International Journal of Intelligence Science, **2014, 4, 29-37**
Published Online January 2014 (http://www.scirp.org/journal/ijis)
http://dx.doi.org/10.4236/ijis.2014.41005

Scientific Research

The Knowledge Base Development for the Web Content Accessibility Guidelines

Yui-Liang Chen[1], Limin Liu[2]
[1]Department of Information Management, Shih Hsin University, Mu-Cha, Taiwan
[2]Department of Applied Mathematics, Chung Yuan Christian University, Chung-Li, Taiwan
Email: lmliu@math.cycu.edu.tw

Received October 18, 2013; revised November 18, 2013; accepted November 25, 2013

ABSTRACT

Web Content Accessibility Guideline (WCAG) proposed by Web Accessibility Initiative is the most recognized regulation in the world in evaluating the accessibility of web contents and is one of the W3C documentations. Currently, the WCAG documents are maintained as a non-computable dictionary like resources. In this study, we proposed a methodology to develop an ontological knowledge base with rules for the WCAG 2.0. Two WCAG techniques in different groups of techniques are used to illustrate the creation of the knowledge base and their application programming interfaces. For demonstrating purpose, a web-based WCAG validation system is built. With the proposed knowledge base and interfaces, computer programs can decide whether certain web content satisfies a particular WCAG technique. In other words, the WCAG documents have been successfully transformed to a computable recourse. Such sharable knowledge base and programming interfaces can be embedded into any system requiring the WCAG knowledge.

KEYWORDS

Knowledge Base; Ontology; Web Accessibility; WCAG; Intelligent System

1. Introduction

In order to improve the accessibility of web contents for people with particular disability, the WAI (Web Accessibility Initiative) had released guidelines called the Web Content Accessibility Guidelines, WCAG 1.0, as one of the W3C (World Wide Web Consortium) documentations since 5 May 1999. The most recent version of the WCAG is the version 2.0 and it is still an active project. Meaningful comments or newly invented technologies may create new guidelines for WCAG 2.0 [1]. The most recent documents of the WCAG 2.0 techniques were updated on Jan. 2012 with 437 techniques [2]. These techniques have a many-to-many relationship to the 61 WCAG success criteria. With such a complexity, it requires significant resources to fully understand the WCAG 2.0, and it is practically impossible to manually verify whether a certain amount of web contents satisfy certain techniques or success criteria. In order to do so, we need to have an intelligent system which has the knowledge listed behind the guidelines of WCAG 2.0.

Ontology-based knowledge systems have been used to handle complicated knowledge intensive problems with many successful cases in different application domains [3, 4]. One of the essential parts of building such intelligent systems is to correctly develop the knowledge base, KB, behind it. The resulting ontology can be considered as shared resources, e.g. UMLS, SNOMED, and WordNet, etc. [5-8]. These ontologies not only properly represent particular domain knowledge, but also can be used by third-party applications via a local installation or application programming interfaces.

The focus of this study is to develop a knowledge base with ontology and semantic rules to preserve the knowledge described in the WCAG 2.0 documentations. Since the knowledge base is a sharable resource, it can be embedded into software systems requiring the knowledge of WCAG 2.0. Developing such a WCAG KB is an expert intensive task that requires both ontology and WCAG

experts to work closely to generate the correct result. Fortunately, such an effort is a one-time investigation and the resulting KB can be shared and reused. In addition to the KB, we also developed a set of WCAG application programming interfaces (APIs). Application developers can use these APIs without the knowledge of ontology, logic, or KB behind them.

The proposed KB and APIs can be used in various ways. For example, one can develop a standalone system to validate the level of accessibility of web contents. The software system called Free go is an example of a tool developed to verify the level of accessibility of web contents on WCAG 1.0 [9]. Since the WCAG 2.0 is much more complicated than version 1.0, sophisticated mechanisms such as ontological knowledge base systems may be more appropriate for developing tools for WCAG 2.0. In this study, a web-based WCAG validation system is built to illustrate the usage of the proposed KB and APIs.

The rest of the paper is organized as follows: Section 2 reviews the background knowledge of this study including WCAG 2.0 and ontology technology. Section 3 illustrates the proposed WCAG knowledge base creation using two specific technologies, H32 and C8. Section 4 contains the experiments of developing the web-based WCAG validation system. Finally, the conclusions and future works are presented in Section 5.

2. Backgrounds

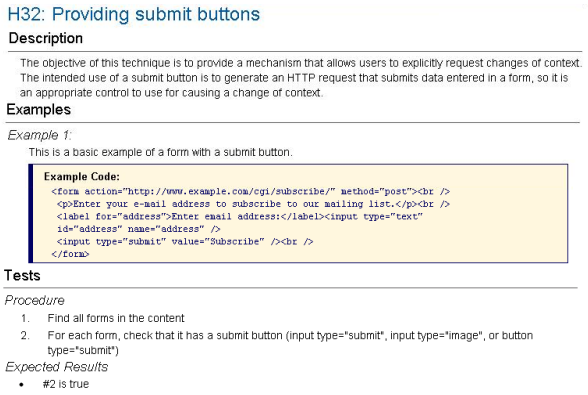
2.1. WCAG 2.0

The WCAG 2.0 documentations are organized in four layers (from top): *Principles*, *Guidelines*, *Success Criteria*, and *Sufficient and Advisory Techniques*. Principles lay the foundation necessary for anyone to access/use web contents. Four principles are listed in the WCAG 2.0: *perceivable*, *operable*, *understandable*, and *robust* (numbered from 1 to 4). Under principles are 12 guidelines. Principle perceivable, operable, understandable, and robust has 4, 4, 3, and 1 guideline, respectively. For example, the second principle (operable) is "User interface components and navigation must be operable" and its first guideline (numbered as 2.1) is "Make all functionality available from a keyboard."

