WAEX: Web Accessibility Evaluator in a single XSLT file

Vicente Luque Centeno, Carlos Delgado Kloos
Carlos III University of Madrid
Email: {vlc,cdk}@it.uc3m.es

Martin Gaedke, Martin Nussbaumer
University of Karlsruhe
Email: {gaedke,nussbaumer}@tm.uni-karlsruhe.de

***Abstract*— Web accessibility rules, i.e., the conditions to be met by Web sites in order to be considered as accessible for all, can be (partially) checked automatically in many different ways. Monolitic applications built on top of procedural programming languages like C, Java, Perl or PHP have been developed during the last years. These applications usually provide their own interpretation of the W3C guidelines, so it is easy to obtain different evaluation results when different evaluation tools are applied to a common sample page.**

Since most accessibility rules are involved with the markup of Web sites and XSLT can be applied to any selectable-by-condition markup, we have developed WAEX as a Web Accessibility Evaluator in a single XSLT file. Such XSLT file contains 50+ singular accessibility rules and XHTML-specific conditions that are in the prose of the XHTML specification because they could not be included in the grammar specification (the DTD or the XML Schemas). This approach of using XSLT as a declarative repository of accessibility rules has lead to a portable and reusable style sheet that generates rather complete (if compared with already existing evaluation tools) accessibility reports for any Web page.

I. Introduction

WAI (Web Accessibility Initiative)'s WCAG (Web Content Accessibility Guidelines) 1.0 [1] from W3C has become an important reference for Web accessibility in the Web community. It has been accepted as a set of guidelines that improve accessibility and eliminate barriers on Web sites. The lack of accessibility affects a large amount of people (between 10% and 20%) that frequently find important barriers when trying to navigate through today's Web sites. Barriers reduce accessibility not only for people with some sort of personal disability (vision or hearing impairment, mobility problems to manage a keyboard or a mouse, or cognitive and neurological problems). Accessibility is also a major step towards device independent Web design, allowing Web interoperability to be independent from devices, browsers or operating systems. Web accessibility allows having cheaper Web site maintenance and a wider target public. Some governments are also requiring that some Web sites become accessible.

The set of the 65 WCAG's checkpoints that accessible documents have to pass is a very heterogeneous set of constraints, which are difficult to evaluate and repair. Both WCAG 1.0 [1] and the HTML Techniques for the new WCAG 2.0 draft [2] specifications are written at a high abstraction level, which is frequently quite open to subjective interpretation (as *too long texts* or *too many* elements), including implicit conditions or, simply, containing constraints whose detection cannot be automated.

There are several tools nowadays which can help us to detect accessibility barriers on Web sites. Watchfire[9], Tawdis [10] or HERA [11] are only a few of them. It is well known that all these tools require a person to **supervise** and **complete** the results of the evaluation tools because a lot of rules are relayed on manual checks by the user. However, we still have the problem of different *particular interpretations* on each tool. This is due to the fact that these tool's authors have had different subjective interpretations of the W3C's guidelines, as depicted at [17]. As a result, we can easily find several tools reporting different evaluation results for a same sample page. Even further, it is difficult indeed to find two evaluation tools that evaluate the same conditions.

One of the main reasons for this heterogeneity is due to the way these tools have been implemented. Since they have been mostly implemented using **procedural** programming languages, the conditions being checked are unclear for people using them for Web site evaluation. Our approach to solve that problem consists on providing **declarative** rules, readable and reusable by anyone, even for those with little programming background.

Our approach lead us to build WAEX [15]: a Web

Accessibility Evaluator in a single XSLT file. Because of its XSLT nature, it can only be applied to well formed HTML pages. Fortunately, HTML reparation software like Tidy [13] and the HTML parser of libxml [14] allows us to apply WAEX to any Web page.

The rest of this paper depicts the most important WAEX's rules, and is organized as follows: Section II deals with accessibility conditions already expressed formally in the XHTML grammar (the DTD or XML Schema) and compares them with the corresponding XSLT templates in WAEX. Section III deals with those accessibility conditions expressible in a XHTML grammar, but not expressed indeed in a DTD or Schema. Section IV covers accessibility conditions not expressible in a XHTML grammar. Section V contains some conclusions and future work.

