DOCTOR OF SCIENCES

Methodology for automating web usability and accessibility evaluation by guideline

Beirekdar, Abdo

Award date:
2004

Awarding institution:
University of Namur

Facultés Universitaires
Notre-Dame de la Paix

A Methodology for Automating Guideline Review of Web Sites

Abdo Beirekdar

Thesis submitted in fulfillment of the requirements for the degree of Doctor of Sciences
(Computer Science Option)

- August 30th, 2004 -

Director:	Professor M. Noirhomme-Fraiture
Co-director:	Professor J. Vanderdonckt, Université Catholique de Louvain, Belgium
Jury:	Professor F. Bodart
	Professor J.-L. Hainaut (President)
	Professor Ch. Kolski, Université de Valenciennes, France
	Professor Ph. Palanque, Université Paul Sabatier - Toulouse III, France

Institut d'Informatique
NAMUR

Chapter 4

A Framework for Evaluation-Oriented Structuring of Web Guidelines

4.1 Introduction

U&A required for use of web sites are today widely recognized as an important requirement for user acceptance. However, despite the fact that U&A guidelines have been proved useful, they still suffer from a series of shortcomings that impede their use and significantly reduce their scope.

The level of guideline expressiveness and the confidence in applying guidelines heavily depends on the source the guidelines come from [Scapin et al. 2000]. Figure 4.1 depicts that guidelines found in the five types of sources range from general guidelines requiring an abstract interpretation to a specific guideline only needed concrete interpretation. The more a guideline is general, the more their applicability domain is wide and the more their interpretation becomes abstract.

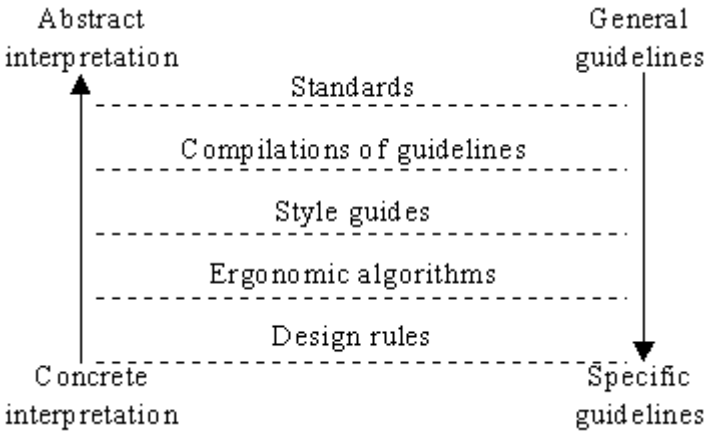


Figure 4.1: Location of sources containing guidelines.

As a consequence, general guidelines cannot be applied per se, thus requiring some concrete interpretation for the intended context of use. On one hand, the format of general guidelines can withdraw experimental conditions under which the guideline has been tested and validated. On the other hand, the lack of these conditions, which are required to ensure a correct interpretation, may invalidate any such interpretation. Specific guidelines no longer require such interpretation, but are so specific that they prevent designers to apply them in other situations without any risk of invalidity. General guidelines are difficult to interpret when and how they need to be applied at design time or evaluated at execution time.

The impact of the above shortcomings is varying according to the goals for which guidelines are considered. If developing a tool for automated or computer-aided evaluation of these guidelines is the ultimate goal, then these shortcomings are important and hard to solve. If providing people with assistance and guidance in applying and evaluating guidelines is the ultimate goal, then these shortcomings are less important.

4.1.1 Requirements

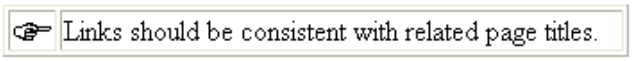
To address the above shortcomings, the need for organizing guidelines into a **practical framework** that would in turn facilitate the structuring and the operationalization of guidelines rapidly emerged.

4.1.2 Related works

Here we will primarily focus on works that have been applied within the domain of automated evaluation of Web guidelines.

Informal classification

[Scapin et al. 2000] proposed a framework covering the whole activities related to the problem: guidelines collection, guidelines organization, and the incorporation of guidelines into approach [Vanderdonckt 1999]. Figure 4.2 shows an example of applying the framework on a Web guideline.



According to the framework, this guideline could be formalized as follows:

```
2.1.2.5 Spec. C.E.-Guidance
2.1.2.5.1.3.2 Identification/Title
2.1.2.5.1.3.2.1 Windows: title = name of task
3.1 Conc. C.E.-Consistency
3.1.5. Labels
3.1.5.1 Links: label = title
```

Figure 4.2: Application of the framework proposed by [Scapin et al. 2000]

The formalization is read as: two situations for testing this guideline may occur: either the link label is exactly the related page title or at least one difference exists. The automation is straightforward in the first situation, while impossible in the second. The level of automation is consequently semi-automatic.

This framework seems difficult to apply because it poses the same problem of interpreting a general guideline in a classification tree that has up to six levels. In addition, it does not provide any kind of formal support to uniformly represent the information that guidelines may involve and to communicate how to apply them.

Another attempt is the well-known WAI of W3C. The WAI proposed fourteen Web Content Accessibility Guidelines (WCAG1.0) that were all defined in systematic manner in order to facilitate their (automated) evaluation. A guideline definition includes:

- The guideline number.
- The statement of the guideline.

- The rationale behind the guideline and some users groups who benefit from it.
- A list of checkpoint definitions.

The checkpoint definitions in each guideline explain how the guideline applies in typical content development scenarios. Each checkpoint definition includes:

- The checkpoint number.
- The statement of the checkpoint.
- The priority of the checkpoint. Priority can be 1, 2 or 3.
- Optional informative notes, clarifying examples, and cross references to related guidelines or checkpoints.
- A list of techniques to precise the way of evaluating the checkpoint.

Each checkpoint is intended to be specific enough so that someone reviewing a page or site may verify that the checkpoint has been satisfied.

This organization of guidelines aims to facilitate their operationalization. The objective is partially achieved because the techniques show how to evaluate these guidelines concretely (provide a text equivalent for every non-text element via "alt", "longdesc", or in element content) for some of them or by giving some hints for others (mark up lists and list items properly), but there is no formalism supporting this organization.

XML support for guidelines classification and evaluation

Thus, it seems that what we need is to add a formal support to a WAI like framework. XML is a good candidate because it allows for extremely large flexibility when describing data format. In addition, one of its main strengths is its suitability for describing structured data. Another advantage of using XML compliant formalism is that we could integrate it with EARL of W3C [W3C 2002].

Many research activities were launched in this direction, especially with the general trend of using XML as an information-structuring format in many fields. Evallris [Abascal et al. 2003] uses XML to structure guidelines and related evaluation techniques and checkpoints as defined in the case of WAI WCAG1.0. Currently, the translation of accessibility guidelines into the proposed XML schema (figure 4.3) is done manually. The designer that performs this translation must master XML and HTML.

According to [Abascal et al. 2003], this structure is valid for most recognized accessibility guidelines. As seen in Figure 2.5, each guideline contains a list of checkpoints expressed in HTML. Each checkpoint is defined by its tag in HTML and, when the affected attribute is needed. For instance, if a checkpoint referring to "*provide a description for the information displayed in a table*" is about to be formatted, in addition to the HTML tag <TABLE>, it is necessary to include summary attributes in the definition.

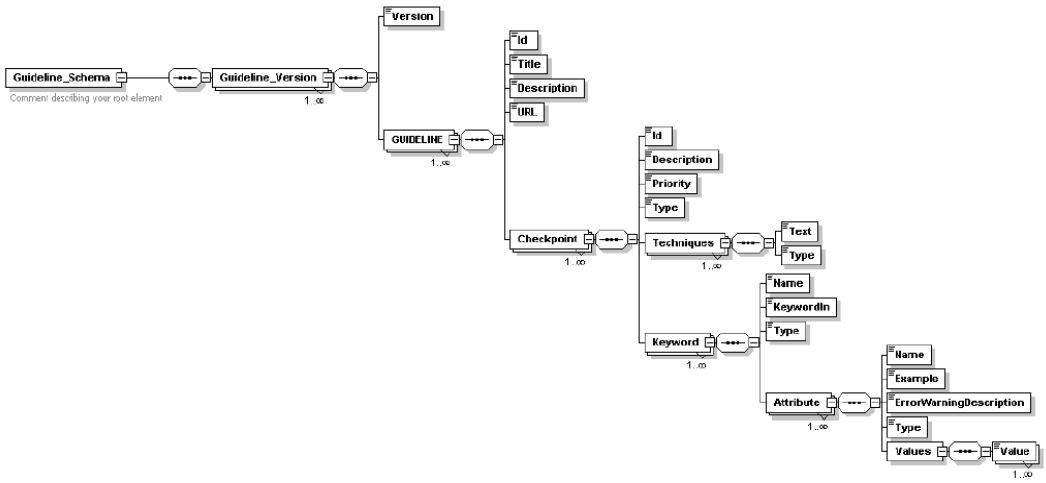


Figure 4.3: XML Schema for WAI guidelines as defined in Evallris

The elements of the proposed accessibility schema are the following:

- *Information regarding the guideline*: identification number, title, description and the URL where the definition of this guideline can be found.
- *Information regarding each checkpoint in the guideline*: identification number, description and priority.

Next is an example of formatting a guideline with Evallris [Abascal et al. 2003].

```
<GUIDELINE>
<Id>X</Id>
        <Title> Do not allow images as web page background.</Title>
        <Description>
           Ensure that images are not defined as the background of a web page as they can make
           the readability of the text difficult.
        </Description>
        <URL>http://XXXXXXXXXXX</URL>
<Checkpoint>
        <Id>X</Id>
        <Description> Do not allow images as web page background</Description>
        <Priority>3</Priority>
        <Type>1</Type>
        <Techniques>
                <Text/>
                <Type/>
        </Techniques>
        <Keyword>
                <Name>BODY</Name>
                <Type>0</Type>
                <Attribute>
                        <Name>BACKGROUND</Name>
                        <Example></Example>
                        <ErrorWarningDescription/>
                        <Type>0</Type>
                        <Values> <Value/> </Values>
                </Attribute>
        </Keyword>
```

```
</Checkpoint>
-----------------------------------------------------------
```

In fact, this structure is a step forward in the direction of automated evaluation. It identifies the HTML elements that must be tracked in a Web page to review the targeted guideline.