Each WCAG guideline has a list of testable success criterion (SC) and there are 61 SC defined in WCAG 2.0. For example, the guideline 2.1 has three SC (numbered from 2.1.1 to 2.1.3). Every SC is assigned with a conformance level (A, AA, or AAA) where level A indicates the lowest level (the easy requirements to fulfill) and AAA the highest (the difficult one). The SC 2.1.1, 2.1.2, and 2.1.3 are set with level A, A, and AAA, respectively. Several factors are evaluated when WAI sets level to SC. For instance, "whether the Success Criterion is essential" is one of the factors.

Since web contents may adopt different kinds of techniques, WCAG 2.0 lists 437 techniques and places them into 12 groups, e.g. group *CSS* techniques (with 22 techniques) and *HTML and XHTML* techniques (with 60 techniques) [2]. **Figure 1** lists an important portion of the document of technique H32 of the HTML and XHTML techniques. WCAG 2.0 documents list not only the description of the technique, but also examples and (test) procedure of the technique. The H32 technique requires all forms in a HTML file having a submit button. Because a submit button can appear in three methods (an input tag with type = "submit", an input tag with type = "image", or a button with type = "submit"), the regulation lists them all in the document.

Figure 2 lists an important portion of the document of technique C8 of the CSS techniques. This technique suggests the white space between characters in each word should be increased to make these words visually readable. The method to do so is to use the CSS *letter-spacing* property. The example shown in **Figure 2**



H32: Providing submit buttons

Description

The objective of this technique is to provide a mechanism that allows users to explicitly request changes of context. The intended use of a submit button is to generate an HTTP request that submits data entered in a form, so it is an appropriate control to use for causing a change of context.

Examples

Example 1:

This is a basic example of a form with a submit button.

Example Code:

```
<form action="http://www.example.com/cgi/subscribe/" method="post"><br />
<p>Enter your e-mail address to subscribe to our mailing list.</p><br />
<label for="address">Enter email address:</label><input type="text"
id="address" name="address" />
<input type="submit" value="Subscribe" /><br />
</form>
```

Tests

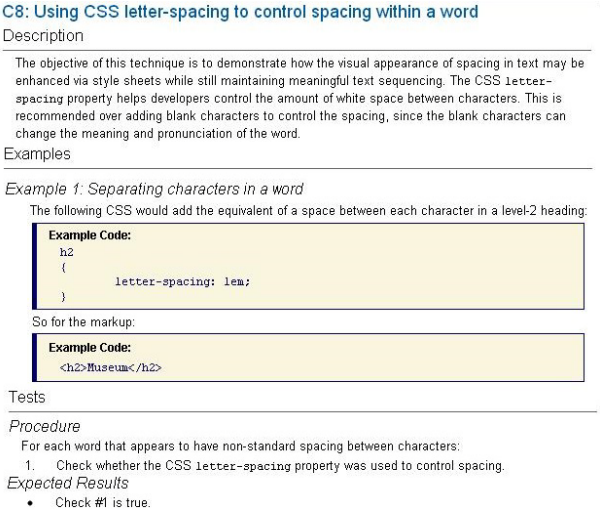
Procedure

1. Find all forms in the content
2. For each form, check that it has a submit button (input type="submit", input type="image", or button type="submit")

Expected Results

- #2 is true

Figure 1. Part of the technique H32 documentation.



C8: Using CSS letter-spacing to control spacing within a word

Description

The objective of this technique is to demonstrate how the visual appearance of spacing in text may be enhanced via style sheets while still maintaining meaningful text sequencing. The CSS `letter-spacing` property helps developers control the amount of white space between characters. This is recommended over adding blank characters to control the spacing, since the blank characters can change the meaning and pronunciation of the word.

Examples

Example 1: Separating characters in a word

The following CSS would add the equivalent of a space between each character in a level-2 heading:

Example Code:

```
h2
{
        letter-spacing: 1em;
}
```

So for the markup:

Example Code:

```
<h2>Museum</h2>
```

Tests

Procedure

For each word that appears to have non-standard spacing between characters:

1. Check whether the CSS `letter-spacing` property was used to control spacing.

Expected Results

- Check #1 is true.

Figure 2. Part of the technique C8 documentation.

has a word “Museum” surrounding by tag “h2” which is defined with *letter-spacing* property in the header. To meet this regulation, one needs to verify all words with their associated heading tags, e.g. “h2”. More precisely, if “<h2> Museum </h2>” is changed to “<h3> Museum </h3>”, then this particular example does not meet the C8 requirement, since the *letter-spacing* property is defined only for heading “h2”.

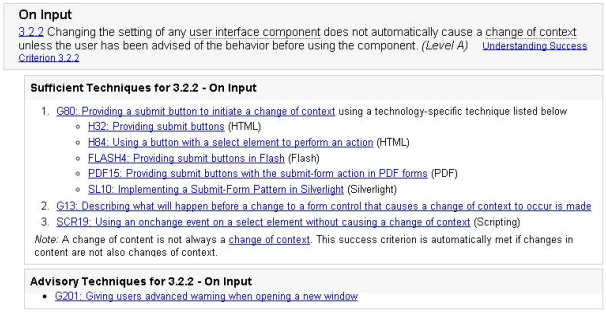
WCAG 2.0 techniques fall into two categories: those that are sufficient for meeting a SC and those that are advisory. In order to carefully cover all cases, sufficient techniques may contain several *situations* to describe different cases. For example, the SC 3.2.5 has four situations (from A to D). Situations listed in a particular criterion can be mutually exclusive, e.g. situations in SC 3.3.5, or in some cases, situations can overlap (can be applied on one single web content), e.g. situations in SC 3.2.5. **Figure 3** shows the sufficient and advisory techniques of the success criterion 3.2.2 (On Input).

In addition, it is possible that a particular technique be part of a sufficient technique of a SC and part of an advisory technique of another SC. For instance, the CSS technique C8 is one of the sufficient techniques of the SC 1.3.2 and also one of the advisory techniques of the SC 1.4.5.

2.2. Ontology Technology

Philosophically, the term ontology refers to the study of the nature of being or the kinds of things [10]. In computer science (or artificial intelligence in specific), an ontology is an explicit specification mechanism or a specification of conceptualization [11]. Ontology is an emerging technology for implementing a shared understanding of information and many ontology-based knowledge base systems have been developed in the past decades [5-8,12,13].

