World Wide Web (2010) 13:419–440
DOI 10.1007/s11280-010-0091-3

Engineering Accessible Web Applications. An Aspect-Oriented Approach

Adriana Martín · Gustavo Rossi · Alejandra Cechich · Silvia Gordillo

Received: 8 July 2009 / Revised: 29 March 2010
Accepted: 5 April 2010 / Published online: 8 May 2010

Abstract The development of accessible Web software is complicated for several reasons. Though some of them are technological, the majority are related with the need to compose different and, many times, unrelated design concerns which may be functional as in the case of most of the specific application's requirements, or non-functional such as Accessibility itself. In this paper, we present a novel approach to conceive, design and develop Accessible Web applications in an aspect-oriented manner. In order to reach our goal, we provide some modeling techniques that we specifically developed for handling the non-functional, generic and crosscutting characteristics of the Accessibility concerns. Specifically, we have enriched User Interaction Diagrams with integration points, which are used to reason and document Accessibility for activity modeling during user interface design. Then by instantiating a Softgoal Interdependency Graph template with association tables, we work on an abstract interface model (composed by ontology widgets) to obtain a concrete and accessible interface model for the Web application being developed. We use a real application example to illustrate our ideas and point out the advantages of a clear separation of concerns throughout the development life-cycle.

Keywords web accessibility · user interface models · aspect-oriented design

A. Martín (✉) · A. Cechich
GIISCo Research Group, Departamento de Ciencias de la Computación,
Universidad Nacional del Comahue, Buenos Aires 1400 Neuquén, Argentina
e-mail: adrianaelba.martin@gmail.com

A. Cechich
e-mail: acechich@uncoma.edu.ar

G. Rossi · S. Gordillo
LIFIA, Facultad de Informática, Universidad Nacional de La Plata and Conicet, La Plata, Argentina

G. Rossi
e-mail: gustavo@sol.info.unlp.edu.ar

S. Gordillo
e-mail: gordillo@sol.info.unlp.edu.ar

1 Introduction

Developing accessible Web applications is usually hard for several reasons. Most developers do not have the necessary skills or training in designing and coding for Accessibility, and most Accessibility specialists have, in turn, limited developing practice [11]. Thus, although there are many available tools [17] and published sources of information on Web Application Accessibility, existing Web Accessibility guidelines and principles (and therefore, experts on these guidelines) do not address additional design issues that may typically arise when developing complex Web applications. In most cases, Accessibility is regarded as a programming issue or even dealt with when the Web application is already fully developed and, consequently, the process of making this application accessible involves significant redesign and recoding, which might be out of the scope of the project and/or hardly affordable [11]. The proliferation of Web-based information services reinforces the need for Web content Accessibility [13]; developing accessible Web applications is no longer a matter confined to persons with disabilities, but a key issue to the entire universe of Web users. As we will show next, the main problem with Accessibility is that it is a non-functional software concern [25], which affects (crosscuts) other application concerns. Moreover, Accessibility is a generic concern that may comprise dozens of specialized concerns and, therefore, many requirements associated with these.

As a motivating example, suppose a Web page whose purpose is the student's login for his/her identification in his/her university system, like the one shown in Figure 1 corresponding to the SIU Guaraní registration system, which is used by many public universities in Argentina. It offers online information and/or diverse registration functionalities to their students. Since this kind of online systems gives support to an educational organization, Accessibility is an outstanding feature for students with disabilities.

As shown in Figure 1, the page for the student's login provides a user interface composed of HTML elements, like *labels* and *textFields*. To ensure an accessible interaction, these HTML elements must fulfill some Accessibility requirements and, it is not surprising that all of these requirements crosscut the same software artifact (the Web page for student's login). For example, and as we see in detail later, an HTML *label* element is, at the presentation, a basic layout Accessibility requirement for many others HTML elements. In particular, the HTML *label* element is critical to the Accessibility of an HTML *textField* element for people using screen readers. Since a Web page for student's login requires at least two *textField* elements (for student's ID and password respectively), the presence of their respectively *label* elements must be tested. So, to ensure an accessible interaction on

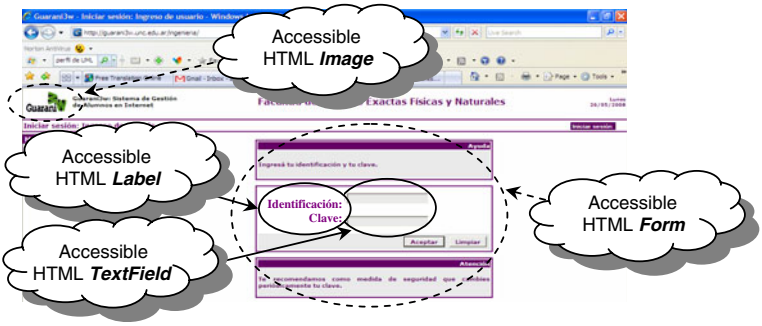


Figure 1 The student's login Web page at the SIU Guaraní registration system.

behalf of the student, this layout requirement must crosscut the same software artifact (the Web page) more than once, according to the number of *textField* elements included in the presentation. It seems natural therefore to address Accessibility using the Aspect-Oriented Software Development (AOSD) approach. The AOSD perspective states that different application concerns should be treated as first-class citizens early from requirements elicitation and specification, modeled separately and then weaved together by using specialized tools. By using the AOSD approach we can avoid typical problems of crosscutting concerns, such as those shown in the previous example.

The main thesis of this paper is that Accessibility manifests mainly during requirements gathering and specification, and user interface design. It obviously manifests while dealing with requirements since this is the step in which stakeholders express the main functional and non-functional properties the application must contain. Besides, Accessibility is one of those concerns that even though they are directly related to user's semantic dialogs, they can dramatically influence the application's context of use. For this reason, Accessibility should be analyzed as a user interface concern, since it is at the user's interface level where Accessibility barriers finally show.

The main contributions of this paper are the following:

- We present an aspect-oriented approach to handle the non-functional, generic and crosscutting characteristics of the Accessibility concerns.
- We introduce specific Accessibility modeling techniques included in a mature methodology for the development of Web applications.
- We illustrate with a real, practical example the applicability and feasibility of the approach.

The rest of the paper is structured as follows: in Section 2 we discuss some background issues. In Section 3 we offer an overview of our approach showing the model we envisage to deal with Accessibility concerns within a Web engineering approach providing a detailed step-by-step explanation of our proposal; and in Section 4 we use a real application example to illustrate our ideas. Finally Section 5 analyzes related work and also gives some insights on how to upgrade our approach from WCAG 1.0 [27] to WCAG 2.0 [28]; and in Section 6 we conclude and present further work we are currently pursuing.

2 Background

In this section we introduce four key topics that we will use throughout the rest of the paper, to make it self-contained. These are: (a) Aspect-Oriented composition, (b) Reference Frameworks and Ontologies, (c) User Interaction Diagrams (UIDs), and (d) Softgoal Interdependency Graphs (SIGs). Our aim is not to discuss these issues in detail, but to stress the most important concepts.

2.1 Aspect-oriented composition

Traditional approaches to software development are not able to deal with concerns which cut across many other concerns, producing scattered (concern's code spread through multiple modules) and tangled (a single module implementing many concerns) representations (e.g. specifications, code) that are difficult to understand and maintain. Typical examples of such crosscutting concerns are non-functional requirements, such as security, availability, persistency and Accessibility, the main topic of this paper. Aspect-Oriented Software Development (AOSD) aims at handling such crosscutting concerns by providing

means to their systematic identification, modularization and composition [10]. Crosscutting concerns are encapsulated in separate modules, known as *aspects*, and composition mechanisms are later used to weave them back with other core modules, at loading time, compilation time, or run-time. Nowadays AOSD focuses not only on the implementation phase of the software lifecycle but also on former phases as design and even earlier as requirements to cover consistently the entire development process [2, 18].

2.2 Reference frameworks and ontologies

Our approach involves two main elements when designing the user interface towards achieving Accessibility of Web applications: a reference framework and ontology.