Another attempt in the same direction is the Simple Guideline Specification Language (SGSL) [Takata et al. 2003]. Similar to currently available tools, SGSL is designed so that one can specify checkpoints based on the syntax of a document (e.g., "Provide the ALT attribute for every IMG element"), leaving a human user to verify the document with respect to checkpoints such as "Use clearest and simplest language appropriate for a site's content". SGSL is a first-order language which includes XPath [W3C 1999a] as first-order atomic predicates. SGSL aims at automatically verifying a given XML document with respect to given guidelines written in SGSL: Guidelines in SGSL are compiled into an XSLT style sheet [W3C 1999b], and verification of the XML document is performed by an arbitrary XSLT processor with the compiled XSLT style sheet. Specification using the SGSL is not limited to accessibility guidelines. For example, it can be used for verifying whether a given official document (e.g., a grant application form) complies with requirements such as: "entry A is mandatory" and "applicant name in page one should be the same as project organizer in page six". The SGSL defines guidelines as a finite set of checkpoints. A checkpoint is specified as a condition on some element in a XML document. For example, to verify the guideline "If ALT text >150 characters, consider providing a separate description", we specify the following SGSL checkpoint:

```
-----------------------------------------------------------
if some $x in //*/@alt satisfies
        string-length($x)>150
then warn("consider providing a separate description")
-----------------------------------------------------------
```

In fact, the main focus of the SGSL is on the evaluation logic of guidelines rather than on the organization of a set of guidelines. A SGSL specification contains a minimum of information about the specified guideline:

```
-----------------------------------------------------------
<guideline id="1">
        <title>Provide alternative text for all images.</title>
        <expression>
                <condition>
                  <every-expression variable="x" select="//html:img">
                        <xpath-expression select="$x/@alt"/>
                  </every-expression>
                </condition>
        </expression>
</guideline>
-----------------------------------------------------------
```

4.1.3 Our framework

Every one of the mentioned attempts focuses partially on the problem: Evallris does not pay big attention to evaluation logic, whereas the SGSL pays almost no attention on guidelines organization. In addition, both approaches deal with a guideline as it is, without any attention to the possibility of adapting its evaluation to multiple contexts of using a Web site (information seek, learning, targeted users, etc.).

Our framework aims to address these issues [Beirekdar et al. 2002]. Its main goals are:

Systematic and consistent structuring of guidelines towards automated evaluation

The framework is intended to help the evaluator to systematically and consistently structure Web guidelines to facilitate their automated evaluation. To do this, the specified structure should:

- Contain a maximum of information about the guidelines: what information is needed to evaluate them (structure), and what to do with this information in order to review the guidelines (logic).
- Enable the adaptation of general guidelines according to a given situation (interpretation).
- Enable the identification of potential semantic similarities and differences among commonly structured guidelines. This identification could be very useful because we have the intention to enable the simultaneous evaluation of guidelines issued from multiple sources.

Estimation of automation feasibility

The framework should also enable the evaluator to estimate the feasibility of the (automated) evaluation. For this purpose, we introduced the concept of automation level. An *automation level* is an indication to what extent a given guideline can be automatically evaluated. We defined two kinds of automation level.

A **theoretical level** quantifies the available elements for the evaluation. It can be:

- *Total*: we estimate that identified HTML elements cover all the aspects of the guideline.
- *Partial*: we estimate that identified HTML elements cover some of the aspects of the guideline.
- *NONE*: we cannot identify HTML elements to evaluate any of the guideline aspects.

A **practical level** qualifies the ability to implement automated evaluation of the identified elements. It can be

- *Total*: we estimate that we can implement automated evaluation for all identified HTML elements needed for the evaluation.
- *Partial*: we estimate that we can implement automated evaluation for some of the identified HTML elements.

- *NONE*: we estimate that we cannot implement automated evaluation for any of the identified HTML elements.

Improvement and flexibility of the evaluation process

The framework should provide means to enable an automated tool based on it to improve the evaluation process when possible. The expected improvement possibilities are:

- **Parsing the evaluated Web page**: improve this process at the level of phase number (ideally one foreword phase) and the information capture level (capture the minimal information needed for evaluation).
- **Evaluation of the captured data**: the framework should improve the execution the evaluation logic against the captured data. Some information is added during the structuring process to enable this kind of improvement.

Table 4.1 shows examples on automation levels.

Guideline	Automation level	Reason
Select colors that will make your page easy to read by people with color blindness [Vanderheiden et al. 1997]	Theoretical: total	There are HTML color-related elements
	Practical: None	Evaluation conditions difficult to formalize
Never have a link that points right back to the same page [Nielsen 1999]	Theoretical: total	There are HTML needed elements
	Practical: total	Evaluation conditions can be formalized and easily realized
Provide equivalent alternatives to auditory and visual content [W3C 1999]	Theoretical: partial	There are HTML elements but not for all desired aspects of the guideline
	Practical.: partial	

Table 4.1: Some guidelines and their estimated evaluation automation level

4.2 The Framework

The framework is composed of steps that correspond to the tasks and sub-tasks that an evaluator would generally accomplish to evaluate a guideline. In figure 4.4, we give the task model of an evaluation task (CTT formalism [Paterno97]) as it is accomplished according to the proposed framework.

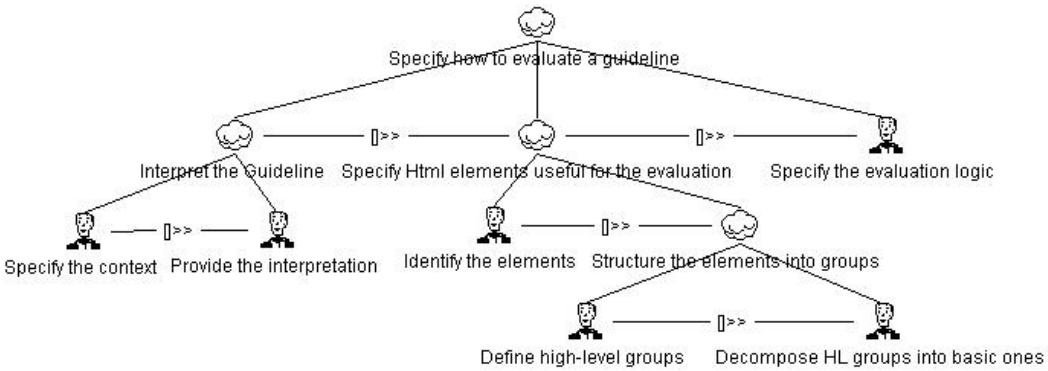


Figure 4.4: Task model of the task "Specify how to evaluate a guideline". Notice that every sub task needs information from the preceding one. In addition, there is a clean separation between the

information needed for evaluation and the evaluation logic. This separation should facilitate any eventual updating of the specification.

Next, we are going to describe the steps of the framework. The concepts manipulated during these steps are presented in figure 4.5. To introduce the framework, we will see how its related concepts are applied on the guideline: "*Select colors that will make your page easy to read by people with color blindness*". We selected this guideline because it can have many interpretations with different degrees of theoretical and practical evaluation levels.

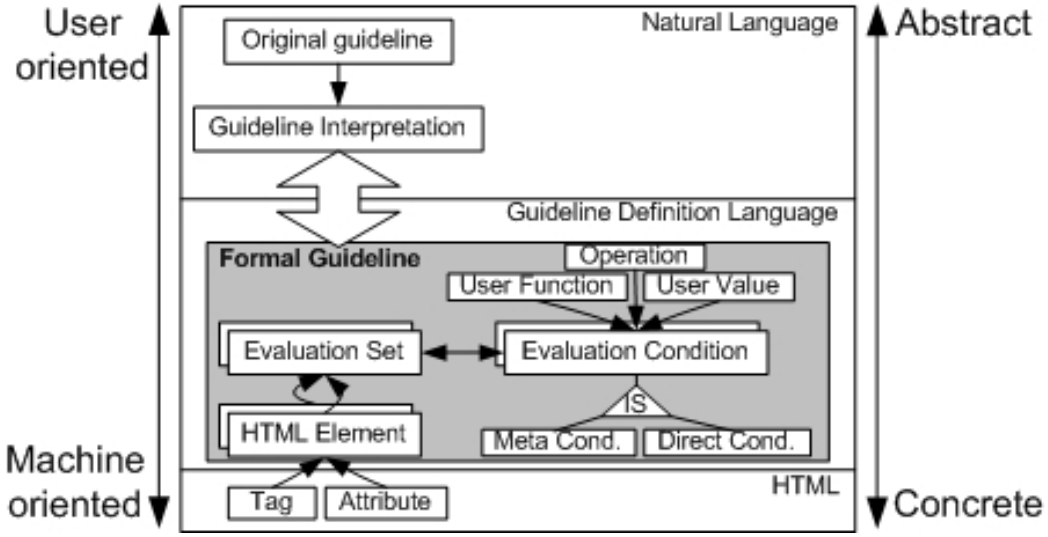


Figure 4.5: fundamental concepts manipulated by the framework

4.2.1 Step 1: Interpret the guideline

As guidelines are expressed as general recommendations independent from any context, a guideline could be interpreted differently from one evaluation context to another; the same guideline could have more than one interpretation, depending on the interpreter, context of use, etc. In some situations, it may turn out that evaluating a guideline, although theoretically possible, is not practically possible for various reasons: too many HTML elements, a lot of code to perform the evaluation, many possible evaluation cases, etc. For example, a Section508 guideline is *"Web pages shall be designed so that all information conveyed with color is also available without color, for example from context or markup"*. This covers a lot of possibilities like making the colored text italic, bolded, underlined, putting it in a table, changing its size, etc. An interpretation of the guideline can not practically cover all these possibilities, thus, those that are more difficult to be implemented or less frequently used will be ignore.

