WCAG Formalization with W3C Standards

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

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

ABSTRACT

Web accessibility consists on a set of checkpoints which are rather expensive to evaluate or to spot. However, using W3C technologies, this cost can be clearly minimized. This article presents a W3C formalized rule-set version for automatable checkpoints from WCAG 1.0.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia; F.4 [**Mathematical Logic and Formal Languages**]: Miscellaneous

General Terms

Algorithms, Standardization, Verification

Keywords

WAI; WCAG; XPath; XPointer; XQuery

1. Introduction

WAI (Web Accessibility Initiative)'s WCAG (Web Content Accessibility Guidelines) 1.0 [1] is an important contribution to Web accessibility, focusing not only on eliminating barriers for disabled people, but also a major step towards device independence, allowing Web interoperability to be independent from devices, browsers or operating systems. WCAG 1.0 have become an important reference for Web accessibility in the Web community. However the set of the 65 WCAG's checkpoints that accessible documents have to pass is a very heterogeneous set of conditions which are difficult to evaluate. WCAG 1.0 specification is written in a high abstraction level which is frequently quite far away from the low level technical detail of the HTML format. Many of those checkpoints are also open to subjective interpretation, including implicit conditions or, simply, containing conditions whose detection can not be automated.

2. WCAG formalization

Rules from WCAG can be classified into the following groups:

1. **Objectively automatable** rules are clearly defined and specify a condition that nobody might reject.
2. **Subjectively automatable** rules specify fuzzy conditions that can be automated, but whose particular *non-fuzzy* interpretation might be accepted or rejected by different groups of people. For these kind of subjective automatable conditions, W3C has defined a set of heuristics [2] that might help to evaluate.
3. **Semi-automated** rules, which can not be evaluated automatically, but tool's assistance can focus user's interest on relevant markup.
4. **Manual** rules, which require human judgement. Both semi-automated and manual rules are very expensive to evaluate and should be kept to a minimum.

Both semi-automated and manual rules can not be evaluated automatically by a program with an acceptable degree of trust. However, both objectively or subjectively automatable rules may be formalized as follows.

2.1. Checkpoints guaranteed by a grammar

XHTML is not a unique language. Since its birth, it has had several versions, starting from Transitional and Strict, launching XHTML Basic [5] and ending by XHTML 1.1 or even XHTML 2.0 (a draft). Those different languages have different accessibility restrictions, some of them had been previously declared in the 1999's WCAG. For example, images' `alt` attribute are mandatory since XHTML Transitional 1.0. Deprecated elements like `font` or `center` were removed in XHTML Strict 1.0. Besides that, rules 3.2 (Create documents that validate to published formal grammars), 3.3 (Use style sheets to control layout and presentation) and 11.2 (Avoid deprecated features of W3C technologies) from WCAG directly depend on XHTML validation.

2.2. Checkpoints declared in an XPath rule

Table 1 shows several examples of WCAG checkpoints that can be formalized as XPath 1.0 [3] rules. Some of these rules cover the following conditions.

- Textual alternatives are required for multimedia.
- Frames should have both `title` description as well as a `noframes` alternative.
- Idiom changes should be explicit.
- Device dependant events should be used in equivalent events by pairs.

WWW 2005, May 10–14, 2005, Chiba, Japan.
ACM 1-59593-051-5/05/0005.

WCAG #	XPath 1.0 rule
1.1b	//input[@type="image"][not(@alt)]
1.1c	//img[*toolong*(@alt)][not(@longdesc)]
1.1d	//object[not(*)][normalize-space(text())=""]
1.1e	//frameset[not(noframes)]
3.5a	//h2[not(preceding::h1)]
4.3	//html[not(@xml:lang)]
5.6	//th[*toolong*(text())][not(@abbr)]
6.4a	//*[@onmouseover != @onfocus]
6.4b	//*[@onmouseout != @onblur]
7.4, 7.5	//meta[@http-equiv="refresh"]
9.2a	//*[@onmousedown != @onkeydown]
9.2b	//*[@onmouseup != @onkeyup]
9.2c	//*[@onclick != @onkeypress]
10.1a	//*[@target="_blank" or @target="_new"]
10.4a	//input[@type!="hidden"][not(@value)]
10.4b	//textarea[normalize-space(text())=""]
12.1	//frame[not(@title)]
12.2	//frame[*toolong*(@title)][not(@longdesc)]
12.3c	//p[*toolong*(text())]