We applied the Larson's user interface design decision framework [15] which defines the following five classes: (a) structural—specifying the structure of the end users' conceptual model; (b) functional—specifying functions (operations) which the user can apply to the conceptual objects; (c) dialog—specifying the content and sequence of information exchange between the user and the application; (d) presentation—choosing interaction objects which make up the end users' interface; and (e) pragmatic—dealing with issues of gesture, space, and hardware devices. Since the last three classes are related to the user interaction and activities with the application's interface, and they are also directly involved with Web Accessibility, we ensure their inclusion in our approach. The Larson's framework [15] gives us a comprehensive and general view that can be instantiated with different conceptual models, such as the approach proposed 11 years later by Baxley in [3]. We omit a comparison between the two approaches for the sake of conciseness.

We also applied the Abstract Widget Ontology [22]. This ontology can be thought of as a set of classes whose instances will comprise a given interface. As shown in Figure 2, an abstract interface widget can be any of the following: (a) *SimpleActivator* (a widget capable of reacting to external events, such as mouse clicks); (b) *ElementExhibitor* (a widget able to exhibit some type of content, such as text or images); (c) *VariableCapture* (a widget able to receive/capture, the value of one or more variables, such as input text fields, menus and check boxes), and (d) *CompositeInterfaceElement* (a composition of any of the abstract interface widget represented by the ontology's previous concepts). *VariableCapture* generalizes two distinct concepts. The first one is the ontology concept *PredefinedVariable,* which represents abstract interface widgets that let the selection of a subset from a set of pre-defined values. The second ontology concept is the *IndefiniteVariable,* which represents abstract interface widgets that allow entering previous unknown values, such as text typed by the user.

2.3 User interaction diagrams

A User Interaction Diagram (UID) [26] is a diagrammatic modeling technique to describe the information exchange between the application and the user. UIDs are simple state

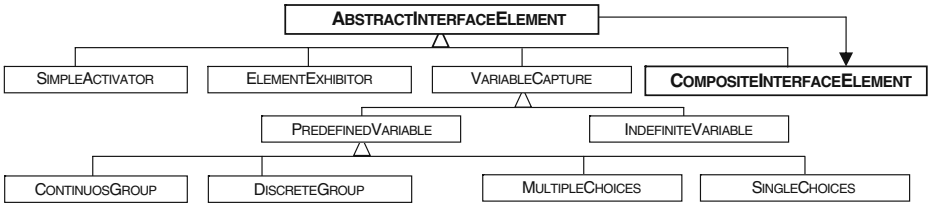


Figure 2 Abstract widget ontology [22].

machines as shown in Figure 3 where we illustrate the use case *"Enrolling a student in an examination board given a course"* in the context of the SIU Guarani registration system.

Ellipses represent an interaction between the user and the system. The initial interaction is particularly recognized by an ellipse with an arrow without a source. Each ellipse offers content to the user such as (a) data entry i.e—data entered by the user and graphically represented by a rectangle; (b) text i.e—descriptive text represented by "XXXX"; (c) a structure with their data items or a set of structures with their data items i.e—selectable elements represented by "element (data items)" or by "…element(data items)" respectively. A more formal description of the original UID's notation can be found in [26]. Arrows show the possible paths according to the user's choices.

In the interaction <1> of Figure 3 a student selects the examination option (represented by "[1]") from an initial set of options (represented by "…"). At interaction <2>, the response of the system is the set of careers in which a student is enrolled. The student chooses one of them and the system returns at interaction <3> a complete set of courses (related to the selected career) in which the student is able to enroll. The student selects a course and the system returns at interaction <4> the registration to an examination board for the course.

Additionally, the user can perform the operation *"Print registration"* (indicated by a line with a black bullet) to get a receipt of the registration completed.

2.4 Softgoal interdependency graphs

Softgoal Interdependency Graphs (SIGs) have been intensively used in software engineering for modeling non-functional requirements as "softgoals" to be "satisfied" [7, 8]. To determine satisficeability, design alternatives or decisions (called operationalizing softgoals) are considered; design tradeoffs are analyzed, design rationale is recorded and design choices are made. The entire process is recorded in a "Softgoal Interdependency Graph" (SIG) and then the selected design decisions (operationalizing softgoals) can be used as a framework for architecture and design [8]. Figure 4 depicts a SIG for the Student Friendliness softgoal in the context of our example. The light cloud indicates an NFR softgoal, denoted with nomenclature Type [Topic] where Type is a non-functional aspect—e.g. Student Friendliness, and Topic is the context for the softgoal—e.g. a Student accessing the SIU Guaraní registration system. Either Type or Topic of each NFR softgoals can be refined, one at a time, with either AND-decomposition (denoted with a single arc) or OR-decomposition (denoted with a double arc). For example, as shown in Figure 4, Student Friendliness [Student—SIU Guaraní system] is OR-decomposed into Student Friendliness [Manifest Model] and Student Friendliness [Technical Model]. The manifest model is the UI model through which the software represents its functioning to the user; while the technical model is the model with which developers feel most comfortable.

Since student friendliness is the NFR under evaluation, the focus is on the Manifest Model token that is AND-decomposed into Student Support [Manifest Model] and UI

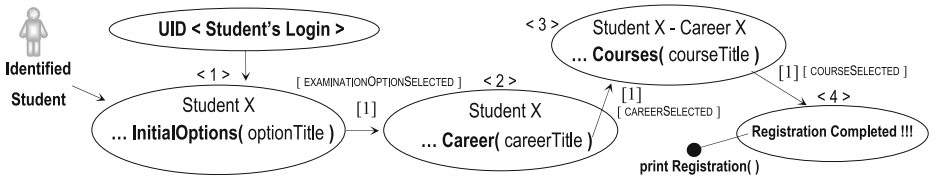


Figure 3 A simple UID: enrolling a student in an examination board given a course.

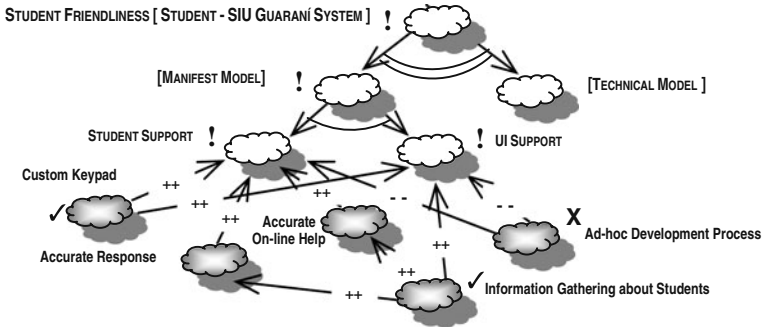


Figure 4 Softgoal interdependency graph (SIG) for student friendliness NFR.

Support [Manifest Model]. The dark cloud indicates an operationalizing softgoal. For example, in most development environments the developers agree on a basic framework and the UI is constructed in an ad-hoc manner when the screens are coded. This kind of practice has a highly negative contribution since a formal UI model is never constructed and this is the reason why in Figure 4, the operationalizing softgoal Ad-hoc Development Process is denied.

3 Our approach in a Nutshell

In the spirit of modern Web Engineering approaches, we propose a model-driven development process in which the construction of a Web application consists of the specification of a set of conceptual models, each addressing a different concern (such as navigation or interface). The model we envisage to deal with Accessibility concerns within a Web engineering approach is illustrated in Figure 5. Columns in Figure 5 indicate: (a) the overall process with their main activities (in the middle), (b) the conceptual tools and languages used (on the right) along with relations to the stage of the process where they are required, and (c) the artifacts provided as input by the WE approach and / or delivered as output by our process (on the left). As highlighted in Figure 5 (1), this process manages Web application requirements looking for those that involve Accessibility needs. Then, as shown in Figure 5 (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 5 (2.1) and (2.2) respectively.

We propose these conceptual tools basically to allow the representation of Accessibility requirements while executing a user's task (the UID technique and the SIG model are described below in Sections 3.2.1 and 3.2.2 respectively). As indicated in Figure 5 (3), this Accessibility knowledge captured at early stages aids designers making decisions through the abstract interface model, as shown in Figure 5 (3.1), and then, as shown in Figure 5 (4) toward its implementation through the concrete interface model as shown in Figure 5 (4.1).