The proposed knowledge base contains two parts: an ontology (taxonomy) and semantic rules. In this study, the Web Ontology Language (OWL) is used to represent the knowledge of the WCAG 2.0. OWL is a XML-based semantic markup language developed by W3C [14].


On Input

3.2.2 Changing the setting of any user interface component does not automatically cause a change of context unless the user has been advised of the behavior before using the component. *(Level A)* Understanding Success Criterion 3.2.2

Sufficient Techniques for 3.2.2 - On Input

1. G80: Providing a submit button to initiate a change of context using a technology-specific technique listed below
   - H32: Providing submit buttons (HTML)
   - H84: Using a button with a select element to perform an action (HTML)
   - FLASH4: Providing submit buttons in Flash (Flash)
   - PDF15: Providing submit buttons with the submit-form action in PDF forms (PDF)
   - SL10: Implementing a Submit-Form Pattern in Silverlight (Silverlight)
2. G13: Describing what will happen before a change to a form control that causes a change of context to occur is made
3. SCR19: Using an onchange event on a select element without causing a change of context (Scripting)

Note: A change of content is not always a change of context. This success criterion is automatically met if changes in content are not also changes of context.

Advisory Techniques for 3.2.2 - On Input

- G201: Giving users advanced warning when opening a new window

Figure 3. The sufficient and advisory techniques of the success criterion 3.2.2 (On Input).

OWL contains *classes*, *individuals*, and *properties* that correspond to *concepts*, *instances*, and *roles* in ontology. Classes provide an abstraction mechanism with similar characteristics and each class is associated with a set of individuals. There are three types of properties: *object*, *data type* and *annotation* properties. The first one represents semantic relations between individuals, the second one links individuals to data values, and the last one indicates additional notes.

According to the W3C specifications, OWL has three increasingly expressive sublanguages for different levels of usability: OWL Lite, OWL DL, and OWL Full. The most popular version is OWL DL where DL stands for Description Logic (DL) which is a decidable fragment of First Order Logic. Inference of OWL ontologies can, therefore, be handled by DL reasoners. This study also employs semantic rules in the Semantic Web Rule Language (SWRL) which provide procedural knowledge to increase the inference power of ontology, especially in identifying relationships between individuals [15]. The ontology editor used in this paper is the Protégé 4.2 [16] with the OWL reasoner Pellet [17].

3. Knowledge Base Modeling

3.1. Ontology Development

Since WCAG 2.0 contains 12 different groups of techniques, each group of techniques will have its own subtree in the ontology. With the subtree representing WCAG 2.0 itself, there are 13 classes at the top level in the class hierarchy (under class **Thing**). For the sake of brevity, the figures shown in this paper contains only three classes at the top level: **WCAG2.0**, **HTML**, and **CSS** because this study uses only techniques H32 and C8 to illustrate the ontology development.

3.1.1. WCAG 2.0 Subtree

The proposed ontology of WCAG 2.0 subtree is rooted with the class named **WCAG2.0** which has five subclasses: **Level**, **Principles**, **Guidelines**, **Success_Criteria**, and **Techniques**, as shown in **Figure 4**.

Since there are only three conformance levels in WCAG 2.0, three mutually exclusive subclasses (**Level_A**, **Level_AA**, and **Level_AAA**) are created under class **Level**. Class **Principles** has four subclasses to represent the four principles listed in the WCAG 2.0. Similarly the 12 guidelines, 61 SC and 437 techniques are organized as subclasses of class **Guidelines**, **Success_Criteria**, and **Techniques**, respectively. Subclasses of **Guidelines** are prefixed with numbers as shown in **Figure 4**. Subclasses of **Success_Criteria** are prefixed with “**SC_**” followed by the number. For instance, SC 3.2.2 shown in **Figure 3** has an associated class **SC_3_2_2** under class **Success_Criteria**. Subclasses of

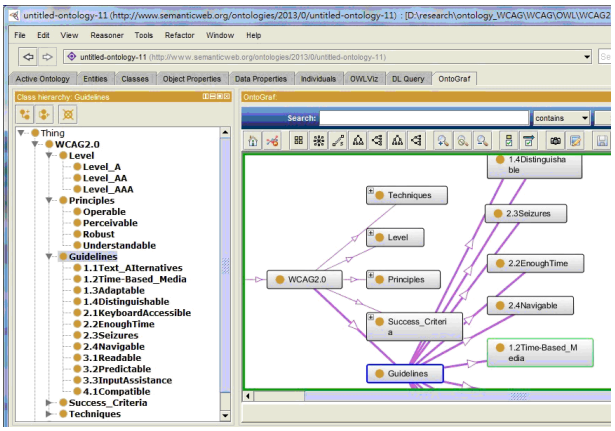


Figure 4. Class WCAG2.0 with five subclasses.

Techniques are prefixed with "**TC_**" followed by the name of the technique. For instance, class **TC_HTML And XHTML** and **TC_CSS** represent the groups of techniques HTML and XHTML and CSS. Particular techniques are defined under their associated class, e.g. class **H_32** and **C_8** are defined under **TC_HTML And XHTML** and **TC_CSS**, respectively.

Object property *hasGuideline* is defined in class **Principles** pointing to class **Guidelines**; property *hasSuccess Criteria* is defined in class **Guidelines** pointing to class **Success_Criteria**; similarly, property *hasTechnique* is defined in class **Success_Criteria** pointing to class **Techniques**. In addition, **Success_Criteria** has an object property *hasLevel* pointing to one of the subclasses of class **Level**, i.e., the range of *hasLevel* is either **Level_A**, **Level_AA**, or **Level_AAA**. Additional datatype properties are defined for these classes for preserving text-based information in WCAG document. For instance, class **Techniques** has datatype property *description* (in plain text) to keep the description defined in WCAG document. For the sake of brevity, these properties are not shown or described in this paper.

