**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 5

# A Formal Evaluation-Oriented Guidelines Definition Language (GDL)

## 5.1 Introduction

Our approach for automated WU&A evaluation is composed of a framework, a formal language, and an evaluation tool. We presented the framework in details in the previous chapter. Here we are going to present the GDL. Our aim is to provide a language by which we (and others) can structure Web guidelines in the systematical and consistent manner proposed by the framework, and to permit the development of a tool that is capable of reading and evaluating structures written by the language. We see great potential for structuring guidelines in an "open source" manner. To do this, the GDL must have a defined, publicly available syntax and semantics so that the meaning of structured guidelines is unambiguous (figure 5.1).

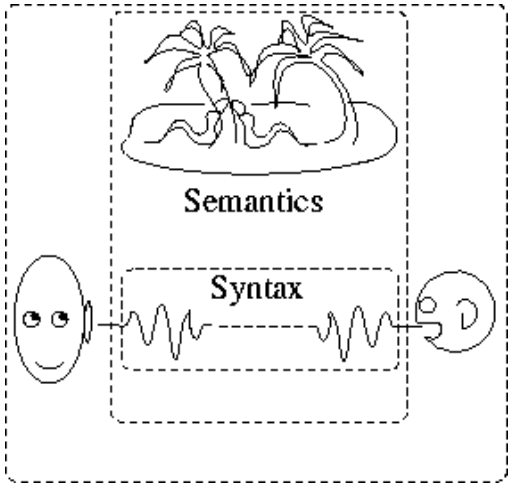


**Figure 5.1**: Syntax vs. Semantics. Inspired from (http://coral.lili.uni-bielefeld.de/Classes/Summer98/PragEngDialogue/pragengdialogue/node8.html)

### 5.1.1 The syntax

The syntax describes the collection of legal structures by means of a set of rules. This set of structure formation rules gives a formal definition of the syntax of a language. The syntax of a language has two forms: abstract and concrete.

The abstract syntax has the following objectives:

- to identify and separately name the abstract syntactic entities;
- to simplify and unify underlying concepts, putting similar things together, and reducing unnecessary duplication.

It is usually defined using (*extended*) *Backus-Naur Form* (E) (BNF) grammar (a context-free grammar). *EBNF* is any variation of the basic BNF notation with (some of) the following additional constructs: square brackets "[..]" surrounding optional items, suffix "*" for Kleene closure (a sequence of zero or more of an item), suffix "+" for one or more of an item, curly brackets enclosing a list of alternatives, and super/subscripts indicating between n and m occurrences. All these constructs can be expressed in plain BNF using extra productions and have been added for readability and succinctness.

A *context-free grammar* (CFG) is composed of four things (is a "4-tuple"). It consists of 1) a set of *terminal symbols*, 2) set of *non-terminal symbols*, 3) set of *productions* that are rewriting rules, and 4) a *start symbol* that is a non-terminal. Following is an example of a grammar for simple arithmetic expressions:

Terminal Symbols: + - * / ( ) i
Non-terminal Symbols: e t f
Start symbol: e
Productions:
e ::= e + t
e ::= e - t
e ::= t
t ::= f * t
t ::= f / t
t ::= f
f ::= i
f ::= ( e )

This grammar will generate simple expressions such as i, i+i, i-i/(i+i) etc. The productions tell how to rewrite one string of symbols as another.

As it is a well known formalism, we will use an EBNF-like notation to describe the abstract syntax of constructs of the GDL.

As for the concrete syntax, we will use XML for the following reasons:

- A key component of any software for automated web usability evaluation is the language used to structure the evaluated guideline. This language must trade-off between the richness of structuring definition and the ease with which this definition can be built [Ivory 2001]. XML has these two characteristics 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.
- XML enables us to easily separate structured guidelines from the evaluation tool. This is one of the major requirements of our approach.
- Using XML compliant language would enable us to integrate the GDL with EARL of W3C [W3C 2002a], especially as EARL can be considered complementary to the GDL in the overall testing process (figure 5.2).

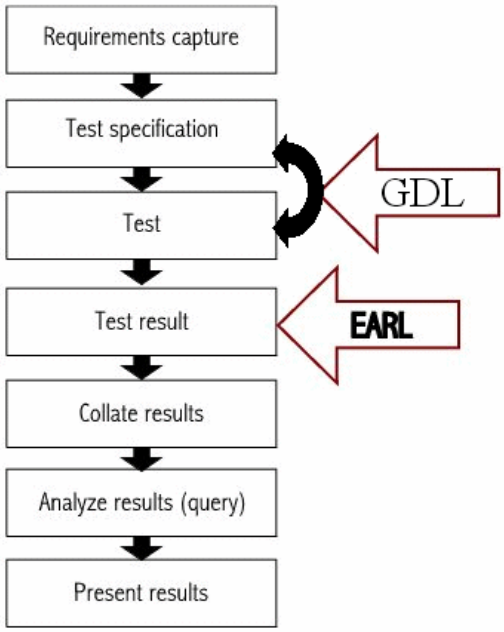


**Figure 5.2**: Complementarity of GDL and EARL in the testing process. The GDL enables the specification of how to conduct the test (ex. what are the examined tags, what data to capture) and provides possibilities to have high control over the test execution, whereas EARL enables the expression of test results. Test results include bug reports, test suite evaluations, and conformance claims.

- As most programming languages integrate XML support (parser, generator, interpreter), using XML enables us to avoid implementing a GDL interpreter to import structured guidelines into the evaluation tool.
- XML enables a potential portability of the evaluation. If we develop an online evaluation service supporting the GDL, it will be possible to use it to evaluate the guidelines stored on the server, as well as GDL-compliant guidelines stored locally. XML is more convenient for such scenario that other Web-oriented formats.

### 5.1.2 The semantics

The semantics reveals the meaning of syntactically valid strings in a language. In addition to the informal approach to expressing the semantics, there are a number of possible formal approaches. Aho and Ullman [1977] identify the following ones:

- **Mathematical (or denotational) semantics**: in this approach a mapping is defined between sentences in the language and mathematical objects that these sentences are said to denote.
- **Axiomatic definition*:*** rules are defined that relate the values of data before and after the execution of each language construct.
- **Extensible Definition*:*** the semantics is defined in terms of a set of primitive operations.
- **Translation**: the semantics of a language is defined through rules that specify how it may be translated into some other language whose semantics are already known, such as the lambda calculus.

- **Operational Semantics**: an abstract machine is described and the enactment and rules are provided for enacting programs on this abstract machine.

We choose to provide the semantics of the GDL in an informal way for the following reasons:

- The GDL is in its beta version and is subject to modifications syntactically and semantically.
- The GDL deals with a little number of concepts; an informal semantics would be sufficient to clarify them and their relationships.
- Many language concepts (HTML element, evaluation set, etc.) have few behavioral possibilities and informal semantics would be sufficient to clarify them.
- Although the semantics is informal, it is at least:
    - *Structured*: there is an underlying structure that organizes the specification;
    - *Categorized*: there are criteria that determine the nature of every element in the structure: HTML element, evaluation set, etc.

We will provide a part of this semantics as an entity-relation-attribute (ERA) schema with a detailed description of its entities. In addition, we will provide intuitive semantics of GDL constructs when we present its abstract and concrete syntaxes.

### 5.1.3 Aims of the GDL

The GDL main objective is to support structuring Web guidelines toward automated evaluation.

The most important, original characteristic of the GDL is its naturalness, i.e. the possibility offered by the language to straightforwardly map the informal natural language guidelines statements onto GDL formal statements.

The GDL is aimed at the modeling of HTML evaluable aspects (color contrast, alternative text for visual content, etc.), other frameworks have to be used to cope with other usability aspects (user satisfaction, consistency and information organization, etc.) that we can not express in terms of evaluation conditions.

### 5.1.4 Models of a GDL specification

The purpose of our language is to describe the admissible structures of a Web guideline in multiple evaluation contexts. This description, called specification of the structure, must formalize the maximum of details relevant to automating the guideline evaluation. The different structures will be *models* of the specification. The word *model* [1]is used here in the sense of a mathematical interpretation structure associated with a logical theory.

[1] In conceptual modeling the word model refers to the specification itself and not to the mathematical interpretation structure.

A GDL specification is structured in terms of guideline structures (the basic units of a GDL specification), a structure represent guideline's formal form in a given evaluation context.

## 5.2 Semantics of the GDL

The language is based on the concepts defined in the framework proposed in chapter 4. Figure 5.3 shows the ERA schema of these concepts and their relationships. We are going to describe the semantics of all the concepts at GDL and HTML level. As for the natural language level, we will describe the Guideline and Interpreted_GL only. The concepts of Reference, Translated_GL, and Criteria are used for a future integration with another schema [Mariage et al. 2003].

### 5.2.1 GUIDELINE

This is the definition of the original Web guideline as it is found in its source. It is characterized by the following attributes:

| Name | Description | Type |
|---|---|---|
| ID_Guideline | Every guideline is identified by a number that specifies its position within the guidelines set. | Simple, Mandatory |
| GLTitle | Concise title for the guideline | Simple, Facultative |
| GLStatement | Statement of the guideline as it is expressed in its source | Simple, Mandatory. |
| Reference | The identifier of the guideline source (a reference of a book, the url of a web site, etc.). | Simple, Mandatory. |
| Criteria | One or more ergonomic criteria that correspond to the taxonomy proposed by [Mariage et al. 2003] | Multiple, Facultative |
| GLComment | Rationale behind the guideline. | Simple, Facultative. |

**Example**

| | |
|---|---|
| GLTitle | Equivalent alternative for multimedia content |
| IGLStatement | Provide equivalent alternatives to auditory and visual content |
| IGLComment | Provide content that, when presented to the user, conveys essentially the same function or purpose as auditory or visual content. |

