Integrating Adaptation Rules for People with Special Needs in Model-Based UI Development Process

Raúl Miñón[1], Fabio Paternò[2], Myriam Arrue[1], Julio Abascal[1]

[1] EGOKITUZ: Laboratory of HCI for Special Needs University of the Basque Country/ Euskal Herriko Unibertsitatea, Manuel Lardizabal 1, 20018 Donostia, Spain

Tel.: +34 943 015113

{raul.minon, myriam.arrue, julio.abascal}@ehu.es

2 ISTI-CNR Via Moruzzi 1, 56124 Pisa, Italy

Tel.: +39 050 3153066

{fabio.paterno}@isti.cnr.it

Abstract. The adaptation of user interfaces for people with special needs is a promising approach in order to enable their access to digital services. Model-based user interfaces provide a useful approach for this purpose since they allow tailoring final user interfaces with a high degree of flexibility. This paper describes a system called Adaptation Integration System aimed at providing Cameleon Reference Framework model-based tools with a mechanism to integrate adaptation rules in the development process. Thus, more accessible user tailored interfaces can be automatically generated. The services provided by the system can be applied at both design time and runtime. At design time, a user interface can be tailored at any abstraction level in the development process. At runtime, changes in the context of use trigger the adaptation process. Adaptation rules are stored in a repository tagged with meta-information useful for the adaptation process, such as the granularity of the adaptations and the abstraction level. As case studies, two applications have been developed using the services provided by the system. One of them exploits the benefits at design time whereas the other application is devoted to describe the adaptation process at runtime. The results obtained in these two scenarios demonstrate the viability and potential of the Adaption Integration System since even inexperienced designers may efficiently produce accessible user interfaces.

Keywords: Accessibility, Design Space, Adaptation Rules and Model-Based UI

1 Introduction

People with disabilities usually find barriers to interact with digital services because the user interfaces for these services are not adequately adapted to their needs. Designers usually claim their lack of knowledge of accessibility techniques as well as the high cost in terms of time and effort of developing accessible user interfaces as the main reasons for this situation. In addition, it is not always possible to provide a single user interface satisfying the needs of all users. Consequently, conscious designers have to deal with different versions of interfaces each one tailored to one group of users.

Development of context-aware accessible user interfaces is even more complex. User interfaces accessible for a particular user in a specific context may not be accessible in other contexts for this same user. For instance, a person with mild deafness watching a video on the web in a silent environment does not have accessibility problems. But when this person moves to a noisy environment she may experience accessibility barriers to get to the video content.

In order to enhance user interface accessibility, this paper presents a user interface development process devoted to incorporating adaptation rules for people with special needs. A system called Adaptation Integration System has been developed to support this process. It facilitates the integration of adaptation rules into user interfaces in any of the abstraction levels of the Cameleon Reference Framework [1]. The adaptation rules can be applied in both design and runtime scenarios. Therefore, the Adaptation Integration System supports designers using model-based user interface (MBUI) tools in the process of integrating accessibility requirements at design time and enables the development of context-aware accessible user interfaces at runtime. This system is connected to an adaptation repository, which stores and provides the different adaptation rules to apply depending on certain parameters. These adaptation rules conform to a comprehensive design space based on four dimensions: user disability, abstraction level, granularity level, and adaptation type. The system has been used for the development of two applications.

The rest of the paper is organized as follows: Section 2 analyses the related work; subsequently, Section 3 describes our approach to the development of the Adaptation Integration System, the system design and the architecture proposed and the implemented Adaptation Repository; Section 4 shows how the approach can be applied through two applications; and finally, conclusions and future work are drawn in Section 5.

2 Related Work

Some systems are devoted to automatically providing adequate user interfaces to users with special needs. Supple [2], for instance, automatically generates user interfaces adapted to users devices, tasks, preferences and abilities, where the interface generation is defined by the authors as a discrete constrained optimization problem. Another example is the Egoki system [3], an adaptive system for automatically generating accessible user interfaces for ubiquitous services. It focuses on the selection of suitable multimedia interaction resources (text, image, video, audio, etc.) for each user interface element considering the specific needs of each user. Nevertheless, none of these systems supports changes depending on the context of use.

Among the various context-aware systems, Bongartz et al. (2012) [4] propose a system supporting context-aware adaptive user interfaces in work environments. They generate graphical, vocal or multimodal user interfaces depending on the type of task the user is performing and the current context of use. They follow a model-based approach to generating the user interfaces. However, they do not provide specific support for people with special needs. Even though multimodal user interfaces can be beneficial for people with disabilities, they are not always enough to meet all their requirements. Daniel et al. (2008) [5] propose the Bellerofonte Framework aiming at enhancing the adaptation process of adaptive web applications. To this end, the authors define a language decoupled from the application logic based on the Event-Condition-Action (ECA) paradigm. It allows implementing an engine for processing and applying the adaptation rules. However, this framework is totally focused on context properties and it does not consider explicitly adaptation rules for providing support to people with special needs. In addition, this approach is only devoted to web user interfaces whereas our approach is independent of the final user interface implementation language.

Finally, there are some other systems that consider both accessible user interfaces and context-aware user interfaces. Stephanidis (2001) [6] discusses the importance of using adaptation techniques to Universal Access in Human-Computer Interaction (HCI) as well as the consideration of the users' context. Moreover, he proposes a development methodology to cope with the diversity of target groups, tasks and environments of use. This methodology entails defining a single user interface specification by means of abstract constructors to fit all users categories. In this way, different accessible versions of user interfaces can be produced both at design time and at runtime. Despite following a similar approach, our proposal is focused on providing support to MBUI tools, emerged in the last decade, at different steps of the development process. Thus, model-based designers can benefit from our system independently their target users, context and the step of the process they are.

Yang and Shao (2007) [7] propose an approach to applying content adaptations to Web-based systems. They use a dynamic adaptation strategy to select the adaptation rules to apply, which is automatically modified when a change in the user context occurs. The adaptation rules take into consideration the user context including parameters associated with disabilities. Thus, they provide an alternative content format when required and rearrange the layout to be suited for different user devices. By contrast, the Adaptation Integration System enables the provision of accessibility-related and context-sensitive adaptation rules to other developers using the Cameleon Reference Framework. In this way, user interfaces can be tailored at any step of the development process. In addition, rather than identifying the interaction elements as atomic elements, as the proposal of Yang and Shao does, the Adaptation Integration System considers different granularity levels. This allows carrying out a more flexible transformation process. Finally, despite their proposal being flexible to include more context parameters, currently, it only considers four different types of disabilities: *blind*, *weak-sighted*, *deaf* and *weak-hearing*. Conversely, while the Adaptation Integration System deals with a wider range of disabilities from its design being able to relate adaptation rules to any disability.

Quade et al. (2010) [8] propose a model-based approach to exploiting a user model with a system model at runtime. By performing this simulation, they dynamically evaluate the adaptation quality for users with disabilities. Then, if usability problems are detected the user interface is automatically updated. In addition, in future work it is expected the consideration of the user context. By contrary, the Adaptation Integration System proposes the building of a repository, which enables not only adaptations at runtime, but also at design time. Consequently, these adaptations are provided to other systems to benefit from them both in the early steps of their design processes and at runtime by using the Adaptation Integration System.
The MyUI infrastructure [9] [10] generates individualized user interfaces and performs adaptations depending on diverse user needs, devices and environmental conditions at runtime. It also provides a repository with the available adaptation rules based on design patterns. Each pattern is linked to a reusable software component and associated with a specific source code representation. The CakePHP framework [11] must be followed in order to be compatible with their infrastructure. In contrast, our system describes the adaptation rules in a self-explanatory way using the Advanced Adaptation Logic Description Language (AAL-DL) [12] without any additional implementation effort. In addition, it is not restricted to any programming language, since the Cameleon Reference Framework [1] approach is compatible with almost any final implementation language. Moreover, the possibility of including accessibility requirements in the first steps of the development process avoids possible design problems in the final steps as described in Miñón et al. (2014) [13].
The Global Public Inclusive Infrastructure (GPII), proposed by Vanderheiden and Treviranus (2011) [14], also considers accessibility and context-aware user interfaces. GPII is an infrastructure aimed at providing access to technology for making the development, identification, delivery, and use easier, less expensive, and more effective. The authors explain that GPII has been designed for automatically providing disabled users with solutions able to enhance their interaction with different public services. For instance, someone needing to access an inaccessible service, in a specific moment and in a concrete place can obtain an accessible interface from GPII. This solution would match user's requirements with the appropriate user interface without negotiating, explaining, qualifying or justifying. To accomplish this process, users store their preferences in a local device or in the cloud. Subsequently, they carry out the login process wherever they are and GPII provides them with a tailored user interface and the required assistive technologies. To this end, GPII proposes a framework to enable developers the provision of accessible solutions and a repository to identify assistive technologies. In addition, the GPII infrastructure also considers the features of the devices from which users are interacting with the public services. In contrast, the system presented in this paper assists designers to identifying the most relevant adaptation rules to apply for a given user at service design time. In addition, the purpose of the Adaptation Integration System is different. Whereas

GPPII is an infrastructure for providing the users with existing accessible solutions, the Adaptation Integration System is focused on the adaptation of applications built using MBUI tools, which can generate implementations adapted to various contexts of use.

3 Approach

3.1. Objective and Rationale

As previously mentioned, the objective of the proposed system is to allow the integration of adaptation rules in the development process of accessible user interfaces when applying MBUI tools. Therefore, designers using MBUI tools can produce accessible user interfaces even if they lack knowledge about accessibility requirements. To this end, the applied MBUI tools and the corresponding User Interface Description Languages (UIDLs) have to support accessibility patterns in their semantic. Accessibility requirements are integrated in the process through adaptation rules and transferred to the final user interface.

