5. APPLYING OUR PROPOSAL

5.1 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 is a main factor for all users but plays a key role for students with disabilities. In the spirit of such systems, we define the case study to apply our Aspect-Oriented approach, reusing the Student's login and the University home page examples, shown in Figures 1.1 and 2.1, respectively.

As Figure 5.1 shows, we propose a case study of 3 (three) level-deep navigation and 2 (two) optional anchors to get some help for data inputs ID and Password at the login Web page. The first level, shown in Figure 5.1 (a), is the student's University home page where the student selects the link to his/her respective Faculty site from a group of consecutive and related links. We highlight that we have already presented and explained this page example in Section 2.2.1 (as shown in Figure 2.1), since it is the one used to exemplify the related work. The second level, shown in Figure 5.1 (b), is the student's Faculty page that provides information about this institution among other functionalities and, offers a link to the SIU Guaraní student registration system. Finally, the third level, shown in Figure 5.1 (c), is the student's login page example, which we also have already presented and described in Section 1.1 (as shown in Figure 1.1) and then in Section 4.2 by the use case *"Login a Student given the Student's ID and Password"*. From this third level, the student has the ability to browse for getting help to ID and/or Password if he/she fails to login to the system. These two pages, shown in Figure 5.1 (d), provide students with some helpful information and the chance to return to the login Web page.

To carry out the implementation of our approach clearly, in Section 5.2 we follow the step-by-step process as we described in Chapter 4 and depicted in Figure 4.1.

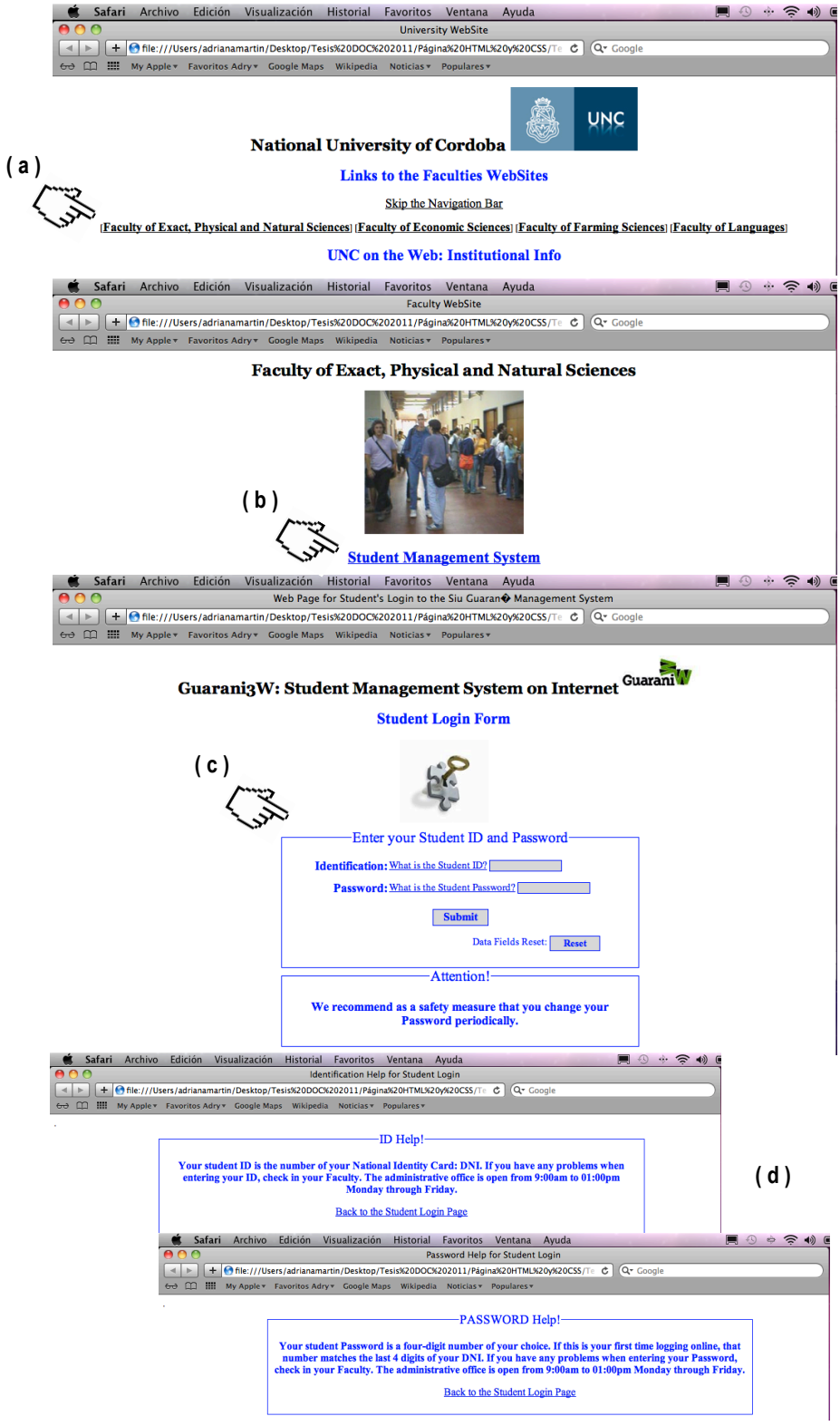

Figure 5.1: A Case Study

5.2 Our Proposal Step-by-Step on the Field

STEP 1. As highlighted in Figure 4.1 (1), we propose to manage the requirements of the case study to identify those that involve user-system interaction. Specifically, we focus on those requirements at the user interface (UI) that let the students reach the login Web page browsing through the three level-deep navigation, which we defined above for the case study, as follow:

- **Level 1 – The Student's University home page**. The corresponding UI design provides the interface widgets[43] that allow the student to choose the anchor to his/her Faculty from a set of Faculty names, which make up the student's University. In this case, as Figure 5.1 (a) shows, the UI design must include at least, for each link to Faculties, a widget of the type *SimpleActivator* at the abstract interface model mapped to the concrete interface model on a widget of the type HTML *link*. Also, as shown in Figure 5.1 (a), the UI design must include an extra link to skip the navigation bar. All these widgets are grouped together into a *CompositeInterfaceElement* at the abstract interface model and mapped to a concrete interface model on HTML *related links*. To complete de understanding of this mapping, refer to the *association table* for the HTML *link* and *button* group introduced in Section 4.5.2 by Table 4.2.
- **Level 2 – The Student's Faculty page**. Basically, as Figure 5.1 (b) shows, the UI design must include, for the link to the SIU Guaraní registration system, a clear widget of the type *SimpleActivator* at the abstract interface model mapped to the concrete interface model on a widget of the type HTML *link*. To complete de understanding of this mapping, refer to the *association table* for the HTML *link* and *button* group introduced in Section 4.5.2 by Table 4.2.
- **Level 3 – The Student's Login page**. The corresponding UI design provides the interface widgets that allow the student to login the SIU Guarani registration system. In this case, as Figure 5.1 (c) shows, the UI design must include at least, for the student's identification purpose, two widgets of the type *IndefiniteVariable* at the

[43] To make this Step-by-Step explanation clearer, whenever we use "widgets" without specifying of which type, we are referring to both, abstract and concrete ones.

abstract interface model mapped to the concrete interface model on two widgets of the type HTML *text field*. The mission of these widgets is to receive the student's ID 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 HTML *related controls* to create a form. To complete the understanding of this mapping, refer to the *association table* for the HTML *control* group introduced in Section 4.3.2 by Table 4.1.

- **Levels 1, 2 and 3.** These three UI designs also provide text and images for student's information purpose. In this case, the UI designs must include three widgets of the type *ElementExhibitor* at the abstract interface models mapped to the concrete interface models on three widgets of the type HTML *image*. The mission of these widgets is to include the University logo (as shown in Figure 5.1 (a)), the Faculty picture (as shown in Figure 5.1 (b)), and the image of the key-lock (as shown in Figure 5.1 (c)). To complete de understanding of this mapping, refer to the *association table* for the HTML *text* and *non-text* group introduced in Section 4.5.2 by Table 4.3.

- **Level 4 – Help pages (Optional).** These two UI designs provide some instructive text about the data inputs ID and Password. In this case, as Figure 5.1 (d) shows, each UI design must include, for allowing the student to go back to the login page, a clear widget of the type *SimpleActivator* at the abstract interface model mapped to the concrete interface model on a widget of the type HTML *link*. To complete de understanding of this mapping, refer to the *association table* for the HTML *link* and *button* group introduced in Section 4.5.2 by Table 4.2.

  It is important to highlight that browsing these pages is optional and therefore, if the student follows these help links, his/her decision will produce a different navigation path. At this point, we are focused on the UI models because, undoubtedly, is at the UI level where Accessibility barrier finally show; but in Section 6.3, we will revisit this argument to discuss the potential of our approach to deal with situations that could affect the Accessibility of the navigational models.

- **Levels 1, 2, 3 and 4.** Also, these four UI designs must consider widgets of the type *ElementStyling* at the abstract interface models mapped to the concrete interface

models on widgets of the type HTML *formatting & positioning*. The mission of these widgets is to define the appearance of the content --i.e. the look-&-feel of the UI. To complete de understanding of this mapping, refer to the *association table* for the HTML *frame* and *style sheet* group introduced in Section 4.5.2 by Table 4.5.

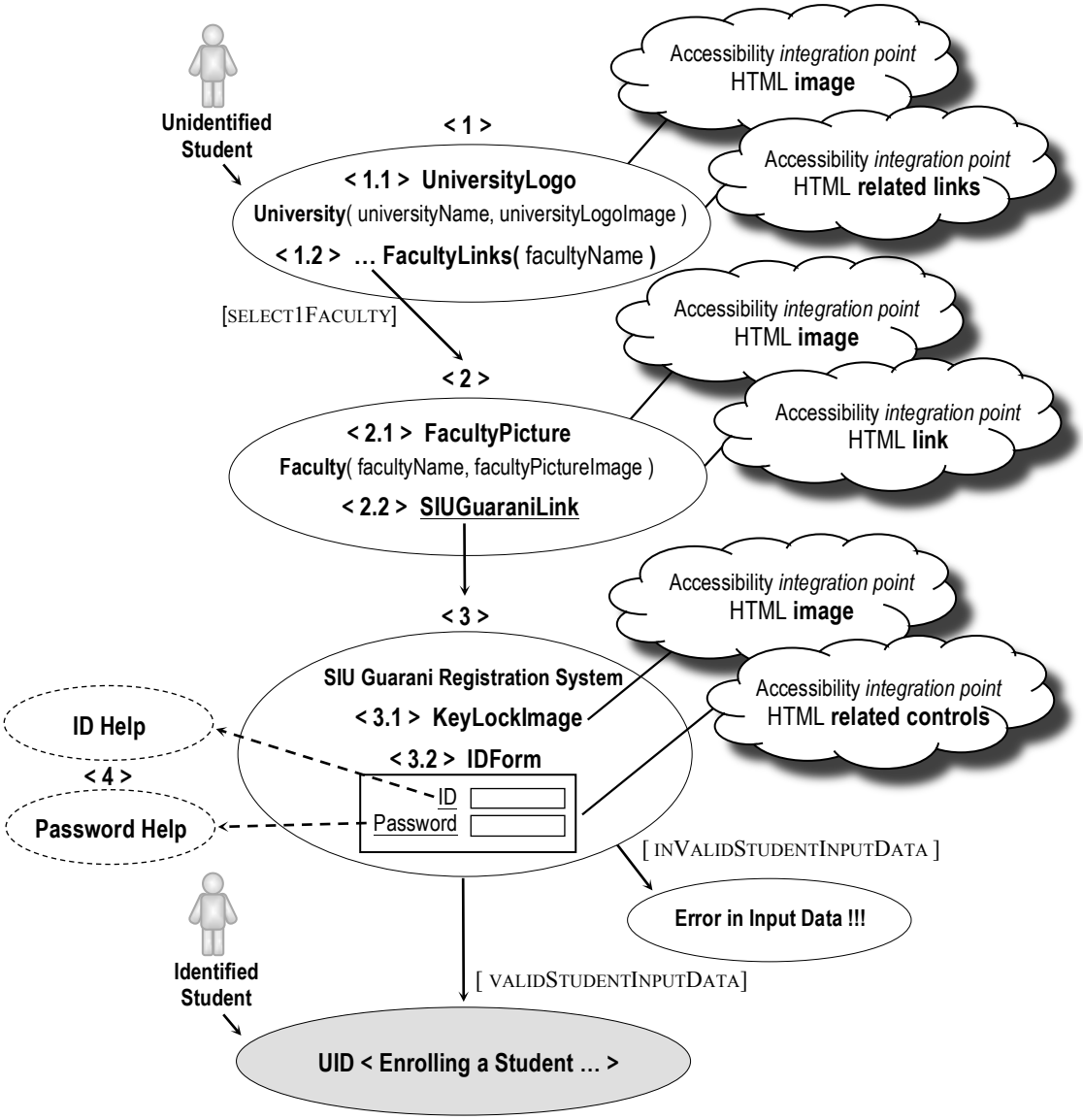

Figure 5.2: UID with integration points for the Case Study

STEP 2. As highlighted in Figure 4.1 (2.1) and (2.2), for specifying Accessibility concerns, we encourage the early capture of these Accessibility requirements by applying the UID and SIG conceptual tools.

