# A Practical Solution for the Automatic Generation of User Interfaces – What Are the Benefits of a Practical Solution for the Automatic Generation of User Interfaces?

Miroslav Sili[1], Christopher Mayer[1,*], Martin Morandell[1], Matthias Gira[1], and Martin Petzold[2]

[1]AIT Austrian Institute of Technology GmbH, Health & Environment Department, Biomedical Systems, Donau-City-Str. 1, 1220 Vienna, Austria
[2] ProSyst Software GmbH, Dürener Str. 405, 50858 Köln, Germany
{miroslav.sili,christopher.mayer,martin.morandell, matthias.gira}@ait.ac.at, m.petzold@prosyst.com

**Abstract.** Older adults benefit from information and communication technology solutions in the Ambient Assisted Living (AAL) domain. The offered user interfaces for these ICT solutions often do not take the special needs, preferences and the physical and mental capabilities of older adults into account. The project AALuis focuses on solutions to increase accessibility, adaptability and usability of user interfaces in the AAL domain. The paper describes the functionality of the AALuis layer and the different steps involved stakeholders have to cover to benefit from the user interface generation framework. A detailed comparison between the traditional user interface design and the AALuis approach lists similarities and identifies differences in the user interface generation process.

**Keywords:** Ambient Assisted Living, Human-Computer Interaction, User Interface, Framework, Task Model, Automatic Adaptation.

## 1 Introduction

Needs and wishes regarding the interaction with ICT solutions change over time and vary between older adults. They depend on the user's physical and cognitive capabilities and his/her preferences. Thus, the user interface (UI), which can be critical to the success or failure of an ICT service, needs to be flexible and adaptable to support the user's abilities. AALuis[1] focuses on solutions within the Ambient Assisted Living (AAL) domain and provides an open middleware layer to guarantee accessible and usable UIs for different services [1].

[1] www.AALuis.eu

M. Kurosu (Ed.): Human-Computer Interaction, Part I, HCII 2014, LNCS 8510, pp. 445–456, 2014.

## 2 Objectives

The main objective of AALuis is to provide the possibility to connect different types of ICT services to various user interfaces and offers thereby an interaction in the user's preferred way. The basic concept is to detach the functionality of the service and its representation to the user, and to provide a standardized way to use the framework embedded into or connected to existing platforms in use, such as HOMER [2] or UMO [3]. This concept supports service developers (SD), user interface designers (UID) and service providers (SP) in providing the services in a usable and accessible way. The end-user (EU) is not directly affected by the separation of concerns, but can benefit from the consistent look and feel of user interfaces for various applications.

The developed solution closes the gap between the functionality of the service and the automatically generated UI (figure 1). Automatic UI generation and runtime adaptation enable reaction to changing needs and preferences regarding the interaction with ICT solutions over time and variations between older adults. As the group of older adults is very heterogeneous the personalization and customization possibilities offered to each single user by AALuis are of high importance.

AALuis uses an interaction model in Concur Task Trees (CTT) notation [4][5] describing the interaction flow of the service. The UI generation process takes into

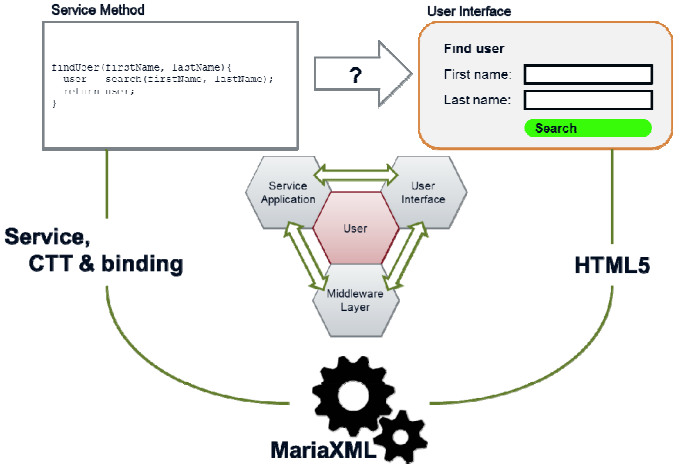


**Fig. 1.** The UI generation process in AALuis. Service methods and the interaction model in CTT notation are used to generate intermediate steps in MariaXML. The final transformation leads to the concrete UI in HTML5.

account the user's preferences and capabilities as well as the context of interaction (e.g., properties of available devices, etc.) [6]. The intermediate steps from the service description to the concrete UI in HTML5 are represented in Model based lAnguage foR Interactive Applications (MariaXML) [7].