3.1.2. HTML Subtree

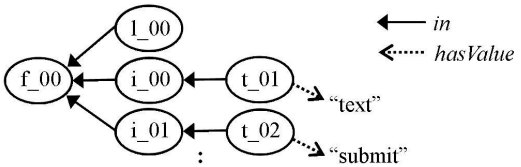
The other two classes other than **WCAG2.0** under **Thing** are **HTML** and **CSS** representing ontologies of HTML and CSS syntax. Ideally, if we have shareable KBs for HTML and CSS, we can adopt those KBs into our WCAG KB and make some necessary extensions for this study. In that case, we do not need to develop HTML and CSS ontologies from scratch. Unfortunately, such shareable resources do not exist based on our knowledge. Therefore, in order to make our WCAG KB functional, we also developed HTML and CSS ontologies. For easy implementation, there are only three levels of classes defined for HTML. Such a design is not perfect but functionally correct. The root class is **HTML** and all valid HTML tags and attributes will have an associated class under **HTML** with a prefix "**HT_**". For example, class **HT_form**, **HT_input**, **HT_type**, **HT_heading**, **HT_button** represents HTML forms, input, type, heading, and button, respectively. In additions, two classes that are not html tag or attribute are also created: class **HT_page** and **HT_word**. Class **HT_page** represents either static or dynamically created web contents and class **HT_word** represents words shown in web content.

One of the object properties defined in **HTML** is *in* that represents the semantic of containing. A datatype property, named *hasValue*, is also defined on class **HTML** since many of the attributes require associated values. For instance, the HTML attribute "type" of a particular input tag can have values like "text", "hidden", or "submit", etc. Such value is stored in the *hasValue*. Furthermore, there are 6 types of HTML heading (h1 to h6) and the actually type of a particular heading is also stored in *hasValue*. For instance, for a text string *s* defined within tag <h2>, an individual of **HT_heading** will be created with property *hasValue* "h2". For each word in *s*, an individual of **HT_word** will be created with property *in* linking to the associated **HT_heading** individual and *hasValue* containing the particular word it represents.

In order to demonstrate techniques H32 and C8, we created eight subclasses under class **HTML**. Individuals with prefix "p_", "f_", "l_", "i_", "b_", "w_", "h_", and "t_" belong to class **HT_page**, **HT_form**, **HT_label**, **HT_input**, **HT_button**, **HT_word**, **HT_heading**, and **HT_type**, respectively.

With **HTML** and its subclasses, one can represent HTML file as individuals in the proposed ontology. The HTML code, shown at the bottom of **Figure 5**, will have the associated individual network shown on top of the same figure. To save space, we show only a portion of the individual network. Individual f_00 represents this form that contains a label (l_00), two inputs (i_00 and i_01), and a paragraph tag (does not show in the figure). Both input tags have type attributes (t_01 and t_02) and they are linked by *in* to their associated input tags. The value of attribute type is preserved by *hasValue* property and shown by a dashed arrow in the figure.

The other two object properties used in this study are: *hasForm* and *pageHasWord*. The domain and range of




```
<form action="http://example.com/a.asp" method="post"><br>
 <p>Enter your e-mail address to subscribed</p><br>
 <label for="address">Enter email address:</label>
 <input type="text" id="address" name="address"/><br>
 <input type="submit" value="Subscribe"/><br>
</form>
```

Figure 5. Representing HTML code to KB individuals.

hasForm are **HT_body** and **HT_form**. The domain and range of *pageHasWord* are **HT_page** and **HT_word**.

3.1.3. CSS Subtree

The CSS subtree is similar to the HTML subtree because most of the tags defined in the CSS style section are HTML tags. The root class of the CSS subtree is called **CSS** and all its subclasses have prefix "**CSS_**". In this study, two subclasses (**CSS_letter_spacing** and **CSS_heading**) are created under class **CSS** with individuals prefixed by "css_ ls_" and "css_".

Figure 6 shows an example of mapping an html page with CSS style section into an individual network. The four words in string "Museum of modern art" are represented by w_01 to w_04 and they are surrounded by two html heading tags <h2> and <h4>, represented by individuals h_01 and h_02. Individual h_01/h_02 *hasValue* "h2"/"h4" that shows what type of the heading it is. The two heading defined in CSS section is represented by css_01 and css_02 with *hasValue* of "h2" and "h4". The *letter-spacing* attribute defined in the CSS heading are represented by individuals css_ls_01 and css_ls_02 with *in* property linking their associated CSS heading individuals. The actual values of letter spacing are defined by the *hasValue* property in css_ls_01 and css_ls_02. Individual css_01, css_02, and b_01 are defined *in* the HTML page p_01. Individual p_01 links to w_01~w_04 via object property *pageHasWord*.

3.2. Semantic Rules Creation

3.2.1. Modeling WCAG Technique H32

To properly represent techniques of WCAG 2.0, ontology engineers need to carefully review the description and test procedure of techniques. They need to follow the testable procedure to translate them to rules in SWRL format. For example, the second step of the procedure of H32 (shown in **Figure 1**) indicates that we need to check whether there is a submit button in each HTML form and there are three possible cases: (1) input type = "submit", (2) input type = "image", and (3) button type = "submit". To represent these three cases, three Horn-like rules are created listed below:

$$HT\_form(?f) \wedge HT\_input(?i) \wedge HT\_type(?t) \wedge in(?i,?f) \wedge in(?t,?i) \wedge hasValue(?t,?s) \wedge stringEqualIgnoreCase(?s,"submit") \rightarrow H\_32\_form(?f) \quad (1)$$

$$HT\_form(?f) \wedge HT\_input(?i) \wedge HT\_type(?t) \wedge in(?i, ?f) \wedge in(?t, ?i) \wedge hasValue(?t,?s) \wedge stringEqualIgnoreCase(?s,"image") \rightarrow H\_32\_form(?f) \quad (2)$$

