AO -WAD: A Supporting Tool to Aspect-Oriented Web Accessibility Design

Rafaela Mazalu[1,2], Fabián Huenuman, Adriana Martin[2,3], Alejandra Cechich[2]

[1]Consejo Nacional de Investigaciones Científicas y Técnicas, Neuquén, Argentina
[2]GIISCO, Facultad de Informática, Universidad Nacional del Comahue, Neuquén, Argentina
[3]Unidad Académica Caleta Olivia, Universidad Nacional de la Patagonia Austral, Caleta Olivia, Santa Cruz, Argentina
{rafaelamazalu, fhuenuman, adrianaelba.martin, acechich} @gmail.com

Abstract. There are a number of tools and proposals to help developers assess Accessibility of Web applications; however looking from the designer perspective, there is no such a similar situation. In this paper, we present a supporting tool that helps users model Web Accessibility by moving from abstract to concrete architectural views using aspect-orientation. Thus, the designers and developers produce accessible interfaces. The proposed tool is based on an approach that takes advantages of modeling Accessibility as an aspect-oriented concern, which is independently treated but related to architectural pieces.

Keywords. Web Accessibility; User Interface Models; Web Engineering; Aspect-Oriented Design; Design Tool.

1 Introduction

Web Accessibility means universal access on the Web, regardless the kind of hardware, software, network platform, language, culture, geographic location and user's capabilities. We have worked for a while on Accessibility [6] [7] and particularly on Accessibility design at early stages of Web applications development process [8] [9][10]. Particularly, we have applied aspect-orientation associated with the WCAG 1.0 as the reference guideline, and we gathered some experiences on the field. Since the WCAG has two documents (1.0 and 2.0), it is important to make clear at this point that we based our work on the WCAG 1.0, which since 1999 is keeping its value as the benchmark for other valuable Accessibility standards [12][15], while the ongoing migration process to WCAG 2.0 [18] [19] is completed worldwide.

In this paper, we introduce a supporting tool based on the approach presented in [9], which includes Accessibility concerns systematically within a methodology for Web application development. Our approach builds upon OOHDM [14] and includes Accessibility concerns in the development life-cycle. Since designing accessible Web applications involves the analysis of different interests, we propose the use of Aspect-Oriented Software Development (AOSD) design principles and Web Content Accessibility Guidelines 1.0 (WCAG 1.0) to support the construction of accessible user

interfaces. Thus, we ensure the handling of non-functional, generic and cross cutting characteristics of the Accessibility concerns naturally.

