensus are reproduced. The other lines can be interpreted by analogy. Similarly, in Fig. 5, the first line means that almost everybody experiences some trouble in reading yellow text on a white background. In other words, nobody says that this is a good colour combination. Cyan on white poses problems to 94% of end users. On a black background, blue is not readable by 87% of end users, followed by red (37%) and magenta (25%).

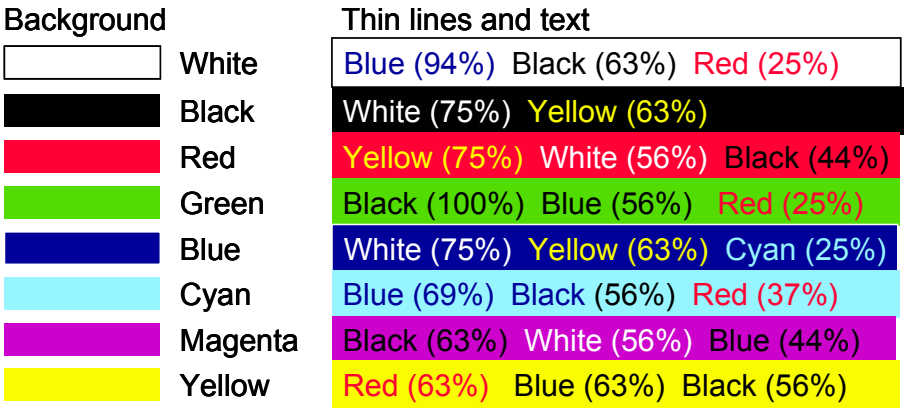

Figure 4 Best colour combinations for thin lines and text according to Murch.

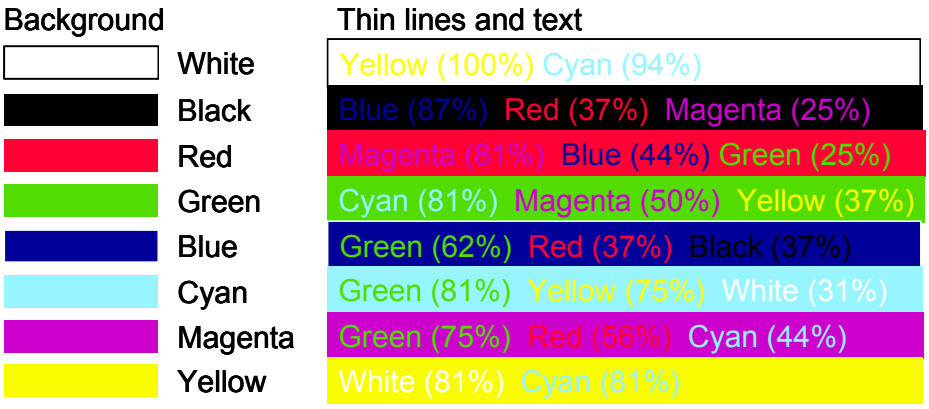

Figure 5 Worst colour combinations for thin lines and text according to Murch.

For Inter_GDL1, we have the following evaluation sets:

- S1controls text color over the whole page. S1={Body.bgcolor^{Page}, Body.text^{Page}}

```
<SET id="S1" name="Global color control" priority="AAA">
  <Element id="E1" tag="Body" Attribute="text" scope="Page"/>
  <Element id="E2" tag="Body" Attribute="bgcolor" scope="Page"/>
</SET>
```

- S2 controls color by Body and Font. S2={Body.bgcolor^{Page}, Font.color$^{Body.\ bgcolor}$}.

```
<SET id="S2" name="Body Font color control" priority="AAA">
  <Element id="E2" tag="Body" Attribute="bgcolor" scope="Page"/>
  <Element id="E3" tag="Font" Attribute="color" scope="E2"/>
</SET>
```

The remaining evaluation sets can be obtained by analogy:

- S3 controls color in Font and Table. S3={Table.bgcolor^{Page}, Font.color$^{Table.bgcolor}$}
- S4: controls color in Font and TH. S4={TH.bgcolor^{Page}, Font.color$^{TH.bgcolor}$}
- S5: controls color in Font and TR. S5={TR.bgcolor^{Page}, Font.color$^{TR.bgcolor}$}
- S6: controls color in Font and TD. S6={TD.bgcolor^{Page}, Font.color$^{TD.bgcolor}$}.
- S7: controls color in Body and TH. S7={TH.bgcolor$^{Body.text}$, Body.text^{Page}}.
- S8: controls color in Body and TR. S8={TR.bgcolor$^{Body.text}$, Body.text^{Page}}.
- S9: controls color in Body and TD. S9={TD.bgcolor$^{Body.text}$, Body.text^{Page}}.
- S10: controls color in Body and Table. S10={Table.bgcolor$^{Body.text}$, Body.text^{Page}}.

According to our experience with HTML 4.0, these sets cover all the possibilities provided by HTML to manipulate colour of normal text (not links). So, we can consider that the evaluation of Inter_GDL1 (and thus GDL1) can be totally automated. The evaluation conditions associated with the above sets are very similar. Thus, we can define a Meta evaluation condition that can instantiated for every evaluation set by mapping the Meta variables to corresponding concrete set elements. The Meta evaluation condition corresponds to the next pseudo specification (PS):

```
(BackgroundColor IN ListOfMurchColors) AND
(ForegroundColor IN ListOfGoodColors(BackgroundColor)) OR
(ForegroundColor NOT IN ListOfBadColors(BackgroundColor))
```

where ListOfGoodColors and ListOfBadColors are two lists of predefined values (colours). As mentioned earlier in this section, we will use Murch's colour combinations. For this purpose, basic colours are defined as user values:

```
<User_V id="Black"     type="Color" val="#000000"/>
<User_V id="White"     type="Color" val="#ffffff"/>
<User_V id="Red"       type="Color" val="#ff0000"/>
<User_V id="Geen"      type="Color" val="#00ff00"/>
<User_V id="Blue"      type="Color" val="#0000ff"/>
<User_V id="Cyan"      type="Color" val="#00ffff"/>
<User_V id="Magenta"   type="Color" val="#ff00ff"/>
<User_V id="Yellow"    type="Color" val="#ffff00"/>
```

Then, we specify lists of values corresponding to Murch colours, good and bad foreground colours for a given background colour:

```
<User_V id="MurchColors" type="Sequence"
        val="Black White Red Green Blue Cyan Magenta Yellow"/>
<User_V id="GoodFgBlackBg" type="Seq" val="White Yellow"/>
<User_V id="BadFgBlackBg"
        type="Sequence" val="Blue Red Magenta"/>
<User_V id="GoodFgWhiteBg"
        type="Sequence" val="Blue BlackRed"/>
<User_V id="BadFgWhiteBg"
```

```
            type=”Sequence” val=”Yellow Cyan”/>
<User_V id=?GoodFg<b>Red</b>Bg?
            type=”Sequence” val=”White Yellow”/>
<User_V id=”BadFgRedBg”
            type=”Sequence” val=?Magenta Blue”/>
<User_V id=”GoodFg<b>Green</b>Bg”
            type=”Sequence” val=”Black Blue Red”/>
<User_V id=”BadFgGreenBg”
            type=”Sequence” val=”Magenta Cyan”/>
<User_V id=”GoodFg<b>Blue</b>Bg”
            type=”Sequence” val=”Green Red”/>
<User_V id=”BadFgBlueBg”
            type=”Sequence” val=”Red Magenta”/>
<User_V id=”GoodFg<b>Cyan</b>Bg”
            type=”Sequence” val=”White Yellow”/>
<User_V id=”BadFgCyanBg”
            type=”Sequence” val=”Yellow White”/>
<User_V id=”GoodFg<b>Magenta</b>Bg”
            type=”Sequence” val=”White Black”/>
<User_V id=”BadFgMagentaBg”
            type=”Sequence” val=”Green Red”/>
<User_V id=”GoodFg<b>Yellow</b>Bg”
            type=”Sequence” val=”Black Blue”/>
<User_V id=”BadFgYellowBg”
            type=”Sequence” val=”White Cyan”/>
```