Almost all WE approaches have an explicit development activity for user interface design and, normally, a user interface is specified by the abstract interface and the concrete interface models, providing respectively the type of functionality offered to the user by the interface elements and the actual implementation of those elements in a given runtime

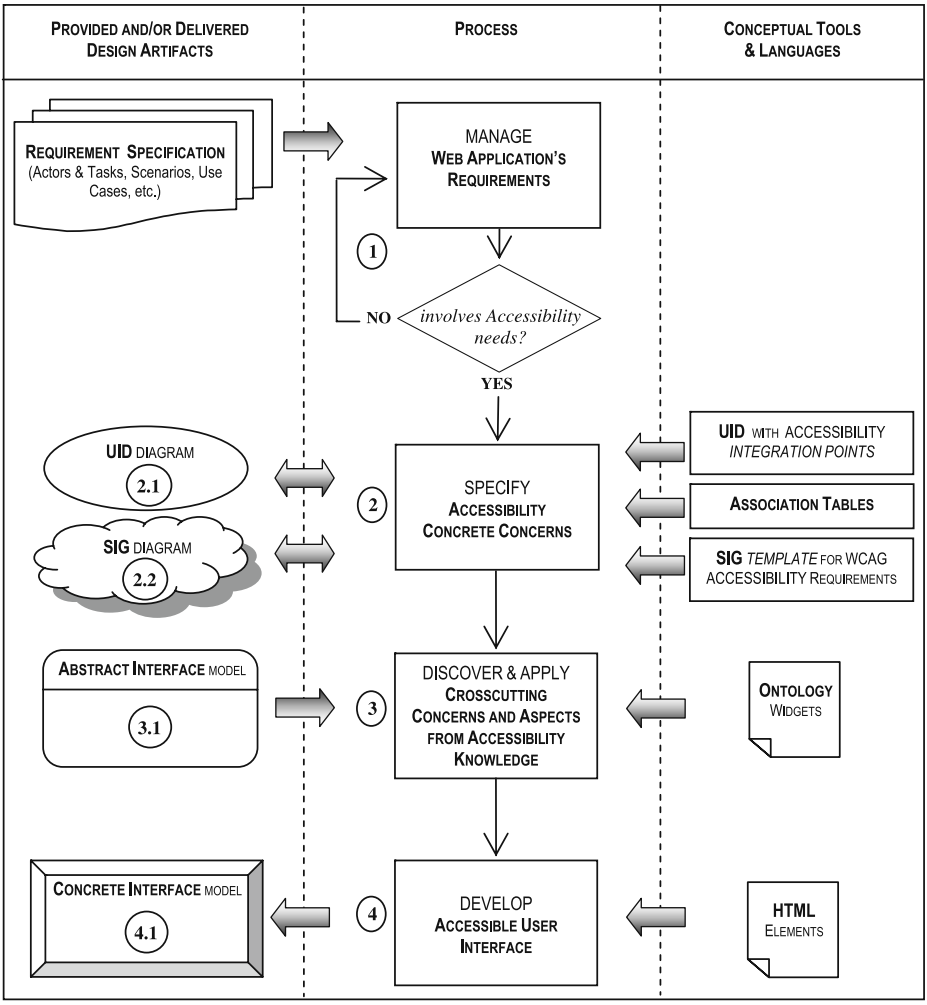


Figure 5 Overview of our approach.

environment. So, given a user's task, the SIG model provides the WCAG 1.0 Accessibility checkpoints that crosscut the interface widgets (both, abstract and concrete ones, as shown in Figure 5 (3.1) and (4.1) respectively), to ensure an accessible user experience. In the following sub-sections, we put all the pieces together to give a detailed step-by-step explanation of our aspect-oriented approach.

3.1 Identifying application's requirements that involve accessibility needs

Since we are particularly interested in discovering Accessibility concerns at the user interface design, we propose as a first step, an iterative and incremental process over the Web application's requirements looking specially at those that involve user-system interaction but also at those derived from all kind of user activity with the application's

interface. As an example, assume that we take into account the following use case *"Login a student given the student's ID and password"*:

USE CASE 1: LOGIN A STUDENT GIVEN THE STUDENT'S ID AND PASSWORD

Brief Description**:** This use case describes how a Student logs into the SUI Guaraní registration system.

Success End Condition: The Student is now logged into the system.

Primary Actor: Student

Description

MAIN SUCCESS SCENARIO:

Step	Action
1.	The system requests that the Student enter his/her ID and password
2.	The Student enters his/her ID and password.
3.	The system validates the entered ID and password and logs the Student into the system.

EXTENSIONS:

Step	Branching Action
3a.	The Student enters an invalid ID and/or password, the system displays an error message, the use case ends.

This use case describes the application's requirements for the online login Web page (introduced in Section 1 by Figure 1 from the SUI Guaraní registration system). The functionality required for the online login involves user-system interaction, since at Step 1 of the main success scenario, the student is requested by the system to enter his/her ID and password. At the SIU Guaraní registration system, Step 2 is satisfied when the student enters its identity card number as an ID and a four-digit key as a password. Then at Step 3 the system executes the validation process yielding the student logged into the system as a success end condition or displaying an error message to end the use case.

This identification process is defined as Step 1 and is graphically represented by (1) in Figure 5.

3.2 Specifying accessibility concrete concerns

Mostly because of the non-functional, generic and crosscutting nature of Accessibility concerns of a user-system interaction, we developed two conceptual tools as extensions of the UID and SIG techniques (introduced earlier in Section 2.3 and 2.4 respectively): the UID technique with *integration points* and SIG *template* for Accessibility.

As an example, let us return to the use case *"Login a student given the student's ID and password"* in Section 3.1 and consider a scenario in which a blind student using an older "screen reader" device wishes to log into the SIU Guaraní registration system. It is a fact, that Accessibility concerns related to the user layout and the user technology support must be considered to guarantee interaction and browsing regardless of the assistive device. Specifically, in this case it means that the HTML elements required for the identification form must be accessible for students using "screen readers". So, when developing functional requirements captured by the use case, we need a way to record Accessibility concerns early and as a reminder for design. With this aim in mind we developed the UID technique with *integration points* and SIG *template* for Accessibility.

3.2.1 Using UIDs with integration points technique

For each application's requirement identified at Step 1, and at Step 2, we firstly develop an UID diagram focusing mainly on outlining *integration points* where Accessibility is

crucial for ensuring a successful information exchange between the application and the user. With the traditional perspective given by techniques like [7, 8] in mind (depicted in Section 2.4), we introduce the concept of UIDs's *integration points* [16] to model the Accessibility concerns of a user-system interaction. Particularly, we define two kinds of UIDs *integration points* as follows:

- 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 interaction diagram.
- 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 of an UID interaction.

Figure 6 shows the resultant UID, corresponding to the use case *"Login a student given the student's ID and password"* (presented in Section 3.1), by applying our *integration points* technique. The development of the UID diagram with *integration points* at Step 2, is graphically represented by (2.1) in Figure 5.

3.2.2 Applying the SIG model

After specifying the Accessibility *integration points* of the UIDs diagrams at Step 2, we develop a SIG diagram for WCAG 1.0 Accessibility requirements [16]. Figure 7 shows our SIG *template* where the Accessibility softgoal denoted with the nomenclature Accessibility [UID integration point] is the root of the tree.

From the root node we identify two initial branches: (a) the user technology support, and (b) the user layout support. The user technology support represents the Accessibility softgoal concerns helping to ensure user's browsing and interaction by improving the Accessibility of user's current and earlier assistive devices and technologies (PDAs, telephones, screen readers, etc.); meanwhile, the user layout support represents the Accessibility softgoal concerns explicitly improving user's browsing and interaction focus on user's interface issues. For example, returning to Figure 6, we establish 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* for the login process at UID interaction <1>. The instantiation process of the SIG template is conducted as a refinement process over the SIG tree using *association tables* as a reference for groups of related HTML

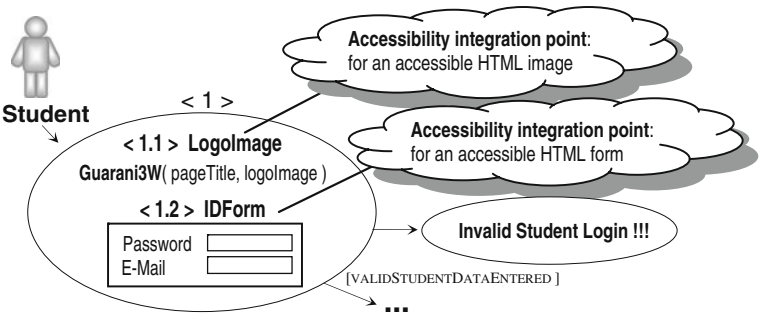


Figure 6 UID with accessibility integration points: login a student given the student's ID and password.