This system is compatible with the different abstraction levels described in the Cameleon Reference Framework [1] (Task & Concepts, Abstract and Concrete) and the integration of adaptation rules can be performed at both design and run time. For this purpose, two services have been designed to enable MBUI tools to interact with the Adaptation Integration System. In this way, any MBUI tool based on the Cameleon Reference Framework will be able to interact with the system regardless the technology used. It is worth pointing out that the Adaptation Integration System is the responsible for providing the necessary semantics to make the developed user interfaces accessible in any of the levels of the Cameleon Reference Framework at design time. However, at design time, the designer (or the MBUI tool) is the responsible for performing the transferring the semantic from one step to the next one in the development process, as it is illustrated in Figure 1 and described in Miñón et al. (2014) [13]. At runtime, the MBUI tool is the responsible for transferring the semantic to the user interface.

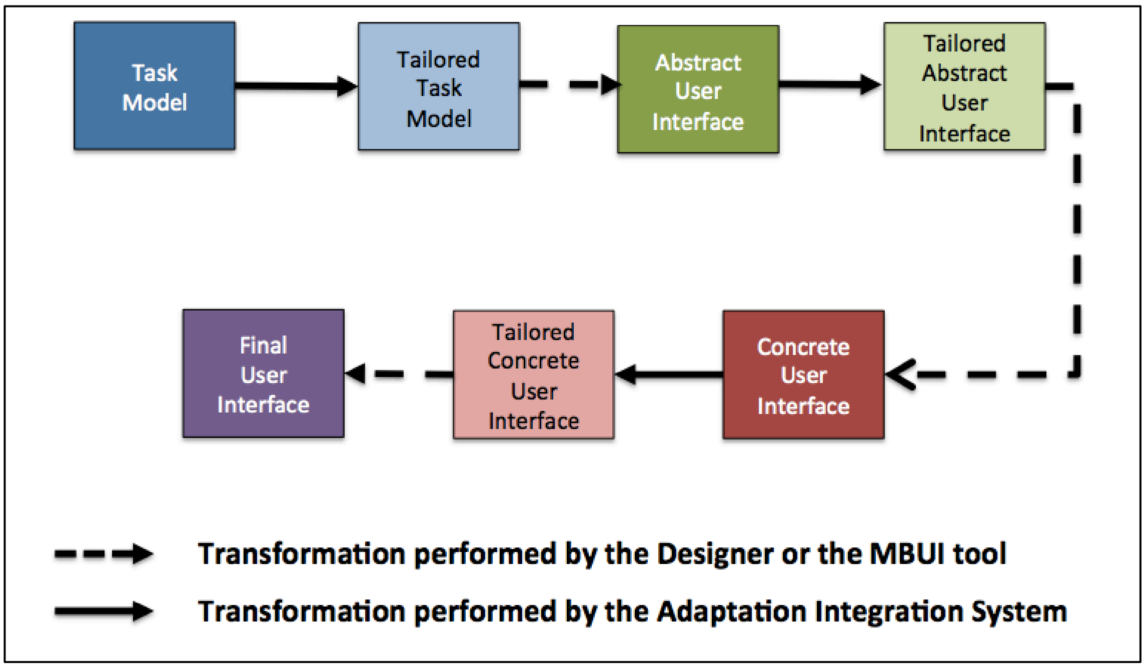

Task Model
Tailored Task Model
Abstract User Interface
Tailored Abstract User Interface
Concrete User Interface
Tailored Concrete User Interface
Final User Interface
Transformation performed by the Designer or the MBUI tool
Transformation performed by the Adaptation Integration System

Fig. 1 Actors responsible for applying each type of transformation at design time.

The adaptation rules designed for the Adaptation Integration System are able to modify the user interface at different granularity levels (Single Element, Group Element, Presentation and Application). Additionally, these adaptation rules are compatible with several user disabilities and platforms.

3.2. System Requirements

Some requirements have to be addressed in order to interact with the system. These requirements vary depending on when the adaptation rules are applied: design time or runtime. At design time, as it is highlighted by Gamecho et al. (2013) [15] it is necessary to provide a logical description of the user interface in any of the abstraction levels described in the Cameleon Reference Framework [1] and the designer of the service must have alternative resources (with the same name of the original but with different extension) to meet the requirements of some adaptation rules. At runtime, the process is more complex and the model-based user interface tool needs to meet the following requirements. A similar approach can be found at [16]:

- The application must have been developed following the Cameleon Reference Framework [1].
- The designer's tool has to integrate a mechanism to detect changes of context and to provide this context to the Adaptation Integration System. The context model proposed in the Serenoa project [17] is used to model the context. It describes the user features (knowledge, preferences, task, disability and position), the technology used (device, connectivity, browser), the environment (position, lighting and noise conditions) and

the social relationships for representing the parameters of a group (friendship, closeness, etc) that the user belongs.

- A mechanism for providing the correspondent user interface in the abstraction level required of the current final user interface to the Adaptation Integration System. Depending on the adaptation rules to apply, different abstraction levels can be required.
- A mechanism to reflect the changes applied at runtime in the final user interface.

Figure 2 illustrates an example, where the abstraction level required is the concrete. It shows the interaction between the designer's MBUI tool and the Adaptation Integration System at runtime. The steps of this interaction are the following:

- Step 1: The context manager module detects some changes in the user context.
- Step 2: The contextual events and the correspondent concrete user interface are sent to the Adaptation Integration System. In this way, the adaptation rules associated with these contextual events and defined for the concrete level are selected and applied by the system.
- Step 3: The system returns the resulting concrete user interface to the Final User Interface Generator module of the designer's MBUI tool.
- Step 4: Changes are applied to the final user interface based on the new concrete user interface.

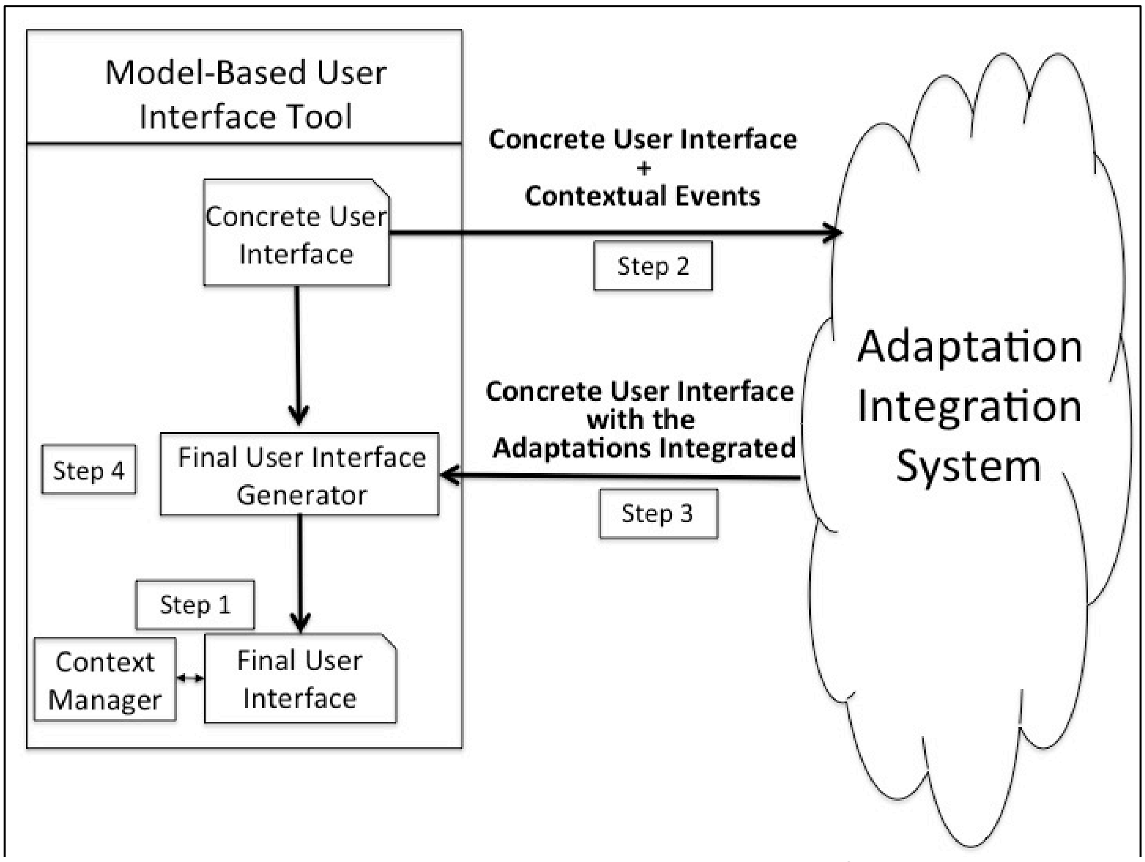

Model-Based User Interface Tool
Concrete User Interface
Final User Interface Generator
Step 4
Step 1
Context Manager
Final User Interface
Concrete User Interface + Contextual Events
Step 2
Concrete User Interface with the Adaptations Integrated
Step 3
Adaptation Integration System

Fig. 2 Interaction between designer's MBUI tool and the Adaptation Integration System at runtime.

3.3. Design and Architecture

The starting point of the adaptation process differs according to when it is applied: design time and runtime. At design time, the process starts when designers provide the Adaptation Integration System with a given user interface specification belonging to any of the levels described in the Cameleon Reference Framework [1] as well as a parameter indicating specific user disabilities. For this purpose, designers can interact with a remote service that provides a list with the disabilities supported by the existing adaptation rules. At runtime, the process starts when the context manager detects a meaningful change in the context.

Figure 3 illustrates the diverse modules implemented in the Adaptation Integration System as well as the process followed by the system in both cases. Concretely, the system is composed of three main modules that are implemented as Java classes: Request Parser, Adaptation Manager, and Adapter.

3.3.1 Request Parser

The Request Parser is the module in charge of analysing and parsing the parameters in the received query. This module implements two different services: one for receiving queries at design time and another one for receiving queries at runtime. At design time, as previously indicated, the designer specifies the following parameters: the logical description of a UI, its abstraction level, the disabilities to be considered in the adaptation, and the UIDL used by the designer (in Figure 3, these parameters are illustrated by the parameter *filter*). At runtime, the current context state is sent automatically to the system, so the user's disability and other required information are obtained from it. In both cases, the acquired information is redirected to the Adaptation Manager module.

