3. BACKGROUND OF OUR PROPOSAL

3.1 Introducing the Basis

In the following Sections we introduce four key topics that we will use throughout the rest of the work, to make it self-contained. These are: (i) Aspect-Oriented Composition, (ii) Reference Frameworks and Ontologies, (iii) User Interaction Diagrams (UIDs), and (iv) Softgoal Interdependency Graphs (SIGs). Our aim is not to discuss these issues in detail; instead we intend to stress the most important concepts. We also devote a special section to the motivation for using the WCAG 1.0 [45] instead of WCAG 2.0 [46].

3.2 Aspect-Oriented Composition

A concern is an area of interest or focus in a system. Since Dijkstra [13], concerns are the primary criteria for decomposing software into smaller, more manageable and comprehensible parts that have meaning to a software engineer. Examples of concerns include requirements, use cases, features, data structures, quality-of-service issues, variants, intellectual property boundaries, collaborations, patterns and contracts. Thus, Separation Of Concerns (SOC), is a long standing idea that refers to the ability of identifying, encapsulating and manipulating parts of software that are crucial to a particular purpose [13]. Software engineering development methods have been created with this principle in mind. However, traditional paradigms to software development, such as Object-Oriented methods and languages, are not able to modularize crosscutting concerns effectively, because they suffer from a limitation called the "Tyranny of the Dominant Decomposition". This limitation means that they allow modularization in only one way at a time, so they are unable to solve the many kinds of concerns that do no align with that main modularization. In other words, given one out of many possible decompositions of the problem (most of them are core functionality concerns), some sub-problems show, such as non-functional and functional requirements, added after facts, etc., which cannot be modularized. These problems are concerns that cut across many other concerns producing "crosscutting symptoms" resulting into representations --e.g. specifications, classes, code, etc., which are difficult to understand and maintain.

An important issue to underline about this kind of behavior is not only manifested for: (i) a given decomposition, but for all possible decompositions, (ii) a given paradigm, such as object-orientation, also in other paradigms and, (iii) at the implementation stage, also in other stages, such as analysis and design. Usually, these crosscutting symptoms manifest in “scattering” and “tangling” problems. We say that the representation of a concern is scattered over an artifact, when the code for the implementation of the concern’s body is spread out over multiple and different modules or classes rather than localized. While the representation of a concern is tangled within an artifact, when the code for the implementation of the concern’s body is intermixed with code that implements other concerns’ bodies. Scattering and tangling often go together, even though they are very different concepts [17].

Typical examples of such crosscutting concerns are non-functional requirements, such as security, availability, persistency, usability and Accessibility, the main topic of this paper. However, crosscutting concerns can also be functional requirements, such as order auditing, validation, and in the Web engineering domain, tracing the user navigation history [21].

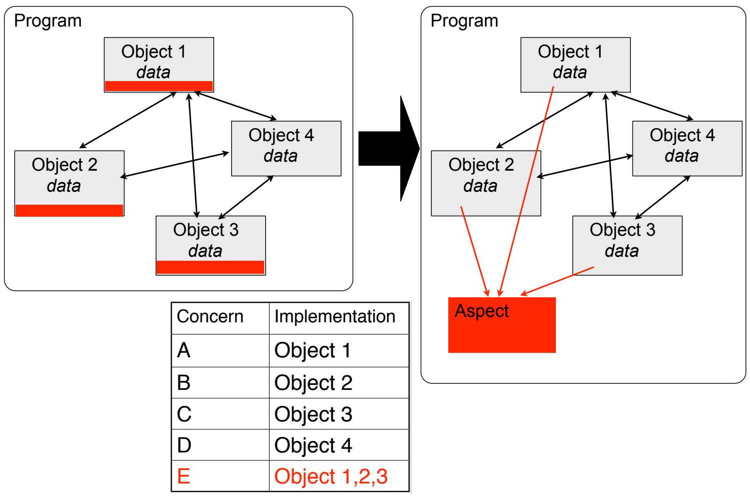
SOC can be supported in many ways, such as by process, by notation, by organization, by language mechanism and, so on. Within the broad theme of SOC, Aspect-Oriented Software Development (AOSD) is distinguished by providing new insight on the separation of crosscutting concerns and in particular leads to the idea that single hierarchical structures are too limiting to effectively separate all concerns in complex systems[36]. AOSD aims at handling such crosscutting concerns at the various levels of the process of software development, by providing means to their systematic identification, modularization and composition [17]. 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. Since aspects are concerns that crosscut a primary or dominant decomposition (other core modules), aspect “weaving” is a composition mechanism that injects aspects into this primary or dominant decomposition.