In fact, the interpretation is the re-formulation of the guideline using the evaluator vocabularies instead of ergonomics expert vocabularies. The difference between the two expressions depends among others on the guideline's abstraction level and on the evaluator comprehension level of the guideline aims. Of course, even with interpretation, evaluation of some guidelines cannot be totally automated [Farenc et al. 1996]. For example, guideline 1 of WCAG1.0 [W3C 1999] recommends to *"Provide a text equivalent for images"*. Although, an experienced developer knows that the only way to do this in HTML is via the alt attribute of the tag IMG, we choose to provide an interpretation of this guideline. It can be *"alt attribute for*

images must exist" or more precisely "*alt attribute for images must not be empty*". Notice that this interpretation already gives a hint about how to evaluate the guideline.

In order to have similar structures for all guidelines, we consider that every guideline has at least one interpretation even if the guideline can be evaluated without interpretation. In such cases, the interpretation is almost the same as the guideline, but this enables us to define a default guideline interpretation that we can use if no evaluation context is specified.

The interpretation context

The context of use of the evaluated Web site directly influences the interpretation. Usually, we define the context of use as a triplet:

- *User*: this attribute characterizes the stereotype of the site users. We use terms like normal, motor handicapped, visual handicapped, etc.
- *Platform*: this attribute characterizes the hardware and software constraints over the use of the site. We use terms like laptop with Windows XP, PDA with Windows CE, etc.
- *Environment*: this attribute characterizes environmental constraints over the use of the site. We use terms like office, car, stressing noise, limited space, etc.

An interpretation is generally the projection of the guideline semantics on the targeted interpretation context, thus, this projection is usually a limitation or specialization of the guideline semantics. For example, for the above Section508 guideline, the interpretation will ignore some of the possibilities to convey colored information, thus, it limits the semantics of the guideline. We can generally characterize this limitation in terms of targeted categories of objects (ex. structuring objects, visual objects, etc.) or simple objects (ex. tables, frames, client-side images, fonts, etc.)

Therefore, we will define the *interpretation context* as a specialization of the context of use by adding to it one additional parameter that we will call *Target_objects.* This parameter gives indication about the limitation of the interpretation to a category of objects or to some simple objects. This means that we can have more than one interpretation in a given context of use, but no more than one in a given interpretation context.

Heuristics for interpretation formulation

Ideally, we must pass from the guideline to the interpretation according to some rules that should systematize as possible and somehow control the process. This is not a trivial matter, and it appears in many situations like knowledge acquisition [Collier 1993], database engineering, etc. As for the ergonomics field, we have works like the framework of Scapin [1990] who proposed a framework to allow stable, unambiguous, and precise translation of human factors recommendations into evaluation rules for WIMP. According to Scapin [1990], this task is very difficult, and must be based on a quite complex iterative examination, from different points of view, of a large number of guidelines.

We had the same difficulty underlined by Scapin, therefore, here are some heuristics that we underlined after applying the frameworks of few guidelines. We

hope that some rules will emerge in the future when applying the framework on more examples:

H1- Use HTML related vocabularies when possible. For example, instead of saying *"Provide a text equivalent for images"* we say *"alt attribute for images must not be empty"*.

H2- Make sure to limit the interpretation semantics to the targeted context. We can control this rule by examining the *Target_Objects* and the context of use.

H3- Use simple sentences: subject, verb, complement.

Application on the guideline example

This guideline, as stated in its source, cannot be automated in a straightforward manner.

First Interpretation (standard context of use, 8 basic colors)

If we refer to the research conducted by Murch [Murch 1987], an interpretation (restriction) of this guideline can be expressed as: *the combination between the background color and the foreground color should belong to the best color combination or should not belong to the worst color combinations*. Figure 4.6 represents good color combinations for thin lines and text. It can be read as follows: for thin lines and text displayed on a white background, blue is a good choice in more or less 94% of cases, black is in 63% of cases, and red is in 25% of cases (Murch's research is based on legibility tests for user acceptance of color combinations). For thin lines and text displayed on a black background, white is a good choice in more or less 75% of cases, yellow is in 63% of cases. The other lines can be read similarly.

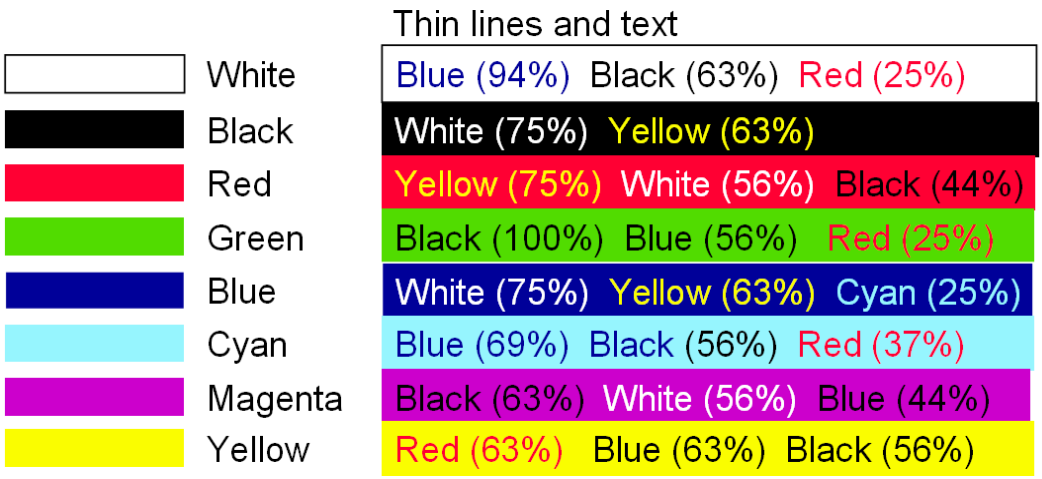


Figure 4.6: Good color combinations for thin lines and text by order of acceptance.

Figure 4.7 represents the good color combinations for bold lines and panels. It can be read as follows: for bold lines and panels displayed on a white background, black is a good choice in more or less 69% of cases, blue is in 63% of cases, and red is in 31% of cases. We observe that usable color combinations also depend of the text style or area. While identifying bold text is easy thanks to the <B> **bold text** </B> tag, identifying bold lines and panels remains more challenging to do automatically.

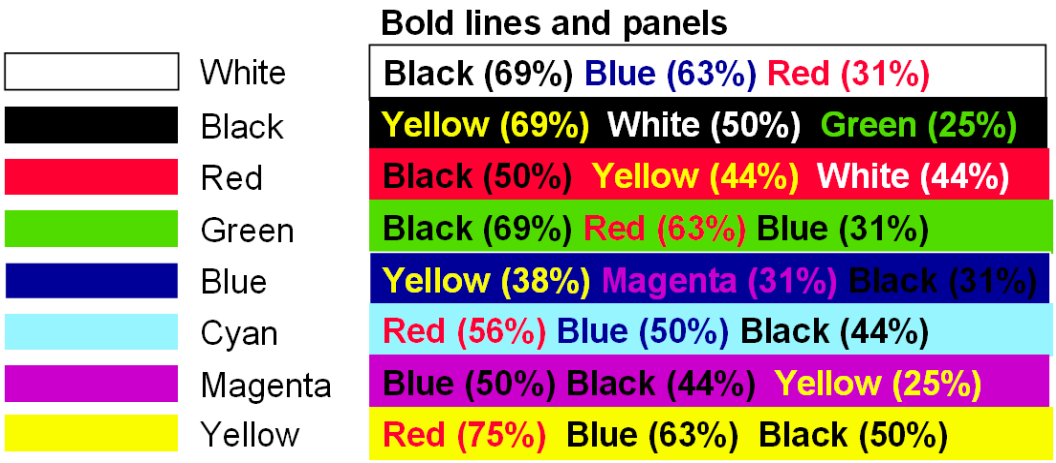


Figure 4.7: Good color combinations for bold lines and panels by order of acceptance.

Figure 4.8 represents the bad color combinations for thin lines and text. It can be read as follows: for thin lines and text displayed on a white background, yellow induces a legibility problem in almost all cases, while cyan does it in 94% of cases, and so forth.

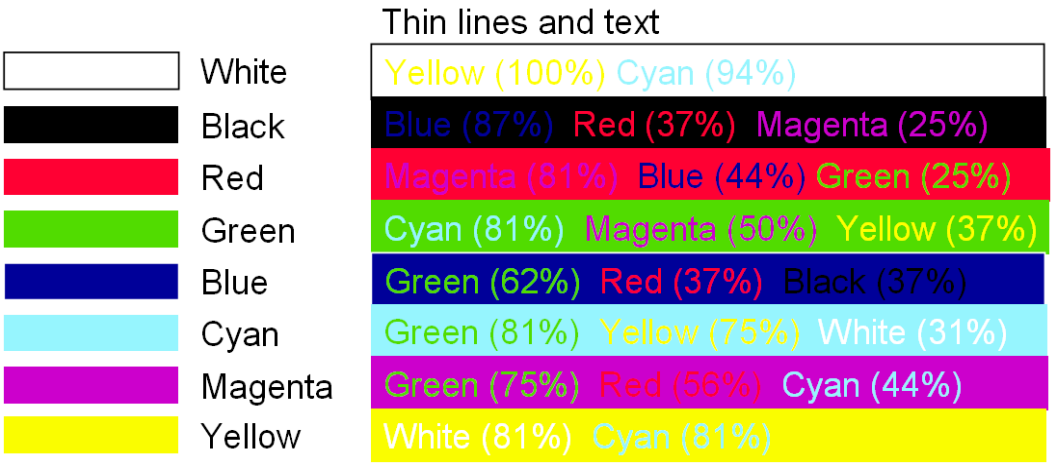


Figure 4.8: Bad color combinations for thin lines and text by reverse order of acceptance.

Figure 4.9 represents the bad color combinations for bold lines and panels. It can be read as follows: for bold lines and panels displayed on a white background, yellow induces a legibility problem in nearly 95% of cases, while cyan does it in 75% of cases, and so forth.

Now, having these experimental results in mind, we can see how to automate the testing of this guideline. The different colors used in the HTML code should be identified for this purpose. Color is rendered on computer-based screens as an additive mixture of three primary colors: red, green, and blue, the intensity of which is indicated by a value ranging from 0 to 255. Thus, in the Red-Green-Blue (RGB) model, each color consists in a triple (red-intensity, green-intensity, blue-intensity). This triple is represented in HTML as a hexadecimal code of the type #XXYYZZ, where XX,YY, and ZZ represents the different intensities respectively. Therefore, these codes need to be transformed into decimal for comparison. When these values belong to the set {00,33,66,99,CC, FF}, the resulting colors are said to be principal as no other palette than the basic palette

should be loaded to display the colors. The bgcolor and color HTML tags specify the foreground and the background text colors respectively.