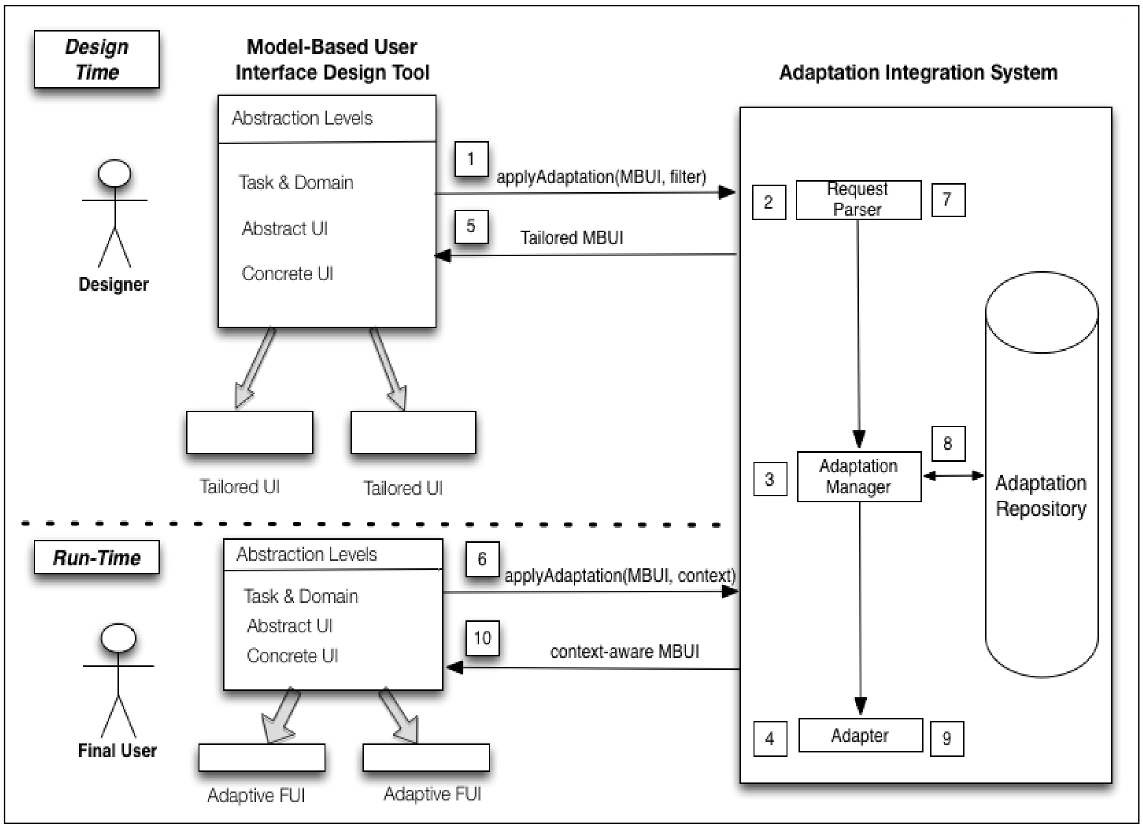


Fig. 3 Architecture of the Adaptation Integration System.

3.3.2 Adaptation Manager

This module selects the adaptation rules corresponding to the query parameters. The adaptation rules are adequately sorted to be applied consistently in order to avoid conflicts among them. For this purpose, the Adaptation Manager retrieves the adaptation rules related to the specified disability and abstraction level from the Adaptation repository. There is an algorithm for ordering the rules based on the granularity level of each one. In order to establish this order among the adaptation rules, firstly, adaptation rules designed for the specific disability and the abstraction level are selected; afterwards, this selection is filtered considering whether they are for design time or for runtime scenarios. Finally, they are sorted considering the granularity level. It is possible to define a strategy for applying first the adaptation rules related to larger granularity levels. This may be sometimes necessary in order to ensure that all adaptations are correctly applied. For instance, if there is a rule changing a group of elements and another one changing only one of such elements, the adaption of the specific single element should not be hidden by the first rule.

These adaptation rules are defined in AAL-DL [12], a language designed to be applicable to languages consistent with the Cameleon Reference Framework, thus ensuring the compatibility with all the abstraction levels. The UML representation of the components of this language is shown in Figure 4. For example, we have applied it with user interfaces described in the Model-based lAnguage foR Interactive Applications (MARIA) [18], which is an engineered notation able to model dynamic interactive applications at various abstraction levels. We have used it as reference point since its definition is publicly available, it addresses the use of various interaction modalities and it is a well-engineered language that describes all the main aspects characterizing interactive applications. So far, we have not encountered any particular issue in transforming descriptions from other UIDLs in it. In addition, MARIA supports the abstractions of the Cameleon Reference Framework and the necessary semantics to integrate accessibility patterns. If a given user interface is not specified in MARIA, a language transformation is previously performed. In the same way, after applying all the adaptation rules, the generated user interface is transformed from MARIA to the original UIDL.

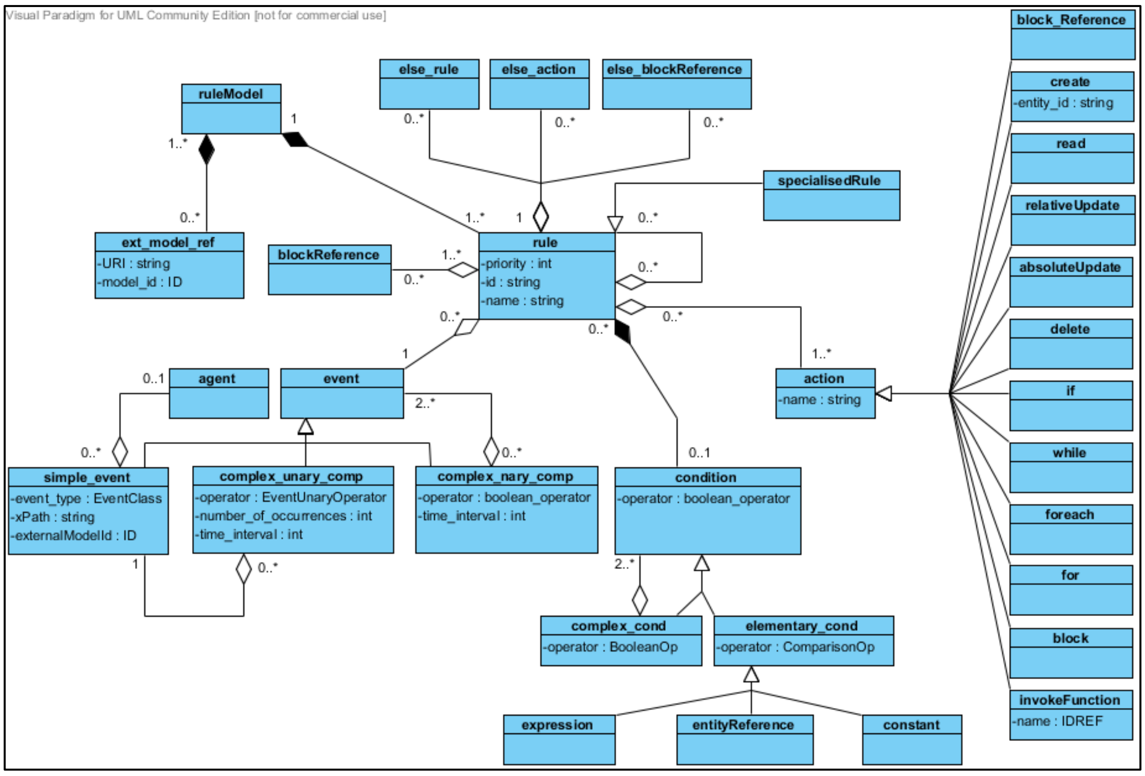


Fig. 4 Components described in AAL-DL.

3.3.3 Adapter

The Adapter module is in charge of sequentially applying the selected adaptation rules and language transformations to the logical description of the user interface, as illustrated in Figure 5. An accessible user interface is obtained as a result for this process. At design time, the designer will obtain a tailored user interface that satisfies the specified accessibility requirements. Whereas at runtime, the user will be provided with an accessible user interface tailored to the user and to the changed context of use.

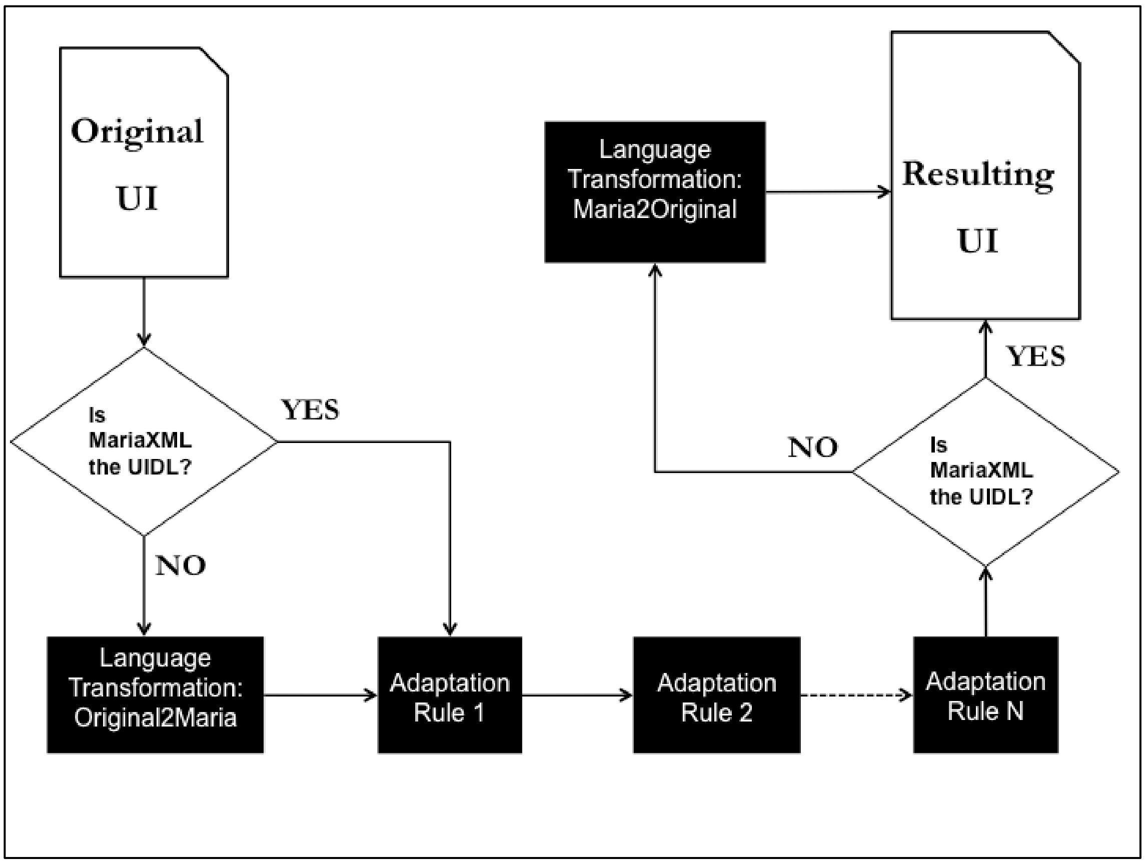


Fig. 5 Flow of the language transformations and the adaptation rules.