II. Accessibility conditions already expressed in the XHTML grammar

One of the most important accessibility issues involve validation against a public grammar. In fact, the XHTML grammar already represents some well known accessibility checkpoints. Specific rules in the DTD or XML Schema of XHTML already require that specific mandatory markup properly appears in XHTML documents. For instance, table I contains rules that state that every image must always have an `alt` attribute. While both the DTD, the XML Schema declare the `alt` attribute as **required** for every image, XSLT can be used to spot any image having no such `alt` attribute as a barrier. Such a rule in WAEX may easily report the absence of this important attribute for accessibility (the `alt` attribute), because all these rules state that every image with no `alt` attribute would be considered as faulty since users that cannot see the image can't read a textual alternative either.[1] [2]

Mandatory elements can also be required by a grammar. For instance, table II contains rules to assert that every document has a `title` (an essential element for the user to differentiate Web pages) and that it must be directly inside the `head` element. In fact, they also assert that only a single title must be present. Having none or multiple titles is considered as an error. While both the DTD and XML Schema express that condition in a regular expression, XSLT may be used to assert that the title must exist and it must be unique.

Validation is also useful for detecting forbidden or deprecated markup within a document. It is important to avoid deprecated markup in order to properly separate structure from presentation. For instance, bullets in list items should be specified only by a CSS (Cascading Style Sheet), instead of using the list's `type` attribute, because they are considered as a *presentational* feature. Even though such an attribute is allowed in XHTML Transitional [2], it has been eliminated on XHTML Strict and successive XHTML versions because it has been deprecated. Table III states that lists should no longer have any such `type` attribute.[3] The XSLT expression provided in table III **explicitly** forbids this, while DTD and XML Schema declare that **implicitly**.

Deprecated and forbidden elements can also be detected if a modern XHTML grammar is being used for validation. Deprecated elements like `b` (bold), `i` (italic), `tt` (tele-type), `center` or `font`, as well as forbidden elements not in the HTML specification like `blink` or `marquee`, can be spotted (as presentational markup that should be left out in favour of CSS) by either a grammar which does not recognize them as valid elements or with an XSLT rule like the one at table IV.

III. Accessibility conditions expressible (but not expressed) in a XHTML grammar

XHTML is not a single language. Since its birth, it has had several versions, starting from Transitional and Strict [2], launching XHTML Basic [4] and ending by XHTML 1.1 [3]. Those different languages have different restrictions, some of them had been previously declared in the 1999's WCAG [1]. For example, the `alt` attribute is mandatory since XHTML Transitional 1.0. Deprecated elements like `font` or `center` were removed in XHTML Strict 1.0. XHTML 1.1 introduced some new rules and removed some deprecated features from XHTML Strict 1.0. XHTML Basic does not allow frames, plugins, scripts, both server or client side image maps or nested tables for layout, thus implying that those WCAG rules referring to these elements simply can't be broken. XHTML Basic was designed as the minimum subset of XHTML tags and attributes that all browsers should understand, especially oriented to small

[1] A similar set of expressions can be used to state that the alt attribute is also required for every `area` element.

[2] Ornamental images containing no information, should have an empty string as the textual alternative, but this attribute must be present anyhow according to WCAG. The fact that the textual alternative is an adequate alternative for the image is another checkpoint which is outside of our scope.

[3] Other deprecated attributes like `bgcolor` or forbidden ones like `bordercolor` are also implicitly forbidden by grammar.

	Expression
DTD	<!ATTLIST img alt CDATA **#REQUIRED**>
XML Schema	<xs:element name="img"> <xs:complexType> <xs:attribute name="alt" use="**required**" type="xs:string"/> </xs:complexType> </xs:element>
XSLT	<xsl:template match="//img[not(@alt)]"> <xsl:message>Images with no alt attribute</xsl:message> </xsl:template>

TABLE I

IMAGES WITHOUT ALTERNATIVE TEXT. THE `img`'S `alt` ATTRIBUTE IS MANDATORY.