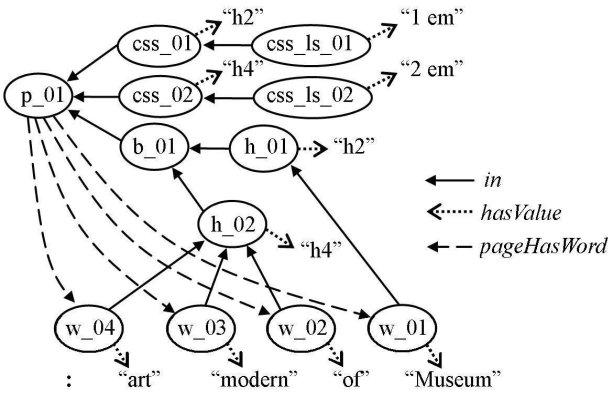
$$HT\_form(?f) \wedge HT\_button(?b) \wedge HT\_type(?t) \wedge in(?b,?f) \wedge in(?t,?b) \wedge hasValue(?t,?s) \wedge stringEqualIgnoreCase(?s,"submit") \rightarrow H\_32\_form(?f) \quad (3)$$

Rules (1)-(3) are translated into SWRL format as shown in **Figure 7**. If a particular form satisfies any of these three rules, the individual represents this form will also be assigned as an inferred individual (member) of class **H_32_form** (a subclass of both class **H_32** and **HT_form**). However, the technique H32 requires "all forms" in a page satisfying this constraint. To handle such a universal quantification, a *defined* class called **H_32_body** is created under class **HT_32**. Class **H_32_body** is defined as "equivalent to" (the necessary and sufficient conditions) "**HT_body** and (*hasForm* some **HT_form**) and (*hasForm_H32* **only H_32_form**)" as shown in **Figure 7**. Object property *hasForm_H32* is a sub property of *hasForm* with range defined as **H_32_form**. Hence, if an **HT_Body** individual contains *hasForm_H32* to only **H_32_form**, it will be assigned as an inferred member of class **H_32_Body** by reasoners.

If a web page, *p*, contains a body satisfying a particular technique, then *p* satisfies this technique by nature. To represent such a concept, a class named **H_32_page** under class **HT_32** and **HT_Page** is created. A Horn-like rule is created for populating individuals of this class:

$$HT\_page(?p) \wedge HT\_32\_body(?b) \wedge in(?b, ?p) \rightarrow H\_32\_page(?p) \quad (4)$$

Hence, if a page containing a body in class **H_32_body**, it will be assigned as an inferred member of class **H_32_page** by this rule (4), as shown in **Figure 7**.




```
<html> ...
 <style type="text/css">
   h2 { letter-spacing: 1em }
   h4 { letter-spacing: 2em } ...
 </style>
<body>
 <h2>Museum</h2><h4>of modern art</h4> ...
</body> </html>
```

Figure 6. Representing HTML code with CSS to KB individuals.

3.2.2. Modeling WCAG Technique C8

Since representing technology C8 in the proposed ontology only requires two CSS styles heading and *letter-spacing*, under class **CSS**, we created only two classes:

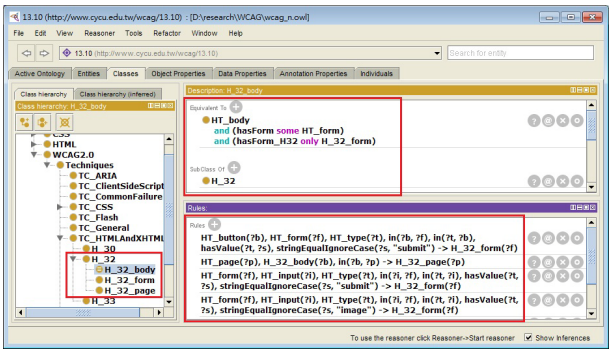


Figure 7. Class H_32_body and rules for technique H_32_form.

CSS_heading and **CSS_letter_spacing**. The CSS technique C8 requires text in an html page has more blank space between characters that can be achieved by using CSS style *letter-spacing* as shown in Figure 2. The only test in the procedure is for each word, "Check whether the CSS *letter-spacing* attribute was used to control spacing." Hence, we created a class called **C_8** (a subclass of class **TC_CSS** under class **Techniques**). Class **C_8** has two subclasses: **C_8_page** and **C_8_word**. Class **C_8_page** plays a similar role as **H_32_page** under **H_32** and **C_8_word** plays a similar role as **H_32_form** under **H_32**. Words satisfying this C8 technique should be set as inferred members of class **C_8_word**. This is achieved by the following Horn-like rule (also shown in Figure 8 in SWRL format):

$$HT\_page(?p) \wedge HT\_body(?b) \wedge HT\_heading(?h) \wedge in(?b,?p) \wedge in(?h,?b) \wedge HT\_word(?w) \wedge in(?w,?h) \wedge hasValue(?h,?hs) \wedge CSS\_heading(?ch) \wedge CSS\_letter\_spacing(?cls) \wedge in(?css,?p) \wedge hasValue(?ch,?chs) \wedge in(?cls,?ch) \wedge stringEqualIgnoreCase(?chs,?hs) \rightarrow C\_8\_word(?w) \quad (5)$$

The rule (5) can properly exclude the following situations: a) words that are not placed inside an html heading tag; b) words within html heading, but the heading is not defined in the CSS style section; c) words within heading defined also in the CSS style section, but the heading does not have *letter-spacing* attribute defined. With this rule, individuals for words that do not satisfy C8 technique will be excluded from class **C_8_word**.

Technique C8 requires "all" words in a web page to follow rule (5). Such a concept (universal quantification) is similar to that H32 requires "all" forms to follow rules (1)-(3). Therefore, we used the approach similar to how we handled the universal quantification of technique H32 by defining a class **C_8_page** under class **C_8**. The class **C_8_page** is defined as equivalent to "**HT_page** and (*pageHasWord* some **HT_word**) and (*pageHasWord_C8* **only C_8_word**)", as shown in Figure 8. Object property *pasHasWord_C8* is a sub property of *pasHasWord* with range defined as **C_8_word**. Hence, any **HT_page** individual containing *pageHasWord_C8* linking to only **C_8_word** will be assigned as an inferred member of class **C_8_page** by reasoners.