3.4. Adaptation Repository

The adaptation repository consists of a compilation of language transformations and adaptation rules devoted to people with special needs. It is worth pointing out that the language transformations and the adaptation rules are automatically provided by the system, consequently situations where the designers insert conflicting rules are minimized. Currently, the developers of the Adaptation Integration System are responsible for providing these transformations and adaptation rules to the system. Thus, designers do not have to learn neither MARIA syntax nor topics related to accessibility. Moreover, there is an XML file describing the meta-information related to each element of the repository. This file is necessary for adequately processing the elements as well as for selecting and effectively sorting them. The language transformations for transforming UIDLs to MARIA require minimum meta-information: the source of the transformation and the name of the original UIDL (indicated as a parameter). The adaptation rules devoted to people with disabilities require more meta-information. This meta-information conforms to the design space for the adaptation rules showed in Figure 6. The main components of this design space are the following:

User Disability. The disability parameter relates each adaptation rule to a particular disability. This is due to the fact that since not all the users have the same abilities, tailored user interfaces taking their specific disabilities into account satisfy better their needs than a "design for all" approach. In consequence, the Adaptation Integration System is able to adapt user interfaces to users' specific disabilities, if there are adaptation rules in the adaptation repository related to these disabilities. At design time, the designer selects the disabilities to be considered in his/her logical description of the user interface by interacting with the corresponding service. At runtime, the disabilities specified in the context model are matched with the adaptation rules. This approach can cause some conflicts, since the disability value of the context model may not coincide with the meta-information of the adaptation even if both have the same semantic meaning. In consequence, the Adaptation Integration System would not be able to select the proper adaptation rules. For instance, if an adaptation rule is related to the term "blindness" and the value for the disability parameter of the context model is "blind", the system is not currently able to identify that both terms are related to the same meaning. In next versions, we plan to integrate a mechanism for doing a proper semantic matching between these two terms. However, this issue goes beyond the scope of this paper.

In addition, specific rules for multiple impairments can be integrated in the system, tagging the rule with the specific disabilities. The only difference is that the designer (at design time) or the context model (at runtime) specifies more than one disability and the system selects the adaptations rules tagged with those disabilities.

Abstraction Level. The Cameleon Reference Framework [1] defines four different abstraction levels in the model–based user interfaces development process: Task and Domain, Abstract User Interface, Concrete User Interface and Final User Interface. The adaptations can be applied in any of these different levels at design time excepting at the final user interface level. For instance, adaptation rules related to task sequencing should be considered at the Task & Domain level, whereas adaptation rules related to some specific user interface modalities have to be considered at the concrete user interface level. At runtime, there are two different approaches: the adaptation rule can be related to the final user interface level if the changes to apply are minimum. However, this approach is not always viable. The final user interfaces can be implemented using different languages (HTML, Java, .Net, Android, etc.), but the adaptation rules would only work for one of these languages. This approach is suitable for ad-hoc projects when the final user interface language is previously defined. However, the research work presented in this paper aims to be compatible with the widest variety of systems. Therefore, a different approach is applied. It involves obtaining the necessary level of abstraction by means of an abstraction process, such as the one described in [1]. The MBUI tools are responsible for performing this task. Therefore, the MBUI tool provides the user interface in an appropriate level of abstraction in order to apply adaptation rules when a change in the context occurs. The results of this adaptation process are reflected in the final user interface. This process has been described in subsection 3.2.

Granularity levels. Adaptation rules can be applied at different granularity levels of the user interface: Application, Presentation, Group of Elements and Single Element. This parameter is required for sorting adequately the selected adaptation rules.

Adaptation Type. Tailored user interfaces are adapted at design time by service designers or by using a system such as the Adaptation Integration System and are instantiated at runtime. In the case of people with special needs, the interfaces are tailored considering the specific disabilities of the users. Thus, interfaces are adapted to their special needs. These are the default interfaces presented to the user. For instance, a blind user would be provided with a tailored user interface with an adequate structure of the headings, with a way to directly access the main content and with textual interaction elements. However, changes in the context of use have to be considered in some cases. Therefore, the user interface has to be adaptive in order to adapt to the context [19]. For example, when the user is walking the interface may adapt to this situation, or when the environment is noisy the audio elements in the interface may get textual. Therefore, context-aware adaptations are necessary in some cases. They may require changes of user interface modality (for instance, from audio to text) when the context changes are triggered.

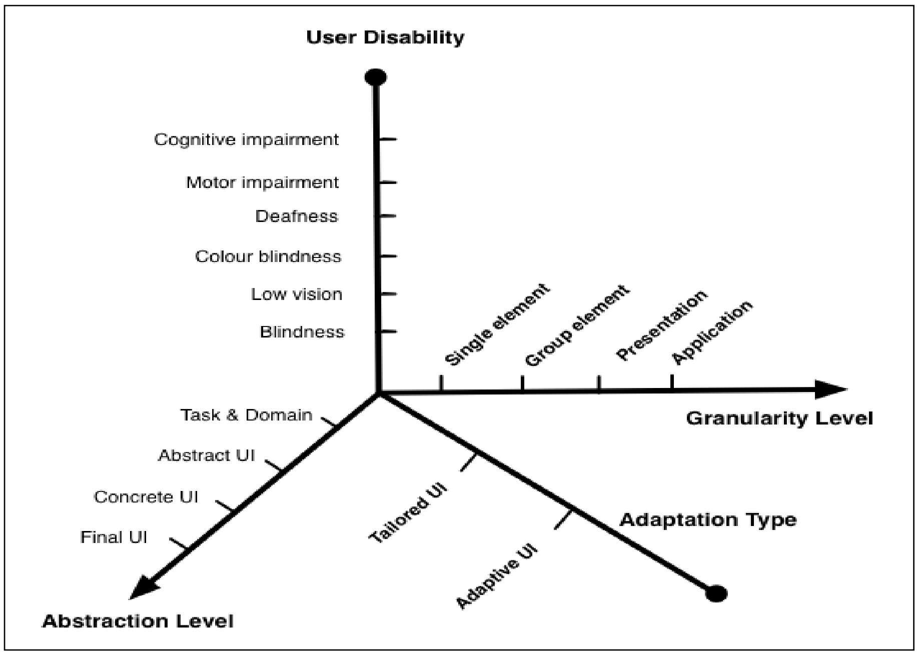


Fig. 6 Design space of the adaptation rules.

3.5. Examples of Adaptation Rules

As reported in Section 3.3, two classes of elements have been integrated into the repository: language transformations and adaptation rules for people with disabilities. The adaptation rules devoted to people with disabilities are derived from the analysis of different research works, such as the Barrier Walkthrough evaluation method [20]. Adaptation rules devoted to users with motor impairments were obtained from Gajos et al. (2010) [2]. Techniques focused on cognitive and sensory impairments were collected from Kurniawan et al. (2006) [21], from National

Institute on Aging and National Library of Medicine [22] and Richards et al. (2004) [23]. Finally specific techniques for visual impairments were obtained from Lunn, D. et al. (2008) [24]. Conversely, the language transformations for transforming the UIDL can be defined analysing each UIDL specification.

In this section some examples of adaptation rules devoted to people with disabilities are presented to provide a better understanding of the approach. Other examples can be found in [25]. Table 1 lists the parameters of each adaptation rule described in this subsection and the meta-information related according to the design space described before. Figures 7 and 8 provide some excerpts of code related to Rule 1. Note that the event part of the rule component is not considered in adaptation rules devoted to tailored user interfaces. These rules have been designed for design time so there is not any context change event triggering the adaptation rule. They are directly integrated in the user interface when the designer interacts with the Adaptation Integration System at design time.

Table 1 Parameters of some adaptation rules

Attributes / **Rules**	**Tailored/Adaptive**	**User Disability**	**Granularity Level**	**Abstraction Level**
1	Tailored UI	Blindness	Group	Abstract UI
2	Tailored UI	Low Vision	Single	Concrete UI
3	Tailored UI	Colour Blindness	Presentation	Concrete UI
4	Adaptive UI	Paraplegia	Application	Concrete UI
5	Adaptive UI	Low Vision	Application	Concrete UI
6	Adaptive UI	Parkinson	Application	Concrete UI

Rule 1

- Condition: the user is blind and he/she accesses an application with many interaction elements.
- Action: a table of content is created to easily access each interaction element.

Rule 2

- Condition: the size of the text of the user interface is smaller than 14 px. and the user has low vision.
- Action: increase the size of the text of the user interface to 14 px.

Rule 3

- Condition: the user is colour-blind.
- Action: change the foreground colour to black and background colour to white.

Rule 4

- Event: a wheelchair starts to move.