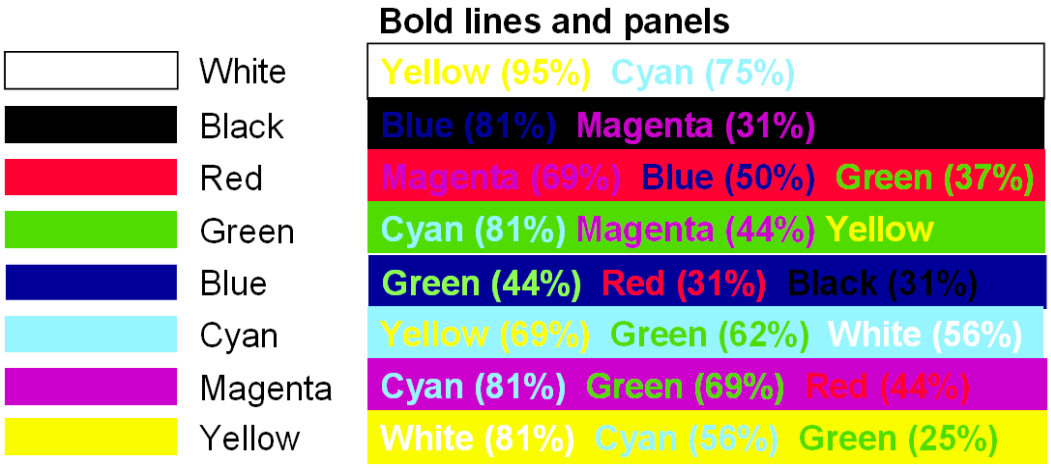


Figure 4.9: Bad color combinations for bold lines and panels by reverse order of acceptance

This is a basic procedure, which does not support the evaluation of hue, saturation and lightness: each composite color is reduced to its corresponding primary color, thus introducing a restriction of the initial guideline. Introducing another algorithm deriving moderator from the codes can solve this shortcoming: for example, “light blue”, “very light blue”. Graphic backgrounds are not supported although the different colors used in GIF or JPEG files could be identified. But in this case, a spatial algorithm should take care of physical appearance of text on top of graphical background colors, which is far more complex. As we can see here, the cost of developing the complete procedure for automated testing of a single guideline can be very high. In this case, the level of support is partial automation. A user testing technique to test the legibility can be used equally.

Second Interpretation (standard context of use, all colors)

A second more practical interpretation of our guideline could be obtained by adopting recent findings in optics about color differences for the human eye. The human vision system perceives images in color using receptors on the retina of the eye which respond to three relatively broad color bands in the regions of red, green and blue (RGB) in the color spectrum (red, orange, yellow, green, blue, indigo, violet). According to Kingsbury [2003], to test if two colors (foreground and background) can be differentiated by human eye, the difference between the values of their Luminance must be greater than a given value. The Luminance of a color can be calculated by the following formula:

$$Y = 0.3*Red + 0.59*Green + 0.11*Blue$$

Where Red, Green, and Blue can have real values in the interval [0..1]

Then we calculate the difference:

$$D = |Yfg-Ybg| \text{ //absolute value}$$

Where Yfg is the luminance of the foreground color, Ybg is the luminance of background color.

If D is too small, the text cannot be read. The problem is to define the threshold value. But we can define it by 0,2 or 0,3. Lower values of D are hard to read.

Let us apply this theory on Murch's combinations. For example, if we want to test the formula for the combination: White background, Blue text:

White: Yw = 0.3*1 + 0.59*1 + 0.11*1 = 1
Blue: Yb = 0.3*0 + 0.59*0 + 0.11*1 = 0,11
D = |Yw-Yb| = 0,89 //D >0.3

Thus blue text is readable on white background.

Let us examine the combination: White background, Black text:

White: Yw = 0.3*1 + 0.59*1 + 0.11*1 = 1
Black: Yb = 0.3*0 + 0.59*0 + 0.11*0 = 0
D = |Yw-Yb| = 1 // D>0.0

Thus black text is readable on white background.

Let us examine the combination: White background, Yellow text:

White: Yw = 0.3*1 + 0.59*1 + 0.11*1 = 1
Yellow: Yy = 0.3*1 + 0.59*1 + 0.11*0 = 0,89
D = |Yw-Yy| = 0,11 //D <0.3

Thus yellow text is not readable on white background.

The formula seems confirming Murch statistical results. (Black text, white background) and (blue text, white background) are visible here whereas (yellow text, white background) is not visible enough.

Notice that this interpretation deals with visibility and not with preference as Murch interpretation does. In addition, it is general and can be applied on any color combination. As for practical implementation, it is also easier than the interpretation 1.

4.2.2 Step 2: Specify HTML elements useful for the evaluation

This step consists in identifying and isolating the HTML elements on which the automated evaluation will be performed. The resulting set of elements can be different from one evaluator to another according to his HTML experience, interpretation, and understanding of the original guideline's aims.

By *HTML element*, we mean a HTML tag like IMG or a tag and one of its attributes like (Body, bgcolor).

As an interpretation of guideline usually limits its semantics, we use only the elements that we estimate useful for the evaluation of the interpretation, we will ignore elements that can be used to evaluate some of the guideline aspects but these aspects are not considered in the interpretation.

Application on the guideline example

According to our experience, HTML provides all the needed information to control text color in Web pages, thus, we estimate that the automation of this

guideline's evaluation is theoretically total. The HTML elements concerned by this evaluation are:

- Body.text: determines the foreground text color all over the Web page.
- Body.bgcolor: determines the background text color all over the Web page.
- Body.link: determines the foreground color of links all over the Web page.
- Body.alink: determines the foreground color of active links all over the Web page.
- Body.vlink: determines the foreground color of already visited links all over the Web page.
- Font.color: determines the foreground text color between <Font> tag and corresponding </Font> tag.
- Table.bgcolor: determines the background text color inside a table (between <Table> tag and corresponding </Table> tag).
- TH.bgcolor: determines the background text color inside a header cell in a table (between <TH> tag and corresponding </TH> tag).
- TD.bgcolor: determines the background text color inside a normal cell in a table (between <TD> tag and corresponding </TD> tag).
- TR.bgcolor: determines the background text color inside a raw of cells in a table (between <TR> tag and corresponding </TR> tag).

4.2.3 Step 3: Structure selected elements into evaluation sets

Tasks of Step1 and step2 are traditional tasks accomplished by any evaluator whatever the strategy used to conduct WU&AE, if the evaluation is based on the analysis of HTML code only.

This simple identification of interesting HTML elements is not sufficient to define a well structured methodology for U&AE, especially if we target maximizing the automation of the evaluation process: HTML elements have different semantics, elements scopes vary according to their position in the page, etc. Working at this granularity level makes it very difficult to underline and to exploit automatable aspects.

In the proposed methodology, we do not stop at this level, because we want our evaluation methodology (and later an evaluation tool) to be flexible and configurable to fit any evaluation context. After identifying the HTML elements which are important for the guideline' scope, we determine whether they could be grouped in some way. For this purpose, we defined the concept of evaluation set. An *evaluation set*, very similar to a WCAG1.0 checkpoint [W3C 1999], is a group of HTML elements that are needed together to evaluate a precise aspect of the guideline. Some of these sets could be more important for the guideline evaluation than others. Thus, each set is assigned to a priority level to express its importance in conformance with W3C specification:

- **Priority 1**: A web content developer **must** satisfy the conditions for positive evaluation of this set. Satisfying this set is a basic requirement for the web content to respect the guideline.
- **Priority 2**: A web content developer **should** satisfy the conditions for positive evaluation of this set. Satisfying this set will remove significant barriers for the web content to respect the guideline.

- **Priority 3**: A web content developer **may** address this set. Satisfying this set will improve respect of the web content to the guideline.

Notice that the priority level of a set may change according to the evaluation context. Of course, this change is due to variation of the associated guideline's impact on usability in the targeted evaluation context. For example, a guideline about using good color combinations must be satisfied (Priority 1) when targeting users with color blindness or using low-depth colored monitors, whereas it is less significant (Priority 2 or even 3) in the context of normal users using modern colored monitors.

An important question arises concerning evaluation sets: what are the criteria for grouping elements into sets? Unfortunately, until now with the few examples that we realized, we could not formalize any general criterion. We hope that such criteria would emerge after applying the framework on more examples.

Application on the guideline example

- Set1 (color of links): {Body.bgcolor, Table.bgcolor, TH.bgcolor, TR.bgcolor, TD.bgcolor, Body.link, Body.vlink, Body.alink}. **Priority 1.**
- Set2 (color of normal text): {Body.bgcolor, Table.bgcolor, TH.bgcolor, TR.bgcolor, TD.bgcolor, Body.text, Font.color}. **Priority 1**.

Why we grouped elements like this? Because we wanted a distinction between links and other text, but it is purely arbitrary. Putting all the elements in one evaluation set about ″color of any text″ is absolutely possible.

Definition of atomic evaluation sets

An evaluation set is said to be *atomic* if it contains the minimal number of HTML elements that allow the evaluation of a precise aspect of the guideline. After grouping the HTML elements into evaluation sets, we examine each set to see whether it is atomic. Some evaluation sets could remain the same when they are already atomic.

In fact, the distinction between normal evaluation set and an atomic one enables us to provide an interesting feature in a tool adopting our methodology. This feature is the ability to work, during guideline structuring, on two abstraction levels concerning HTML elements:

- **HTML level**: a HTML experienced developer may directly work at HTML level by using HTML tags and attributes. For example, to evaluate our example, he will directly use the possible combinations of the HTML elements that selected in step1 to specify evaluation sets.
- **Concept level**: an inexperienced developer may not be aware of all color control possibilities provided by HTML. In this case, he may use abstract concepts to specify evaluation sets, and the tool would automatically determine HTML corresponding elements. In our color example, the user may specify that the combination {foreground color, background color} must be evaluated, and the tool will use the abstraction tree depicted in figure 4.10 to determine the HTML combinations mentioned above.