The HTML example listed in Figure 6 is realized in Protégé as shown in Figure 9 with the inference result showing that individuals w_01-w_04 do satisfy technique C8 and, therefore, should be (inferred) members of class **C_8_word**. Since the individual p_01 meets the definition of class **C_8_page**, it will be an inferred member of **C_8_page** as shown in Figure 10.

Since there are 437 techniques listed in WCAG 2.0 techniques, ideally, class **HT_page** will have 437 subclasses. They are either *primitive* classes such as **H_32_page** or *defined* classes such as **C_8_page**. All these 437

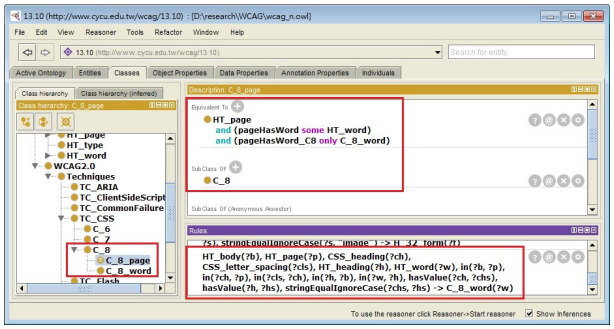


Figure 8. Class C_8_page and the rule for technique C8.

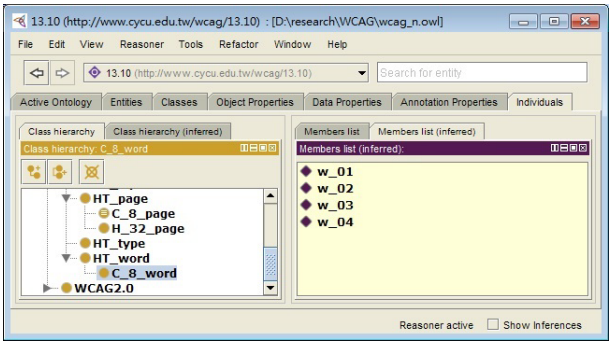


Figure 9. Reasoning result of class C_8_word.

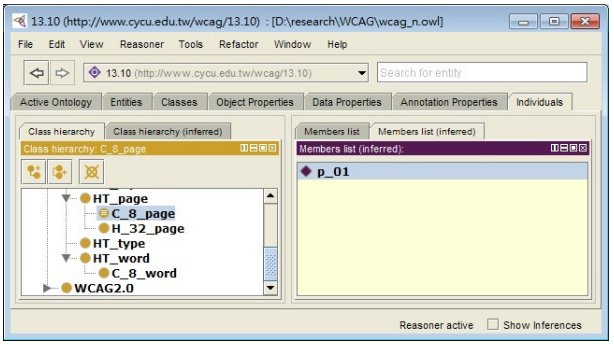


Figure 10. Reasoning result of class C_8_page.

classes will also be subclasses under **Techniques**. More precisely, they will be subclasses of subclasses of class **Techniques**. For example, the **H_32_page** is a subclass of class **H_32** and **C_8_page** is a subclass of class **C_8**, shown in Figures 7 and 8, respectively.

3.2.3. Modeling WCAG Success Criterion

With the proposed modeling method described in the previous subsections, SC can be handled by mapping sufficient and advisory techniques to rules. For the time being, we consider situations in a sufficient technique are mutually exclusive. Each situation and advisory technique will be translated to a single rule. A page that satisfies any rule of a SC will be considered satisfying that particular SC. According to this model, the criterion 3.2.2 shown in Figure 3 will have two rules as shown in Figure 11. Since technique G80 described in this criterion suggests the use of one of the five techniques listed below (H32, H84, FLASH4, PDF15, and SL10), a page *p* satisfies anyone of them will be considered satisfying technique G80. Individual *p* will be also an individual of class **G_80_page**. To fulfill criterion 3.2.2, a page must satisfy techniques G80, G13 and SCR19 (for the sufficient part) or technique G201 (for the advisory part).

3.3. Using WCAG Knowledge Base

Developing the proposed WCAG KB is a labor and expert intensive task requiring ontology engineers to work closely with WCAG experts. By carefully examining regulations (techniques and SC), the whole WCAG KB can be constructed.

The constructed result WCAG KB can be used and embedded into any application directly. However, if engineers wish to do so, then they need to have a certain level of understanding on ontology, logic, and KB. Engineers also to know how to manipulate individuals in the KB, trigger the reasoner, retrieve, and explain the result. A better approach to use the proposed WCAG KB is to add an application programming interface layer for developers. Details are explained in the next section.

$$
\begin{aligned}
&\text{H\_32\_page}(?p) && \rightarrow \text{G\_80\_page}(?p)\\
&\text{H\_84\_page}(?p) && \rightarrow \text{G\_80\_page}(?p)\\
&\text{PDF\_15\_page}(?p) && \rightarrow \text{G\_80\_page}(?p)\\
&\text{SL\_10\_page}(?p) && \rightarrow \text{G\_80\_page}(?p)\\
&\text{FLASH\_4\_page}(?p) && \rightarrow \text{G\_80\_page}(?p)\\
&\text{G\_80\_page}(?p) \wedge \text{G\_13\_page}(?p) \wedge\\
&\text{SCR\_19\_page}(?p) && \rightarrow \text{SC\_3\_2\_2}(?p)\\
&\text{G\_201\_page}(?p) && \rightarrow \text{SC\_3\_2\_2}(?p)
\end{aligned}
$$

Figure 11. Rules for criterion 3.2.2 (On Input).

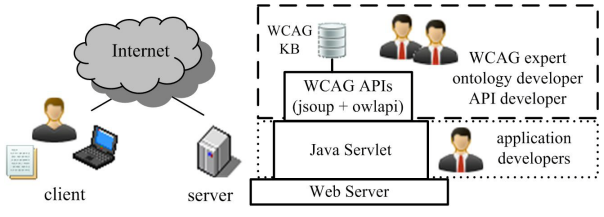


Figure 12. The system architecture of the web-based WCAG validation system.

4. Experiments

To illustrate how the proposed WCAG KB can be used, we developed a web-based WCAG validation system. The system was developed using the Java Plateform (Java Servlet), jsoup (HTML parser), OWL API (v3), and Apache web server [17-21]. The web-based application we built for this study realized the H32 technique described in previous sections.