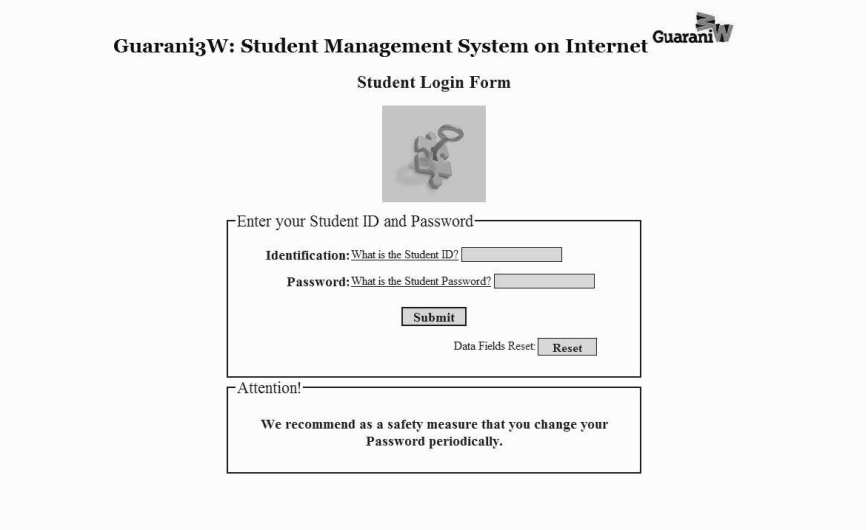

Fig. 1. A Student's Sign-in Web Page

As a simple example to illustrate the approach's ideas underlying the proposed tool, let us suppose a typical Web page whose purpose is a student's sign-in aiming at his/her identification at the Argentine university system. As shown in the Figure 1, the Web page for the student's sign-in provides a user interface composed of HyperText Markup Language (HTML) elements, like labels and text fields. To ensure an accessible interaction these HTML elements must fulfill some Accessibility requirements, which crosscut the same software artifact (the Web page for the student's sign-in). For example, and as we will see in detail later, at the presentation level an HTML label element is a basic layout Accessibility requirement for many others HTML elements. Since a Web page for student's sing-in requires at least two text field elements (for the student's ID and password respectively), the presence of their respective label elements must be tested. So, to ensure an accessible interaction on behalf of the student, this layout requirement must crosscut the same software artifact (the Web page) more than once, according to the number of text field elements included in the presentation. Additionally, it is highly important to consider the positioning of the label element with respect to a text field element; this technological requirement for "until user agents" also crosscuts the Web page. Clearly this kind of behavior perfectly fits the typical crosscutting[1] symptoms --i.e. the "scattering" and "tangling" problems[2]

[1] "Croscutting" is a term used for certain type of functionality whose behavior causes code spreading and intermixing through layer and tiers of an application which is affected in a loss of modularity in their classes. Quality requirements (such as Accessibility), exception handling, validation and login managements are all examples of this common functionality, which is usually described as "crosscutting concerns" and should be centralized in one location in the code where possible.

that motivate the main AOSD principles. The aim of this work is to provide a supporting tool, following the spirit of our aspect-oriented approach for Web Accessibility design, to assist designers and developers to produce accessible interfaces.

The rest of the paper is structured as follows: in Section 2, we review some key background concepts and tools described in [9] that are used by our design approach and in consequence by the proposed tool. We also offer a brief overview of our approach using a real application example as a case study to illustrate the context of use for the tool. In Section 3, we introduce the supporting tool, describing its use and architecture. Finally in Section 4 we conclude and present some further work.

2 Over Overview of the AO-WAD Approach

Firstly, we briefly review two main background concepts applied by our approach [9] and on which underlie the functioning of the proposed tool.

Accessibility through UIDs with Integration Points. A User Interaction Diagram (UID) [16] is diagrammatic modeling technique focusing exclusively on the information exchange between the application and the user. With the traditional perspective given by techniques like [1] [2] in mind, our approach introduces the concept of UIDs's integration points [6] to model the Accessibility concerns of a user-system interaction. Particularly, the approach defines two kinds of UIDs integration points:

- User- UID Interaction (U-UI) integration point. This is an integration point for Accessibility at UID interaction level --i.e. to propitiate an accessible communication and information exchange between the user and a particular interaction of a UID.
- User- UID Interaction's component (U-UIc) integration point. This is an integration point for Accessibility at UID interaction's component level --i.e. to propitiate an accessible communication and information exchange between the user and a particular UID interaction's component.

These integration points with different granularity provide two alternatives for evaluating Accessibility during the interaction between the user and the system. Figure 2 shows the resultant UID, corresponding to the use case "Sign-in a student given the student's ID and password" (introduced in Section 1 by Figure 1), by applying our integration points technique. Notice that all the students (including those with disabilities) will need to interact with this online sign-in Web page. The figure shows two integration points at UID interaction <1> representing the student's sign-in user-system interaction to consider, from the beginning, the Accessibility requirements that ensure the access for all the students.

[2] "Scattering" and "Tangling" symptoms are typical cases of "crosscutting concerns" and they often go together, even though they are different concepts. A concern is "scattered" over a class if it is spread out rather than localized while a concern is "tangled" when there is code pertaining to the two concerns intermixed in the same class (usually in a same method).