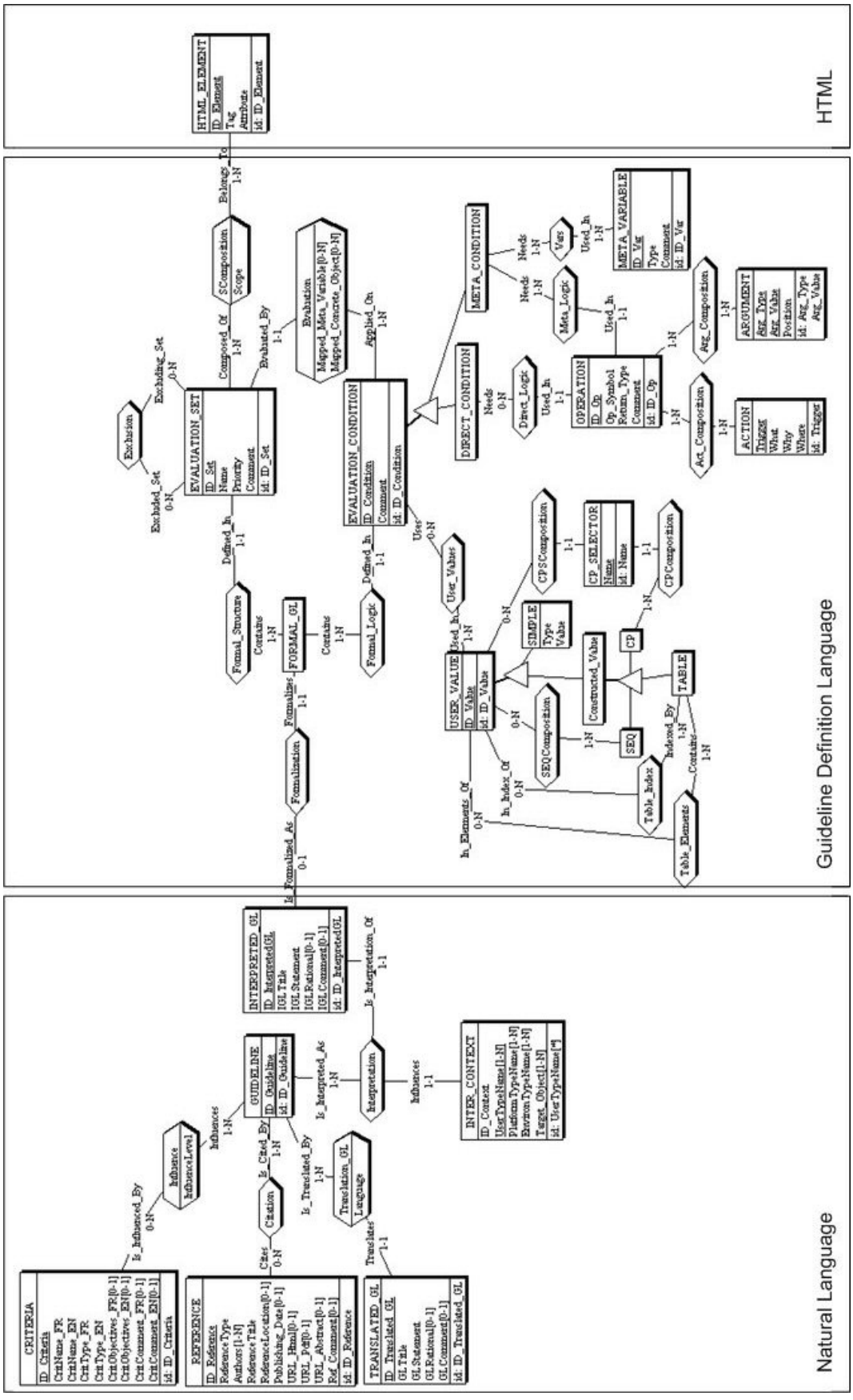


**Figure 5.3**: Schema ERA of the GDL concepts

### 5.2.2 INTERPRETED_GL

As we mentioned earlier, guidelines are rarely concrete enough or specific to a precise context of Web use. Thus, an interpretation of the original guideline is generally needed to limit the semantics of the guideline to a specific context.

An interpreted guideline is specified as a quadruplet:

| Name | Description | Type |
|---|---|---|
| ID_InterpretedGL | Every interpreted guideline is identified by a number that specifies its position in the whole set of guidelines' interpretations. | Simple, Mandatory |
| IGLTitle | Concise title for the interpreted guideline | Simple, Facultative |
| IGLStatement | Statement of the interpreted guideline. This statement must be more concrete that the original guideline, ideally in terms that can be mapped directly onto terms of HTML. | Simple, Mandatory. |
| IGLComment | Rationale behind the interpretation. | Simple, Facultative. |

**Example**

| | |
|---|---|
| IGLTitle | Equivalent for visual content |
| IGLStatement | Provide a text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content) |
| IGLComment | Screen reader of text browsers can render text elements. |

### 5.2.3 INTER_CONTEXT

An interpretation of a guideline **must be** done with respect to an associated context. We could call it *interpretation context* or *evaluation context* because it corresponds to the context in which the evaluation of the interpretation is considered equivalent to that of the original guideline. For the reasons mentioned in the previous chapter (4.2.1), we consider the interpretation context as an extended context of use by adding the Target_Object attribute to it.

| Name | Description | Type |
|---|---|---|
| ID_Context | An identifier to uniquely refer to a context. | Simple, Mandatory. |
| UserTypeName | Short description of the user stereotypes targeted by the evaluation | Multiple, Mandatory. |
| PlatformTypeName | A couple software-hardware that may significantly influence the context-sensitivity. | multiple, Mandatory |
| EnvironTypeName | Short description of the environment in which the user accomplishes the task. | Simple, Facultative. |
| Target_Object | Objects that limit the scope of the interpretation. They are provided individually (ex. IMG) or as categories (ex. Visual objects, client-side images). If not specified, the interpretation covers all objects. | Multiple, facultative |

**Example**

| | |
|---|---|
| ID_Context | Default context |
| UserTypeName | Ordinary healthy user. |
| PlatformTypeName | PC with Windows XP |
| EnvironTypeName | Low noise, non stressing environment |

### 5.2.4 FORMAL_GL

As we mentioned in chapter 4, the formal guideline is mainly used to enforce systematical aspect of the structuring framework. It is used to make a link between the guideline and its formal structure. A formal guideline consists of the result of the formalization of an interpreted guideline. Formalization only applies to interpreted guideline.

We want to have the ability to store all original guidelines of a given source, even those that can not be automated.

With respect to the structuring steps of the framework, a formal guideline is composed of the following parts that are further described in the subsequent sections:

- **Evaluation Structure**: in this section we specify:
  - *HTML elements*: in this section we provide the set of all HTML elements that we estimate needed to conduct the evaluation of the targeted guideline. This section corresponds to **step2** of the framework.
  - *Evaluation sets*: sub-sets of the set of HTML elements; and any exclusion relationships among them. Exclusions are specified here to be optionally used during parsing phase. This section corresponds to **step3** of the framework.
- **Evaluation Logic**: in this section we specify the evaluation conditions associated with the defined evaluation sets. If we use any user values or user functions, they must be introduced in this section. This section corresponds to **step4** of the framework.

### 5.2.5 HTML_ELEMENT

This entity represents a HTML tag or attribute. If it is an attribute, it must have its parent tag because some attributes (ex. alt) can be found in many tags, therefore, we need to specify the tag in order to precise the scope of the attribute.

As the schema shows, we must have at least one HTML element in our formal guideline. The identifier of the element will be used later in the specification, so, we can use a very simple one like *E1* or a more speaking one like *bodyText*.

The attributes of this entity are:

| Name | Description | Type |
|---|---|---|
| ID_Element | An identifier to uniquely refer to the element. | Simple, Mandatory. |
| Tag | The HTML tag (ex. BODY) | Simple, Mandatory. |

| | | |
|---|---|---|
| Attribute | The HTML attribute (ex. text of BODY.text). This attribute is facultative because some HTML elements like <B> have no attributes. | Simple, Facultative. |

**Example**

(Italic, I, NULL), (BodyBg, Body, bgcolor)

### 5.2.6 EVALUATION_SET

As described in the framework, we can have two levels of evaluation sets: high level and atomic level. In this GDL version we deal with atomic sets only because dealing with high level sets requires defining an abstract tree of HTML elements (see fig 4.10).

An evaluation set is characterized by the following attributes:

| Name | Description | Type |
|---|---|---|
| ID_Set | An artificial identifier of the set. | Simple, Mandatory. |
| Name | Optional name to provide concise information about the set. | Simple, Facultative. |
| Priority | Priority level of the set specified in conformance with W3C WCAG1.0 levels. | [1\|2\|3], Mandatory. |
| Comment | Provides short supplementary information about the set. | Simple, Facultative. |

We can see in the ERA schema (the relation SComposition) that we need to specify the scope of an HTML element when we add it to an evaluation set. A constraint is we cannot specify in the schema is: the Scope attribute of this relation can be the constant Page, another HTML element, inside one of many elements (ScopeOR), or inside more that one element in the same time (ScopeAND).

Another constraint on evaluation sets (the relation Exclusion) is that a set can not exclude itself, and two sets can not be mutually excluded.

### 5.2.7 USER_VALUE

Before specifying evaluation conditions, we can specify some values that can be of predefined simple or of constructed data type. Simple data types are: INTEGER, FLOAT, BOOLEAN, STRING, and HEXADECIMAL. These values are declared as triplet (Identifier, value type, value). For example: (VisibilityCoeff, Float, 0.3).

In addition to values of simple data types, we can define values of three constructed data types:

- SEQUENCE: represents an ordered list of objects of the same data type (ex. sequence of strings: PC=Sequence{“Case”, “Monitor”, “Hard disk”}).
- TABLE: represents a mapping between two objects: the index of the table and corresponding table content. For example, table of student results in an exam:

Exam= Table[ Marc → {55, 80, 60}
Alain → {55, 70, 90}]

- CARTESIAN PRODUCT: represents a record-like structure composed of N selector. A selector has an identifier and a data type of values that it can take.

Person= CP[Name : String, Age: float, Profession : String}

These values can then be used in specifying evaluation conditions. This approach has the following advantages:

- It maximizes the systematical aspect of our GDL. Every value will be specified before it is used.
- Values can be assigned to significant identifiers that reflect their semantics, which makes reading and validating the specification easier. For example, BlackColor is more meaningful than #000000.
- As it is possible to have a repetitive use of some values, the use of their identifiers makes the specification of evaluation conditions easier and more readable, especially for constructed values.

Simple data types can be implicitly used in the specification of user values, but as we are using XML as formalism for our specification, constructed data types must be defined in the GDL DTD to enable specification of user values of these types.

Every value is assigned to an identifier to facilitate its use in the expressions of evaluation conditions or in other constructed user values: as we will see later, complex values can be constructed from defined values by using their identifiers or from new values by using a complete structure.

### 5.2.8 EVALUATION_CONDITION

Every evaluation set is associated to an evaluation condition that specifies the logic to be applied on the set elements. A condition can be associated to more than one evaluation set.

If we have an evaluation logic that can be applied on many evaluation sets with some difference, we can define a *Meta condition* for all these sets, and then we defined some *mapping couples* (Mapped_Meta_Variable, Mapped_concrete_object) for every evaluation set. These couples determine how to instantiate the Meta variables in order to execute the Meta condition on instances of the set. If a set needs specific evaluation logic, we specify a *direct condition* for it. In this case, we directly use concrete objects (set element, user value, etc.) in the specification of the condition.

Evaluation conditions are characterized by the following attributes:

| Name | Description | Type |
|---|---|---|
| ID_Condition | An identifier to uniquely refer to a condition. | Simple, Mandatory. |
| Comment | Provides short supplementary information about the condition. | Simple, Facultative. |
| IsMeta | A Boolean value to indicate if a condition is a Meta one or not. | (True\| False), Mandatory |

### 5.2.9 META_VARIABLE

A Meta condition must have at least one Meta variable because by definition it must be mapped to some element in an evaluation set. A meta condition may hold one to many meta variables.

A Meta variable is characterized by the following attributes:

| Name | Description | Type |
|---|---|---|
| ID_Var | A significant denomination of the variable. | Simple, Mandatory. |
| Type | The type of the variable. It must be one of the predefined data types (simple or constructed). | Simple, Mandatory. |
| Comment | Supplementary information about the variable. | Simple, Facultative. |

We can see in the ERA schema (the relation Evaluation) that in the case of evaluating a Meta condition on the sets, we will need to make a mapping between meta variables and the corresponding concrete objects.