The architecture of this validation system is shown in Figure 12. An additional layer called WCAG APIs is added between the WCAG KB and Java Servlet to hide the complexity of directly using WCAG KB as described in the previous subection. In this way, application developers only need to know which function interfaces to invoke without know anything about ontology, logic, and KB. The dashed-rectangle represents the WCAG KB and APIs, created by WCAG experts, ontology experts, and API developer. The dotted-rectangle represents the application layer created by application developers. When a client makes a request, the web server will initiate necessary Java Servlet objects and these objects invoke WCAG APIs with necessary parameters.

Figure 13 shows a piece of Java Servlet program to invoke the WCAG APIs. First, an object of class H32, from the WCAG API package, is created with necessary parameter retrieved from Servlet objects. The constructor of class H32 instantiates necessary KB individuals illustrated in Figure 5. Then by invoking function *do_validation*(), *print_result*(), and *clear*(), the system performs validation, prints result in html format, and releases allocated resources, respectively. The complexity of directly using WCAG KB is totally hidden behind the WCAG APIs.

Since the system was built for demonstration purpose, it validates only the technique H32 described in the Section 2 (Figure 1). The input text area shown in the Figure 14 takes a HTML file with the name *htmlstr* that will link to the code, *request. getParameter* ("*htmlstr*") shown

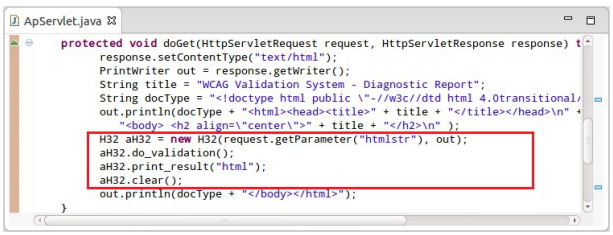


Figure 13. Sample code to invoke the WCAG APIs.

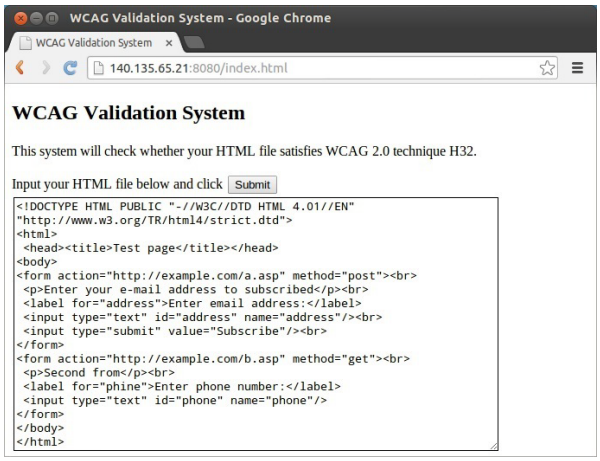


Figure 14. An invalid H32 HTML file.

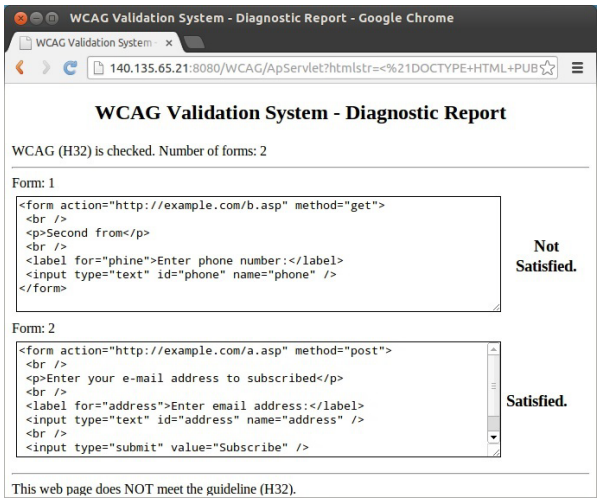


Figure 15. The validation result of Figure 14.

in Figure 13. Since the second form does not satisfy the regulation, the result "Not Satisfied." is printed aside the form. Because the order of forms is irrelevant, the WCAG APIs do not keep track of their order. In this particular case, the second form in the input HTML file appears on top of Figure 15.

If one changes the type of the second form of the input HTML file shown in Figure 14 from "*text*" to "*submit*" as shown in Figure 16, this particular input will meet the H32 regulation. The result is shown in Figure 17 and the bottom line shows "The web page meets the guideline (H32)."

This system can be easily modified into a batch system taking a large number of HTML files and validating them one by one. As aforementioned, to validate a number of HTML files by human experts is practically impossible, but such a task can be achieved swiftly and accurately with the proposed WCAG KB and the experimental system.

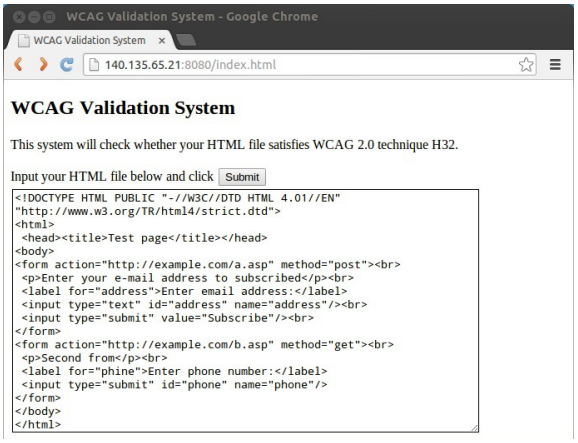


Figure 16. A valid H32 HTML file.