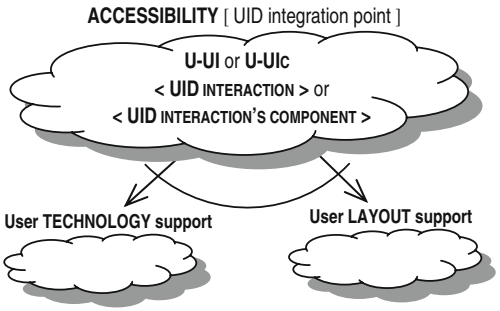


Figure 7 SIG template for accessibility.

elements. For example, Table 1 introduces the *association table* that we have developed for the HTML *control* group.

Basically, *association tables* link each ontology concept—i.e. abstract widget, with their respective HTML elements—i.e concrete widgets, and with the Accessibility concerns prescribed for those widgets by the WCAG 1.0 checkpoints. We further describe *association tables* in section 3.4.2.

3.3 Discovering crosscutting and applying aspects

The purpose at Step 3 is to find out how WCAG 1.0 Accessibility concerns "crosscut" interface widgets required for the online login Web page, aided by the abstract interface model shown in Figure 5 (3.1). More specifically, the SIG diagrams and the *association tables* work together to discover the required WCAG 1.0 checkpoints for assuring the student's login but also to show how aspect-oriented "symptoms" ("scattering" and/or "tangling") manifest their crosscutting nature on the HTML *image* and HTML *form* elements. For example, and as we will see in-depth later, from guidelines 10 responding to the statement "use interim[1] solutions", satisfacing the 10.4 checkpoint is a "mandatory" goal (set with an "M") required for every HTML *control* element, and establishes that empty controls must be handled correctly "until user agents".[2] So, to ensure this Accessibility requirement, the checkpoint 10.4 will be "scattered" at the login Web page of the SUI Guaraní registration system every time that an HTML *textField* element (corresponding to an *IndefiniteVariable* widget) is present. It is important to highlight that ensuring compliance to Accessibility is, in several cases, similar for those HTML elements sharing the same HTML group. As we can see on Table 1, this is the case for the HTML *control* group.

3.4 Designing with accessible interface widgets

3.4.1 A mapping between ontology concepts and HTML elements

Taking into account the Abstract Widget Ontology [22] described in Section 2.2, we map the ontology concepts onto HTML elements. We have materialized this mapping using

[1] Interim is used by the W3C as a temporary recommendation to ensure that while assistive technologies and older browsers exist they will operate correctly.

[2] "Until User Agent" (UUA) is used by W3C to refer to "User Agents" (UA) i.e.—software or assistive technologies, that require content developers to provide additional support for Accessibility until most user agents readily available to their audience include the necessary Accessibility features.

Table 1 Association table for the HTML control group.

ASPECT	ONTOLOGY WIDGETS (ABSTRACT WIDGETS)	HTML ELEMENTS (CONCRETE WIDGETS)		WCAG 1.0 CHECKPOINTS AND THEIR PRIORITIES: [1] [2] OR [3]					DESIGN DECISION CLASS related to USER-APPLICATION INTERACTION
				9.4 9.5 [3]	10.2 [2]	10.4 [3]	12.3 [2]	12.4 [2]	
				D-P 🛠	P	P	D-P	P	DIALOG (D) PRESENTATION (P) PRAGMATIC 🛠
I. TSCONTROL SIG'S USER TECHNOLOGY SUPPORT BRANCH	INDEFINITEVARIABLE	TEXT FIELD	INPUT TEXT…	✓	✓	✓			
		TEXT AREA	TEXTAREA…	✓	✓	✓			
		RELATED CONTROLS	FIELDSET…		✓	✓			
	PREDEFINEDVARIABLE MULTIPLECHOICES	CHECK BOX	INPUT CHECKBOX…	✓	✓	M			
		MULTIPLE OPTION MENU	SELECT MULTIPLE…	✓	✓	✓			
		RELATED OPTIONS	OPTGROUP…		✓	✓			
	PREDEFINEDVARIABLE SINGLECHOICES	RADIO BUTTON	INPUT RADIO…	✓	✓	M			
		SIMPLE OPTION MENU	SELECT…	✓	✓	✓			
II. LSCONTROL SIG'S USER LAYOUT SUPPORT BRANCH	INDEFINITEVARIABLE	TEXTFIELD	INPUT TEXT…				✓	✓	
		TEXTAREA	TEXTAREA…				✓	✓	
		RELATED CONTROLS	FIELDSET…				✓	✓	
	PREDEFINEDVARIABLE MULTIPLECHOICES	CHECKBOX	INPUT CHECKBOX…				✓	✓	
		MULTIPLE OPTION MENU	SELECT MULTIPLE …				✓	✓	
		RELATED OPTIONS	OPTGROUP…				✓	✓	
	PREDEFINEDVARIABLE SINGLECHOICES	RADIO BUTTON	INPUT RADIO…				✓	✓	
		SIMPLE OPTION MENU	SELECT…				✓	✓	

UML class diagrams to explain the relationships between each abstract interface widget presented by the ontology concepts, and the concrete interface widget in HTML elements. Figure 8 shows the UML class diagram for the ontology concept *VariableCapture*, whose functionality is to capture the value of one or more variables, implemented in HTML by *control* elements. They can be grouped together in an HTML *form* element, which is a possible implementation of the ontology concept *CompositeInterfaceElement*. Users interact with an HTML *form* through HTML *controls* by modifying their values before submitting the form to an agent, like a Web server or a mail server, for processing.

In this way, we map the ontology concepts onto five groups of HTML elements: (a) the *VariableCapture* maps onto the HTML *control* group, as we shown in Figure 8; (b) the *SimpleActivator,* which is capable of reacting to external events such as mouse clicking, maps onto HTML *link* and *button* group; (c) the *ElementExhibitor,* which is able to exhibit

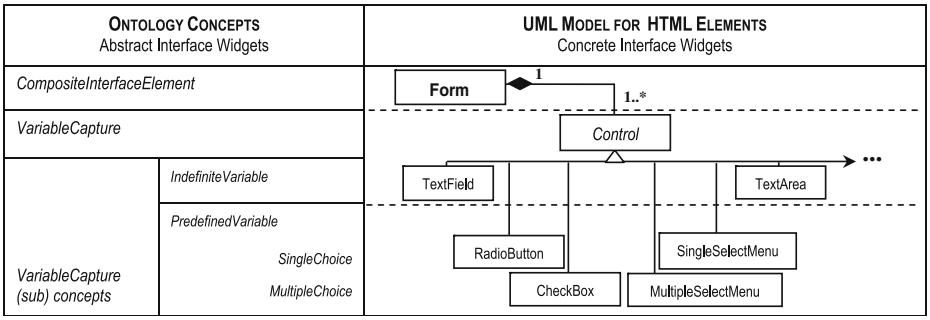


Figure 8 Mapping between some ontology concepts and HTML elements.

different types of content, such as text, images or applets, maps onto HTML *text* and *non-text* groups; (d) the *LogicalStructuring*, which is able to logically organize the content of the document, maps onto the HTML *structural* group; and (e) the *ElementStyling*, that is able to display the content with a certain appearance, maps onto HTML *frame* and *style sheet* groups.

Since most of the HTML elements are composed by other HTML elements, an accessible HTML element requires the Accessibility of all its components. So a deeper look about HTML elements composition is required to work properly with Accessibility issues.

Figure 9 explains HTML element composition providing a more detailed description of the HTML *control* elements: *textField* and *texArea*; *radioButton* and *singleChoiceMenu*; and *checkBox* and *multipleChoiceMenu* (see Figure 9a, b and c respectively). For example, *label* is a very important element to achieve the goal of making a HTML *form* accessible, because, if used correctly, it can provide helpful support to people with disabilities. The WCAG 1.0 is very clear about the Accessibility role of the *label* element when developing HTML *form* elements. Specifically, the document provides two checkpoints, one related to the user layout support and the other to the user technology support—i.e precisely the two initial branches of our SIG *template* for Accessibility, to be consider when "labeling" HTML *control* elements that are in an HTML *form*.

