Journal of Web Engineering, Vol. 5, No. 4 (2006) 313–331
© Rinton Press

WEB COMPOSITION WITH ACCESSIBILITY IN MIND

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

Received July 25, 2005
Revised April 10, 2006

Web accessibility should be a part of the Web design process instead of being a *post-design* repair process. Thus, it should be more integrated within the internal authoring tools' mechanism of generating new accessible Web contents. Web pages are usually composed of small pieces of HTML code which, dynamically nested and combined, generate full Web pages. This Web composition, specially when creating Web pages from data extracted from heterogeneous or external sources, should have accessibility into account in order to guarantee that the final page being constructed is accessible. This paper presents the set of rules that, in a Web composition process, a design tool must follow in order to guarantee that the Web pages being generated are accessible. These rules are formalized with W3C standards like XPath and XQuery expressions (so they are vendor-neutral). We also present WSLS as an accessibility enabled authoring tool that makes this task feasible, and focus on how this tool incorporates accessibility into the process of generating new Web contents.

Keywords: Web accessibility; WCAG; Web composition; formalized conditions

1 Introduction

Web browsers have evolved since their inception at the beginning of the Web. We can find plenty of them in very different and heterogeneous devices. Wireless devices like PDA, palmtops or mobile phones can be connected to Internet, as well as other new devices like digital TV's, or vehicle's assistants. However, many difficulties (namely barriers) arise when these devices are used for surfing legacy Web sites, i.e., Web sites without accessibility features in their designer's mind, perhaps specifically designed for a concrete browser running in a desktop computer.

Disabled people also experiment similar problems when using specific platforms adapted to their special needs, like Braille displays, voice browsers or screen magnifiers to Web sites having these accessibility barriers. As a result, accessibility has become a very important feature, and W3C defined WCAG (Web Content Accessibility Guidelines) [1,2] as a set of guidelines that Web pages should follow in order to be considered accessible. The ability for a Web site to be surfed by any device has become a major target for Web designers.

However, most authoring tools still lack from features for generating accessible and deviceindependent markup. At this point, accessibility is usually treated as a *post-design* repair process, most of the times guided by an accessibility evaluator tool, like Watchfire (new version of the very well known Bobby tool) [9], Tawdis [10], Torquemada [11] or HERA [12] that spots accessibility barriers that specialized persons have to supervise with no little effort.

Removing accessibility barriers for already made Web sites involves a huge amount of human effort because a lot of human supervision is required for these evaluation tools. This *post-design* repair process is much less effective than having accessibility features in mind during the design process. Since authoring tools should produce high quality markup for new Web contents, we believe that most of accessibility benefits can be achieved if authoring tools implement accessible-aware creation process mechanisms that guarantee that their generated Web content will be accessible.

2 Related work

Web composition [13] has emerged as a consistent, flexible and dynamic way of generating Web contents by combining reusable pieces of HTML code into other reusable pieces that, guided by a composition process, will be used to create whole Web pages and whole Web sites. This is the approach of several authoring tools like WSLS [15], that extracts dynamic contents from heterogeneous sources and automatically generates new updated Web contents without the need of a person to guide the process in running time. For these tools, it would be highly desirable to guarantee that final Web contents will be accessible, provided that those smaller pieces of reusable HTML code coming from external sources to be used in the Web composition process are already accessible. In other words, provided that a Web Composition based authoring tool manages small pieces of accessible HTML code (both static templates or dynamically generated) we want to guarantee that the Web composition process will always yield an accessible Web page (or site). So, we must make sure that no kind of new accessibility barriers can be introduced by the Web composition process itself. This paper is focused on formalizing the conditions to be met so that accessible chunks of Web pages can be safely compounded into a page that also results accessible from a WCAG's point of view. As we will see, non accessible results can be obtained from combining accessible pieces, unless some rules are followed. This paper provides a formalized set of such rules.

At the present moment, we have no knowledge of formalized, vendor-neutral mechanisms that can be reused on different tools for the purpose of providing accessibility during the Web creation process. Most of the tools related to accessibility, like [9,10,11,12] only provide a *post-design* analysis. Some other authoring tools prompt for specific accessibility markup to the author (i.e., they ask for the `alt` attribute whenever a new image is inserted). However, we have no knowledge on any of them that guarantees accessibility by combining different HTML snippets obtainable from heterogeneous sources or that guarantee that combining accessible HTML snippets in a bottom-up process will result accessible.

3 WCAG compliant composition

WCAG defines three accessibility levels, being A, AA and AAA. In order to declare some common names in our rules, we establish the following definitions.

1. Given \$S1 and \$S2 compoundable pieces of HTML markup.[a]

2. Given *wailevel*(\$S) = {No | A | AA | AAA} (an ordered set, so that No < A < AA < AAA).[b]

3. Provided that snippets \$S1 and \$S2 are WCAG compliant, i.e., *wailevel*(\$S1) ∈ {A | AA | AAA} ∧ *wailevel*(\$S2) ∈ {A | AA | AAA}.[c]

4. Provided that snippet \$S2 will be nested inside snippet \$S1 and that *pos*(\$S1,\$position) is the insertion **point** for \$S2 inside \$S1, as depicted in figure 1.[d]

5. Assuming bottom-up document construction, i.e., \$S2 has no position where other HTML reusable pieces might be inserted.[e]

6. Provided that *comp*(\$S1,\$S2,\$position) is the composition of \$S2 inserted into *pos*(\$S1,\$position), i.e., \$S2 nested somewhere in \$S1.