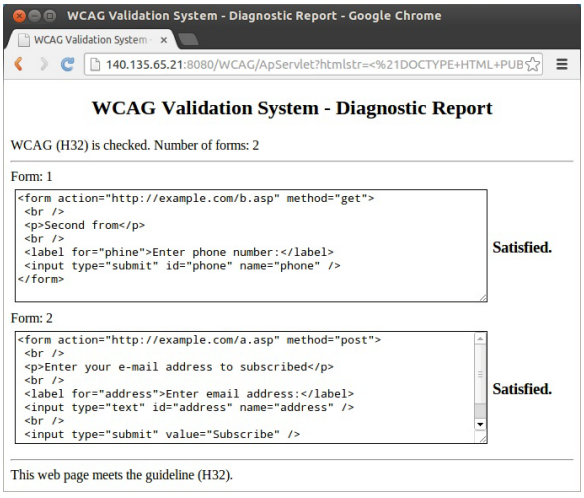


Figure 17. The validation result of Figure 16.

5. Conclusions

WCAG 2.0 is the most well-known specification for evaluating the accessibility of web contents. In this study, we presented a modeling method to translate the dictionary like documents to a computable knowledge base. We examined two major groups of techniques and used a technique in each group to illustrate the proposed model. We also demonstrated how to translate WCAG test procedures to SWRL rules. With the proposed knowledge base, OWL reasoners can properly generate the expected inferred results.

There are two major contributions of this study. First, we illustrate that the complicated WCAG 2.0 documents indeed can be represented as a knowledge base. In other words, we show that the knowledge described in WCAG documents can be preserved in a knowledge base, a computable resource. Secondly, the proposed knowledge base is shareable in nature. Although creating such a reusable resource is a labor and expert intensive task, it is still a one-time investigation.

In this study, we also developed a web-based WCAG

validation system to illustrate how to use the proposed WCAG knowledge base. The knowledge base is wrapped in a set of application programmer interfaces. With these interfaces, application programmers can develop a system without having any understanding of ontology, logic, and knowledge base behind them.

Several topics related to this study merit further investigation. First, the HTML tags and attributes have more complicated hierarchy, relationships, and constraints compared to the model proposed in this paper. For instance, Tag *input* has attribute *type* that is not a valid attribute for tags like <p>. The proposed ontology places tags and attributes at the same level under class **HTML** because a) the proposed method functions well for our purpose in creating WCAG knowledge base, and b) the main focus of this study is not to develop ontology for HTML. The class **CSS** has a similar situation that CSS styles have complicated hierarchy, relationships, and constraints, e.g. *letter-spacing* can only be defined inside headings and not the other way round. Although an interesting research topic, developing knowledge bases for HTML and CSS is beyond the scope of this study. Secondly, there are 60 HTML and XHTML techniques and 22 CSS techniques defined in WCAG 2.0. Readers with resources can translate all these 82 techniques into an integrated ontology. Third, some of the WCAG 2.0 techniques require human inspection. For instance, technique C27 "Making the DOM order match the visual order" must be checked only by human experts visually. Incorporating techniques requiring human inspection into the proposed model are also a topic worth exploring.

REFERENCES

[1] The W3C, "Web Content Accessibility Guidelines (WCAG) 2.0," 2013. http://www.w3.org/TR/WCAG

[2] The W3C, "Techniques for WCAG 2.0," 2013. http://www.w3.org/TR/WCAG20-TECHS

[3] R. Stevens, C. A. Goble and S. Bechhofer, "Ontology-Based Knowledge Representation for Bioinformatics," *Brief Bioinform*, Vol. 1, No. 4, 2000, pp. 398-414. http://dx.doi.org/10.1093/bib/1.4.398

[4] Y. L. Chi, "Rule-Based Ontological Knowledge Base for Monitoring Partners across Supply Networks," *Expert Systems with Applications*, Vol. 37, No. 2, 2010, pp. 1400-1407. http://dx.doi.org/10.1016/j.eswa.2009.06.097

[5] US National Library of Medicine, "UMLS: Unified Medical Language System," 2013. http://www.nlm.nih.gov/research/umls/

[6] International Health Terminology Standards Development Organization, "SNOMED Clinical Terms," 2013. http://www.snomed.org

[7] G. A. Miller, "WordNet: A Lexical Database for English," *Communications of the ACM*, Vol. 38, No. 11, 1995, pp. 39-41. http://dx.doi.org/10.1145/219717.219748

[8] C. Fellbaum, "WordNet: An Electronic Lexical Database," MIT Press, Cambridge, 1997.

[9] Development and Evaluation Commission, Executive Yuan, Taiwan, "Freego," 2009. http://www.webguide.nat.gov.tw/wSite/ct?xItem=36316&ctNode=14521&mp=1

[10] The Merriam-Webster Dictionary, 2013. http://www.merriam-webster.com/

[11] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, Vol. 5, No. 2, 1993, pp. 199-220. http://dx.doi.org/10.1006/knac.1993.1008

[12] N. Guarino, "Understanding, Building, and Using Ontologies," *International Journal of Human Computer Studies*, Vol. 46, No. 2-3, 1997, pp. 293-310. http://dx.doi.org/10.1006/ijhc.1996.0091

[13] N. F. Noy and C. D. Hafner, "The State of the Art in Ontology Design: A Survey and Comparative Review," *AI Magazine*, Vol. 18, No. 3, 1997, pp. 53-74.

[14] The W3C, "OWL Web Ontology Language Reference," 2013. http://www.w3.org/TR/owl-ref

[15] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer and D. Tsarkov, "Owl Rules: A Proposal and Prototype Implementation," *Journal of Web Semantics*, Vol. 3, No. 1, 2005, pp. 23-40. http://dx.doi.org/10.1016/j.websem.2005.05.003

[16] N. F. Noy, R. W. Fergerson and M. A. Musen, "The Knowledge Model of Protege-2000: Combining Interoperability and Flexibility," *Proceedings of* 12*th International Conference of Knowledge Acquisition, Modeling and Management*, Juan-les-Pins, 2-6 October 2001, pp. 17-32.

[17] Pellet: OWL 2 Reasoner for Java, 2013. http://clarkparsia.com/pellet/

[18] Java Platform, 2013. http://www.oracle.com/technetwork/java/index.html

[19] The OWL API, 2013. http://owlapi.sourceforge.net/

[20] The Jsoup: Java HTML Parser, 2013. http://jsoup.org

[21] The Apache HTTP Server Project, 2013. http://httpd.apache.org