3.4.2 Association between ontology concepts-HTML elements-WCAG checkpoints

To develop and exploit the SIG diagrams for managing crosscutting in an aspect-oriented manner, we establish five *association tables*, one for each group of HTML elements defined

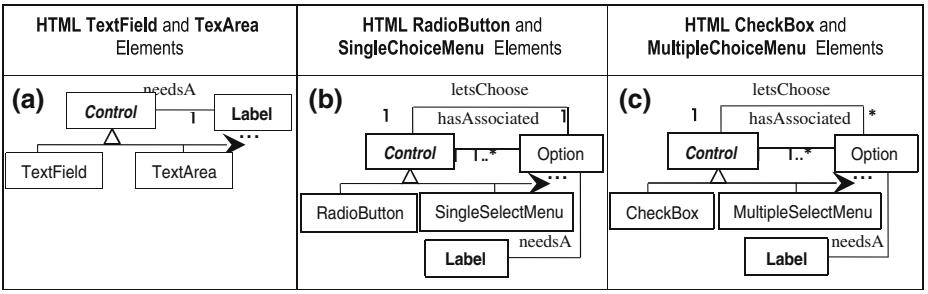


Figure 9 UML model for HTML control elements.

in Section 3.4.1: (a) the HTML *control* group as we shown in Figure 8 and Figure 9; (b) the HTML *link* and *button* group; (c) the HTML *text* and *non-text* group; (d) the HTML *structural* group; and (e) the HTML *frame* and *style sheet* group. We called them *association tables* because of two strong reasons. On one hand, they bind the WACG 1.0 checkpoints required for ensuring Accessibility of the interface widgets present at each HTML group. On the other hand, they help to classify these WCAG 1.0 checkpoints into the two initial branches of our SIG *template* for Accessibility.

Table 1 introduces the *association table* for the HTML *control* group. A checkpoint cell for a specific interface widget is selected only when the HTML element requires ensuring Accessibility by the checkpoint. As we can see in Table 1, this *association table* also indicates each checkpoint priority level assigned by the WCAG 1.0 [27]: (a) [Priority 1] checkpoints that "must" be satisfied, (b) [Priority 2] checkpoints that "should" be satisfied and, (c) [Priority 3] checkpoints that "may" be satisfied. This information allows interface designers to keep in mind the impact of the Accessibility barrier when not satisfying each checkpoint. To address Accessibility of the HTML *control* elements, guidelines 9, 10 and 12 deal with the question of what to do to make HTML *form* elements accessible [27, 29].

On Table 1, Aspect I called "TSControl" evaluates control's widgets Accessibility to improve user's current and earlier assistive devices and technologies. The association between Accessibility softgoal concerns (represented by the WCAG 1.0 checkpoints and their priorities) and the design decision classes is showed in the table with a "P" for the presentation class, a "D" for the dialog class and by the "🛠" symbol for the pragmatic class. As examples over this branch, satisfying checkpoints 9.4 and 9.5 responding to the statement "design for device-independence" of guideline 9 and, checkpoints 10.2 and 10.4 responding to the statement "use interim solutions" of guideline 10, are goals required for every HTML *control* element. The checkpoint 9.4 establishes that we should "create a logical tab order through links, form controls, and objects [Priority 3]" [27]. While the checkpoint 9.5 establishes that we should "provide keyboard shortcuts to important links (including those in client-side image maps), form controls, and groups of form controls [Priority 3]" [27]. The checkpoint 10.2 establishes that "until user agents support explicit associations between labels and form control, for all form control with implicitly associated labels, ensure that the label is properly positioned [Priority 2]" [27]. While the checkpoint 10.4 establishes that "until user agents handle empty controls correctly, include default, place-holding characters in edit boxes and text areas [Priority 3]" [27].

On Table 1, Aspect II called "LSControl" evaluates control's widgets Accessibility to improve user's interface issues, and it is supported by softgoals to be satisfied at the SIG's user layout support branch. Over this branch, the checkpoint 12.4 establishes "associate labels explicitly with their controls [Priority 2]" [27]; while, checkpoint 12.3 establishes "divide large blocks of information into more manageable groups where natural and appropriated [Priority 2]" [27]. Checkpoints 10.3 and 10.4 are goals required for all the HTML *control* elements and are focused on providing context and orientation information to help users understand complex pages or HTML elements. For example, complex relationships between HTML *control* elements as parts of an HTML *form* on a Web page may be difficult for people with cognitive or visual disabilities a to interpret.

Similarly, to Table 1, we developed Tables to describe the HTML *link* and *button*, HTML *text* and *non-text*, HTML *structural* and HTML *frame* and *style sheet* groups respectively. For brevity reasons, we include here only the HTML *control* group as shown in Table 1.

4 A case study

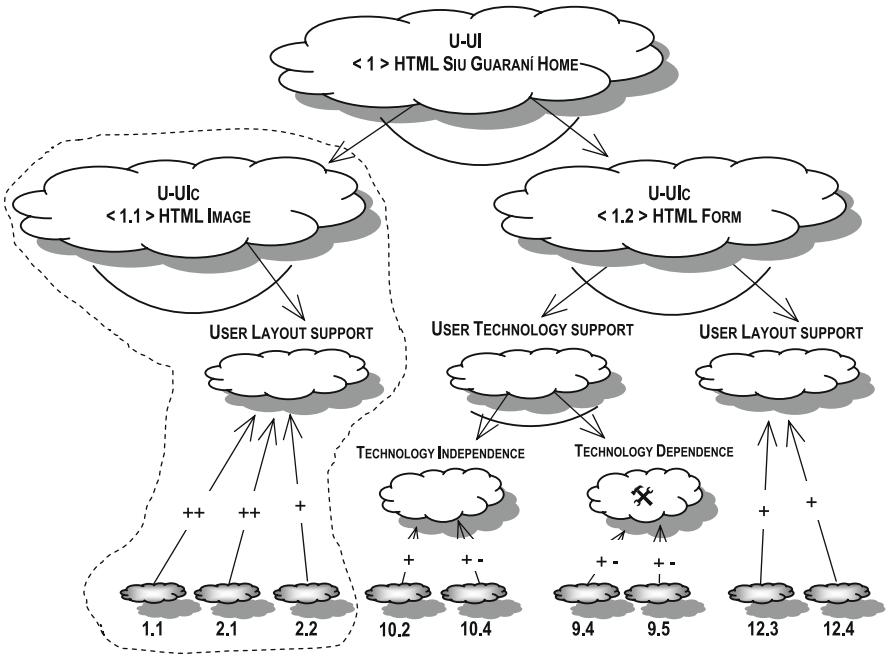
The SIU Guaraní student registration system is been used by a number of public universities in Argentina. It offers online information and/or diverse registration functionalities to their students. Since these kind of online systems give support to an educational organization, Accessibility has a main role for students with disabilities. We use the online student's identification function, shown in Figure 1, as the case study to apply our aspect-oriented approach. To carry out the implementation of our approach clearly, we follow the step-by-step process as we defined in Section 3.

Step 1. This kind of application, which is archetypical in educational domains, allows students to login at his/her university system. During the execution of the task, the requirements for a student's identification involve user-system interaction. The corresponding user interface design must provide the interface widgets (both abstract and concrete ones) required for student's login. In this case, the Web engineering activity that develops the user interface design must include at least, for the student's identification purpose, two widgets of the type *IndefiniteVariable* at the abstract interface model mapped to the concrete interface model on two widgets of the type HTML *textField*. The mission of these widgets is to receive the student's identification and password values. Normally, these two widgets are grouped together into a *CompositeInterfaceElement* at the abstract interface model and mapped to the concrete interface model on an HTML *form*.