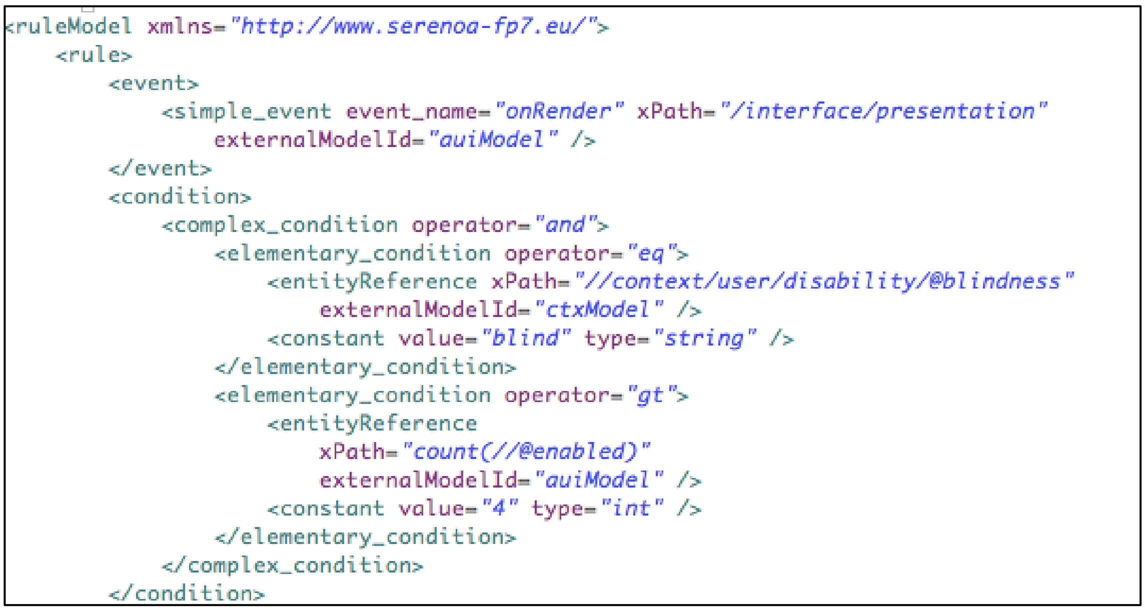
- Condition: the user has paraplegia and the user interface is not rendered with the vocal modality.
- Action: the user interface modality is changed to the vocal modality.

Rule 5

- Event: the user's walking speed is increased.
- Condition: the speed of the user while walking is greater than 3 km/h AND the user has low vision AND the modality of the user interface is graphical.
- Action: the user interface modality is changed to multimodal modality to allow the user selecting the most confortable modality for him/her while walking.

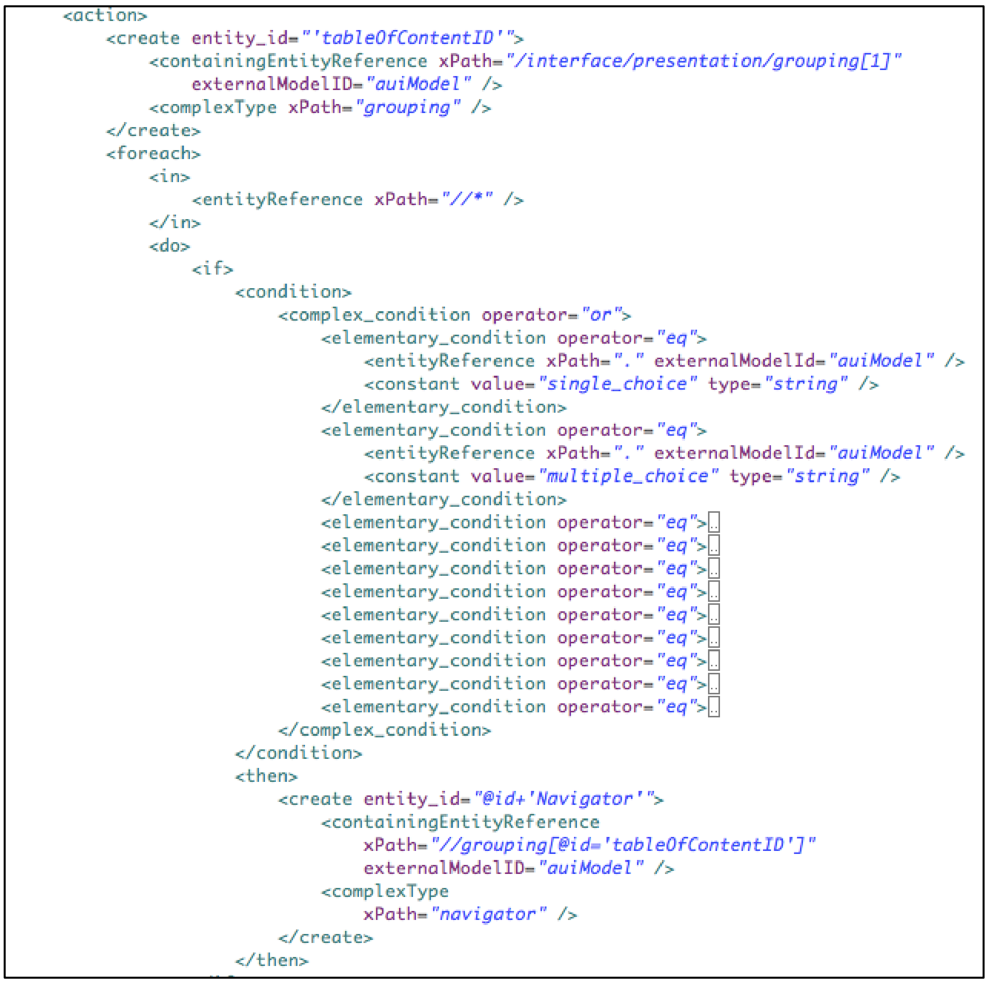
Rule 6

- Event: the user starts to move.
- Condition: the user has Parkinson disease AND the user interface is not of the type multimodal.
- Action: the user interface is changed to the multimodal type.



```xml
<ruleModel xmlns="http://www.serenoa-fp7.eu/">
    <rule>
        <event>
            <simple_event event_name="onRender" xPath="/interface/presentation"
                externalModelId="auiModel" />
        </event>
        <condition>
            <complex_condition operator="and">
                <elementary_condition operator="eq">
                    <entityReference xPath="//context/user/disability/@blindness"
                        externalModelId="ctxModel" />
                    <constant value="blind" type="string" />
                </elementary_condition>
                <elementary_condition operator="gt">
                    <entityReference
                        xPath="count(//@enabled)"
                        externalModelId="auiModel" />
                    <constant value="4" type="int" />
                </elementary_condition>
            </complex_condition>
        </condition>
```

Fig. 7 An excerpt of the specification of Adaptation Rule 1's event and condition elements. The event section specifies that the adaptation rule is applied when the user interface is rendered. The condition part describes a complex condition containing two elementary conditions. Both elementary conditions have to be satisfied to apply the adaptation rule. The first one is satisfied if the user is blind and the second one if there are more than four interaction elements enabled in the user interface.



```
<action>
    <create entity_id="'tableOfContentID'">
        <containingEntityReference xPath="/interface/presentation/grouping[1]"
            externalModelID="auiModel" />
        <complexType xPath="grouping" />
    </create>
    <foreach>
        <in>
            <entityReference xPath="//*" />
        </in>
        <do>
            <if>
                <condition>
                    <complex_condition operator="or">
                        <elementary_condition operator="eq">
                            <entityReference xPath="." externalModelId="auiModel" />
                            <constant value="single_choice" type="string" />
                        </elementary_condition>
                        <elementary_condition operator="eq">
                            <entityReference xPath="." externalModelId="auiModel" />
                            <constant value="multiple_choice" type="string" />
                        </elementary_condition>
                        <elementary_condition operator="eq">…
                        <elementary_condition operator="eq">…
                        <elementary_condition operator="eq">…
                        <elementary_condition operator="eq">…
                        <elementary_condition operator="eq">…
                        <elementary_condition operator="eq">…
                        <elementary_condition operator="eq">…
                        <elementary_condition operator="eq">…
                        <elementary_condition operator="eq">…
                    </complex_condition>
                </condition>
                <then>
                    <create entity_id="@id+'Navigator'">
                        <containingEntityReference
                            xPath="//grouping[@id='tableOfContentID']"
                            externalModelID="auiModel" />
                        <complexType
                            xPath="navigator" />
                    </create>
                </then>
```

Fig. 8 Representation of the action element of the Adaptation Rule 1. If Event and Condition parts are satisfied (see Figure 7), the action part creates a container with the value *tableOfContentId* for the id attribute. The aim of this container is to allocate the links for the table of contents. Then, for each element of the user interface that represents an interaction element (*text_field* element, *single_choice* element, *multiple_choice* element, etc.), a link is generated and inserted in the container previously created.

4 CASE STUDY

The objective of this case study is to assess that the Adaptation Integration System is able to provide different model-based tools based on the Cameleon Reference Framework with adaptation rules dedicated to including accessibility requirements and to support context-awareness, both at design time and runtime. Consequently, we wanted to consider the following hypotheses:

- The Adaptation Integration System is able to tailor a user interfaces provided by a MBUI tool at design time.
- The Adaptation Integration System is able to receive user interfaces created by using a MBUI tool and make them adaptive at runtime.
- The Adaptation Integration System supports different UIDLs based on the Cameleon Reference Framework.

To this end, two different applications have been developed using different MBUI tools and the Adaptation Integration System. For the first application, some adaptation rules are applied at design time in order to obtain user-tailored interfaces (Application 1). This application is devoted to assessing adaptation during design time at the abstract user interface level and the normalization of the UIDL by the use of language transformations. The second application is devoted to demonstrating the runtime adaptation (Application 2). It presents an example where the modality of the user interface is changed from graphical to vocal, at the concrete user interface level.