Then, a meta condition can be defined corresponding to the above pseudo specification (PS). One will easily notice that specification of conditions is relatively long. This is due to our desire to provide a GDL specification language that is as flexible and rich as possible.

```
<Meta_Condition MC_ID="MurchModel">
    <Meta_Vars>
        <Meta_Var Name="BgColor" Type="Color"/>
        <Meta_Var Name="FgColor" Type="Color"/>
    </Meta_Vars>
    <Model Expression=””>
        <Operation  id=”O1” Symbol=”IN” return=”Bool” Order=”1”
                    Stop_Val=”False” Stop_Msg=”Unrecognized Background color.”>
            <Arg type=”Var” value=”BgColor” Pos=”1”>
            <Arg type=”value” value=”MurchColors” Pos=”2”>
        </Operation>
        <Operation  id=”O2” Symbol=”IN” return=”Bool” Order=”2”
                    Stop_Val=”True” Stop_Msg=”Good color combination.”>
            <Arg type=”Var” value=”FgColor” Pos=”1”>
            <Arg type=”value” value=”GoodFg(BgColor)” Pos=”2”>
        </Operation>
        <Operation  id=”O3” Symbol=”NOT IN” return=”Bool” Order=”3”
                    Stop_Val=”True” Stop_Msg=”Not bad color combination.”>
            <Arg type=”Var” value=”FgColor” Pos=”1”>
            <Arg type=”value” value=”BadFg(BgColor)” Pos=”2”>
        </Operation>
        <Operation id=”O4” Symbol=”OR” return=”Bool” Order=”3”
                Stop_Val=”True” Stop_Msg=”Good color combination.” Stop_Val=”False”>
            <Arg type=”Op” value=”O2” Pos=”1”>
            <Arg type=”Op” value=”O3” Pos=”2”>
        </Operation>
        <Operation id=”O5” Symbol=”AND” return=”Bool” priority=”3”>
            <Arg type=”Op” value=”O1”>
            <Arg type=”Op” value=”O4”>
        <Operation>
    </Model>
</Meta_Condition>
```

Notice that the above specification is long because we wanted to have complete control over the execution by using the concept of Stop Value that allows stopping the execution after any operation if

its result corresponds to a given value. In this way, we can generate highly customised output messages via the Stop_Msg. The evaluation expression can also be specified as a single piece of text (the Expression attribute of the Model element) to be interpreted by the evaluation engine at execution time. This way is clearer and shorter than the above way, but the specified expression must respect a predefined syntax to be correctly interpreted. After defining the meta-condition we instantiate it on every evaluation set via the mapping rules.

```
<Mapped_Condition Set_ID="S1" Meta_ID="MurchModel">
    <Meta_Mapping Meta="BgColor" Instance="E1"/>
    <Meta_Mapping Meta="FgColor" Instance="E2"/>
</Mapped_Condition>
<Mapped_Condition Set_ID="S2" Meta_ID="MurchModel">
    <Meta_Mapping Meta="BgColor" Instance="E1"/>
    <Meta_Mapping Meta="FgColor" Instance="E3"/>
</Mapped_Condition>
```

Other mappings can be defined similarly for the remaining sets. Let us know consider another guideline oriented toward accessibility: "Web pages shall be designed so that all information conveyed with color is also available without color" (Section 508). GDL needs to be interpreted as well. As the guideline suggests, information conveyed by color can be conveyed using markup. We will consider the following markup tags: Bold <b>, Italic <i>, text size <Font.size> and text font <Font.face>. Thus, the interpretation of GDL2 could become Inter_GDL2: "Web pages shall be designed so that all information conveyed with color is also available using any combination of the above markup elements". This means that, in our evaluation context, Inter_GDL2 is considered violated even if colored information was conveyed using other means than the above markup tags. For Inter_GDL2, we have the following evaluation sets:

- S1A: conveying colored information using bold tag.
```
<SET id="S1A" name="bold conveyance" Priority="AAA">
        <Element id="E1" tag="Body" Attribute="bgcolor" scope="Page"/>
        <Element id="E2" tag="Font" Attribute="color" scope="E1"/>
        <Element id="E3" tag="b" Attribute="" scope="E2"/>
</SET>
```
- S1B: conveying colored information using bold tag.
```
<SET id="S1B" name="bold conveyance" Priority="AAA">
        <Element id="E1" tag="Body" Attribute="bgcolor" scope="Page"/>
        <Element id="E3" tag="b" Attribute="" scope="E1"/>
        <Element id="E2" tag="Font" Attribute="color" scope="E3"/>
</SET>
```
- S2A: conveying colored information using italic tag.
```
<SET id="S2A" name="italic conveyance" Priority="AAA">
        <Element id="E1" tag="Body" Attribute="bgcolor" scope="Page"/>
        <Element id="E2" tag="Font" Attribute="color" scope="E1"/>
        <Element id="E4" tag="i" Attribute="" scope="E2"/>
</SET>
```
- S2B: conveying colored information using italic tag.
```
<SET id="S2B" name="italic conveyance" Priority="AAA">
        <Element id="E1" tag="Body" Attribute="bgcolor" scope="Page"/>
        <Element id="E4" tag="i" Attribute="" scope="E1"/>
        <Element id="E2" tag="Font" Attribute="color" scope="E4"/>
</SET>
```
- S3: conveying colored information using font face.
```
<SET id="S3" name="font face conveyance" Priority="AAA">
        <Element id="E1" tag="Body" Attribute="bgcolor" scope="Page"/>
        <Element id="E2" tag="Font" Attribute="color" scope="E1"/>
```

```
        <Element id="E5" tag="Font" Attribute="face" scope="E2"/>
    </SET>
```

- S4: conveying colored information using font size.

```
    <SET id="S4" name="font size conveyance" Priority="AAA">
        <Element id="E1" tag="Body" Attribute="bgcolor" scope="Page"/>
        <Element id="E2" tag="Font" Attribute="color" scope="E1"/>
        <Element id="E6" tag="Font" Attribute="size" scope="E2"/>
    </SET>
```

Notice that we defined S1A and S1B to cover the case of bold tag. We needed to do so because the two expressions <Font color="...">Colored bold text</Font> and <Font color="...">Colored bold text</Font> give the same visual result. We did the same thing for italic tag. To specify the evaluation conditions, we can see that the evaluation logic is similar in all the above sets and it corresponds to the following pseudo condition: EXIST(FgColor, bgColor, Alternative) where alternative can be one of the HTML tags: b, I, Font.face, and Font.size. The XML form of this condition would be:

```
<Meta_Condition MC_ID="ColoredInfoModel">
    <Meta_Vars>
        <Meta_Var Name="BgColor" Type="Color"/>
        <Meta_Var Name="FgColor" Type="Color"/>
        <Meta_Var Name="Alternative" Type="HTML_Elem"/>
    </Meta_Vars>
    <Model Expression="Exist(FgColor, BgColor, Alternative) AND IN(Alternative, AltList)"/>
</Meta_Condition>
```

where EXIST is a predefined GDL function. AltList is the user value given as follows:

```
<User_V id="AltList" type="Sequence" val="b I Font.face Font.size"/>
```

Notice that the evaluation expression is provided a non structured text. As mentioned earlier, this is very simple but requires that the specified text respects the GDL syntax for text evaluation expression. After defining the Meta condition we instantiate it on every evaluation set via the mapping rules. We specify similar mappings for the remaining sets.

```
<Mapped_Condition Set_ID="S1A" Meta_ID="ColoredInfoModel">
    <Meta_Mapping Meta="BgColor" Instance="E1"/>
    <Meta_Mapping Meta="FgColor" Instance="E2"/>
    <Meta_Mapping Meta="Alternative" Instance="E3"/>
</Mapped_Condition>
<Mapped_Condition Set_ID="S1B" Meta_ID=" ColoredInfoModel ">
    <Meta_Mapping Meta="BgColor" Instance="E1"/>
    <Meta_Mapping Meta="FgColor" Instance="E2"/>
    <Meta_Mapping Meta="Alternative" Instance="E3"/>
</Mapped_Condition>
```

5.2 Evaluation Optimisation

Decomposing the evaluation process into independent steps in the phases as described above offers the possibility for optimising evaluation at each of them.

Structuring step. As parsing web pages is based on evaluation sets and evaluation conditions defined in this step, we can optimise the evaluation at two levels: for a single guideline, there are two ways: identifying the minimum ensemble of sets needed to evaluate the targeted guideline, and expressing conditions in the most forward way to minimise the number of operations that evaluation engine would need to execute them. At the level of many guidelines, we can optimise evaluation by identifying common structures or sub-structures. This optimisation cannot be neglected since guidelines are expressed at a high abstraction level, and as they come from different sources, it is very possible to have guidelines that are totally or partially semantically identical.