Basically, the UID with integration points notation prescribes the inclusion of a cloud for every UID interaction or UID interaction's component where Accessibility is essential to the user's task completeness. The first cloud establishes the <1.1> integration point to ensure that the semantics of the logo image is correctly transmitted; while the second cloud establishes the <1.2> integration point to ensure an accessible form for user identification.

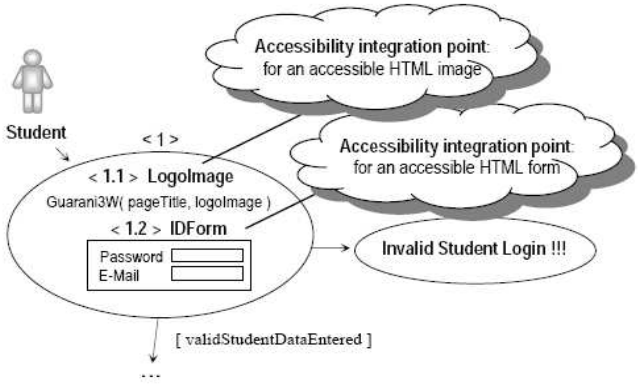

Fig. 2. UID with Accessibility Integration Points: Sign-in a Student given the Student's ID and Password

SIG Template for Accessibility. After specifying the Accessibility integration points of the UIDs, our approach proposes the development of SIG diagrams for WCAG 1.0 Accessibility requirements [6]. Figure 3 shows our SIG template conceptual tool, where the Accessibility softgoal denoted with the nomenclature Accessibility [UID integration point] is the root of the tree. The kind of the UID integration point is highlighted into the root light cloud and related to a particular UID interaction or UID interaction's component number. From the root node identifies two initial branches: (i) The user technology support, and (ii) the user layout support.

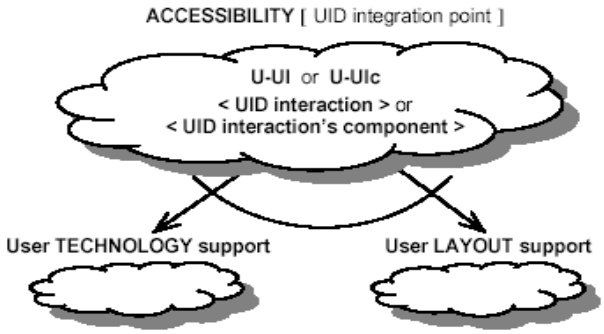

Fig. 3. SIG template for Accessibility

For example, returning to Figure 2, it it shows the Accessibility softgoal for the interaction's components <1.1> LogoImage and <1.2> IDForm to guarantee accessible

image and text input fields for all the students by defining two User-UID Interaction's components (U-UIc) integration points at UID interaction <1>. Finally, to instantiate the SIG template for ensuring Accessibility concerns (shown in Figure 3) the approach works with the W3C-WAI WCAG 1.0 guidelines [17] and establishes association tables for groups of related HTML elements. Basically, these association tables have the tasks of linking each abstract interface element present at a user interface model (ontology concepts from an Abstract Widget Ontology [14]) with their respective concrete HTML elements, and with the Accessibility concerns prescribed for those elements by the WCAG 1.0 checkpoints.

Our Process in a Nutshell. As highlighted in Figure 4(1), the process manages Web application requirements looking for those that involve Accessibility needs. This is because it is at the user's interface level where Accessibility barriers finally show, so we are particularly interested in discovering Accessibility requirements at the user interface design. Then, as shown in Figure 4(2), we propose an early capture of Accessibility concrete concerns by developing two kinds of diagrams: the UID with Accessibility integration points and the Softgoal Interdependency Graph (SIG) template for WCAG 1.0 Accessibility requirements, as shown in Figure 4(2.1) and (2.2) respectively.