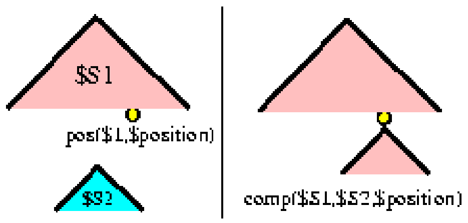


Fig. 1. Web Composition of HTML snippets

The purpose of this section is to declare which conditions should be met so that *wailevel*(*comp*(\$S1,\$S2,\$position)) ≥ MINIMUM(*wailevel*(\$S1), *wailevel*(\$S2)).[f]

These conditions should be applyable to any HTML snippets without other assumptions than the ones already mentioned. Particularly, these HTML snippets might be dynamically generated from heterogeneous sources. These conditions should preferably be evaluated automatically to any HTML snippet, without human intervention, in order to allow the Web composition process to continue.

[a]Also called HTML snippets.
[b]The accessibility level defined by WCAG from W3C for HTML snippet \$S.
[c]This is required to apply induction. If all basic snippets are accessible and the composition process guarantees that, provided accessible snippets, the result is also accessible, then all composed contents will always be accessible (by induction).
[d]This insertion point can also be considered, for simplicity, as a virtual *empty* element within \$S1 that will be substituted by \$S2. This latter approach is applyable for Web composition based on technologies like, for example, JSP or ASP, because these formats have special tags whose purpose is to be replaced by some other contents when generating new Web pages.
[e]In other words, \$S2 has no free insertion point, while \$S1, at least has an insertion point for \$S2.
[f]We are not really interested on upgrading our pieces' accessibility level (though this could happen if both \$S1 nd \$S2 provide complimentary accessibility markup). Our focus resides on not degrading our pieces' accessibility level because of an unfortunate composition.

4 Process-aware checkpoints

This section covers all WCAG checkpoints that should be reviewed before allowing Web composition to produce results. This is because, even provided that $S1 and $S2 comply with them, *comp*($S1,$S2,$position) might not comply if some restrictions are not followed. Thus, this section covers all *these extra* conditions to be met in order to guarantee accessibility, provided $S1 and $S2 are accessible. If these conditions are not met, accessibility barriers could be incidentally introduced.

The way these conditions will be guaranteed by authoring tools can be rather different from one tool to another and is an issue of these tools. Some tools might *stop* the composition process with an error or exception if these required process-aware checkpoints are detected as not followed. Some tools might present a form so that the author indicates some extra required information or, maybe, allow the operation under her own responsibility. Some tools might automatically and transparently add new markup or contents in order to eliminate a possible barrier, ... Behaviour are not treated in this section. Only the conditions are treated here.

Rule 1.4: Global synchronization of timed events

Provided that both $S1 and $S2 have all their timed events (if any) internally synchronized (intra-synchronization), *comp*($S1,$S2,$position) might also need some synchronization between $S1's events with $S2's events (inter-synchronization). The way to achieve this depends on the authoring tool. If the authoring tool does really manage event synchronization (which is rarely implemented, because it combines multimedia and scripting), then this inter-synchronization can be automatically guaranteed. Otherwise, it should be guaranteed by an external agent, perhaps a person under her own responsibility.

Rule 2.2: Color combinations

This is important for people with color blindness (or using monochrome displays), though normal users might also experiment problems if bad color combinations are chosen. Yellow over white, for example, can be painfully read by most users, because they have little contrast with each other. Provided that both snippets $S1 and $S2 have all their colors properly combined (i.e. there exists a high contrast between foreground and background colors in all elements within both $S1 and also $S2), this could, however be broken if a bad color combination between a background (or foreground) color from $S1 and a foreground (or background) color from $S2 would be applied to any element of the composition of $S1 and $S2. This is only possible for elements within $S2 (the internal snippet) that have provided a foreground color but no background color (or vice-versa). In that case, the non-specified color will be **inherited** from $S1 (the external snippet), which will be combined with the (already) specified counterpart color within $S2. So we must make sure that such a color combination, if it exists, will look good having an acceptable color contrast. Figure 2 shows an example of how two accessible snippets might produce different results.

Figure 3 expresses the condition that all $S2's elements having a foreground color but not a background color (or vice-versa) will have a good color combination with the corresponding color inherited from $S1. This formalized condition is only concerned on color combination derived from HTML color specification, i.e., in the contents of the `color` and `bgcolor` HTML

```
$S2 = ... <font color="red"> ... </font> ...
$S1 = ... <font color="yellow" bgcolor="red"> ... $S2 ... </font> ...
result = BAD for colour distinction (red over red), which is a problem for visual users.

$S2 = ... <font color="red" bgcolor="white"> ... </font> ...
$S1 = ... <font color="yellow" bgcolor="red"> ... $S2 ... </font> ...
result = GOOD
```

Fig. 2. Example of accessible snippets that produce different results based on colour combinations

attributes.[g]

```
let $bgcolor1 := pos($S1,$position)//ancestor-or-self::*[@bgcolor][1]/@bgcolor return
let $fgcolor1 := pos($S1,$position)//ancestor-or-self::*[@color][1]/@color return
let $elemsf := $S2//font[@color and not(ancestor-or-self::*[@bgcolor])] return
let $elemsb := $S2//font[@bgcolor and not(ancestor-or-self::*[@color])] return
let $contrastbg1fg2 := every $elemf in $elemsf satisfies highcontrast($bgcolor1, $elemf/@color)
return
let $contrastbg2fg1 := every $elemb in $elemsb satisfies highcontrast($fgcolor1, $el-
emb/@bcolor) return
$contrastbg1fg2 and $contrastbg2fg1
```

Fig. 3. XQuery pre-condition for disabling undesirable HTML-based color combinations for text

Rule 3.2: Grammar validation

Most of the rules that should be followed in order to build accessible Web pages from accessible already built HTML snippets rely on grammar validation. These rules should be followed in order to build valid documents that validate against a well known language, i.e., a HTML or XHTML [6] grammar. Every one of the following rules is described in a DTD or XML Schema for XHTML and must be followed in the Web Composition process in order to get accessible results:

1. $S2 should **fit** *grammatically* into *pos*($S1,$position) according to the DTD or XML Schema being used for the final document being built. This implies that $S2 should be allowed to be placed as a direct descendant part within the $S1's node just above *pos*($S1,$position). If $S2 has a single root element, this root element should be an allowed child within $S1's node just above *pos*($S1,$position). For example, table rows can only be inserted within a table (nowhere else), list items can only be inserted within a list (prohibited elsewhere), ... As a general rule, HTML tags are divided into *block-type* and *inline-type*, thus requiring that any block content is inserted only inside elements which are allowed to have block-type children (inline elements cant't have block children). Figure 4 shows an example of bad and good element composition according to this grammar. Generic tags like `div` or `span` are recommended to be used as a generic type of *block-type* or *inline-type* pieces respectively.