STEP 2.1. We develop the UID diagram with *integration points* for the case study. As shown in Figure 5.2, at the UID interactions <1>, <2>, <3> and <4>, we outline the *integration points* that remain the Accessibility concerns that are crucial at each navigation level described above, as follow:

- **Level 1 – UID Interaction <1>.** We set <1.2> *integration point* for the HTML HTML *related links* corresponding to the links to Faculties.
- **Level 2 – UID interaction <2>.** We set <2.2> *integration point* for the HTML *link* corresponding to the link to the SIU Guarani registration.
- **Level 3 – UID interaction <3>.** We set <3.2> *integration point* for the HTML *related controls* corresponding to the form for the student's identification. The Accessibility concerns, which are required by the related HTML *text fields* that make up the form, are relevant to a successful login information exchange between the student and the application, during the execution of the identification function.
- **Levels 1, 2 and 3 – UID interactions <1, 2, 3>.** We set <1.1>, <2.1> and <3.1> *integrations points* for the HTML *images* corresponding to the images of the University logo, the Faculty picture and the key-lock, respectively.
- **Level 4 – UID interactions <4> (Optional).** As we already said before, from **Level 3**, it is possible to browse to get some help for data inputs ID and Password. Although in Figure 5.2 we have not included details about the *integration points* required for these pages, we can set them for the HTML *text* and the HTML *link* corresponding to a helpful text and a link that clearly allows the student to return to the login Web page, respectively.
- **Levels 1, 2, 3 and 4 – UID interactions <1, 2, 3, 4>.** In Figure 5.2 we have not set *integrations points* for the HTML *formatting & positioning* to make simpler the understanding of the diagram and because, as we will see in **Step 2.2**, these are Accessibility concerns required in general for all Web pages.

STEP 2.2. We instantiate the SIG *template* for the Accessibility *integration points* outlined by the UID interactions <1>, <2>, <3> and <4> in **Step 2.1**, to identify WCAG 1.0 Accessibility requirements. In Section 3.5, we presented the basis of the SIG's notation and vocabulary and then, in Section 4.3.2, we explained how we extended this

conceptual tool into a template to handle the Accessibility concerns. At this template, the focus of the Accessibility softgoal is highlighted into the root light cloud. 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. At this point, note that the *association tables* presented in Sections 4.3.1 and 4.5.2 help to the SIG instantiation process. Applying the SIG *template* for Accessibility, we develop the SIG diagrams at each navigation level, as follow:

- **Level 1 – SIG diagram at the UID interaction <1>.** As shown in Figure 5.3, we focus the main Accessibility softgoal on the UID interaction (U-UI) <1> called HTML University home. From this root, we define an Accessibility softgoal for the UID interaction component (U-UIc) <1.2> FacultyLinks, to help to accessible related links for all the students, including those with disabilities. In this case, to support the SIG instantiation process, we use Table 5.2 for the HTML *link* and *button* group, since the Accessibility softgoal is defined for the HTML *related links* element to Faculties. Next, we explain the refinement process for the SIG instantiation at the UID interaction <1>.

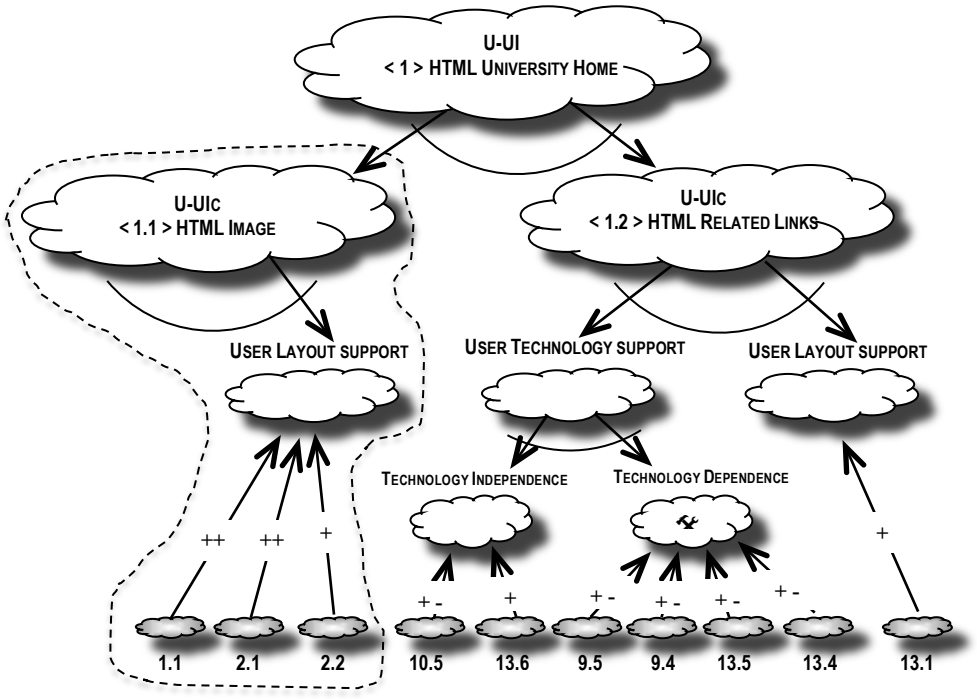

Figure 5.3: SIG instantiation for the UID interaction <1>

Firstly, looking at the user technology support branch in Figure 5.3, a distinction between "technology independence" and "technology dependence" is made in concordance with the distinction made in Section 4.3.2. To help to the universal access of devices to the HTML *related links* 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 guidelines 10 and 13 for checkpoints 10.5 and 13.6 compliance are required. Otherwise for "technology dependence", satisfying goals related to guidelines 9 and 13 for checkpoints 9.4 and 9.5; 13.5 and 13.4 compliance are required. Now looking at the user layout support, satisfying goals related to guideline 13 for checkpoint 13.1, compliance is required for the HTML *related links* element.

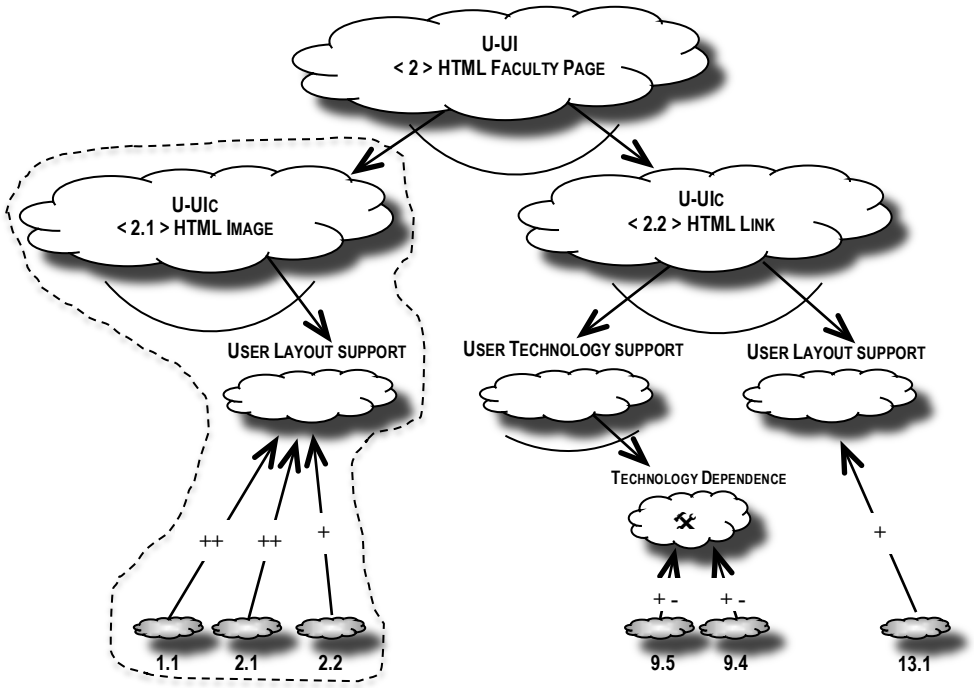

Figure 5.4: SIG instantiation for the UID interaction <2>

- **Level 2 – SIG diagram at the UID interaction <2>.** As shown in Figure 5.4, we focus the main Accessibility softgoal on the UID interaction (U-UI) <1> called HTML Faculty page. From this root, we define an Accessibility softgoal for the UID interaction component (U-UIc) <2.2> SIUGuaraniLink, to help to an accessible link. Here, to support the SIG instantiation process, we also use Table 5.3 for the HTML

link and *button* group, since the Accessibility softgoal is defined for the HTML *link* element to the SIU Guarani registration system. Next, we explain the refinement process for the SIG instantiation at the UID interaction <2>.

Firstly, looking at the user technology support branch in Figure 5.4, "technology dependence", for satisfying goals related to guideline 9 for checkpoints 9.4 and 9.5, compliance are required for the HTML *link* element. Now looking at the user layout support, for satisfying goal related to guideline 13 for checkpoint 13.1, compliance is required for the HTML *related links* element.