### 5.2.10 OPERATION

An operation enables the formal specification of evaluation expressions by dividing it into smaller pieces and providing explicit control points over the execution of the expression.

Another advantage is that specifying an expression with operations facilitates its reading by non-GDL experts because many information must be explicitly provided as operation attributes. In addition, this XML-compliant form facilitates the display of evaluation conditions in a web page because it can be controlled by a style sheet more easily that the case of a free text expression.

An operation is characterized by the following attributes:

| Name | Description | Type |
|---|---|---|
| ID_Op | An identifier of the operation to facilitate repetitive utilization of the operation in the specification. Its use in an expression means the use of the result of the operation. | Simple, Mandatory. |
| Op_Symbol | The operation symbol that must be that of a predefine operation (=, +, Sub_string, etc.). | Simple, Mandatory. |
| Return_Type | The return type of the operation: one of the predefined types. | Simple, Mandatory. |
| Comment | Short supplementary information about the operation | Simple, Facultative. |

**Example**

| | |
|---|---|
| ID_Op | Op1 |
| Op_Symbol | + |
| Return_Type | Integer |
| Comment | Addition of |

## 5.2.11 ARGUMENT

Operations work on one or more arguments. An argument is characterized by the following attributes:

| Name | Description | Type |
|---|---|---|
| Arg_Type | The type of the argument: Val if the argument is a predefined user value, Op if it is the result of another operation, SetElem/Set if it is a set element id/Set id (operations in direct conditions), and Var if it is a Meta variable (operations in Meta conditions). | (Val \| Var\| Op\| Set\| SetElem), Mandatory. |
| Arg_Value | The argument content: the identifier of a predefined value or an operation. | Simple, Mandatory. |
| Position | The position of the argument in the operation call if the position is relevant (like for Sub_string). This attribute is optional because, by default, the position of an argument is its position in the arguments list. | Integer, Facultative. |

**Example (for the operation IN (bgcolor ,MurchColors))**

| | | |
|---|---|---|
| Arg_Type | Val | Var |
| Arg_Value | Murch_Colors | Bgcolor |
| Position | 2 | 1 |

## 5.2.12 ACTION

We introduced this concept to obtain the desired high control level of the execution of operations and the generation of customized messages (error, warning, explanation, etc.).

After executing an operation, we specify the actions to be taken depending on the execution result.

| Name | Description | Type |
|---|---|---|
| Trigger | The specific value, of the execution result of the operation, that triggers the action | Simple, Mandatory. |
| What | What action to take when obtaining the triggering value. Possible actions are: Error to stop and indicate an ergonomic error, Jump to indicate the next operation, WarnStop to stop and generate a warning, WarnJump to warn and continue, and Success to stop and indicate that there is no problem. | (Error \| Jump \| WarnStop \| WarnJump \| Success), Mandatory. |
| Why | The message to generate in the different situations. It is facultative because it is not necessary when Jump and eventually Success cases. | Simple, Facultative. |
| Where | If the action is to jump, this attribute holds the identifier of the next operation. | Simple, Facultative |

**Example (for the operation IN (bgcolor ,MurchColors))**

| | | |
|---|---|---|
| Trigger | False | True |
| What | WarnStop | Jump |

| | | |
|---|---|---|
| Why | Background color does not belong to Murch colors | UNDEF |
| Where | UNDEF | Opi (ex. IN (fgcolor ,MurchColors)) |

Now that we presented the semantics of the language, we are going to present its syntax. We will present the two forms (abstract and concrete) simultaneously, but before that, as we use XML to specify the concrete syntax of the language, we are going to give a brief description of main XML concepts that are relevant for U&AE.

# 5.3 XML in a nutshell

As we are going to use XML to specify the concrete form of the GDL constructs, we will give here a brief description of main XML concepts [Harold & Means 2002].

## 5.3.1 XML Documents

An XML document contains text, never binary data. A XML document is composed of a tree of elements. Every element is delimited by the start-tag <element_name> and the end-tag </element_name>. Everything between the start-tag and the end-tag of the element (exclusive) is called the element's content. The white space is part of the content. Next is an example of an element:

<Profession> Engineer </Profession>

### XML Tags

XML tags look superficially like HTML tags. However, unlike HTML tags, we are allowed to make up new XML tags as we go along. The names of the tags generally reflect the type of content inside the element, not how that content will be formatted.

XML is case sensitive. <Person> is not the same as <PERSON>. We are free to use upper or lowercase or both as we choose. We just have to be consistent within any one element.

### Document Type Definitions (DTDs)

While XML can be expended with new elements and attributes, not all the programs that read particular XML documents are so flexible. Many programs can work with only some XML applications but not others. The solution is a *document type definition* (DTD). DTDs are written in a formal syntax that explains precisely which elements and entities may appear in the document and what the elements' contents and attributes are. A DTD can make statements such as "Every employee element must have a social_security_number attribute" Different XML applications can use different DTDs to specify what they do and do not allow.

There are many things the DTD does not say. In particular, it does not say the following:

- What the root element of the document is
- How many instances of each kind of element appear in the document
- What the character data inside the elements looks like

- The semantic meaning of an element; for instance, whether it contains a date or a person's name

DTDs allow us to place some constraints on the form an XML document takes, but there can be quite a bit of flexibility within those limits. A DTD never says anything about the length, structure, meaning, allowed values, or other aspects of the text content of an element.

## XML Trees

XML documents form a tree data structure (figure 5.4). Every XML document has one element that does not have a parent. This is the first element in the document and the element that contains all other elements. It is called the *root element* of the document. It is also sometimes called the document element.

XML gives each child exactly one parent. Each element (with one exception for the root element) has exactly one parent element. That is, it is completely enclosed by another element. If an element's start-tag is inside some element, then its end-tag must also be inside that element. Overlapping tags are prohibited in XML. The following example is a person element that contains information marked up to show its meaning.

```
<person>
        <name>
                <first_name>Alan</first_name>
                <last_name>Turing</last_name>
        </name>
        <profession>cryptographer</profession>
</person>
```

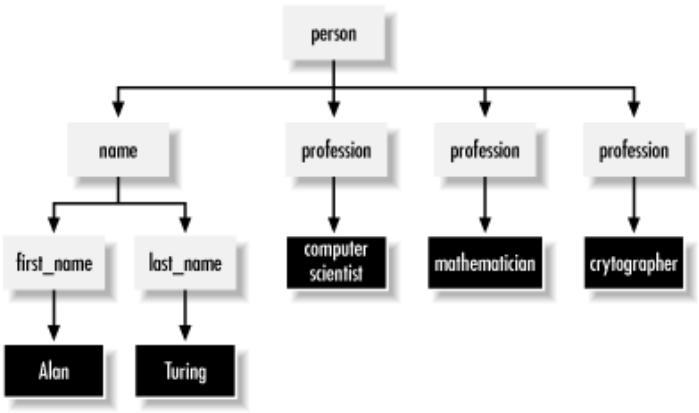


**Figure 5.4**: Tree structure for the above example

## Element Declarations

Every element used in a valid document must be declared in the document's DTD with an element declaration. Element declarations have this basic form:

```
<!ELEMENT element_name content_specification>
```

The name of the element can be any legal XML name. The content specification specifies what children the element may or must have in what order. Content specifications can be quite complex. They can say, for example, that an element must have three child elements of a given type, or two children of one type

followed by another element of a second type, or any elements chosen from seven different types interspersed with text.

## Child Elements

Another simple content specification is one that says the element must have exactly one child of a given type. In this case, the content specification simply consists of the name of the child element inside parentheses:

```
<!ELEMENT fax (phone_number)>
```

A fax element may not contain anything else except the phone_number element, and it may not contain more or less than one of those.

## Sequences

Most elements contain either parsed character data or (at least potentially) multiple child elements. The simplest way to indicate multiple child elements is to separate them with commas. This is called a sequence. It indicates that the named elements must appear in the specified order. For example, this element declaration says that a name element must contain exactly one first_name child element followed by exactly one last_name child element:

```
<!ELEMENT name (first_name, last_name)>
```

## The Number of Children

Not all instances of a given element necessarily have exactly the same children. it is possible to affix one of three suffixes to an element name in a content specification to indicate how many of that element are expected at that position. These suffixes are:

? Zero or one of the element is allowed (facultative or mandatory).

* Zero or more of the element is allowed.

+ One or more of the element is required.

## Choices

Sometimes one instance of an element may contain one kind of child, and another instance may contain a different child. This can be indicated with a choice. A choice is a list of element names separated by vertical bars. For example, this declaration says that a User_Value element can be either a Basic_Value child or a Constructed_Value child:

```
<!ELEMENT User_Value (Basic_Value | Constructed_Value)>
```

However, it cannot contain both at once. Each User_value element must contain one or the other. Choices can be extended to an indefinite number of possible elements.

## Attribute Declarations

Elements can have attributes. Attributes declaration is done with ATTLIST. A single ATTLIST can declare multiple attributes for a single element type. However, if the same attribute is repeated on multiple elements, then it must be declared separately for each element where it appears.

```
<!ATTLIST Original_Guideline
        O_ID            ID      #REQUIRED
        Name            CDATA   #IMPLIED
        Statement       CDATA   #REQUIRED
        Source          CDATA   #REQUIRED
        Comment         CDATA   #IMPLIED
>
```

## Attribute Types

Attribute values can be any string of text. The only restrictions are that any occurrences of < or & must be escaped as < and & and whichever kind of quotation mark, single or double, is used to delimit the value must also be escaped. However, a DTD allows you to make somewhat stronger statements about the content of an attribute value. The most used attribute types in XML are[2]:

### *CDATA*

A CDATA attribute value can contain any string of text acceptable in a well-formed XML attribute value. This is the most general attribute type.

### *Enumeration*

An enumeration is the only attribute type that is not an XML keyword. Rather, it is a list of all possible values for the attribute, separated by vertical bars. Each possible value must be an XML name token.

```
<!ATTLIST Meta_Var
        Type (Int | Hex | Uri | Str) #REQUIRED
>
```

### *ID*

An ID type attribute must contain an XML name (not a name token but a name) that is unique within the XML document. More precisely, no other ID type attribute in the document can have the same value. (Attributes of non-ID type are not considered.) Each element may have no more than one ID type attribute.

```
<!ATTLIST employee social_security_number ID #REQUIRED>
```

### *IDREF*

An IDREF type attribute refers to the ID type attribute of some element in the document. Thus, it must be an XML name. IDREF attributes are commonly used to establish relationships between elements when simple containment won't suffice.

### *IDREFS*

An IDREFS type attribute contains a whitespace-separated list of XML names, each of which must be the ID of an element in the document. This is used when one element needs to refer to multiple other elements.

[2] There are many more attribute types [Harold & Means 2002].