However, aspects, as well as their compositions, also have an important role to play

[36] AOSD community at http://www.aosd.net/wiki/index.php?title=Main_Page

before the implementation. On one hand, the notion of "early aspects" means it is important to consider aspects early on in the software engineering lifecycle during analysis and design, as opposed to only at the implementation and testing stages. At these early stages of the development process, aspects will allow the modularization of crosscutting concerns that cannot be encapsulated by a single use case, for example, and are typically spread across several of them. Composition, on the other hand, allows the developers to picture the whole system and to identify conflicting situations whenever a concern contributes negatively to others [17].

Traditionally, AOSD has focused mainly on the implementation phase of the software lifecycle since aspects are identified and captured mainly at coding. But aspects have been also applied to former phases as design and even earlier as requirements to cover consistently the entire development process [2] [28].



Concern	Implementation
A	Object 1
B	Object 2
C	Object 3
D	Object 4
E	Object 1,2,3

Figure 3.1: Aspects modularization [4]

3.2.1 Aspectual Implementation: Advices and Pointcuts

Aspect-orientation proposes a fundamentally new kind of modularization that goes beyond generalized procedures: an aspect. An aspect is a module that can localize the implementation of a crosscutting concern. The aspectual decomposition modularizes scattering problems --i.e. one concern in many modules, and tangling problems --i.e. one module, many concerns. Thus, the key to this modularization technique lies in its module composition mechanism. Figure 3.1 shows graphically the idea supporting aspects using an example at the implementation level. While subroutines explicitly invoke the behaviors implemented by other subroutines, aspects have an implicit invocation mechanism [4]. This mechanism that injects aspects into the primary or dominant decomposition is called “aspect weaving”. The implicit invocation mechanism requires that the aspect itself specifies “where or when” it needs to be invoked and also “what” needs to be injected.

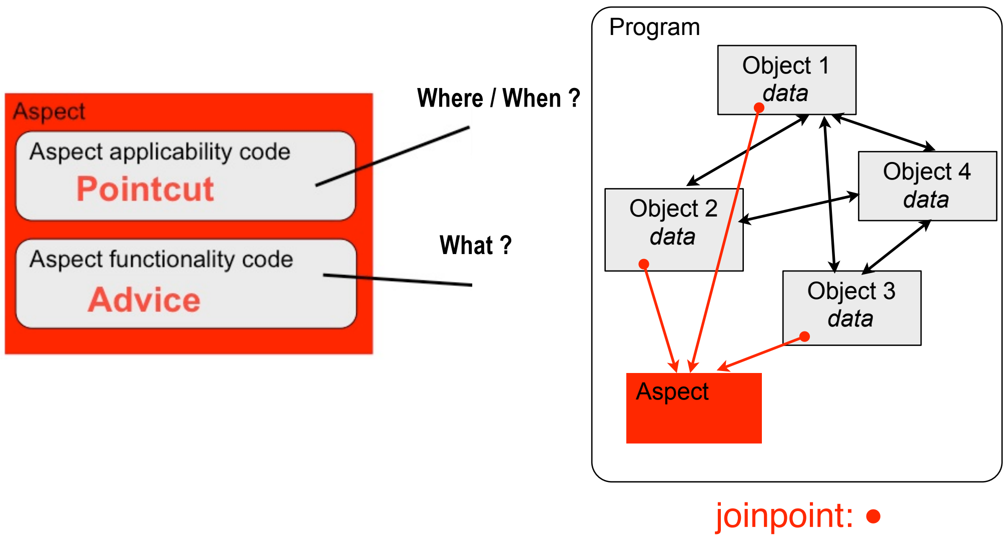


Figure 3.2: Aspects implementation [4]

Consequently, as Figure 3.2 shows, an aspect implementation consists of two conceptually different parts: the aspect functionality code --i.e. aspect functional implementation, and the aspect applicability code –i.e. aspect control over implicit invocation. The aspect functionality code is not essentially different from regular code and is executed when the aspect is invoked. This invocation of the aspect is determined by the aspect applicability code. This code contains statements that specify where or when the aspect needs to be invoked. In standard AOSD terminology, this aspect applicability code is referred to as a “pointcut” expression, which must match a join point, and the aspect functionality code is referred to as the aspect “advice” code. Since a single aspect can consist of multiple different functionalities that need to be invoked from various different places in the code, an aspect implementation can consist of several pointcuts and advice code segments.