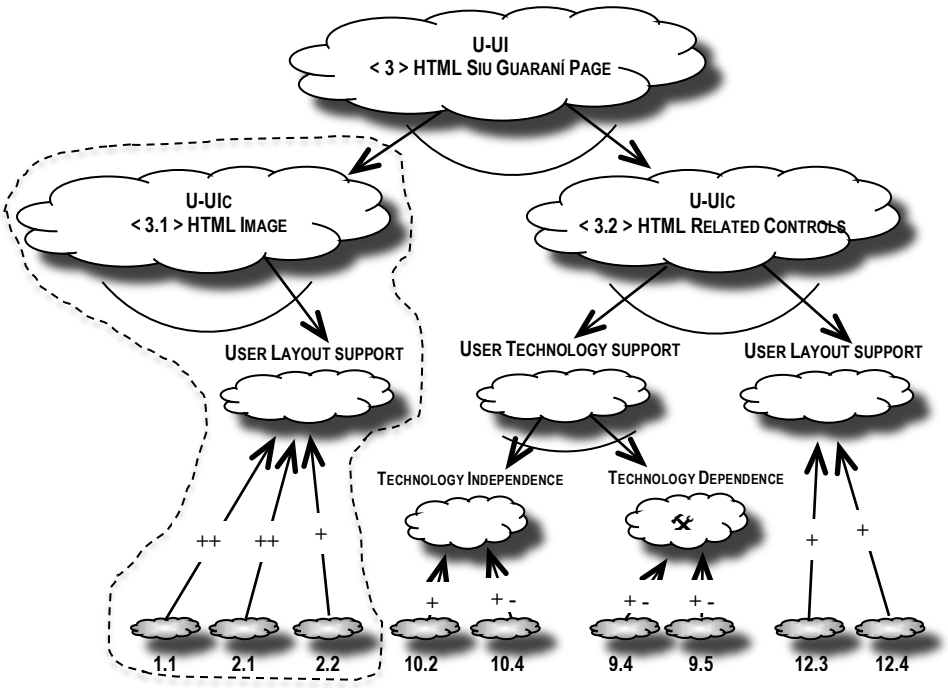

Figure 5.5: SIG instantiation for the UID interaction <3>

- **Level 3 – SIG diagram at the UID interaction <3>.** As shown in Figure 5.5, we focus the main Accessibility softgoal on the UID interaction (U-UI) <3> called HTML SIU Guarani page. From this root, we define an Accessibility softgoal for the UID interaction components (U-UIc) <3.2> IDForm, to help to accessible related controls. In this case, to support the SIG instantiation process, we use Table 5.1 for the HTML *control* group, since the Accessibility softgoal is defined for the HTML *related controls* element, which is a form composed of two HTML *text fields* for student identification purpose. Next, we explain the refinement process for the SIG instantiation at the UID interaction <3>.

Firstly, looking at the user technology support branch in Figure 5.5, we chose an AND-decomposition, as we already did at the SIG instantiation at UID interaction <1> and for the same reasons. For "technology independence", for satisfying goals related to guideline 10 for checkpoints 10.2 and 10.4, compliance are required. Otherwise for "technology dependence", for satisfying goals related to guideline 9 for checkpoints 9.4 and 9.5, compliance are required. Now looking at the user layout support, for satisfying goals related to guideline 12 for checkpoint 12.3 and 12.4, compliance are required for the HTML *related controls* element.

- **Levels 1, 2 and 3 – SIG diagrams at UID interactions <1, 2, 3>.** As shown in Figures 5.3, 5.4 and 5.5, we focus the main Accessibility softgoals on the UID interactions (U-UI) <1, 2, 3>. From these roots, we define Accessibility softgoals for the UID interaction components (U-UIc) <1.1> UniversityLogo, <2.1> FacultyPicture and <3.1> KeyLockImage to help to accessible HTML *image* elements at each page. In this case, to support the SIG instantiation process, we use Table 5.3 for the HTML *text* and *non-text* group, since these Accessibility softgoals are defined for the HTML *image* elements of the University logo, the Faculty picture and the key-lock respectively. Next, we explain the refinement process for the SIG instantiation at the UID interactions <1, 2, 3>.

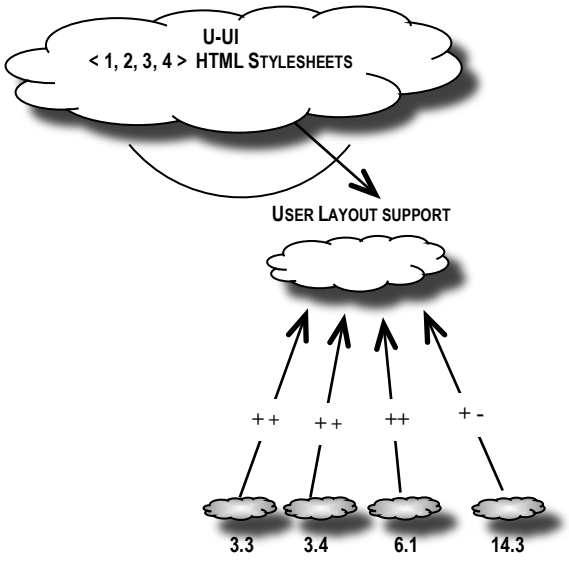

Figure 5.6: SIG instantiation for the UID interactions <1, 2, 3, 4>

Looking at the user layout support branches in Figures 5.3, 5.4 and 5.5, **for** satisfying goals related to guidelines 1 and 2 for checkpoints 1.1, 2.1 and 2.2,

compliance are required for the HTML *image* elements. In Section 4.1, we have already said, that there are situations in which we can develop artifacts once and then reused them, as they are required; at **Step 2** in Figure 4.1 (2.1) and (2.2), we have indicated the reuse capability of our approach with input/output arrows. Clearly, this is one of those situations, since the Accessibility softgoal for the HTML *image* element can be modeled once and then applied for the SIG instantiation, as they are required. As Figures 5.3, 5.4 and 5.5 show, we surrounded with dotted lines the UID interaction components (U-UIc) <1.1>, <2.1> and <3.1> for the HTML *image* elements to highlight the reusable artifact applied to the SIG diagrams of the case study.

- **Level 4 – SIG diagram at UID interactions <4> (Optional).** At this level, we proceed in the same way as for the previous levels. We do not give details about this optional level, because we consider it doesn't provide new knowledge about developing the SIG diagrams for Accessibility concerns.
- **Levels 1, 2, 3 and 4 – SIG diagram at UID interactions <1, 2, 3, 4>.** As shown in Figure 5.6, we focus the main Accessibility softgoal on the UID interactions (U-UI) <1, 2, 3, 4> called HTML Stylesheets. Here, to help the SIG instantiation process, we use Table 5.5 for the HTML *frame* and *style sheet* group, since the Accessibility softgoals are defined for the HTML *style sheet* elements to provide formatting and positioning support to the user layout. Next, we explain the refinement process for the SIG instantiation at the UID interactions <1>, <2>, <3> and <4>.

  Looking at the user layout support branch in Figure 5.6, for satisfying goals related to guidelines 3, 6 and 14 for checkpoints 3.3 and 3.4, 6.1, 14.3, compliance are required for the HTML *style sheet* element.

STEP 3. As highlighted in Figure 4.1 (3), for the user interface design activity, we exploit the Accessibility knowledge captured and organized by SIG diagrams in **Step 2.2**. The purpose here is to find out how WCAG 1.0 Accessibility concerns "crosscut" the user interface widgets (abstract and concrete ones). In order to make our discussion clear, we focus on explaining 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 each HTML element corresponding to an abstract interface ontology

widget. 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 elements at the user interface, but also allow Aspects to be modeled and instantiated appropriately to avoid "scattering" and "tangling" problems. Then Aspects can be seamless injected by the "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. As shown in Figure 4.1 (3.1), we work on the abstract user interface required at each navigation level, as follow:

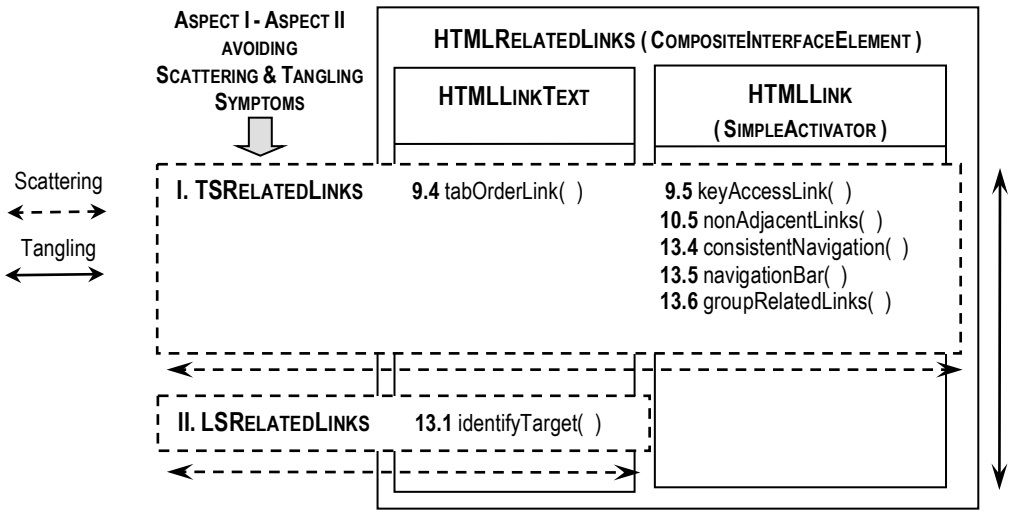

Figure 5.7: SIG's operationalizing goals (WCAG 1.0 checkpoints) crosscutting an HTML *related links* element (Concrete Interface Widget) corresponding to a *CompositeInterfaceElement* (Abstract Interface Widget)

- **Level 1 – UI model at UID interaction <1>.** As shown in Figure 5.7 through a diagram similar to UML, whenever there is an HTML *related links* element at the user interface model, Aspect I "TSRelatedLink" and Aspect II "LSRelatedLinks", focused on solving technology and layout Accessibility issues respectively, are injected to avoid the "scattered" and "tangling" nature of Accessibility checkpoints 9.4 and 9.5, 10.5, 13.4 and 13.5, 13.6 and 13.1 over HTML *related links* classes.

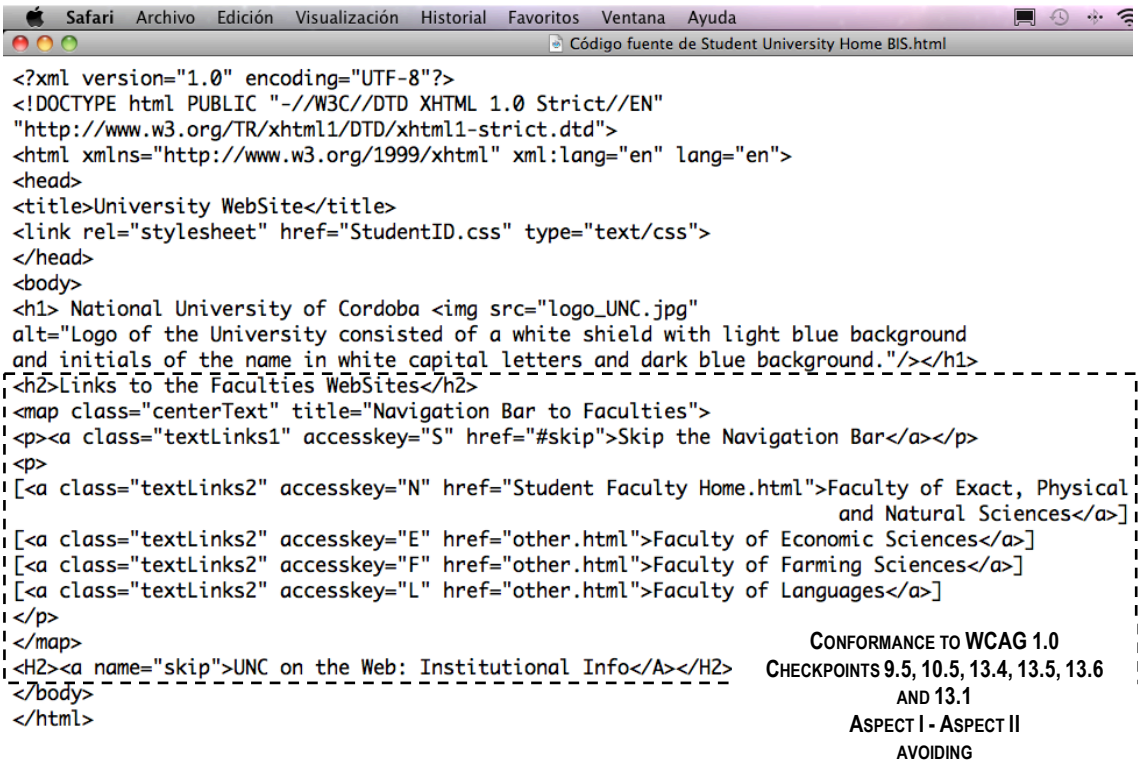

Figure 5.8: Accessible HTML code as a result of a "seamless" injection of Aspects I and II in the UI model at UID interaction <1>

The addition of Aspect I "TSRelatedLinks" and Aspect II "LSRelatedLinks" reminds later, at the implementation of the concrete interface model (as shown by Figure 4.1 (4.1), conformance to the following Accessibility concerns for each HTML *related links* element: (i) creating a logical tab order and/or providing keyboard shortcuts for links, (ii) including non-link, printable characters (surrounded by spaces) between adjacent links, (iii) using navigation mechanisms in a consistent manner and providing navigation bars to highlight and give access to the navigation mechanism, (iv) grouping related links, identifying the group and providing a way to bypass the group and, (v) clearly identifying the target of each link. Figure 5.8 shows the accessible HTML corresponding to the student's University home example, whose screenshot is shown in Figures 2.1 and 5.1 (a).