### Attribute Defaults

As well as providing a data type, each ATTLIST declaration includes a default declaration for that attribute. There are four possibilities for this default:

- **#IMPLIED**: The attribute is optional. Each instance of the element may or may not provide a value for the attribute. No default value is provided.
- **#REQUIRED**: The attribute is required. Each instance of the element must provide a value for the attribute. No default value is provided.
- **#FIXED**: The attribute value is constant and immutable. This attribute has the specified value regardless of whether the attribute is explicitly noted on an individual instance of the element. If it is included, though, it must have the specified value.
- **Literal**: The actual default value is given as a quoted string.

### Parameter Entities

It is not uncommon for multiple elements to share all or part of the same attribute lists and content specifications. In this case, we can declare a parameter entity to represent the common part, then, we use this entity later in the document.

A parameter entity reference is declared much like a general entity reference. However, an extra percent sign is placed between the <!ENTITY and the name of the entity. For instance, the following declaration defines the entity Cons_Expr as an enumeration of all the allowed constructed values in GDL:

```
<!ENTITY % Cons_Expr "Set_Value | Seq_Value | Table_Value | Cp_Value">
```

The entity can then be used in element declarations like:

```
<!ELEMENT Constructed_Values (%Cons_Expr;)+>
```

## 5.3.2 GDL restrictions caused by XML

We decided to specify Web guidelines in a XML-compliant form. This raises the question about what are the restrictions of XML that may impose further restrictions on the GDL.

By examining the concepts manipulated by the GDL, we can see that the major restriction appears for the specification of evaluation conditions. XML is suitable to specify GDL concepts like guideline, interpretation, HTML elements, evaluation sets, etc. because all what we need for these concepts is structure them in a suitable manner. As for the evaluation logic, more precisely the operations within conditions, we need a long specification to implement the functional behavior of the evaluation logic.

# 5.4 Specification of GDL-compliant structure for a Web guideline

In this section we will provide detailed description of what we specify in a GDL-compliant structure for a Web guideline (specification for short).

### 5.4.1 Scope of the specification

A specification must respect the following constraints:

- It contains one Web guideline only;
- It contains at least one interpretation of the guideline (default interpretation);
- An interpretation is at most associated to one formal guideline;
- A formal guideline must at least have one HTML element.
- A formal guideline must at least have one evaluation set;
- All the declared HTML elements must be covered by evaluations sets;
- A HTML element can be added to more that one set;
- A formal guideline must at least have one evaluation condition. If a Meta condition is defined, the formal guideline must at least have one mapped condition.

### 5.4.2 Organization of the specification

Figure 5.5 depicts a global view of a GDL specification. There are two constraints on this organization that are not explicitly presented in the figure:

- The horizontal position of objects determines their order in the specification. For example, we specify HTML_elements then sets then exclusions. The only exception is the children of Constructed_Values.
- We specify one ore more mapped condition if and only if we specified some Meta conditions.

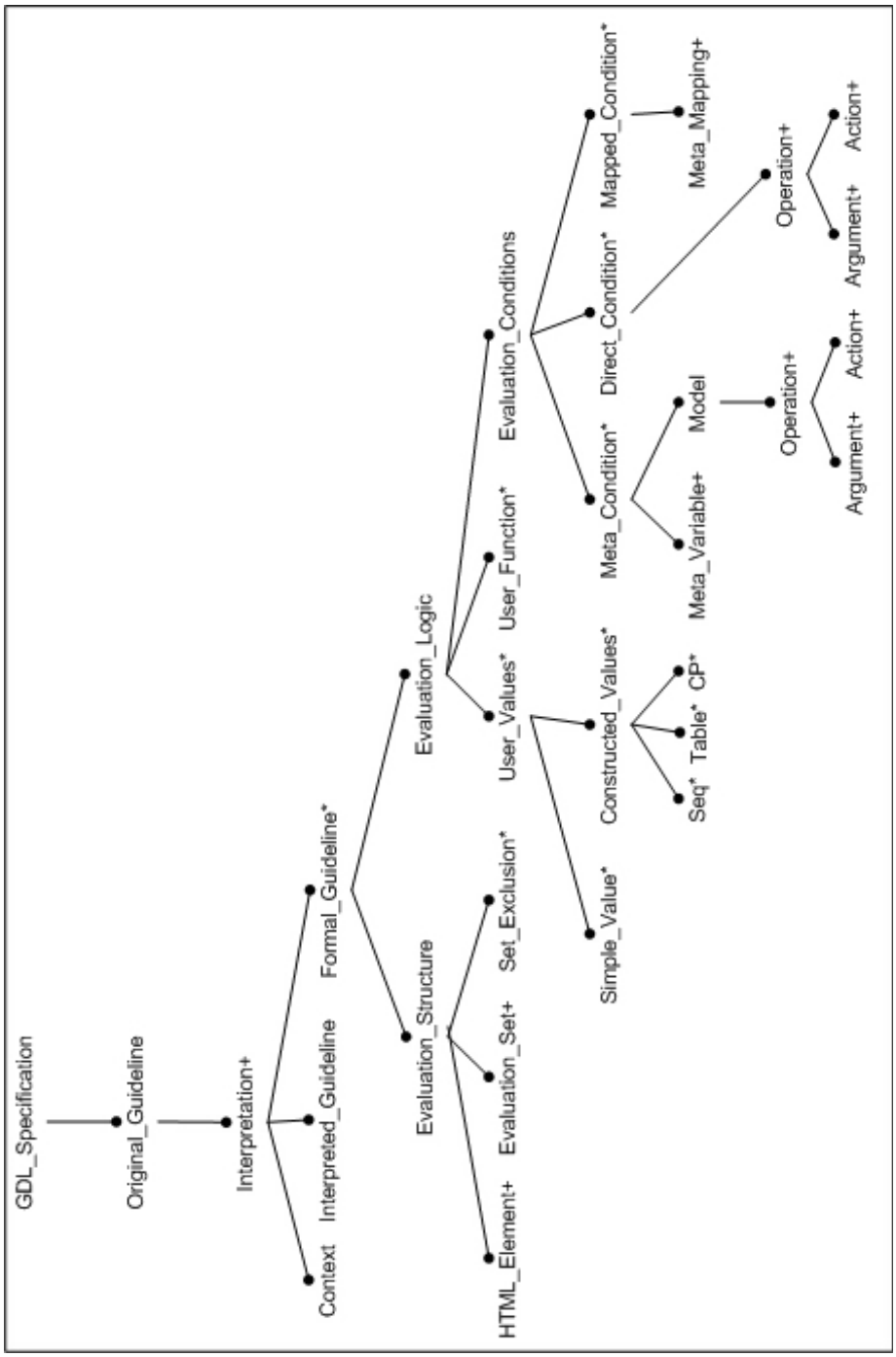


**Figure 5.5**: Organization of a GDL specification. The specification contains 0 or more occurrences of objects followed by * (ex. Seq), 1 or more occurrences of objects followed by + (ex. Action).

### 5.4.3 Abstract and concrete syntaxes

We will present the two syntaxes of the specification in the following formalisms:

#### EBNF abstract Syntax

We will use an EBNF context-free grammar:

- Non terminals are denoted by brackets (e.g. <a>).
- Terminals are written in bold. For example, **<GDL_Specification>** is the terminal that stands for a XML starting tag for the element GDL_Specification. Notice that,

as we use XML-compliant concrete syntax, < and > in terminals are interpreted as XML opening and closing tags characters.

In the derivation rules, we use the following conventions:

- "<a> | <b>" means <a> or <b>.
- "[<a>]" means <a> is optional.
- "{<a>}$^{*}$" means <a> *n* times ($n \geq 0$).
- "{<a>}$^{+}$" means <a> *n* times ($n \geq 1$).
- "<a-LIST>" is a shortcut for "<a> {, <a>}*".

### XML Concrete Syntax

As our intention is to separate guidelines from the evaluation tool, a XML DTD is defined for GDL-compliant guidelines structures. By this way, structures can be manually edited and then used by the UE tool to conduct the evaluation of the specified guidelines.

In the remaining of this section we are going to detail the description of abstract and concrete syntax of the nodes in figure 5.4. We will provide the abstract syntax, the concrete syntax, and an example for first two elements only. For the rest, we will give examples only. The complete version of the following section is provided in Annex A.

## 5.4.4 A GDL specification

This is the root element of the document containing the formal structure of a single guideline. It contains at least the original guideline a default interpretation with/without corresponding formal structure.

### Abstract Syntax

<SPEC> ::= < GDL_Specification >
<GDL-DEF> <INTER-DEF> |
<GDL-DEF> {<INTER-DEF> <FORMAL-GDL-DEF>}$^{+}$
< / GDL_Specification >

Notice that the combinations <Element_Name> and </Element_Name> must be used to indicate the beginning and the end of each specification element as dictated by the semantics inherited from XML.

### Concrete Syntax

```
<!DOCTYPE GDL_Specification [
<!ELEMENT GDL_Specification ((Guideline, Interpretation) |
                             (Guideline, (Interpretation, Formal_Guideline)+))>
```

## 5.4.5 Guideline

This is the original guideline as it is cited in the literature. It could be found in one or more references.

### Abstract syntax

<GDL-DEF> ::= < Guideline
ID_Guideline = <GLID-STRING>

```
                    [GLTitle          = <TITLE-STRING>]
                    GLStatement       = <STATE-STRING>
                    ID_References     = <IREFD-LIST>
                    [Criteria         = <CRITID-LIST>]
                    [GLComment        = <COMMENT-STRING>]
          / >
```

### Concrete Syntax

```
<!ELEMENT Guideline EMPTY>
<!ATTLIST Guideline
        ID_Guideline    ID      #REQUIRED
        GLTitle         CDATA   #IMPLIED
        GLStatement     CDATA   #REQUIRED
        ID_Reference    CDATA   #REQUIRED
        Ergo_Criteria   CDATA   #IMPLIED
        GLComment       CDATA   #IMPLIED
>
```

### Example

Following is the specification of the W3C WCAG1.0 guideline 1 [W3C 1999]. It is related to the accessibility criteria.

```
<Guideline
        ID_Guideline="1"
        GLTitle="Alternatives for multimedia content"
        Source="W3C_WCAG"
        Ergo_Criteria="Accessibility"
        GLStatement="Provide equivalent alternatives to auditory and visual content"/>
```

In order to facilitate the reading of the remaining of this chapter, we will show the concrete syntax only. The omitted parts can be reviewed in annex A.

## 5.4.6 Interpretation

An interpretation is composed of two elements: the context and the interpreted guideline.

### Example

Next is the specification of an interpretation of the a W3C WCAG1.0 guideline 1 in a normal context of use, but limiting the interpretation to simple images only

```
<Interpretation>
<Context
        ID_Context="C1"
        UserTypeName="Normal"
        PlatformTypeName="PC-Windows9x,XP "
        Environment="home, office"/>
<Interpreted_Guideline
        ID_InterpretedGL="I1"
        IGLTitle="Alternative for images"
        IGLStatement="Provide alternative text for simple images via alt attribute"
        IGLComment=""/>
</Interpretation>
```

### 5.4.7 Formal Guideline

The formal guideline specifies the structure of the associated interpretation. It is composed of two parts: evaluation structure and evaluation logic. This entity is purely structural to separate the formal structure from guideline and interpretation definitions.