	Expression
DTD	<!ELEMENT head (%head.misc;, ((**title**, %head.misc;, (base, %head.misc;)?) | (base, %head.misc;, (**title**, %head.misc;)))))>
XML Schema	<xs:element name="head"> <xs:complexType> <xs:sequence> <xs:group ref="head.misc"/> <xs:choice> <xs:sequence> <xs:element ref="**title**"/> <xs:group ref="head.misc"/> <xs:sequence minOccurs="0"> <xs:element ref="base"/> <xs:group ref="head.misc"/> </xs:sequence> </xs:sequence> <xs:sequence> <xs:element ref="base"/> <xs:group ref="head.misc"/> <xs:element ref="**title**"/> <xs:group ref="head.misc"/> </xs:sequence> </xs:choice> </xs:sequence> </xs:complexType> </xs:element>
XSLT	<xsl:template match="/html[count(./head/title)!=1]"> <xsl:message>Document without a unique title</xsl:message> </xsl:template>

TABLE II

DOCUMENT WITHOUT A UNIQUE TITLE. DOCUMENT'S TITLE IS MANDATORY.

	Expression
DTD	*Though declarations such as* <!ATTLIST ul type "(disc|square|circle)" #IMPLIED> *might exist (in older XHTML versions), such attributes should not be used*
XML Schema	*Though declarations such as* <xs:attribute name="type"/> inside <xs:element name="ul"/> *might exist (in older XHTML versions), such attributes should not be used*
XSLT	<xsl:template match="(//ul|//ol)[@type]"> <xsl:message>List item's bullet should be specified by CSS</xsl:message> </xsl:template>

TABLE III

LIST ITEM'S BULLET SHOULD BE SPECIFIED BY CSS

	Expression
DTD	*No such* <!ELEMENT b ...> *nor* <!ELEMENT i ...> ... *(in new XHTML versions)*
XML Schema	*No such* <xs:element name="b"/> *nor* <xs:element name="i"/> ... *(in new XHTML versions)*
XSLT	<xsl:template match="//b | //i | //tt | //font | //center | //blink"> <xsl:message>Deprecated elements no longer in use</xsl:message> </xsl:template>

TABLE IV
DEPRECATED ELEMENTS NO LONGER IN USE

devices with smaller capabilities. Its usage involves that presentation completely relies on CSS.

The refinement process started by XHTML 1.0 and ended by XHTML 1.1 has already included rules expressible in a grammar into successive DTDs and/or XML Schemas. However, some refinements still remain uncompleted. For example, WCAG 1.0 states that all frames should have a mandatory title attribute. Even though this rule is very similar to the one exposed at table I, the XHTML grammar still defines this attribute as optional, instead of required. For this reason, rules from table V are required, unless we locally redefine this rule in our own DTD or XML Schema.[4]

Table VI contains a summary of XHTML prohibitions stated in the prose of XHTML specification. Such restrictions could be easily expressed formally in the XHTML DTD. However, they were not formalized by any rule.

The best of having checkpoints guaranteed by a grammar is that they are **very cheap and easy to detect**: just choosing a modern grammar to validate documents against to, and using an existing XML-ized validator like [8], may easily spot where barriers belonging to this category appear within a document.

IV. ACCESSIBILITY CONDITIONS NOT EXPRESSIBLE IN A XHTML GRAMMAR

There are some accessibility conditions that are not expressible indeed in a DTD or XML Schema. Many tools have their own internal implementation of these conditions hard-coded within lines of a program coded in an **procedural** programming language like Java, C or PHP. This might lead to complex algorithms which (usually) implement something different from the desired checkpoint. Approaches like [12] have tried to represent such rules in a set of **declarative expressions** declared in self-developed XML files that represent rules which can only be interpreted by a self-developed ad-hoc program. Using already existing software was the aim of the XQuery-based WCAG formalization approach [16]. However, though the XQuery language [6] is quite compact, it is still a draft having little implementation support. A more accurate and also cheaper implementation of checkpoint evaluation can be achieved when using other already known constraint languages for XML, like XSLT. This approach implies **cheaper** evaluation because no specific software needs to be developed indeed, and there are specific programs already supporting this language. As a result, we have a XSLT-based declarative ruleset which is **smaller**, **more reusable** and easier to understand and homogeneize than the equivalent procedural routines.

As a starting example, we will begin with attributes that enhance accessibility, but whose presence is required only under certain circumstances. For instance, table VII states that every image input, (i.e., input form fields whose type attribute is *image*) requires the `alt` attribute (as any other image). It would not be nice to declare the `alt` attribute as mandatory for every `input` element, because input form fields having a `type` attribute different of *image* don't really expect such `alt` attribute. Conditions that indicate whether some markup is mandatory or not, are not expressible in DTDs or XML Schemas.