- **Level 2 – UI model at UID interaction <2>.** As shown in Figure 5.9 through a diagram similar to UML, whenever there is an HTML *link* element at the user interface model, Aspect I "TSLink" and Aspect II "LSLink", focused on solving technology and layout Accessibility issues respectively, are injected to avoid the

"scattered" and "tangling" nature of Accessibility checkpoints 9.4 and 9.5, and 13.1 over HTML *link* classes.

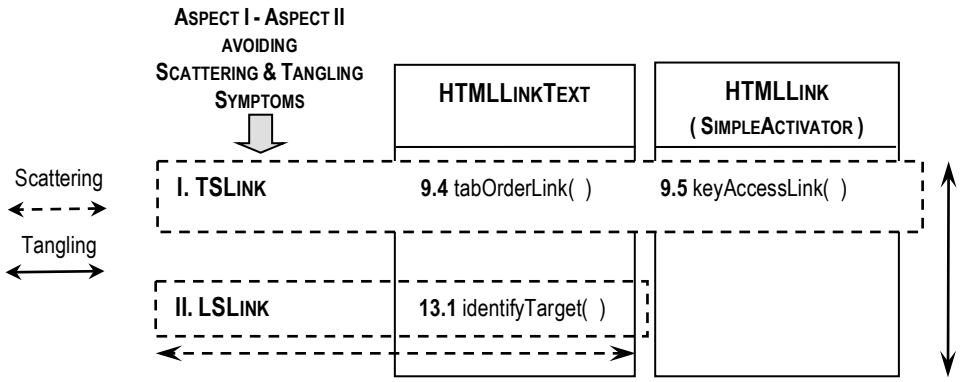

Figure 5.9: SIG's operationalizing goals (WCAG 1.0 checkpoints) crosscutting an HTML *link* element (Concrete Interface Widget) corresponding to a *SimpleActivator* (Abstract Interface Widget)

The addition off Aspect I "TSLink" and Aspect II "LSLink" reminds later, at the implementation of the concrete interface model (as shown by Figure 4.1 (4.1)), conformance to the following Accessibility concerns for each HTML *link* element: (i) creating a logical tab order and/or providing keyboard shortcuts for links and, (ii) clearly identifying the target of each link.

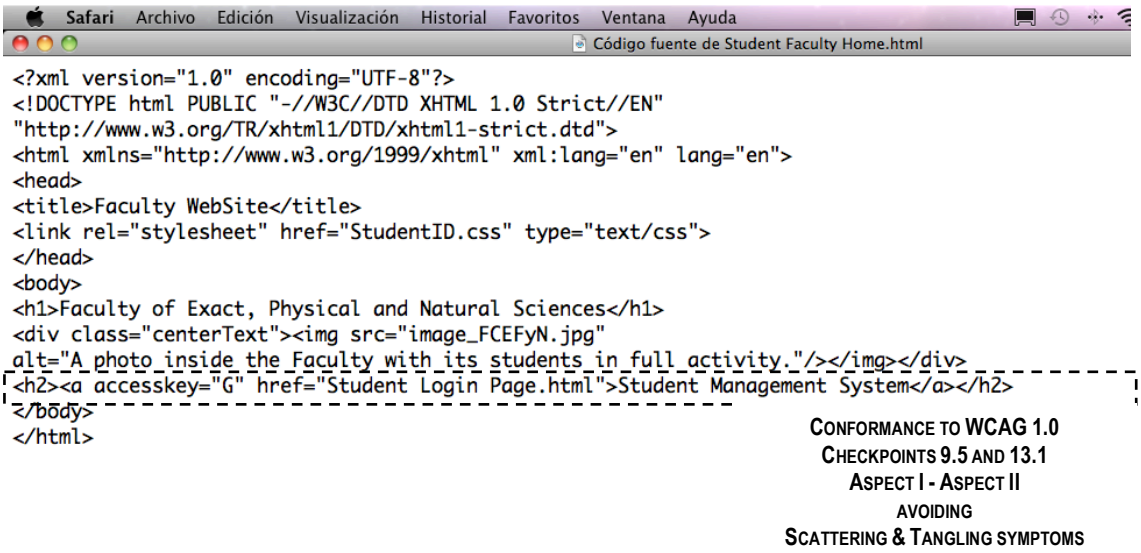

Figure 5.10: Accessible HTML code as a result of a "seamless" injection of Aspects I and II in the UI model at UID interaction <2>

Figure 5.10 shows the accessible HTML code corresponding to the student's Faculty page example, whose screenshot is shown in 5.1 (b).

- **Level 3 – UI model at UID interaction <3>.** As shown in Figure 5.11 through a diagram similar to UML, whenever there is an HTML *related controls* element, which in this case comprises two HTML *text field* elements at the user interface model, Aspect I "TSRelatedControls" and Aspect II "LSRelatedControls", focused on solving technology and layout Accessibility issues respectively, are injected to avoid the "scattered" and "tangling" nature of Accessibility checkpoints 9.4 and 9.5, 10.2 and 12.4, 10.4 and 12.3 and over HTML *related controls* classes.

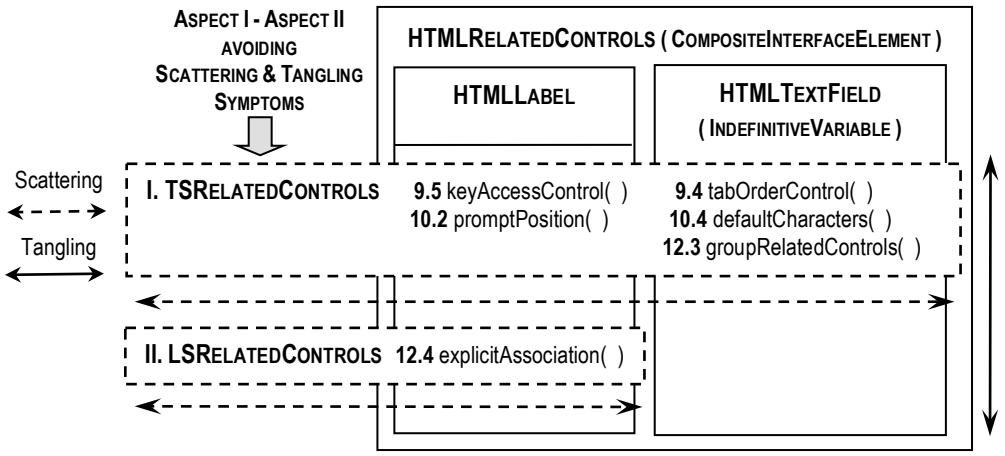

Figure 5.11: SIG's operationalizing goals (WCAG 1.0 checkpoints) crosscutting an HTML *related controls* element (Concrete Interface Widget) corresponding to a *CompositeInterfaceElement* (Abstract Interface Widget)

The addition off Aspect I "TSRelatedControls" and Aspect II "LSRelatedControls" reminds later, at the implementation of the concrete interface model (as shown by Figure 4.1 (4.1)), conformance to the following Accessibility concerns for each HTML *related controls* element: (i) creating a logical tab order and/or providing keyboard shortcuts for controls, (ii) supporting explicit association between HTML *label* elements and controls, (iii) handling empty controls correctly by including default, place-holding characters and, (iv) grouping related controls with HTML *fieldset* and *legend* elements. Figure 5.12 shows the accessible HTML code corresponding to the student's login page example, whose screenshot is shown in Figures 1.1 and 5.1 (c).

Figure 5.12: Accessible HTML code as a result of a "seamless" injection of Aspects I and II in the UI model at UID interaction <3>

- **Level 1, 2 and 3 – UI models at UID interactions <1, 2, 3>.** As shown in Figure 5.13 through a diagram similar to UML, whenever there is an HTML *image* element, Aspect II "LSImage", focused on solving layout Accessibility issues, is injected to avoid the "scattered" nature of Accessibility checkpoints 1.1, 1.2 and 2.2 over HTML *image* classes.

  The addition of Aspect II "LSImage" reminds later, at the implementation of the concrete interface models (as shown by Figure 4.1 (4.1)), conformance to the following Accessibility concerns for each HTML *image* element: (i) adding a text equivalent for every image with a HTML *alt-text* element and, (ii) not relying on images' color alone to convey information. Figures 5.8, 5.10 and 5.12 show the

accessible HTML corresponding to the student's University home page, the Faculty page and the login page examples, whose screenshot are shown in Figures 5.1 (a), 5.1 (b) and 5.1 (c), respectively. As we can see in these HTML files, all the HTML *image* elements have their corresponding text equivalent.

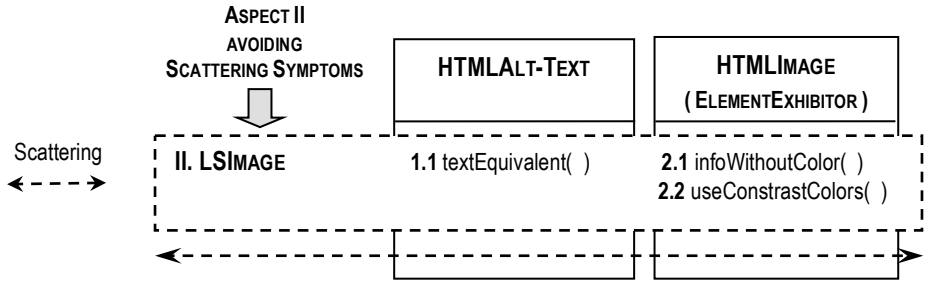

Figure 5.13: SIG's operationalizing goals (WCAG 1.0 checkpoints) crosscutting an HTML *image* element (Concrete Interface Widget) corresponding to an *ElementExhibitor* (Abstract Interface Widget)

- **Level 4 – UI models at UID interaction <4> (Optional).** At this level, we proceed in the same way as for the previous levels. We do not give details about this optional level, because we consider it doesn't provide new knowledge about developing the user interface models.

- **Level 1, 2, 3 and 4 – UI models at UID interactions <1, 2, 3, 4>.** As shown in Figure 5.14 through a diagram similar to UML, whenever there is an HTML *style sheet* element, Aspect II "LSStylesheet" focused on solving layout Accessibility issues, is injected to avoid the "scattered" nature of Accessibility checkpoints 3.3, 3.4, 6.1 and 14.3 over HTML *style sheet* classes.