3.3 Reference Frameworks and Ontologies

Our approach involves two main elements when designing the user interface towards achieving Accessibility of Web applications. Firstly, a reference framework can serve us as a conceptual structure for making design decisions when building useful user interface models for Accessibility purpose. Secondly, ontologies can provide us with a formal specification for the abstract interface vocabulary. In the following sections, we introduce these two main elements.

3.3.1 Design Decisions within a User Interface Framework

There are many decisions that developers must make during the design of a user interface. As with any complex decision-making process, it is useful to partition the set of decisions into classes and concentrate on the decision in each class, separately. A design decision framework consists of a collection of design decision classes. When decisions in each of the design decision classes are combined, an overall design is synthesized [27]. The criteria for identifying and constructing decision classes are separation, completeness, sufficiency, understandability, independence, reusability and soundness.

We applied in our work the Larson's user interface design decision framework [27] that defines the following five classes:

- *Structural* decision class, which specifies the structure of the end users' conceptual model. These specifications include a description of the conceptual objects that are consumed, produced, and/or accessed by the end users and application functions.
- *Functional* decision class, which specifies functions (operations), which the user can apply to the conceptual objects. Functional decisions determine what requests the users can express and what results the application functions can present to the user.
- *Dialog* decision class, which specifies the content and sequence of information exchange between the user and the application. In this class, the designer specifies the dialog style taking into account: (i) what the units of information exchanged between the user and the application are, (ii) how these units of information are structured into messages exchanged between the user and the application and, (iii)

what the appropriate sequences of message exchanged are. These units of information, which have a formally defined meaning, are called “semantic tokens”.

- *Presentation* decision class, where the designer chooses interaction objects that make up the end users’ interface. Informally, interaction objects are visible widgets on a screen that the user can manipulate to enter lexical tokens and which the user views to obtain lexical tokens. A “lexical token” is a keystroke, mouse movement, or mouse click entered by the user or a character, icon, or elementary sound presented to the user.
- *Pragmatic* decision class, which deals with issues of gesture, space, and hardware devices. Often these decisions are determine by designers in conjunction with ergonomic specialist.

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. As an example, consider decisions involving Accessibility requirements in the case of playing a song’s track at a music Web site. The *Dialog* decision class must describe a sequence of commands for turn-on / turn-off the song’s track. While in the *Presentation* decision class, the designer chooses the appropriate vocabulary and widgets for individualizing these two commands clearly to the user. Finally, in the *Pragmatic* decision class, the designer chooses the hardware, such as a mouse or a touchscreen, for selecting these commands.

Larson's framework [27] gives us a comprehensive and general view that can be instantiated with different conceptual models, such as the approach proposed eleven years later by Baxley in [3]. This proposal describes a universal model of a user interface that can be applied to any interactive medium or product based on the established model of structure-behavior-presentation.

Table 3.1 shows how this early proposal, can be easily mapped to design decision classes introduced by the Larson’s framework to add additional levels of granularity or specificity. For example, Larson’s presentation class (corresponding to Baxley’s presentation tire) can be specified in depth at layout, style and Baxley’s text layers. This can be useful if the design for the user interface under development requires the explicit identification of these components at the presentation model.

Table 3.1: Mapping between Larson's framework [27] and Baxley's model [3]

Baxley's Universal Model of User Interface		**Larson's User Interface Design Decision Framework**
Tires	Layers	Classes
Structure	Conceptual Model	Structural & Functional
	Task Flow	
	Organization Model	
Behaviour	Viewing & Navigational	Dialog
	Editing & Manipulation	
	User Assistance	
Presentation	Layout	Presentation
	Style	
	Text	

3.3.2 An Ontology to share Abstract Interface Vocabulary

Any hypermedia Web application exchange information through its user interface with its environment in order to fulfill a task. The most abstract level is called abstract user interface and focuses on the various types of functionality that can be played by interface widgets with respect to the information exchange between the user and the application.