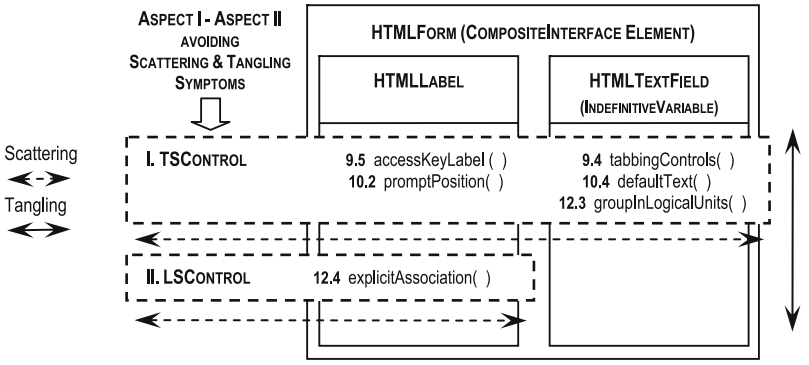
Step 2. For specifying Accessibility concerns we encourage the early capture of these Accessibility requirements applying the UID and SIG conceptual tools as follow:

2.1. We develop an UID diagram describing the task and outlining two *integration points* at the UID interaction <1> corresponding to the student's identification process. As shown in Figure 6, the UID for this task has set two *integration points* at the UID interaction <1>. This is because Accessibility is crucial for ensuring a successful login information exchange between the student and the application, during the execution of the identification function.

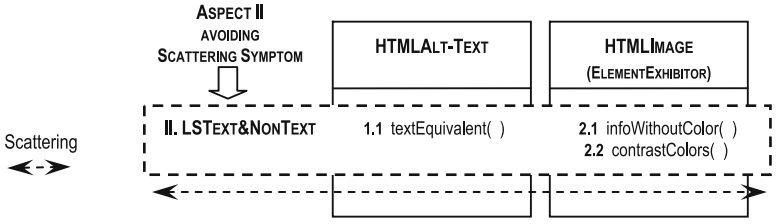
2.2. We develop a SIG diagram for the Accessibility *integration points* outlined by the UID in Step 2.1 to identify WCAG 1.0 Accessibility requirements. Specifically, we define Accessibility softgoals for the UID interaction's components <1.1> LogoImage and <1.2> IDForm, to guarantee accessible image and text input fields for all the students, including those with disabilities. Using the SIG's notation and vocabulary, the HTML *image* and the HTML *form* at the concrete interface model, corresponding to an *ElementExhibitor* and to a *CompositeInterfaceElement* of the abstract interface model respectively, are the focus of the Accessibility softgoal highlighted into the root light cloud. As shown in Figure 10a, the user technology support and the user layout support branches are specified into light clouds and dark clouds respectively. The light clouds represent the refined Accessibility softgoal—i.e. the required WCAG 1.0 guidelines; while the dark clouds represent operationalizing goals—i.e. the required checkpoints to be satisfied. In this case, we use the *association table* for HTML *Text and Non-Text* elements and Table 1 for the HTML *control* elements, since the



(a) SIG instantiation for an Accessible Student's Login at UID Interaction < 1 > of **Fig.** 6



(b) WCAG 1.0 checkpoints crosscuting an HTML *form* element (Concrete Interface Widget) corresponding to a *CompositeInterfaceElement* (Abstract Interface Widget)



(c) WCAG 1.0 checkpoints crosscuting an HTML *image* element (Concrete Interface Widget) corresponding to a *ElementExhibitor* (Abstract Interface Widget)

Figure 10 Managing crosscutting symptoms in an aspect-oriented manner.

Accessibility softgoal is defined for a LogoImage and an IDForm. Then, the refinement process for the SIG instantiation is carried out as follow.

Firstly, looking at the user technology support branch in Figure 10a, a distinction between "technology independence" and "technology dependence" is made and to ensure the universal access of devices to the HTML *form* element, we chose an and-decomposition; but the choice for an AND / OR decomposition will depend on the designer's decisions and the application's constraints. For "technology independence", satisfying goals related to guideline 10 for checkpoints 10.2 and 10.4 compliance are required. Otherwise for "technology dependence", satisfying goals related to guideline 9 for checkpoints 9.4 and 9.5 compliance are required. Now looking at the user layout support, satisfying goals related to guideline 12 for checkpoints 12.3 and 12.4 compliance are required for the HTML *form* element; while satisfying goals related to guidelines 1 and 2 for checkpoints 1.1; 2.1 and 2.2 compliance are required for the HTML *image* element.

Step 3. For the user interface design activity, we exploit the Accessibility knowledge captured and organized by SIG diagrams at Step 2.2. The purpose here is to find out how WCAG 1.0 Accessibility requirements "crosscut" interface widgets required for a LogoImage and an IDForm.

In order to make our discussion clear, Figure 10b and c illustrate how the SIG's operationalizing goals—i.e. the required WCAG 1.0 checkpoints to be satisfied for an accessible student's login—"crosscut" the components of an HTML *form* element and an HTML *image* element, corresponding to a *CompositeInterfaceElement* and *ElementExhibitor* ontology widgets respectively. 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, it is clear that aspect-orientation is the natural approach to solve these crosscutting symptoms. The SIG diagrams not only provide Accessibility technology and layout support respectively for any of the HTML *image* and HTML *form* components at the user interface, but also allow Aspects I and II to be modeled and instantiated appropriately to avoid "scattering" and "tangling" problems. Then Aspects I and II can be seamless injected by aspect "weaving" mechanism into the core—i.e. user interface models, to achieve the Accessibility softgoal and as a consequence an HTML code with the desired conformance to the WCAG 1.0.

For example, as shown in Figure 10c, whenever there is an HTML *image* element at the user interface model, Aspect II "LSText&Non-Text" is injected to avoid the "scattered" nature of Accessibility checkpoints 1.1, 2.1 and 2.2 over HTML *image* classes. The addition off Aspect II "LSText&Non-Text" guarantees later, at the concrete interface model implementation, conformance to the following Accessibility properties for each HTML *image* element: (a) adding an HTML *alt-text* description and, (b) not relying on images' color alone to convey information. While, as shown in Figure 10b, whenever there is an HTML *form* element at the user interface model, Aspect I "TSControl" and Aspect II "LSControl", focused on solving technology and layout Accessibility issues respectively, are injected to avoid the "scattered" and "tangling" nature of Accessibility checkpoints 9.4, 9.5, 10.2, 10.4 12.3 and 12.4 over HTML *form* classes.

Step 4. Finally, as a result of modeling Aspect I and Aspect II (using SIG's prescription for WCAG 1.0 checkpoints) and the addition of these aspects to deal with the targeted interface widgets, we produce an accessible HTML implementation for the concrete interface model required by the online student's ID.

5 Related work and discussion

The main advantage of our approach is the possibility to treat Accessibility as an independent AOSD concern at early stages of the development's life cycle, therefore eliminating crosscutting and as a consequence allowing more modular system development and reuse of Accessibility aspects.

Further on, we review similar approaches that consider modeling the Accessibility concerns in at least, some of the stages of the development life-cycle.

The main goal in Plessers et al. [21] is to generate the annotations for visually impaired users automatically from the explicit conceptual knowledge existing during the design process. The approach integrates the Dante [31] annotation process into the Web Site Design Method (WSDM) [9] that allows Web applications to be developed in a systematic way. The annotations are generated from explicit conceptual knowledge captured during the design process by means of WSDM's modeling concepts. These WSDM's modeling concepts, used in the different phases, are described by using the WSDM OWL[3] ontology. To generate code that is annotated with concepts from the Dante's WAfA[4] ontology [31], a relationship between the concepts in the WSDM ontology and the WAfA ontology is established. By using these mapping rules, the authors establish a transformation process that takes the conceptual design models as input and generates a set of annotations as a consequence. We have decided to include this approach primarily because it is a project that has its main focus on Accessibility, which is addressed from a Web Engineering (WE) perspective. This is a point in common with our proposal, since both are embedded into a recognized WE approach: the WSDM and the OOHDM respectively. However, although this approach is embedded in a WE method, the mayor effort is focused on the interface transformation process to improve the support for screen readers, using for this purpose their own concepts from the WAfA ontology [31]. As a consequence, the approach has a restricted dependency on this ontology and it does not work directly with concepts from world-wide recognized Accessibility standards. Here, there is a substantial difference with respect to our proposal since we follow established Accessibility guidelines and, by providing specific design techniques, we carry out these guidelines systematically over models of a WE process allowing developers to improve interface designs for all kinds of disabilities.