Who are the main stakeholders to benefit from the UI generation framework and what do they need to do to benefit? The solution mainly helps service developers, service providers and finally older adults using the provided services and their functionality. To finally reach the beneficial solution, of customized service interaction, each stakeholder group has to conduct certain tasks, which are described in the methodology section.

## 3 Brief State of the Art

In recent years, an increasing amount of research focused on the user interface and thus on the representation of services in general and in particular for older adults. In the following some selected and relevant research projects and their approaches are described. GUIDE has focused on a novel adaptive accessibility framework and a characterization of individual users for creating accessible TV applications [8]. MyUI has addressed the provision of individualized UIs which are accessible to a broad range of users by the collection of information about the user during the interaction and updating the user profile accordingly [9]. EGOKI uses a similar approach as presented in the paper. It is based on the UCH [10], which acts as a middleware for ubiquitous interaction, and UIML for the abstract representation of the UIs [11]. The Universal Remote Console (URC) framework facilitates pluggable and handcraft user interfaces, which are designed for a specific target group, context of use and application based on the so-called (user interface) socket [12]. The project universAAL follows an automatic UI generation approach [13] and is based on the usage of XForms for the definition of an abstract data model combined with a set of abstract user interface components [14]. The Cameleon Reference Framework (CRF) distinguishes four layers of user interfaces, namely tasks and concepts, the abstract user interface, the concrete user interface, and the final user interface [15]. A similar concept is applied in the AALuis approach.

## 4 Methodology

In order to achieve the above mentioned objectives the layer is developed in a flexible way. The different involved stakeholder groups (figure 2) have to cover the following steps to benefit from the user interface generation framework.

Service developers (SD) need to develop the service functionality in a first step. The service can be either included as a web service or as a separate Open Service Gateway initiative (OSGi) [16] component. Besides the implemented service, the interaction model, which represents the logical activities to be conducted to reach the user's goal, has to be provided using the CTT notation, a W3C working draft [17]. Alongside, a binding file in XML format connects concrete service methods to its

corresponding CTT tasks. An optional content file can be used by the service developer to provide necessary additional resources for the UI generation (e.g., sign language videos, pictures). If there is a need, user interface designers (UID) can optionally update and change the used transformation rules for optimizing (e.g., corporate identity) the UI and add additional I/O modalities.

The service providers (SP) need to deploy the AALuis middleware layer and to provide access to the connected services. Additionally, they have to enable the I/O devices to be used by the end-user. These devices run a dedicated application responsible for communication with the AALuis layer and presenting the final UI. A user model can be selected and adapted to meet the user's preferences and to map his/her physical and cognitive capabilities, as used in the transformation process. These user preferences can be modified at any time (real-time) by either the end-user or his/her (in-) formal caregiver.

The end-user (EU) can directly use the AALuis service on all enabled I/O devices and benefit from the dynamically adaptable UIs.

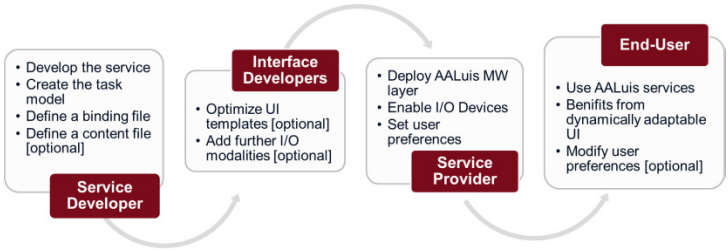


**Fig. 2.** Tasks to be fulfilled by the different stakeholder groups to benefit from AALuis

### 4.1 The User Interface Development Process

During the development process, service developers and UI designers have to consider several steps from the service on the one side, towards the final user interface on the other side. Table 1 depicts required and optional steps in a traditional, handcrafted UI design process and in the AALuis UI design process. The aim of the table is to uncover benefits, but also potential weaknesses, of the AALuis layer in comparison to a traditional, handcrafted UI design process.

The comparison illustrates that both approaches have some steps in common (steps A, D, E and F), but also that one step on the traditional approach (step H) and three steps on the AALuis approach (steps B, C and G) are not necessary in the other approach. The different colors in the table refer to the different stakeholders to be involved in each step. The blue color indicates the service developers, and in contrast the green colors stand for the user interface designer. The service provider is represented by the yellow colors in step I. The final step J is performed by the end-user interacting with the user interface and thus the service.