[g]CSS-based color combination is outside the scope of a rule written in XPath [3] or XQuery [4], and it is also a difficult-to-deal-with issue, since many color combination failures will reside on user's and author's external styles or default browser's configurations, not in Web documents themselves. As a result, CSS-based good color combination is outside the scope of Web composition, and must be guaranteed by a CSS-aware

$S2 = <p> ... </p>
$S1 = ... <strong> ... $S2 ... </strong> ...
result = BAD (block element "p" inside inline element "strong", which is forbidden by grammar)

$S2 = <strong> ... </strong>
$S1 = ... <p> ... $S2 ... </p> ...
result = GOOD (inline element "strong" inside block element "p", which is OK)

$S2 = <li> ... </li>
$S1 = ... <div> ... $S2 ... </div> ...
result = BAD (list item elements "li" not inside a list element, which is forbidden by grammar)

$S2 = <li> ... </li>
$S1 = ... <ul> ... $S2 ... </ul> ...
result = GOOD (list items inside a list element, which is OK)

Fig. 4. Example for bad and good element composition according to the grammar

2. No focusable elements should be nested. Focusable elements are those that can receive user's focus interaction (via mouse or keyboard or any other input device), like links or form fields. If such focusable elements nest, there would be ambiguity when jumping from focusable elements to other focusable elements. There should also be ambiguity about choosing which behaviour should be executed when the users provides an interactive event to a pair of nested focusable elements. Should both behaviours be executed? Should the internal behaviour replace the external one, or vice-versa? The user will probably have no chance to choose because browsers do not expect such element combinations. Provided that neither $S1 nor $S2 have nested focusable elements, the problem arises if their composition has a forbidden combination of nested focusable elements derived from the composition process (i.e., a $S2's focusable element within a $S1's focusable element). In order to guarantee that Web Composition will not yield such forbidden nested combination, we must make sure that $S2 has no focusable element or (if it has), that *pos*($S1,$position) is not within the domain of a focusable element, like a link or a label. Figure 5 shows an example of (undesirable) nested focusable elements and one possible solution to avoid this.

   Figure 6 expresses the condition for avoiding nested focusable elements.

3. Forms should also be unnested. Otherwise, there would be ambiguity about how form fields could be mapped into forms. Figure 7 shows how two accessible HTML snippets, each having a form, might result in a bad result if forms are nested.

   Figure 8 expresses the XPath condition that guarantees that *comp*($S1,$S2,$position) will not have nested forms, provided that no nested form can be found within $S1 nor $S2 snippets.

combination process.

$S2 = ... <a> ... </a> ...
$S1 = ... <a> ... $S2 ... </a> ...
result = BAD (focusable elements nested). Nested links are forbidden.

$S2 = ... <label> ... </label> ...
$S1 = ... <a> ... $S2 ... </a> ...
result = BAD (focusable elements nested). Links and labels should not nest. Trying to activate the link would activate the form field associated with the label, or vice-versa.

$S2 = ... <label> ... </label> ...
$S1 = ... <a> ... </a> ... $S2 ...
result = GOOD (no focusable elements nested). Every active area in the document should have a well defined meaning.

Fig. 5. Example for avoiding nested focusable elements

$S2//(a | area | label | input | select | textarea | button) = () or
pos($S1,$position)//ancestor::(a | area | label | input | select | textarea | button) = ()

Fig. 6. XPath pre-condition for disabling undesirable nesting of focusable elements in Web composition. No focusable element from the internal snippet might result inside a focusable element from the external snippet.

$S2 = ... <form> ... </form> ...
$S1 = ... <form> ... $S2 ... </form> ...
result = BAD (nested forms, which is not allowed)

$S2 = ... <form> ... </form> ...
$S1 = ... <form> ... </form> ... $S2 ...
result = GOOD (no forms nested)

Fig. 7. Example for avoiding nested forms

$S2//form = () or *pos*($S1,$position)//ancestor::form = ()

Fig. 8. XPath pre-condition for disabling undesirable nesting of forms in Web composition. The internal snippet must have no form inside. Otherwise the insertion point in the external snippet must not be insidea a form either.

4. If XHTML Basic [7,8] is the target format of our page, no nested tables should be allowed either. Even though other HTML or XHTML variants allow nested tables, nested tables are not recommended either, because they provide several difficulties for layout on small screens. Provided that neither $S1 nor $S2 have nested tables, this prohibition is guaranteed as long as $S2 has no table inside or, if it has, that it is not going to be inserted within a $S1's table's scope, as depicted in figure 9.

$S2 = ... <table> ... </table> ...
$S1 = ... <table> ... $S2 ... </table> ...
result = BAD. Nested tables are not forbidden, but they are not recommended because devices with small displays usually have a lot of problems when using tables for layout.

$S2 = ... <table> ... </table> ...
$S1 = ... <div> ... $S2 ... </div> ...
result = GOOD (layout with CSS, avoiding nested tables).

Fig. 9. Example for avoiding nested tables

Figure 10 expresses the XPath condition that guarantees that the composition of $S1 and $S2 will not have nested tables, provided that no nested tables can be found within $S1 nor $S2.