The Accessibility knowledge captured and organized by SIG diagrams at early stages aids designers making decisions through the abstract interface model, as shown in Figure 4(3.1). As we can see in Figure 4, at this point is where the proposed tool get involved in the process helping to concrete stages 3 and 4. The purpose here is to find out how WCAG 1.0 Accessibility requirements "crosscut" interface widgets required for an IDForm. Since applying the required WCAG 1.0 checkpoints to be satisfied at the user interface causes typical crosscutting symptoms –i.e. "scattering" and "tangling" problems. A detailed discussion of these Accessibility aspects can be found in [9].

3 A Supporting Tool for AO-WAD

To accomplish with its main purpose the tool must deal with the concepts previously described, such SIG diagram, association tables and abstract user interface. Also, the tool should be at the user fingertips, i.e. the tool should be part of the user's development environment. To solve the second issue, the tool was developed as an Eclipse plug-in, integrating an XML editor with the views needed to inform the user about missing information (tags and attributes) required for an Accessible interface, and also providing the option to fix these missing information.

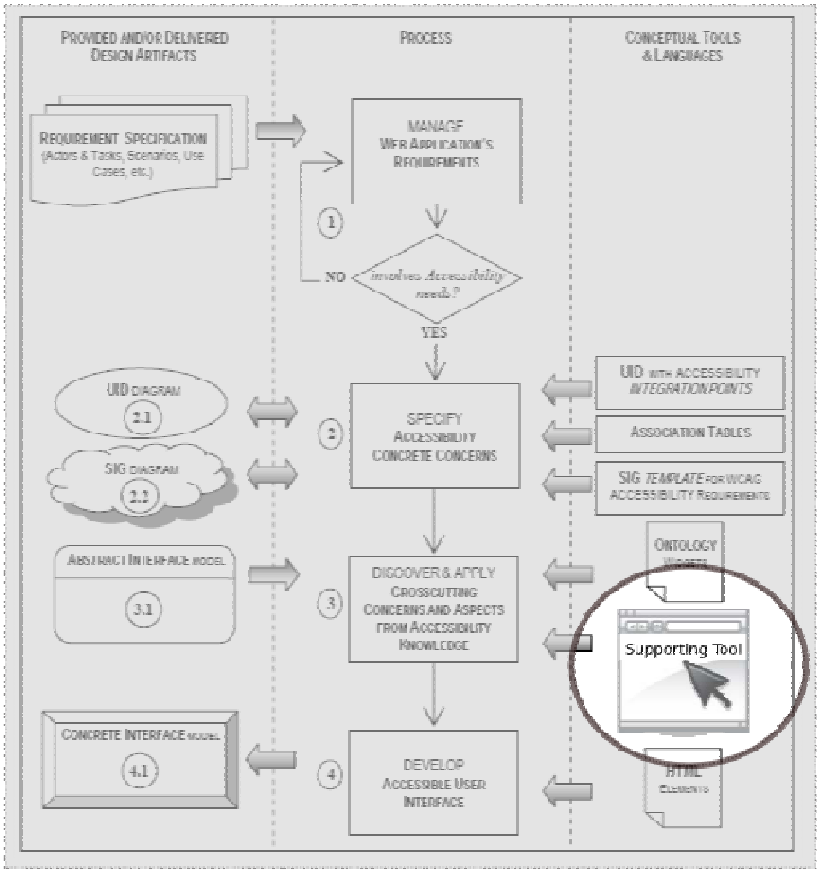

Fig. 4. Overview of our approach

3.1 Insights of the tool

From the user point of view the interaction with the tool follows an "open-save-close" cycle on a document, specifically, the developer designs an abstract interface for a web page by editing and saving changes in an XML-based document, also this methodology is known as document-centered work schema. For this reason, one of the main components of the user interface is the XML editor, which is complemented with the view *WCAConsole* to show and resolve the non-commitment to the Accessibility guidelines. Figure 5 shows these two components integrated in the Eclipse platform.