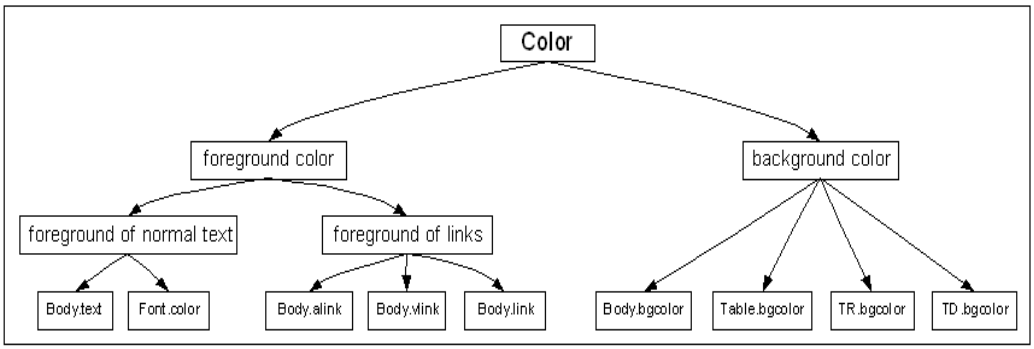


Figure 4.10: Abstraction levels for colors in HTML4.0

Implementing this feature is complex because it is not an easy task to build a semantic network[1] for the 91 tags (and their associated attributes) of the current version of HTML (http://www.w3.org/TR/html4/index/elements.html).

Scope of set elements

A major difference between our framework and other structuring ones (ex. WAI's WCAG1.0 [W3C 1999]) is that, in addition to structuring guidelines, we provide information about how to capture the data needed for the evaluation. This information is provided in the definition of atomic evaluation sets. Every set element has a scope. Based on the concept on the scope of HTML elements (figure 4.11), a *scope of a set element* is the zone of the Web page where encountered instances of this element must be captured. This means that we ignore element instances outside its scope as precised for a given set.

Thus, for every set element *E1* we specify a scope that can have one of following values:

- **Scope E1=Page**: means that we capture all the instances of *E1* wherever we encounter them inside the evaluated page.
- **Scope E1=E2**: means that we capture the instances of *E1* if and only if we encounter them inside the start and end tags of *E2*. Of course, *E1* and *E2* must be elements of the same set.
- **Scope E1=E2 OR E3…OR En**: means that we capture the instances of *E1* if and only if we encounter them inside the start and end tags of any of *E2…En*. Again, *E1, E2…En* must be elements of the same set.
- **Scope E1=Ei AND Ej…AND En** ⇔ Scope E1=Ei AND scope Ei=Ej AND Scope Ej=En.

Thus, the precision of an element scope is crucial because it leads the parsing phase of the evaluated page.

Notice that at least one element of an atomic set must have the scope *Page* to start a new instance of the set.

[1] A semantic network or net is a graphic notation for representing knowledge in patterns of interconnected nodes and arcs. Computer implementations of semantic networks were first developed for artificial intelligence and machine translation, but earlier versions have long been used in philosophy, psychology, and linguistics. What is common to all semantic networks is a declarative graphic representation that can be used either to represent knowledge or to support automated systems for reasoning about knowledge. Some versions are highly informal, but other versions are formally defined systems of logic [Sowa 1999].

Application of atomic sets for the guideline example

Set1 will be decomposed into the following atomic sets. All of them are of **Priority 1**:

- AS1={Body.bgcolor[Page], Body.link[Page]}: color of links.
- AS2={Body.bgcolor[Page], Body.alink[Page]}: color of active links.
- AS3={Body.bgcolor[Page], Body.vlink[Page]}: color of visited links.

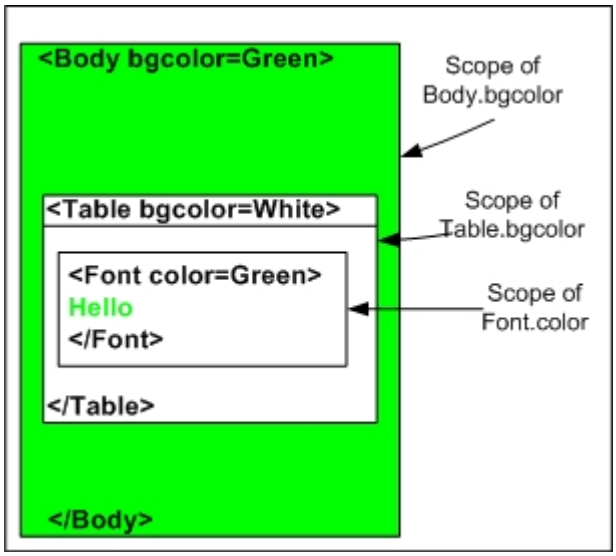


Figure 4.11: Scope of HTML elements. Body.bgcolor has no effect of the text inside Font start and end tags because Table.bgcolor overcomes Body.bgcolor. This is true only if we precise that Scope Font.color=Table.bgcolor.

Every one of these sets can have one instance at most because all elements are in the Body tag, which occurs only once in a Web page.

Set2 will be decomposed into the following atomic sets:

- AS4={Body.bgcolor[Page], Body.text[Page]}: text color over the whole page.
- AS5={Body.bgcolor[Page], Font.color[Body.bgcolor]}: color of text delimited by Font tags outside the scope of any element (ex. Table).
- AS6={Table.bgcolor[Page], Font.color[Table.bgcolor]}: color of text delimited by Font tags inside a table.
- AS7={TR.bgcolor[Page], Font.color[TR.bgcolor]}: color of text delimited by Font tags inside a table row.
- AS8={TH.bgcolor[Page], Font.color[TH.bgcolor]}: color of text delimited by Font tags inside a table header cell.
- AS9={TD.bgcolor[Page], Font.color[TD.bgcolor]}: color of text delimited by Font tags inside a table cell.
- AS10={Body.text[Page], Table.bgcolor[Body.text]}: color of text inside a table.
- AS11={Body.text[Page], TR.bgcolor[Body.text]}: color of text inside a table row.
- AS12={Body.text[Page], TH.bgcolor[Body.text]}: color of text inside a table header cell.
- AS13={Body.text[Page], TD.bgcolor[Body.text]}: color of text inside a table data cell.

Q&A

Q1: is this the only possibility of decomposing higher evaluation sets into atomic evaluation sets?

A1: in this evaluation context, the answer is YES. We can notice that every atomic evaluation set corresponds to a precise aspect (very depending of its position in the page): links, visited links, text outside Tables, etc. it is clear from figure 4.9 that it is meanless to put in the same atomic evaluation set more that one element controlling background color or foreground color. Thus, every set will have at most one element from every category. Therefore, every high-level evaluation set was decomposed into possible combinations of (foreground color, background color) of its elements.

Q2: Are there any criteria to guide the evaluator during decomposition?

A2: We provide here a number of criteria that are potentially useful for sets decomposition. They are still invalidated by a sufficient number of examples:

C1) atomicity of atomic sets: deleting one set element will make it impossible to evaluate the objective targeted by the set.

C2) completeness of atomic sets: the set of aspects covered by all the atomic sets must enable the evaluation of the aspects covered by the upper evaluation set.

C3) Uniqueness of an atomic set: every atomic set must be unique inside the upper evaluation set. We measure this uniqueness in terms of covered aspect.

In the case of our example, we have one foreground color and one background color in an atomic set, thus, the atomicity criterion is verified. The atomic sets cover all possibilities of controlling text color, thus, the completeness criterion is verified. No two atomic sets cover the same scope in a Web page, thus, the uniqueness criterion is verified.

Exclusion among evaluation sets

We mentioned earlier (sect. 3.4.2) that exclusion among evaluation sets is a very interesting concept. We can use this aspect to improve evaluation process at two levels:

- **The parsing level**: we can choose to avoid instances of excluded sets, so, they are not captured with usability data. For example, it is meanless to capture instances of Font.bgcolor to add it to instances of AS5={Body.bgcolor[Page], Font.color[Body.bgcolor]} if Font.bgcolor is inside a table that has bgcolor attribute.
- **The evaluation level**: we can choose, for some reason, to keep data related to excluded sets during parsing phase. In this case, it will remain possible to ignore this data during evaluation.

Exclusions among sets are specified after the definition of evaluations sets. By examining the specified atomic evaluation sets, we have the following exclusion relations (based on the scope of HTML elements):

- AS13 excludes AS11, AS10 and AS4 (TD.bgcolor overcomes TR.bgcolor, Table.bgcolor and Body.bgcolor).
- AS12 excludes AS11, AS10 and AS4 (TH.bgcolor overcomes TR.bgcolor, Table.bgcolor and Body.bgcolor).
- AS11 excludes AS10 and AS4 (TR.bgcolor overcomes Table.bgcolor and Body.bgcolor).
- AS10 excludes AS4 (Table.bgcolor overcomes Body.bgcolor).
- AS9 excludes AS7, AS6 and AS5 (TD.bgcolor overcomes TR.bgcolor, Table.bgcolor and Body.bgcolor).

- AS8 excludes AS7, AS6 and AS5 (TH.bgcolor overcomes TR.bgcolor, Table.bgcolor and Body.bgcolor).
- AS7 excludes AS6 and AS5 (TR.bgcolor overcomes Table.bgcolor and Body.bgcolor).
- AS6 excludes AS5 (Table.bgcolor overcomes Body.bgcolor).
- AS5 excludes AS4 (Font.color overcomes Body.text).

4.2.4 Step 4: Specify the evaluation logic

The evaluation logic specifies how to analyze the data that we capture in a Web page in order to check if this data respects or violates the evaluated guidelines. The logic is specified in terms of evaluation conditions. Conditions are defined on evaluation sets using sets elements, some constant values (numeric, Boolean, sets of values, etc), mathematical operations, and logical operators. A single evaluation condition may concern one or more evaluation sets.

We defined two forms of conditions: direct and Meta condition. Next, we will describe each of them.

Direct condition

A *direct condition* is the specification of evaluation logic on one or more evaluation sets by using the concrete objects (ex. set elements) needed for the evaluation.