#### Example

Any specification will have the following part:

```
<Formal_Guideline>
<Evaluation_Structure>
                        .....
        </Evaluation_Structure>
        <Evaluation_Logic>
                        .....
        </Evaluation_Logic>
<Formal_Guideline>
```

### 5.4.8 Evaluation Structure

This section specifies the HTML elements available to evaluate the guideline, and how these elements are regrouped into evaluation sets. If there are exclusion relations among sets, they are specified after the specification of sets

### 5.4.9 HTML Element

Identified HTML elements are given unique identifiers to facilitate their use in the specification.

#### Example

```
<HTML_Elements>
        <HTML_Element ID_Element="Bdbg"      Tag="Body"  Attribute="bgcolor"/>
        <HTML_Element ID_Element="Fcolor"    Tag="Font"   Attribute="color"/>
        <HTML_Element ID_Element="Bold"      Tag="B"/>
        <HTML_Element ID_Element="Italic"    Tag="I"/>
</HTML_Elements>
```

### 5.4.10 Evaluation Set

A set regroups some HTML elements needed to evaluate a precise aspect of the guideline. Every set has a priority level (1,2,3). The formal guideline must at least specify one set. Here we deal with atomic sets only.

#### Example

The following is a set to evaluate if a colored text is also italicized or bolded.

```
<Evaluation_Set
        ID_Set="S1"
        Name="bolded or italicized colored text"
        Priority="1"
        Description="check if colored text is also italicized or bolded">
        <Set_Element E_ID="Bdbg" Scope="Page"/>
        <Set_Element E_ID="Fcolor" ScopeOR="Bdbg Italic bold"/>
```

```
        <Set_Element E_ID="Italic" ScopeOR="Bdbg Fcolor"/>
        <Set_Element E_ID="bold" ScopeOR="Bdbg Fcolor"/>
</Evaluation_Set>
```

According to this specification,

```
<Body bgcolor=White>
        <Font color=Red> this is a red text </Font>
        → S1={Bdbg=White, Fcolor=Red, Bold=null, Italic=null}: KO

        <Font color=Red><I> this is a red italicized text</I></Font>
        → S1={Bdbg=White, Fcolor=Red, Bold=null, Italic!=null}: OK

        <B><Font color=Blue> this is a bold blue text</Font></B>
        → S1={Bdbg=White, Bold!=null, Fcolor=Blue, Italic=null, }: OK

        <I>this is an italicized text</I>
        → S1={Bdbg=White, Italic!=null, Fcolor=null, Bold=null, }: captured but irrelevant within
our context because the text is not colored.
</Body>
```

### 5.4.11 Set Exclusion

In this section we specify any exclusion relationships among the specified evaluation sets. An exclusion relationship is characterized by two attributes:

- **Excluding_Set**: the identifier of the excluding set.
- **Excluded_Set**: a list of one or more identifiers of sets excluded by the set.

#### Example

We saw the following sets in the previous chapter at page 76.

```
<Set_Exclusions>
        <Set_Exclusion Excluding="S7" Excluded="S6"/>
        <Set_Exclusion Excluding="S9" Excluded="S7 S8 S10"/>
</Set_Exclusions>
```

### 5.4.12 Evaluation Logic

This section specifies the evaluation conditions associated with evaluation sets that we specified in the structure section. An evaluation condition can be applied on one or more set. In fact, flexibility and the power of the GDL reside in the richness of conditions specification.

Before specifying the conditions we declare any user values and/or user functions that we intend to use inside these conditions.

### 5.4.13 User values

Evaluation logic is usually expressed using constant values of HTML elements. For example, to check if an image has an alternative text, we need to specify the logic: IMG.alt !=NULL or IMG.alt<>"" (Empty String).

In order to provide a clear and coherent specification of evaluation conditions, we must declare all values that we use in condition in the section of User_Values. If it is

a simple value (ex. "", 5, 0.3, etc.), we declare it as Simple_Value. If it is a constructed value (ex. Seq{White, Blue, Green}), we declare it as Constructed_Value. Every declared value has an identifier. A previously declared value can be used in the declaration a new constructed value by using its identifier. For example, we may declare V1, V2, ..., Vn, then we declare ListValues=Seq{V1, V2, ..., Vn}.

### Concrete Syntax

```
<!ELEMENT User_Values (Simple_Values?, Constructed_Values?)>
<!ELEMENT Simple_Values (Simple_Value+)>
<!ELEMENT Constructed_Values (Seq_Value | Table_Value | Cp_Value)+>
```

## 5.4.14 Predefined Simple Data Types and Operations

Some simple data types and operations are predefined in the GDL to be used when specifying user values. By examining the syntax of HTML we find that attributes values in a Web page have one of these data type: String (ex. for IMG.alt), Integer (ex. for Table.Width), Float (ex. for Font.Size), and Hexadecimal (especially for colors). We add to them the Boolean data type because we use it in evaluation conditions.

Next we will present the predefined simple data types and the predefined operations of each of them.

### Example

```
<Simple_Value V_ID="White" Value="#FFFFFF" Type="Hex"/>
<Simple_Value V_ID="MaxFontNumber" Value="3" Type="Integer"/>
```

The predefined simple data types are:

### ➔ BOOLEAN

This type is for Boolean values, i.e. *True* and *False*. The predefined operations on this type are: NOT, AND, OR, =.

### ➔ INTEGER

This type includes the positive and negative integer numbers. The predefined operations on this type are: INTEGER, ODD, EVEN, +, -, *, %, MOD, DIV, =, >, >, <=, >=.

### ➔ FLOAT

This type is for real numbers. The predefined operations on this type are: FLOAT, +, -, *, /, =, >, >, <=, >=.

### ➔ HEX (Color)

We introduced this data type to deal with colors because in HTML they are generally expressed as hexadecimal values (for example, Red=#FF0000). Only 16 of them are widely known by their names with their sRGB values:

| | |
|---|---|
| Black = #000000 | Green = #008000 |
| Silver = #C0C0C0 | Lime = #00FF00 |
| Gray = #808080 | Olive = #808000 |
| White = #FFFFFF | Yellow = #FFFF00 |
| Maroon= #800000 | Navy = #000080 |

| | | | |
|---|---|---|---|
| Red | = #FF0000 | Blue | = #0000FF |
| Purple | = #800080 | Teal | = #008080 |
| Fuchsia | = #FF00FF | Aqua | = #00FFFF |

The operations on this type are: =, getRED, getGREEN, getBLUE.

### ➔ STRING

This type is for strings of characters. The predefined operations on this type are: LENGTH, SUB_STRING, +, =, <, >.

## 5.4.15 Predefined Constructed Data Types and Operations

In addition to the simple data types, we introduced some constructed data types that can be used to define user values.

Contrarily to simple data types, we need specify to define the constructed data types in the GDL DTD to explicit their syntax.

The predefined constructed data types and their operations are:

### ➔ The Set Type Constructor (SET)

A set of values of a given data type is characterized by the facts that:

- The order of the members in the set is not relevant;
- The number of identical members in the set is not relevant;
- A set may be empty.

### Example

```
<Set_Value ID_Value="Colors">
        <Id_Value ID_Ref="White"/>
        <Id_Value ID_Ref="Black"/>
        <Id_Value ID_Ref="Red"/>
        <Id_Value ID_Ref="Blue"/>
        <Id_Value ID_Ref="Yellow"/>
        <Id_Value ID_Ref="Cyan"/>
</Set_Value>
```

The operations predefined on sets are: EMPTY, SIZE, IN, UNION, INTERSECT, DIFF, INCLUDE, EQUAL.

### ➔ The Cartesian product type constructor (CP)

A Cartesian product of values of given type(s) is characterized buy the facts that:

- A value at a given place in the CP has always the same type;
- A CP value has always the same number of fields;
- There must be at least one field in a CP.
- A field can also be named so that we can refer to it by its name.

### Example

```
<Cp_Value V_ID="ColorsForWhiteBg">
        <Selector Name="Good">
```

```
                <Seq_Value>
                        <Id_Value ID_Ref="Black"/>
                        <Id_Value ID_Ref="Red"/>
                        <Id_Value ID_Ref="Blue"/>
                </Seq_Value>
        </Selector>
        <Selector Name="Bad">
                <Seq_Value>
                        <Id_Value ID_Ref="Yellow"/>
                        <Id_Value ID_Ref="Cyan"/>
                </Seq_Value>
        </Selector>
</Cp_Value>
```

We predefined the following operation on CP:

**Id$_i$**: CP[id$_1$:T$_1$, ..., Id$_i$: T$_i$, ...] → T$_i$

Idi(*x*) returns the value of Id$_i$ of *x*. An operation is defined for each field of the CP.

### ➔ The Sequence type constructor (SEQ)

A sequence of values of a given type is characterized by the facts that:

- The order of the elements in the sequence is relevant;
- The number of identical elements in the sequence is relevant;
- A sequence may be empty.

The operations predefined on SEQ are the followings: EMPTY, SIZE, ELEMENT, SUB_SEQ, IN.

### ➔ The Table type constructor (TABLE)

A table is constructed on two types: the type of the elements of the table and the type of the index. A table of elements of type T1 indexed by T2 is characterized by the membership function which associates for each value of type T2 a value of type T1+{UNDEF} (i.e. a value of type T1 or the special value UNDEF).

### Example

```
<Table_Value V_ID="MapTable">
        <Table_Index>
                <Id_Value ID_Ref="White"/>
        </Table_Index>
        <Table_Elements>
                <Table_Element>
                        <Id_Value ID_Red="ColorsForWhiteBg">
                </Table_Element>
        </Table_Elements>
</Table_Value>
```

The predefined operations on TABLE are: EMPTY, IN, [].

### 5.4.16 Evaluation Conditions

In this section we specify the evaluation logic that must be applied on the captured usability and/or accessibility data (instances of the evaluation sets) in order to check the respect/ violation of the targeted set of guidelines.

In order to have high flexibility level we introduced some concepts concerning the specification of evaluation conditions. The major concept is that of Meta conditions (see section 4.2.5).

This section of a specification contains zero or more Meta evaluation condition(s), a number of direct evaluation conditions or concrete evaluation conditions less or equal to the number of evaluation sets because we must have at least one evaluation condition per evaluation set, but one evaluation condition may be applied on more than one evaluation set.

A direct condition and a Meta condition are specified in the same way. The only difference is that, for Meta conditions we use Meta variables in their expressions to provide generic evaluation logic that can later be mapped to concrete one(s), whereas in direct evaluation conditions we use concrete user values and elements from the associated evaluation sets.

### 5.4.17 Meta evaluation Condition

A Meta evaluation condition is composed of:

- **At least one Meta variable**: a Meta variable has an identifier and is of one of the predefined simple or constructed data types.
- **One Meta model**: the model is the place where we specify the evaluation expression. The model's utility is just to separate the variables from the list of operations defined in the Meta condition.

### 5.4.18 Operations in evaluation conditions

As the syntax of the Meta condition depicts, its model can be specified as a sequence of smaller operations. This way of specifying evaluation expression is relatively complex, but it enables us to add some addition information especially for controlling the execution of the expression and for providing specific output messages.