The table illustrates also mandatory steps (A, B, C, D, E for handcrafted approach and A, D, E for the AALuis approach) and optional steps (F and G for handcrafted approach and F and H for the AALuis approach).

**Table 1.** Comparison of required and optional steps that different stakeholder groups have to consider during the user interfaces generation process

| Step | Traditional UI generation process | AALuis UI generation process |
|---|---|---|
| A | (SD) Define the service (business logic) | |
| B | (UID) Select the target device (tablet, PC, TV, other) | - |
| C | (UID) Select the application type (web, native, hybrid) | - |
| D | (UID) Implement UIs for the target device | (SD) Create the interaction model |
| E | (UID) Implement connectors, handlers, listeners etc. to connect the service and the UIs | (SD) Create the binding file which defines connections between the service and the interaction model |
| F | (UID) Optional: Implement or adopt UIs for different target devices | (UID) Optional: Crate new transformations for new target devices which are not provided so far |
| G | (UID) Optional: Adopt UIs to special user needs | - |
| H | - | (UID) Optional: Adopt the transformation if the generated UIs do not fulfil the expected results |
| I | (SP) Publish the service and the UIs | |
| J | (EU) Use the service and the UIs | |

**Step A.** The separation of the service from the final user interface is an important issue in the UI development process. Both approaches have this step in common. Regardless of the approach – the business logic defined in a remote web service or in the same application – a back-end side is necessary, to which an UI can act as a front-end. A clear separation between this back-end and the UI allows service developers to focus on the service functionality, rather than on the user interface. They need not be concerned with requirements for specific users or specific target groups regarding service data representation. Responsibility for a suitable, user-specific representation of data and interaction of a service is at the UI designer or in the case of AALuis mainly in the layer itself.

As mentioned before, the AALuis layer is able to generate user interfaces for any kind of service. Generally a service can be described as a piece of software program

that is able to exchange data with a user or another service. By using this description it becomes clear, that AALuis can be used for remote located web services but also for the interaction with a local application or device. The following two examples illustrate the usage of the AALuis layer with two complementary types of services:

- AALuis in the context of interaction with a local heating control device in a smart home environment.
- AALuis in the context of an external mobile caregiver and its "meals on wheels ordering" web service.

To facilitate the service inclusion process for already existing services, but also to reduce frame conditions for new, upcoming services two different approaches for the service integration in the AALuis have been developed. Services can be integrated via the Simple Object Access Protocol (SOAP) [18] but also via the OSGi specification. The former offers a great opportunity for service developers because it allows the service implementation on any machine, any platform or in any programming language. The only two constrains for this approach are: a) the service is reachable via LAN or WAN and b) the service is accessible via SOAP. The latter approach, the service definition via the OSGi specification, is especially useful for local devices or for local services provided by an OSGI based AAL middleware platforms like HOMER and universAAL [19].

**Step B.** In the traditional UI generation process the UI designer has to select a specific target device or at least to be aware of its technical constraints. In contrast, the AALuis approach comes already with a default set of supported target devices, like the tablet/smartphone, PC or TV. Thus, using the AALuis layer neither service developers nor user interface designers have to select a specific target device for the new service.

**Step C.** Step C is closely related to Step B. In many cases application types are determined by the device constrains. However, in the traditional approach UI designers have to decide the best suited application type for the service and for the selected target device(s). Service developers using the AALuis approach do not need to decide this for a new service. The built in set of default target devices are already implemented as hybrid applications. They benefit from the native hard- und software advantages of the specific target device but use also web based methods for the communication with the AALuis layer.

**Step D.** Traditional, handcrafted UIs can be built in many different ways. UI designers may use their preferred programming langue to generate a graphical user interface (GUI), use HTML for web based UIs or proprietary implementations for specific devices. One of the advantages of this approach is the possibility to tailor a specific UI for a specific use case. At the same time, this can be considered a disadvantage of tailored UIs, since every new service requires a newly tailored user interface.

Regarding the interaction model, handcrafted UIs have an implicit user-service interaction model. The front-end knows how to handle user actions and how to send them to the service in the back-end. The services in the back-end know how to update the UI in the front-end. In contrast, AALuis needs, an explicit interaction model. The interaction model describes possible interaction steps between the user on the one side and the service on the other side in a formal way. This formal description becomes necessary when both sides have the potential to alternate. AALuis provides automatic generated user interfaces (alternation on the one side) for different services (alternation on the other side).