For example, to specify the evaluation logic on the set AS1={Body.bgcolor[Page], Body.link[Page]}, we provide the following direct condition (case of interpretation with the luminance concept):

```
IF |Luminance of Body.link – Luminance of Body.bgcolor| >0.3  THEN
        The guideline is respected
ELSE
        The guideline is violated
```

We used the set elements (directly) to specify the evaluation logic.

Meta condition

It is probable that many evaluation conditions, defined on some different evaluation sets, express similar evaluation logic. In such case we define one Meta condition to represent all these conditions.

A *Meta condition* is a condition whose expression can be applied on various terms independently. For this reason, instead of using set elements directly, we use some Meta variables to specify how to conduct the evaluation. A *Meta variable* is an object that we use to specify evaluation logic in Meta conditions instead of using direct objects, then we instantiate the Meta condition on evaluation sets by mapping its Meta variables to their corresponding concrete elements in the sets.

For example, the evaluation logic for all the atomic sets of our case example is identical and we can express it as the following Meta condition (case of interpretation with the luminance concept):

```
IF |Luminance of backgroundColor – Luminance of foregroundColor| >0.3  THEN
        The guideline is respected
ELSE
        The guideline is violated
```

Where backgroundColor and foregroundColor are Meta variables.

Operations in evaluation conditions

In both kinds of evaluation condition, we decompose the condition into smaller parts that we call operations.

An *operation* is a piece of evaluation logic in which we use one single operator on some valid operands. For example:

A + B //A and B are real values

This is addition operation of two real values.

Operations allow us to have high control level on how to run the condition, when to stop it, what message to generate for different evaluations results, etc.

In the first version of our formal language we define basic simple data types: Integer, Float, Boolean, String and Hex (Color[2]). Every type has some predefined operations: And, Or, XOR and Not for Boolean, Substring for String, getRed, getGreen, getBlue for Color, etc. There are also three constructed data types with their predefined operations: Table, Sequence, and Cartesian Product. The detailed description of these data types and predefined operations will be given in the next chapter on the GDL.

User values

In order to have a maximum of flexibility, the evaluator can specify (declare) values of the predefined data types, and then use them in the operations.

A complex value can be assigned an identifier to simplify and clarify the specification of evaluation expression. For example, in the above Meta condition, we declare a user value (Id=VisibilityCoeff, type=Float, value= 0.3), then we use its id in the evaluation expression instead of directly using the value 0.3.

User functions

As the Java programming language will be used to implement the supporting tool, we can use some Java predefined functions by applying a Java mechanism called *Method invocation*. Therefore, evaluation expressions can contain functions that the user specifies as Java functions to be invoked during condition execution.

This mechanism is similar to DLL evaluation routines used in Sherlock [Grammenos et al. 2000].

Application of evaluation conditions on the guideline example

The evaluation conditions associated with the specified atomic sets are very similar. Thus, we can define a Meta condition to specify the evaluation logic, then we instantiate it for every set by mapping the Meta variables to corresponding concrete set elements.

[2] Color data type was defined because color is very important in Web pages

Condition for Interpretation 1 (Murch combinations)

The Meta condition (MC1) corresponds to the next pseudo specification:

```
IF      [(BgColor NOT IN ListOfMurchColors) OR
         (FgColor NOT IN ListOfMurchColors)] THEN
        Unrecognized Color
ELSE IF [(FgColor IN ListOfGoodColors(BgColor)) OR
           (FgColor NOT IN ListOfBadColors(BgColor))] THEN
        Guideline is respected
ELSE IF [(FgColor IN ListOfBadColors(BgColor))]] THEN
        Guideline s violated
```

According to this condition:

- If the background color or the foreground color does not belong to Murch colors, we cannot evaluate the guideline (more precisely, the interpretation of the guideline).
- If the foreground color is of Murch colors but not in the good list nor in the bad list for the considered background color, we cannot decide if the guideline is respected or violated.

The condition uses the following objects:

User Values (to be declared before specifying the condition)

```
Black   ="#000000"
White   ="#ffffff"
Red     ="#ff0000"
Green   ="#00ff00"
Blue    ="#0000ff"
Cyan    ="#00ffff"
Magenta="#ff00ff"
Yellow  ="#ffff00"
ListOfMurchColors= Sequence[Black, White, Red, Green, Blue, Cyan, Magenta, Yellow]
ListOfGoodColors= Table[Black   → Sequence[White, Yellow]
                        White   → Sequence[Blue, Black, Red]
                        Red     → Sequence[Yellow, White, Black]
                        Green   → Sequence[Black, Blue, Red]
                        Blue    → Sequence[White, Yellow, Cyan]
                        Cyan    → Sequence[Blue, Black, Red]
                        Magenta→ Sequence[Black, White, Blue]
                        Yellow  → Sequence[Red, Blue, Black]]
ListOfBadColors= Table[ Black   → Sequence[Blue, Red, Magenta]
                        White   → Sequence[Yellow, Cyan]
                        Red     → Sequence[Magenta, Blue, Green]
                        Green   → Sequence[Cyan, Magenta, Yellow]
                        Blue    → Sequence[Green, Red, Black]
                        Cyan    → Sequence[Green, Yellow, White]
                        Magenta→ Sequence[Green, Red, Cyan]
                        Yellow  → Sequence[White, Cyan]]
```

Meta Variables

BgColor : HEX (for background colors)

FgColor: HEX (for foreground colors)

Predefined operations

ListOfGoodColors(Color X) and ListOfBadColors(Color X) are predefined operations on Table data type. For the index X, the operations return the corresponding value in the table, in this case a value of Sequence data type.

Decomposition of the condition into operations

As we normally decompose evaluation conditions into operations, the above Meta condition is composed of the following operations:

OP1: (BgColor IN ListOfMurchColors)
Result=FALSE → Stop evaluation: "Unrecognized background color".
Result=TRUE → Continue to Op2

OP2: (FgColor IN ListOfMurchColors)
Result=FALSE → Stop evaluation: "Unrecognized foreground color".
Result=TRUE → Continue to Op3

Notice that operations enable us to generate highly customized warnings or error messages to provide the evaluator by clear information about the evaluation result.

OP3: (FgColor IN ListOfGoodColors)
Result=TRUE → Stop evaluation: "Good color combination".
Result=False → Continue to Op4

OP4: (FgColor IN ListOfBadColors)
Result=TRUE → Stop evaluation: "Bad color combination".
Result=FALSE → Stop evaluation: "Good color combination".

Meta condition for Interpretation 2 (Luminance formula)

The Meta condition (MC2) corresponds to the following pseudo specification:

```
IF ABS[((0.3*getRed(BgColor)+0.59*getGreen(BgColor)+0.11*getBlue(BgColor)) –
        ((0.3*getRed(FgColor)+0.59*getGreen(FgColor)+0.11*getBlue(FgColor))] >VisibilityCoeff
THEN
        Guideline is respected
ELSE
        Guideline is violated
```

The condition uses the following objects:

User Value

VisibilityCoeff=0.3

Predefined operations

- getRed, getGreen, and getBlue are predefined operations on Hex data type. They correspond to getting red, green, and blue component of a color.

- ABS (a-b) is a predefined operation on numbers. It returns the absolute value of (a-b).

This condition covers all possible colors. The only subjective consideration is the value under which we consider that the text is not readable. In fact, such situation shows the strength of our methodology, because it is very easy to change this value in a given context.

Decomposition of the condition into operations

As we normally decompose evaluation conditions into operations, the above Meta condition is composed of the following operations:

OP1: getRed(BgColor)→ V1 (0..1)
Result=ANY → Continue to Op2

OP2: Green(BgColor)→ V2 (0..1)
Result=ANY → Continue to Op3

OP3: Blue(BgColor) → V3 (0..1)
Result=ANY → Continue to Op4

OP4: *(V1, 0.3) → V4
Result=ANY → Continue to Op5

OP5: *(V2, 0.59) → V5
Result=ANY → Continue to Op6

OP6: *(V3, 0.11) → V6
Result=ANY → Continue to Op7

OP7: +(V4, V5, V6) → V7
Result=ANY → Continue to Op8

We provide the above operations to do the corresponding calculation with the background color. We can execute them in any order.

OP8: getRed(FgColor)→ V8 (0..1)
Result=ANY → Continue to Op9

OP9: getGreen(FgColor)→ V9 (0..1)
Result=ANY → Continue to Op10

OP10: getBlue(FgColor)→ V10 (0..1)
Result=ANY → Continue to Op11

OP11: *(V8, 0.3)→ V11
Result=ANY → Continue to Op12

OP12: *(V9, 0.59)→ V12
Result=ANY → Continue to Op13

OP13: *(V10, 0.11)→ V13

Result=ANY → Continue to Op13

OP14: +(V8, V9, V10)→ V14
Result=ANY → Continue to Op15

We provide the above operations to do the corresponding calculation with the foreground color. We can execute them in any order.

OP15: -(V7, V14)→ V15 //Calculate the difference of luminance
Result=ANY → Continue to Op16

OP16: ABS(V15) → V16 //Get absolute value of the difference of luminance
Result=ANY → Continue to Op16

OP17: LESS(V16, VisibilityCoeff)
Result=TRUE → "Bad color combination".
Result=FALSE → "Visible color combination".

Remarks on evaluation conditions

- As we use operations to compose the evaluation condition, the number of these operations depends on the good formulation of the condition. For example, the second condition (PS2) could be formulated as:

```
ABS[0.3*(Red(BgColor)-Red(FgColor))+
    0.59*(Green(BgColor)-Green(FgColor))+
    0.11*(Blue(BgColor)-Blue(FgColor))] >0.3
```

This pseudo condition will need 15 operations instead of 17 needed for MC2.

- For clarity reasons, we suppose that all operations are of depth one. For example, the following expression: +(*****(V8, 0.3), *****(V8, 0.59), *****(V10, 0.11)) is not allowed because the depth of operations * is two.
- In order for the proposed methodology to a have an added value over existing U&AE methods, its underlying GDL must be flexible enough to enable the specification of U&AE logic of any practically evaluable guideline. For this reason, we break evaluation conditions as small as possible to obtain this flexibility. In fact, a major requirement of the tool supporting the proposed methodology is to be able to automatically evaluate a high percent (ideally 100%) of guidelines evaluable by existing tools.