An operation is composed of one or more arguments, and its execution triggers one of many possible actions. We provide the argument's position in the operation header explicitly. An action is triggered by a specific result value of the operation execution.

#### Example

The following operation checks whether the set instance has an instance of the bg (background color) element or not. If the result is true, it jumps to the operation Op2, otherwise, it stops the execution and sends a warning message.

```
<Operation ID_Operation="Op1" Op_Symbol="setInstanceHasElement"  Return_Type="Boolean">
        <Argument Arg_Type="Var" Arg_Value="set"/>
        <Argument Arg_Type="Var" Arg_Value="bg"/>
        <Action Result="true" What="Jumb" Where="Op2"/>
```

```
        <Action Result="false" What="WarnStop" Why="No bgcolor."/>
</Operation>
```

### 5.4.19 Direct Evaluation Condition

A direct evaluation condition is the same as a Meta evaluation condition, but instead of Meta variables, we directly use identifiers of elements of evaluation sets on which the condition must be applied.

The same difference between direct and Meta conditions exists between operations in direct conditions and operations in Meta conditions. Their syntax is identical, but in the first case, we use identifiers of sets elements instead of identifiers of Meta variables.

#### Example

Following is the specification of the formal guideline for evaluating the guideline "*Ne pas utiliser plus de 2-3 polices de caractères*" [Nogier 2002].

To evaluate the guideline we will provide a default interpretation: "*Ne pas utiliser plus de 3 polices de caractères*". We will need the HTML element Font.face only. Therefore, we will have one evaluation set {Font.face[Page]} and finaly, we will need one direct evaluation condition: the number of instances of this set is inferior or equal to 3.

```
<Formal_Guideline>
<Evaluation_Structure>
        <HTML_Elements>
                <HTML_Element ID_Element="E1" Tag="Font" Attribute="face"/>
        </HTML_Elements>
        <Evaluation_Sets>
                <Evaluation_Set ID_Set="S1" Priority="1">
                        <Set_Element E_ID="E1" Scope=”Page”/>
                </Evaluation_Set>
        </Evaluation_Sets>
</Evaluation_Structure>

<Evaluation_Logic>
        <User_Values>
                <Simple_Values>
                        <Simple_Value ID_Value="MaxFontNbr" Value="3" Type="Int"/>
                </Simple_Values>
        </User_Values>
        <Evaluation_Conditions>
                <Direct_Condition ID_Condition="C1" ID_Set="S1">
                        <Operation ID_Operation="Op1"  Symbol="NumberOfInstances"
                                Return_Type="Int">
                        <Argument Arg_Type="SET" Arg_Value="S1" Pos="1"/>
                        <Action Result=”ANY” What=”Jump” Where=”Op2”/>
                  </Operation>
                  <Operation ID_Operation="Op2" Symbol="LESS"   Return_Type="Boolean">
                        <Argument Arg_Type="Op" Arg_Value="Op1" Pos="1"/>
                        <Argument Arg_Type="Val" Arg_Value="MaxFontNbr" Pos="2"/>
                        <Action Result="true" What="Success"/>
                        <Action Result="false" What="Error" Why="fonts >3."/>
                  </Operation>
```

```
                </Direct_Condition>
        </Evaluation_Conditions>
</Evaluation_Logic>
</Formal_Guideline>
```

### 5.4.20 Mapped Evaluation Condition

A mapped evaluation condition is the instantiation of a Meta evaluation condition for a given evaluation set. The instantiation is realized by using a set of Meta mappings to map between Meta variables in the Meta condition and concrete corresponding elements in evaluation sets.

Now that we introduced all the theoretical basis of our approach, we will discuss two aspects related to automated evaluation: defining a kind of predictive function to predict the evaluation result, and evaluating the feasibility of the proposed approach.

## 5.5 Evaluation function

Theoretically, we could follow different evaluation approaches (strategies):

- **By page**: we evaluate a specific page by identifying the object found in the page, and then we select the guidelines related to these objects only. For example, if we know that our page contains only textual content, we could unselect (ignore) all guidelines related to multimedia content.
- **By object**: we decide to focus the evaluation on a single object of the page (the form F1, the action button Btn2, etc.), or a type of objects (Frames, Tables, action buttons, etc.). In this case, we evaluate the guidelines related to these objects only.
- **By a set of guidelines**: we evaluate the whole set of guidelines on the whole page. This set can have guidelines issued from different sources.
- **By a sub-set of guidelines**: we choose to evaluate a sub-set of guidelines (from one or more sources) on the whole page.
- **By priority**: we choose to evaluate sub-guidelines (evaluation sets) by selecting a priority level: 1, 2, or 3.
- **By Ergonomic Criterion**: we select the ergonomic criteria (usability, accessibility, etc.) or sub-criteria (consistency, flexibility, background consistency, etc.) to be checked.
- **Checking vs. Review**: in guideline checking we evaluate if a guideline is respected/violated, thus, if one guideline-related aspect is violated, we consider that the guideline is violated, thus, will only have two possible result values (respected, violated). In guideline review we examine to which degree a guideline is respected/violated, thus, the result we be respect for some aspects of the guideline and violation for others. The importance (priority) of aspects determines the global evaluation result.

Having all the needed information, we wish to introduce a function named EVAL that formalizes the expression of the evaluation result in function of some evaluation parameters (selected guidelines, selected evaluation scope, etc.) that we can use to configure an evaluation session.

The GDL allows us to define a very flexible evaluation function. We propose the following evaluation parameters:

- **Evaluation strategy parameters**:
  - *By guideline*: we select the list of guidelines to be reviewed in the evaluated page. This can be done by the traditional way used in most existing evaluation tools (separated sets of guidelines) or by selecting a mixture of guidelines from the available guideline sets.
  - *Evaluation by Ergonomic Criterion*: we select the ergonomic criteria (usability, accessibility, etc.) or sub-criteria (consistency, flexibility, background consistency, etc.) to be checked. The tool then identifies the available guidelines related to these criteria.
  - *Evaluation by priority*.
  - *Evaluation by page object*: we select some objects (tables, fonts, images, etc.). The tool identifies the available guidelines related to these objects.
  - *Checking vs. Review*.
- **Efficiency Parameters**:
  - Capture all or N instances of evaluation sets during the parsing phase.
  - Evaluate all or N of the captured instances of evaluation sets related to the selected guidelines.
  - Stop the evaluation after detecting N violations of the same guideline, or stop the evaluation after detecting N violations of the evaluated guidelines. Of course, if we used stop parameters, the parsing and evaluation steps must be executed simultaneously. Notice that such stopping parameters are useful only if we want to evaluate the page (for example, a standard responsible uses the tool to see if the page or the site is conforming to a desired set of guidelines). If we want to repair the page (ex. a designer is using the tool to detect all ergonomic problems), the tool must detect a maximum of the existing problems.
- **All possible combinations of the above parameters**: for example, we can select to evaluate W3C guidelines related to images, evaluated all available priority 1 guidelines but only priority 2 and 3 guidelines related to usability, etc.

Notice that, whatever the parameters used, we will end by a list of guidelines to be evaluated, thus, a list of the evaluation sets for which we will search the instances in the evaluated page.

In order to define the EVAL function precisely, let:

- A Web page $p$.
- $j=1,\ldots, m$ guideline sources.
- $i=1,\ldots, n$ guidelines in a source.
- $EVAL\_SET_{i,j}$ the set of evaluation sets associated to the guideline $G_{i,j}$ and that will be used for the evaluation of the evaluated page.
- $INST\_EVAL\_SET_{i,j}$ the set of captured instances of $EVAL\_SET_{i,j}$ in $p$.

- EVAL_CONDi, j the set of k conditions associated to EVAL_SETi, j.

$EVAL\_COND_{i,j} = \{EVAL\_COND^{1}_{i,j}, EVAL\_COND^{2}_{i,j}, \ldots, EVAL\_COND^{k}_{i,j}\}$

In **guideline checking**, we define the EVAL function as follows:

$EVAL\ [p, EVAL\_SET_{i,j}] = Check(EVAL\_COND_{i,j}\{INST\_EVAL\_SET_{i,j}\})$
$= \{\text{“Respected” | ”Violated”}\}$

Where, the function Check consists in executing the evaluation conditions on the captured UD.

In practice, the EVAL function checks the satisfaction of each ECi.j condition, and then it combines the results to have the overall result for the guideline. Some guidelines are "Respected" if the execution of all the corresponding evaluation conditions, over all the captured instances of the evaluation sets associated with the guideline, is positive. If at least one condition is negative, the corresponding guideline(s) is considered "Violated".

We can formalize these evaluation results through the next three definitions.

### 5.5.1 "Respected" Guideline

We say that a page satisfies a guideline (i.e. the guideline is respected) if and only if all conditions of this guideline are satisfied on all the instances of the guideline related evaluation sets. Formally:

$EVAL[P, G_{i.j}] = \text{“Respected”} \Leftrightarrow \wedge\ EVAL\_COND_{i,j}\{INST\_EVAL\_SET_{i,j}\} = TRUE$

### 5.5.2 "Violated" Guideline

We say that a page does not satisfy a guideline (i.e. the guideline is violated) if and only if at least one condition of this guideline is not satisfied. Formally:

$EVAL[P, G_{i.j}] = \text{“Violated”} \Leftrightarrow \exists\ k: EVAL\_COND^{k}_{i,j}\{INST\_UES_{i,j}\}=FALSE$
$\Leftrightarrow \vee\ EVAL\_COND_{i,j}\{INST\_EVAL\_SET_{i,j}\} = FALSE$

Notice that generally this is not accurate, because we do not distinguish between violating a priority 1 ergonomic aspect or priority 3 one. A page that violates only priority 3 aspects of a guideline could still be considered (very) good page.

In fact, the concept of evaluation function is inspired from [Leporini 2003]. Leporini conducted a study to define the usability of Web sites, in order to improve their accessibility for “special users”, who are obliged to navigate on the internet through screen readers. She proposed 19 criteria (general principles) and 54 checkpoints defining each criterion (technical solutions); then, she specified possible ways of application of such criteria and checkpoints. Leporini defined a formal EVAL function that takes one page, and returns the check results computed by using her proposed checkpoints. The function result is the application status of a criterion (thus the checkpoints needed to check it) to a Web page. The result can be "Applied", "Not applied" and "To be reviewed".

The GDL enables a GDL-based evaluation tool to review guidelines in a flexible manner. In fact, as we have structured guidelines in term of evaluation sets, we can review these guidelines to generate more accurate result than just “Violated”,

"Respected". To do this, we will need a quality model to balance the evaluation result.

In the next section, we will present a simple quality model before formalizing the result of the function EVAL in the case of GDL-based guideline review.

### 5.5.3 A Quality Model for the evaluation result

Using the above evaluation parameters allows us to define a kind of quality model [Brajnik 2001; Brajnik 2002] to balance the evaluation result. Contrarily to the binary model used by most existing evaluation tools (a guideline is violated? Yes or No), we can use a weight concept to express the evaluation result. This weight can be predefined as default values associated with the evaluation parameters. And for more flexibility and customization ability, the evaluator should be able to change these values before an evaluation session.