In fact, conditions that involve some comparison or calculation over the document's content, are not expressible in DTDs or XML Schemas. Other accessibility-related attributes are also required in other elements under other conditions as well. Table VIII contains such required attributes, their container elements and their conditions to be mandatory. They are summarized, but they can be used just like the expressions in table VII.

Not only the presence or absence of markup determines accessibility. The conditions themselves may become an evaluation result. For instance, Web acces-

[4]A similar approach (DTD or Schema redefinition) could be taken for the usage of the `noframes` element, because the public DTD does not properly require such element even though it is required to be present as an alternative for frames for browsers that can't support frames.

	Expression
DTD	<!ATTLIST frame title CDATA **#REQUIRED**> *However, it is defined as* **#IMPLIED** *(not required)*
XML Schema	<xs:element name="frame"> <xs:complexType> <xs:attribute name="title" use="**required**" type="xs:string"/> </xs:complexType> </xs:element> *However, it is defined as* **optional** *(not required)*
XSLT	<xsl:template match="//frame[not(@title)]"> <xsl:message>Frames without description.</xsl:message> </xsl:template>

TABLE V

FRAMES WITHOUT DESCRIPTION. THE `frame`'S `title` ATTRIBUTE IS MANDATORY.

Container element	**Forbidden contents**	**Condition to be forbidden**
pre	img, object, big, small, sub, sup	Always forbidden
a, label	a, area, label, input, select, textarea, button	Always forbidden
form	form	Always forbidden
table	table	In XHTML Basic [4]
ins	Any *block-type* element	If the `ins` element is inside an *inline-type* element

TABLE VI

RESTRICTIONS IN XHTML SPECIFICATION (PROSE) NOT EXPRESSED IN THE DTD

	Expression
DTD	*Not feasible*
XML Schema	*Not feasible*
XSLT	<xsl:template match="//input[@type='image'][not(@alt)]"> <xsl:message>Image input with no alt</xsl:message> </xsl:template>

TABLE VII

IMAGE INPUTS WITHOUT ALTERNATIVE TEXT. THE `alt` ATTRIBUTE IS MANDATORY.

Container element	**Attribute**	**Condition to be mandatory**
img,area	alt	Always
frame	title	Always
input	alt	If input's type is "image"
img	longdesc	If image's `alt` attribute is not enough
html	lang	If document's `xml:lang` is missing
frame	longdesc	If frame's `title` attribute is not enough
th	abbr	If table header's text is too long
input	value	If form field collects editable text
table	summary	If table contains tabular data, i.e, it is not a *layout-only* table
abbr,acronym	title	If definition not previously defined

TABLE VIII

ATTRIBUTES CONDITIONALLY REQUIRED.

sibility checkpoints state that both keyboard and mouse should behave similarly in order to have some device independence properties. This involves that any JavaScript routine triggered by a mouse event should also be triggered as well by the corresponding keyboard event, and vice-versa. Table IX contains an XSLT template checking whether the `onmouseup` event is being used for other purpose than the `onkeyup` event. A more complete rule should also check the same behaviour for other events.[5]

Not only conditionally mandatory attributes can be required, but also conditionally mandatory elements as well. For instance, multimedia elements like plugins, applets or videos should have alternative contents for those users that cannot interact with them properly. These alternative contents must be placed directly inside the `object` element itself (the `object` tag acts as a container of its alternative), as stated in table X. Take notice that `param` elements are not considered as alternative contents.

Thus, any `object` element should have some other alternative element inside if it does not have a readable alternative text. Other conditionally required elements are represented in table XI. For instance, some accessibility checkpoints state that `fieldset` elements should be present in order to make manageable groups of form fields. When the number of options in a combo box (`select` element) is high, it is a good idea to group them with the `optgroup` element. The expressions from table XI are summarized, but they can be used just like the expressions in table X.

Some other important accessibility rules don't really involve existence or absence, but misusage of HTML markup. These rules state that some HTML attributes should not be improperly used. For example, it is a very bad practice to trigger JavaScript routines from the `href` attribute of active elements like links or client-side map areas. It is much better to use event-focused attributes like `onclick` for that purpose, and leave the `href` attribute usable for users that have browsers with no JavaScript support. The XSLT template from table XII detects this bad practice.