For the interaction model in AALuis the Concur Task Trees (CTT) notation is used. The CTT notation distinguishes between interaction, system, user and abstract tasks. Interaction tasks are performed by user interactions with the system. User tasks represent internal cognitive or physical activities performed by the user and abstract tasks are used for complex actions which need sub-tasks of different categories [20]. Service developers may use a graphical user interface tool, namely the Concur Task Trees Environment (CTTE), to design and test interaction models for their services [21].

**Step E.** Step E focuses on the connection between the service, and the user interface and the interaction model, respectively. Handcrafted UIs have usually a concrete connection to the service via a specific controller, some handlers or listeners. The UI is aware of the service in the back-end and vice versa. This awareness is not directly present in the AALuis approach, because the layer is designed to generate multiple user interfaces for multiple services. In this case, each service needs to be connected to the interaction model and not to a concrete user interface. Figure 3 illustrates two common and very often used design patterns (figure 3a and figure 3b) for software development and their connection between the front-end (viewer or user interface) and the back-end (model or service). Moreover, figure 3 clarifies the correlation related to the front-end-back-end communication between the design pattern in figure 3b and the AALuis approach in figure 3c.

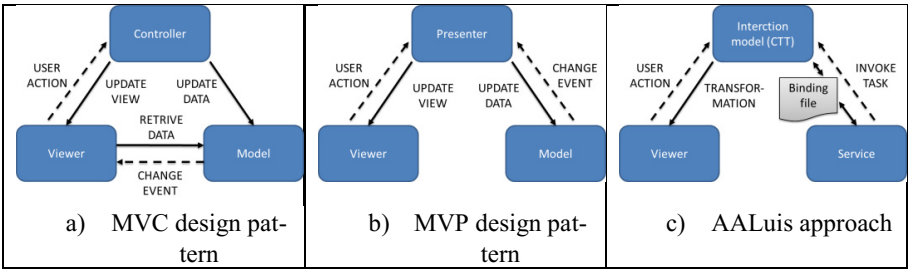


**Fig. 3.** Visualization of data and event flow in the a) Model Viewer Controller (MVC) design pattern, the b) Model Viewer Presenter (MVP) design pattern and in the c) AALuis approach

The MVC design pattern was developed at Xerox PARC in the late 1970's [22]. As illustrated in figure 3a, the pattern is based on three components the viewer, the controller and the model. The model contains the data which should be represented by the viewer. In most cases this component also contains the business logic and the back-end service, respectively. The model is loosely coupled with the viewer and separated from the controller. It may send notifications towards the viewer about change events in the related data. The viewer is responsible for rendering the retrieved model data. It also relays user actions towards the controller. The controller determinates how to act on user actions and updates the model data accordingly. In most cases the controller has a reference to the view and may cause the viewer to update the current shown view.

The MVP design pattern, as illustrated in figure 3b, was developed at Taligent in the 1990's [23][24]. The viewer and the model have the same functionality as in the MVC design pattern. In contrast to the MVC design pattern, the MVP completely separates the view from the model. In this case, the communication is carried out by the presenter. Seen from the viewer side, the presenter is responsible to receive user actions and to cause the viewer to update the current shown view. Considered from the model side, the presenter is responsible to receive change events from the model and to cause an update on the models data.

The AALuis approach, as illustrated in figure 3c, follows roughly a MVP design pattern. The interaction model acts as the presenter in the MVP design pattern and it is aware of the viewer but also of the service and model, respectively, or the business logic. The viewer in the AALuis approach achieves the same goal as in the MVP pattern. The main difference is that AALuis automatically generates the viewer on-the-fly and the interaction model is used as the basis for this transformation process. The service contains the data that the viewer should present but also the business logic. A simple binding file in XML format connects the interaction model with the service. Moreover the binding file defines the mapping between interaction tasks (CTT tasks) and input/output parameters of the service functionality.

**Step F.** Step F is optional in both approaches. In the traditional, handcrafted UI design process, the UI designer has to generate either new UIs for a new target device or at least to adapt the previously designed UI to its capabilities (e.g. screen resolution, I/O modality, etc.). This step may generate significant additional cost because the step needs to be repeated for every new service.

In the AALuis approach, the integration of a new, currently unsupported, target device is the first optional step that may require UI designer involvement. In contrast to the handcrafted approach, this adaption happens only once per new device and is independent of the service functionality. A device included in this way may serve afterwards various services without the need of a repetitive adaption.