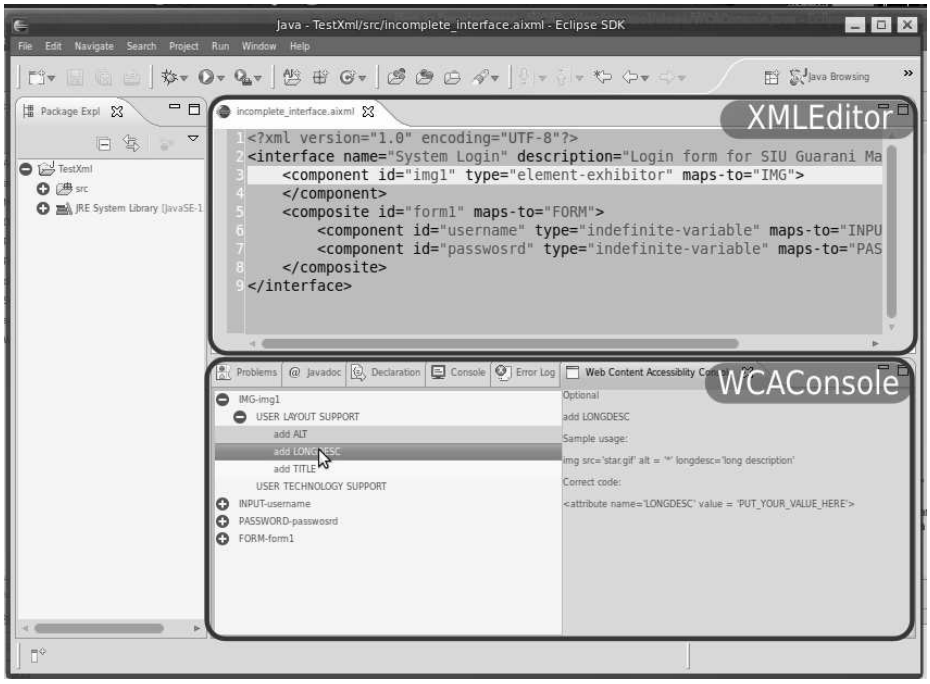

Fig. 5. The components integrated in the Eclipse platform

In the figure, the XML editor is shown in the upper box and is used by the developer to edit the abstract interface model. At the moment of saving the changes, the analysis of the structure and commitment to the Accessibility guides is started. Then, the analysis result is shown in a structured manner using the view *WCAConsole*. It is composed of two other components, a tree view in which, for every tag present in the abstract user interface, the missing or erroneous attributes are shown. Also, the view shows related tags that should be in the abstract user interface model. The other component is a read-only description field that shows, for each selected item, information as described below:

- *Attribute/Tag condition (Required/Optional)*: Indicates whether the tag or attribute is mandatory for commitment to the guidelines.
- *Action (Add/Remove)*: Indicates the action to perform with the tag or attribute, if it should be added to the abstract interface or removed.
- *Sample usage*: An example about the usage of the suggested tag or attribute.
- *Correct code*: Shows the code necessary to insert in the abstract interface model to commit to the Accessibility guidelines.

3.2 Basic Architecture

Figure 6 shows the main components of the tool's architecture which has three main layers: (1) Object Storage, (2) Core, and (3) Presentation. The Presentation layer, as

the name implies, represents the user interface, in this case, a designer or developer. The main classes in this layer are:

- *AccessibilityTool*, which represents the XML editor.
- *InterfaceParser*, which includes the functionality of identifying and highlighting syntax errors.
- *WCAConsole*, which provides functionality to show the non-commitment to the WCAG in a structured way. The name of this view stands for Web Content Accessibility Console, as a general view to contain all the Accessibility issues.