Other bad practices involving typically misused attributes are collected in table XIII. As we can see, it is bad practice to use client side auto-refresh or auto-redirect, unadvised emerging windows, improper keyboard short-cuts or unclear tabulation orders.

However, the expressive power of XSLT becomes clearer when we leave out these previous simple conditions involving the relationship between elements and their contents and we start looking at **relationships** among elements placed **anywhere** in the document. For instance, headers (`h1-h6` elements) in a document should be properly used. This involves not skipping header labels. As stated at [1], "h2 elements should follow h1 elements, h3 elements should follow h2 elements, etc. Content developers should not skip levels (e.g., h1 directly to h3)". Table XIV contains a XSLT template that spots those headers that skip levels. The `preceding` XPath axis is used here to determine if the current header is used accordingly to the immediately preceding header in the document. These constraints between one element and its preceding element cannot be expressed with a DTD or XML Schema, but they can fit easily in a XSLT template.[6]

Relationships among non-similar elements can be addressed with constraint languages as well. For example, each client-side map area should have a redundant link (somewhere in the document) for those users that cannot use client-side maps. This implies that, for every `area` element, at least one link in the same document should share the same destination (i.e. point to the same `href`). In other words, some link in the document should replicate the `href` attribute for each area. As inferred from table XV, we have to distinguish between the `area` element itself and the link before comparing their `href` attributes. In XSLT, we can use the <xsl:variable> element to reference the area element when the context node has been changed. The XSLT expression from table XV states that any `area` whose `href` attribute is not being replicated by a link somewhere else in the document will be spotted as an accessibility barrier.

Expressions involving more than one variable or following internal references can also be expressed with XSLT. This is the case for the relationship between form fields and their `label` elements. According to accessibility good practices, all non-hidden form fields should have a description stating the purpose of the form field (or what kind of data is expected to be entered).

[5] Also for onmousedown, onkeydown, onkeypress, onmouseover, onmouseout, onmousemove, ondblclick, ...

[6] For reducing verbosity, they are presented all together in a singular XSLT template. They might have been written in a template for each header, but we wanted to reduce verbosity and maintain singular templates only.

	Expression
DTD	*Not feasible*
XML Schema	*Not feasible*
XSLT	<xsl:template match="//*[@onmouseup or @onkeyup] [not(@onmouseup=@onkeyup)]"> <xsl:message>Device dependence. Both keyboard and mouse should behave similarly.</xsl:message> </xsl:template>

TABLE IX

DEVICE DEPENDENCE. BOTH KEYBOARD AND MOUSE SHOULD BEHAVE SIMILARLY.

	Expression
DTD	*Not feasible*
XML Schema	*Not feasible*
XSLT	<xsl:template match="//object[normalize-space(.)="] [not(*[name()!='param'])]"> <xsl:message>Multimedia objects without alternative. The alternative inside is mandatory.</xsl:message> </xsl:template>

TABLE X

MULTIMEDIA OBJECTS WITHOUT ALTERNATIVE. THE ALTERNATIVE INSIDE IS MANDATORY.

Container element	**Required element**	**Condition to be mandatory**
head	title	Always
html	head	Always
frameset, iframe	noframes	Always
object	*some alternative inside*	If object does not contain alternative text
form	fieldset	If too many form fields
select	optgroup	If too many options
table	caption	If table contains tabular data, i.e, it is not a *layout-only* table

TABLE XI

ELEMENTS CONDITIONALLY REQUIRED

	Expression
DTD	*Not feasible*
XML Schema	*Not feasible*
XSLT	<xsl:template match="(//a|//area)[starts-with(@href,'javascript:') and not(@onclick)]"> <xsl:message>JavaScript code should appear in event attributes.</xsl:message> </xsl:template>

TABLE XII

NOT ONLY JAVASCRIPT-BASED NAVIGATION. JAVASCRIPT CODE SHOULD APPEAR IN EVENT ATTRIBUTES.