The transformation is composed of three separate phases. The output of the first transformation is the Abstract User Interface (AUI), which is modality and device independent. Based on the AUI the Concrete User Interface (CUI) is generated. The CUI is more specific and modality and device dependent. The final phase uses the CUI to create the Renderable User Interface (RUI). This represents the final user

interface and is already enriched with preferred user settings. The AUI and the CUI are represented in MariaXML whereas the RUI may vary from used device and output modalities. The current implementation returns HTML5 as output but also other output formats like VoiceXML [25] are possible and already under development.

**Step G.** Step G, the adoption of user interfaces to special user needs, is optional for traditional UI creation, and obsolete for the AALuis approach. Traditionally, to achieve accessibility and making a service, or interaction, available to as many people as possible, demands a high level of expertise in the domain of assistive technologies and underlying impairments. A thorough understanding of the target group of the service at hand, and additionally multiple international standards, guidelines, like the Web Content Accessibility Guidelines (WCAG) [26] or legal obligations, may also apply.

This knowledge and practices have to be applied to every single user interface to achieve accessibility. Appliers of AALuis on the other hand, can forget the above mentioned methods and strategies. Accessibility is provided "out-of-the-box". The user generation process of AALuis utilizes a user context model, which is currently based on the MyUI approach. Through the description of the user, his/her preferences, capabilities and limitations, AALuis automatically selects suitable input/output devices and creates suitably adopted user interfaces for them. In contrast the traditional UI designer may have to target larger categories (e.g. visually impaired, hearing impaired, etc.) of users to limit resource spending. He also has to ensure that each user employs the "correct" UI, suited best for him.

Thus Step G, adopting the UI to a certain user is managed by the AALuis layer automatically and does not demand additional human interference, to achieve accessibility or usability.

Due to the nature of AALuis, it is also possible to react to future requirements of accessibility that go beyond the current state. Adopting the user profile, the transformation process, or adding device technology is possible. This would immediately benefit all other services and users as well.

**Step I.** Step I is mandatory for both approaches. The service providers have to distribute services and UIs accessible to the end-users. Unfortunately it is not possible to generalize the distribution possibilities for traditional UIs because this can be realized in different ways. Concerning AALuis, service providers have to fulfil the following distribution tasks:

- Deploy the AALuis middleware layer and make it available to the target devices via LAN or WAN.
- Initialize a default user preference set for each AALuis user.
- Enable and configure I/O devices so that they are able to connect to the deployed AALuis middleware layer.

**Step J.** The final Step J represents the end-user who interacts with the provided UIs and the underlying services. This step is also common for both approaches.

## 5 Results and Conclusion

This paper has given an overview of the functionality of the AALuis layer and its practical deployment of automatic user interface generation. The presented comparison between the traditional, handcrafted approach and the AALuis approach listed similarities, and identified differences in the UI generation process. Although both procedures are able to produce user interfaces that satisfy the needs and preferences of the end-user, the comparison has shown that the two approaches differ regarding the effort each stakeholder has to fulfil to achieve this goal. One of the most significant findings to emerge from this comparison is that the AALuis layer is able to generate UIs for different services without any involvement of the user interface designer. In general, the overall goal of AALuis project is to provide a common tool which is able to reduce the development costs for new and innovative user interfaces and services especially for the target group of older adults. AALuis is currently still in the development stage. The upcoming user trials will help to identify weaknesses in the UI generation process as well as in the interaction with the implemented services. AALuis will be released as open source in autumn 2014.

**Acknowledgement.** The project AALuis is co-funded by the AAL Joint Programme (REF. AAL-2010-3-070) and the following National Authorities and R&D programs in Austria, Germany and The Netherlands: bmvit, program benefit, FFG (AT), BMBF (DE) and ZonMw (NL).

## References

1. Mayer, C., Morandell, M., Hanke, S., Bobeth, J., Bosch, T., Fagel, S., et al.: Ambient Assisted Living User Interfaces. In: Gelderblom, G.J., et al. (eds.) Everyday Technology for Independence and Care, AAATE 2011. Assistive Technology Research Series, vol. 39, pp. 456–463. IOS Press (2011)
2. Fuxreiter, T., Mayer, C., Hanke, S., Gira, M., Sili, M., Kropf, J.: A modular platform for event recognition in smart homes. In: 2010 12th IEEE International Conference on e-Health Networking Applications and Services (Healthcom), pp. 1–6. IEEE (2010)
3. Verklizan: Intelligent software for monitoring centres, `http://verklizan.info/content/umo-platform/system-overview/` (accessed: January 2014)
4. Paternó, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction, INTERACT 1997, pp. 362–369. Chapman & Hall (1997)
5. Paternó, F.: Concur Task Trees: An Engineered Notation for Task Models. In: The Handbook of Task Analysis for Human-Computer Interaction, pp. 483–503. Lawrence Erlbaum Associates (2003)
6. Mayer, C., et al.: User interfaces for older adults. In: Stephanidis, C., Antona, M. (eds.) UAHCI/HCII 2013, Part II. LNCS, vol. 8010, pp. 142–150. Springer, Heidelberg (2013)

