A FLEXIBLE APPROACH TO SEMI-AUTOMATIC ACCESSIBILITY EVALUATION

Mireille Blay-Fornarino, Karima Boudaoud, Anne-Marie Dery-Pinna,
Rainbow project, I3S
930, route des Colles / B.P 145
06903 Sophia Antipolis cedex, France

Charles McCathieNevile
World Wide Web Consortium (W3C)
2004 route des lucioles / BP 93
06902 Sophia Antipolis cedex, France

ABSTRACT
Experience in preparing Web content for people with visual disabilities shows that automatic testing of accessibility is clearly insufficient to meet real-world demands, and that human reviews are needed.
At the same time automated or semi-automated testing is an important part of reducing the demands on a human evaluator. In this paper we discuss tools we have developed with a design approach that addresses these points. As a consequence, we propose a flexible tool architecture, based on the integration of evaluation and repair functions.

KEYWORDS
Accessibility, architecture model, human review, automatic evaluation, EARL/RDF

1. INTRODUCTION
Our aim is to develop or repair HTML pages to make their content accessible to people with visual disabilities. The World Wide Web Consortium (W3C) [W3C] has defined guidelines (WCAG) explaining how to make Web content accessible to people with disabilities [WCAG]. The guidelines are intended for all Web content developers. But as Web site design is a creative activity, and accessibility is not always easy to achieve, applying these guidelines currently requires experience and a good knowledge of Web technology.

In our work we have applied the Human Computer Interface principle of co-operative Interface evaluation by users [Monk et al.]. In the evaluation method, we have considered two kinds of users: Web site developers (including people expert in their own subject but with little Web expertise, who are required to put content online) and people with vision impairments.

From the Web developers' point of view, the main constraints are to obtain an accessible Web site without a minimum of additional work, and to maintain or improve the existing interface quality and aesthetics of the Web site.
A visually-impaired user may use adaptive strategies to access the Web site.

Our experimentation and evaluation have identified several drawbacks of existing tools during the steps of creation, evaluation and repair of HTML pages:

**Creation of HTML pages:** for efficiency, or lack of familiarity with HTML code, developers generally use visually-oriented page editors. Some of these tools generate such poor quality code that accessibility cannot be reached. Few editors provide effective support for accessibility, and in particular for intuitive separation of presentation style from content structure.

**Verification of HTML pages:** It is rarely feasible to test accessibility with an appropriate group of actual users, so different tools have been created to verify accessibility, following the WCAG guidelines. Testing
Accessibility with automatic tools such as Bobby [Bobby] or Cynthia [Cynthia] is fast and simple. But automatic evaluation doesn't identify all accessibility issues [CynthiaBobby], so human reviews are necessary to verify that the page meets subjectively evaluated requirements such as: “is there clear colour contrast?” or “is the language used as simple as it could be?”

**Repair of HTML pages:** The first problem with repair of accessibility problems is that some common evaluation tools do not provide any integration of repair functionality, and the second is that almost all of them require working with the underlying HTML code, while many website authors are not familiar with HTML coding.

Some particular difficulties authors encounter include knowing how to separate presentation from structure, or what presentations are appropriate and useful for different types of users.

The following table (Table 1.) summarises some of the features which we expect in a flexible tool for evaluation and repair, and which of those features are available or lacking in some commonly used tools.

<table>
<thead>
<tr>
<th>Tool / Feature</th>
<th>AccessValet</th>
<th>AccRepair</th>
<th>A-prompt</th>
<th>Bobby</th>
<th>Lift for DreamWeaver</th>
<th>WaiDoctor</th>
<th>WAINU with Annoteamez</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include Human testing (1)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Integrated in Authoring Tool</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Integrated Repair</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WYSIWYG repair</td>
<td>Not</td>
<td>Applicable</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatically extract presentation</td>
<td>No</td>
<td>No</td>
<td>Partial</td>
<td>No</td>
<td>Partial</td>
<td>In progress</td>
<td>In progress</td>
</tr>
<tr>
<td>Record results</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Not yet(2)</td>
<td>Yes</td>
</tr>
<tr>
<td>Test scripts / backends</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Use image context</td>
<td>No</td>
<td>No</td>
<td>Partial</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Integrate other tools’ results</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Not yet(2)</td>
<td>Yes</td>
</tr>
<tr>
<td>Extensible</td>
<td>No</td>
<td>Yes (3)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes (4)</td>
<td>Yes (4)</td>
</tr>
</tbody>
</table>

Table 1. Features required for an efficient evaluation and repair tool

Notes:
1. Certain testing functions only.
2. The open source model used means functions developed in one of Waidoctor [Djabri2003], Wainu [Djabri2002] or Annoteamez [Queinnec and Roure] can be reintegrated into the other tools.
3. AccVerify™ [AccVerify] and AccRepair™ [AccRepair] allow for the addition of new test rules – essentially text patterns to be matched, along with some conditional rules.
4. Wainu and Annoteamez are open source, designed to allow any developer to extend the software in any way at all. This may include developing completely new functionalities, such as Annoteamez itself, or new tools such as the current commercial development of Waidoctor.