$S2//table = () or *pos*($S1,$position)//ancestor::table = ()

Fig. 10. XPath pre-condition for disabling undesirable nesting of tables in Web composition. The internal snippet must not habe tables inside. Otherwise the insertion point of the Web composition within the external snippet must not be inside a table.

5. No common *id* attributes, should be allowed. Because *id* attributes should be unique within a document, *id* attributes should not be commonly used in both $S1 and $S2. Thus, the intersection of all $S1's *id* values and all $S2's *id* values should be the empty set.[h] Figure 11 describes an example of the problem of building an identifier overlap from snippets that have no such a problem.

$S2 = ... <div id="id1"> ... </div> ...
$S1 = ... $S2 ... <img id="id1" /> ...
result = BAD (two elements will conflict sharing the same identifier). Identifiers must be unique.

$S2 = ... <div id="id1"> ... </div> ...
$S1 = ... $S2 ... <img id="id2" /> ...
result = GOOD (no elements will conflict sharing the same identifier).

Fig. 11. Example of snippets overlapping identifiers

[h]$S1//*[@id]/@id ∩ $S2//*[@id]/@id = ∅

Figure 12, thus requires that the intersection of identifiers coming from both snippets is the empty set, formalized in a XPath expression.

$S1//*[@id]/@id intersect $S2//*[@id]/@id = ()

Fig. 12. XPath pre-condition for disabling the appearance of two elements with a common `id` attribute (identifier) in Web composition

6. The previous rule also applies to *name* attributes whenever used as *id* (i.e. when applied to elements depicted in figure 13), something which, in fact, is a deprecated feature which is explicitly forbidden in the most recent versions of XHTML (in favour of the *id* attribute).

$S2 = ... <a name="id1"> ... </a> ...
$S1 = ... $S2 ... <a name="id1"> ... </a> ...
result = BAD (two anchors sharing the same name). Having ambiguous anchors may confuse the user (as well as the browser).

$S2 = ... <a name="id1"> ... </a> ...
$S1 = ... $S2 ... <a name="id2"> ... </a> ...
result = GOOD (no anchors sharing the same name)

Fig. 13. Example of snippets overlapping name (as identifiers)

Figure 14, thus requires that the intersection of `name` attributes (used as identifiers) coming from both snippets is the empty set, formalized in a XPath expression. In other words, there should be no elements sharing the same name in both snippets.

$S1//(a | meta | object | img | applet | map | form)/@name intersect
$S2//(a | meta | object | img | applet | map | form)/@name = ()

Fig. 14. XPath pre-condition for disabling the appearance of two elements with a common name in Web composition

7. Name attribute has other usages, though, when applied to other tags. No common *name* attributes in *param* tags should appear within the same *object*. Even though two different objects could share parameters with the same name, parameters for a single object should have different names, in order to be properly parametrized. Figure 15 illustrates that $S2's orphan params inserted within a $S1's object should not find a sibling *param* with a common name. Otherwise, two different *params* with a common *name* would be inside the same *object*. In other words, a document can have several *param* elements sharing a common name, but they should not be inside the same multimedia objects.

8. Finally, no common *name* attributes in form fields (in those that must have unique names) should appear within the same form, as expressed in figure 16. This does not

$S2//param[not (../(object|applet))]/@name intersect
pos($S1,$position)/../(object|applet)//param/@name = ()

Fig. 15. XPath pre-condition for disabling the common-named parameters for objects

apply for *radio* or *check-box* buttons, however, because they can share a common name with other similar buttons.

$S2//(textarea | select | button | input[@type != "radio" and @type != "checkbox"])[not (ancestor::form)]/@name intersect
pos($S1,$position)/ancestor-or-self::form[1]//(
textarea | select | button | input[@type != "radio" and @type != "checkbox"])/@name = ()

Fig. 16. XPath pre-condition for disabling the common-named form fields. The internal snippet must not contain any form field with the same name of any other form field within any form that contains the insertion point in the external snippet.

Rule 4.1: Language shifts

Idiom changes **within** a document should be explicit. Whenever a piece of text is written in a different idiom, proper markup should be used. Provided that both $S1 and $S2 obbey this rule, we still must guarantee that, if $S1's default language is different from $S2's default language, the `xml:lang` is used explicitly on the top of the internal snippet $S2, as depicted in figure 17.

$S2/@xml:lang or
pos($S1,$position)//ancestor-or-self::*[@xml:lang][1]/@xml:lang = *lang*($S2)

Fig. 17. XPath pre-condition for requiring xml:lang attributes (language shifts). The internal snippet $S2 must have explicitly declared an `xml:lang` attribute or it must use the same language as the external snippet $S1.

Idiom changes **between** documents should also be explicit, i.e., the `hreflang` attribute should also be properly used. Though, assuming that both $S1's and $S2's links follow this rule, in order to guarantee that *comp*($S1,$S2,$position) will also follow it, we still must make sure that all links within $S1 or $S2 (i.e., any compoundable piece of HTML page), properly use a `hreflang` if needed. This is a difficult issue for those authoring tools that have no means for specifying which idiom is being used, specially for those extracting data from heterogeneous and external information sources. In order to properly manage the `hreflang` attribute for every link, authoring tools should take into account the idiom of every target and the idiom of every link.

Rule 4.2: Proper abbreviations and acronyms

Abbreviations and acronyms should be properly used. This requires, among other things, that any single text defined with an abbreviation or acronym is specified only once at most within a document.[i] In other words, $S1 and $S2 should have no common abbreviation or

[i] ($S1//abbr/text() ∪ $S1//acronym/text()) ∩ ($S2//abbr/text() ∪ $S2//acronym/text()) = ∅

acronym's text within a document, as shown at figure 18.[j]

$S2 = ... <acronym title="United Nations">UN</acronym> ...
$S1 = ... <acronym title="Unified Notation">UN</acronym> ... $S2 ...
result = BAD. The term "UN" yields to ambiguous definitions that might confuse the reader.