Parsing step. The first significant optimisation at this step is the use of the concept of *exclusion* among evaluation sets. By definition, one evaluation set *Excludes* one (many) other evaluation set(s) if its presence excludes its evaluation. This concept is based on the *Scope* concept related to HTML elements (tags and attributes). Generally, the excluding set has an element whose scope is within the scope of an element of the excluded set. Of course, these two elements must have the same rendering effect. For example (Fig. 6), in the context of text color evaluation, a set containing the attribute Table.bgcolor (like S1={Body.text, Table.bgcolor}) excludes a set containing the attribute Body.bgcolor (like S2={Body.bgcolor, Body.text}), because the scope of Table.bgcolor is within the scope of Body.bgcolor. The second optimisation is to combine parsing and evaluation in one step. This means that an evaluation condition is triggered as soon as an instance of the associated evaluation set is completely detected in the evaluated web page. This combination would be optional because, in some situations like the need for a detailed evaluation report, it is desired to capture all instances of evaluation sets (even non completed or negative ones).

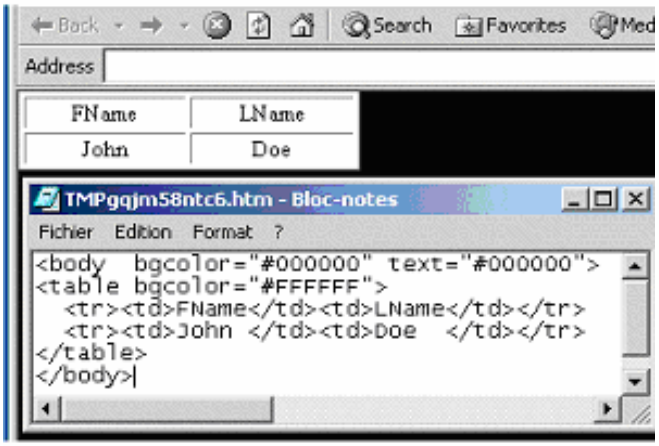

Figure 6 Scope of Table.bgcolor is within the scope of Body.bgcolor

Evaluation step. The optimisation that can be done at this step relies mainly on optimising the execution of evaluation conditions. We introduced the concept of *basic condition* to identify similar or identical parts of evaluation conditions. For example, in Inter_GDL 1 of our example, the evaluation conditions for S1 (Cond1) and S2 (Cond2) are very similar (same Body.bgcolor). As Cond1 will be executed before Cond2, the results of executing Cond1 can be kept if another instance of Cond2 is met with Font.color having the same value of Body.text. Fig. 4 shows how multiple instances of foreground and background colors can be evaluated positively (the color combination is one of the recommended ones), negatively (the colour combination is not recommended at all), or unknown (the colour combination does not belong to any recognised colour pattern). In Fig. 7, we can observe that

- In instance !1, the colour combination does not belong neither to the set of good colours nor to the set of bad colours. Therefore, no conclusion can be made for this instance. Table.bgcolor has no effect because TD.bgcolor overcomes it, but we captured Instance1 of Set3 because we did not consider TD in our evaluation sets. This is very easy to repair by modifying the formal structure.

- In instance !2, the colour combination is reported to belong to the set of good colour combinations. Therefore, it is considered as respected and the guidelines is checked for this case.
- In instance !3, Set2 excludes Set1 and Set3, thus the instance of Font.color did not cause the creation of instances of Set1 or Set3. The resulting instance is said to be in the set of bad colour combinations.