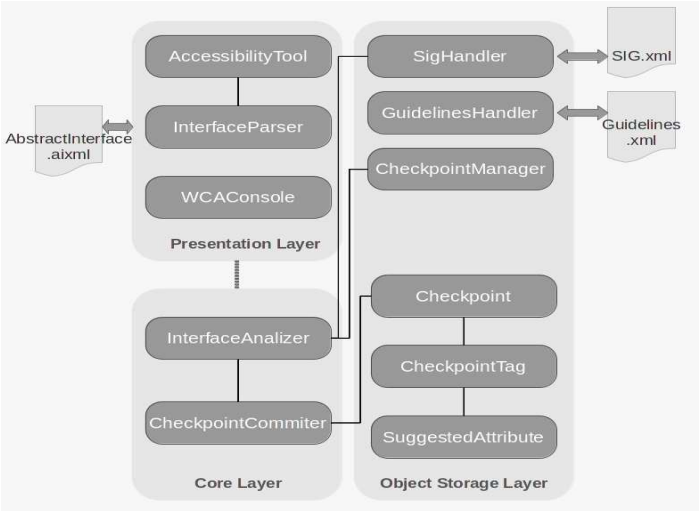

Fig. 6. Main Components of Our Supporting Tool's

The *Object Storage* layer represents an abstraction for the different underlying resource structures. Then, requests for information about checkpoint, present in the SIG structure or in the database, are solved using the services of this layer. The main classes for the layer are:

- *SIGHandler*, which provides the necessary functionality to access the contained information in SIG structure --i.e. the checkpoints to commit for a specified tag present in the abstract user interface.
- *GuidelinesHandler*, which as the previous class, provides the needed functionality to access the contained information in the Guidelines file.
- *CheckpointManager*, which provides the needed functionality to access information of different checkpoints. This class uses *CheckpointManager* to retrieve information about a checkpoint from the database file and maintain a pool of previously retrieved checkpoints.
- *Checkpoint, CheckpointTag, SuggestedAttribute*, these classes represent the models for access information about the element that each one represents. Specifically,

SuggestedAttribute represents an attribute which needs to be added or deleted in a tag (*CheckpointTag*) to meet a specific *Checkpoint*.

Finally, the Core layer includes those classes that play a central role for the tool's functionality. Those classes are:

- *CheckpointCommiter*, whose functionality includes the analysis and determination of commitment of tag to the WCAG. Also, it provides the functionality to generate the element code (tag or attribute) to fix the non-commitment.
- *InterfaceAnalizer*, which provides the functionality of coordination for the analysis of the abstract interface model. This class has an aspect-based implementation (AspectJ), which is the central feature that will allow the completion of the analysis in a transparent manner as described below.

Particularly, in Figure 6, we focus on the Presentation layer, which is isolated from the other layers and it is only related to the Core layer by a dotted line, meaning that there is no straight interaction between these two layers. Thus, the interaction between these two layers, which includes reading and analyzing the abstract interface (XML file) under treatment, takes place in a transparent manner. To reproduce this behavior, the tool uses the Observer pattern and their classes Subject and Observer; each instance of the Subject class maintains a list of instances of the Observer class which are notified of the changes that occur in their respective instance of the Subject class. Applying these design concepts, the *Accessibility Tool* class plays the role of Subject, while the *InterfaceAnalizer* class plays the role of Observer. Then, the update notification is implemented by the aspects environment (AspectJ). Thus, when the developer saves the XML document edited for the abstract interface model, this automatically triggers this aspect-oriented functionality which is not explicitly invoked by some element of the Presentation layer. As shown in Figure 4(4.1), the consequence is an HTML code with the desired conformance to the WCAG 1.0.

3.3 Discussion

At this point, we introduce some discussion about the supporting tool and its preliminary results when assisting developers in the implementation of cases. As it is shown in Figure 4, the tool provides support at stage 3 of the development process, proposed by our aspect-oriented approach, to discover and apply crosscutting concerns and aspects from knowledge about Accessibility. Following the approach's basis, the type of support and features covered by the tool can be described as those which usually provide a Computer-Aided Software Engineering (CASE) tool with Model-Driven Development (MDD[3]) properties.