Fig. 18. Example of two different acronyms sharing ambiguous definitions for the same text. Providing different definitions for the same text results in semantic barriers that might confuse the reader.

Figure 19 contains the XPath pre-condition requiring that both snippets have no common acronym or abbreviation.

($S1//abbr/text() union $S1//acronym/text()) intersect
($S2//abbr/text() union $S2//acronym/text()) = ()

Fig. 19. XPath pre-condition for avoiding undesirable repeated abbreviations or acronyms. Any text must be defined once at most, not only in both snippets, but also in the composition result.

Rule 9.4: Unique and consecutive tabulation order

Tabulation order should be consistent, if specified. This requires that no more than a single element is focusable for a given tabulation order. Figure 20 shows an example of two links, ambiguosly sharing a tabulation order.

$S2 = ... <a tabindex="1">A</a> ...
$S1 = ... <a tabindex="1">B</a> ... $S2 ...
result = BAD. Tabulation order must be based on unique positive and consecutive numbers. Otherwise the tabulation order will be ambiguous.

Fig. 20. Example showing a bad combination of elements with tabulation order

Both $S1 and $S2 snippets should then have no common `tabindex` attribute, as depicted in figure 21.[k]

Rule 9.5: Unique keyboard shortcuts

Keyboard shortcuts should also be unique. Otherwise, as expressed in figure 22 we would have some ambiguity.

Both snippets should have no common keyboard shortcut, as expressed in figure 23 [l]

Rule 10.5: Non consecutive links

Provided that all $S1's and $S2's links have no consecutive links (some printable text between links), their composition could have consecutive links without such printable characters if a $S2's link appears just in front of a $S1's link. Figure 24 shows an example of how two links might appear as consecutive during the Web composition process.

For this rule, XPath is not enough, but a combination of XPath and XPointer [5] operators can formalize this condition, as depicted in figure 25.[m]

[j] In fact, this should be applied not only to a page, but to the related set of pages.
[k] $S1//*/@tabindex ∩ $S2//*/@tabindex = ∅
[l] $S1//*/@accesskey ∩ $S2//*/@accesskey = ∅
[m] $S2//a = ∅ ∨ (printable characters before first $S2's link ∧ printable characters after last $S2's link)

$S1//*/@tabindex intersect $S2//*/@tabindex = ()

Fig. 21. XPath pre-condition for avoiding elements with a common tabulation order. It is not required to check that they are positive numbers because it is already asumed that the snippets being compounded are already accessible.

$S2 = ... <a accesskey="C">A</a> ...
$S1 = ... <a accesskey="C">B</a> ... $S2 ...
result = BAD. Keyboard shortcuts must be based on unique keyboard-activable characters. Otherwise the shortcuts are ambiguous.

Fig. 22. Example showing a bad combination of keyboard shortcuts.

$S1//*/@accesskey intersect $S2//*/@accesskey = ()

Fig. 23. XPath pre-condition for avoiding elements with a common shortcut. No other restriction is needed to be checked.

$S2 = <a href="A">A</a> ...
$S1 = ... <a href="B">B</a> $S2 ...
result = BAD. Two consecutive links will appear with nothing more than a whitespace in the middle. This might confuse readers of voice browers looking as a single "big" link (instead of two different links)

$S2 = <a href="A">A</a> ...
$S1 = ... <a href="B">B</a> | $S2 ...
result = GOOD (something printable between consecutive links). Users will crearly identify two separate links.

Fig. 24. Example showing how two links might appear as consecutive, and a possible solution

$S2//a = () or
(*pos*($S1,$position)/preceding::a = () or string-length(normalize-space((end-point(
pos($S1,$position)/preceding::a[1])/range-to(*pos*($S1,$position)))/text())) > 0 or
string-length(normalize-space((start-point($S2)/range-to($S2//a[1]))/text())) > 0)
and
((*pos*($S1,$position)/following::a = () or string-length(normalize-space((end-point(
pos($S1,$position))/range-to(*pos*($S1,$position)/following::a[1]))/text())) > 0 or
string-length(normalize-space((end-point($S2//a[last()])/range-to(end-point($S2)))/text()))
> 0))

Fig. 25. XPath + XPointer pre-condition for avoiding consecutive links without printable non-linkable characters between them. It the internal snippet contains a link in one of its borders and the insertion point is close to a link in the external point, both links will appear next to each other, resulting in a problem. This expression guarantees that such condition will not happen.

Rule 13.1: Clear links

There should be no links sharing both a text and a title but pointing to different targets.[n] Provided $S1 and $S2 have no such ambiguous links, there exist a functional dependency such that for every pair of (link's contents, link's title) only a single target may be found in both $S1 and $S2. In that case, we should also make sure that no link in $S1 is similarly described in $S2 (and pointing to a different target), or vice-versa. If so, an ambiguity would be introduced in the composed result. Figure 26 describes such condition.

$S2 = ... <a href="B">A</a> ...
$S1 = ... <a href="C">A</a> ... $S2 ...
result = BAD (two similar links, sharing the same text, but pointing to different resources). This leads to navigation problems based on semantic inconsistencies.

$S2 = ... <a href="B" title="D">A</a> ...
$S1 = ... <a href="C">A</a> ... $S2 ...
result = GOOD (though the links share the same text, their "title" attributes are different). It is OK to have several links having the same text inside, but if they also have the same "title" attributes, they should point to the same URL.