4.2.5 The formal Guideline

The formal guideline regroups all the formal elements of a guideline structure: HTML element, evaluation sets and exclusion relationships, user values and user functions, and evaluation conditions. Thus, the main purpose of this concept is to enforce the systematical structuring aspect of the framework.

By comparing our formal guideline with the specifications of Evallris [Abascal et al. 2003] and the SGSL [Takata et al. 2003], we estimate that our specification is generally richer and more flexible:

- It provides good information about the guidelines.
- The very important concept of interpretation is absent in both approaches.

- The concept of evaluation set corresponds to the concept of checkpoint for Evallris.
- The concept of evaluation condition corresponds to the concept of checkpoint for the SGSL.
- The concept of Meta condition is original. It should give us higher flexibility and facility in specifying the evaluation logic.
- Both approaches do not optimize the capture of evaluation data in the Web page: they scan the page completely to search for occurrences of checkpoints. We do not need to do so because evaluation sets specify what data to capture.
- The only potential advantage of the SGSL over our approach could be in working directly on XML documents, whereas we target HTML documents only.

4.3 Framework Advantages

By examining the concepts introduced in the framework, we can highlight some advantages of the proposed UE methodology over existing approaches:

4.3.1 Control of the evaluation process

In addition to the possibility to conduct an evaluation by guideline or by Web page as traditionally done in existing UE tools, it will be possible to conduct evaluation by HTML object (like tables, fonts, etc.): the framework allows us to identify guidelines having the given HTML element in their structure. This identification can be coarse grained (table, font, etc.) or fine grained (table width, font color, etc.).

Notice that if we choose to conduct evaluation by HTML object, the UE tool is supposed to identify all guidelines having this object in one or more evaluation sets of their structures. In this case, we will have at least two possibilities:

- Evaluate the identified guidelines partially by evaluating the evaluation sets having the HTML object only.
- Evaluate the identified guidelines totally.

Implementing these options in the UE tool is easy.

4.3.2 Support for multiple guidelines sources and interpretations

An obvious advantage is the possibility to evaluate guidelines from different sources in the same time. These guidelines can be well established (like WCAG1.0 [W3C 1999]) or user own guidelines.

In addition, we can provide multiple interpretations for a guideline to enable adequate adaptation of the guideline evaluation in multiple contexts of use. A potential utilization of this support is the possibility to evaluate some heuristic and empirical findings by transforming them into guidelines like forms. For example, Ivory [2001] presents a set of rules based on 157 page-level and site-level metrics formulated after an extensive survey of design recommendations from recognized

experts and usability studies. We can use such rules as additional guidelines in our kernel database.

Example of Ivory metrics-based rule

```
IF ((Italicized Body Word Count is not missing AND
   (Italicized Body Word Count > 2.5)))
Class = Poor
```

This rule classifies Web pages as having poor design quality if they contain more than two italicized words in the body text.

It will be possible to transform this rule into a guideline by considering that poor means ergonomic problem: *"Avoid having more than two and a half italicized body words in a Web page"*.

Formalizing this guideline gives the following:

Interpretation

Ivory [2001] provided the value 2.5 because of some statistical calculations that she used in her work. According to our HTML knowledge, we can't normally count fraction of words in a Web page, therefore, we will interpret the guideline as: *"The Web page must have less than three italicized body words"*. We consider that this interpretation is for normal context: ordinary user, standard PC, and comfortable environment.

HTML Elements

HTML provides a single element to italicize text:

I: italicizes text between <I> and </I> tags.

But the guideline speaks about words count, and the tag I is not sufficient to find this information in a Web page, thus, the framework can't formalize the guideline unless we interpret the guideline as: *"The Web page must have less than Three italic tags"*. Unfortunately, this will be correct only if every italic structure (….) enclosed one word at most, which is not usually the case.

In fact, we mentioned that the framework deals with HTML elements only, and the evaluation of this guideline deals with information that we cannot find in these elements only. The solution is to introduce a special element that represents any textual content of a Web page. For example, if we have the following HTML code in a Web page:

```
<I> this is an italicized text </I>
```

This special element will have the value: "this is an italicized text".

Fortunately, there is no problem in implementing this extension with Java (we use it to implement our evaluation tool), and capturing instances of this element during the parsing of a Web page is done in the same way used for HTML tags and attributes: the Java HTML parser sends back the value of textual content when it encounters it. For example, when it encounters the above HTML code, it returns:

```
Start tag: I
```

Data: this is an italicized text
End tag: I

Therefore, we will add this special HTML element to our elements list:

BodyText: textual content.

Evaluation sets

We will have a single (atomic) set:

S1={I[Page], BodyText[I]}. **Priority 1**

We consider that the set is of priority 1 because Ivory [2001] design classification has three levels: poor, average, and good. By correspondence with our priority levels, we will suppose that poor means that we violated a set of priority 1.

User Values

We will need one user value of type Integer:

MaxItalicizedWordNbr = 3

Evaluation conditions

We have one direct condition composed of one operation:

OP1: (Length(BodyText)) $\rightarrow$ V1
Result=ANY $\rightarrow$ Continue to Op2

OP2: (V1 < MaxItalicizedWordNbr)
Result=True $\rightarrow$ "Not poor page"
Result=False $\rightarrow$ "Poor page because it has more that 3 italicized words".

4.3.3 Semantic similarities and differences among guidelines

Structuring all guidelines in the same systematic manner will enable us to discover semantic similarities and differences among them.

Let us have the guidelines G1 and G2. We can examine these relationships at different levels:

HTML Elements

Let G1 have elems1={N elements} and G2 have elems2={M elements}. At this level, we can see what is the percentage of HTML elements that we identify to evaluate the guidelines. We have three possibilities:

- If elems1$\cap$ elems2={$\varnothing$}, we can say that G1's and G2's semantics are totally different.
- If elems1=elems2, we can say that they are semantically similar, without being able to affirm that they are semantically identical.
- If elems1$\cap$ elems2={k elements}, we can say that G1 and G2 have some semantic similarity.

When we have common elements, we can not be sure about similarity because we do not know at this level how the elements will be manipulated. For example, the guideline "images must have alt text" and the guideline "don't use images as page background" have the common element IMG, but at first look they have different semantics.

Evaluation sets

If we have (some or all) common HTML elements between G1 and G2, evaluation sets including these elements will give the same indication about semantics relationships because both concepts (element, set) are used at the structuring level. Let G1 has sets1={N1 sets} and G2 has sets2={M1 sets}:

- If sets1∩sets2={∅}, we can say that G1's and G2's semantics are totally different.
- If sets1=sets2, we can say that they are semantically similar. The possibility of being semantically identical becomes higher, but we still cannot affirm that.
- If sets1∩sets2={k1 sets}, we can say that G1 and G2 have some semantic similarity.

Therefore, even with evaluation sets, we cannot decide the effective similarity/difference degree between G1 and G2. We still need to examine the evaluation logic applied on these sets before we give the final judgment.

Evaluation conditions

The only possibility at this level is to examine conditions associated with identical evaluation sets.

Let:

G1 has the set set1

G2 has the set set2

set1=set2
G1 has cond1 the condition associated to set1

G2 has cond2 the condition associated to set2

IF cond1=cond2 //they have same operations with same arguments and actions.
⇒ set1 and set2 are semantically identical.
ELSE ⇒ need deep examination of conditions.

Let us examine the figure 4.13. We can see that G1 and G2 are identical at step2 and step3. At Step4, we can see that the first operation is identical in both conditions but not the global conditions.

By decomposing the two conditions into operations:

Condition1

OP1: (Input.type IN {"Submit", "Reset"}) → V1
Result=False → Stop //button is not for action
Result=True → Continue to Op2

OP2: (Length(Input.type)<=20) → V2
Result=False → Stop: Error
Result=True → Continue to Op3

OP3: (V1 AND V2)
Result=False → Stop: Error
Result=True → Stop: Success

Condition2

OP1: (Input.type IN {"Submit", "Reset"}) → V1
Result=False → Stop //button is not for action
Result=True → Continue to Op2

OP2: (Input.type=Input.value) → V2
Result=False → Stop: Error
Result=True → Continue to Op3

OP3: (V1 AND V2)
Result=False → Stop: Error
Result=True → Stop: Success

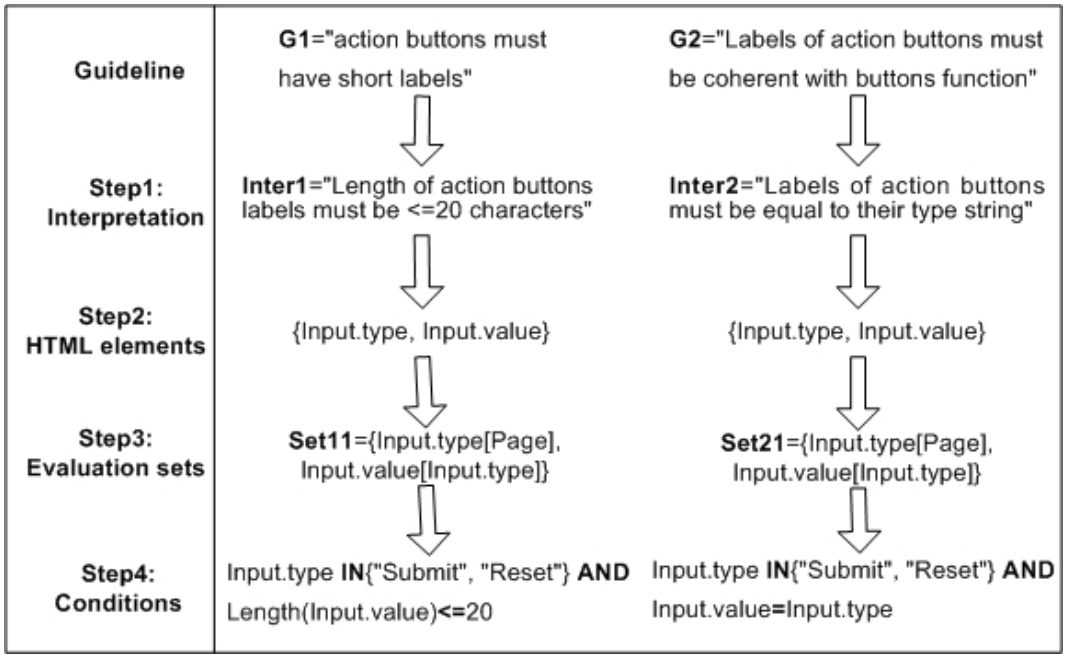


Figure 4.13: Guidelines similarities and differences.