7. Paternó, F., Santoro, C., Spano, L.C.: MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact. 16, 19:1–19:30 (2009)
8. Duarte, C., Langdon, P., Jung, C., Coelho, J., Biswas, P., Hamisu, P.: GUIDE: Creating Accessible TV Applications. In: Gelderblom, G.J., et al. (eds.) Everyday Technology for Independence and Care, AAATE 2011. Assistive Technology Research Series, vol. 29, pp. 905–912. IOS Press (2011)
9. Peissner, M., Häbe, D., Janssen, D., Sellner, T.: MyUI: generating accessible user interfaces from multimodal design patterns. In: Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2012, pp. 81–90. ACM, New York (2012)
10. Zimmermann, G., Vanderheiden, G.: The Universal Control Hub: An Open Platform for Remote User Interfaces in the Digital Home. In: Jacko, J.A. (ed.) HCI 2007. LNCS, vol. 4551, pp. 1040–1049. Springer, Heidelberg (2007)
11. Miñón, R., Abascal, J.: Supportive adaptive user interfaces inside and outside the home. In: Ardissono, L., Kuflik, T. (eds.) UMAP Workshops 2011. LNCS, vol. 7138, pp. 320–334. Springer, Heidelberg (2012)
12. ISO/IEC. ISO/IEC 24752. Information Technology - User Interfaces - Universal Remote Console. Part 1: Framework. 1st edn. ISO/IEC (2008)
13. Stocklöw, C., Grguric, A., Dutz, T., Vandommele, T., Kuijper, A.: Resource Management for Multimodal and Multilingual Adaptation of User Interfaces in Ambient Assisted Living Environments. In: Stephanidis, C., Antona, M. (eds.) UAHCI/HCII 2013, Part III. LNCS, vol. 8011, pp. 97–106. Springer, Heidelberg (2013)
14. Boyer, J.M.: XForms 1.1. W3C Recommendation (October 20, 2009), http://www.w3.org/TR/xforms/
15. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computer 15(3), 289–308 (2003)
16. OSGi Service Platform Core Specification (2011), http://www.osgi.org/download/r4v43/osgi.core-4.3.0.pdf (accessed: February 2014)
17. W3C MBUI - Task Models, http://www.w3.org/TR/2012/WD-task-models-20120802/ (accessed: October 2013)
18. W3C SOAP Version 1.2 Part 1: Messaging Framework (2nd edn.), http://www.w3.org/TR/soap12-part1/ (accessed: February 2014)
19. Hanke, S., Mayer, C., Hoeftberger, O., Boos, H., Wichert, R., Tazari, M.-R., Wolf, P., Furfari, F.: universAAL – an open and consolidated AAL platform. In: Ambient Assisted Living, pp. 127–140. Springer (2011)
20. Miroslav, S., Matthias, G., Christopher, M., Martin, M., Martin, P.: A Framework for the Automatic Adaptation of User Interfaces. In: Assistive Technology: From Research to Practice: AAATE 2013, pp. 1298–1304 (2013)
21. Mori, G., Paternó, F., Santoro, C.: CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. IEEE Transactions on Software Engineering 28(8), 797–813
22. Burbeck, S.: Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller, MVC (1992), http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html (accessed: February 2014)

23. Potel, M., MVP: Model-View-Presenter; The Taligent Programming Model for C++ and Java (1996), http://www.wildcrest.com/Potel/Portfolio/mvp.pdf (accessed: January 2014)
24. Zhang, Y., Luo, Y.: An architecture and implement model for Model-View-Presenter pattern. In: 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT) (2010) doi: 10.1109/ICCSIT.2010.5565090
25. W3C Voice Extensible Markup Language (VoiceXML) 3.0, http://www.w3.org/TR/voicexml30/ (accessed: January 2014)
26. W3C Web Content Accessibility Guidelines (WCAG) 2.0, http://www.w3.org/TR/2008/REC-WCAG20-20081211/ (accessed: February 2014)