4.1. Application 1: Design Time

This application supports making appointments with a doctor. The following interaction elements had to be included in the application:

- Medical Number
- Birth Date
- Hospital Name
- Doctor Name
- Date of the Appointment
- Time
- Reason of the appointment
- Submit Button
- Confirmation of the Appointment

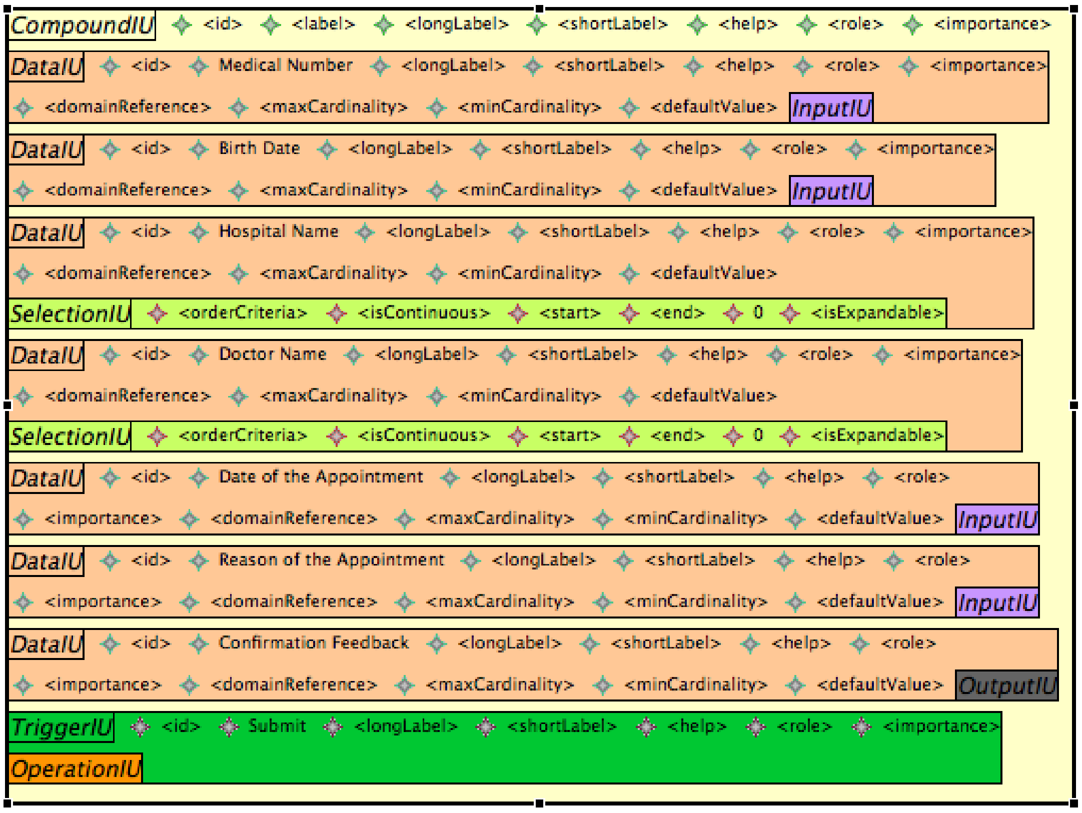
The development of the application was performed following a model-based approach. SPA4USXML [26], a tool that assists designers integrating accessibility requirements in the task and abstract models of the User Interface eXtended Markup Language (UsiXML) [27], was used in order to create the abstract user interface. Figure 9 shows a screenshot of the abstract user interface created by this tool. It was sent to the Adaptation Integration System indicating that this is an abstract user interface and requesting a tailored interface for blind people. The adaptation process consists of the following steps:

- The designer submits the abstract user interface to the Adaptation Integration System.
- The system detects that the original UIDL is UsiXML and applies the UsiXML to MARIA language transformation.
- Adaptation rules devoted to people with blindness are selected and applied. Figure 10 shows a screenshot of the result of applying the adaptation rule to the abstract user interface. Concretely, one adaptation rule is selected and applied:

   - Adaptation rule related to the WCAG 2.0 2.4.5 success criterion “provide blind users with a table of content to access easily the content” (see Rule 1 in Subsection 3.5)

- The MARIA to UsiXML language transformation is applied.

Afterwards, the designer continues the development process generating the concrete user interface. The final user interface will be tailored for people with blindness as the necessary accessibility requirements have already been integrated at the abstract level and are supported in the concrete user interface.



CompoundIU <id> <label> <longLabel> <shortLabel> <help> <role> <importance>
DataIU <id> Medical Number <longLabel> <shortLabel> <help> <role> <importance>
<domainReference> <maxCardinality> <minCardinality> <defaultValue> InputIU
DataIU <id> Birth Date <longLabel> <shortLabel> <help> <role> <importance>
<domainReference> <maxCardinality> <minCardinality> <defaultValue> InputIU
DataIU <id> Hospital Name <longLabel> <shortLabel> <help> <role> <importance>
<domainReference> <maxCardinality> <minCardinality> <defaultValue>
SelectionIU <orderCriteria> <isContinuous> <start> <end> 0 <isExpandable>
DataIU <id> Doctor Name <longLabel> <shortLabel> <help> <role> <importance>
<domainReference> <maxCardinality> <minCardinality> <defaultValue>
SelectionIU <orderCriteria> <isContinuous> <start> <end> 0 <isExpandable>
DataIU <id> Date of the Appointment <longLabel> <shortLabel> <help> <role>
<importance> <domainReference> <maxCardinality> <minCardinality> <defaultValue> InputIU
DataIU <id> Reason of the Appointment <longLabel> <shortLabel> <help> <role>
<importance> <domainReference> <maxCardinality> <minCardinality> <defaultValue> InputIU
DataIU <id> Confirmation Feedback <longLabel> <shortLabel> <help> <role>
<importance> <domainReference> <maxCardinality> <minCardinality> <defaultValue> OutputIU
TriggerIU <id> Submit <longLabel> <shortLabel> <help> <role> <importance>
OperationIU

Fig. 9 Original abstract user interface devoted to making appointments with the doctor. This abstract user interface provides four interaction elements of type *inputIU* to insert a value (*Medical Number*, *Birth Date*, *Date of the Appointment* and *Reason of the Appointment*), two interaction elements of type *SelectionIU* (*Hospital Name* and *Doctor Name*) for selecting a value from a set of options, one interaction element of type *OutputIU* for providing feedback to the user when the form is submitted (*Confirmation Feedback*) and one interaction element of type *OperationIU* for submitting the form (*Submit*).

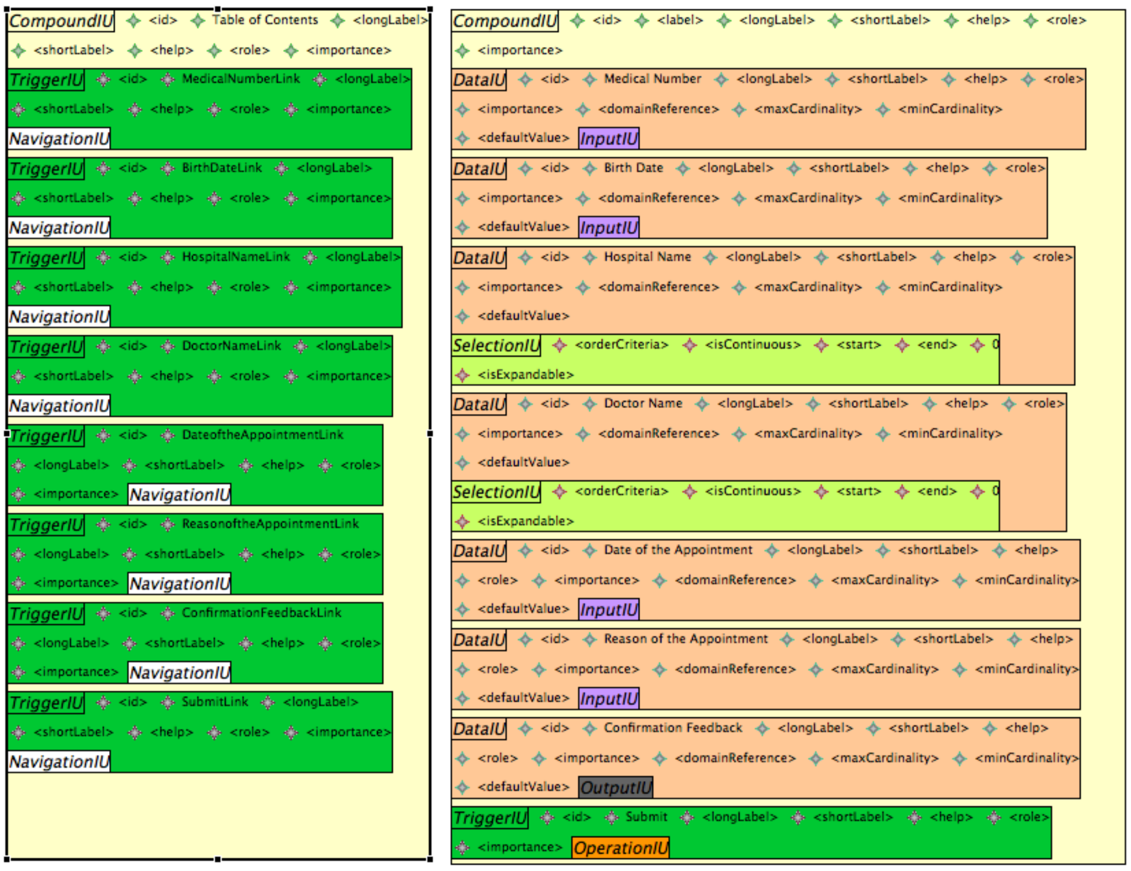

CompoundIU <id> Table of Contents <longLabel> <shortLabel> <help> <role> <importance>
TriggerIU <id> MedicalNumberLink <longLabel> <shortLabel> <help> <role> <importance> NavigationIU
TriggerIU <id> BirthDateLink <longLabel> <shortLabel> <help> <role> <importance> NavigationIU
TriggerIU <id> HospitalNameLink <longLabel> <shortLabel> <help> <role> <importance> NavigationIU
TriggerIU <id> DoctorNameLink <longLabel> <shortLabel> <help> <role> <importance> NavigationIU
TriggerIU <id> DateoftheAppointmentLink <longLabel> <shortLabel> <help> <role> <importance> NavigationIU
TriggerIU <id> ReasonoftheAppointmentLink <longLabel> <shortLabel> <help> <role> <importance> NavigationIU
TriggerIU <id> ConfirmationFeedbackLink <longLabel> <shortLabel> <help> <role> <importance> NavigationIU
TriggerIU <id> SubmitLink <longLabel> <shortLabel> <help> <role> <importance> NavigationIU
CompoundIU <id> <label> <longLabel> <shortLabel> <help> <role> <importance>
DataIU <id> Medical Number <longLabel> <shortLabel> <help> <role> <importance> <domainReference> <maxCardinality> <minCardinality> <defaultValue> InputIU
DataIU <id> Birth Date <longLabel> <shortLabel> <help> <role> <importance> <domainReference> <maxCardinality> <minCardinality> <defaultValue> InputIU
DataIU <id> Hospital Name <longLabel> <shortLabel> <help> <role> <importance> <domainReference> <maxCardinality> <minCardinality> <defaultValue>
SelectionIU <orderCriteria> <isContinuous> <start> <end> <isExpandable>
DataIU <id> Doctor Name <longLabel> <shortLabel> <help> <role> <importance> <domainReference> <maxCardinality> <minCardinality> <defaultValue>
SelectionIU <orderCriteria> <isContinuous> <start> <end> <isExpandable>
DataIU <id> Date of the Appointment <longLabel> <shortLabel> <help> <role> <importance> <domainReference> <maxCardinality> <minCardinality> <defaultValue> InputIU
DataIU <id> Reason of the Appointment <longLabel> <shortLabel> <help> <role> <importance> <domainReference> <maxCardinality> <minCardinality> <defaultValue> InputIU
DataIU <id> Confirmation Feedback <longLabel> <shortLabel> <help> <role> <importance> <domainReference> <maxCardinality> <minCardinality> <defaultValue> OutputIU
TriggerIU <id> Submit <longLabel> <shortLabel> <help> <role> <importance> OperationIU