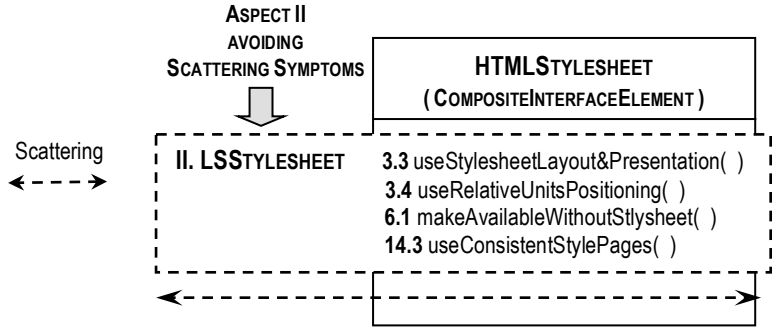

Figure 5.14: SIG's operationalizing goals (WCAG 1.0 checkpoints) crosscutting an HTML *style sheet* element (Concrete Interface Widget) corresponding to a *CompositeInterfaceElement* (Abstract Interface Widget)

The addition of Aspect II "LSStylesheet" reminds later, at the implementation of the concrete interface models (as shown by Figure 4.1 (4.1)), conformance to the following Accessibility concerns for each HTML *style sheet* element: (i) using style sheets to control page layout and presentation, (ii) using relative rather than absolute units in markup language attribute values and style sheet property values, (iii) organizing documents so they may be read without style sheets and, (iv) creating a style of presentation that is consistent across pages. The HTML pages corresponding to the student's University home page, the Faculty page, the login page and the help pages examples, whose screenshot are shown in Figures 5.1 (a), 5.1 (b), 5.1 (c) and 5.1 (d) respectively, keep a consistent styling across pages. As we can see in Figures 5.8, 5.10 and 5.12, for formatting and positioning purpose, these pages use an HTML *style sheet* element.

STEP 4. As highlighted in Figure 4.1 (4), for the user interface developing activity we exploit the aspects applied for solving Accessibility crosscutting concerns discovered in **Step 3**. As another way of illustrating how these aspects were seamless injected in an abstract user interface to obtain a concrete user interface (at the design level) and then an accessible and well formed HTML at the implementation level, we can express the Accessibility concerns conveyed by aspects using a pseudo-code language. We provide some examples for each level defined for the case study in Figure 5.1, as follow:

- **Level 1 – Aspect I and Aspect II in the UI model at UID interaction <1>.**

  ASPECT I. TSRELATEDLINKS

  POINTCUT ALL INTERFACE WIDGETS WITH CompositeInterfaceElement.SimpleActivator == HTML *related links*
  PROPERTY ADVICE ADD ACCESSIBILITY CONDITIONS
  9.4 tabOrderLink == HTML *tabindex* element ∧ **9.5** keyAccessLink == HTML *accesskey* element ∧
  10.5 nonAdjacentLinks == HTML *printable characters* as "[" and "]" ∧
  13.4 consistentNavigation == W3C *Core Techniques* for navigation ∧
  13.5 navigationBar AND **13**.6groupRelatedLinks == HTML *map* element.

  ASPECT II. LSRELATEDLINKS

  POINTCUT ALL INTERFACE WIDGETS WITH CompositeInterfaceElement.SimpleActivator == HTML *related links*
  PROPERTY ADVICE ADD ACCESSIBILITY CONDITION **13.1** identifyTarget == HTML *clear link text* OR HTML *tittle* element.

- **Level 2 – Aspect I and Aspect II in the UI model at UID interaction <2>.**

  ASPECT I. TSLINK

  POINTCUT ALL INTERFACE WIDGETS WITH SimpleActivator == HTML *link*

  PROPERTY ADVICE ADD ACCESSIBILITY CONDITIONS

  9.4 tabOrderLink == HTML *tabindex* element ∧ **9.5** keyAccessLink == HTML *accesskey* element.

  ASPECT II. LSLINK

  POINTCUT ALL INTERFACE WIDGETS WITH SimpleActivator == HTML *link* PROPERTY ADVICE ADD ACCESSIBILITY CONDITION **13.1** identifyTarget == HTML *clear link text* OR HTML *tittle* element.

- **Level 3 – Aspect I and Aspect II in the UI model at UID interaction <3>.**

  ASPECT I. TSRELATEDCONTROLS

  POINTCUT ALL INTERFACE WIDGETS WITH CompositeInterfaceElement.IndefiniteVariable == HTML *related controls*

  PROPERTY ADVICE ADD ACCESSIBILITY CONDITIONS

  9.4 tabOrderControl == HTML *tabindex* element ∧ **9.5** keyAccessControl == HTML *accesskey* element ∧

  10.2 promptPosition == HTML *for* element ∧

  10.4 defaultCharacters == HTML *value* element ∧

  12.3 groupRelatedControls == HTML *fieldset* element AND HTML *legend* element.

  ASPECT II. LSRELATEDCONTROLS

  POINTCUT ALL INTERFACE WIDGETS WITH CompositeInterfaceElement.IndefiniteVariable == HTML *related controls*

  PROPERTY ADVICE ADD ACCESSIBILITY CONDITION **12.4** explicitAssociation == HTML *for* element.

- **Level 1, 2 and 3 – Aspect II in UI models at UID interactions <1, 2, 3>.**

  ASPECT II. LSIMAGE

  POINTCUT ALL INTERFACE WIDGETS WITH ElementExhibitor == HTML *image*

  PROPERTY ADVICE ADD ACCESSIBILITY CONDITIONS

  1.1 textEquivalent == HTML *alt* element OR HTML *longdesc* element ∧

  2.1 infoWithoutColor AND **2.2** useContrastColor == W3C *HTML, Core* AND *CSS Techniques* for color.

- **Level 4 – Aspects in UI models at UID interaction <4> (Optional).** At this level, we proceed in the same way as for the previous levels. We do not give details about this optional level, because we consider it doesn't provide new knowledge about injecting aspects in UI models.

- **Level 1, 2, 3 and 4 – Aspect II in UI models at UID interactions <1, 2, 3, 4>.**

  ASPECT II. LSSTYLESHEET

  POINTCUT ALL INTERFACE WIDGETS WITH ElementStyling.Formating&Positioning == HTML *stylesheet*

PROPERTY ADVICE ADD ACCESSIBILITY CONDITIONS

3.3 useStyleSheetLayout&Presentation AND **3.4** useRelativeUnitsPositioning AND **6.1** makeAvailableWithoutStylesheet AND **14.3** useConsistentStylePages == W3C *HTML, Core* AND *CSS Techniques* for controlling layout and presentation.

These are high-level specifications to avoid "scattering" and/or "tangling" symptoms caused by Accessibility concerns. The pointcut/advice pair specifies that, for all HTML widget of a specific kind (the pointcut specification), conditions satisfying Accessibility requirements are added (the advice specification).

As a result of modeling these aspects (using SIGs prescriptions for WCAG 1.0 checkpoints) and the addition of these aspects to deal with the targeted interface widgets, Figures 5.8, 5.10 and 5.12 show the accessible implementations for the concrete user interface models for the 3 (three) level-deep navigation case study in Figure 5.1, in terms of "well formed" HTML like W3C document [45].

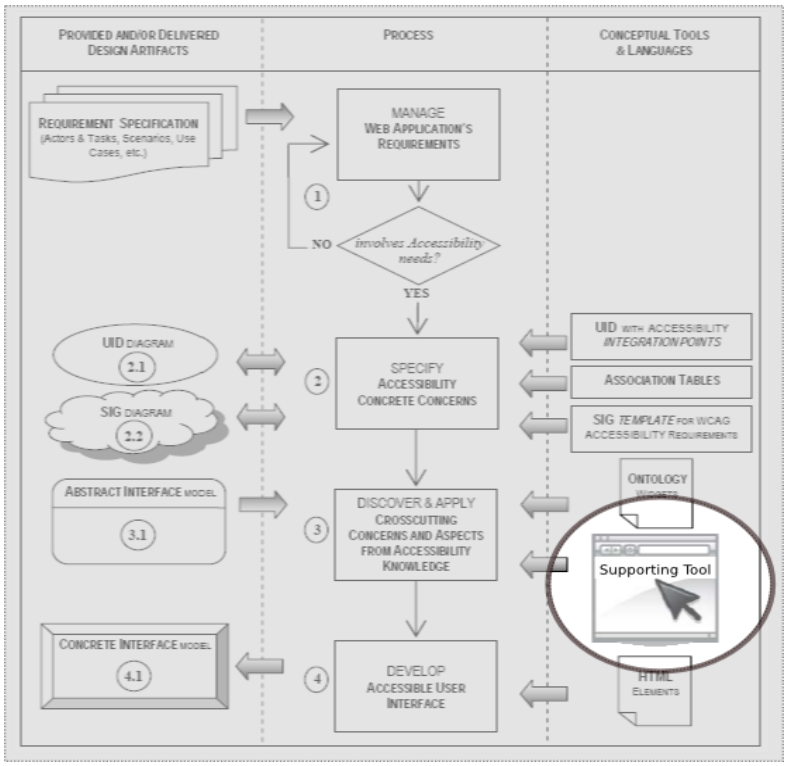

Figure 5.15: The supporting tool within our Aspect-Oriented design process

5.3 A Supporting Tool for Our Approach

Today, no one can deny the significance of having a supporting tool. The supporting tool and the kind of support given and features covered by the tool is relevant, especially to a design proposal. Related to this issue, our approach provides an initiative for a supporting tool to assist developers in the implementation of cases, and on the creation of their corresponding models by using reusable components. Currently, as Figure 5.15 shows, the tool provides assistance at **Step 3** of the design process for applying the Accessibility aspects (prescribed by the SIGs diagrams) to user interface models --i.e. abstract and concrete user interface models.

To achieve with its main purpose, the tool must deal with the concepts previously described, such as SIG diagrams, *association tables* and abstract user interface models. Also, the tool should be at the user's fingertips --i.e. the tool should be part of the users' development environment. To solve the second issue, the tool was developed as an Eclipse[44] plug-in, integrating an XML[45] editor in combination with the necessary views to inform the user about the missing information required for an accessible user interface --i.e. tags and attributes for a well-formed and accessible markup, as we describe in Section 5.3.2, and also to provide options to fix these missing information.

At this point, we introduce a brief explanation for the rational of choosing XML as the markup language to support resources and their future development as the tool evolves. Since XML allows writing our own markup language, we are not restricted to a limited set of tags defined by proprietary vendors. Custom tags are used to bring meaning to the data being displayed and when stored this way, data becomes extremely portable because it carry with their description rather than their display. In this way, XML allows the display to be extracted from the data and incorporated into a style sheet. Some of the benefits of this important XML characteristic are: (i) changes to display do not require futzing with the data, since a style sheet will specify the display, (ii) searching the data is easy and efficient, since tags provide the search engines with the intelligence they lack, (iii) complex relationships like trees and inheritance can be communicated and,

[44] The Eclipse Foundation at http://www.eclipse.org/

[45] W3C Extensible Markup Language (XML) at http://www.w3.org/XML/

(iv) the XML code is much more legible to a person coming into the environment with no prior knowledge. Other XML properties are: (i) it has stricter grammar rules than HTML that helps to develop well-formed documents --e.g. forgetting a label in an XML document makes the file unusable, (ii) it is a platform independent language and widely distributed and, (iii) it was developed by the W3C that also keeps its specification. The design goals of XML emphasize simplicity, generality, and usability over the Internet.

Following we introduce the proposed tool, describing the basis of its architecture, layers and classes, and also the resources and interfaces through which developers interact for designing accessible user interfaces.

5.3.1 Architecture's Overview: Layers and Classes

Figure 5.16 shows the tool's architecture and its three main layers, which are: *Presentation*, *Object Storage* and *Core*.