We applied the Abstract Widget Ontology [36], which provides an abstract interface vocabulary to represent the various types of functionality that can be played by interface widgets with respect to the activity carried out, or the information exchanged between the user and the application. This ontology can be thought of as a set of classes whose instances will comprise a given interface.

As shown in Figure 3.3, an abstract interface widget can be any of the following [36]:

- *SimpleActivator* widget, which represents elements capable of reacting to external events, such as mouse clicks on links or action buttons.
- *ElementExhibitor* widget, which represent elements able to exhibit some type of content, such as text or images.
- *VariableCapture* widget, which represent elements able to receive/capture, the value of one or more variables. As we can see in Figure 3.3, the *VariableCapture* widget generalizes two distinct (sub) concepts. The first one is the ontology (sub) concept *PredefinedVariable*, which represents elements that allow the selection of a subset from a set of predefined values, such as buttons and check boxes; often this selection must be a singleton. The second ontology (sub) concept is the

IndefiniteVariable, which represents elements that allow the user to enter data (previous unknown values) through the keyboard, such as text typed by the user in a text box on a form.

- *CompositeInterfaceElement* widget, which is a composition of any of the abstract interface widget represented by the ontology's previous concepts.

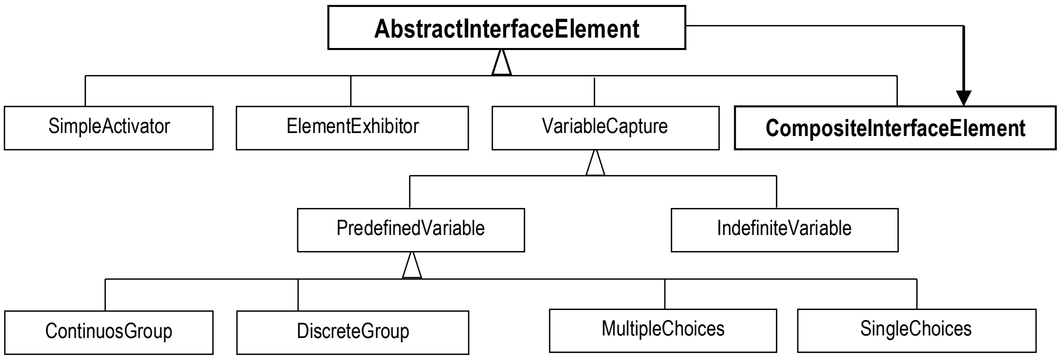


Figure 3.3: Abstract Widget Ontology [36]

It becomes evident from this ontology the essential roles that interface elements play with respect to the interaction --i.e. they exhibit information, or they react to external events, or they accept information. Composite elements allow us to build more complex interfaces out of simpler building blocks [36]. Once the abstract interface model has been defined, each widget is mapped onto a concrete widget to specify the concrete interface model. An abstract interface widget provides a type of functionality to the user by using an interface element, while a concrete interface widget is the actual implementation of that interface element in a given mark-up language or a runtime environment.

Since HTML is the "lingua franca" --i.e. a means of communication between people of different languages for publishing hypertext on the World Wide Web, in Sections 5.3.2 and 5.4 we map these ontology concepts onto HTML elements; this mapping is presented when we describe our model for user interface concerns.

3.4 User Interaction Diagrams

A User Interaction Diagram (UID) [44] is a diagrammatic modeling technique focusing exclusively on the information exchange between the application and the user. UIDs are an outstanding tool to support the communication between different stakeholders during requirements specification and are particularly valuable considering the interactive nature of Web applications. UIDs can be used to enrich the use case models but they are also key graphical tools for linking requirements at later stages of a WE development process to obtain conceptual, navigational and user interface diagrams [43].