The work by Centeno et al. [5] presents the set of rules that, in a Web composition process, a design tool must follow in order to create accessible Web pages. These rules are formalized with W3C standards like XPath[5] and XQuery[6] expressions, defining conditions to be met in order to guarantee that Accessible chunks of Web pages are safely composed into a page that also turns out to be Accessible. The XPath and XQuery expressions spot HTML nodes and attributes having Accessibility problems. This work proposes to properly manage these spot elements by an authoring tool, so that the author's attention can be directly brought to these barriers in a semi-automated edition process. The WSLS approach follows the AOSD separation of concerns principle to decompose complexity and control Accessibility over six distinguished categories: Data, Presentation, Navigation, User, Interaction, Process and Communication. The six elements are mediated by a service control function. Beyond the advantage of the reuse aspect of these components, separation

[3] OWL Web Ontology Language at http://www.w3.org/TR/owl-ref/

[4] Web Authoring for Accessibility (WAfA) at http://hcw.cs.manchester.ac.uk/experiments/ontologies/wafa.owl

[5] W3C XML Path Language at www.w3.org/TR/xpath

[6] W3C XML Query Language at www.w3.org/TR/xquery

of concerns facilitates also being compliant to the underlying guidelines [5]. Although the fact this approach works like ours with mature WCAG standards, the focus of the proposal is on providing Accessibility support to Web composition processes managed by an authoring tool. The proposal results in a pragmatic approach to sustain Accessibility principles while a graphic designer develops websites from reusable chunks of accessible HTML code. However, since the approach bases its strategy on the assumption that there is a repository of accessible pieces of HTML markup, the delivery of accessible Web pages is performed only as a product of the composition of such pieces. It does not pay attention to Web application modeling issues to support accessible design. On the other hand, our approach aims at providing full support to Accessibility with modeling techniques specifically designed to be integrated within the steps of a Web application development process.

Using "the best existing practices of software engineering" for Accessibility purposes, the approach by Zimmermann & Vanderheiden [32] presents a methodology for accessible design and testing to capture functional requirements. The approach defines a new way of using proven tools of software engineering, like use cases, scenarios, test cases, guidelines and checkpoints for Accessibility purposes; and to relate them to each other, thus facilitating automation as much as possible. The resultant methodology consists of: (a) capturing Accessibility requirements in a way that makes them tangible and comprehensible, through use cases and the technique of user profiling "personas" [32], (b) making Accessibility requirements concrete through scenarios and guidelines for accessible design, (c) manual and automatic testing based on test cases and Accessibility checkpoints that are derived from guidelines, and (d) complementary user testing and expert reviews, and continuously improving the overall process model. In this way, for design projects that are applying a use case driven methodology, this approach enables us to incorporate accessible design into the existing processes rather than having to add Accessibility as a new process [32]. The approach has two important features to highlight. The first one is encouraging the use of existing standards for Accessibility (although, it does not apply in particular to any of these standards). The second one is incorporating accessible design into proven development processes using a use case driven methodology. While an initial analysis seems to show points in common with our proposal, there is a conceptual difference between both approaches when it comes to the Accessibility requirements. The accessible design proposed by Zimmermann & Vanderheiden [32] is based on "the best existing practices of software engineering", which basically uses cases and scenarios that were designed to meet functional requirements. From this perception, the approach merely attaches Accessibility requirements to functional diagrams, for the purpose of reminding them during the testing stage, and verifying guidelines and checkpoints derived from the applied Accessibility criteria. Since the beginning of our work, we have claimed about the non-functional, generic and crosscutting characteristics of Accessibility and certainly, this inherent nature requires approaches which ensure an appropriate treatment to the Accessibility concerns. Keeping this conceptual principle in mind, our approach gives an accurate answer extending "proven design best practices of WE".

Casteleyn et al. [4], focus on how to extend an application with a new functionality without having to redesign the entire application. The work states that since creating a Web application has become an increasingly complex task, various design issues like device-dependence, privacy, security, Accessibility, localization, personalization, etc. have become extremely relevant to the performance of the application. To add new functionality, the authors propose to separate additional design concerns and describe them independently. They demonstrate how an aspect-oriented approach can support the high-level specification

of these (additional) design concerns at a conceptual level. The work first illustrates how to add adaptation to an existing Hera-based Web application [12] by using a component-based implementation. A further work [19] over this foundation proposes an aspect-oriented view on adaptation engineering within the AMACONT[7] framework. By separating the specification of adaptation from the underlying application in the form of so-called adaptation aspects, the proposal adds new or modifies existing adaptation concerns on demand. Although, this approach is primarily focused on adapting an existing Web application, we include it because the approach suggests adding relevant design concerns, like Accessibility, in an aspect-oriented manner and, it is representative of other similar works in the adaptation field, like [1, 23]. Although this is not an Accessibility approach itself, we consider it because of two folds: (a) it clearly establishes the crosscutting nature of adaptation concerns like personalization, localization, security and even Accessibility, (b) it recognizes that WE approaches fail to cope with this crosscutting nature of adaptation concerns. Over these foundations the authors developed an interesting work to manage transformations required for an adaptation concern. Nevertheless, two points should be noted: the support effort is focused on the presentation generation level and we do not have evidence of the result of implementing this proposal on Accessibility standards and issues.

The objective of our approach is different from the former ones because it is focused on providing specific modeling techniques to include Accessibility systematically in a methodology for Web applications development. The fact that we choose the aspect orientation to develop our proposal ensures handling naturally the non-functional, generic and crosscutting characteristics of the Accessibility concerns. As we have already seen in this section we have not found evidence of approaches addressing Accessibility early from requirements and through design to implementation, following the Web engineering philosophy.

At this point, we want to reflect on the advantages/disadvantages of model-driven approaches and how this issue benefits/affects our proposal. It is a fact that applying "unified", model-driven approaches brings the benefit of having full documentation and automatic application generation at the expense of introducing some bureaucracy into the development process. For instance, consider the spectrum of model-based approaches for developing Web applications as UWE [14], OOHDM [22], WebML [6] or WSDM [9]. Since our proposal suggests the early treatment of the Accessibility concerns through models specified in the context of the OOHDM method, we may still be influenced by this reality and its disadvantages—i.e. time and cost consuming, complexity, learning effort, etc. Being aware of this situation, we would like to address some insights that can assist the application of our proposal.

Firstly and since the Accessibility guidelines are quite independent from the Web application under development, there are many cases to which the same Accessibility solution can be applied. On this basis, it is very common to find recurrent Accessibility situations when designing user interfaces for Web applications. Then, recording such situations (e.g. using patterns) might contribute to reuse them, which contributes to reduce the development effort when implementing our proposal. This is possible because aspects, as we have already explained, could be developed once and be reused in different Web projects. For example, user interfaces typically include images and actions that should be carried out to break down the Accessibility barrier for blind users

Secondly, and in order to simplify the implementation of our approach, we are working on the development of a tool to assist designers in the implementation of cases, and on the

[7] System Architecture for Multimedia Adaptive WebCONTent at http://www-mmt.inf.tu-dresden.de/Forschung/Projekte/AMACONT/index_en.xhtml

creation of their corresponding models by using reusable components. Currently, we are working on supporting the first phase of our approach by assisting and implementing the application of SIGs diagrams to interface models.

5.1 Migrating to WCAG 2.0

Since the WCAG has two documents (1.0 and 2.0), it is important to make clear at this point why we chose the 1.0 document. WCAG 1.0 [27] has been used worldwide since 1999 as a reference material or cited as a normative from many other accessibility documents in the world [20, 24]. It also has been implemented by many tools and approaches. Although the WCAG 2.0 has been released in December 2008 and has already been a standard for about a year, it is a fact that so far the rate of adoption has been relatively slow and the number of countries and other regulating authorities now using WCAG 2.0 are still not enough compared to what was expected.

However, aware that the new guidelines and the move to technological neutrality are undoubtedly good, we do not see major inconveniences to upgrade our approach to WCAG 2.0 when necessary. As we discussed before, our approach is based on the use of UIDs with *integration points* and the SIG *template* for Accessibility linked by *association tables.* These conceptual tools are able to support the success criteria from WCAG 2.0 instead of checkpoints from WCAG 1.0 applying some straightforward redefinitions and adjustments. We highlight that to realize this upgrade we use the comparison provided by W3C-WAI in [30], since there are still some discrepancies at the Accessibility community[8] when providing mappings between the WCAG 1.0 checkpoints onto the WCAG 2.0 success criteria. A complete analysis of this upgrade is outside the scope of the paper.