Fig. 26. Example of bad and good combination of links pointing to different targets but sharing the same text

Figure 27 shows the XPath 2.0 condition that must be met to avoid confusion. All $S1's links sharing the same text and the same title attribute of any $S2's link, also must share the same target (`href` attribute), and vice-versa.

```
(every $a1 in $S1//a satisfies
$S2//a[text() = $a1/text() and @title = $a1/@title and @href != $a1/@href] = ()) and
(every $a2 in $S2//a satisfies
$S1//a[text() = $a2/text() and @title = $a2/@title and @href != $a2/@href] = ())
```

Fig. 27. XPath pre-condition for avoiding ambiguous links. Both external and internal snippets must not have a link with the same text inside, the same `title` attributes and different targets.

5 Guaranteed-by-process checkpoints

Previous pre-conditions should be met in order to guarantee propagation of WAI compliance from both $S1 and $S2 snippets to the composition result of both pieces. Those rules are required to be followed for this propagation. Otherwise, accessibility features asumed in those snippets will probably not be achieved in the result. On the other hand, any other checkpoint not depicted in section 3 will be automatically propagated. In other words, if both $S1 and $S2 follow these other checkpoints mentioned in this section, the composition result *comp*($S1,$S2,$position) will surely follow them too. Here are some examples and their explanations. We don't include all of them. They are all WCAG which are not included within section 3.

[n]This would lead to ambiguity, because two similar starting links point to different targets.

Rule 1.5: Redundant links for client image maps

Provided that both $S1 and $S2 have redundant links for their client image maps (areas), the composition result of $S1 and $S2 will also have those redundant links (at least), thus it will also comply WCAG 1.5.

PROVIDED that snippets have redundant links for their areas, i.e. that any link inside an area is redundantly also somewhere else in the document, i.e.
$S1//area[let $href := self::area/@href return count($S1//a[@href = $href])= 0] = () ∧
$S2//area[let $href := self::area/@href return count($S2//a[@href = $href])= 0] = ()

WE CONCLUDE that composition result will also have redundant links
comp($S1,$S2,$position)//area[let $href:=self::area/@href return
count(*comp*($S1,$S2,$position)//a[@href=$href])=0]=()

BECAUSE those redundant links will ALSO be in the composition result
comp($S1,$S2,$position)//a = $S1//a ∪ $S2//a

Rule 12.4: Labels

Provided that both $S1 and $S2 have no label shared by two form fields, the composition result of $S1 and $S2 will not have any label shared by two form fields, because it is not possible that the other piece brings a form field whose `id` attribute is equal to any other element already used. So, because `id` attributes are required to be unique (see rule 3.2: validation), the number of form fields pointed by a label can not incidentally be increased in the composition process. It must be 1. Otherwise a validation error message spotting that two elements share a common `id` will appear.

PROVIDED that snippets have proper labels which are being mapped to form fields, i.e.
$S1//label[let $l:=self::label return
count($S1//(select | input | textarea)[@id=$l/@for]) != 1] = () ∧
$S2//label[let $l:=self::label return
count($S2//(select | input | textarea)[@id=$l/@for]) != 1] = ()

WE CONCLUDE that composition result will also have proper labels being properly mapped to form fields with no cross-reference between a link from one snippet and a label from the other snippet
comp($S1,$S2,$position)//label[let $l:=self::label return
count((*comp*($S1,$S2,$position)//(select | input | textarea)[@id=$l/@for])!=1]=()

BECAUSE *id* attributes must be unique, making impossible that a label from one snippet points to a form field from the other snippet.

Of course, it would be possible to have a label in one snippet and its form field in other snippet, but that would break our asumption that both snippets comply with this checkpoint. It that case, accessibility would be upgraded during the composition process, but we would be joining two non-accessible snippets.

Rule 12.4: Form fields

Provided that both $S1 and $S2 have no form field shared by two labels, the composition result of $S1 and $S2 will have no form fields shared by two labels, because it is not possible that the other snippet brings a label whose `for` attribute is equal to any other element already used. Only if such label was not associated to any form field, we could have a valid document that presents a form field having more that one associated label. However, this is not possible provided the conditions from previous rule 12.4 for labels.

PROVIDED that snippets have proper form fields, i.e.
$S1//(select | textarea | input[@type="text" or @type="password" or @type="radio" or @type="checkbox"])[let $ff:=self::node() return
count($S1//label[@for=$ff/@id]) != 1] = () ∧
$S2//(select | textarea | input[@type="text" or @type="password" or @type="radio" or @type="checkbox"])[let $ff:=self::node() return
count($S2//label[@for=$ff/@id]) != 1] = ()

WE CONCLUDE that composition result will also have proper snippets
comp($S1,$S2,$position)//(select | textarea |
input[@type="text" or @type="password" or @type="radio" or
@type="checkbox"])[let $ff:=self::node() return
count(*comp*($S1,$S2,$position)//label[@for=$ff/@id]) != 1] = ()

BECAUSE *id* attributes must be unique

5.1 Other checkpoints

All other checkpoints not previously mentioned, though not detailed fall within this category. The existence of alternative contents and the proper usage of headings, lists or cites are characteristics that will never be broken by the Web Composition process.[o]

6 WCAG support

XPath and XQuery expressions from previous sections spot HTML nodes and attributes having accessibility problems. These elements should be properly managed by an authoring tool, so that author's attention can be directly brought to these barriers in a semi-automated edition process like our WSLS platform, making easier the creation of accessible contents.

6.1 Aspects of a Supporting System

The high availability of processors and tools to apply the presented XML-based technologies eases the implementation of the formalized rules. Unfortunately, , not all WCAG rules can be expressed with W3C standards like XPath or XPointer. Sections 3 and 4 introduced a set of checkpoints that exceeds the capabilities of such functional query languages. To automate or at least semi-automate, these checkpoints demand for a supporting system that facilitates compliance in the integration of Web contents obtained from distributed heterogeneous sources by *process* and by *architecture*. Compliance by process means that checkpoints are controlled by a system providing dedicated processes. On the other hand, compliance

[o]As long as snippets are accessible and XHTML validation is achieved.

by architecture relies on the overall structure of the system and its design decisions to keep checkpoints or provide mechanisms to formulate them. Formalizing the checkpoints from sections 3 and 4 requires for imperative programming languages like Java, C# or C++ to express the complex algorithms. Furthermore, the supporting system must provide additional environment information. This information can be of different fashion concerning the audience, the user agent or even constraints which a user is operating under, e.g. noisy surrounding or hands-free environment.