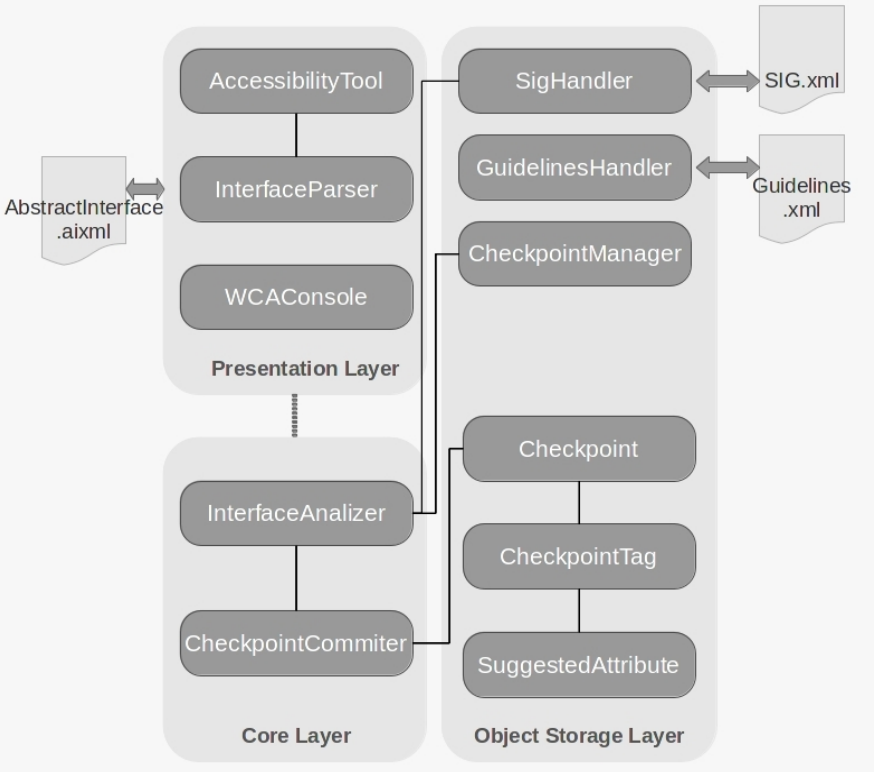

Figure 5.16: Main components of Our supporting tool

The *Presentation* layer represents the user interface for designers and developers. The main classes in the *Presentation* layer are:

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

The *Object Storage* layer represents an abstraction for the different underlying resource structures. Then, requests for information about WCAG 1.0 checkpoints [45], present in the SIG structure or in the tool database, are solved using the services of this layer. The main classes for the *Object Storage* layer are:

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

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

- **CheckpointCommiter** class, whose functionality includes the analysis and determination of commitment of an HTML tag to the WCAG recommendations. Also, it provides the functionality to generate the element code --i.e. HTML tag or attribute, to fix the non-commitment.

- **InterfaceAnalizer** class, which provides the functionality of coordination for the analysis of the abstract user interface model. This class has an aspect-based implementation done in AspectJ[46], which is the central feature that will allow the completion of the analysis in a transparent manner --i.e. solving Accessibility crosscutting problems by injecting aspects smoothly.

Particularly, in Figure 5.16, we focus on the *Presentation* layer, which is isolated from the other layers and it is only related to the *Core* layer by a dotted line, meaning that there is no straight interaction between these two layers. Thus, the interaction between these two layers, which includes reading and analyzing the abstract user interface model under treatment, takes place in a transparent manner. This abstract user interface model is an XML file, as we following see in Section 5.3.2. To reproduce this behavior, the tool uses the *Observer pattern*[47] and their classes *Subject* and *Observer*; each instance of the *Subject* class maintains a list of instances of the *Observer* class that are notified of the changes that occur in their respective instance of the *Subject* class. By applying these design concepts, the *AccessibilityTool* class plays the role of *Subject*, while the *InterfaceAnalizer* class plays the role of *Observer*. Then, the aspects environment --i.e. the AspectJ capabilities, manages the update notifications. Thus, when the developer saves the XML document edited for the abstract user interface model, this automatically triggers this aspect-oriented functionality, which is not explicitly invoked by some element of the *Presentation* layer. As shown in Figure 5.15, the consequence at **Step 4.1** is the deliverable of a concrete HTML user interface model that improves conformance to WCAG 1.0 Accessibility requirements.

5.3.2 Tool's Resources: XML Schemas and Specifications

Figure 5.16 shows three XML files representing the input/output resources of the tool, which are *AbstractInterface*, *SIG*, and *Guidelines*. Following, we explain the relationship of these resources with our design proposal and we also provide their

[46] The AspectJ Development Tool at http://www.eclipse.org/ajdt/

[47] Object-Oriented Design and Programming: Observer Pattern at http://www.oodesign.com/observer-pattern.html

respective XML schema. Using examples, we show how to instantiate these XML schema for specifying the XML files.

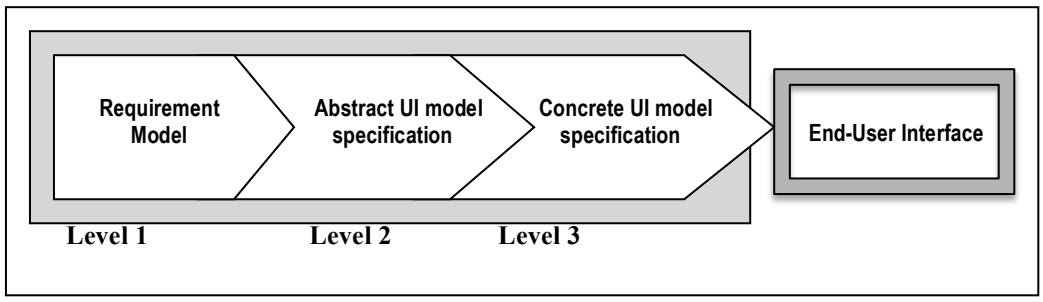

Figure 5.17: Model-driven principles applied to UI model development

The ***AbstractInterface* XML file** represents the abstract user interface model. As we have explained in previous chapters, our design approach uses the model-driven paradigm to develop high-level descriptions of the user interface structure and behavior and, from these declarative models to obtain the end-user interface. Figure 5.17 illustrates these design concepts, which are implemented by WE methods [31], such as OOHDM [36], which we have applied to develop our approach and supporting tool.

Figure 5.18 shows, the ***AbstractInterface* XML schema**[48] that we develop for specifying machine-understandable abstract user interface models. The most important tags of this XML schema are *Interface*, *Component*, *Composite* and *Attribute*.

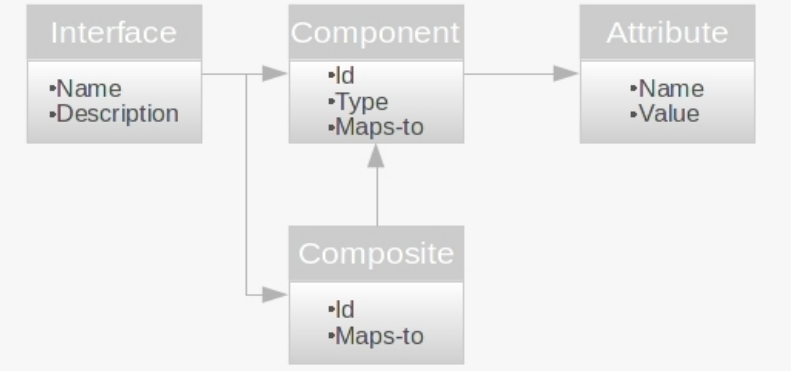

Figure 5.18: XML schema for the Abstract User Interface model

The specification of documents based on this schema begins with an *Interface* element, which can comprise *Composite* and *Component* elements. Also, a *Composite* element

[48] W3C XML Schema at http://www.w3.org/XML/Schema

can comprise *Component* elements resulting in a hierarchy of elements. Each tag has a modeling function within the *AbstractInterface* XML schema and its own descriptive attributes, as follow:

- The **Interface** tag is the container for the structure of an abstract user interface. The *Interface* tag has two descriptive attributes: (i) name, which identifies the *Interface* element under develop and, (ii) description, which states the purpose of the *Interface* element and the *Composite* and *Component* elements that are comprised within the *Interface* element.
- The **Component** tag represents the widgets that make up the abstract user interface. The *Component* tag has three descriptive attributes: (i) id, which identifies the *Component* element under development, (ii) type, which assign to the *Component* element a simple ontology widget and, (iii) maps-to, which links the *Component* element to a simple HTML element --e.g. an HTML *text field* element which is usually codified by using an HTML *input* element.
- The **Composite** tag is a container within an *Interface* element that comprises *Component* elements. The *Composite* tag has two descriptive attributes: (i) id, which identifies the *Composite* element under development and, (ii) maps-to, which links the *Composite* element to a composite HTML element --e.g. an HTML *related controls* element which is usually codified by using an HTML *fieldset* element.
- The **Attribute** tag represents the attributes that will be part of a concrete HTML element conveyed by "map-to" attributes. To complete the user interface design, the user adds some of these attributes, while the tool suggests others to solve Accesibility concerns.

Figure 5.19 shows the XML file specified applying the *AbstractInterface* XML schema to part of the case study shown in Figure 5.1 (c). As we can see in this specification, a *Composite* element is included at line 4 to represent the student identification FORM, which is a composite HTML element comprising two *Component* elements. These two INPUTs are *Component* elements included at lines 5 and 7 respectively, to represent the HTML *text field* elements required for the student's name and password. The pair of attributes type and maps-to allow the association between ontology widget-HTML

element --e.g. the *Component* elements at lines 5 and 7 are of the ontology type **indefiniteVariable** and maps-to HTML *input* elements.

```
1.  <interface name="student's login" description="An interface for
    the student's login at the SIU Guarani registration system">
2.  <component id="guaraniLogo" type="elementExhibitor" maps-to="IMG">
3.  </component>
4.  <composite id="studentID" maps-to="FORM">
5.  <component id="studentName" type="indefiniteVariable" maps-
    to="INPUT">
6.  </component>
7.  <component id="studentPassword" type="indefiniteVariable" maps-
    to="INPUT">
8.  </component>
9.  </composite>
10. </interface>
```

Figure 5.19: XML specification of an abstract user interface model

The ***SIG* XML file** represents the Softgoal Interdependency Graph (SIG) *template* for Accessibility and, as shown in Figure 5.20, we develop the ***SIG* XML schema** for specifying machine-understandable SIG diagrams. The most important tags of this *SIG* XML schema are *SIG*, *Node* and *Relation*.

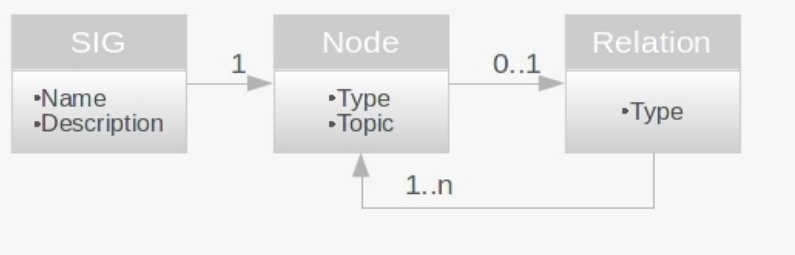

Figure 5.20: XML schema for the SIG template for Accessibility

The specification of documents based on this *SIG* XML schema begins with a *SIG* element linked to a main *Node* element, which in turn can comprises one or more *Node* elements through a *Relation* element. Thus, the *Relation* element allows a hierarchy specification for a *SIG* element. Each tag has a modeling function within the *SIG* XML schema and its own descriptive attributes, as follow:

- The **SIG** tag is the container for the structure of a SIG diagram for Accessibility. The *SIG* tag has two descriptive attributes: (i) name, which identifies the *SIG* element under develop and, (ii) description, which focus on the Accessibility softgoal of the *SIG* element through its main *Node* element --i.e. which, as we already explained in Section 5.2, is called the root light cloud of the SIG diagram applying the SIG terminology.
- The **Node** tag represents a node, which, as we have already explained in Section 5.2, is called a cloud of the SIG diagram applying the SIG terminology. Thus, a *Node* element can represent a root or a refined Accessibility softgoal –i.e. a white cloud of the SIG diagram applying the SIG terminology, or an operationalizing goal for the required checkpoints to be satisfied –i.e. a dark cloud of the SIG diagram applying the SIG terminology. The *Node* tag has two descriptive attributes: (i) type, which specifies the type of a *Node* element depending on its Accessibility softgoal and, (ii) topic, which describes the Accessibility softgoal to be satisfied. While, the type of the *Node* attribute can be one of the following:
    - **U-UI** type, if the softgoal comprises Accessibility requirements to be satisfied at an interaction level in the UID diagram. We can use the U-UI type for a *Node* element representing a root Accessibility softgoal in the SIG diagram --e.g. in Figure 5.5, the U-UI root cloud for the SIU Guarani home page.
    - **U-UIc** type, if the softgoal represents Accessibility requirements to be satisfied at a component level in the UID interaction. We can use the U-UIc type for a *Node* element representing a refined or an operationalizing goal of the SIG diagram --i.e. in Figure 5.5, the U-UIc refined cloud for the HTML *related controls* element representing the student's identification form.
    - **Decomposition** type, if the *Node* element represents an Accessibility softgoal refinement by decomposition –i.e. in Figure 5.5, the Decomposition cloud at the User Technology Support branch for the HTML *related controls* element.
- **Operationalizing** type, if the *Node* element represents an Accessibility operationalizing goal –i.e. in Figure 5.5, the Operationalizing dark clouds representing Accessibility requirements to be satisfied.