6 Conclusions and future work

In this paper, we presented a novel WE approach to conceive, design and develop accessible Web applications using aspect-oriented concepts, which enabled us to address Accessibility early from requirements and through design to implementation. We used a real application example to illustrate our ideas and point out the advantages of a clear separation of concerns throughout the development life-cycle.

First of all, aspect-orientation capabilities constitute an important driver to efficiently capturing the orthogonal properties that are typical of the Accessibility's nature. Secondly, organizing these properties into a model-driven approach gives us better visibility of the components at different levels—i.e. from its conceptualization to its instantiation by particular Accessibility rules. This is especially important when reasoning about the different properties, because their complexity may be adequately addressed.

In addition, we provided explicit analysis and design techniques aiming at facilitating the capture of early Accessibility concerns. These techniques might be combined with traditional web engineering methods, which would help introduce and deploy our approach in the industry. However, we must take into account that the inclusion of new conceptual tools for treating Accessibility requires an extra effort for developers to get familiar with them. In this sense, we are currently incorporating our ideas into design tools to assist developers to design model-driven accessible Web applications.

[8] See http://www.w3.org/WAI/WCAG20/from10/comparison/; http://wipa.org.au/papers/wcag-migration.htm; http://www.usability.com.au/resources/wcag2./

Considering the extensibility of our approach, it is important to highlight, that although in this work we focused on visual disabilities, the proposal can be extended to all kinds of disabilities as the conceptual tools we provided (the UID with *integration points* and SIG *template* for Accessibility) are generic enough to capture Accessibility requirements for all types of impairments. The reason why we use visual impairment is based on the fact that ensuring Accessibility requirements for blind people, to a certain extent, covers Accessibility requirements for other disabilities. For example, the checkpoint 1.1 of the WCAG 1.0 establishes that text equivalents must be written to convey all essential content; therefore ensuring compliance to checkpoint 1.1 is vital for visually impaired users. The fact is that the absence of non-text equivalents represents a critical Accessibility barrier for people with visual disabilities, but ensuring text-equivalent also improves Accessibility for users with deafness, cognitive and learning disabilities. So, we considered the treatment of visual impairments as a good starting point.

Finally, we should further validate our proposal beyond the case study presented in this work. To do so, we are currently analyzing the impact of applying our proposal on quality attributes of the resulting system, such as *extensibility*, *reuse*, and *modularity*; and the developing effort required when using the approach. We are currently carrying out some guided experiments in the area of Web-based systems for academic domains.

References

1. Baumeister, H., Knapp, A., Koch, N., Zhang, G.: Modelling adaptivity with aspects. In ICWE (2005) doi:10.1007/11531371_53
2. Baniassad, E.L.A., Clements, P.C., Araújo, J., Moreira, A., Rashid, A., Tekinerdoga, B.: Discovering early aspects. IEEE Softw **23**(1), 61–70 (2006)
3. Baxley, B.: Universal model of a user interface. DUX (2003) doi:10.1145/997078.997090
4. Casteleyn, S., Fiala, Z., Houben, G-J., van der Sluijs, K.: Considering additional adaptation concerns in the design of web applications. AH (2006) doi:10.1007/11768012_28
5. Centeno, V., Kloos, C., Gaedke, M., Nussbaumer, M.: Web composition with WCAG in mind. W4A (2005) doi:10.1145/1061811.1061819
6. Ceri, S., Brambilla, M., Fraternali, P.: The history of WebML lessons learned from 10years of model-driven development of web applications. Concept Model (2009). doi:10.1007/978-3-642-02463-4_15
7. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-functional requirements in software engineering. Kluwer Academic Publishers, Boston (2000)
8. Chung, L., Supakkul, S.: Representing FRs and NFRs: a goal-oriented and use case driven approach. SERA (2004) doi:10.1007/11668855_3
9. Troyer, D., Casteleyn, S., Plessers, P.W.S.D.M.: 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)
10. Filman, R., Elrad, T., Clarke, S., Aksit, M.: Aspect-oriented software development. Addison-Wesley, Vancouver (2004)
11. Hoffman, D., Grivel, E., Battle, L.: Designing software architectures to facilitate accessible web applications. IBM Syst J **44**(3), 467–484 (2005)
12. Houben, G-J., van der Sluijs, K., Barna, P., Broekstra, J., Casteleyn, S., Fiala, Z., Fransincar, F. Hera: In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web engineering: modeling and implementing web applications, pp. 163–302. Springer-Verlag, London (2008)
13. Kim, Y.B.: Accessibility and usability of user-centric web interaction with a unified-ubiquitous name-based directory service. World Wide Web J **13**(1–2), 105–120 (2010)
14. Koch, N., Knapp, A., Zhang, G., 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)
15. Larson, J.: Interactive software: tools for building interactive user interfaces. Prentice Hall, NJ (1992)
16. Martín, A., Cechich, A., Gordillo, S., Rossi, G.: A Three-layered approach to model web accessibility for blind users. LA-WEB (2007) doi:10.1109/LA-WEB.2007.56

17. Martín, A., Cechich, A., Rossi, G.: Comparing approaches to web accessibility assessment. In: Calero, C., Moraga, M.Á., Piattini, M. (eds.) Handbook of research on web information systems quality, pp. 181–205. Information Science Reference, Hershey (2008)
18. Moreira, A., Araújo, J., Rashid, A.: A concern-oriented requirements engineering model. CAiSE (2005) doi:10.1007/11431855_21
19. Niederhausen, M., Fiala, Z., Kopcsek, N., Meissner, K.: Web software evolution by aspect-oriented adaptation engineering. WSE (2007) doi:10.1109/WSE.2007.4380237
20. PAS 78. Publicly Available Specification: A Guide to Good Practice in Commissioning Accessible Websites, ICS 35.240.30. Disability Rights Commission (DRC) http://shop.bsigroup.com/en/ProductDetail/?pid=000000000030129227 (2006). Accessed 25 January 2010.
21. Plessers, P., Casteleyn, S., Yesilada, Y., De Troyer, O., Stevens, R., Harper, S., Goble C.: Accessibility: a web engineering approach. WWW (2005) doi:10.1145/1060745.1060799
22. Rossi, G., 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)
23. Schauerhuber, A., Wimmer M., Schwinger, W., Kapsammer, E., Retschitzegger, W.: Aspect-oriented modeling of ubiquitous web applications: the aspectWebML approach. ECBS MBD (2007) doi:10.1109/ECBS.2007.20
24. Section 508. Electronic and Information Technology Accessibility Standards http://www.access-board.gov/sec508/508standards.pdf (2000). Accessed 25 January 2010
25. Sommerville, I.: Software engineering, 8th edn. Pearson Education Limited, Harlow (2007)
26. Vilain, P., Schwabe, D., Sieckenius de Souza, C.: A diagrammatic tool for representing user interaction in UML. UML (2000) doi:10.1007/3-540-40011-7_10
27. W3C: Web Content Accessibility Guidelines 1.0. (WCAG 1.0). http://www.w3.org/TR/WAI-WEBCONTENT/ (1999). Accessed 15 April 2009
28. W3C: Web Content Accessibility Guidelines 2.0 (WCAG 2.0). http://www.w3.org/TR/WCAG20/ (2008). Accessed 25 January 2010
29. W3C: HTML Techniques for Web Content Accessibility Guidelines 1.0. http://www.w3.org/TR/WCAG10-HTML-TECHS/ (2000). Accessed 15 April 2009
30. W3C-WAI: Comparison of WCAG 1.0 Checkpoints to WCAG 2.0. http://www.w3.org/WAI/WCAG20/from10/comparison/ (2008). Accessed 25 January 2010.
31. Yesilada, Y., Harper, S., Goble, G., Stevens, R.: DANTE: annotation and transformation of web pages for visually impaired users. WWW (2004) doi.acm.org/10.1145/1013367.1013540
32. Zimmermann, G., Vanderheiden, G.: Accessible design and testing in the application development process: considerations for an integrated approach. Univ. Access Inf. Soc. **7**(1–2), 117–128 (2008)