Table 1: XPath 1.0 rules

2.3. Checkpoints declared in a XQuery 1.0 rule

Elements breaking some complex conditions involving several elements can be addressed by XQuery 1.0 [4] expressions. For example, rule 1.5 requires alternative redundant links for every link within a client-side image map. This means that links defined within the map should also be defined somewhere else in the document, as formalized in the XQuery expression of figure 1.

```
//area[let $area:=self::area return
count(//a[@href = $area/@href]) = 0]
```

Fig. 1: XQuery 1.0 expression for client side maps breaking WCAG 1.5

Figure 2 contains a XQuery 1.0 expressions for addressing `h3` elements breaking WCAG 3.5 (Use headings properly). Similar rules can be written for `h4`, `h5` and `h6` elements.

```
//h3[let $h3:=self::h3 return
let $h2:=$h3/preceding::h2[last()] return
let $h1:=$h3/preceding::h1[last()] return
$h1=() or $h2=() or $h1>>$h2]
```

Fig. 2: XQuery 1.0 expression for h3 elements breaking WCAG 3.5b

No abbreviation or acronym should be defined more than once. Figure 3 shows an XQuery expression which detects multiple-defined acronyms and abbreviations.

```
(//abbr | //acronym)[let $a:=self::node() return
count((//abbr | //acronym)[text() = $a/text()]) != 1]
```

Fig. 3: XQuery expression for abbr and acronym elements breaking WCAG 4.2a

Tab order, if explicitly specified, should be consistently. Figure 4 shows an XQuery expression which address elements with an improper tab order.

```
//*[@tabindex][let $n:=self::node()/@tabindex return
not(isnumber($n)) or count(//*[@tabindex=$n]) != 1 or
number($n)<1 or number($n)>count(//*[@tabindex])]
```

Fig. 4: XQuery expression for elements breaking WCAG 9.4

Keyboard shortcuts should be used properly. This means that every `accesskey` attribute should have a unique character within a document. Figure 5 addresses elements with improperly reused `accesskey` attributes.

```
//*[@accesskey][let $c:=self::node()/@accesskey return
not(ischar($c)) or count(//*[@accesskey=$c]) != 1]
```

Fig. 5: XQuery expression for elements breaking WCAG 9.5

Rule 12.4 requires that every visible form field should have a `label` whose `for` attribute matches the form field's `id` attribute. This does not apply to hidden form fields or submit buttons, as expressed in figure 6.

```
(//select|//textarea|//input[@type="text" or
@type="password" or @type="radio"
or @type="checkbox"])[let $ff:=self::node() return
count(//label[@for=$ff/@id]) != 1]
```

Fig. 6: XQuery expression for form fields breaking WCAG 12.4

Rule 13.1 requires that every link's text should be meaningful when read out alone. The expression within figure 7 can be used to address those links sharing the same text (and title), but pointing to different URLs (a practice which some readers consider confusing).

```
(//a | //area)[let $a:=self::node() return
(//a | //area)[@title = $a/@title and
text() = $a/text() and @href != $a/@href] != ()]
```

Fig. 7: XQuery expression for links breaking WCAG 13.1

3. Acknowledgements

The work reported in this paper has been partially funded by the projects INFOFLEX *TIC2003-07208* and SIEMPRE *TIC2002-03635* of the Spanish Ministry of Science and Research.

1. REFERENCES

[1] W3C *Web Content Accessibility Guidelines 1.0* `www.w3.org/TR/WCAG10`

[2] W3C *Techniques For Accessibility Evaluation And Repair Tools W3C Working Draft, 26 April 2000* `www.w3.org/TR/AERT`

[3] W3C *XML Path Language (XPath) Version 1.0 W3C Recommendation 16 November 1999* `www.w3.org/TR/xpath`

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

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