In the above example, we have identical HTML elements and identical evaluation sets and similar evaluation conditions, therefore, we cannot give accurate answer without deep examination of evaluation conditions.

In fact, the first operation in both conditions would indicate that both guidelines deal with action buttons (input object with type="Submit" or "Reset"), thus, there is some semantic similarity between them. As for the second operation, we can deduce that in both cases it:

- Has Input.value as one of its arguments;
- Returns logical value;
- It generates an error for the False value and continues for True.
- The symbols in the two operations have some similarity (=, <=).

But the second argument is the only difference: in the first operation it is a number, and in the second it is a string. Therefore, there is a semantic DIFFERENCE.

The above discussion shows that the identification of similarities is potentially feasible, but it needs a very good analysis of evaluation conditions.

4.4 Approach Extensibility

The proposed approach targets Web usability and accessibility evaluation at a single page level and it deals with content composed of pure HTML code.

In fact, we think that the approach is potentially extensible over two axes: Scope level (single-page towards multi-page) and Technology level (pure HTML towards CSS).

4.4.1 Level Extension

We estimate that it is possible to extend the approach towards multi-page (site) evaluation. Of course, in this case we target multi-page guidelines.

In fact, if we examine the concepts of evaluation set and set element, we can see that it is possible to extend the approach by adding additional attribute to a set component to indicate explicitly the source of the captured data. To exemplify this, let us consider the following guideline: *links behavior must be coherent all over the site* [Nogier 2002]. Next we will see how we can structure it with our (extended) framework for a multi-page evaluation.

Step1: Interpretation

The same link can have several different behaviors like the navigation or the triggering of functions. Our intention here is not to study all the possibilities but to evaluate the applicability of the framework on site guidelines. For this reason, we will limit our interpretation to navigation behavior only. In this case, for the interpretation context {standard context of use, visible navigation links}, we provide the following interpretation: "*links that have the same label must point to the same destination and, conversely, that a page is always indicated by the same text*".

Step2: HTML elements

Links in HTML can be defined in two ways:

1) Using the A tag to define an anchor:

```
<A href="http://www.w3.org/">W3C Web site</A>
```

The A element's content defines the position of the anchor. The href attribute makes this anchor the source anchor of exactly one link. In the above example, the source anchor is the text "W3C Web site" and the destination page is http://www.w3.org/.

2) The LINK element enables an author to insert links that express other relationships between resources than simply "activate this link to visit that related resource".

Unlike A, the element LINK may only appear in the HEAD section of a document, although it may appear any number of times. Although LINK has no content, it conveys relationship information that may be rendered by user agents in a variety of ways (e.g., a tool-bar with a drop-down menu of links).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
        <TITLE>Chapter 2</TITLE>
        <LINK rel="Index" href="../index.html">
        <LINK rel="Next"  href="Chapter3.html">
        <LINK rel="Prev"  href="Chapter1.html">
</HEAD>
<Body>
…..
</Body>
```

This example illustrates how several LINK definitions may appear in the HEAD section of a document. The current document is "Chapter2.html". The "rel" attribute specifies the relationship of the linked document with the current document. The values: "Index" refers to a document providing an index for the current document, "Next" refers to the next document in a linear sequence of documents, and "Prev" refers to the previous document in an ordered series of documents.

As our interpretation context targets visible links only, we identify {A.href} as the only HTML element available to evaluate our guideline because LINK is used for links that have no content (invisible).

The HTML elements that we keep are:

A.href
BodyText

We will need the special element BodyText to capture the content of the link.

Step3: Evaluation Set

We do not have many choices, the only evaluation set is:

Set1= {A.href[Page], BodyText[A.href]}

We will look for links all over a Web page, and then we will capture the text inside them. This set is already atomic.

Step4: Evaluation Condition

If we examine the identified evaluation set, we can notice that this set doesn't satisfy our need for multi-page evaluation because there is no way to know if two instances of this set belong to the same page or more. *Where to add such information? In the set definition or in the set element definition?*

If we add it to the set definition, we will have a new extended set definition like:

Set1[PageUrl]= {A.href[Page], BodyText[A.href]}.

The captured data from two pages will be something like:

Set1(PageUrl1)={A.href, Text1}
Set1(PageUrl2)={A.href, Text2}

In this case, and by examining the provided interpretation, we can specify an evaluation condition like the following:

IF PageUrl1=PageUrl2 THEN
  {IF (Text1<>Text2) → Error: "The page is the destination of two distinct links"}
ELSE
  {IF (Text1=Text2) → Error: "The same link points towards two distinct pages"}

Now, if we add the page information to the set element definition, we will have a new extended set definition like:

Set1= {A .href[PageUrl], BodyText[A.href]}

Notice that the scope of A.href is no longer Page (any page), but it is one specific page at a time. This page is identified by the page URL. For the element BodyText, the scope is also implicitly changed to the scope PageUrl.A.href.

The captured data from two pages will look like:

Set1={A.href[PageUrl1], Text1}
Set1={A.href[PageUrl2], Text2}

In this case, and by examining the provided interpretation, we can specify an evaluation condition like the following:

IF PageUrl1=PageUrl2 THEN
  {IF (Text1<>Text2) → Error: "The page is the destination of two distinct links"}
ELSE
  {IF (Text1=Text2) → Error: "The same link points towards two distinct pages"}

From this single example, it looks like it is the same to add the information to the set definition or to the set element definition. Anyway, our objective was to show the extensibility of our approach towards multi-page evaluation.

4.4.2 Technology Extension

The proposed framework is valid to evaluate Web usability and accessibility of pages composed of HTML elements only.

As the use of Cascading Style Sheets (CSS) mechanism for adding style (e.g. fonts, colors, spacing) to Web documents is becoming more frequent- in fact, one of the W3C WCAG1.0 guidelines recommends the use of CSS, it is interesting to examine the possibility to extend our approach to the evaluation of Web pages containing CSS code.

The basis of applying styles to documents is the rule. Each rule is composed of a number of components, each of which has a specific name and function (figure 4.15).

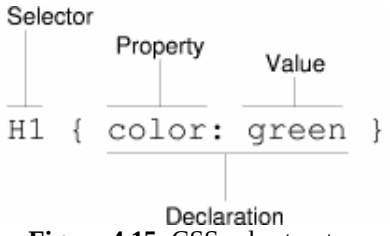


Figure 4.15: CSS rule structure

- The selector is the part that determines which portions of the document will be matched by the rule. The rule's styles will be applied to the selected element(s). For example, a selector of H1 means that all H1 elements will be selected. Multiple selectors can be grouped in a single rule by separating them with commas.
- The property is a quality or characteristic that something possesses. In the previous example, it is **color**. CSS2 defines around 120 properties and we can assign values to all of them.
- The value is a precise specification of the property. In the example, it is "green," but it could just as easily be blue, red, yellow, or some other **color**.

For any style sheet to affect the HTML document, it must be "glued" to the document. That is, the style sheet and the HTML document must be combined so that they can work together to present the document. This can be done in any of four ways:

1) Apply the basic, document-wide style sheet for the document by using the style element.

Here's a style sheet (shown in bold) glued to a sample document by using the style element.

```
<HTML>
        <TITLE>My home page</TITLE>
        <STYLE>
                H1, H2 {color: green}
        </STYLE>
        <BODY>
…
        </BODY>
</HTML>
```

2) Apply a style sheet to an individual element using the style attribute.

```
<p style="color: red">This paragraph's text will be colored red.</p>
```

3) Link an external style sheet to the document using the link element.

```
<link rel="stylesheet" type="text/css" href="http://www.my.site/styles/basic.css">
```

4) Import a style sheet using the CSS @import notation.

```
<style type="text/css">
        @import url(http://www.my.site/styles/autumn.css);
</style>
```

For more information about CSS, the interested reader can refer to http://www.w3.org/Style/CSS/.

We think that the approach is extensible to CSS for the following reasons:

- We can see that using CSS doesn't directly change the HTML code in three of the above four ways of gluing CSS to HTML (cases 1, 3 and 4). In these cases, the styling code is separated from the HTML code. So, we still can specify our formal guideline (HTML elements, evaluation sets, etc.) in the old way and, before starting the parsing of a Web page having CSS, we create a temporary new HTML page from the original one by applying the CSS to it. This has the inconvenient of decreasing to some extent the performance of the evaluation tool but it has a more important advantage which is enabling us to extend the approach to CSS.
- In the case of mixed HTML and CSS via the style attribute (case 2), we can do the same as for the above case but it will be very difficult because we need to parse every tag to check if it contains the style attribute or not. Another solution is to enable the use of CSS tags and attributes in the specification phase. This we put the difficulty on the side of the person who accomplishes the specification, which means that in addition to have good HTML knowledge s/he must have similar CSS knowledge and this is very demanding.
- The explanation given above shows that extension to CSS is feasible at the cost of developing one of the two methods.

4.5 Summary

In this chapter we have presented in details the first pillar of the proposed methodology: a framework for structuring WU&A guidelines in a systematical and consistent way to enable an improved and flexible automated evaluation of these guidelines. The framework is based on concepts that act at different levels during the structuring process:

- Interpretations provide a mean to adapt a general guideline to multiple interpretation (evaluation) contexts.
- HTML elements and evaluation sets enable the isolation and structuring of regions of interest (for evaluation) in a Web page.
- Evaluation conditions represent a key element in enabling an improved and flexible specification on the evaluation logic.

The framework enabled us to underline some interesting advantages of the proposed approach with respect to existing works. In addition, we demonstrated the possibility to use the framework for single page and multi-page evaluation by static analysis of CSS and/or HTML Web pages.

In the next chapter we will detail the second pillar of the proposed methodology: a formal Guideline Definition Language (GDL) to formalize the concepts of the framework.