Fig. 10 Tailored abstract user interface for appointments with the doctor. This abstract user interface represents the original application with a table of content added. The yellow rectangle on the left area of the figure represents the table of content inserted after applying the adaptation rules selected, whereas the yellow rectangle on the right area of the figure represents the elements of the original application. The table of content includes a container including a link targeting to each interaction element of the elements of the right area. For instance, on the left area the first *navigationUI* element is a link identified as *MedicalNumberLink*. This link targets to the first element on the right area corresponding to an element of type *InputIU* identified as *MedicalNumber*.

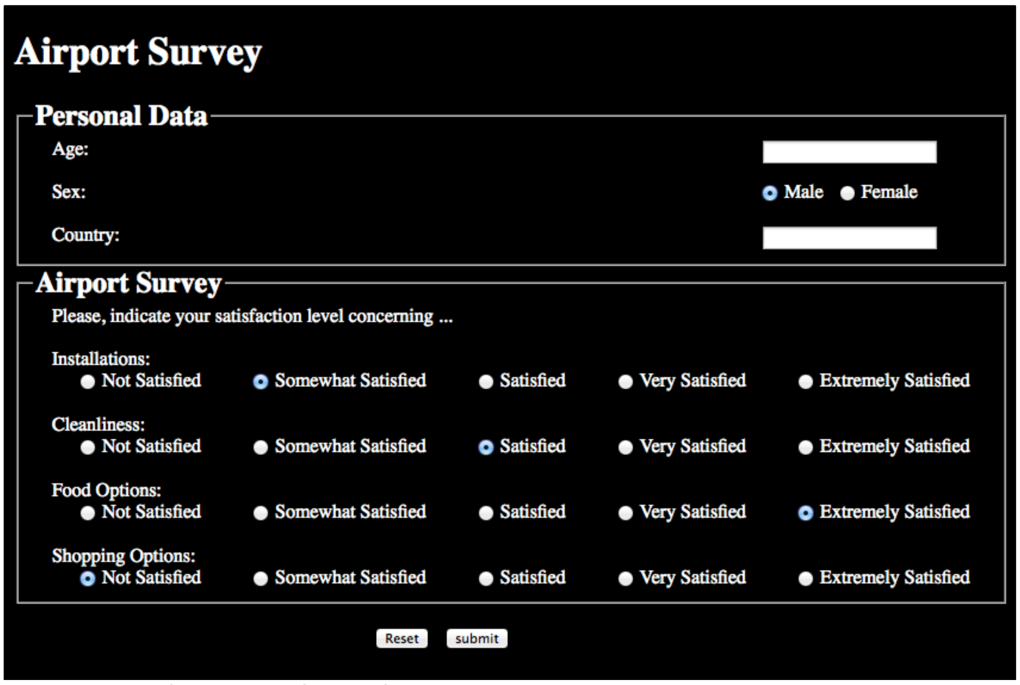
4.2. Application 2: Runtime

The second application shows the runtime adaptation. It is an Airport survey application implemented in HTML (see Figure 11) built by using MARIA as underlying UIDL. In this application, the user interface needed to be adaptive to the context of a person with paraplegia using a wheelchair. The elements of the context model used for this application are shown in Figure 12.

As described in the subsection 3.2, the service has to be integrated with a context manager system for detecting the relevant changes in the context. In this concrete scenario, the context change detected is that the user's wheelchair starts to move and that his/her disability is paraplegia. The process for adapting the user interface was the following:

- The service provides the correspondent concrete user interface version of the user interface (See Figure 13) and the context model states ("the user's wheelchair starts to move" and "the user has paraplegia") as input parameters for the Adaptation Integration System.

- The Adaptation Integration System identifies one rule to be applied:
    - Change graphical modality to vocal modality (Rule 4 described in Section 3.5)
- The tailored concrete user interface is generated (see Figure 14) applying the adaptation rule to the original concrete user interface. Therefore, vocal interaction elements are integrated allowing the user to interact by voice.
- The original service gets the result and reflects the changes in the final user interface.



Airport Survey

Personal Data

Age:

Sex: [✓ Male] Female

Country:

Airport Survey

Please, indicate your satisfaction level concerning ...

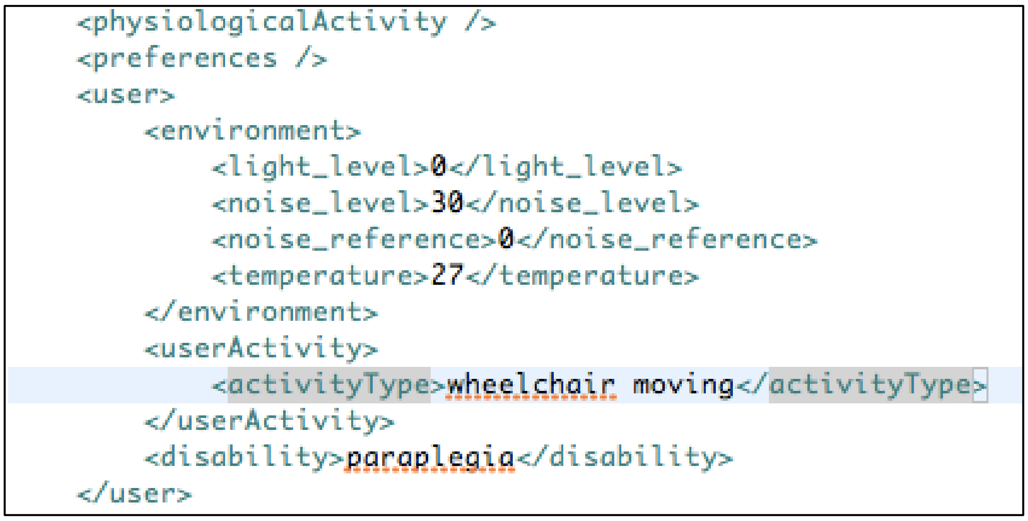
Installations: Not Satisfied [✓ Somewhat Satisfied] Satisfied Very Satisfied Extremely Satisfied

Cleanliness: Not Satisfied Somewhat Satisfied [✓ Satisfied] Very Satisfied Extremely Satisfied

Food Options: Not Satisfied Somewhat Satisfied Satisfied Very Satisfied [✓ Extremely Satisfied]

Shopping Options: [✓ Not Satisfied] Somewhat Satisfied Satisfied Very Satisfied Extremely Satisfied

Fig. 11 HTML of the Airport Survey Service.



```
<physiologicalActivity />
<preferences />
<user>
    <environment>
        <light_level>0</light_level>
        <noise_level>30</noise_level>
        <noise_reference>0</noise_reference>
        <temperature>27</temperature>
    </environment>
    <userActivity>
        <activityType>wheelchair moving</activityType>
    </userActivity>
    <disability>paraplegia</disability>
</user>
```

Fig. 12 Context Model of the application.

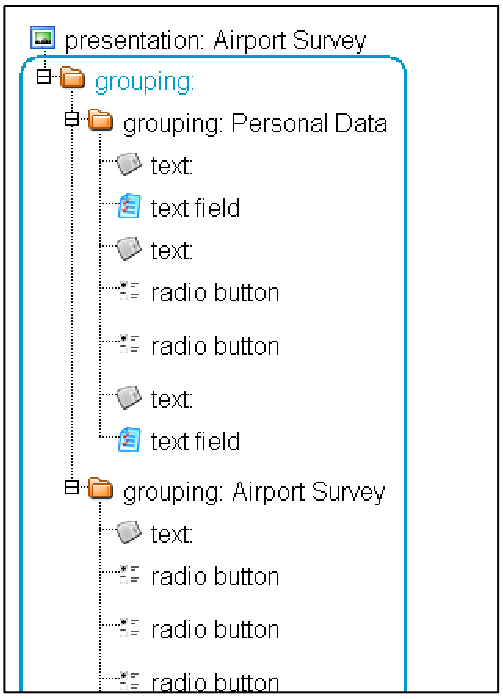


Fig. 13 Excerpt of the original concrete user interface.

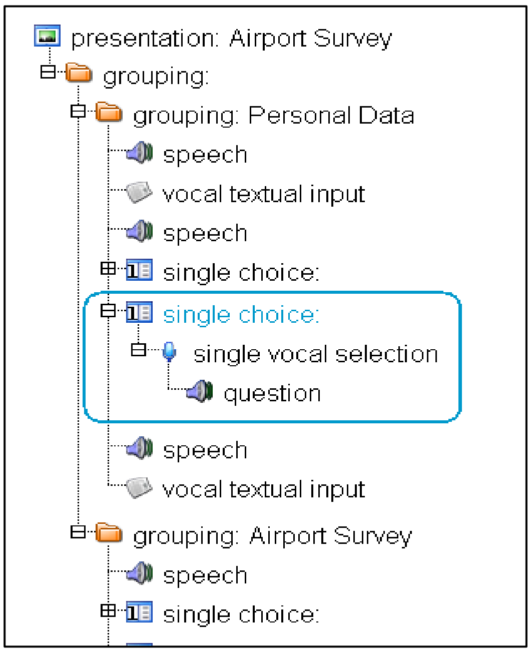


Fig. 14 Excerpt of the tailored concrete user interface with the vocal modality.