Container element	**Misused attribute**	**Misusage condition**
a, area	href	JavaScript present at `href` attribute, instead of at `onclick` attribute
meta	http-equiv	Badly used for auto-refresh or auto-redirect
input	alt	If input's type is not "image"
* (*any tag*)	alt	`Alt` attribute equals `src` attribute
* (*any tag*)	target	Badly used for emerging windows
* (*any tag*)	tabindex	Not a set of unique consecutive numbers
* (*any tag*)	accesskey	Not a unique character

TABLE XIII

ATTRIBUTES TYPICALLY MISUSED

	Expression
DTD	*Not feasible*
XML Schema	*Not feasible*
XSLT	<xsl:template match="//h6[not(**preceding::***[self::h1 or self::h2 or self::h3 or self::h4 or self::h5 or self::h6][1][self::h6 or self::h5])] | //h5[not(**preceding::***[self::h1 or self::h2 or self::h3 or self::h4 or self::h5 or self::h6][1][self::h6 or self::h5 or self::h4])] | //h4[not(**preceding::***[self::h1 or self::h2 or self::h3 or self::h4 or self::h5 or self::h6][1][self::h6 or self::h5 or self::h4 or self::h3])] | //h3[not(**preceding::***[self::h1 or self::h2 or self::h3 or self::h4 or self::h5 or self::h6][1][self::h6 or self::h5 or self::h4 or self::h3 or self::h2])] | //h2[not(**preceding::***[self::h1 or self::h2 or self::h3 or self::h4 or self::h5 or self::h6][1])]"> <xsl:message>Improper headers</xsl:message> </xsl:template>

TABLE XIV

PROPERLY USED HEADERS

	Expression
DTD	*Not feasible*
XML Schema	*Not feasible*
XSLT	<xsl:template match="//area"> <**xsl:variable** name="area" select="." /> <xsl:if test="not(//a[@href = $area/@href])"> <xsl:message>Missing redundant area's link</xsl:message> </xsl:if> </xsl:template>

TABLE XV

A REDUNDANT LINK IS REQUIRED FOR EACH CLIENT-SIDE MAP AREA.

This description should either be in its `title` attribute or in a `label` tag referring to it. The XSLT template from table XVI establishes that for any form field that has no `title` attribute, a `label` referring to the form field must be present.

Blind people usually find a lot of barriers within a document. They usually read Web pages with screen readers, so they usually navigate through the list of links in a document in order to navigate faster. Thus, it is important that the link's contents can be easily understood when read out of context. Links saying the same words (like "click here" or "read more") and pointing to different targets make navigation difficult. We also must consider that blind users are non-case sensitive when they use screen readers. For that reason, it is a bad practice if two links pointing to different targets share the same pronounceable text. The pronounceable text of a link is a mixture of its text contents, their `title` attribute (which provides extra information) and the `alt` attributes of all the images inside the link. Although other features like punctuation signs have not been considered in the XSLT expression from table XVII, we can use them to detect similar links pointing to different targets.

V. CONCLUSIONS AND FUTURE WORK

We needed to express rules or templates easy to understand that could be applied to any Web page easily and transparently, i.e., that users could easily understand how to comply with them. We needed that such rules could be reusable and easy to modify. We found that XSLT [7] could be used to express such templates. By applying a self-developed XSLT file, we could generate accessibility reports directly from any Web page.

Soon, we became aware of the expressing power of this approach. We became aware that we could easily express conditions from already built Web Accessibility evaluators, no matter if they were simple or complex in a fine and clean way. We also realized that we could express conditions not previously being checked. For instance, we included conditions from the XHTML specification not previously formalized in the corresponding DTD or Schema.

We have dealt with subjective conditions using the approach of having some customizable parameters in the XSLT file. This way, people might customize some values of the XSLT according to their own needs.

We would have liked to find the required expressing power in DTDs or XML Schema. If so, we would have

	Expression
DTD	*Not feasible*
XML Schema	*Not feasible*
XSLT	<xsl:template match="//select | //textarea | input[@type='password' or @type='text' or @type='checkbox' or @type='radio' or @type='file']"> <**xsl:variable** name="ff" select="." /> <**xsl:variable** name="l" select="//label[@for=$ff/@id]" /> <xsl:if test="count($l)>1 or (count($l)=0 and not(@title))"> <xsl:message>Missing form field's label</xsl:message> </xsl:if> </xsl:template>