[3] Model-Driven Development is a software development methodology which focuses on creating and exploiting domain models –i.e. abstract representations of the knowledge and activities that govern a particular application domain, rather than on the computing (or algorithmic) concepts. MDD allows people to work together on a project even if their individual experience levels vary greatly.

As a CASE tool, our supporting tool results helpful to designers in creating models of cases by using reusable components and this is possible because of two reasons. On one hand, the Accessibility guidelines are quite independent from the Web application under development, so there are many cases to which the same Accessibility solution can be applied. Then, recording such recurrent situations (e.g., using patterns) enables to reuse them, which contribute to reduce the development effort when implementing our proposal. On the other hand, the Accessibility aspects as we propose in [9] could be developed once and be reused in different Web projects. For example, returning to the Student's Sign-in Web Page example in Section 1, establishing a logical tab order for accessing the HTML text field elements for the student ID and password, is an Accessibility concern that forces crosscutting in the implementation. The early identification of this situation allows modeling a reusable Accessibility aspect which is going to be in charge of providing an HTML tabindex attribute for each text field element at the user's layout. Currently, since the function for reusing components is not fully implemented, our tool provides assistance for applying the Accessibility aspects (prescribed by some predefined and stored SIG diagrams) to an abstract user interface model loaded by the designer.

As visible disadvantages of our supporting tool, we believe it is important to highlight the following issues: (i) although the part of the approach that is supported by the tool is completely documented and self-contained within a well-known Web engineering approach, its comprehension requires a prior knowledge of the WCAG 1.0 (or 2.0) guidelines and their specific terminology and also of the AOSD basis; (ii) although the tool helps to transfer Accessibility concerns, the engineering staff members should not be ruled by ad hoc practices, or used to apply approaches, which have not incorporated the design and documentation of the application under development as an standard discipline. These two issues demand changes in the development process that must be supported by the organizations.

It is a fact that for Web development, quality is often considered as higher priority than time-to-market with the mantra later-and-better [11 even though they mean extra time and cost consuming. In this sense, our supporting tool aims to help Web development with the Accessibility quality factor in mind.

4 Conclusions and Future Work

A main factor for the lack of Accessibility at the Web is the major knowledge gap that normally exists between developers and Accessibility specialists. Moreover, it is still a quiet frequent practice to consider Accessibility at the very last stages of the development process, or when applications are already coded. At this point "make these applications accessible" can mean a great deal of redesign and reprogramming effort usually outside the scope of the project --i.e. not previously planned and/or budgeted from the beginning.

In this paper, we propose a supporting tool to help designers and developers to produce accessible interfaces providing necessary information at early stage in the development process. Since we are aware that the new W3C-WAI guidelines and the move

to technological neutrality are undoubtedly good, we are almost ready to migrate from WCAG 1.0 to WCAG 2.0 ; we have already finished the migration of our aspect-oriented design approach and we are currently working on the migration of our supporting tool as well. Finally, we will extend the tool's functionality to fully implement the reusing components capabilities and to cover all our WE approach, intending to propitiate industry adoption.

5 Acknowledgments

This work is partially supported by the UNComa project 04E/072 (Identificación, Evaluación y Uso de Composiciones Software). Also by UNPA-UACO project 21/B107 (Mejora del Proceso de Selección de Componentes para Sistemas de Información Geográficos).