After our early experience, we started developing tools to help evaluate and improve the accessibility of Web pages. In this paper, we will not detail the features which are similar to those supported by existing tools but we will focus on how to maximise the value of human intervention during the evaluation and repair of Web pages, by integrating:

- presentation, warnings and code correction, and
- recording results of the evaluations already done on a specific Web page
  in an open source tool developed using an extensible architecture, and available for other developers to integrate functionalities as desired.

This paper is divided into four parts. Following this introduction, we present the main functionalities offered by the tools Waidoctor, Wainu and Annoteamez developed in our team. In the third part, we describe
briefly the new model on which we are working. Finally, we draw some conclusions from this work and give some perspectives on future directions.

2. TOWARDS AN INTEGRATED TOOL

In this part, we focus on code correction and particularly on three main points: 1) designing the user interface, 2) extracting structures from presentation, 3) recording evaluation results.

2.1 User interface based on control point correction

Depending on how the developer prefers correcting errors, he could interact through a user interface at different times and with different techniques for each control point. For example,

There is an automatic evaluation carried out in one step, which is the case with the tool Waidoctor. The developer can then fix all the problems at once, or can fix all problems of a specific type at once. In order to make correction more efficient, to avoid repeating the same completion when a reference (link or image) is used several times in the page, the tool proposes to the developer to automatically correct the corresponding references elsewhere in the page. Or,

During the evaluation, every time a problem is encountered the author can repair it, which is the case with the tool Wainu (see section 2.3).

One of the major benefits of using this approach is that it allows the user to be sure that the checking has been rigorously applied. It is relatively easy for a user to examine the first few instances of a feature in an HTML page and conclude that the same process has been followed throughout. And it is very difficult for a user to verify several thousand instances of the same piece of code without introducing errors into the evaluation.

By contrast an automated process is ideal for finding the one case in five, or in five hundred, that is different, and ensuring it is tested as rigorously as the first case would be.

As well as the structure and code views present in Waidoctor the developer can preview the page in a browser, updated as the code is changed in the tool. Another view is under development, which will provide an audio version of the page as it might be rendered by a screen reader.

2.2 Extracting structures from presentation

The WCAG says: "The content of a document refers to what it says to the user through natural language, images, sounds, movies, animations, etc. The structure of a document is how it is organized logically (e.g., by chapter, with an introduction and table of contents, etc.)."

The way in which content is presented helps to highlight the essential information. However whether a particular presentation is really helpful depends on the final user. For users with disabilities, it is natural to change the presentation according to users preferences, which may include using voice and navigation through keyboard arrows, or needing to use large fonts and high-contrast colours, etc.

This is very much easier if the structure rules have been defined, and the presentation layer can be removed, or a new one applied. In HTML or XML this is best achieved by the use of Cascading Style Sheets. A similar approach has been available in Word processors such as Microsoft Word, or formats such as Postscript and TeX for many years. The difference with the Web is that it is relatively easy for a user to define their own presentation format, and relatively common for people to use different devices and therefore need to do so.

How to separate presentation from contents and structure is often not evident to developers. Indeed, many people use presentational effects to specify the structure, such as directly placing numbers, bold and large font, centred text and so on in the body of the text instead of using structured lists and headings.

Repairing this problem in an HTML page is therefore often difficult, boring and error-prone. When the page is consistently styled, it can be achieved by code substitution and style file building. In Waidoctor, we are currently working on determining the structure in an automated way, recognising common style features and analysing them to generate a proposed structure of the document.
2.3 Recording evaluation results

Many tools have been developed to evaluate automatically the accessibility of Web sites and Web pages (WAI-ERT), such as Bobby, Wainu, Waidoctor or Waizard [Giboyau et al.]. However, we have seen that some evaluations must be done manually because they require human judgement. An example is the "alt" attribute of an image. If this attribute is not specified, a tool such as Wainu can detect that automatically and determine that the page doesn’t meet the WCAG. But if this attribute exists, a manual evaluation is needed to verify that the text equivalent given corresponds to the actual image.

When evaluating a Web page the same tests can be needed done many times. This means, in the case of the previous example, asking the human evaluator if the same "alt" attribute correspond to the same image each time a page is evaluated. It is more efficient to record the results of evaluations to avoid repeating the same questions.

The aim of Annoteamez tool is to integrate the human evaluation with the automatic evaluation of Wainu in order to minimize the need for human intervention. So, Annoteamez has two main features: 1) to permit to the developer to report or correct errors, through a user interface, for each checkpoint; 2) to record the results of evaluations already done on a web page.

For evaluation reports, Annoteamez uses the W3C specification EARL [EARL] (currently in development) in order to be interoperable with the results of other tools such as AccessValet [AccessValet], or AccVerify™ [AccVerify], whose results are available in EARL. So, in addition to integrating human evaluation results with results obtained automatically by Wainu, Annoteamez could integrate the results of any other tools using EARL.