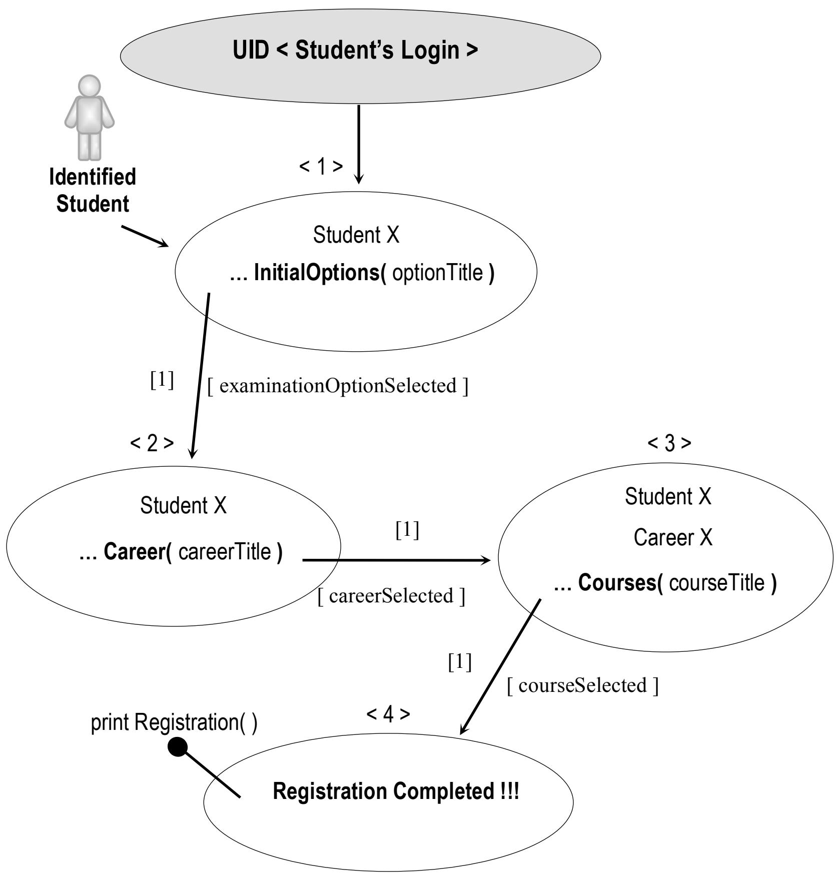


Figure 3.4: A simple UID: Enrolling a Student in an Examination Board given a Course

UIDs are simple state machines, and at the same time an effective instrument to convey the evolution of a Web application process and to support traceability from requirements to later design steps, smoothing the way to implementation. In Figure 3.4 we show a simple UID to express the use case *"Enrolling a Student in an Examination Board given a Course"* in the context of the SIU Guarani registration system.

To ease the comprehension of Figure 3.4, we include here some remarks about the UID's notation. The ellipse represents an interaction between the user and the system and is assigned a number representing its order in the interaction sequence. An ellipse with an arrow without a source particularly recognizes the initial interaction; the results of each subsequence interaction, which cause processing in the system, should be represented as a separate ellipse, connected to the preceding interaction by an arrow. Each ellipse offers content to the user that depends on the interaction sequence of the task represented by the UID. For example, an ellipse can provide the user with any of the following widgets: (i) a data entry i.e-- data entered by the user and graphically represented by a rectangle; (ii) text i.e--descriptive text represented by "XXXX"; (iii) 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 [43] [44].

In the first interaction of Figure 3.4 (indicated by <1> and an incoming arrow), a student already identified at the SIU Guarani system by a previous UID corresponding to the use case *"Login a Student given the Student's ID and Password"*, selects only 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. Notice that this set always has at least two elements and this is because even if the student is enrolled in only one career, the SIU Guarani system offers examination enrolling for admission's courses or career's courses. 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. The complete syntax for UIDs can be found in [44].

3.5 Softgoal Interdependency Graphs

Softgoal Interdependency Graphs (SIGs) have been intensively used in software engineering for modeling non-functional requirements [11] [12]. For example, a framework for integrating non-functional requirements (NFRs) with functional ones in the use case model is proposed in [12]. In this framework, NFRs are represented as "softgoals" to be "satisfied". 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 [12].