- The **Relation** tag applies for a parent *Node* element and its children, allowing a hierarchy specification for a *SIG* element. The *Relation* tag has only one descriptive attribute, type, which specifies the type of the relationship established between the parent *Node* element and its children. While, the type of the *Relation* attribute can be one of the following:
    - **AND** type, which represents the conjunction relationship, where all the children representing Accessibility softgoals must be satisfied to satisfy its parent *Node* element.
    - **OR** type, which represents the disjunction relationship, where satisfying some of the children representing Accessibility softgoals satisfied the parent *Node* element.
    - **OPERATIONALIZING** type, which represents the Accessibility operationalizing goal of the parent *Node* element. These operationalizing goals implement concrete Accessibility requirements on which a validation can be performed to establish conformance. For the instantiation of the Accessibility requirements, our tool applies the WCAG 1.0 checkpoint [45], but as we will explain in Chapter 6, our design proposal can work also with the WCAG 2.0 success criteria [46].
- The **NodeList** tag is a container for a list of *Node* elements within a *Relation* element. Therefore, the *NodeList* tag can comprise one or more *Node* elements that are children of a parent *Node* element.

Figure 5.21 shows the XML file specified applying the *SIG* XML schema to part of the XML specification of the abstract user interface model in Figure 5.20. As shown at line 1, the softgoal to be satisfied --i.e. the Accessibility concern of the SIG diagram, is set in order to improve the Accessibility for all the students accessing the SIU Guarani registration system. The root *Node* element at line 2 is of the type **U-UI** because its Accessibility softgoal targets the UID interaction representing the home page of the system. This root *Node* element is decomposed into two refined *Node* elements at lines 5 and 19 by a *Relation* element of the type **AND** at line 3. These two *Node* elements are of the type **U-UIc** because their Accessibility softgoals target the IMG and FORM components at the UID interaction representing the home page of the system. The softgoal refinement process continues over the tree to develop the SIG diagram for

Accessibility, until specific operationalizing goals are met. For example, at line 11 the *Node* element is of the type **operationalizing** and in consequence instantiates the topic attribute with the **checkpoint 1.1** to establish a concrete Accessibility requirement to be satisfied.

```
1.  <sig name="student's login" description="SIG instantiation for
    an accessible user interface for the student's login at the SIU
    Guarani registration system">
2.  <node type="U-UI" topic="HTML SIU Guarani Page">
3.  <relation type="AND">
4.  <nodeList>
5.     <node type="U-UIc" topic="IMG">
6.     <relation type="AND">
7.     <nodeList>
8.       <node type="decomposition" topic="USER LAYOUT SUPPORT">
9.       <relation type="OPERATIONALIZING">
10.      <nodeList>
11.        <node type="operationalizing" topic="1.1" />
12.        ...
13.      </nodeList>
14.      </relation>
15.      </node>
16.    </nodeList>
17.    </reation>
18.    </node>
19.    <node type="U-UIc" topic="FORM">
20.    <relation type="AND">
21.    <nodeList>
22.      <node type="decomposition" topic="USER TECHNOLOGY LAYOUT">
23.      ...
```

Figure 5.21: XML specification of a SIG diagram for Accessibility

The ***Guidelines* XML file** represents the Accessibility guidelines from the WCAG 1.0 recommendations [45], which are stored accordingly to a structured language we especially develop. As we have already seen in previous chapters, there is a gap between the abstract knowledge transmitted by guidelines, which are expressed in natural language, and their implementation using a markup language such as HTML,

which is based on a technical specification[49]. Trying to reduce this gap, we propose a structured language for guidelines, which we called in Spanish LEP (Lenguaje de Estructura de Pautas). As Figure 5.22 shows, LEP is positioned between natural language and HTML, simplifying not only the human comprehension of guidelines but also their storage as structures specified by a XML schema. Therefore, LEP is a specification language to adapt the structure of the Accessibility guidelines from WCAG 1.0 recommendations and make them possible to be managed by our tool.

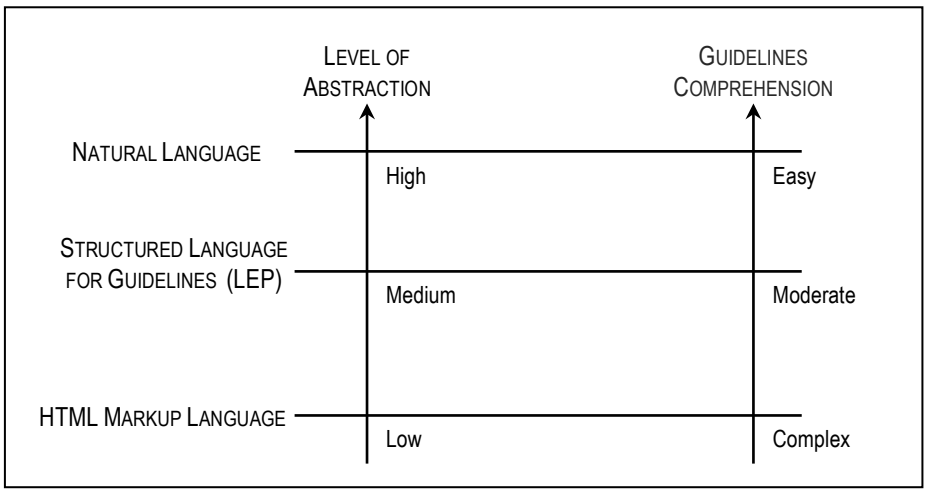

Figure 5.22: Levels of expressiveness to Accessibility Guidelines comprehension

The W3C-WAI [50] has specified systematically the 14 (fourteen) guidelines of the WCAG 1.0 recommendations (see the complete document at Appendix I). Each guideline within the WCAG 1.0 recommendations [45] includes: (i) the guideline number, (ii) the statement of the guideline (iii) the rationale behind the guideline and some groups of users who benefit from it and, (iv) a list of checkpoint definitions. The checkpoint definitions in each guideline explain how the guideline applies in typical content development scenarios. Each checkpoint definition includes: (i) the checkpoint number, (ii) the statement of the checkpoint, (iii) the priority of the checkpoint (the priority levels are 1, 2, 3), (iv) optional informative notes, clarifying examples, and cross references to related guidelines or checkpoints and, (v) a list of techniques where implementations and examples of the checkpoint are discussed to facilitate the checkpoint evaluation and conformance.

[49] W3C HTML 4 Specification at http://dev.w3.org/html5/spec/Overview.html

Now, to adapt this Accessibility information provided by WCAG 1.0 recommendations, we consider the formalization of those elements that are relevant to the expressiveness of the stored structures for providing the proper support required by the tool. Figure 5.23 shows the ***Guidelines* XML schema** we develop based on LEP --i.e. our supporting language, to allow the adaptation of the Accessibility guidelines and to store their structures as machine-understandable representations. The most important tags of the *Guidelines* XML schema are *Guidelines, Guideline*, *Checkpoint*, *Tag* and *Attribute*.

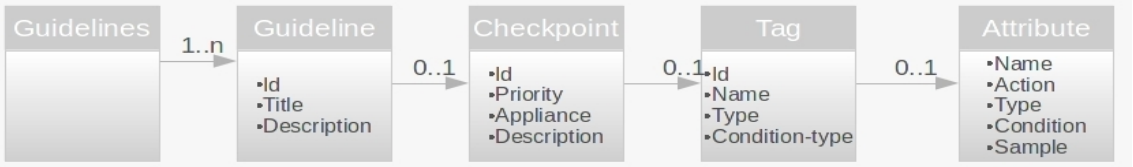

Figure 5.23: XML schema for the Accessibility guidelines from WCAG 1.0

As we can see in Figure 5.23, each *Guideline* element has a list of *Checkpoint* elements and each *Checkpoint* element has a list of *Tag* elements --i.e. HTML tags, which are the target of the *Checkpoint* element. For example, if a *Checkpoint* element establishes that an HTML *table* element must summary its content --i.e. checkpoint 5.5 from WCAG 1.0, the *Checkpoint* element will include a *Tag* element for the HTML *table* element and, the *Tag* element will include an *Attribute* element for the HTML *summary* element.

[**GUIDELINE NUMBER**] – [**STATEMENT** OF THE GUIDELINE]	
[**CHECKPOINT NUMBER**] – [**STATEMENT** OF THE CHECKPOINT] – [**PRIORITY** OF THE CHECKPOINT]	
PRESCRIPTION OF THE CHECKPOINT	**APPLIANCE**
Provides an explanation of the checkpoint and its foundations to compliance.	[SEMI-AUTOMATIC] *Requires the developer's manual intervention with the tool's support.* OR [MANUAL] *Requires the developer's manual intervention without the tool's support.*
SAMPLE: Provides topics on how to implement the checkpoint using well-formed and accessible HTML.	
SAMPLE IN LEP SPECIFICATION: Provides examples of how the checkpoints are specified in LEP.	

Figure 5.24: Adapting the WCAG 1.0 checkpoints to the schema based on LEP

The *Guidelines* XML schema based on LEP, convey information through the following tags:

- The **Guidelines**, which allow beginning a new file and containing its structure.
- The **Guideline**, which provides id, title and description of a specific WCAG 1.0 guideline; also includes a list of its checkpoints.

GUIDELINE 1. PROVIDE EQUIVALENT ALTERNATIVES TO AUDITORY AND VISUAL CONTENT

CHECKPOINT 1.1 Provide a text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content). *This includes*: images, graphical representations of text (including symbols), image map regions, animations (e.g., animated GIFs), applets and programmatic objects, ascii art, frames, scripts, images used as list bullets, spacers, graphical buttons, sounds (played with or without user interaction), stand-alone audio files, audio tracks of video, and video. **[PRIORITY 1]**

PRESCRIPTION OF THE CHECKPOINT	APPLIANCE
• Use "alt" for the IMG, INPUT, and APPLET elements, or provide a text equivalent in the content of the OBJECT and APPLET elements. • For complex content (e.g., a chart) where the "alt" text does not provide a complete text equivalent, provide an additional description using, for example, "longdesc" with IMG or FRAME, a link inside an OBJECT element, or a description link. • For image maps, either use the "alt" attribute with AREA, or use the MAP element with A elements (and other text) as content.	[SEMI-AUTOMATIC]

SAMPLE:

```
<img src="guarani3w.jpg"
alt=""
longdesc="../descrip/decor.htm#guarani3w">
```

SAMPLE IN LEP SPECIFICATION:

```
<tagList>
  <tag id="1" name="IMG" type="" condition-type="">
     <attributes>
       <attribute name="ALT" sample"img src="guarani3w.jpg" alt="*"
       action="add" type="HTMLAttribute" condition="mandatory"/>
     </attributes>
  </tag>
</tagList>
```

Figure 5.25: Adapting checkpoints 1.1 to the schema based on LEP

- The **Checkpoint**, which provides id, priority (1, 2, 3) and description of a specific WCAG 1.0 checkpoint; also includes the appliance, which is "semi-automatic" when the checkpoint requires the developer's manual intervention with the tool's support or is "manual" when requires the developer's manual intervention without the tool's support, and a list of the HTML tags concerning to the checkpoint.
- The **Tag**, which provides id, which is a number assigned for identification purpose and is not related with WCAG 1.0 guidelines and checkpoints numbers, name (the

HTML tag name), and type/condition-type, which allow to specify the tag use case/s where the guideline/checkpoint applies to the tag; also includes a list of its attributes.