References

1. Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Boston (2000)
2. Chung, L. and Supakkul, S. Representing FRs and NFRs: A Goal-oriented and Use Case Driven Approach. In 2nd International Conference on Software Engineering Research, (Los Angeles, USA, 2004), Springer, 29-41 doi:10.1007/11668855_3
3. De Troyer O., Casteleyn, S., and Plessers, P. WSDM: Web Semantics Design Method. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modeling and Implementing Web Applications. pp. 303-351. Springer-Verlag, London (2008)
4. Fons, J., Pelechena, V., Pastor, O., Valderas, P., and Torres, V. Applying the OOWS Model-Driven Approach for Developing Web Applications. The Internet Movie Database Case Study. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modeling and Implementing Web Applications. pp. 65-108. Springer-Verlag, London (2008)
5. Koch, N., Knapp, A., Zhang, G., and Baumeister, H. UML-Based Web Engineering: An Approach Based on Standards. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modeling and Implementing Web Applications. pp. 157-191. Springer-Verlag, London (2008)
6. Martín, A., Cechich, A., Gordillo, S., and Rossi, G. A Three-Layered Approach to Model Web Accessibility for Blind Users. in 5th Latin American Web Congress, (Santiago de Chile, Chile, 2007), IEEE Computer Society, 2007, 76-83 doi:10.1109/LA-WEB.2007.56
7. Martín, A., Cechich, A., and Rossi, G.: Comparing Approaches to Web Accessibility Assessment. In: Calero, C., Moraga, Mª Á., Piattini, M. (eds.) Handbook of Research on Web Information Systems Quality, pp. 181-205. Information Science Reference, Hershey New York (2008)
8. Martín, A., Cechich, A., and Rossi, G.:Accesibility at early stages: insights from the designer perspective. In 8th International Cross-Disciplinary Conference on Web Accessibility (2011). ISBN: 978-1-4503-0476-4. doi:10.1145/1969289.1969302

9. Martín, A., Rossi, G., Cechich, A., and Gordillo, S. Engineering Accessible Web Applications. An Aspect-Oriented Approach. World Wide Web Journal (2010). Pp 419-440 doi 10.1007/s11280-010-0091-3.
10. Martin, A., Mazalu, R, and Cechich, A. Supporting an Aspect-Oriented Approach to Web Accessibility Design.in 5th International Conference on Software Engineering Advances, (Nice, France, 2010), IEEE, 20-25
11. Offutt, J. Quality Attributes of Web Software Applications. IEEE Software, 19(2), 2002, 25-32 doi:10.1002/stvr.425
12. PAS 78. Publicly Available Specification: A Guide to Good Practice in Commissioning Accessible Websites. Retrieved January 20, 2011 from: http://www.hobo-web.co.uk/seoblog/pas-78/
13. Plessers, P., Casteleyn, S., Yesilada, Y., De Troyer, O., Stevens, R., Harper, S., and Goble, C. Accessibility: A Web Engineering Approach. in 14th International Conference on World Wide Web, (Chiba, Japan, 2005), ACM, 353-362 doi:10.1145/1060745.1060799
14. Rossi, G. and Schwabe, D. Modeling and Implementing Web Applicactions with OOHDM. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modeling and Implementing Web Applications. pp. 109-155. Springer-Verlag, London (2008)
15. Section 508. Electronic and Information Technology Accessibility Standards. Retrieved January 20, 2011 from http://www.section508.gov/
16. Vilain, P., Schwabe, D., and Sieckenius de Souza, C. A Diagrammatic Tool for Representing User Interaction in UML. in 3rd International Conference on UML (York, UK, 2000), Springer, 133-147 doi:10.1007/3-540-40011-7_10
17. W3C: Web Content Accessibility Guidelines 1.0. (WCAG 1.0). http://www.w3.org/TR/WAI-WEBCONTENT/ (1999). Accessed 02.07.2011
18. W3C: Web Content Accessibility Guidelines 2.0. (WCAG 2.0). http://www.w3.org/TR/WCAG20/ (2008). Accessed 04.28.2011
19. W3C-WAI: Comparison of WCAG 1.0 Checkpoints to WCAG 2.0. Retrieved January 20, 2010 from: http://www.w3.org/WAI/WCAG20/from10/comparison/