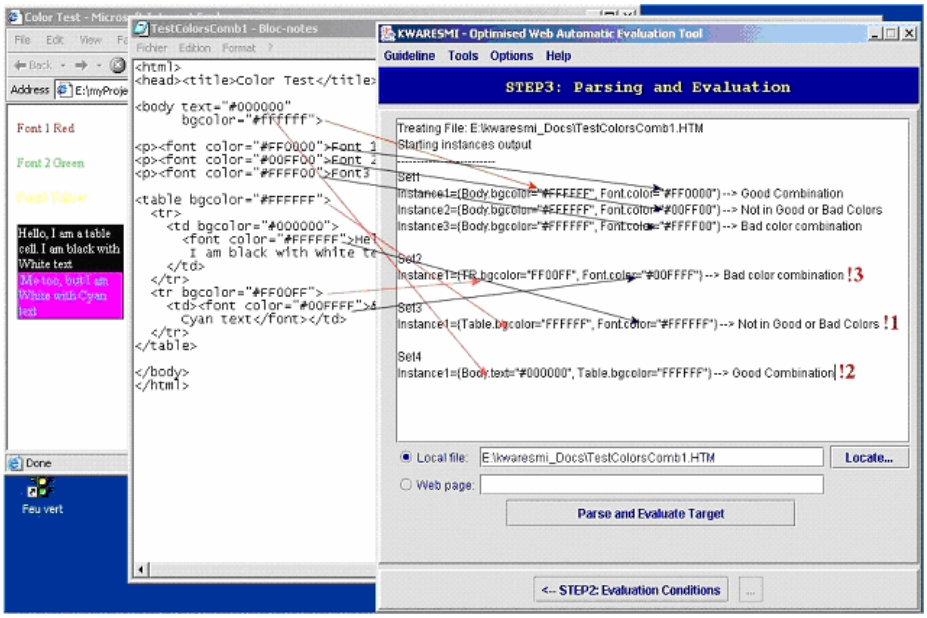

Figure 7 Example of evaluation report

6 CONCLUSION

This paper presented an approach for optimising automated evaluation of Web U&A guidelines based on the concepts of evaluation sets and conditions. This approach would present some advantages over approaches adopted by existing U&A evaluation tools:

- **Targeted guidelines**: traditional evaluation tools cannot evaluate any guideline outside the precompiled set of guidelines hard coded in the evaluation engine of the tool. As for a tool adopting our evaluation approach, its main distinctive feature is its capability to enable the evaluation of any evaluable guideline. A guideline is said to be *evaluable* if we can find HTML elements that reflect its semantics (e.g., the foreground and background colors) and if we can specify the needed evaluation conditions using the vocabularies provided by the evaluation tool. Thus, such a tool should at least be capable of evaluating guidelines that are evaluable by existing tools.
- **Improvement of the evaluation process**: using the same methodological framework to structure all guidelines enables us to obtain non conflicting structures: the structuring would show common evaluation sets and common evaluation conditions if some exist. Partially similar or conflicting guidelines can be identified as well. In this way, no guideline is evaluated twice and no evaluation

condition will be checked more than once. We can also combine parsing and evaluation steps to stop evaluating a guideline if one of its evaluation conditions is not verified. This is useful for guideline checking, where there is only a need to know whether the guideline is verified, as opposed to guideline checking, where we want to know to what extent a guideline is respected.

- **Flexibility of the evaluation process**: separating evaluation logic from evaluation engine in independent phases gives many new evaluation possibilities like choosing to evaluate a part of a guideline by using a sub-set of its evaluation sets, choosing to evaluate particular HTML elements (e.g., images, tables) by selecting guidelines that have these elements in one of their evaluation sets, etc.
- **Customisation of evaluation reports**: the flexibility of our approach should allow us to generate a custom evaluation report (e.g., a possible simple format is given in Fig. 4). In addition to traditional guideline-based evaluation reports generated by existing evaluation tools, we should be able to generate reports based on objects (images, fonts), customised error messages, etc.
- **Identification of conflicts and similarities among guidelines**: expressing guidelines in a logical and structured form allows us to identify potential conflicts and/or common elements among guidelines.
- **Guidelines Management**: at anytime, a guideline can be added, removed or modified without consequence on the evaluation engine. This independence allows the evaluation system to import new sets of guidelines from outside into the tool repository.