The simplest quality model is the following:

- All ergonomic criteria and page objects have the same weight. Thus, to respect usability or accessibility or other criteria gives the same result. On the contrary of this, we could say, for example, that in a given context, it is usability is more important than accessibility. In this case, we could say that usability criteria have more weight that accessibility ones.
- Priority (1) evaluation sets have a weight of 0.7.
- Priority (2) evaluation sets have a weight of 0.2.
- Priority (3) evaluation sets have a weight of 0.1.

So, let us define the EVAL function in the case of **guideline review**:

Let $NBR\_EVAL\_SET_{i,\,j,\,1|2|3}$ be the number of evaluation sets of priority 1|2|3 that must be evaluated.

These sets are the result of considering all the evaluation parameters during the phase of sets identification.

Let $NBR\_KO\_EVAL\_SET_{i,\,j,\,1|2|3}$ be the number of evaluation sets of priority 1|2|3 for which one or more instances were captured in the evaluated page and gave a negative result (violation).

Let $NBR\_OK\_EVAL\_SET_{i,\,j,\,1|2|3}$ be the number of evaluation sets of priority 1|2|3 for which one or more instances were captured in the evaluated page and gave a positive result (respect).

$$NBR\_EVAL\_SET_{i,\,j,\,1|2|3} = NBR\_KO\_EVAL\_SET_{i,\,j,\,1|2|3} + NBR\_OK\_EVAL\_SET_{i,\,j,\,1|2|3}$$

The positive evaluation result will be:

$$Rp = \frac{(NBR\_OK\_EVAL\_SETi,\,j,1 \times 0.7) + (NBR\_OK\_EVAL\_SETi,\,j,2 \times 0.3) + (NBR\_OK\_EVAL\_SETi,\,j,3 \times 0.1)}{(NBR\_EVAL\_SETi,\,j,1 \times 0.7) + (NBR\_EVAL\_SETi,\,j,2 \times 0.3) + (NBR\_EVAL\_SETi,\,j,3 \times 0.1)}$$

The negative evaluation result will be:

$$Rn = \frac{(NBR\_KO\_EVAL\_SETi, j, 1 \times 0.7) + (NBR\_KO\_EVAL\_SETi, j, 2 \times 0.3) + (NBR\_KO\_EVAL\_SETi, j, 3 \times 0.1)}{(NBR\_EVAL\_SETi, j, 1 \times 0.7) + (NBR\_EVAL\_SETi, j, 2 \times 0.3) + (NBR\_EVAL\_SETi, j, 3 \times 0.1)}$$

Such formula allows the simple classification of the evaluated page according to a classification scale. For example, Rp<0.5 means bad page, 0.5<=Rp<.75 means good page, and Rp>=0.75 means very good page. Such classification could be enough to publish a list of top N usable sites, or for a webmaster who wants to have a rapid estimation of the quality of designed pages before deciding to go into details about usability problems.

Having all the needed information, the next section will discusses the limits of automation of an evaluation based on the proposed methodology.

## 5.6 Feasibility of Automatic Evaluation

In this section, we will classify guidelines according to the automation level of their evaluation with an evaluation tool adopting the proposed methodology.

In our evaluation context, we consider that a guideline can be evaluated if all the information required to verify it are included in the source files: the HTML code.

Before giving our classification, we will examine some classifications that were proposed by researchers in automatic evaluation in UIs.

### 5.6.1 ERGOVAL

ERGOVAL [Barthet 1994] is a theoretical evaluation method for measuring the ergonomic quality of the WIMP UIs. This method corresponds to the checking on the presentation of an interface of the compliance with the ergonomic rules contained in the guides of recommendations.

In ERGOVAL, ergonomic rules are classified on an automation-level basis. It considers that a rule can be automated, whatever the implemented methods, when all of the information required to verify it, can be found in the system. Example: "*Any non-accessible action must be grayed*". At a moment "t", it is virtually possible to know all the actions that can not be accessed. It is also possible to know whether the object of this action is grayed or not. All the information is in the system, therefore the rule can be automated.

After this, rules are classified into two classes: Rules that require information automatically retrievable whatever the implemented methods are, and rules that require information not automatically retrievable whatever the implemented methods are.

After that, each of these classes is devised in two sub-classes: Rules that require information related to items included in the application and rules that require information related to items not included in the application. For both classes, rules are also classified based on the type of information required for running them. A summary of these various classifications is shown in figure 5.6.

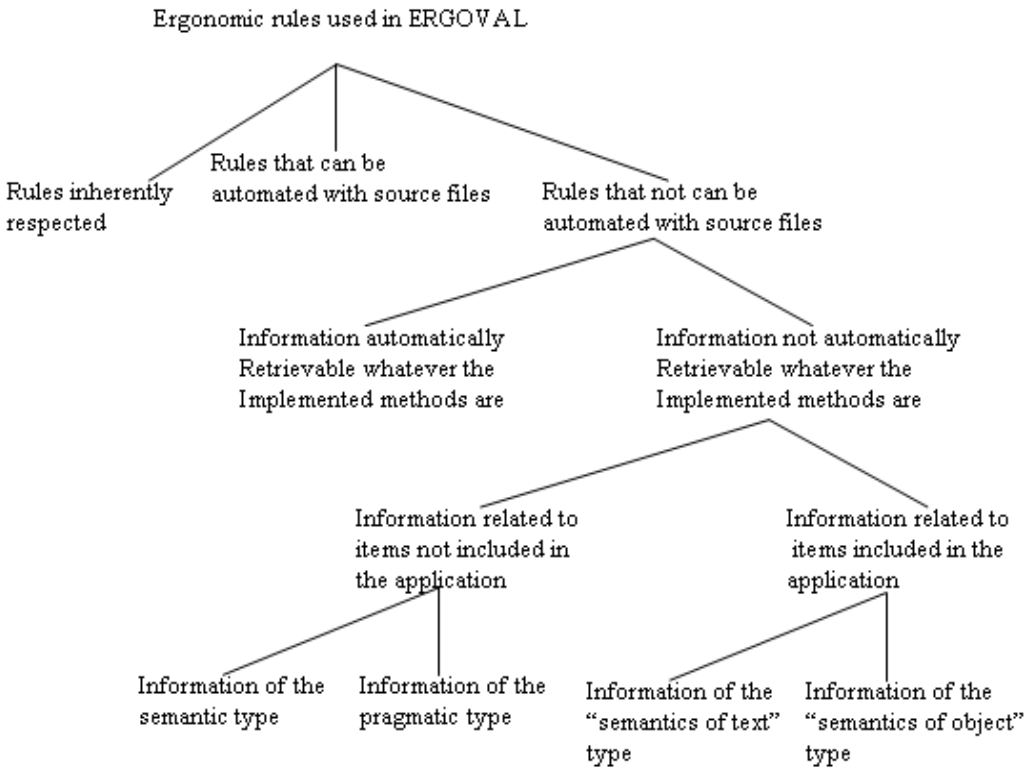


**Figure 5.6**: Automation-based classification of ergonomic rules in ERGOVAL

Table 5.2 shows how the rules are distributed for the above three main categories [Farenc et al. 1996].

| | **Rules/presentation** | **Rules/behavior** | **Total (%)** |
|---|---|---|---|
| **Rules inherently respected (1)** | 28 | 64 | 93 (22.9%) |
| **Rules that can be automated with source files (2)** | 82 | 2 | 84 (20.6%) |
| **Rules that can not be automated with source files (3)** | 161 | 69 | 230 (56.5%) |
| **Total** | 271 | 135 | 406 (100%) |

**Table 5.2**: Summary of the automation feasibility of the evaluation of ergonomic rules in ERGOVAL

An example of a class (1) rule is "Labels for push buttons must be centered". An example of a class (2) rule is "All boxes and windows must have a title". An example of a class (3) rule is "For any input field, if there are any acceptable values, such values must be displayed".

[Farenc et al. 1996] estimated that the minimum number of rules that can be automated ERGOVAL is 44% (22.9% + 20.6%) rules that are automatically verifiable using the resource files. Furthermore, the maximum percentage of ergonomic rules that can be incorporated into a totally automated evaluation is 78%.

One of the main shortcomings explained by the above study is that only resource files of the UI were supposed to be accessible and exploitable. In the case of Web sites, it is highly expected that since HTML code is accessible and exploitable (and not only the resources), more guidelines could be automatically evaluated.

### 5.6.2 WAI guidelines

The big advantage with web sites is that their HTML code can be downloaded and examined remotely, which is not the case for traditional interactive applications. It is therefore expected that the automated evaluation of web design guidelines will go beyond the barrier of 44% thank to the code accessibility. In [Cooper et al. 1998], we find another study that evaluates the automation limits of WAI guidelines [W3C 1999] by Bobby [Cooper 1999]. Bobby has three levels of support for WAI guidelines: manual, partial, and full. Table 5.3 shows the percentage of these levels [Cooper 1998].

| Level/Support | Manual | Partial | Full | Total (%) |
|---|---|---|---|---|
| 1 | 16 | 2 | 8 | 26 (35%) |
| 2 | 14 | 18 | 2 | 34 (45%) |
| 3 | 7 | 6 | 1 | 14 (20%) |
| Total | 37 (50%) | 26 (35%) | 11 (15%) | 74 (100%) |

**Table 5.3**: Repartition of support level for WAI guidelines by Bobby.

If we sum up the partial and full support, we can reach the percentage of 50% of guidelines automatically evaluated, which is only a little bit beyond the 44% barrier.

### 5.6.3 ISO 9241 - 12

Another automation estimation of a specific set of guidelines is the classification of the guidelines of section 12 of the standard ISO 9241 [Esselinckx 2000]. Essenlinckx studied the 97 guidelines of section 12 in order to determine what is needed to incorporate these guidelines into an automatic evaluation tool. Inspired by the classification of Farenc [[Farenc et al. 1996]], Esselinckx classified the targeted guidelines as depicted in figure 5.7.

Some examples of the different categories are:

- **Guidelines are not applicable to Web sites**: if the same displayed information is used by many users/operators in simultaneous interaction, it is convenient to provide for every user a visually distinct cursor and/or a pointer.
- **Web guidelines naturally respected**: authorize the user to select the windows format and to save it as default format.
- **Guidelines easily implemented with HTML code**: provide a unique identifier for every window (ex. its title).
- **Guidelines more difficult to implement with HTML code**: it is convenient that labels be grammatically coherent.