TABLE XVI

A LABEL IS REQUIRED FOR EACH FORM FIELD

	Expression
DTD	*Not feasible*
XML Schema	*Not feasible*
XSLT	<xsl:template match="//a|//area"> <xsl:variable name="a" select="." /> <xsl:for-each select="$a/**following::a**\|$a/**following::area**"> <xsl:if test="@href != $a/@href and **translate**(normalize-space(**concat(@title, .//@alt, .)**), 'abcdefghijklmnopqrstuvwxyzáéíóúñ', 'ABCDEFGHIJKLMNOPQRSTUVWXYZÁÉÍÓÚÑ') = **translate**(normalize-space(concat($a/@title, $a//@alt, $a)), 'abcdefghijklmnopqrstuvwxyzáéíóúñ', 'ABCDEFGHIJKLMNOPQRSTUVWXYZÁÉÍÓÚÑ')"> <xsl:message> Different links sharing the same pronounceable text <!– $a and . are such links –> </xsl:message> </xsl:if> </xsl:for-each> </xsl:template>

TABLE XVII

AVOID SIMILAR LINKS POINTING TO DIFFERENT TARGETS.

written our conditions in such a grammar (and WAEX would probably be an enhanced DTD or Schema). However, we found that DTDs and XML Schemas don't have the required expressivity, so that's why we used XSLT.

We are planning to adapt WAEX to generate EARL [5] reports as well.

VI. ACKNOWLEDGEMENTS

We gratefully acknowledge support from the MOSAIC *TSI-2005-08225-C07* and INFOFLEX *TIC2003-07208* projects of the "Ministerio de Educación y Ciencia" (Spanish ministry). The second author also gratefully acknowledges support from his University and the "Secretaría de Estado de Universidades e Investigación del Ministerio de Educación y Ciencia" of Spain for his sabbatical stay at MIT during the academic year 2005/06, and to the MIT for hosting him during this year in an inspiring environment.

REFERENCES

[1] W3C *Web Content Accessibility Guidelines 1.0 (Recommendation, May 1999)*
www.w3.org/TR/WCAG10

[2] W3C *XHTML 1.0 ^{TM} 1.0 - The Extensible HyperText Markup Language (Second Edition), A Reformulation of HTML 4 in XML 1.0, W3C Recommendation 26 January 2000, revised 1 August 2002*
www.w3.org/TR/xhtml1

[3] W3C *XHTML 1.1 ^{TM} 1.1 - Module-based XHTML W3C Recommendation 31 May 2001*
www.w3.org/TR/xhtml11

[4] W3C *XHTML Basic W3C Recommendation 19 December 2000*
www.w3.org/TR/xhtml-basic

[5] W3C *Evaluation and Report Language (EARL) 1.0 Guide - Editors' Draft 14 February 2006*
www.w3.org/TR/EARL10

[6] W3C *XQuery 1.0: An XML Query Language W3C Working Draft 29 October 2004*
www.w3.org/TR/xquery

[7] W3C *XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999*
www.w3.org/TR/xslt

[8] W3C *Markup Validation Service*
validator.w3.org

[9] Watchfire *WebXACT Accessibility tool*
webxact.watchfire.com

[10] CEAPAT, Fundación CTIC, Spanish Ministry of Employment and Social Affairs (IMSERSO) *Online Web accessibility test*
www.tawdis.net

[11] Fundación SIDAR *Accessibility testing with Style*
www.sidar.org/hera

[12] Jean Vanderdonckt, Abdo Beirekdar, Monique Noirhomme-Fraiture *Automated Evaluation of Web Usability and Accessibility by Guideline Review*
4th Web Engineering International Conference (ICWE 2004), Munich

[13] Sourceforge *HTML parser and pretty printer in Java*
jtidy.sourceforge.net

[14] Daniel Veillard *The XML C parser and toolkit of Gnome*
www.xmlsoft.org

[15] Vicente Luque Centeno *WAEX: Web Accessibility Evaluator in a single XSLT file*
www.it.uc3m.es/vlc/waex.html

[16] Vicente Luque Centeno, Carlos Delgado Kloos, Martin Gaedke, Martin Nussbaumer *WCAG Formalization with W3C Standards*
The 14th International World Wide Web Conference (WWW2005), ISBN 1-59593-051-5, pags. 1146-1147 Chiba, Japan, May 11-14, 2005

[17] Vicente Luque Centeno, Carlos Delgado Kloos, Jesús Arias Fisteus, Norberto Fernández García *Estudio comparativo de herramientas de evaluación de la accesibilidad web*
V Jornadas de Ingeniería Telemática, JITEL 2005, ISBN 84-8408-346-2 Vigo, 12-14 de Septiembre de 2005