6.2 Web-Composition Service Linking System

The Web-Composition Service Linking System (WSLS) [15] is a component-based system which applies the service-oriented paradigm. It supports issues of composing, discovering and reusing services. A WSLS-based service is a maintainable, manageable and configurable building block. Hence, it forms a unit for reuse and allows the composition of Web applications similar to the building block principle. The process of composing such services is guided in a systematic way by the underlying WSLS framework. Figure 28 depicts the general form of each service within WSLS. A service is understood as a component that consists of a set of *service elements*. A service element achieves dedicated tasks corresponding to its classification. The classification scheme provides six distinguished categories: Data, Presentation, Navigation, User Interaction, Process and Communication. The six elements are *mediated* by a service control function. The WSLS approach follows the separation of concerns principle.

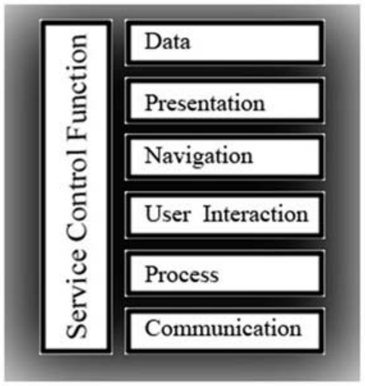


Fig. 28. WSLS follows the separation of concerns principle to decompose complexity and control accessibility

Beyond the advantage of the reuse aspect of these components, the separation of concerns facilitates also being compliant to underlying guidelines. Thus, the presentation of a service can be adapted in a fast and efficient fashion to local prevailing presentation constraints. Furthermore, besides presentation guidelines that simply concern the layout of services, more powerful rules regarding the accessibility of a service can be realized, like e.g. U.S. Section 508 Guidelines [14].

Providing services through the Web is rapidly becoming the emerging trend. As more and more governments adopt laws regarding accessibility, enterprises are also affected. While some laws, like Section 508, currently impact only the US. federal government, there is even a trickle-down effect to private enterprises. Companies with contracts to supply government with products stand to lose their business if the products are not accessible or suitable for creating accessible Web sites. WSLS faces these requirements by providing support for designing and implementing accessible building blocks as well as procedures for their composition.

6.3 WCAG support in WSLS

The WSLS framework is implemented on top of the .NET framework. Therefore, the support of XML technologies like XPath or XSL(T) is given and can be applied systematically. Hence, we started implementing the automatable checkpoints regarding these technologies. The realization of the unaffected rules stated in sections 3 and 4 poses a non-trivial challenge at the underlying supporting system. Subsequently, we show for some selected rules how WSLS supports the abidance on these.

Rule 13.2: "Provide meta-data"

"Providing additional data to add semantic information to pages and sites."

Figure 29 depicts the general structure of a data element as prescribed in the WSLS framework. Specialized data objects can be created by inheriting from the base *DataObject* class to fulfill dedicated tasks. By default, each data element used in WSLS possesses meta-data. This comprises at least the Dublin Core standard [18]. In order to support other specifications needed to satisfy arbitrary environments, these objects are also extensible. Thus, each data object can provide specialized meta-data, if necessary. This is particularly useful in different scenarios, e.g. to indicate a document's author, content type together with the last modification date. Furthermore, this extensible approach allows for enhanced scenarios where the semantic description is reused by other processes.

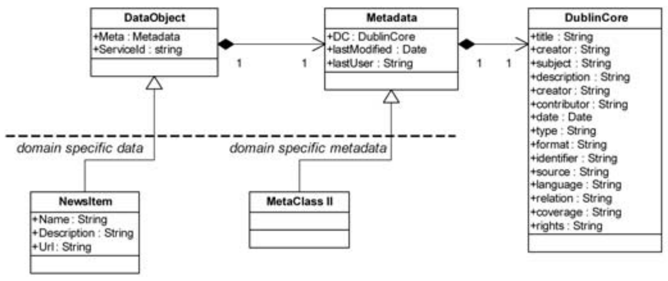


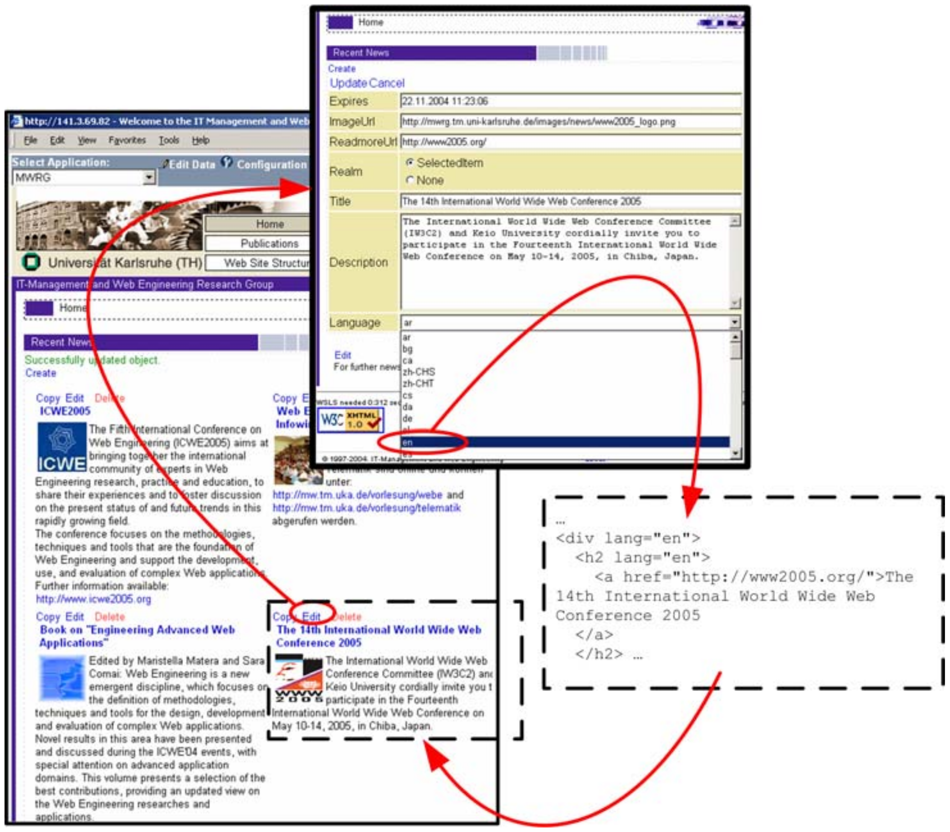
Fig. 29. Prescribed structure of a data element in WSLS