3. A MODEL FOR EVALUATION TOOLS

3.1 Design considerations

As we have seen, automatic testing of accessibility is clearly insufficient to meet real-world demands of users with disabilities, in particular failing many of the "last mile" tests – those that make a real difference to the ease and efficacy of the user experience, rather than those that simply enable the user's assistive technology to provide some basic access to content in some form. Managing information gathering in manual parts of accessibility evaluation, and maximising the effective use of that information is therefore extremely important.

Automated or semi-automated testing is nevertheless an important part of reducing the demands on an human evaluator to the essential work they are required to do. By increasing the accuracy of the results, and the benefit for end users, and efficiently managing the interaction to maximise the returns, it is easier for the author to justify the additional time taken to perform a careful manual evaluation of accessibility.

The parallel development of an open-source library of functions which can be integrated into various tools and an open-source evaluation tool is an important part of this model. It provides a framework which allows many developers to rapidly develop new functions applicable in their particular area of expertise, and by integrating them in the existing tool, make them available to a wide range of users. This framework goes beyond the simple ability to write a new testing rule.

For example the use of an internationally developed open standard implemented in several existing systems for recording results allows for easy integration of results from new evaluation tools, or the integration of tools into new systems. Because there are several different tools which produce EARL (Access Valet, AccRepair, MUTAT [MUTAT]) and because of its flexibility, it is relatively easy to integrate the result of a new test into an existing one. For example, a new function library that could test automatically for clear and simple writing (WCAG checkpoint 14.1) could produce EARL to state that this checkpoint had been met. Wainu can use that result, and therefore would not need to ask for a human evaluation of the writing style.
3.2 Current development

3.2.1 Extracting Structure

A tool which can automatically extract a structure and present it to the user for verification can help to teach a user how to recognise the structure they are implying with presentation. A good tool will allow the user to do this easily. This kind of help has long been available in tools such as Amaya and Microsoft Word, but because they currently lack an accessibility evaluation function many users do not know about, let alone learn to use, this functionality. The open source libraries developed can be integrated into complete editors, to allow the developer to use the same natural interface for repair that they use for creating content.

3.2.2 Integrating tool results

The results produced by the Annoteamez extensions to Wainu are interoperable with a simple experimental interview-based tool from W3C. The tools are being developed to use the same mechanism for storage of results – an Annotea-based store, which will allow each toolset to automatically use the results generated by the other.

3.3 Future development

The model proposed, along with existing gaps in functionality that is widely available have suggested several important areas of development.

3.3.1 Content Management Systems

More and more websites are produced not as single pages, but as collections of information in a content management system which then uses templates to assemble pages. It is therefore interesting to develop strategies for managing content in such a system, analysing the templates, the content used to populate them, and the code that handles the process of populating a template to produce a page. This work will rely in part on the W3C’s Authoring Tool Accessibility Guidelines [ATAG]. An important part of this approach is to automatically link a part of the content of a result page with the part of the script or code which generates that particular section.

3.3.2 Contextual information

For example, the meaning of a logo could be reused each time it is used. Some information needs to be explode in non contextual information (this image shows the Bellet team) and contextual (this image shows a team working together on a boat: it illustrates collaboration).

3.3.3 Trust management

The use of EARL requires that the name of the evaluator, and whether it is a tool or a person, are recorded. This opens the possibility of extending the interface to provide simple trust management – users could decide whether they trust a particular person or tool over other results available. This possibility reduces the risks that would normally be associated with the incorporation of results from a tool which performs some tests reliably but others very unreliably. Third parties (such as WAI, or organisations representing people with particular needs) would be able to provide trust profiles that users could simply adopt, while experienced users may decide to manage their own trust profiles.

4. CONCLUSIONS

We have presented the functionalities of the tools developed in our team. Based on our results using these tools for evaluation we have proposed a model for accessibility. In addition to integrating the functionality of the three tools, we are developing new functions to:

- develop strategies to deal with content produced automatically
- take into account the contextual aspect of images;
• integrate evaluation results of other tools.

The framework has incorporated new design features that provide for more efficient interoperability with existing tools and tools currently under development.

Importantly, we provide to the accessibility tool community at large a basis for experimentation which is simple enough to be useful in ongoing research and development work, powerful enough to provide a realistic testbed system, and with a sufficiently open and flexible architecture to develop at the leading edge of automated or assisted accessibility evaluation.

ACKNOWLEDGEMENTS

We thank Y. Giboyau, G. Nirelep, P. Seurat, M. Simkovitch, V. Djabri, P. Quinneec, R. Roure, of ESSI school, the rainbow team, Daniel Dardailler of W3C, and AbsoluteVox company who helped us to experiment and to develop the different paradigms presented in this paper.

REFERENCES

[ATAG] Authoring Tool Accessibility Guidelines – http://www.w3.org/TR/ATAG10
[EARL] EARL – http://www.w3.org/TR/EARL10
[MUTAT] MUTAT – http://www.w3.org/QA/Tools/MUTAT
[W3C] W3C – http://www.w3.org
[WCAG] Web Content Accessibility Guidelines 1.0 – http://www.w3.org/TR/WCAG10