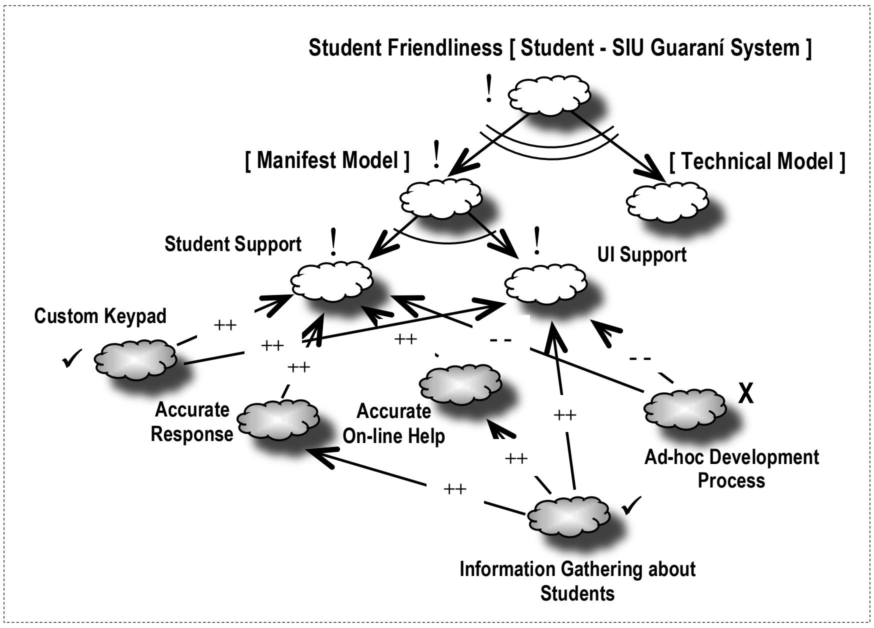


Figure 3.5: Softgoal Interdependency Graph (SIG) for Student Friendliness NFR

In Figure 3.5 we partially depict a SIG for the Student Friendliness softgoal in the context of the SIU Guaraní registration system. 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 3.5, 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 and it is built around task, people and business objects; while the technical model is the model with which developers feel most comfortable and it is built around objects, method, algorithms and data structures [26].

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 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 3.5, the operationalizing softgoal Ad-hoc Development Process is denied.

3.6 Web Content Accessibility Guidelines Documents

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 has been used worldwide since 1999 as a reference material or cited as a normative from many other Accessibility documents in the world [34] [38] [40]. Many tools and approaches also have implemented it.

Although the WCAG 2.0 has been released in December 2008 and it is a fact that so far the rate of adoption has been relatively slow. For example, though it appears that within UK government departments there is a growing acceptance that websites under development should conform to WCAG 2.0, the official government policy still remains WCAG 1.0. As another example, in Germany, despite not using the WCAG, all public websites are beginning to use the usability regulation which incorporates WCAG 1.0 and migration of the Accessibility national guideline to WCAG 2.0 is just beginning; meanwhile in Spain, where any rule specified by legislation refers to a national standard based on WCAG 1.0, as far as we know, there is no regulation oriented toward WCAG 2.0 yet. Finally, since Section 508 [38] is undergoing a revision over the next couple of years [42], we have to wait approximately until 2011-2012 for the WCAG 2.0 to be harmonized into this Accessibility standard. At this point we emphasize that we are pre-supporting new issues addressed by W3C-WAI, but in light of how the migration of Accessibility regulations toward WCAG 2.0 is evolving, we think that the WCAG 2.0 is still in its infancy and therefore some time must pass before it is widespread adopted.

As we already mention in Section 2.1, the situation in Argentina is less developed, since Web Accessibility is an issue that has been recently included in the State's agenda. The legislation 26.653 called "Guía de Accesibilidad para Sitios Web del Sector Público Nacional[37]", which adheres to WCAG 1.0 document, was approved by Resolution 69/2011 on June 27th 2011. In August 2011, Argentina became a member of the W3C[38]. As argentine citizens committed with Accessibility, we have much expectation about this first steps towards an inclusive government Web for all.

In addition to the reasons stated above, we selected the WCAG 1.0 because it is a mature, committed to all possible Accessibility barriers and stable document version and part of a series of valuable and related Accessibility guidelines published by the W3C-WAI [50] with which WCAG 1.0 can be applied in conjunction. We revisit this discussion in Section 7.3.1 where we also provide some insights on how we upgraded our approach to WCAG 2.0 [46].

[37] Access to Public Information by Law 26.653 at http://www.infoleg.gov.ar/infolegInternet/anexos/175000-179999/175694/norma.htm

[38] Argentina became a member of the W3C at http://www.puntogov.com/nota.asp?nrc=2641