After applying the GDL on guidelines with different automation level, we can underline the following advantages:

- HTML elements, evaluation set, operations in evaluation conditions, etc., in addition to other information like stop values and messages, enable us to practically have good control of the evaluation process. An important and direct result of this flexibility is the ability to provide very customizable evaluation reports.
- The above application example gives a first prove of the feasibility of the approach and the ability to evaluate complex guidelines.
- Evaluators need to have detailed knowledge about the HTML elements and conditions that need to be evaluated for each guideline. However, provided that an expert structures and verifies the correctness of guidelines, the structured guidelines can be used broadly by other evaluators. This is facilitated by using XML to specify structures.
- Meta evaluation conditions and operations provide a powerful mechanism to optimise evaluation of relatively similar guidelines. This optimization is very likely to be exploited because existing well established guideline sources (like W3C, ISO, and Section508) are composed of very similar guidelines, and it is generally de-sired to consider all these sources in order to have sites of high ergonomic rating.
- In addition to the possibility to conduct an evaluation by guideline or by Web page as traditionally done in existing U&A tools, a GDL-based tool will be able to conduct an evaluation by HTML object (like tables, fonts, etc.). This can be done at tag level (table, font, etc.) or attribute level (table width, font color, etc.).

References

1. Abascal, J., Arrue, M., Fajardo I., Garay, N., Tomas, J. Use of Guidelines to automatically verify web accessibility. Universal Access in the Information Society, 3(1), 2004, 71–79.
2. ATRC, A-Prompt: Web Accessibility Verifier. Adaptive Technology Resource Center (University of Toronto) and Trace Center (University of Wisconsin), Canada & USA. On-line at http://www.snow.utoronto.ca.
3. Beirekdar, A., Vanderdonckt, J. and Noirhomme-Fraiture, M.: A Framework and a Language for Usability Automatic Evaluation of Web Sites by Static Analysis of HTML Source Code. in Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, 15-17 May 2002). Kluwer Academics Pub., 2002, 337–348.
4. Brajnik, G.: Automatic Web Usability Evaluation: Where is the Limit? in Proc. of the 6th Conf. on Human Factors & the Web (Austin, 19 June 2000). University of Texas, Austin, 2000. On-line at http://www.tri.sbc.com/hfweb/ brajnik/hfweb-brajnik.html
5. Cooper, M., Evaluating Accessibility and Usability of Web Sites. in Proc. of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99 (Louvain-la-Neuve, 21-23 October 1999). Kluwer Academics Publisher, 1999, 33–42.
6. Forrester Research: Why most web sites fail. 1999. On-line at http://www. forrester.com/Research/ReportExcerpt/0,1082,1285,00.html
7. Ivory, M.Y. and Hearst, M.A., State of The Art in Automating Usability Evaluation of User Interfaces. ACM Computing Surveys, 2001.
8. Ivory, M.Y., Mankoff, J. and Le, A., Using Automated Tools to Improve Web Site Usage by Users With Diverse Abilities. IT&Society, 1(3). 195–236. On-line at http://www.stanford.edu/ group/siqss/itandsociety/v01i03/v01i03a11.pdf
9. Jackson-Sanborn, E., Odess-Harnish, K. and Warren, N., Website Accessibility: A Study of ADA Compliance. Technical Report TR-2001-05. School of Information and Library Science, University of North Carolina at Chapel Hill, 2001. On-line at http://ils.unc.edu/ils/research/reports/accessibility.pdf
10. Murch, G.M., Colour Graphics - Blessing or Ballyhoo? The Visual Channel. in Baecker, R.M. and Buxton, W.A.S. eds. Readings in Human-Computer Interaction - A Multidisciplinary Approach. Morgan Kaufmann Publishers, 1987, 333–341.
11. Scapin, D., Leulier, C., Vanderdonckt, J., Mariage, C., Bastien, C., Farenc, C., Palanque, P. and Bastide, R., A Framework for Organizing Web Usability Guidelines. in Kortum, Ph. and Kudzinger, E. eds. Proc. of the 6th Conf. on Human Factors & the Web (Austin, 19 June 2000). University of Texas, Austin, 2000. On-line at http://www.tri.sbc.com/hfWeb/scapin/Scapin.html
12. Vanderheiden, G.C., Chisholm, W.A., Ewers, N. and Dunphy, S.M., Unified Web Site Accessibility Guidelines, Version 7.2, Trace Center, University of Wisconsin-Madison, 1997. On-line at http://trace.wisc.edu/text/guidelns/ htmlgide/htmlgide.htm