Rule 4.1: "Identify changes in the natural language"

"Clearly identify changes in the natural language of a document's text and any text equivalents (e.g., captions)."

As described above, each data element provides at least Dublin Core meta-data. This information is used to provide language relevant information for presented data. Figure 30 depicts a Web application that was built on top of the WSLS framework. The scenario shows a news service that displays recent news items. WSLS provides standard user interaction behaviour like creating, editing and deleting data elements of a given service. Therefore, the system even supports the automatic generation of forms for editing data elements of standardized Web services. The process of editing such a data element is demonstrated.

Besides the application of the presented formalization rules, the content author is prevented from writing wrong or insufficient specified HTML code by supporting guiding processes that query for the necessary data. This data is automatically incorporated into the HTML output as prescribed by the corresponding formalization rule. In this example the author is asked for the specific language of the content provided.


http://141.3.69.82 - Welcome to the IT Management and Web
File Edit View Favorites Tools Help
Select Application:
MWRG
Edit Data
Configuration
Home
Publications
Web Site Structur
Universität Karlsruhe (TH)
IT-Management and Web Engineering Research Group
Recent News
Successfully updated object.
Create
Copy Edit Delete
ICWE2005
Book on "Engineering Advanced Web Applications"
The 14th International World Wide Web Conference 2005
Update Cancel

Expires	22.11.2004 11:23:06
ImageUrl	http://mwrg.tm.uni-karlsruhe.de/images/news/www2005_logo.png
ReadmoreUrl	http://www2005.org/
Realm	(● SelectedItem) ○ None
Title	The 14th International World Wide Web Conference 2005
Description	The International World Wide Web Conference Committee (IW3C2) and Keio University cordially invite you to participate in the Fourteenth International World Wide Web Conference on May 10-14, 2005, in Chiba, Japan.
Language	ar

```
…
<div lang="en">
  <h2 lang="en">
    <a href="http://www2005.org/">The
14th International World Wide Web
Conference 2005
  </a>
  </h2> …
```

Fig. 30. Using metadata to provide natural language relevant information

7 Conclusions and future work

This article has presented conditions formalized with W3C standards that guarantee that the Web Composition [13] approach will always yield accessible contents provided that accessible ingredients are used in that composition. As expressed in section 4, some accessibility conditions will be inherited in a bottom-up propagation from smaller HTML pieces being composed towards the final composition result, because the Web Composition process guarantees the bottom-up propagation from the ingredients to the result. However, other accessibility features require some extra conditions in order to be bottom-up propagated. Such conditions have been formally expressed in section 3. Thus, we have formalized the conditions that will avoid that accessibility barriers might be introduced by the Web Composition process itself. These conditions enable the propagation of accessibility from the HTML ingredients towards the result. We also introduced WSLS, a supporting system that affords compliance to these rules by process and by architecture. The realization of selected rules in WSLS yielded already to promising results. For the future we plan to adopt architectural issues as well as augmented processes that guide the design, implementation and evolution of sites and applications towards the vision of the compliant and accessible Web.

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.

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 *XML Pointer Language (XPointer), W3C Working Draft 16 August 2002*
   `www.w3.org/TR/xptr`
6. W3C *XHTML 1.0* ^{TM} *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`
7. W3C *XHTML Basic W3C Recommendation 19 December 2000*
   `www.w3.org/TR/xhtml-basic`
8. W3C *Markup Validation Service*
   `validator.w3.org`
9. Watchfire *WebXM Accessibility tool*
   `www.watchfire.com/products/webxm`
10. CEAPAT, Fundación CTIC, Spanish Ministry of Employment and Social Affairs (IMSERSO) *Online Web accessibility test*
    `www.tawdis.net`
11. Fondazione Ugo Bordoni *Torquemada, Web for all*
    `www.webxtutti.it/testa_en.htm`
12. Fundación SIDAR *Accessibility testing with Style*
    `www.sidar.org/hera`
13. Hans-Werner Gellersen, Robert Wicke and Martin Gaedke *WebComposition: An object-oriented support system for the Web engineering lifecycle*
    Computer Networks and ISDN Systems, Vol 29, 8-13, 1997, pages 1429-1437
14. Center for IT Accommodation (CITA) *U.S. Section 508 Guidelines*
    `www.section508.gov`
15. Gaedke, M., Nussbaumer, M., and Meinecke, J. *WSLS: An Agile System Facilitating the Production of Service-Oriented Web Applications*
    Engineering Advanced Web Applications, M. Matera and S. Comai, Editors. 2004, Rinton Press