4.3. Discussion

Both case studies provided positive feedback. The first case study shows that the SPA4USXML tool provided with an abstract user interface described in UsiXML, which was tailored considering the needs of blind users. In the second case study, the runtime adaptation is illustrated by modifying the modality of a concrete user interface from graphical to vocal.
Despite having achieved positive feedback, more studies have to be carried out for a complete validation. In both case studies only one adaptation rule has been applied. Therefore, the system has to support a greater number of adaptation rules to evaluate, and it should be able to detect specific situations that can cause conflicts among the adaptation rules. In addition, no problems have been found in the transformations at the abstract user interface level from UsiXML to MARIA and vice versa. However, more specific tests have to be performed to adequately assess the support of the other abstraction levels and to identify whether there is specific semantics that cannot be transferred between both UIDLs.

5 Conclusions and Future Work

In this paper we presented the Adaptation Integration System aiming at integrating accessibility requirements in model-based user interfaces, both at design time and at runtime. Two case studies have been described, showing how this approach can support application designers in the process of integrating accessibility requirements in a transparent way. As consequence, designers inexperienced in the area of accessibility can benefit from such approach, since it does not require special accessibility knowledge to apply it. In the same way, both case studies also illustrate that it is useful for experienced designers producing accessible interfaces, because it decreases the development time necessary for building a user interface tailored to the needs of each user type, since designers do not have to implement, manually, each adaptation for each group of users. Moreover, the case studies also indicate that model-based context-aware applications based on the Cameleon Reference Framework, even not described in MARIA, can benefit from the Adaptation Integration System since it ensures the accessibility of adaptive user interfaces when there is a change in the context of use.

The two applications provided in the paper show that the approach works both at design time and at runtime and that it is compatible with different model-based tools based on Cameleon Reference Framework supporting different UIDLs.

Future work will be dedicated to improving the Adaptation Integration System in order to:

- Test the transformations between MARIA and UsiXML at all the abstraction levels of the Cameleon Reference Framework and to integrate UIDLs that do not follow this framework. Indeed, other UIDLs do not follow the same structure and semantic provided by the Cameleon Reference Framework. Therefore, there is not a direct mechanism to map the elements among different structures. In order to solve this obstacle, it is necessary to identify the semantic of the UIDL elements and to map each element to the correspondent element of the Cameleon Reference Framework.
- Add a new granularity level type called "External" [28]. This granularity level would indicate that a given user requires a specific assistive technology for interacting with the user interface. Consequently, the adaptation rule would consist in initializing that assistive technology provided it had been previously installed.
- Develop a graphical tool to allow experts to include new language transformations and new adaptation rules in the Adaptation Integration System. This approach would need a system for checking the correctness of the new rules. In this way, the adaptation repository will be globally accessible to enable the community the use of the repository by other systems and the inclusion of new adaptation rules. In this version, some conflicts can arise when adaptation rules and the context models describe an element with the same semantic by means of two different terms.
- Integrate a semantic matching mechanism into the system for identifying when different terms correspond to the same meaning in order to overcome this issue.

Acknowledgements

This research work has been partly funded by the Spanish Ministry of Science and Innovation ModelAccess project (grant TIN2010-15549), and by the EU FP7 STREP SERENOA project (http://www.serenoa-fp7.eu). The EGOKITUZ Research Lab is funded by the Department of Education, Universities and Research of the Basque Government (grant IT395-10). In addition, Raúl Miñón holds a Ph.D. scholarship from the Research Staff Training Programme of the same Department.

References

1. Calvary G, Coutaz J, Bouillon L, Florins M, Limbourg Q, Marucci L, Paternò F, Santoro C, Souchon N, Thevenin D, Vanderdonckt J (2002) The CAMELEON reference framework. In: Deliverable 1.1, CAMELEON Project, 2002. http://www.w3.org/2005/Incubator/model-based-ui/wiki/Cameleon_reference_framework. Accessed 09 July 2014.
2. Gajos KZ, Weld DS, Wobbrock JO (2010) Automatically Generating Custom User Interfaces for Users with Physical Disabilities. J. Artificial Intelligence, 174:12-13:910-950.
3. Abascal J, Aizpurua A, Cearreta I, Gamecho B, Garay-Vitoria N, Miñón R (2011) Automatically Generating Tailored Accessible User Interfaces for Ubiquitous Services. Proc. of the 13th Int. ACM SIGACCESS Conf. on Computers and Accessibility, ASSETS, 187-194.
4. Bongartz S, Jin Y, Paternò F, Rett J, Santoro C, Spano LD (2012) Adaptive User Interfaces for Smart Environments with the Support of Model-Based Languages. In: Paternò F, de Ruyter B, Markopoulos P, Santoro C, van Loenen E, Luyten K (eds.) Ambient Intelligence, LNCS, 7683:33-48.
5. Daniel F, Matera M, Pozzi G (2008) Managing Runtime Adaptivity through Active Rules: the Bellerofonte Framework. J. Web Eng., 7:3:179-199.
6. Stephanidis C (2001) Adaptive Techniques for Universal Access. J. on User Modelling and User-Adapted Interaction, 11:1-2:159-179.
7. Yang SJH, Shao NWY (2007) Enhancing Pervasive Web Accessibility with Rule-Based Adaptation Strategy. J. on Expert Systems with Applications, 32:4:1154-1167.
8. Quade M, Blumendorf M, Albayrak S (2010) Towards Model-Based Runtime Evaluation and Adaptation of User Interfaces. Proc. of the 1st Int. Workshop on User Modelling and Adaptation for Daily Routines (UMADR): Providing Assistance to People with Special and Specific Needs, 31-36.
9. Peissner M, Häbe D, Janssen D, Sellner T (2012) MyUI: generating accessible user interfaces from multimodal design patterns. Proc. of the 4th ACM SIGCHI symposium on Engineering interactive computing systems 81-90.
10. Garcia A, Sánchez J, Sánchez V, Hernández JA (2012) Integration of a Regular Application into a User Interface Adaptation Engine in the MyUI Project. In: Miesenberger K, Karshmer A, Penaz P, Zagler W (eds) Computers Helping People With Special Needs, 13th International Conference, ICCHP, Linz, Austria. Proc. Part I, L.N.C.S., Springer Verlag, Berlin 7382:311-314.
11. CakePHP framework, http://cakephp.org Accessed 09 July 2014.

12. Serenoa Project, deliverable 3.3.2 AAL-DL: Semantics, Syntaxes and Stylistics, http://www.serenoa-fp7.eu/wp-content/uploads/2013/09/SERENOA_D3.3.2.pdf, Accessed 09 July 2014.
13. Miñón R, Moreno L, Martínez P, Abascal J (2014) An Approach to the Integration of Accessibility Requirements into a User Interface Development Method. In: Vanderdonckt J, López-Jaquero V (eds.) International Journal Science of Computer Programming (SCP), Special Issue on Tool Support for User Interface Description Languages 86:58-73
14. Vanderheiden GC, Treviranus J (2011) Creating a Global Public Inclusive Infrastructure. In: Proc. of HCI International, LNCS 6765, 5:517-526.
15. Gamecho B, Miñón R, Abascal J (2013) Design Issues in Accessible User Interface Generation for Ubiquitous Services through Egoki. In the 12th European AAATE Conference, 1304-1309.
16. Ghiani G, Manca M, Paternò F, Porta C (2014) Beyond Responsive Design: Context-Dependent Multimodal Augmentation of Web Applications. Proc. of 11th International Conference on Mobile Web Information Systems, MobiWis, LNCS 8640, 71-85.
17. Serenoa project: deliverable 4.4.1, Context of Use Runtime and Infrastructure, http://www.serenoa-fp7.eu/wp-content/uploads/2012/07/SERENOA_D4.4.1.pdf. Accessed 09 July 2014.
18. Paternò F, Santoro C, Spano LD (2009) MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment. ACM Transactions on Computer-Human Interaction, 16:4:19:1-19:30.
19. Model-Based UI XG Final Report, http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/. Accessed 29 December 2014.
20. Barrier Walkthrough Procedure, http://sole.dimi.uniud.it/~giorgio.brajnik/projects/bw/bw.html. Accessed 09 July 2014.
21. Kurniawan SH, King AD, Evans G, Blenkhorn PL (2006) Personalising web page presentation for older people. Interacting with Computers 18:3:457-477.
22. National Institute on Aging and National Library of Medicine. Making Your Web Site Senior Friendly: A Checklist. , NIH & NLM, 2002, http://www.nlm.nih.gov/pubs/checklist.pdf. Accessed 09 July 2014.
23. Richards JT, Hanson VL (2004) Web Accessibility: A Broader View. Proc. of the 13th Int. conference on World Wide Web, 72-79.
24. Lunn D, Bechhofer S, Harper S (2008) The SADIe transcoding platform. Proc. of the 2008 international cross-disciplinary conference on Web accessibility, W4A, 128-129.
25. Adaptation Repository Web Page, http://sipt07.si.ehu.es/aptrep/html/index.html. Accessed 09 July 2014.
26. Miñón R, Moreno L, Abascal J (2013) A Graphical Tool to Create User Interface Models for Ubiquitous Interaction Satisfying Accessibility Requirements. Universal Access in the Information Society, 12:1-13.
27. Limbourg Q, Vanderdonckt J, Michotte B, Bouillon L, López V (2005) UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In: Bastide R, Palanque P, Roth J (eds.) Engineering Human Computer Interaction and Interactive Systems LNCS 3425:200-220.

28. Miñón R, Paternò F, Arrue M (2013) An environment for designing and sharing adaptation rules for accessible applications. Proc. of the Int. Conf. EICS, 43-48.