- The **Attribute**, which provides name (the HTML attribute or tag name), action (add, modify, update or delete), type (HTML tag, HTML attribute, text attributes, etc.), condition, which allows specifying if the attribute is mandatory or optional, and sample, which provides an application example.

The preservation of the WCAG philosophy was our goal when we worked on the Accessibility guidelines seeking for a specification manageable by the tool. Figure 5.24 summarizes the basis for analyzing and adapting the WCAG 1.0 checkpoints to the *Guidelines* XML schema based on LEP, while Figure 5.25 shows part of the analysis and adaptation for **checkpoint 1.1**. For example, this specification applies to satisfy the **operationalizing** softgoal in the SIG diagram shown in Figure 5.21, line 11.

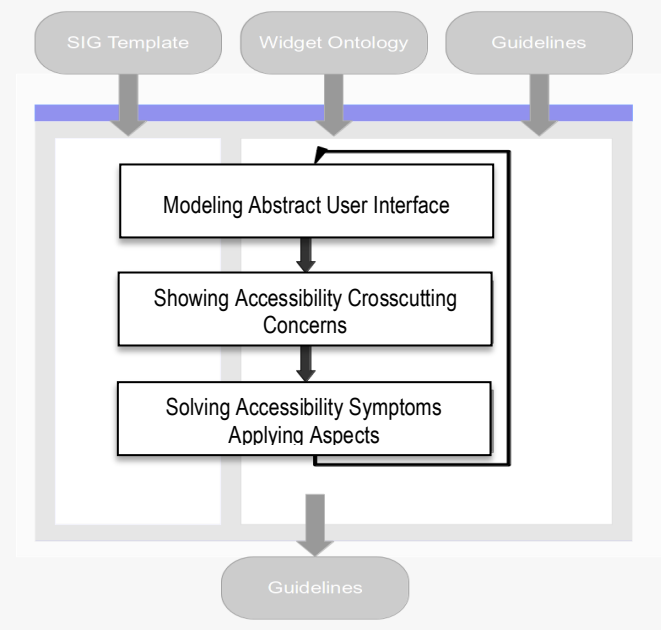

Figure 5.26: Basis of the Aspect-Oriented design cycle

5.3.3 Tool's User Interfaces

From the user's point of view the interaction with the tool applies an "open-save-close" cycle to the document under develop. The developer designs an abstract user interface

for a given Web page by editing and saving changes in an XML-based document. This mode for developing documents is usually known as document-centered work schema.

Figure 5.26 shows the basis of the aspect-oriented design cycle in the interaction between the developer and our tool, where we can identify the following steps:

- **Modeling Abstract User Interface**, the developer designs the abstract user interface model choosing widgets from the abstract widget ontology.
- **Showing Accessibility Crosscutting Concerns**, the tool shows how the Accessibility concerns crosscut the interface widgets selected to compose the user interface by the developer.
- **Solving Accessibility Symptoms Applying Aspects**, the developer decides, based on the information provided by the tool and the tool wraps, these Accessibility crosscutting concerns into Accessibility aspects for their modularization and transparent injection in the user interface under design.

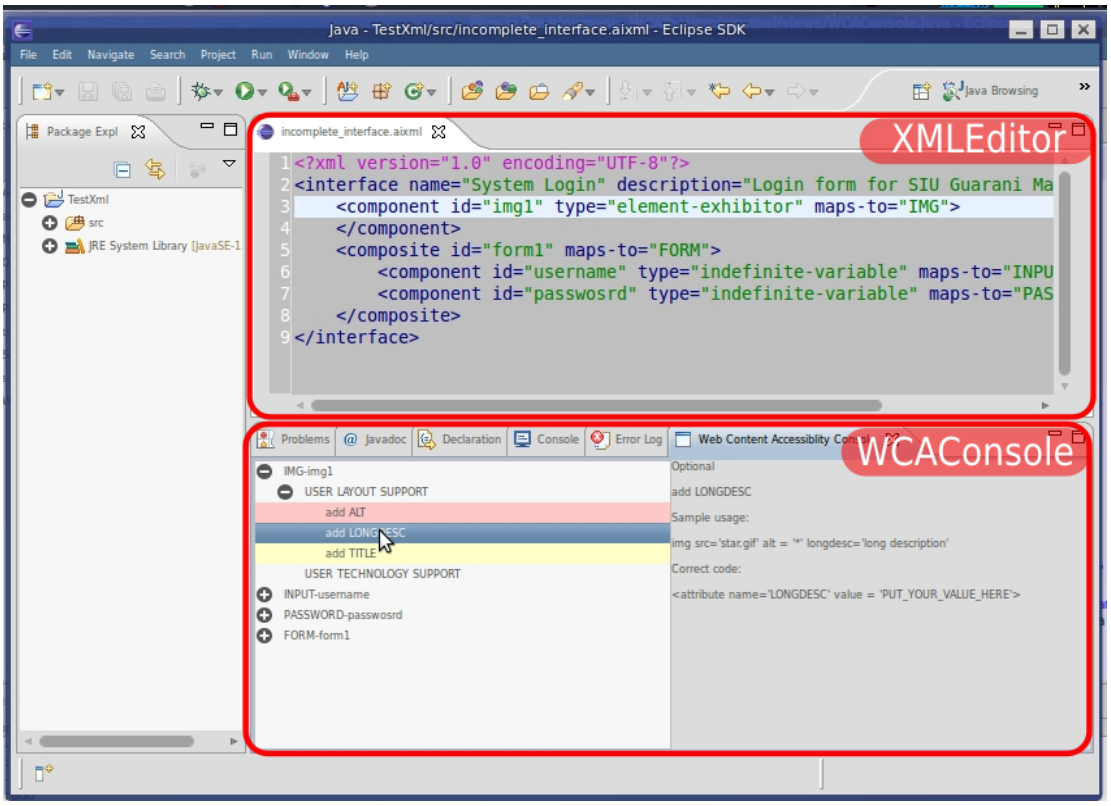

Figure 5.27: The components integrated in the Eclipse platform

For this reason, one of the main components of the tool's UI is the *XMLEditor*, which is complemented with the view *WCAConsole* for showing, and allow solving the non-

commitment to the Accessibility guidelines. Figure 5.27 shows a screenshot of these tool components integrated in the Eclipse platform. The *XMLEditor* is shown in the upper box of screen in Figure 5.27 and is used by the developer to edit the abstract user interface model. When the developer saves the XML file and its changes, the analysis of the structure and commitment to the Accessibility guidelines is launched. The analysis result is shown in a structured manner using the view *WCAConsole*, which is shown in the lower box of the screen in Figure 5.27 and also and also in Figure 5.28. The *WCAConsole* comprises two other components. The one on the left side of the *WCAConsole* is a tree view, which shows to the developer the missing elements and/or errors in the implementation of elements for every tag present in the abstract user interface. This tree view is based on the SIG diagram for Accessibility and also shows related tags that should be in an accessible a well-formed user interface.

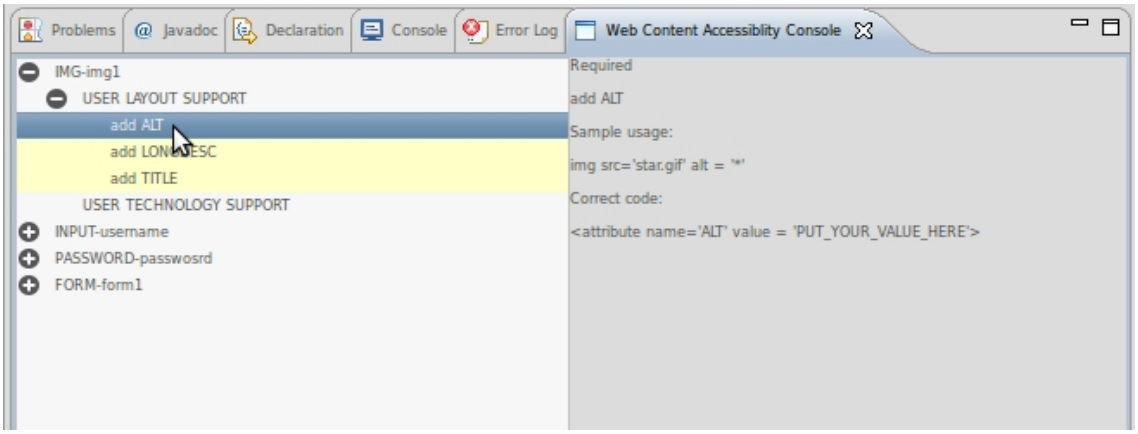

Figure 5.28: The WCAConsole component

The other component on the right side of the *WCAConsole* is a read-only description view, which shows to the developer the following information, for each selected element of the component on the left side:

- **Attribute/Tag condition** (**Mandatory/Optional**): Indicates to the developer whether the selected element (tag or attribute), is mandatory, as shown in Figure 5.28, or optional, as shown in Figure 5.27, to satisfy the guideline/checkpoint.
- **Action** (**Add/Remove**): Indicates to the developer the action to perform with the selected element (tag or attribute), if the element should be added (or must be added if the condition is mandatory) to the abstract user interface or removed.

- **Sample usage**: Provides to the developer an example on how to properly use in HTML the element (tag or attribute).
- **Correct code**: Shows to the developer the necessary XML code to insert the element (tag or attribute) in the abstract interface model to commit to the Accessibility guidelines.

5.3.4 Some Insights about the Tool

Our supporting tool, which was conceived prioritizing early Accessibility design, helps developers on the application of our Aspect-Oriented proposal to create user interfaces. The tool provides support at **Step 3** of the design process to discover crosscutting concerns and apply aspects from the knowledge captured about Accessibility requirements in previous stages. Following the approach's basis, the type of support and features covered by the tool can be described as those that usually provide a Computer-Aided Software Engineering (CASE) tool with model-driven properties. As a CASE tool, our supporting tool results helpful in creating models of cases. These models can be developed using reusable components and this is possible because of two reasons. On one hand, the Accessibility guidelines are quite independent from the Web application under development, so there are many cases to which the same Accessibility solution can be applied. Then, recording such recurrent situations (e.g., using patterns) enables to reuse them, which contribute to reduce the development effort when implementing our proposal. On the other hand, the Accessibility aspects as we proposed, could be developed once and be reused in different Web projects. For example, returning to the student's login Web page example in Figure 5.1 (c), establishing a logical tab order for accessing the HTML text field elements for the student ID and password, is an Accessibility concern that forces crosscutting in the implementation. The early identification of this situation allows modeling a reusable Accessibility aspect that is going to be in charge of providing an HTML *tabindex* element for each text field element at the user's layout. Currently, since the function for reusing components is not fully implemented, our tool provides assistance for applying the Accessibility aspects (prescribed by some predefined and stored SIG diagrams) to an abstract user interface model loaded by the designer.

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

As a final note, we provide our supporting tool aiming to help and, as a consequence, encourage, Web development in designing user interfaces with the Accessibility quality factor in mind.