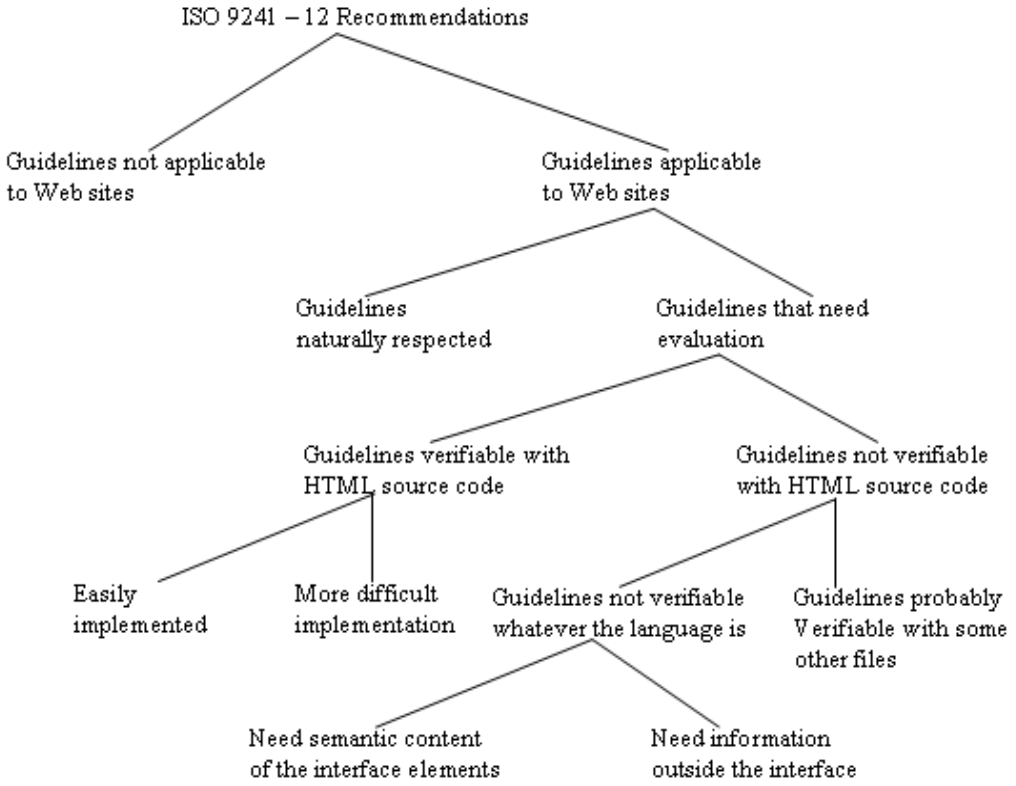


**Figure 5.7**: Classification tree of ISO 9241 - section 12 recommendations

Table 5.3 shows the classification result of ISO 9241 – 12 recommendations [Esselinckx 2000].

| Category | | Number of Guidelines | Applicable guidelines (%) |
|---|---|---|---|
| Guidelines not applicable to Web sites | | 7 | 7 |
| Guidelines naturally respected | | 22 | 24.4 |
| Guidelines that can be automated | Easily implemented with HTML code | 19 | 21.1 |
| | More difficult implementation | 17 | 18.9 |
| | Probably verifiable with some other files | 8 | 8.9 |
| Guidelines that cannot be automated | Not verifiable because need semantic content of elements | 9 | 10 |
| | Not verifiable because need information outside the interface | 15 | 16.7 |
| Total | | 97 | 100 |

**Table 5.3**: Classification of ISO 9241 – 12 recommendations

As a conclusion of this study, it is possible to automate the evaluation of up to 73% of ISO 9421-12 guidelines: guidelines naturally respected (24.4%) + guidelines that can be automated (21.1%+18.9%+8.9%).

Now, let us try to see the automation limits based on the proposed evaluation approach.

### 5.6.4 Automation limits of our approach

The classification based on our approach would have some common parts with the one given in [Esselinckx 2000] because we are dealing with Web interfaces (sites). Figure 5.8 depicts our classification of guidelines.

From our point of view, targeted guidelines can be:

- **Concrete**: the guideline expression makes clear reference to HTML elements. Example, "Documents shall be organized so they are readable without requiring an associated style sheet" [Section508].
- **Abstract**: in this case, we try to interpret the guideline in the targeted evaluation context. Interpretation means re-expressing the guideline in concrete way. Example, "*Provide equivalent alternatives to auditory and visual content*" [W3C 1999].
- By definition, concrete or interpreted guidelines are **theoretically verifiable with HTML** source code of the evaluated pages if we can find HTML elements to verify the respect or violation of the guideline. If the evaluator considers that the identified useful HTML elements are sufficient to evaluate all the aspects of the guideline, then the guideline is said to be **theoretically totally verifiable**, else, the guideline is said to be **theoretically partially verifiable**. In fact, this level corresponds to the structuring of guidelines (evaluation sets) in term of HTML elements.
- Theoretically verifiable guidelines can then be practically verifiable or not. A guideline is said to be **practically verifiable** when we can provide a GDL expression of the evaluation logic that must be applied on evaluation sets to verify the guideline. As figure 5.10 shows (easy or more difficult implementation), the possibility to provide an expression is related to its availability and not to the difficulty of implementing it. When we cannot provide any evaluation expression, we consider that the guideline is **not practically verifiable**.
- It is probable that, although we find HTML elements to structure a guideline, we cannot implement the evaluation of all these elements. In this case, the guideline is said to be **practically partially verifiable**, else, it is practically **totally verifiable**. For example, in the case of our example about color, we consider that the interpreted guideline is **practically verifiable** because we used the research results of Murch, but it is **partially verifiable** because these results concern basic 8 colors only.

Notice that a guideline status may change for many raisons like:

- New research results that enable the expression of some old-non expressible evaluation logic.
- New technologies that provide new possibilities of reflecting the guideline semantic in Web pages or new possibilities to touch more content of the evaluated page.
- The HTML experience of the person in charge of structuring the guideline, or his/her interpretation of the guideline semantics.

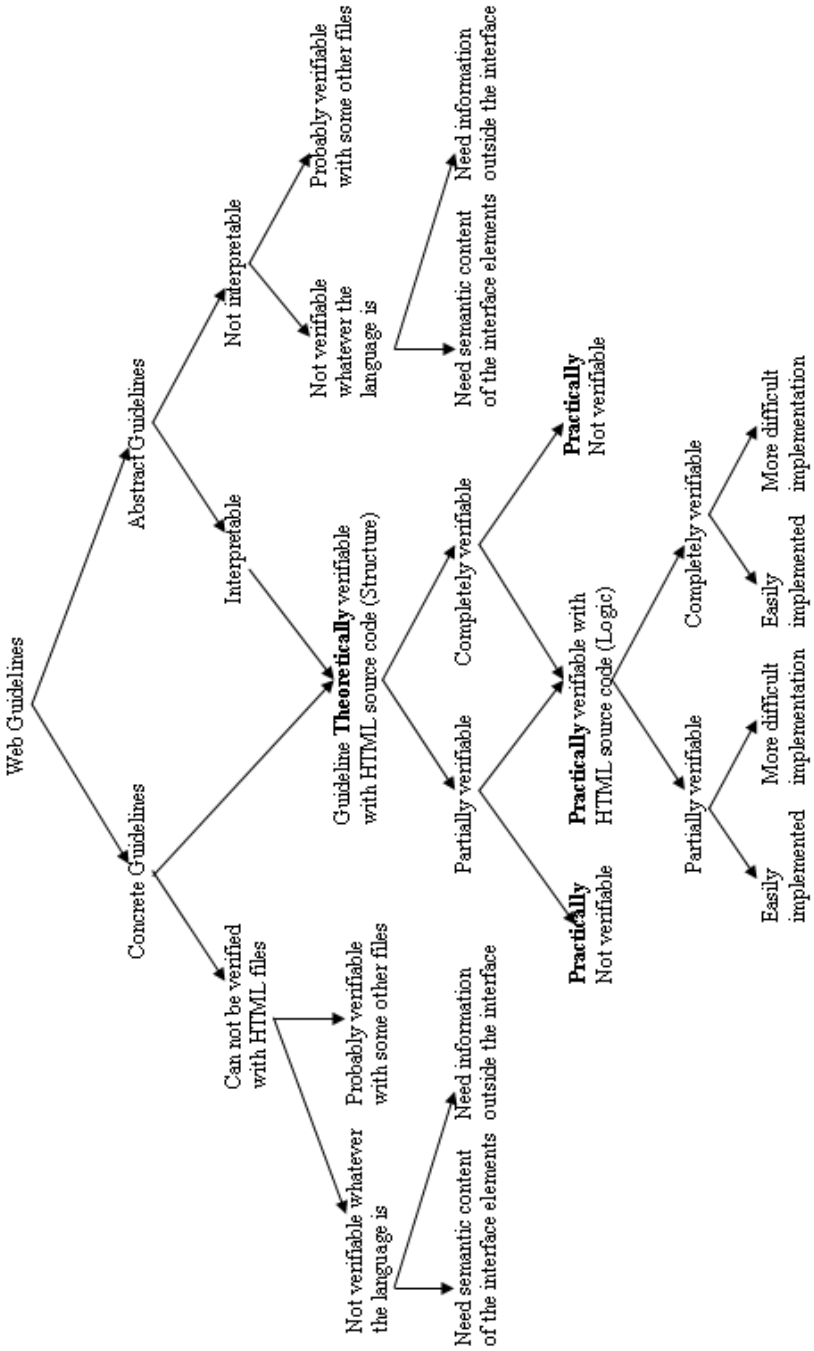


**Figure 5.8**: Automation-oriented classification of guidelines according to the proposed evaluation approach

As a conclusion, we can give estimation of the evaluation automation limits of our approach based on the above Web-related studies (Bobby, ISO9421). In fact, each of these studies dealt with a single set of guidelines, which is not our case. We propose an approach to automate evaluation of any Web guideline as soon as it is possible to evaluate it with HTML code only; therefore, a tool based on our approach is theoretically able to automate the evaluation of:

- All Bobby fully automatable guidelines because their evaluation is theoretically and practically totally feasible.

- All the ISO 9421-12 naturally respected guidelines and automatable guidelines.
- For the remaining guidelines, like those partially supported by Bobby, the estimation of their automation levels depends on the interpretation context. Contrarily to our approach, Bobby does not consider contexts, therefore, some of these guidelines can be considered totally supported in some contexts.

Therefore, we estimate that our approach is theoretically able to go beyond the Bobby limits. To confirm this supposition, we need to apply the approach on a sufficient number of guidelines partially supported by Bobby.

## 5.7 Summary

In this chapter we presented the detailed syntax of a formal language to support the proposed methodology by formalizing the definition of Web usability and accessibility guidelines based on the concepts of the framework of chapter 4 in addition to some other concepts related to the operationalization of the framework's concepts in the context of Web automated evaluation.

The chapter provided the GDL with respect to:

- Its semantics;
- Its abstract syntax that can be used to implement a general GDL interpreter.
- It concrete (XML-compliant) that specifies the GDL DTD.

A deep examination of the GDL syntax should enable us to highlight the advantages underlined in the summary of chapter 4:

- The flexible structure of a formal guideline: HTML elements, evaluation sets, 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: by page, by (sub) guideline, by object, by ergonomic criteria.
- The examples provided all over this chapter give a proof-of-concept of the feasibility of the approach and the ability to evaluate complex guidelines. This fact will be reinforced in next chapter by applying the approach on various types of guidelines.

In addition, we can now underline the following advantages:

- Evaluators do not 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. Normally, it is the responsibility of human factors expert to formalize the guidelines.
- Meta evaluation conditions and operations provide a powerful mechanism to improve evaluation of relatively similar guidelines. This improvement 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 desired to consider all these sources in order to have sites of high ergonomic rating.

- Using a quality model to balance the evaluation result enables the evaluator to manimize the subjectivity of the evaluation.